Formal Verification of Software – Exercises

Harald Glanzer Bernd-Peter Ivanschitz Lukas Petermann

May 2012

Exercise 1 (1 point) Show that the givenTPL program is syntactically correct:
$x := x + y; if\ x < 0\ then\ abort; else\ while\ x \neq y\ do\ x := x + 1; y := y + 2; od\ fi$

- $\mathcal{P} \Rightarrow \mathcal{P};\mathcal{P}$
- $\Rightarrow \mathcal{V} := \mathcal{E};\mathcal{P}$
- $\Rightarrow x := (\mathcal{EBE});\mathcal{P}$
- $\Rightarrow x := (\mathcal{V} + \mathcal{V});\mathcal{P}$
- $\Rightarrow$ x:=x + y;$\mathcal{P}$
- $\Rightarrow$ x:=x + y; if $\mathcal{E}$ then $\mathcal{P}$ else $\mathcal{P}$ fi
- $\Rightarrow$ x:=x + y; if $(\mathcal{EBE})$ then $\mathcal{P}$ else $\mathcal{Q}$ if
- $\Rightarrow$ x:=x + y; if $\mathcal{V} < \mathcal{N}$ then $\mathcal{P}$ else $\mathcal{Q}$ if
- $\Rightarrow$ x:=x + y; if x $<$ 0 then $\mathcal{P}$ else $\mathcal{Q}$ if
- $\Rightarrow$ x:=x + y; if x $<$ 0 then abort; else $\mathcal{Q}$ if
- $\Rightarrow$ x:=x + y; if x $<$ 0 then abort; else while $\mathcal{E}$ do $\mathcal{P}$ od if
- $\Rightarrow$ x:=x + y; if x $<$ 0 then abort; else while $(\mathcal{EBE})$ do $\mathcal{P}$ od if
- $\Rightarrow$ x:=x + y; if x $<$ 0 then abort; else while $(\mathcal{E} \neq \mathcal{E})$ do $\mathcal{P}$ od if
- $\Rightarrow$ x:=x + y; if x $<$ 0 then abort; else while $(\mathcal{V} \neq \mathcal{E})$ do $\mathcal{P}$ od if
- $\Rightarrow$ x:=x + y; if x $<$ 0 then abort; else while $(\mathcal{V} \neq \mathcal{V})$ do $\mathcal{P}$ od if
- $\Rightarrow$ x:=x + y; if x $<$ 0 then abort; else while x$\neq$y do $\mathcal{P};\mathcal{P}$ od if
- $\Rightarrow$ x:=x + y; if x $<$ 0 then abort; else while x$\neq$y do $\mathcal{E};\mathcal{P}$ od if
- $\Rightarrow$ x:=x + y; if x $<$ 0 then abort; else while x$\neq$y do $(\mathcal{EBE});\mathcal{P}$ od if

- $\Rightarrow$ x:=x + y; if x $<$ 0 then abort; else while x$\neq$y do $(\mathcal{V}+\mathcal{N});\mathcal{P}$ od if
- $\Rightarrow$ x:=x + y; if x $<$ 0 then abort; else while x$\neq$y do x:= x + 1;$\mathcal{P}$; od if
- $\Rightarrow$ x:=x + y; if x $<$ 0 then abort; else while x$\neq$y do x:= x + 1;$\mathcal{E}$; od if
- $\Rightarrow$ x:=x + y; if x $<$ 0 then abort; else while x$\neq$y do x:= x + 1;($\mathcal{EBE}$); od if
- $\Rightarrow$ x:=x + y; if x $<$ 0 then abort; else while x$\neq$y do x:= x + 1;($\mathcal{V}$+$\mathcal{N}$); od if
- $\Rightarrow$ x:=x + y; if x $<$ 0 then abort; else while x$\neq$y do x:= x + 1; y:= y + 2; od if

Here the idea is to construct the wanted(given) program by starting with with an 'empty' program and extending this program by substitution until we get the final program.

