Deductive Verification of Software
Exercises and Solutions

(6.0 VU Formal Methods in Computer Science)

Gernot Salzer

SS 2018

Exercise 1

Let p be the following program:

```
x := x + y;
if x < 0 then
   abort
else
   while x ≠ y do
      x := x + 1;
      y := y + 2
   od
fi
```

Moreover, let σ be a state such that $\sigma(x) = 1$ and $\sigma(y) = 5$. Compute $[p]\,\sigma$ using

(a) the structural operational semantics of TPL;

(b) the natural semantics of TPL.

Solution

(a) We compute a complete program run. The final state is, by the definition of the semantics, the result of $[p]\,\sigma$.

$$
\begin{aligned}
(p, \sigma) &= (x := x + y; \mathsf{if}\ \cdots\ \mathsf{fi},\ \sigma) \\
&\quad \left[\begin{array}{l} (x := x + y,\ \sigma) \\ \Rightarrow \sigma_1 \qquad \text{where } \sigma_1(x) = [x + y]\,\sigma = 6 \text{ and } \sigma_1(y) = \sigma(y) = 5 \end{array} \right. \\
&\Rightarrow (\mathsf{if}\ x < 0\ \mathsf{then\ abort\ else\ while}\ x \neq y\ \mathsf{do}\ \cdots\ \mathsf{od\ fi},\ \sigma_1)
\end{aligned}
$$

$\Rightarrow$ (while $x \neq y$ do $x := x + 1$; $y := y + 2$ od, σ_1)

since $[x < 0]\,\sigma_1 = (\sigma_1(x) < 0) = (6 < 0) = 0$

$\Rightarrow$ ($x := x + 1; y := y + 2$; while $x \neq y$ do $x := x + 1$; $y := y + 2$ od, σ_1)

since $[x \neq y]\,\sigma_1 = (\sigma_1(x) \neq \sigma_1(y)) = (6 \neq 5) = 1$

$\lfloor\!\lceil$ ($x := x + 1; y := y + 2$, σ_1)
$\lfloor\!\lceil$ ($x := x + 1$, σ_1)
$\lfloor\!\lceil$ $\Rightarrow \sigma_2$ where $\sigma_2(x) = [x + 1]\,\sigma_1 = 7$ and $\sigma_2(y) = \sigma_1(y) = 5$
$\Rightarrow$ ($y := y + 2$, σ_2)

$\Rightarrow$ ($y := y + 2$; while $x \neq y$ do $x := x + 1$; $y := y + 2$ od, σ_2)

$\lfloor\!\lceil$ ($y := y + 2$, σ_2)
$\lfloor\!\lceil$ $\Rightarrow \sigma_3$ where $\sigma_3(y) = [y + 2]\,\sigma_2 = 7$ and $\sigma_3(x) = \sigma_2(x) = 7$

$\Rightarrow$ (while $x \neq y$ do $x := x + 1$; $y := y + 2$ od, σ_3)

$\Rightarrow \sigma_3$

since $[x \neq y]\,\sigma_3 = (\sigma_3(x) \neq \sigma_3(y)) = (7 \neq 7) = 0$

Therefore we have $[p]\,\sigma = \sigma_3$.

(b)
$$
\begin{aligned}
[p]\,\sigma &= [x := x + y; \text{if } \cdots \text{ fi}]\,\sigma \\
&= [\text{if } \cdots \text{ fi}]\,[x := x + y]\,\sigma \\
&= [\text{if } x < 0 \text{ then abort else while } x \neq y \text{ do } \cdots \text{ od fi}]\,\sigma_1 \\
&\quad \text{where } \sigma_1(x) = [x + y]\,\sigma = 6 \text{ and } \sigma_1(y) = \sigma(y) = 5 \\
&= [\text{while } x \neq y \text{ do } x := x + 1;\ y := y + 2 \text{ od}]\,\sigma_1 \\
&\quad \text{since } [x < 0]\,\sigma_1 = (\sigma_1(x) < 0) = (6 < 0) = 0 \\
&= [\text{while } x \neq y \text{ do } x := x + 1;\ y := y + 2 \text{ od}]\,[x := x + 1; y := y + 2]\,\sigma_1 \\
&\quad \text{since } [x \neq y]\,\sigma_1 = (\sigma_1(x) \neq \sigma_1(y)) = (6 \neq 5) = 1 \\
&= [\text{while } x \neq y \text{ do } x := x + 1;\ y := y + 2 \text{ od}]\,[y := y + 2]\,[x := x + 1]\,\sigma_1 \\
&= [\text{while } x \neq y \text{ do } x := x + 1;\ y := y + 2 \text{ od}]\,[y := y + 2]\,\sigma_2 \\
&\quad \text{where } \sigma_2(x) = [x + 1]\,\sigma_1 = 7 \text{ and } \sigma_2(y) = \sigma_1(y) = 5 \\
&= [\text{while } x \neq y \text{ do } x := x + 1;\ y := y + 2 \text{ od}]\,\sigma_3 \\
&\quad \text{where } \sigma_3(y) = [y + 2]\,\sigma_2 = 7 \text{ and } \sigma_3(x) = \sigma_2(x) = 7 \\
&= \sigma_3 \\
&\quad \text{since } [x \neq y]\,\sigma_3 = (\sigma_3(x) \neq \sigma_3(y)) = (7 \neq 7) = 0
\end{aligned}
$$

Exercise 2

Let p be the program of exercise 1. Show that $\{\, x = 2y \wedge y > 2 \,\}\, p\, \{\, x = y \,\}$ is totally correct using weakest preconditions.

Solution

We show that the precondition implies the weakest precondition of the program with respect to the postcondition.

$\mathrm{wp}(x := x + y; \mathsf{if}\ \cdots,\ x = y) = \mathrm{wp}(x := x + y,\ \mathrm{wp}(\mathsf{if}\ \cdots,\ x = y))$

$\mathrm{wp}(\mathsf{if}\ \cdots,\ x = y)$
$= ((x < 0 \wedge \mathrm{wp}(\mathsf{abort}, x = y)) \vee (x \geq 0 \wedge \mathrm{wp}(\mathsf{while}\ \cdots,\ x = y)))$
$= ((x < 0 \wedge \mathrm{false}) \vee (x \geq 0 \wedge \mathrm{wp}(\mathsf{while}\ \cdots,\ x = y)))$
$= (\mathrm{false} \vee (x \geq 0 \wedge \mathrm{wp}(\mathsf{while}\ \cdots,\ x = y)))$
$= (x \geq 0 \wedge \mathrm{wp}(\mathsf{while}\ \cdots,\ x = y))$

$\mathrm{wp}(\mathsf{while}\ \cdots,\ x = y) = \exists i \geq 0\, F_i$
$F_0 : \neg(x \neq y) \wedge x = y$
$F_1 : x \neq y \wedge \mathrm{wp}(x := x + 1; y := y + 2,\ x = y)$
$= (x \neq y \wedge x + 1 = y + 2)$
$= (x = y + 1)$
$F_i : x = y + i \quad$ (guess)
$F_{i+1} : x \neq y \wedge \mathrm{wp}(x := x + 1; y := y + 2,\ F_i)$
$= x \neq y \wedge (x + 1) = (y + 2) + i$
$= x \neq y \wedge x = y + i + 1$
$= (x = y + i + 1) \quad$ (proof)
$= \exists i \geq 0\, (x = y + i)$
$= x \geq y$

$= (x \geq 0 \wedge x \geq y)$

$= \mathrm{wp}(x := x + y,\ x \geq 0 \wedge x \geq y)$
$= (x + y) \geq 0 \wedge (x + y) \geq y$
$= (x + y) \geq 0 \wedge x \geq 0$

It remains to show that the precondition implies the weakest precondition.

$$\begin{aligned} x = 2y \wedge y > 2 &\Rightarrow \mathrm{wp}(p, x = y) \\ &\Rightarrow (x + y) \geq 0 \wedge x \geq 0 \end{aligned}$$

The two conjuncts of the conclusion can be proven separately:

$$\begin{aligned} x = 2y \wedge y > 2 &\Rightarrow (x + y) \geq 0 \\ x = 2y \wedge y > 2 &\Rightarrow x \geq 0 \end{aligned}$$

The first implication is valid, since $x = 2y$ and $y > 2$ imply $(x + y) = 3y > 6 \geq 0$. The second implication holds since $x = 2y$ and $y > 2$ imply $x > 4$, which implies the conclusion $x \geq 0$.

