Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

D. Basin and P. Müller

Formal Methods and Functional Programming

Solutions of Exercise Sheet 11: Axiomatic Semantics

Assignment 1

a)

Recall from the lecture the rule for a while statement:

$$\frac{\{\,\mathcal{B}[\![b]\!] \land P\,\}\ s\ \{\,P\,\}}{\{\,P\,\}\ \texttt{while}\ b\ \texttt{do}\ s\ \texttt{end}\ \{\,\neg\mathcal{B}[\![b]\!] \land Q\,\}}$$

where P is called the loop invariant.

The formula (3) is not a loop invariant of the while loop, since it's not preserved by the loop's body:

```
while i < k do
    $\{\, \mathtt{i} \geq 0 \ \land\ \mathtt{i} < \mathtt{k} \ \land\ \mathtt{r} = \mathtt{n}^{\mathtt{i}} \ \land\ \mathtt{i} < \mathtt{k} \,\}$
        $\not\Rightarrow$ FAILS if i = k - 1
    $\{\, \mathtt{i}+1 \geq 0 \ \land\ \mathtt{i}+1 < \mathtt{k} \ \land\ \mathtt{rn} = \mathtt{n}^{\mathtt{i}+1} \,\}$
        i := i + 1;
        r := r * n
    $\{\, \mathtt{i} \geq 0 \ \land\ \mathtt{i} < \mathtt{k} \ \land\ \mathtt{r} = \mathtt{n}^{\mathtt{i}} \,\}$
end
```

The formula (2) is a loop invariant of the while loop. This is shown by the following derivation tree:

$$\dfrac{\dfrac{\{0 \leq \mathtt{i} < \mathtt{k} \land \mathtt{r} = \mathtt{n}^{\mathtt{i}}\}\ \mathtt{i} := \mathtt{i}+1\ \{0 \leq \mathtt{i}-1 < \mathtt{k} \land \mathtt{r} = \mathtt{n}^{\mathtt{i}-1}\} \quad \dfrac{\{1 \leq \mathtt{i} \leq \mathtt{k} \land \mathtt{r}*\mathtt{n} = \mathtt{n}^{\mathtt{i}}\}\ \mathtt{r} := \mathtt{r}*\mathtt{n}\ \{0 \leq \mathtt{i} \leq \mathtt{k} \land \mathtt{r} = \mathtt{n}^{\mathtt{i}}\}}{\{0 \leq \mathtt{i}-1 < \mathtt{k} \land \mathtt{r} = \mathtt{n}^{\mathtt{i}-1}\}\ \mathtt{r} := \mathtt{r}*\mathtt{n}\ \{0 \leq \mathtt{i} \leq \mathtt{k} \land \mathtt{r} = \mathtt{n}^{\mathtt{i}}\}}\ \text{conseq.}}{\{0 \leq \mathtt{i} < \mathtt{k} \land \mathtt{r} = \mathtt{n}^{\mathtt{i}}\}\ \mathtt{i} := \mathtt{i}+1; \mathtt{r} := \mathtt{r}*\mathtt{n}\ \{0 \leq \mathtt{i} \leq \mathtt{k} \land \mathtt{r} = \mathtt{n}^{\mathtt{i}}\}}}{\{\mathtt{i} < \mathtt{k} \land \mathtt{i} \geq 0 \land \mathtt{i} \leq \mathtt{k} \land \mathtt{r} = \mathtt{n}^{\mathtt{i}}\}\ \mathtt{i} := \mathtt{i}+1; \mathtt{r} := \mathtt{r}*\mathtt{n}\ \{\mathtt{i} \geq 0 \land \mathtt{i} \leq \mathtt{k} \land \mathtt{r} = \mathtt{n}^{\mathtt{i}}\}}\ \text{conseq.}$$

The proof that formula (1) is also a loop invariant is similar and omitted here.

b)