Exercise 2 (1 point) Let σ be a state satisfying $\sigma(x) = \sigma(y) = 1$, and let p be the program given in exercise 3. Compute $[p]\,\sigma$, using

(a) the structural operational semantics

- $(p, \sigma) = (x := x + y; if\ x < 0\ then\ abort; else\ while\ x \neq y\ do\ x := x + 1; y := y + 2; od\ fi, \sigma)$

  Regel: $(p; q)]\sigma = (q)(p)\sigma$

  $(x := x + y, \sigma) \Rightarrow \sigma(x \rightarrow [x + y]\sigma) = \sigma_1$

- $\Rightarrow (if\ x < 0\ then\ abort; else\ while\ x \neq y\ do\ x := x + 1; y := y + 2; od\ fi, \sigma_1)$

  Regel: $[if\ e\ then\ p\ else\ q\ fi]\sigma = \begin{cases} (p.\sigma) \Rightarrow^* \sigma^a & if[e]\sigma \neq 0 \\ (p.\sigma) \Rightarrow^* \sigma^a & if[e]\sigma = 0 \end{cases}$

- $\Rightarrow (while\ x \neq y\ do\ x := x + 1; y := y + 2; od\ fi, \sigma_1)$
  $\Rightarrow (x := x + 1; y := y + 2, while..., \sigma_1)$
  $(x := x + 1; y := y + 2, \sigma_1)$
  $(x := x + 1, \sigma_1) \Rightarrow \sigma_1(x \rightarrow [x + 1]\sigma_1) = \sigma_2 \Rightarrow (y := y + 2, while..., \sigma_2)$
  $(y := y + 2, \sigma_1) \Rightarrow \sigma_2(y \rightarrow [y + 2]\sigma_2) = \sigma_3$

- $\Rightarrow (while\ x \neq y\ do\ x := x + 1; y := y + 2; od\ fi, \sigma_3)$

- $\Rightarrow \sigma_3$

  The States in detail:

- $\sigma : x \rightarrow 1, y \rightarrow 1$
- $\sigma_1 : x \rightarrow [x + y]\sigma = 2$
  $x \rightarrow 2, y \rightarrow 1$

- $\sigma_2 : x \rightarrow [x+1]\sigma_1 = 3$
  $x \rightarrow 3, y \rightarrow 1$
- $\sigma_3 : y \rightarrow [y+2]\sigma_2 = 3$
  $x \rightarrow 3, y \rightarrow 3$
  $[x \neq y]\sigma_3 = 0(false)$

(b) the natural semantics

- $p[\sigma] = [x := x + y; if\ x < 0\ then\ abort; else\ while\ x \neq y\ do\ x := x + 1; y := y + 2; od\ fi]\sigma$

  Regel: $[p;q]\sigma = [q][p]\sigma$

  $[x := x + y; if...]\sigma = [if...][x := x + y]\sigma$

  $\sigma : x \mapsto 1, y \mapsto 1$
- $= [if\ x < 0\ then\ abort; else\ while\ x \neq y\ do\ x := x + 1; y := y + 2; od\ fi]\sigma_1$

  Regel: $[if\ e\ then\ p\ else\ q\ fi]\sigma = \begin{cases} [p]\sigma, & if[e]\sigma \neq 0 \\ [q]\sigma, & if[e]\sigma = 0 \end{cases}$

  $\sigma_1 : x \mapsto 2, y \mapsto 1$

  $[x < 0]\sigma_1 = 0(false)$
- $= [while\ x \neq y\ do\ x := x + 1; y := y + 2; od]\sigma_2 =$

  Regel: $[while\ e\ do\ p\ od]\sigma = \begin{cases} [while\ e\ do\ p\ od][p]\sigma, & if[e]\sigma \neq 0 \\ \sigma, & if[e]\sigma = 0 \end{cases}$

  $\sigma_2 : x \mapsto 2, y \mapsto 1$

  $[x \neq y] = 1(true)$

  $= [while\ x \neq y...][y := y + 2; x := x + 1]\sigma_2$

  $\sigma_3 : x \mapsto [x+1]\sigma_2 = 3, y \mapsto 1$
- $= [while\ x \neq y...][y := y + 2]\sigma_3$

  $\sigma_4 : x \mapsto 3, y \mapsto [y+2]\sigma_3 = 3$
- $= [while\ x \neq y\ do...od]\sigma_4$

  $\sigma_4 : x \mapsto 3, y \mapsto = 3$

  $[x \neq y] = 0(false)$
- $= \sigma_4$

of TPL.