Exercise 3

Let p be the program of exercise 1. Show that $\{\, x = 2y \wedge y > 2 \,\}\, p\, \{\, x = y \,\}$ is totally correct using the Hoare calculus.

Solution

The Hoare derivation is shown in figure 1. It remains to find formulas F and Inv and an expression t such that the implications 1, 2, 4, 6 and 7 are valid.

We guess a suitable formula F by forward reasoning. The new value of x is the old one plus the value of y, hence the old value can be obtained after the assignment by evaluating $x - y$. We obtain $x - y = 2y \wedge y > 2$ as description of the states after the assignment, therefore we choose $F = (x = 3y \wedge y > 2)$. Now we are able to prove implications 1 and 2.[1]

Validity of implication 1:

$$(x = 2y \wedge y > 2) \Rightarrow F[x/x + y]$$
$$(x = 2y \wedge y > 2) \Rightarrow (x + y = 3y \wedge y > 2)$$

The first conjunct of the conclusion, $x + y = 3y$, is implied by the first conjunct of the premise, and the second conjunct of the conclusion, $y > 2$, is part of the premise.

Validity of implication 2:

$$(F \wedge x < 0) \Rightarrow \text{false}$$
$$(x = 3y \wedge y > 2 \wedge x < 0) \Rightarrow \text{false}$$

The premise is contradictory: $y > 2$ implies $3y > 6$, while $x = 3y$ and $x < 0$ imply $3y < 0$. Therefore the implication holds in every state.

To obtain a suitable variant t we rewrite the loop condition $x \neq y$ as $x - y \neq 0$. We observe that the expression $x - y$ decreases in each iteration and equals zero when the loop terminates. Therefore we guess $t = x - y$.

The main purpose of invariant Inv is to guarantee the partial correctness of the loop: It has to be strong enough to imply the postcondition (implication 6), while being at the same time weak enough to be implied by the precondition of the loop (implication 4) and being maintained throughout the iterations (first conclusion in implication 7). In this example implication 6 is valid for any choice of Inv, since the negated loop condition

[1] Formally, this particular choice for F is obtained as the strongest postcondition of $x := x + y$ with respect to the postcondition:

$$\begin{aligned} F &= \text{sp}(x := x + y,\ x = 2y \wedge y > 2) \\ &= \exists x'(x' = 2y \wedge y > 2 \wedge x = x' + y) \\ &= \exists x'(x' = 2y \wedge y > 2 \wedge x = 2y + y) \\ &= \exists x'(x' = 2y) \wedge y > 2 \wedge x = 3y \\ &= (\text{true} \wedge y > 2 \wedge x = 3y) \\ &= (y > 2 \wedge x = 3y) \end{aligned}$$

$$\dfrac{\dfrac{(Inv \wedge x \neq y \wedge t = z) \Rightarrow (Inv \wedge 0 \leq t < z)[y/y+2][x/x+1] \overset{(7)}{} \quad \{\,(Inv \wedge 0 \leq t < z)[y/y+2][x/x+1]\,\}\, x := x+1 \,\{\,(Inv \wedge 0 \leq t < z)[y/y+2]\,\}}{\{\,Inv \wedge x \neq y \wedge t = z\,\}\, x := x+1 \,\{\,(Inv \wedge 0 \leq t < z)[y/y+2]\,\}} \overset{\text{(5, see above)}}{} \quad \{\,(Inv \wedge 0 \leq t < z)[y/y+2]\,\}\, y := y+2 \,\{\,Inv \wedge 0 \leq t < z\,\}}{\dfrac{\{\,Inv \wedge x \neq y \wedge t = z\,\}\, x := x+1 \,\{\,Inv \wedge x \neq y \wedge t = z\,\}\, x := x+1; y := y+2 \,\{\,Inv \wedge 0 \leq t < z\,\}}{\{\,Inv\,\}\,\mathsf{while}\ x \neq y\ \mathsf{do}\ x := x+1; y := y+2\ \mathsf{od}\,\{\,Inv \wedge \neg x \neq y\,\}} \overset{(\mathrm{wht}')}{} \quad (Inv \wedge \neg x \neq y) \Rightarrow x = y}{\{\,F \wedge x \not< 0\,\}\,\mathsf{while}\ x \neq y\ \mathsf{do}\ x := x+1; y := y+2\ \mathsf{od}\,\{\,x = y\,\}} \overset{(6)}{} \overset{(\mathrm{sc})}{} \overset{(\mathrm{lc})}{}$$

$$\overset{\text{(5, see above)}}{\{\,Inv \wedge x \neq y \wedge t = z\,\}\, x := x+1 \,\{\,(Inv \wedge 0 \leq t < z)[y/y+2]\,\}} \quad \overset{(\mathrm{as})}{\{\,(Inv \wedge 0 \leq t < z)[y/y+2]\,\}\, y := y+2 \,\{\,Inv \wedge 0 \leq t < z\,\}}$$

$$\dfrac{\dfrac{\dfrac{(x = 2y \wedge y > 2) \Rightarrow F[x/x+y] \overset{(1)}{} \quad \{\,F[x/x+y]\,\}\, x := x+y \,\{\,F\,\}}{\{\,x = 2y \wedge y > 2\,\}\, x := x+y \,\{\,F\,\}} \overset{(\mathrm{as})}{} \quad \dfrac{(F \wedge x < 0) \Rightarrow \mathrm{false} \overset{(2)}{} \quad \{\,\mathrm{false}\,\}\,\mathsf{abort}\,\{\,x = y\,\}}{\{\,F \wedge x < 0\,\}\,\mathsf{abort}\,\{\,x = y\,\}} \overset{(\mathrm{abt})}{}}{\{\,F\,\}\,\mathsf{if}\ x < 0\ \mathsf{then\ abort\ else\ while}\ x \neq y\ \mathsf{do}\ x := x+1; y := y+2\ \mathsf{od\ fi}\,\{\,x = y\,\}} \overset{(\mathrm{lc})}{} \quad \{\,F \wedge x \not< 0\,\}\,\mathsf{while}\ \cdots\ \mathsf{od}\,\{\,x = y\,\}}{\{\,x = 2y \wedge y > 2\,\}\, x := x+y; \mathsf{if}\ x < 0\ \mathsf{then\ abort\ else\ while}\ x \neq y\ \mathsf{do}\ x := x+1; y := y+2\ \mathsf{od\ fi}\,\{\,x = y\,\}} \overset{(\mathrm{if})}{} \overset{(\mathrm{sc})}{} \overset{\text{(3, see above)}}{}$$

It remains to find formulas F and Inv and an expression t such that the following implications become valid:

(1) $(x = 2y \wedge y > 2) \Rightarrow F[x/x+y]$

(2) $(F \wedge x < 0) \Rightarrow \mathrm{false}$

(4) $(F \wedge x \not< 0) \Rightarrow Inv$

(6) $(Inv \wedge \neg x \neq y) \Rightarrow x = y$

(7) $(Inv \wedge x \neq y \wedge t = z) \Rightarrow (Inv \wedge 0 \leq t < z)[y/y+2][x/x+1]$

The auxilary variable z is not used anywhere outside of the loop and 'stores' the value of the variant t at the beginning of the loop, so that we can compare it to the value of t at the end of the loop. Note that we need different variants, invariants and auxiliary variables for each loop.