We claim that for the programme s defined as

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

it holds that $\vdash \{\,\mathtt{k}\geq 1\,\}\; s\; \{\,\mathtt{r}=\mathtt{n}^{\mathtt{k}}\,\}$. This is shown by the following derivation tree:

$$\dfrac{\dfrac{\vdots}{\{\mathtt{k}\geq1\}\ \mathtt{i}:=0\ \{\mathtt{k}\geq1\wedge\mathtt{i}=0\}} \qquad \dfrac{\dfrac{\vdots}{\{\mathtt{k}\geq1\wedge\mathtt{i}=0\}\ \mathtt{r}:=1\ \{\mathtt{k}\geq1\wedge\mathtt{i}=0\wedge\mathtt{r}=1\}} \quad \dfrac{\dfrac{T}{\{\mathtt{i}\geq0\wedge\mathtt{i}\leq\mathtt{k}\wedge\mathtt{r}=\mathtt{n}^{\mathtt{i}}\}\ \mathtt{while}\ldots\mathtt{end}\ \{\mathtt{i}\geq\mathtt{k}\wedge\mathtt{i}\geq0\wedge\mathtt{i}\leq\mathtt{k}\wedge\mathtt{r}=\mathtt{n}^{\mathtt{i}}\}}}{\{\mathtt{k}\geq1\wedge\mathtt{i}=0\wedge\mathtt{r}=1\}\ \mathtt{while}\ldots\mathtt{end}\ \{\mathtt{r}=\mathtt{n}^{\mathtt{k}}\}}\ \text{conseq.}}{\{\mathtt{k}\geq1\wedge\mathtt{i}=0\}\ \mathtt{r}:=1;\mathtt{while}\ldots\mathtt{end}\ \{\mathtt{r}=\mathtt{n}^{\mathtt{k}}\}}}{\{\mathtt{k}\geq1\}\ s\ \{\mathtt{r}=\mathtt{n}^{\mathtt{k}}\}}$$

where T is the derivation tree for showing that formula (2) is a loop invariant. We have omitted obvious steps in the derivation tree. We can also use a different notation by annotating the programme with pre- and postconditions:

```
 $\{\,k \geq 1\,\}$
$\Rightarrow$
 $\{\,k \geq 1 \wedge 0 = 0\,\}$
      $i := 0;$
 $\{\,k \geq 1 \wedge i = 0\,\}$
$\Rightarrow$
 $\{\,k \geq 1 \wedge i = 0 \wedge 1 = 1\,\}$
      $r := 1;$
 $\{\,k \geq 1 \wedge i = 0 \wedge r = 1\,\}$
$\Rightarrow$
 $\{\,i \geq 0 \wedge i \leq k \wedge r = n^{i}\,\}$
      while $i < k$ do
          $\{\,i < k \wedge i \geq 0 \wedge i \leq k \wedge r = n^{i}\,\}$
              $i := i + 1;$
          $\{\,i - 1 < k \wedge i - 1 \geq 0 \wedge i - 1 \leq k \wedge r = n^{i-1}\,\}$
        $\Leftrightarrow$
          $\{\,i - 1 < k \wedge i - 1 \geq 0 \wedge i - 1 \leq k \wedge r * n = n^{i}\,\}$
              $r := r * n$
          $\{\,i - 1 < k \wedge i - 1 \geq 0 \wedge i - 1 \leq k \wedge r = n^{i}\,\}$
        $\Rightarrow$
          $\{\,i \geq 0 \wedge i \leq k \wedge r = n^{i}\,\}$
      end
```

$\{\, \mathtt{i} \geq \mathtt{k} \land \mathtt{i} \geq 0 \land \mathtt{i} \leq \mathtt{k} \land \mathtt{r} = \mathtt{n}^{\mathtt{i}} \,\}$
$\Rightarrow$
$\{\, \mathtt{r} = \mathtt{n}^{\mathtt{k}} \,\}$

Assignment 2

The intuition of the **sound rule of consequence** is the following: if we execute a statement s in a state satisfying the constraints $\mathbf{P}$ (the precondition, e.g. $x \geq 0$) and if the final state satisfies the constraints $\mathbf{Q}$ (the postcondition, e.g. $x \leq 5$), we then can conclude that s will also execute successfully in a state satisfying the *stronger* constraints $\mathbf{P}'$ (e.g. $x \geq 5$) and that the final state at least satisfies the *weaker* constraints $\mathbf{Q}'$ (e.g. $x \leq 0$).

