Assignment 1

Consider the following IMP program s_1:

```
i := 0;
s := 0;
while i <= x do
  s := s + i;
  i := i + 1
end
```

You can find the corresponding Viper program on the website: `program_1_adding_numbers.vpr`.

Task 1.1. Assume $\mathtt{x} \geq 0$ initially (i.e. as a pre-condition). What is the value of `s` at the end of the function? Write a post-condition to ensure that `s` has the right value after completion (keyword `ensures` in Viper). Also add the pre-condition to the program (keyword `requires` in Viper).

Solution. Mathematics tells us that $\mathtt{s} = \frac{(\mathtt{x}+1)\mathtt{x}}{2}$. See the solution program for the pre-condition and post-condition in Viper.

Task 1.2. Find a suitable loop invariant which enables Viper to prove the post-condition given the pre-condition.

Solution. A suitable loop invariant is:

$$\mathtt{s} = \frac{(\mathtt{i}-1)\mathtt{i}}{2} \wedge \mathtt{i} \leq \mathtt{x}+1$$

Observe that after the execution `i` equals $\mathtt{x}+1$ to guess the first conjunct. The second conjunct is needed to actually prove $\mathtt{i} = \mathtt{x}+1$ after the execution (we get $\mathtt{i} \geq \mathtt{x}+1$ from the negated loop condition).

Task 1.3. Prove that

$$\{\ \mathtt{x} = X \wedge \mathtt{x} \geq 0\ \}\ s_1\ \{\ \mathtt{s} = \frac{(X+1)X}{2}\ \}$$

Solution. We choose the invariant to be

$$\mathtt{i} \leq \mathtt{x}+1 \wedge \mathtt{s} = \frac{(\mathtt{i}-1)\mathtt{i}}{2} \wedge \mathtt{x} = X$$

Axiomatic Semantics

(1)

[a] $\frac{x \cdot (x+1)}{2}$ (kleiner Gauss)

[b] → um die loop invariante zu finden, ist es oft eine gute Idee, sich die Werte des Vars für versch. loop-iterationen anzuschauen

loop iteration	i	s
0	0	0
1	1	0
2	2	1
3	3	1+2
⋮		
n	n	$1+2+\dots+(n-1)$

also $s = \frac{(i-1) \cdot i}{2}$

Am Ende wollen wir dann

$\{\dots \wedge s = \frac{(i-1)\,i}{2} \wedge \neg(i \leq x)\}$ ← das gilt nach Ende der loop

$\vDash \{s = \frac{x \cdot (x+1)}{2}\}$

Wir müssen also irgie $i = x+1$ am Ende schlussfolgern können.
Trick: nehme $i \leq x+1$ in die loop invariante, denn
$\{i \leq x+1 \wedge \neg(i \leq x)\} \vDash \{i = x\}$

Also loop invariant: $s = \frac{(i-1) \cdot i}{2} \wedge i \leq x+1$

↳ Finden der loop invariante braucht Übung und bleibt nicht-trivial

c Axiomatic Semantics Beweise werden oft "bottom-up" geschrieben, also man fängt mit der Postcond. an und arbeitet sich nach oben

Meist ist es klug, die loop invariante davor zu bestimmen

Die Syntax der Regeln muss exakt eingehalten werden! Sh. ③
→ wie damals bei ND trees: kein $A \wedge B \equiv B \wedge A$ oder $\neg(i<0) \equiv i \geq 0$

$\{x = X \wedge x \geq 0\}$

$\vDash \{0 \leq x+1 \wedge 0 = \frac{(0-1)\cdot 0}{2} \wedge x = X\}$ ← der Schritt erscheint sehr willkürlich, aber lies den Beweis von unten nach oben