Figure 1: Derivation of $\{\,x = 2y \wedge y > 2\,\}\, p\, \{\,x = y\,\}$ in the Hoare calculus, where p is the program of exercise 3.

already implies the postcondition. Hence we choose the weakest possible invariant, $Inv = \text{true}$, which obviously also satisfies the other implications.

The second purpose of the invariant is to ensure properties of the variables needed for showing that t is a variant. To prove $t \geq 0$ we need to assume $x \geq y$; obviously the loop only terminates for such states. This property has to be guaranteed at the start of the loop and after each iteration. Therefore we add it to the invariant and obtain $Inv = (\text{true} \wedge x \geq y) = x \geq y$.

Validity of implication 4:

$$(F \wedge x \not< 0) \Rightarrow Inv$$
$$(x = 3y \wedge y > 2 \wedge x \geq 0) \Rightarrow x \geq y$$

$x \geq y$ follows from $x = 3y$ and the fact that y is non-negative (because of $y > 2$).

Validity of implication 6:

$$(Inv \wedge \neg x \neq y) \Rightarrow x = y$$
$$(Inv \wedge x = y) \Rightarrow x = y$$

The conclusion is part of the premise.

Validity of implication 7:

$$(Inv \wedge x \neq y \wedge t = z) \Rightarrow (Inv \wedge 0 \leq t < z)[y/y+2][x/x+1]$$
$$(x \geq y \wedge x \neq y \wedge x - y = z) \Rightarrow (x \geq y \wedge 0 \leq x - y < z)[y/y+2][x/x+1]$$
$$(x \geq y \wedge x \neq y \wedge x - y = z) \Rightarrow (x \geq (y{+}2) \wedge 0 \leq x - (y{+}2) < z)[x/x+1]$$
$$(x \geq y \wedge x \neq y \wedge x - y = z) \Rightarrow (x{+}1 \geq y{+}2 \wedge 0 \leq (x{+}1) - (y{+}2) < z)$$
$$(x \geq y \wedge x \neq y \wedge x - y = z) \Rightarrow (x \geq y + 1 \wedge 0 \leq x - y - 1 < z)$$
$$(x > y \wedge x - y = z) \Rightarrow 0 \leq x - y - 1 < z$$

(Note that $x \geq y + 1$ is the same as $0 \leq x - y - 1$.) The condition $x > y$ in the premise is equivalent to $0 \leq x - y - 1$, and the condition $x - y = z$ implies $x - y - 1 < z$.

Since all implications obtained by the Hoare calculus are valid, this initial correctness assertion is totally correct.

Exercise 4

Let p be the program of exercise 1. Show that $\{\, x = 2y \wedge y > 2 \,\}\, p \,\{\, x = y \,\}$ is totally correct using annotation rules.

Solution

We annotate the program with additional assertions. The order of rule applications is indicated by the numbering of formulas. The order as well as the kind of rules is not uniquely determined; other annotations are possible.

$\{\, F_1 \colon x = 2y \wedge y > 2 \,\}$
$x := x + y;$
$\{\, F_3 \colon \exists x'(x' = 2y \wedge y > 2 \wedge x = x' + y) \,\}$ as↓
if $x < 0$ then
  $\{\, F_4 \colon F_3 \wedge x < 0 \,\}$ if↓
  $\{\, F_6 \colon \text{false} \,\}$ abt
  abort
  $\{\, F_7 \colon \text{false} \,\}$ abt
  $\{\, F_8 \colon x = y \,\}$ fi↑
else
  $\{\, F_5 \colon F_3 \wedge x \not< 0 \,\}$ if↓
  $\{\, F_{10} \colon \mathit{Inv} \,\}$ wht″
  while $x \neq y$ do
    $\{\, F_{11} \colon \mathit{Inv} \wedge x \neq y \wedge t = z \,\}$ wht″
    $\{\, F_{15} \colon (\mathit{Inv} \wedge 0 \leq t < z)[y/y + 2][x/x + 1] \,\}$ as↑
    $x := x + 1;$
    $\{\, F_{14} \colon (\mathit{Inv} \wedge 0 \leq t < z)[y/y + 2] \,\}$ as↑
    $y := y + 2$
    $\{\, F_{12} \colon \mathit{Inv} \wedge 0 \leq t < z \,\}$ wht″
  od
  $\{\, F_{13} \colon \mathit{Inv} \wedge \neg(x \neq y) \,\}$ wht″
  $\{\, F_9 \colon x = y \,\}$ fi↑
fi
$\{\, F_2 \colon x = y \,\}$

We start by simplifying formula F_3:

$$\begin{aligned}\exists x'(x' = 2y \wedge y > 2 \wedge x = x' + y) &= \exists x'(x' = 2y \wedge y > 2 \wedge x = 2y + y) \\ &= y > 2 \wedge x = 3y \wedge \exists x'(x' = 2y) \\ &= y > 2 \wedge x = 3y\end{aligned}$$

For Inv and t we choose the same invariant and variant as above. It remains to prove the validity of five implications: $F_4 \Rightarrow F_6$, $F_7 \Rightarrow F_8$, $F_5 \Rightarrow F_{10}$, $F_{11} \Rightarrow F_{15}$, and $F_{13} \Rightarrow F_9$. The implication $F_7 \Rightarrow F_8$ is trivially true (false implies everything). The other four implications are the same as derived by the Hoare calculus above.

Exercise 5

Prove that $[p; q]\,\sigma = [q]\,[p]\,\sigma$ holds for all programs p and q and all states σ.

Hint: You have to show "$[p; q]\,\sigma = \phi$ holds if and only if $[p]\,\sigma = \tau$ and $[q]\,\tau = \phi$ hold for some state τ". The next step consists in replacing the semantics functions $[p; q]$, $[q]$ and $[p]$ by their definition via structural operational semantics.

Solution

$[p;q]\,\sigma = [q]\,[p]\,\sigma$ means that

$[p;q]\,\sigma = \phi$ holds if and only if $[p]\,\sigma = \tau$ and $[q]\,\tau = \phi$ hold for some state τ.

By the semantics of TPL this is the same as

$(p;q,\sigma) \Rightarrow^* \phi$ holds if and only if $(p,\sigma) \Rightarrow^* \tau$ and $(q,\tau) \Rightarrow^* \phi$ hold for some state τ.

Only-if direction: We show by induction that the following assertion holds for all $n \geq 2$.

$A(n)$: If $(p;q,\sigma) \Rightarrow^n \phi$, then $(p,\sigma) \Rightarrow^* \tau$ and $(q,\tau) \Rightarrow^* \phi$ for some state τ.