Assume we have proved the triple $\{\ x \geq 0\ \}\ s\ \{\ x \leq 5\ \}$ for an algorithm that we implemented as s and that now is to be used by our co-worker Alice.
She does not need to know the actual implementation, but we provide her with the pre- and postcondition (the *contract*) so that she knows when (i.e. in which states) she can successfully use our algorithm and which final states her own programme needs to be able to handle afterwards.

We could provide her with the conditions $\mathbf{P}$ and $\mathbf{Q}$, but we decide to give her $\mathbf{P}'$ and $\mathbf{Q}'$ instead. This is valid, because our algorithm s will execute successfully if invoked in a state where $x \geq 5$, since we proved that it does so in all states where $x \geq 0$.
Due to the postcondition $\mathbf{Q}'$ that we gave her, Alice implemented her programme in a way such that it successfully operates on all states where $x \leq 0$. This is perfectly fine since s only yields final states where $x \leq 5$.

Now consider the **unsound rule**. This time, let $\mathbf{P}$, $\mathbf{Q}$, $\mathbf{P}'$ and $\mathbf{Q}'$ be $x \geq 5$, $x \leq 0$, $x \geq 0$ and $x \leq 5$, respectively.

If Alice invokes s in a state where $x \geq 0$, an error might occur since our algorithm only guarantees successful termination in all states where $x \geq 5$.
Analogous, if Alice expects that the states resulting from the invocation of s satisfy $x \leq 5$, her own computations might fail since s can actually yield states where $x \leq 0$.

Let's consider a **concrete counterexample** where $\{\ \mathbf{P}'\ \}\ s\ \{\ \mathbf{Q}'\ \}$ is a valid triple, where $\mathbf{P}' \Rightarrow \mathbf{P}$ and $\mathbf{Q} \Rightarrow \mathbf{Q}'$, but where $\{\ \mathbf{P}\ \}\ s\ \{\ \mathbf{Q}\ \}$ is not a valid triple:

$$\frac{\{\ \mathrm{x} > 1\ \}\ \mathrm{x} := \mathrm{x} + 1\ \{\ x > 2\ \}}{\{\ \mathrm{x} \geq 1\ \}\ \mathrm{x} := \mathrm{x} + 1\ \{\ x > 3\ \}}$$

If we begin a state where $x = 1$ then the pre-condition of this triple holds, but after execution of the statement, the post-condition of the triple will be false. Therefore, this rule allows us to deduce unsound conclusions.

Assignment 3

Let's first consider the "suitable precondition $\mathbf{P}$", which we intend to be the weakest precondition guaranteeing that the postcondition holds [1]:

1. We claimed that the programme computes "the quotient and the remainder of $\frac{X}{Y}$", which isn't defined for $Y = 0$. But since we only want to verify the postcondition $\mathtt{X} = \mathtt{x} + \mathtt{Y} * \mathtt{z} \wedge \mathtt{Y} > \mathtt{x}$ (which does not mention $\frac{X}{Y}$) we do not need to require that $Y \neq 0$.
2. We observe that s does not terminate if $Y < 0$. Since we consider partial correctness only, there is also no need to require "$Y \geq 0$".
3. It is not obvious how one should define quotient and remainder on $\mathbb{Z}$, but this, once again, does not affect the verification of our postcondition.

Thus, $\mathbf{P}$ can be $true$ and we therefore simply omit it.

Using $\{\ X = \mathtt{z}Y + \mathtt{x} \ \wedge \ \mathtt{y} = Y\ \}$ as the loop invariant we now prove that
$\vdash \{\ \mathtt{x} = X \wedge \mathtt{y} = Y\ \}\ s\ \{\ \mathtt{X} = \mathtt{x} + \mathtt{Y} * \mathtt{z} \wedge \mathtt{Y} > \mathtt{x}\ \}$:

```
{ $\mathtt{x} = X \ \wedge \ \mathtt{y} = Y$ }
    $\Rightarrow$
{ $X = 0 \cdot Y + \mathtt{x} \wedge \mathtt{y} = Y$ }
    z := 0;
{ $X = \mathtt{z}Y + \mathtt{x} \ \wedge \ \mathtt{y} = Y$ }
    while y <= x do
        { $X = \mathtt{z}Y + \mathtt{x} \ \wedge \ \mathtt{y} = Y \ \wedge \ \mathtt{y} \leq \mathtt{x}$ }
            $\Rightarrow$
        { $X = (\mathtt{z} + 1)Y + \mathtt{x} - \mathtt{y} \ \wedge \ \mathtt{y} = Y$ }
            z := z + 1;
        { $X = \mathtt{z}Y + \mathtt{x} - \mathtt{y} \ \wedge \ \mathtt{y} = Y$ }
            x := x - y
        { $X = \mathtt{z}Y + \mathtt{x} \ \wedge \ \mathtt{y} = Y$ }
    end
{ $X = \mathtt{z}Y + \mathtt{x} \ \wedge \ \mathtt{y} = Y \ \wedge \ \mathtt{y} > \mathtt{x}$ }
    $\Rightarrow$
{ $X = \mathtt{z}Y + \mathtt{x} \ \wedge \ Y > \mathtt{x}$ }
```