Exercise 3 (1 point) Let p be the following program:

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

Show that $\{ x = 2y \wedge y > 2 \} p \{ x = y \}$ is totally correct by computing the weakest precondition of the program.
We search for the weakest precondition witch satisfies:
$Wp(p, S_{out})p(S_{out})$

- $wp(x := x + y; if\ x < 0\ then\ abort; else\ while\ x \neq y\ do\ x := x + 1; y := y + 2; od\ fi$,x=y)
- $= wp(x := x + y; wp(if\ x < 0\ then ... fi$,x=y))
- = wp($x := x + y$; $(x < 0 \wedge wp(abort, x = y)\ ) \vee (x \geq y \wedge wp(while...., x = y)))$
- = wp($x := x + y$; $(x < 0 \wedge FALSE) \vee (x \geq y \wedge wp(*)\ ))$

- (*) = (while $x \neq y$ do $x := x + 1; y := y + 2;$ od , x=y)
- $\rightarrow F_0 = (x = y \wedge x = y)$
- $\rightarrow F_1 = (x \neq y \wedge wp(x := x + 1; y := y + 2, F_0)$=$(x \neq y \wedge wp(x := x + 1; wp(y := y + 2, F_0))$
  $= (x \neq y \wedge x = (y + 2) - 1) = (x \neq y \wedge x = y + 1)$
  guess:
- $\rightarrow F_i = (x \neq y \wedge wp(x := x + 1; y := y + 2, F_{i-1})$ =$(x \neq y \wedge wp(x := x + 1$ $wp(y := y + 2, F_{i-1}))$
  $= (x \neq y \wedge x = (y + i))$
- $\rightarrow F_{i+1} = (x \neq y \wedge wp(x := x + 1; y := y + 2, F_i)$ =$(x \neq y \wedge wp(x := x + 1$ $wp(y := y + 2, F_i))$
  $= (x \neq y \wedge x + 1 = (y + i + 2)) = (x \neq y \wedge x = (y + i + 1))$
  $\rightarrow wp(while...) = \exists i((i \geq 0) \wedge x = y + i) = ((i \geq 0) \wedge x - y = 1) = x - y \geq 0$

- = wp($x := x + y$; $(x < 0 \wedge FALSE$,$x = y\ ) \vee (x \geq y \wedge (*) wp(while...., x = y)))$
- = wp($x := x + y$; wp(while...),$x = y$)

- $= (x := x + y \wedge (x + y) - y \geq 0)$
- $= (x := x + y \wedge x \geq 0) =$ Weakest Precondition

Exercise 4 (1 point) Let p be the program given in exercise 3. Use the Hoare calculus to show that

$$\{\, x = 2y \wedge y > 2 \,\}\, p \,\{\, x = y \,\}$$

is totally correct.

Solution:

$$\dfrac{\dfrac{(x = 2y \wedge y > 2) \Rightarrow I \quad \overbrace{\{\, I \,\}\, x := x + y \,\{\, F \,\}}^{(as)}}{\{\, x = 2y \wedge y > 2 \,\}\, x := x + y \,\{\, F \,\}}\ \text{(lc)} \qquad \dfrac{\dfrac{(F \wedge x < 0) \Rightarrow false \quad \overbrace{\{\, false \,\}\, abort \,\{\, G \,\}}^{(abt)}}{\{\, F \wedge x < 0 \,\}\, abort \,\{\, G \,\}}\ \text{(lc)} \quad \dfrac{CONTINUE \quad \ldots}{\{\, F \wedge \neg(x < 0) \,\}\, while\ x \neq y\ do\ S\ od \,\{\, G \,\}}\ \text{(lc)}}{\{\, F \,\}\, if\ x < 0 \ldots \{\, G \,\}}\ \text{(if)}}{\{\, x = 2y \wedge y > 2 \,\}\, p \,\{\, x = y \,\}}\ \text{(sc)}$$