Base case $n = 2$: Every program needs at least one transition for a complete run, so the only run with two steps is $(p;q,\sigma) \Rightarrow (q,\sigma') \Rightarrow \phi$. By the semantics of sequential composition the first step implies $(p,\sigma) \Rightarrow \sigma'$. Choosing $\tau = \sigma'$ we obtain $(p,\sigma) \Rightarrow \tau$ and $(q,\tau) \Rightarrow \phi$, hence we have $(p,\sigma) \Rightarrow^* \tau$ and $(q,\tau) \Rightarrow^* \phi$.

Inductive step: Using $A(n)$ as induction hypothesis we show that $A(n+1)$ also holds. Suppose we have $(p;q,\sigma) \Rightarrow^{n+1} \phi$. According to the semantics of sequential composition there are two possibilities for the first step.

If $(p,\sigma) \Rightarrow \sigma'$, then we have $(p;q,\sigma) \Rightarrow (q,\sigma') \Rightarrow^n \phi$. Choosing $\tau = \sigma'$ we obtain $(p,\sigma) \Rightarrow \tau$ and $(q,\tau) \Rightarrow^n \phi$, hence we have $(p,\sigma) \Rightarrow^* \tau$ and $(q,\tau) \Rightarrow^* \phi$.

If $(p,\sigma) \Rightarrow (p',\sigma')$, then we have $(p;q,\sigma) \Rightarrow (p';q,\sigma') \Rightarrow^n \phi$. By induction hypothesis there is a state τ such that $(p',\sigma') \Rightarrow^* \tau$ and $(q,\tau) \Rightarrow^* \phi$. Combining $(p,\sigma) \Rightarrow (p',\sigma')$ and $(p',\sigma') \Rightarrow^* \tau$ we obtain $(p,\sigma) \Rightarrow^* \tau$.

If direction: We show by induction that the following assertion holds for all $n \geq 1$.

$B(n)$: If $(p,\sigma) \Rightarrow^n \tau$ and $(q,\tau) \Rightarrow^* \phi$ for some state τ, then $(p;q,\sigma) \Rightarrow^* \phi$.

Base case $n = 1$: By the semantics of sequential composition, $(p,\sigma) \Rightarrow \tau$ implies $(p;q,\sigma) \Rightarrow (q,\tau)$. Combining it with $(q,\tau) \Rightarrow^* \phi$ we obtain $(p;q,\sigma) \Rightarrow^* \phi$.

Inductive step: Using $B(n)$ as induction hypothesis we show that $B(n+1)$ also holds. Suppose we have $(p,\sigma) \Rightarrow^{n+1} \tau$ and $(q,\tau) \Rightarrow^* \phi$ for some state τ. The first part can be written as $(p,\sigma) \Rightarrow (p',\sigma') \Rightarrow^n \tau$ for some state σ'. By the semantics of sequential composition, $(p,\sigma) \Rightarrow (p',\sigma')$ implies $(p;q,\sigma) \Rightarrow (p';q,\sigma')$. By induction hypothesis, $(p',\sigma') \Rightarrow^n \tau$ and $(q,\tau) \Rightarrow^* \phi$ together imply $(p';q,\sigma') \Rightarrow^* \phi$. Combining the former with the latter we obtain $(p;q,\sigma) \Rightarrow^* \phi$.

Exercise 6

Consider the following modified while-rule:

$$\frac{\{\, Inv \,\}\, p \,\{\, Inv \,\}}{\{\, Inv \,\}\, \mathsf{while}\ e\ \mathsf{do}\ p\ \mathsf{od}\, \{\, Inv \wedge \neg e \,\}}\ \text{mw}$$

(a) Show that this rule is admissible (sound) regarding partial correctness.

Hint: Derive the new rule using rules that we alreay know to be admissible.

(b) Show that the Hoare calculus for partial correctness is no longer complete, if we replace the regular while-rule by the modified one.

Hint: Find a correctness assertion that is correct (argue why it is!) but that cannot be derived in the modified calculus (explain why it can't!).

Solution

(a) We have to show that the conclusion $\{\, Inv \,\}\, \mathsf{while}\ e\ \mathsf{do}\ p\ \mathsf{od}\, \{\, Inv \wedge \neg e \,\}$ is true whenever the premise $\{\, Inv \,\}\, p\, \{\, Inv \,\}$ is true, for all formulas Inv, e and programs p. This can be done by deriving the conclusion from the premise using rules that are already known to be admissible, like the regular rules of Hoare calculus.

$$\dfrac{\dfrac{(Inv \wedge e) \Rightarrow Inv \quad \{\, Inv \,\}\, p\, \{\, Inv \,\}}{\{\, Inv \wedge e \,\}\, p\, \{\, Inv \,\}}\ \text{lc}}{\{\, Inv \,\}\, \mathsf{while}\ e\ \mathsf{do}\ p\ \mathsf{od}\, \{\, Inv \wedge \neg e \,\}}\ \text{wh}$$

The formula $(Inv \wedge e) \Rightarrow Inv$ is a tautology. Therefore, by the correctness of the Hoare calculus, the conclusion $\{\, Inv \,\}\, \mathsf{while}\ e\ \mathsf{do}\ p\ \mathsf{od}\, \{\, Inv \wedge \neg e \,\}$ is true whenever the premise $\{\, Inv \,\}\, p\, \{\, Inv \,\}$ is true.

(b) To show the incompleteness of the modified Hoare calculus we give a counter-example, which consists of a concrete correctness assertion that is true with respect to partial correctness, but which cannot be derived in the modified calculus. Consider the assertion

$$\{\, x \geq 0 \,\}\, \mathsf{while}\ x > 0\ \mathsf{do}\ x := x - 1\ \mathsf{od}\, \{\, x \geq 0 \wedge x \not> 0 \,\}\ .$$

This assertion is true as can e.g. be shown by deriving it in the regular calculus:

$$\dfrac{\dfrac{\overset{\text{valid}}{x \geq 0 \wedge x > 0 \Rightarrow x - 1 \geq 0} \quad \{\, x - 1 \geq 0 \,\}\, x := x - 1\, \{\, x \geq 0 \,\}^{\text{as}}}{\{\, x \geq 0 \wedge x > 0 \,\}\, x := x - 1\, \{\, x \geq 0 \,\}}\ \text{lc}}{\{\, x \geq 0 \,\}\, \mathsf{while}\ x > 0\ \mathsf{do}\ x := x - 1\ \mathsf{od}\, \{\, x \geq 0 \wedge x \not> 0 \,\}}\ \text{wh}$$

But this assertion cannot be derived in the modified calculus as we will show by contradiction. Suppose it can be derived. Then the derivation must have the form

$$\dfrac{x \geq 0 \Rightarrow F \quad \dfrac{\dfrac{\text{some derivation of } \{\, F \,\}\, x := x - 1\, \{\, F \,\}}{\{\, F \,\}\, x := x - 1\, \{\, F \,\}}}{\{\, F \,\}\, \mathsf{while}\ x > 0\ \mathsf{do}\ x := x - 1\ \mathsf{od}\, \{\, F \wedge x \not> 0 \,\}}\ \text{mw} \quad (F \wedge x \not> 0) \Rightarrow (x \geq 0 \wedge x \not> 0)}{\{\, x \geq 0 \,\}\, \mathsf{while}\ x > 0\ \mathsf{do}\ x := x - 1\ \mathsf{od}\, \{\, x \geq 0 \wedge x \not> 0 \,\}}\ \text{lc}$$

for some suitable invariant F. Note that lc and mw are the only rules that can have our counter-example as conclusion. Moreover, mw has to be applied once since it is

the only rule that can introduce a while statement. (In fact, the logical consequence rule can be applied several times, but the effect of several applications can always be achieved with just one application.)