---

[1] If we had a free choice, we could even set $\mathbf{P}$ to $false$, which would make the proof trivial, but this is not "suitable" - it results in an essentially useless triple.

Assignment 4

We prove the claim by an induction over the structure of the statement s.

Base cases

- $s = \texttt{skip}$:

  The following derivation tree shows that for any property P, we have that $\vdash \{\, P \,\}\ \texttt{skip}\ \{\, 0 = 0 \,\}$:

  $$\frac{\{\, P \,\}\ \texttt{skip}\ \{\, P \,\}}{\{\, P \,\}\ \texttt{skip}\ \{\, 0 = 0 \,\}}\ \text{weakening of the postcondition}$$

- $s = \texttt{x} := e$:

  The following derivation tree shows that for any property P, we have that $\vdash \{\, P \,\}\ \texttt{x} := e\ \{\, 0 = 0 \,\}$:

  $$\frac{\{\, 0 = 0 \,\}\ \texttt{x} := e\ \{\, 0 = 0 \,\}}{\{\, P \,\}\ \texttt{x} := e\ \{\, 0 = 0 \,\}}\ \text{strengthening of the precondition}$$

  Note that $0 = 0[\texttt{x} \mapsto e]$ is $0 = 0$ and thus, the axiom for assignment applies.

Step cases

- $s = r; t$:

  Let P be an arbitrary property. From the induction hypothesis, we have that for all properties Q and R, we have that $\vdash \{\, Q \,\}\ r\ \{\, 0 = 0 \,\}$ and $\vdash \{\, R \,\}\ t\ \{\, 0 = 0 \,\}$. Let T_1 be a derivation tree that shows $\vdash \{\, Q \,\}\ r\ \{\, 0 = 0 \,\}$ and let T_2 be a derivation tree that shows $\vdash \{\, 0 = 0 \,\}\ t\ \{\, 0 = 0 \,\}$. With the rule for sequential composition, we construct the following derivation tree that shows $\vdash \{\, P \,\}\ r; t\ \{\, 0 = 0 \,\}$:

  $$\frac{T_1 \quad T_2}{\{\, P \,\}\ r; t\ \{\, 0 = 0 \,\}}$$

- $s = \texttt{if}\ b\ \texttt{then}\ r\ \texttt{else}\ t\ \texttt{end}$:

  Let P be an arbitrary property. By induction hypothesis, we have derivation trees T_1 and T_2 for showing $\vdash \{\, P \land \mathcal{B}[\![b]\!] \,\}\ r\ \{\, 0 = 0 \,\}$ and $\vdash \{\, P \land \neg\mathcal{B}[\![b]\!] \,\}\ t\ \{\, 0 = 0 \,\}$, respectively. With the rule for conditionals we construct the following derivation tree that shows $\vdash \{\, P \,\}\ \texttt{if}\ b\ \texttt{then}\ r\ \texttt{else}\ t\ \texttt{end}\ \{\, 0 = 0 \,\}$:

  $$\frac{T_1 \quad T_2}{\{\, P \,\}\ \texttt{if}\ b\ \texttt{then}\ r\ \texttt{else}\ t\ \texttt{end}\ \{\, 0 = 0 \,\}}$$

- $s =$ `while` b `do` s `end`:

  Let P be an arbitrary property. By induction hypothesis, we have a derivation tree T that shows $\vdash \{\, \mathcal{B}[\![b]\!] \wedge 0 = 0 \,\}\ s\ \{\, 0 = 0 \,\}$. We construct the following derivation tree by strengthening the precondition and weakening the postcondition:

$$\frac{\dfrac{T}{\{\, 0 = 0 \,\}\ \texttt{while}\ b\ \texttt{do}\ s\ \texttt{end}\ \{\, \neg\mathcal{B}[\![b]\!] \wedge 0 = 0 \,\}}}{\{\, P \,\}\ \texttt{while}\ b\ \texttt{do}\ s\ \texttt{end}\ \{\, 0 = 0 \,\}}$$

Assignment 5 - Headache of the week

1) $\{\ x = X_0 \wedge y = Y_0 \wedge X_0 > 0 \wedge Y_0 > 0\ \}\ s\ \{\ z = gcd(X_0, Y_0)\ \}$

2) A suitable loop invariant is: $\mathit{gcd}(x, y) = \mathit{gcd}(b, c) \wedge b > 0 \wedge c > 0 \wedge x = X_0 \wedge y = Y_0$ (preservation shown below)

3) Here is the proof outline:

```
{$x = X_0 \wedge y = Y_0 \wedge X_0 > 0 \wedge Y_0 > 0$}
b := x;
{$x = X_0 \wedge y = Y_0 \wedge X_0 > 0 \wedge Y_0 > 0 \wedge b = X_0$}
c := y;
{$x = X_0 \wedge y = Y_0 \wedge X_0 > 0 \wedge Y_0 > 0 \wedge b = X_0 \wedge c = Y_0$}
$\Rightarrow$
{$gcd(x, y) = gcd(b, c) \wedge b > 0 \wedge c > 0 \wedge x = X_0 \wedge y = Y_0$}
while b#c do
  {$gcd(x, y) = gcd(b, c) \wedge b > 0 \wedge c > 0 \wedge x = X_0 \wedge y = Y_0 \wedge b \neq c$}$^\star$
  if b < c then
    {$gcd(x, y) = gcd(b, c) \wedge b > 0 \wedge c > 0 \wedge x = X_0 \wedge y = Y_0 \wedge b \neq c \wedge b < c$}
    $\Rightarrow$
    {$gcd(x, y) = gcd(b, (c - b + b)) \wedge b > 0 \wedge (c - b) > 0 \wedge x = X_0 \wedge y = Y_0 \wedge b < (c - b + b)$}
    c := c - b;
    {$gcd(x, y) = gcd(b, (c + b)) \wedge b > 0 \wedge c > 0 \wedge x = X_0 \wedge y = Y_0 \wedge b < (c + b)$}
    $\Rightarrow$
    {$gcd(x, y) = gcd(b, c) \wedge b > 0 \wedge c > 0 \wedge x = X_0 \wedge y = Y_0$}
  else
    {$gcd(x, y) = gcd(b, c) \wedge b > 0 \wedge c > 0 \wedge x = X_0 \wedge y = Y_0 \wedge b \neq c \wedge b \geq c$}
    $\Rightarrow$
    {$gcd(x, y) = gcd((b - c + c), c) \wedge (b - c) > 0 \wedge c > 0 \wedge x = X_0 \wedge y = Y_0 \wedge (b - c + c) > c$}
    b := b - c;
    {$gcd(x, y) = gcd((b + c), c) \wedge b > 0 \wedge c > 0 \wedge x = X_0 \wedge y = Y_0 \wedge (b + c) > c$}
    $\Rightarrow$
    {$gcd(x, y) = gcd(b, c) \wedge b > 0 \wedge c > 0 \wedge x = X_0 \wedge y = Y_0$}
  end
  {$gcd(x, y) = gcd(b, c) \wedge b > 0 \wedge c > 0 \wedge x = X_0 \wedge y = Y_0$}
end;
{$gcd(x, y) = gcd(b, c) \wedge b > 0 \wedge c > 0 \wedge x = X_0 \wedge y = Y_0 \wedge b = c$}
z := b
{$gcd(x, y) = gcd(b, c) \wedge b > 0 \wedge c > 0 \wedge x = X_0 \wedge y = Y_0 \wedge b = c \wedge z = b$}
$\Rightarrow$
{$z = gcd(X_0, Y_0)$}
```