CONTINUE:

$$\dfrac{(F \wedge x \geq 0) \Rightarrow Inv \qquad \dfrac{\dfrac{\dfrac{\{\, Inv \wedge x \neq y \wedge t = t_0[x|x+1] \,\}\, x := x + 1 \,\{\, Inv \wedge x + 1 \neq y \wedge t = t_0 \,\}}{\{\, Inv \wedge x \neq y \wedge t = t_0 \,\}\, x := x + 1 \,\{\, SC2 \,\}}\ \text{(as)} \quad \dfrac{\dfrac{\{\, Inv \wedge x + 1 \neq y \wedge t = t_0[y|y+2] \,\}\, y := y + 2 \,\{\, \overbrace{Inv \wedge x \neq y + 1 \wedge t = t_0}^{B} \,\}}{\{\, SC2 \,\}\, y := y + 2 \,\{\, Inv \wedge 0 \leq t < t_0 \,\}}\ \text{(as)} \quad B \Rightarrow (Inv \wedge 0 \leq t < t_0)}{\{\, SC2 \,\}\, y := y + 2 \,\{\, Inv \wedge 0 \leq t < t_0 \,\}}\ \text{(lc}}{\{\, Inv \wedge x \neq y \wedge t = t_0 \,\}\, x := x + 1; y := y + 2 \,\{\, Inv \wedge 0 \leq t < t_0 \,\}}\ \text{(sc)}}{\{\, Inv \,\}\, while\ x \neq y\ do\ x := x + 1; y := y + 2\ od \,\{\, Inv \wedge x = y \,\}}\ \text{(wh)}}{\{\, F \wedge (x \leq 0) \,\}\, while\ x \neq y\ do\ x := x + 1; y := y + 2\ od \,\{\, G \,\}}\ \text{(lc)}$$

- $x = 2y \wedge y > 2 \Rightarrow I$
  $x = 2y \wedge y > 2 \Rightarrow F[x|x + y]$
  $x = 2y \wedge y > 2 \Rightarrow x + y = 2y \wedge y > 2$
  $x = 2y \wedge y > 2 \Rightarrow x = y \wedge y > 2$

- $F \wedge x < 0 \Rightarrow False$
  $x = y \wedge y > 2 \wedge x < 0 \Rightarrow False$

- $F \wedge x \geq 0 \Rightarrow Inv (= TRUE\ by\ Assumption)$
  $x = y \wedge y \geq 2 \wedge x \geq 0 \Rightarrow TRUE$

- $Boundfunction\ t = y - x$

Exercise 5 (1 point) Extend our toy language 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:

For the syntax we have to replace P from TLP with :
P::= skip | ... | while e do P od | assert e
Semantics:

Transition Relation for TPL:
Since we have to treat assert e like an first class citizen we are not allows to use statements like skip and abort.

For NS:

$$[\text{ assert e}]\sigma = \begin{cases} \sigma, & if[e]\sigma \neq 0 \\ undefined, & if[e]\sigma = 0 \end{cases}$$

For SOS

$$(asserte, \sigma) = \begin{cases} \sigma, & if[e]\sigma \neq 0 \\ undefined, & if[e]\sigma = 0 \end{cases}$$

Hoare calculus:

Partial correctness

First we show partial correctness. Therefor we use the wlp as follows:
$wpl(\mathsf{assert\ e,G}) = e \Rightarrow G$
We use the Hoare calculus and replace the assert rule with an if statement.

$$\dfrac{\dfrac{\dfrac{\dfrac{---}{\{F\wedge e\}\Rightarrow F^a\ \{F^a\}\mathsf{skip}\{F^a\}\ F^a\Rightarrow G}\,()}{\{F\wedge e\}\mathsf{skip}\ G}\,(lc)\qquad \dfrac{\dfrac{---}{\{F\wedge\neg e\}\Rightarrow F^a\ \{F^a\}\ \mathsf{abort}\ G}\,()}{\{F\wedge\neg e\}\ \mathsf{abort}\, G}\,(lc)}{\{F\}\mathsf{if}\ e\ \mathsf{then\ skip\ else\ abort\ fi}\{G\}}\,(if)}{\{F\}\mathsf{assert\ e}\{G\}}\,(rp)$$

We see that we have a problem with the assert false statement since we can not reach the postcondition G. Those the rule assert false is not semantically equivalent to the "while true do skip od" which we now is semantically equivalent to the abort statement. Now we have to show that we can reach the postcondition G from each of the Statements of the Hoare calculus $\{F \wedge e\}$ $\{F \wedge \neg e\}$.