Now, if such a derivation indeed exists, then the formulas $x \geq 0 \Rightarrow F$ and $(F \wedge x \not> 0) \Rightarrow (x \geq 0 \wedge x \not> 0)$ are valid and the correctness assertion $\{\, F \,\}\, x := x - 1 \,\{\, F \,\}$ is true (since it can be derived). We show that this cannot happen simultaneously, hence the derivation does not exist. This means that the calculus is incomplete, since we have found a true correctness assertion that cannot be derived.

Consider a state σ with $\sigma(x) = 0$. Since $x \geq 0 \Rightarrow F$ is supposed to be valid and $x \geq 0$ is true for σ, F must also be true for σ. Since $\{\, F \,\}\, x := x - 1 \,\{\, F \,\}$ is true, we conclude that F is also true for σ', where $\sigma'(x) = -1$ (state after executing the assignment). But the implication $(F \wedge x \not> 0) \Rightarrow (x \geq 0 \wedge x \not> 0)$ does not hold for σ': The premise is true since $\sigma'(x) \not> 0$, but the conclusion $x \geq 0 \wedge x \not> 0$ is not true, since σ' does not satisfy $x \geq 0$.

Exercise 7

Determine the strongest postcondition of while-loops, i.e., find a formula equivalent to $\mathrm{sp}(F, \mathsf{while}\ e\ \mathsf{do}\ p\ \mathsf{od})$ similar to the weakest precondition in the course.

Solution

$\mathrm{sp}(F, \mathsf{while}\ e\ \mathsf{do}\ p\ \mathsf{od}) = \neg e \wedge \exists i \geq 0\, G_i$, where G_i is defined recursively by

$$G_0 = F$$
$$G_{i+1} = \mathrm{sp}(e \wedge G_i, p)$$

This formula can be derived e.g. by loop unrolling, making use of the equivalence

$$[\mathsf{while}\ e\ \mathsf{do}\ p\ \mathsf{od}] = [\mathsf{if}\ e\ \mathsf{then}\ p; \mathsf{while}\ e\ \mathsf{do}\ p\ \mathsf{od}\ \mathsf{else}\ \mathsf{skip}\ \mathsf{fi}]$$

(see example 1 in the document *"Some examples with solutions"* in Tuwel). We obtain:

$$\begin{aligned}
&\mathrm{sp}(F, \mathsf{while}\ e\ \mathsf{do}\ p\ \mathsf{od}) \\
&= \mathrm{sp}(F, \mathsf{if}\ e\ \mathsf{then}\ p; \mathsf{while}\ e\ \mathsf{do}\ p\ \mathsf{od}\ \mathsf{else}\ \mathsf{skip}\ \mathsf{fi}) \\
&= (\neg e \wedge F) \vee \mathrm{sp}(\mathrm{sp}(e \wedge F, p), \mathsf{while}\ e\ \mathsf{do}\ p\ \mathsf{od}) \\
&= (\neg e \wedge F) \vee \mathrm{sp}(\mathrm{sp}(e \wedge F, p), \mathsf{if}\ e\ \mathsf{then}\ p; \mathsf{while}\ e\ \mathsf{do}\ p\ \mathsf{od}\ \mathsf{else}\ \mathsf{skip}\ \mathsf{fi}) \\
&= (\neg e \wedge F) \vee (\neg e \wedge \mathrm{sp}(e \wedge F, p)) \vee \mathrm{sp}(\mathrm{sp}(e \wedge \mathrm{sp}(e \wedge F, p), p), \mathsf{while}\ e\ \mathsf{do}\ p\ \mathsf{od}) \\
&= \cdots
\end{aligned}$$

The general scheme of this calculation is

$$\mathrm{sp}(G_i, \mathsf{while}\ e\ \mathsf{do}\ p\ \mathsf{od}) = (\neg e \wedge G_i) \vee \mathrm{sp}(G_{i+1}, \mathsf{while}\ e\ \mathsf{do}\ p\ \mathsf{od})$$
$$\text{where } G_{i+1} = \mathrm{sp}(e \wedge G_i, p)$$

Starting from $G_0 = F$ we obtain

$$\begin{aligned}\mathrm{sp}(F, \mathsf{while}\ e\ \mathsf{do}\ p\ \mathsf{od}) &= (\neg e \wedge G_0) \vee (\neg e \wedge G_1) \vee (\neg e \wedge G_2) \vee \cdots \\ &= \neg e \wedge (G_0 \vee G_1 \vee G_2 \vee \cdots) \\ &= \neg e \wedge \exists i \geq 0\, G_i\end{aligned}$$

Exercise 8

Show that wp and wlp are dual to each other, i.e., show that $\mathrm{wlp}(p, G) = \neg\mathrm{wp}(p, \neg G)$ holds. Use this relationship to find a formula for $\mathrm{wlp}(\mathsf{while}\ e\ \mathsf{do}\ p\ \mathsf{od}, G)$ similar to the weakest precondition in the course.

Use your formula to compute the weakest liberal precondition of the program

$$z := 0;\ \mathsf{while}\ y \neq 0\ \mathsf{do}\ z := z + x;\ y := y - 1\ \mathsf{od}$$

with respect to the postcondition $z = x * y_0$. Compare the result to the weakest precondition computed in the course and explain the differences.

Solution

Given a program p and a postcondition G, there are three disjoint types of states: those states, for which p does not terminate; those states, for which p terminates in a G-state; and those states, for which p terminates in a $\neg G$-state.

$$\begin{aligned}\mathcal{S} = \quad & \{\, \sigma \in \mathcal{S} \mid [p]\,\sigma \text{ undefined} \,\} \\ \dot\cup\, & \{\, \sigma \in \mathcal{S} \mid [p]\,\sigma \text{ defined and } [G]\,[p]\,\sigma = \mathrm{true} \,\} \\ \dot\cup\, & \{\, \sigma \in \mathcal{S} \mid [p]\,\sigma \text{ defined and } [\neg G]\,[p]\,\sigma = \mathrm{true} \,\}\end{aligned}$$

where $\dot\cup$ denotes disjoint union. The first two sets can be interpreted as the weakest liberal precondition of p with respect to G, whereas the third one is the weakest preconditon of p with respect to $\neg G$.

$$\mathcal{S} = \mathrm{wlp}(p, \{\, G \,\}) \,\dot\cup\, \mathrm{wp}(p, \{\, \neg G \,\})$$

Subtracting the set $\mathrm{wp}(p, \{\, \neg G \,\})$ on both sides results in:

$$\mathcal{S} - \mathrm{wp}(p, \{\, \neg G \,\}) = \mathrm{wlp}(p, \{\, G \,\})$$

If we represent these state sets as formulas, the complement of a set with respect to the set of all states corresponds to negation. Hence we obtain:

$$\neg\mathrm{wp}(p, \neg G) = \mathrm{wlp}(p, G)$$

i.e., wp and wlp are indeed dual operators.