$i := 0;$

$\{i \leq x+1 \wedge 0 = \frac{(i-1)\cdot i}{2} \wedge x = X\}$

$s := 0;$

$\{i \leq x+1 \wedge s = \frac{(i-1)\cdot i}{2} \wedge x = X\}$

: unsere loop invariante sh. Wh_{Ax} Regel

while $i <= x$ do

$\{i \leq x \wedge i \leq x+1 \wedge s = \frac{(i-1)\cdot i}{2} \wedge x = X\}$

$\vDash \{i+1 \leq x+1 \wedge s+i = \frac{((i+1)-1)\,(i+1)}{2} \wedge x = X\}$

$s := s + i;$

$\vDash \{i+1 \leq x+1 \wedge s = \frac{((i+1)-1)\cdot(i+1)}{2} \wedge x = X\}$

$i := i+1$

$\{i \leq x+1 \wedge s = \frac{(i-1)\cdot i}{2} \wedge x = X\}$

Ass_{Ax} Regel: ersetze jedes i durch ein i+1

end

$\{\neg(i \leq x) \wedge i \leq x+1 \wedge s = \frac{(i-1)\cdot i}{2} \wedge x = X\}$ ← das schleppen wir in Beweisen oft mit

$\vDash \{s = \frac{(X+1)X}{2}\}$

komplizierteste Schritte erklären:

* $i \leq x \Rightarrow i+1 \leq x+1$

$s+i = \frac{(i-1)\cdot i}{2} + i$

$= \frac{i^2 - i + 2i}{2}$

$= \frac{(i+1)\cdot i}{2} = \frac{((i+1)-1)\cdot(i+1)}{2}$

→ Und jetzt nochmal von oben nach unten durchgehen!

d $x - i$ will slowly decrease w/ each loop iteration & $i \leq x \Rightarrow x - i \geq 0$ (see WhTot rule)

Assignment 2 (Greatest Common Divisor)

Consider the following program s computing the greatest common divisor (gcd) of two given positive integers:

```
b := x;
c := y;
while b # c do
```

```
    if b < c then
      c := c - b
    else
      b := b - c
    end
  end;
  z := b
```

You can find the corresponding Viper program on the website: `program_2_gcd.vpr`.

Note: In case it helps you to think about the questions, you might want to recall the definition of the *total* function gcd: For *positive* integers x and y, the number z is the greatest common divisor of x and y iff $z|x$ and $z|y$ and there is no z', with $z' > z$, such that $z'|x$ and $z'|y$. Here, $z|x$ means that z divides x, i.e., $z \cdot k = x$, for some $k \in \mathbb{N}$.

Note: For the tasks below, you can write $gcd(m, n)$ in assertions, to denote the actual greatest common divisor of two positive integers m and n. You may assume that $\forall n \cdot gcd(n, n) = n$ and $\forall m, n \in \mathbb{N}^+ \cdot gcd(m + n, n) = gcd(m, n) = gcd(m, m + n)$ (equivalently - $\forall m, n \in \mathbb{N}^+ \cdot m > n \Rightarrow gcd(m - n, n) = gcd(m, n) = gcd(m, m - n)$).

Task 2.1. Formalise the claim that the above program computes the gcd of `x` and `y` as pre- and postcondition $\mathbf{P}$ and $\mathbf{Q}$, respectively.

Solution. The pre- and postconditions are:

$$\mathbf{P} \equiv \mathtt{x} = X \wedge \mathtt{y} = Y \wedge X > 0 \wedge Y > 0$$
$$\mathbf{Q} \equiv \mathtt{z} = gcd(X, Y)$$

Task 2.2. Find a suitable invariant for the loop.

Hint: Consider using a relationship between the input variables `x`, `y` and the 'loop' variables `b` and `c` as part of your loop invariant.

Solution. A suitable loop invariant is:

$$gcd(\mathtt{x}, \mathtt{y}) = gcd(\mathtt{b}, \mathtt{c}) \wedge \mathtt{b} > 0 \wedge \mathtt{c} > 0 \wedge \mathtt{x} = X \wedge \mathtt{y} = Y$$

Task 2.3. Give a suitable loop variant to prove that the program terminates.

Solution. $b + c$: Either b or c decreases, while the other stays constant.