We show that:

$$\begin{cases}\{F\wedge e\} \\ \{F\wedge\neg e\}\end{cases} \Rightarrow F^a$$

$\equiv ((F \wedge e) \vee (F \wedge \neg e)) \Rightarrow F^a$
$\equiv \neg((F \wedge e) \vee (F \wedge \neg e)) \vee F^a$
$\equiv ((\neg F \vee \neg e) \wedge (\neg F \vee e)) \vee F^a$
$\equiv ((\neg F \vee \neg e \vee F^a) \wedge (\neg F \vee e \vee F^a))$
$\equiv (F \Rightarrow (\neg e \vee F^a)) \wedge (F \Rightarrow (e \vee F^a))$

Since $(F \Rightarrow (e \vee F^a)$ is ture because e is true and F^a is not defined:

$\equiv (F \Rightarrow (\neg e \vee F^a)) \wedge$ true
$\equiv (F \Rightarrow (\neg e \Rightarrow F^a))$

now we use the fact that $F^a = G$

$\equiv (F \Rightarrow (\neg e \Rightarrow G))$

Now we can see that : $\dfrac{F \Rightarrow (e \Rightarrow G)}{\{F\}\mathsf{assert\ e}\{G\}}\,(pc)$

and we can see that the statement is partial correct.

Total correctness:

$$\dfrac{\dfrac{\dfrac{\dfrac{- - -}{\{F \wedge e\} \Rightarrow F^a \ \{F^a\}\mathsf{skip}\{F^a\} \ F^a \Rightarrow G}}{\{F \wedge e\}\mathsf{skip}\ G}(lc) \dfrac{\dfrac{\dfrac{- - -}{\{F^a\} \Rightarrow \text{false} \ \{\text{false}\}abort\{G\}}(lc)}{\{F \wedge \neg e\} \Rightarrow F^a \ \{F^a\}\ \mathsf{abort}\ G}(lc)}{\{F \wedge \neg e\}\ \mathsf{abort}\, G}}{\{F\}\mathsf{if}\ e\ \mathsf{then\ skip\ else\ abort\ fi}\{G\}}(if)}{\{F\}\mathsf{assert\ e}\{G\}}(rp)$$

For the total correctness we use a different abort rule. So we show that:

$$\begin{cases}\{F \wedge e\} \\ \{F \wedge \neg e\}\end{cases} \Rightarrow F^a$$
$\equiv ((F \wedge e) \vee (F \wedge \neg e)) \Rightarrow F^a$
$\equiv \neg((F \wedge e) \vee (F \wedge \neg e)) \vee F^a$
$\equiv ((\neg F \vee \neg e) \wedge (\neg F \vee e)) \vee F^a$
$\equiv ((\neg F \vee \neg e \vee F^a) \wedge (\neg F \vee e \vee F^a))$
$\equiv ((\neg F \vee \neg e \vee G) \wedge (\neg F \vee e \vee \text{false}))$
$\equiv \neg F \vee ((\neg e \vee G) \wedge (e \vee \text{false}))$
$\equiv \neg F \vee ((\neg e \vee G) \wedge e)$
$\equiv \neg F \vee ((\neg e \wedge e) \vee (G \vee e))$
$\equiv \neg F \vee (G \vee e)$
$\equiv F \Rightarrow (G \wedge e)$

Therefor we can compute : $\dfrac{F \Rightarrow (e \wedge G)}{\{F\}\mathsf{assert\ e}\{G\}}(tc)$

Exercise 6 (1 point) Verify that the following program doubles the value of x. For which inputs does it terminate? Choose appropriate pre- and postconditions and show that the assertion is totally correct. Use $y = 2x_0 + x$ as a starting point for the invariant, where x_0 denotes the initial value of x.

```
$y := 3x$;
while $2x \neq y$ do
    $x := x + 1$;
    $y := y + 1$;
od
```