Applying this relationship to while-loops gives us a formula for computing wlp for while-loops.

$$\begin{aligned}\mathrm{wlp}(\mathsf{while}\ e\ \mathsf{do}\ p\ \mathsf{od}, G) &= \neg\mathrm{wp}(\mathsf{while}\ e\ \mathsf{do}\ p\ \mathsf{od}, \neg G)\\ &= \neg\exists i\,(i \geq 0 \wedge F_i)\\ &= \forall i\,(i \geq 0 \Rightarrow \neg F_i)\\ \text{where}\quad F_0 &= \neg e \wedge \neg G\\ F_{i+1} &= e \wedge \mathrm{wp}(p, F_i) = e \wedge \neg\mathrm{wlp}(p, \neg F_i)\end{aligned}$$

Since F_i occurs in negated form only, we rewrite the recursion such that it defines $\neg F_i$.

$$\begin{aligned}\neg F_0 &= e \vee G\\ \neg F_{i+1} &= \neg e \vee \mathrm{wlp}(p, \neg F_i)\end{aligned}$$

After renaming $\neg F_i$ to E_i we obtain the following compact definition of the weakest liberal precondition:

$$\begin{aligned}\mathrm{wlp}(\mathsf{while}\ e\ \mathsf{do}\ p\ \mathsf{od}, G) = \forall i \geq 0\, E_i\ &= \forall i\,(i \geq 0 \Rightarrow E_i) = \forall i\,(i < 0 \vee E_i)\\ \text{where}\quad E_0 &= e \vee G\\ E_{i+1} &= \neg e \vee \mathrm{wlp}(p, E_i)\end{aligned}$$

As an example, the weakest liberal precondition of the multiplication program can be computed as follows.

$\mathrm{wlp}(z := 0; \mathsf{while}\ y \neq 0\ \mathsf{do}\ z := z + x; y := y - 1\ \mathsf{od},\ z = xy_0)$
$= \mathrm{wlp}(z := 0,\ \mathrm{wlp}(\mathsf{while}\ y \neq 0\ \mathsf{do}\ z := z + x; y := y - 1\ \mathsf{od},\ z = xy_0))$
$= \mathrm{wlp}(z := 0,\ \forall i\,(i < 0 \vee E_i))$

$\forall i\,(i < 0 \vee E_i)$: We compute E_i for some values of i, guess $E_i = (y \neq i \vee z = x(y_0 - i))$ and prove the guess by induction.

Base case: $E_0 = (y \neq 0 \vee z = xy_0) = e \vee G$ ✓

Induction hypothesis: $E_i = (y \neq i \vee z = x(y_0 - i))$ holds.

Induction step ($i \geq 0$):

$$\begin{aligned}E_{i+1} &= \neg e \vee \mathrm{wlp}(p, E_i)\\ &= (y = 0 \vee \mathrm{wlp}(z := z + x; y := y - 1,\ (y \neq i \vee z = x(y_0 - i))))\\ &= (y = 0 \vee (y - 1) \neq i \vee (z + x) = x(y_0 - i))\\ &= (y = 0 \vee y \neq (i + 1) \vee z = x(y_0 - (i + 1)))\\ &= (y \neq (i + 1) \vee z = x(y_0 - (i + 1)))\quad \checkmark\end{aligned}$$

$$\begin{aligned}\forall i \geq 0\,(i < 0 \vee E_i) &= \forall i\,(i < 0 \vee y \neq i \vee z = x(y_0 - i))\\ &= \forall i\,(y < 0 \vee y \neq i \vee z = x(y_0 - y))\\ &= y < 0 \vee \forall i\,(y \neq i) \vee z = x(y_0 - y)\\ &= y < 0 \vee \mathrm{false} \vee z = x(y_0 - y)\\ &= y < 0 \vee z = x(y_0 - y)\end{aligned}$$

$\mathrm{wlp}(z := 0, \forall i\,(i < 0 \vee E_i))$
$= \mathrm{wlp}(z := 0, (y < 0 \vee z = x(y_0 - y)))$
$= (y < 0 \vee 0 = x(y_0 - y))$
$= (y < 0 \vee x = 0 \vee y = y_0)$

Comparison of weakest and weakest liberal precondition: Let p be the multiplication program. In the lecture we have computed the weakest precondition of p with respect to the postcondition $z = xy_0$ as

$$\mathrm{wp}(p,\, z = xy_0) = y \geq 0 \wedge (x = 0 \vee y = y_0)$$

We use this formula to rewrite the weakest liberal precondition:

$$\begin{aligned}\mathrm{wlp}(p,\, z = xy_0) &= y < 0 \vee x = 0 \vee y = y_0 \\ &= y < 0 \oplus (y \geq 0 \wedge (x = 0 \vee y = y_0)) \\ &= y < 0 \oplus \mathrm{wp}(p,\, z = xy_0)\end{aligned}$$

($\oplus$ denotes exclusive disjunction.) As we see, the two preconditions differ by the formula $y < 0$, which characterises exactly the states for which the result of p is undefined.

Exercise 9

Extend TPL by statements of the form "assert e". When the condition e evaluates to true, the program continues, otherwise the program aborts.

Specify the syntax and semantics of the extended language. Determine the weakest precondition, the weakest liberal precondition, the strongest postcondition, and Hoare rules (partial and total correctness) for assert-statements. Show that they are correct.

Treat the assert-statement as a first-class citizen, i.e., do not refer to other program statements in the final result. However, you may use other statements as intermediate steps when deriving the rules.

Solution

Syntax: $\mathcal{P} ::= \mathsf{skip} \mid \mathsf{abort} \mid \mathcal{V} := \mathcal{E} \mid \mathcal{P}; \mathcal{P} \mid \mathsf{if}\ \mathcal{E}\ \mathsf{then}\ \mathcal{P}\ \mathsf{else}\ \mathcal{P}\ \mathsf{fi} \mid \mathsf{while}\ \mathcal{E}\ \mathsf{do}\ \mathcal{P}\ \mathsf{od} \mid \mathsf{assert}\ \mathcal{E}$

Structural operational semantics: $(\mathsf{assert}\ e, \sigma) \Rightarrow \sigma \quad \text{if } [e]\sigma \neq 0$

Natural semantics (alternative to SOS): $[\mathsf{assert}\ e]\,\sigma = \sigma \quad \text{if } [e]\sigma \neq 0$

Regarding verification we observe that assert e is equivalent to if e then skip else abort fi.

Weakest precondition:

$$\begin{aligned}\mathrm{wp}(\mathsf{assert}\ e,\, G) &= \mathrm{wp}(\mathsf{if}\ e\ \mathsf{then\ skip\ else\ abort\ fi}, G) \\ &= (e \wedge \mathrm{wp}(\mathsf{skip}, G)) \vee (\neg e \wedge \mathrm{wp}(\mathsf{abort}, G)) \\ &= (e \wedge G) \vee (\neg e \wedge \mathrm{false}) \\ &= (e \wedge G)\end{aligned}$$

Weakest liberal precondition:

$$\begin{aligned}\mathrm{wlp}(\mathsf{assert}\ e,\, G) &= \mathrm{wlp}(\mathsf{if}\ e\ \mathsf{then\ skip\ else\ abort\ fi}, G)\\ &= (e \wedge \mathrm{wlp}(\mathsf{skip}, G)) \vee (\neg e \wedge \mathrm{wlp}(\mathsf{abort}, G))\\ &= (e \wedge G) \vee (\neg e \wedge \mathrm{true})\\ &= (e \wedge G) \vee \neg e\\ &= (G \vee \neg e)\\ &= (e \Rightarrow G)\end{aligned}$$

(It is a matter of taste, which of the last two formulas is the more natural one and therefore should be considered to be the result.)