Task 2.4. Prove that

$$\{\ \mathtt{x} = X_0\ \wedge\ \mathtt{y} = Y_0\ \wedge\ X_0 > 0\ \wedge\ Y_0 > 0\ \}\ s\ \{\ \Downarrow \mathtt{z} = \mathit{gcd}(X_0, Y_0)\ \}.$$

② Greatest Common Divisor

b Wieder: Was gilt vor der ersten und nach jeder Ausführung der Loop?

$$gcd(x_0,y_0) = gcd(b,c) \wedge \underbrace{b>0 \wedge c>0}_{\text{brauchen wir später}}$$

c loop variant: $b+c$ (wird in jeder Iteration kleiner, falls $b \neq c$ und loop invariante halten, dann $b+c \geq 0$)

d

$\{x = x_0 \wedge y = y_0 \wedge x_0 > 0 \wedge y_0 > 0\}$

$\vdash \{gcd(x_0,y_0) = gcd(x,y) \wedge x>0 \wedge y>0\}$

loop invariant

loop variant

$b := x;$

$\{gcd(x_0,y_0) = gcd(b,y) \wedge b>0 \wedge c>0\}$

$c := y;$

$\{gcd(x_0,y_0) = gcd(b,c) \wedge b>0 \wedge c>0\} \equiv P$ ← nenne loop invariante mal P

while $b \neq c$ do ← loop cond.

(loop variant $b+c$, since $b \neq c \wedge P \vDash b+c \geq 0$)

↳ hinschreiben?

$\{ b \neq c \wedge gcd(x_0,y_0) = gcd(b,c) \wedge b>0 \wedge c>0 \wedge b+c = z\}$

↑ sh. WhTot$_{IA}$ Regel

if $b<c$ then

$\{ b<c \wedge b \neq c \wedge gcd(x_0,y_0) = gcd(b,c) \wedge b>0 \wedge c>0 \wedge b+c = z\}$

$\vdash^{*} \{ gcd(x_0,y_0) = gcd(b,c-b) \wedge b>0 \wedge c-b>0 \wedge b+(c-b) < z\}$

$c := c-b$

$\{\Downarrow gcd(x_0,y_0) = gcd(b,c) \wedge b>0 \wedge c>0 \wedge b+c < z\}$ *

* $0<b<c$ w/ hint: $gcd(b,c) = gcd(b,c-b)$

else

$\{ \neg(b<c) \wedge b \neq c \wedge gcd(x_0,y_0) = gcd(b,c) \wedge b>0 \wedge c>0 \wedge b+c = z\}$

$\vdash^{**} \{ gcd(x_0,y_0) = gcd(b-c,c) \wedge b-c>0 \wedge c>0 \wedge (b-c)+c < z\}$

$b := b-c$

$\{\Downarrow gcd(x_0,y_0) = gcd(b,c) \wedge b>0 \wedge c>0 \wedge b+c < z\}$

** $0<c<b$ w/ hint $gcd(b,c) = gcd(b-c,c)$

end

$\{\Downarrow \gcd(x_0, y_0) = \gcd(b, c) \wedge b > 0 \wedge c > 0 \wedge b + c < z\}$ *

(sh. WhTot_{Ax} Regel)

end;

$\{\Downarrow \neg(b \neq c) \wedge \gcd(x_0, y_0) = \gcd(b, c) \wedge b > 0 \wedge c > 0\}$

$\vDash \{\Downarrow b = \gcd(x_0, y_0)\}$

$z := b$

$\{\Downarrow z = \gcd(x_0, y_0)\}$

merke: $\Downarrow$ kommt durch WhTotal

* merke: hier steht exakt das gleiche, weil die If_{Ax} Regel das so fordert

Assignment 3

Consider the following IMP program s:

```
i := 0;
r := 1;
while i < k do
  i := i + 1;
  r := r * n
end
```

7 Fehler

The following *incorrect* total correctness proof outline contains various errors.