Solution:

```
{ 1: $a \geq 0$ }
{ 9: $Inv[y/3x]$ }
$y := 3x;$
{ 3: $Inv : y = 2x_0 + x$ }
while $2x \neq y$ do
   { 4: $Inv \wedge 2x \neq y \wedge t = t_0$ }
   { 8: $Inv \wedge 0 \leq t \leq t_0[y/y+1][x/x+1]$ }
   $x := x + 1;$
   { 7: $Inv \wedge 0 \leq t \leq t_0[y/y+1]$ }
   $y := y + 1;$
   { 5: $Inv \wedge 0 \leq t \leq t_0$ }
od
{ 6: $Inv \wedge 2x = y$ }
{ 2: $2 * x_0 = x$ }
```

prove $4 \Rightarrow 8$:
first step is prooving partial correctness:

$$\begin{gathered}
Inv \wedge 2x \neq y \Rightarrow Inv[y/y+1][x/x+1] \\
y = 2x_0 + x \wedge 2x \neq y \Rightarrow y = 2x_0 + x[y/y+1][x/x+1] \\
2x \neq 2x_0 + x \Rightarrow (2x_0 + x + 1) = 2x_0 + (x+1) \\
x \neq 2x_0 \Rightarrow 2x_0 + x + 1 = 2x_0 + x + 1
\end{gathered}$$

The right side is always valid and therefore the assertion is valid

second step is prooving termination:
The bound function t is set to $t = y - 2x + 1$.
$Inv \wedge 2x \neq y \wedge t = t_0 \Rightarrow 0 \leq t[y/y+1][x/x+1] < t_0$
$y = 2x_0 + x \wedge 2x \neq y \wedge t = t_0 \Rightarrow 0 \leq y - 2x + 1[y/y+1][x/x+1] < t_0$
$2x \neq 2x_0 + x \wedge t = t_0 \Rightarrow 0 \leq (y+1) - 2(x+1) + 1 < t_0$
$x \neq 2x_0 \Rightarrow 0 \leq (y+1) - 2(x+1) + 1 < y - 2x + 1$
$x \neq 2x_0 \Rightarrow 0 \leq 2x_0 + x - 2x < 2x_0 + x - 2x + 1$
$x \neq 2x_0 \Rightarrow 0 \leq 2x_0 - x < 2x_0 - x + 1$
not valid because $2x_0 - x$ can be smaller then 0. There we need to extend the invariant with $x \leq 2x_0$.

Our new invariant is: $y = 2x_0 + x \wedge x \leq 2x_0$. Now we need to start the calculation again with the new invariant.

prove $4 \Rightarrow 8$:
first step is prooving partial correctness:

$$Inv \wedge 2x \neq y \Rightarrow Inv[y/y+1][x/x+1]$$

$$y = 2x_0 + x \wedge x \leq 2x_0 \wedge 2x \neq y \Rightarrow y = 2x_0 + x[y/y+1][x/x+1]$$
$$2x \neq 2x_0 + x \wedge x \leq 2x_0 \Rightarrow (2x_0 + x + 1) = 2x_0 + (x+1)$$
$$x \neq 2x_0 \wedge x \leq 2x_0 \Rightarrow 2x_0 + x + 1 = 2x_0 + x + 1$$

The right side is always valid and therefore the assertion is valid

second step is prooving termination:
The bound function t is set to $t = y - 2x + 1$.

$$Inv \wedge 2x \neq y \wedge t = t_0 \Rightarrow 0 \leq t[y/y+1][x/x+1] < t_0$$
$$y = 2x_0 + x \wedge x \leq 2x_0 \wedge 2x \neq y \wedge t = t_0 \Rightarrow 0 \leq y - 2x + 1[y/y+1][x/x+1] < t_0$$
$$2x \neq 2x_0 + x \wedge x \leq 2x_0 \wedge t = t_0 \Rightarrow 0 \leq (y+1) - 2(x+1) + 1 < t_0$$
$$x \neq 2x_0 \wedge x \leq 2x_0 \Rightarrow 0 \leq (y+1) - 2(x+1) + 1 < y - 2x + 1$$
$$x \neq 2x_0 \wedge x \leq 2x_0 \Rightarrow 0 \leq 2x_0 + x - 2x < 2x_0 + x - 2x + 1$$
$$x \neq 2x_0 \wedge x \leq 2x_0 \Rightarrow 0 \leq 2x_0 - x < 2x_0 - x + 1$$
$$x < 2x_0 \Rightarrow 0 \leq 2x_0 - x < 2x_0 - x + 1$$

valid because $2x_0 - x < 2x_0 - x + 1$ is always valid and if the left side is true, then $0 \leq 2x_0 - x$ is also valid.