Strongest postcondition:

$$\begin{aligned}\mathrm{sp}(F,\, \mathsf{assert}\ e) &= \mathrm{sp}(F,\, \mathsf{if}\ e\ \mathsf{then\ skip\ else\ abort\ fi})\\ &= \mathrm{sp}(F \wedge e,\, \mathsf{skip}) \vee \mathrm{sp}(F \wedge \neg e,\, \mathsf{abort})\\ &= (F \wedge e) \vee \mathrm{false}\\ &= (F \wedge e)\end{aligned}$$

Hoare calculus for partial correctness: We have three possibilities to derive a rule.

Method 1: Use the equivalence of assert and if e then skip else abort fi and apply the rules of Hoare calculus to determine the open premises (i.e. those that remain to be proven).

$$\dfrac{\dfrac{(F \wedge e) \Rightarrow G \quad \{\, G \,\}\, \mathsf{skip}\, \{\, G \,\}}{\{\, F \wedge e \,\}\, \mathsf{skip}\, \{\, G \,\}} \qquad \{\, F \wedge \neg e \,\}\, \mathsf{abort}\, \{\, G \,\}}{\{\, F \,\}\, \mathsf{if}\ e\ \mathsf{then\ skip\ else\ abort\ fi}\, \{\, G \,\}}}{\{\, F \,\}\, \mathsf{assert}\ e\, \{\, G \,\}}$$

The assertions $\{\, G \,\}\, \mathsf{skip}\, \{\, G \,\}$ and $\{\, F \wedge \neg e \,\}\, \mathsf{abort}\, \{\, G \,\}$ are axioms. Therefore we obtain the rule

$$\dfrac{(F \wedge e) \Rightarrow G}{\{\, F \,\}\, \mathsf{assert}\ e\, \{\, G \,\}}$$

Method 2: For every program p and formula G, the assertion $\{\, \mathrm{wlp}(p, G) \,\}\, p\, \{\, G \,\}$ is partially correct. Using the wlp of assert from above we obtain the axiom

$$\{\, e \Rightarrow G \,\}\, \mathsf{assert}\ e\, \{\, G \,\}$$

If we prefer a rule that is able to handle arbitrary preconditions of assert, we use the fact that $\{\, F \,\}\, p\, \{\, G \,\}$ is partially correct if and only if $F \Rightarrow \mathrm{wlp}(p, G)$.

$$\dfrac{F \Rightarrow (e \Rightarrow G)}{\{\, F \,\}\, \mathsf{assert}\ e\, \{\, G \,\}}$$

The premise is equivalent to $(F \wedge e) \Rightarrow G$, i.e., the rule is equivalent to the rule obtained by method 1.

Method 3: For every program p and formula F, the assertion $\{\, F \,\}\, p \,\{\, \mathrm{sp}(F,p) \,\}$ is partially correct. Using the sp of assert from above we obtain the axiom

$$\{\, F \,\}\, \mathsf{assert}\ e \,\{\, F \wedge e \,\}$$

If we prefer a rule that is able to handle arbitrary postconditions of assert, we use the fact that $\{\, F \,\}\, p \,\{\, G \,\}$ is partially correct if and only if $\mathrm{sp}(F,p) \Rightarrow G$.

$$\frac{(F \wedge e) \Rightarrow G}{\{\, F \,\}\, \mathsf{assert}\ e \,\{\, G \,\}}$$

This rule is again the same as obtained by the other methods.

Hoare calculus for total correctness: We have two possibilities to derive a rule.

Method 1: Use the equivalence of assert and if e then skip else abort fi and apply the rules of Hoare calculus to determine the open premises (i.e. those that remain to be proven).

$$\frac{\dfrac{(F \wedge e) \Rightarrow G \quad \{\, G \,\}\, \mathsf{skip} \,\{\, G \,\}}{\{\, F \wedge e \,\}\, \mathsf{skip} \,\{\, G \,\}} \quad \dfrac{(F \wedge \neg e) \Rightarrow \mathrm{false} \quad \{\, \mathrm{false} \,\}\, \mathsf{abort} \,\{\, G \,\}}{\{\, F \wedge \neg e \,\}\, \mathsf{abort} \,\{\, G \,\}}}{\dfrac{\{\, F \,\}\, \mathsf{if}\ e\ \mathsf{then\ skip\ else\ abort\ fi} \,\{\, G \,\}}{\{\, F \,\}\, \mathsf{assert}\ e \,\{\, G \,\}}}$$

The assertions $\{\, G \,\}\, \mathsf{skip} \,\{\, G \,\}$ and $\{\, \mathrm{false} \,\}\, \mathsf{abort} \,\{\, G \,\}$ are axioms. Therefore we obtain the rule

$$\frac{(F \wedge e) \Rightarrow G \quad (F \wedge \neg e) \Rightarrow \mathrm{false}}{\{\, F \,\}\, \mathsf{assert}\ e \,\{\, G \,\}}$$

Simplifying the propositional formula $((F \wedge e) \Rightarrow G) \wedge ((F \wedge \neg e) \Rightarrow \mathrm{false})$ yields the more elegant rule

$$\frac{F \Rightarrow (e \wedge G)}{\{\, F \,\}\, \mathsf{assert}\ e \,\{\, G \,\}}$$

Method 2: For every program p and formula G, the assertion $\{\, \mathrm{wp}(p,G) \,\}\, p \,\{\, G \,\}$ is totally correct. Using the wp of assert from above we obtain the axiom

$$\{\, e \wedge G \,\}\, \mathsf{assert}\ e \,\{\, G \,\}$$

If we prefer a rule that is able to handle arbitrary preconditions of assert, we use the fact that $\{\, F \,\}\, p \,\{\, G \,\}$ is totally correct if and only if $F \Rightarrow \mathrm{wp}(p,G)$.

$$\frac{F \Rightarrow (e \wedge G)}{\{\, F \,\}\, \mathsf{assert}\ e \,\{\, G \,\}}$$

This rule is the same as obtained by method 1.

Annotation rules for partial correctness: Based on wlp and sp from above one can define the following rules for annotating programs containing assert statements:

$$\begin{array}{rcll} \mathsf{assert}\ e\,\{G\} & \mapsto & \{G \vee \neg e\}\,\mathsf{assert}\ e\,\{G\} & (\text{assert}\uparrow) \\ \{F\}\,\mathsf{assert}\ e & \mapsto & \{F\}\,\mathsf{assert}\ e\,\{F \wedge e\} & (\text{assert}\downarrow) \end{array}$$

Annotation rules for total correctness: Based on wp from above one can define the following rule for annotating programs containing assert statements:

$$\mathsf{assert}\ e\,\{G\} \mapsto \{G \wedge e\}\,\mathsf{assert}\ e\,\{G\} \quad (\text{assert}\uparrow)$$

Exercise 10

Prove the total correctness of the following assertion. Describe the function computed by the program.

$\{\, x \geq 0 \,\}$
$y := 0;$
$z := x + 1;$
while $y + 1 \neq z$ do
  $t := (y + z)/2;$
  if $t^2 \leq x$ then
    $y := t;$
  else
    $z := t$
  fi
od
$\{\, y^2 \leq x < (y+1)^2 \,\}$

Hint: Use the invariant $y < z \leq x + 1 \wedge y^2 \leq x < z^2$.