prove $1 \Rightarrow 9$:

$$a \geq 0 \Rightarrow Inv[y/3x]$$
$$a \geq 0 \Rightarrow y = 2x_0 + x \wedge x \leq 2x_0[y/3x]$$
$$a \geq 0 \Rightarrow 3x = 2x_0 + x \wedge x \leq 2x_0$$
$$a \geq 0 \Rightarrow 2x = 2x_0 \wedge x \leq 2x_0$$

valid because at this point, the beginning of the program, $x = x_0$ and therefore $2x = 2x_0$ and $x \leq 2x_0$ is valid. Because the whole right side is always valid, no matter what stands on the left side the implications is valid.

prove $6 \Rightarrow 2$:

$$Inv \wedge 2x = y \Rightarrow 2 * x_0 = x$$
$$y = 2x_0 + x \wedge x \leq 2x_0 \wedge 2x = y \Rightarrow 2 * x_0 = x$$
$$2x = 2x_0 + x \wedge x \leq 2x_0 \Rightarrow 2 * x_0 = x$$
$$x = 2x_0 \wedge x \leq 2x_0 \Rightarrow 2 * x_0 = x$$

valid because when the left side is true, then the right side is also true, because the have the same property $x = 2x_0$.

Exercise 7 (1 point) Show that the following correctness assertion is totally correct. Describe the function computed by the program if we consider a as its input and c as its output.

```
{ 1: $a \geq 0$ }
$b := 1$;
$c := 0$;
{ $Inv$: $b = (c+1)^3 \wedge 0 \leq c^3 \leq a$ }
while $b \leq a$ do
   $d := 3 * c + 6$;
   $c := c + 1$;
   $b := b + c * d + 1$
od
{ 2: $c^3 \leq a < (c+1)^3$ }
```

Solution:

```
{ 1: $a \geq 0$ }
{ 11: $Inv[c/0][b/1]$ }
$b := 1$;
{ 10: $Inv[c/0]$ }
$c := 0$;
{ $Inv$: $b = (c+1)^3 \wedge 0 \leq c^3 \leq a$ }
while $b \leq a$ do
   { 4: $Inv \wedge b \leq a \wedge t = t_0$ }
   { 9: $(Inv \wedge 0 \leq t \leq t_0)[b/b + c * d + 1][c/c + 1][d/3 * c + 6]$ }
   $d := 3 * c + 6$;
   { 8: $(Inv \wedge 0 \leq t \leq t_0)[b/b + c * d + 1][c/c + 1]$ }
   $c := c + 1$;
   { 7: $(Inv \wedge 0 \leq t \leq t_0)[b/b + c * d + 1]$ }
   $b := b + c * d + 1$
   { 5: $Inv \wedge 0 \leq t \leq t_0$ }
od
{ 6: $Inv \wedge b > a$ }
{ 2: $c^3 \leq a < (c+1)^3$ }
```

prove $1 \Rightarrow 11$:

$$a \geq 0 \Rightarrow Inv[z/0][b/1]$$
$$a \geq 0 \Rightarrow b = (c+1)^3 \wedge 0 \leq c^3 \leq a[z/0][b/1]$$
$$a \geq 0 \Rightarrow 1 = (0+1)^3 \wedge 0 \leq 0^3 \leq a$$
$$a \geq 0 \Rightarrow 1 = 1 \wedge 0 \leq 0 \leq a$$
$$a \geq 0 \Rightarrow 0 \leq 0 \leq a$$
$$a \geq 0 \Rightarrow 0 \leq a$$

valid

prove $4 \Rightarrow 9$:
first step is prooving partial correctness:

$$
\begin{gathered}
Inv \wedge b \leq a \Rightarrow Inv[b/b+c*d+1][c/c+1][d/3*c+6] \\
b=(c+1)^3 \wedge 0 \leq c^3 \leq a \wedge b \leq a \Rightarrow b=(c+1)^3 \wedge 0 \leq c^3 \leq a[b/b+c*d+1][c/c+1][d/3*c+6] \\
0 \leq c^3 \leq a \wedge (c+1)^3 \leq a \Rightarrow 0 \leq c^3 \leq a[b/b+c*d+1][c/c+1][d/3*c+6] \\
0 \leq (c+1)^3 \leq a \Rightarrow 0 \leq (c+1)^3 \leq a \\
\text{valid}
\end{gathered}
$$

second step is prooving termination:
The bound function t is set to $t = a - c^3$.

$$
\begin{gathered}
Inv \wedge b \leq a \wedge t = t_0 \Rightarrow 0 \leq t[b/b+c*d+1][c/c+1][d/3*c+6] < t_0 \\
b=(c+1)^3 \wedge 0 \leq c^3 \leq a \wedge b \leq a \wedge t = t_0 \Rightarrow 0 \leq a - c^3[b/b+c*d+1][c/c+1][d/3*c+6] < t_0 \\
0 \leq c^3 \leq a \wedge (c+1)^3 \leq a \wedge t = t_0 \Rightarrow 0 \leq a-(c+1)^3 < t_0 \\
0 \leq c^3 \leq a \wedge (c+1)^3 \leq a \Rightarrow 0 \leq a-(c+1)^3 < t \\
0 \leq (c+1)^3 \leq a \Rightarrow 0 \leq a-(c+1)^3 < a - c^3
\end{gathered}
$$

valid because a is always bigger as $(c+1)^3$ and therefore $a-(c+1)^3$ is always ≥ 0 and $a-(c+1)^3 < a - c^3$ is always valid.

prove $6 \Rightarrow 2$:

$$
\begin{gathered}
Inv \wedge b > a \Rightarrow c^3 \leq a < (c+1)^3 \\
b = (c+1)^3 \wedge 0 \leq c^3 \leq a \wedge a < b \Rightarrow c^3 \leq a < (c+1)^3 \\
0 \leq c^3 \leq a \wedge a < (c+1)^3 \Rightarrow c^3 \leq a < (c+1)^3 \\
0 \leq c^3 \leq a < (c+1)^3 \Rightarrow c^3 \leq a < (c+1)^3 \\
\text{valid}
\end{gathered}
$$

The function computed by the program is $\lfloor \sqrt[3]{a} \rfloor$.

Exercise 8 (1 point) Prove that the rule

$$
\frac{\{\, Inv \wedge e \,\}\, p \,\{\, Inv \,\}}{\{\, Inv \,\}\ \mathsf{while}\ e\ \mathsf{do}\ p\ \mathsf{od}\ \{\, Inv \wedge \neg e \,\}}\ ^{(\mathrm{wh})}
$$

is correct regarding partial correctness, i.e., show that $\{\, Inv \,\}$ while e do p od $\{\, Inv \wedge \neg e \,\}$ is partially correct whenever $\{\, Inv \wedge e \,\}\, p \,\{\, Inv \,\}$ is partially correct.

Exercise 9 (2 points) Determine the weakest liberal precondition of while-loops, i.e., find a formula equivalent to wlp(while e do p 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

For the wlp(while e do p od$, G)$ is the weakest precondition defined as follows:

All states such that loop terminates after a finite number of iterations in a G-state.
$\{F_i\}$ . . . set of states such that$p$ executes i times and leads to G-state

- 0 iterations: $F_0 = \neg e \vee G$
- 1 iteration: $F_1 = e \wedge wp(p, F_0)$
- 2 iterations:$F_2 = e \wedge wp(p, F_1)$
- ...
- ...
- i iterations: $Fi = e \wedge wp(p, F_{i-1})(for\ i > 0)$

$F_i = e^w p(p, F_{i1})$ . . . set of states such that

- p is executed once (because e is true), resulting in a state where
- i 1 further iterations will lead to a G-state.

For the WLP we some adjustments have to be made. For the 0 iterations we have to modify the Wp. In detail we have to change the first iteration step.

- 0 iterations: $F_0 = (\neg e \Rightarrow G)$
- 1 iteration: $F_1 = e \wedge wp(p, F_0)$
- 2 iterations:$F_2 = e \wedge wp(p, F_1)$
- ...
- ...
- i iterations: $Fi = e \wedge wp(p, F_{i-1})(for\ i > 0)$

The computation of the wp is similar to the program witch was presented in the lecture. Since we only need to change the termination step we can use the wp from the lecture and compute the wlp from the program.

The weakest precondition from the lecture was defined as $(y \geq 0 \wedge (x = 0 \vee y_0 = y))$. From this wp the wlp is defined as $(x = 0 \vee y_0 = y) \vee y < 0)$.