Solution

$\{ F_1 \colon x \geq 0 \}$
$\{ F_{15} \colon Inv[z/x+1][y/0] \}$ (as↑)
$y := 0;$
$\{ F_{14} \colon Inv[z/x+1] \}$ (as↑)
$z := x + 1;$
$\{ F_3 \colon Inv \}$ (wht″)
while $y + 1 \neq z$ **do**
  $\{ F_4 \colon Inv \wedge y + 1 \neq z \wedge s = s_0 \}$ (wht″)
  $t := (y + z)/2;$
  $\{ F_{11} \colon Inv \wedge y + 1 \neq z \wedge s = s_0 \wedge t = (y + z)/2 \}$ (as↓)
  if $t^2 \leq x$ **then**
    $\{ F_{12} \colon Inv \wedge y + 1 \neq z \wedge s = s_0 \wedge t = (y + z)/2 \wedge t^2 \leq x \}$ (if↓)
    $\{ F_9 \colon (Inv \wedge 0 \leq s < s_0)[y/t] \}$ (as↑)
    $y := t;$
    $\{ F_7 \colon Inv \wedge 0 \leq s < s_0 \}$ (fi↑)
  else
    $\{ F_{13} \colon Inv \wedge y + 1 \neq z \wedge s = s_0 \wedge t = (y + z)/2 \wedge t^2 > x \}$ (if↓)
    $\{ F_{10} \colon (Inv \wedge 0 \leq s < s_0)[z/t] \}$ (as↑)
    $z := t$
    $\{ F_8 \colon Inv \wedge 0 \leq s < s_0 \}$ (fi↑)
  fi
  $\{ F_5 \colon Inv \wedge 0 \leq s < s_0 \}$ (wht″)
od
$\{ F_6 \colon Inv \wedge y + 1 = z \}$ (wht″)
$\{ F_2 \colon y^2 \leq x < (y + 1)^2 \}$

We choose the invariant $Inv \equiv y < z \leq x + 1 \wedge y^2 \leq x < z^2$ and the bound function $s := z - y$. We have to prove the following implications:

$$F_1 \colon x \geq 0 \Rightarrow F_{15} \colon Inv[z/x+1][y/0]$$
$$F_{12} \colon Inv \wedge y + 1 \neq z \wedge t = (y + z)/2 \wedge t^2 \leq x \Rightarrow F_{9a} \colon Inv[y/t]$$
$$F_{12} \colon Inv \wedge y + 1 \neq z \wedge t = (y + z)/2 \wedge t^2 \leq x \Rightarrow F_{9b} \colon 0 \leq s[y/t] < s$$
$$F_{13} \colon Inv \wedge y + 1 \neq z \wedge t = (y + z)/2 \wedge t^2 > x \Rightarrow F_{10a} \colon Inv[z/t]$$
$$F_{13} \colon Inv \wedge y + 1 \neq z \wedge t = (y + z)/2 \wedge t^2 > x \Rightarrow F_{10b} \colon 0 \leq s[z/t] < s$$
$$F_6 \colon Inv \wedge y + 1 = z \Rightarrow F_2 \colon y^2 \leq x < (y + 1)^2$$

$F_1 \Rightarrow F_{15}$**:**

$$x \geq 0 \Rightarrow Inv[z/x+1][y/0]$$
$$x \geq 0 \Rightarrow 0 < x + 1 \leq x + 1 \wedge 0^2 \leq x < (x + 1)^2$$

The conditions on the right-hand side are obviously true. Note that $0 < x+1$ and $0^2 \leq x$ hold because of $x \geq 0$.

$F_{12} \Rightarrow F_{9a}$**:**

$$\begin{aligned} & \mathit{Inv} \wedge y+1 \neq z \wedge t = (y+z)/2 \wedge t^2 \leq x \\ \Rightarrow\ & \mathit{Inv}[y/t] \end{aligned}$$

$$\begin{aligned} & y < z \leq x+1 \wedge y^2 \leq x < z^2 \wedge y+1 \neq z \wedge t = (y+z)/2 \wedge t^2 \leq x \\ \Rightarrow\ & t < z \leq x+1 \wedge t^2 \leq x < z^2 \end{aligned}$$

The conditions on the right-hand side also occur on the left-hand side except $t < z$, which we therefore have to prove. Since $y < z$ (first condition on the left-hand side) holds, the value of $t = (y+z)/2$ is at most $(z-1+z)/2 = z-1$, hence $t < z$ holds.

$F_{12} \Rightarrow F_{9b}$**:**

$$\begin{aligned} & \mathit{Inv} \wedge y+1 \neq z \wedge t = (y+z)/2 \wedge t^2 \leq x \\ \Rightarrow\ & 0 \leq s[y/t] < s \end{aligned}$$

$$\begin{aligned} & y < z \leq x+1 \wedge y^2 \leq x < z^2 \wedge y+1 \neq z \wedge t = (y+z)/2 \wedge t^2 \leq x \\ \Rightarrow\ & 0 \leq z-t < z-y \end{aligned}$$

The conclusion can be written as $y < t \leq z$. In the proof of the implication $12 \Rightarrow 9a$ we show $t < z$, hence we also have $t \leq z$. Moreover, in the proof of implication $13 \Rightarrow 10a$ we show $y < t$ using only premises also occurring in formula 12.

$F_{13} \Rightarrow F_{10a}$**:**

$$\begin{aligned} & \mathit{Inv} \wedge y+1 \neq z \wedge t = (y+z)/2 \wedge t^2 > x \\ \Rightarrow\ & \mathit{Inv}[z/t] \end{aligned}$$

$$\begin{aligned} & y < z \leq x+1 \wedge y^2 \leq x < z^2 \wedge y+1 \neq z \wedge t = (y+z)/2 \wedge t^2 > x \\ \Rightarrow\ & y < t \leq x+1 \wedge y^2 \leq x < t^2 \end{aligned}$$

The conditions on the right-hand side also occur on the left-hand side except $y < t \leq x+1$, which we therefore have to prove. The condition $t \leq x+1$ holds, since $t < z$ (see argument above) and $z \leq x+1$ (second condition on the left-hand side). To show $y < t$, note that $y < z$ and $y+1 \neq z$, i.e., $z \geq y+2$. Therefore the value of $(y+z)/2$ is at least $(y+y+2)/2 = y+1$, hence $y < t$.

$F_{13} \Rightarrow F_{10b}$**:**

$$\begin{aligned} & \mathit{Inv} \wedge y+1 \neq z \wedge t = (y+z)/2 \wedge t^2 > x \\ \Rightarrow\ & \mathit{Inv}[z/t] \end{aligned}$$

$$\begin{aligned} & y < z \leq x+1 \wedge y^2 \leq x < z^2 \wedge y+1 \neq z \wedge t = (y+z)/2 \wedge t^2 > x \\ \Rightarrow\ & 0 \leq t-y < z-y \end{aligned}$$

The conclusion can be written as $y \leq t < z$. In the proof of the implication $12 \Rightarrow 9a$ we show $t < z$. Moreover, in the proof of implication $13 \Rightarrow 10a$ we show $y < t$, hence we also have $y \leq t$.

$F_6 \Rightarrow F_2$**:** The right-hand side of

$$Inv \wedge y + 1 = z \Rightarrow y^2 \leq x < (y+1)^2$$

is part of the invariant if we replace z by $y + 1$.

Function computed by the program. The result of the program satisfies the postcondition $y^2 \leq x < (y+1)^2$, where x is the input and y is the output of the program. We use the postcondition to express y as a function of x.

$y^2 \leq x < (y+1)^2$	take the square root
$y \leq \sqrt{x} < y + 1$	definition of the floor function
$y = \lfloor \sqrt{x} \rfloor$	

Hence the program computes the integer square root of x.