Course: *Reasoning about Programs I*
Examples

Observation: `return` statement used below is to emphasize which is the value returned by the algorithm also appearing in the postcondition.

Example 1

Consider the following algorithm computing the maximum between two integer numbers.

```
int max (int a, int b)
    if a >= b then
        max := a
    else
        max := b
    return max
```

Write for the algorithm suitable specification and prove the partial correctness of the algorithm.
Solution. $P :\Longleftrightarrow \; a \in \mathbb{Z} \land b \in \mathbb{Z}$ and the postcondition is $Q :\Longleftrightarrow \; ((a \geq b \Rightarrow max = a) \land (a < b \Rightarrow max = b))$.
We have the following:

$$
\dfrac{
\dfrac{\dfrac{\dots}{P \land a \geq b \Rightarrow wp(max := a, Q)}}{\{P \land a \geq b\} \;\; max := a \; \{Q\}}
\qquad
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\checkmark}{P \land \neg(a \geq b) \Rightarrow \neg(a \geq b)}}{P \land \neg(a \geq b) \Rightarrow (\neg(a \geq b)) \land True}}{P \land \neg(a \geq b) \Rightarrow ((a \geq b \Rightarrow False) \land (a < b \Rightarrow True))}}{P \land \neg(a \geq b) \Rightarrow ((a \geq b \Rightarrow b = a) \land (a < b \Rightarrow b = b))}}{P \land \neg(a \geq b) \Rightarrow Q[max \to b]}}{P \land \neg(a \geq b) \Rightarrow wp(max := b, Q)}}{\{P \land \neg(a \geq b)\} \;\; max := b \; \{Q\}}
}{
\{P\} \;\; \texttt{if} \;\; a \geq b \;\; \texttt{then} \;\; max := a \;\; \texttt{else} \;\; max := b \;\; \{Q\}
}
$$

Example 2

Consider the following algorithm computing the natural power a^p of a non-zero real number $a \in \mathbb{R}^*$, $p \in \mathbb{N}$.

```
int power (int a, int p)
    rez := 1;
    i   := 0;
    while i < p do
        i := i + 1;
        rez := rez * a
    return rez
```

Write for the algorithm a suitable specification, derive a loop invariant and prove the partial correctness of the algorithm.

Solution. The precondition is $P :\iff a \in \mathbb{R}^* \ \wedge \ p \in \mathbb{N}$ and the postcondition is $Q :\iff rez = \prod_{i=1}^{p} a$. We have:

$$\frac{\ldots}{\{P\} \ \ rez := 1;\ i := 0;\ \texttt{while}\ i < p\ \texttt{do}\ i := i + 1;\ rez := rez * a \ \ \{Q\}}$$

This example is typical for the case when it is convenient to combine forward reasoning (sp) with backward reasoning (wp). We apply sp intuitively without giving the corresponding rules for sequence (;) and assignment (:=) commands. Applying forward reasoning, we have:

$$\frac{\dfrac{\ldots}{\{P \wedge rez = 1 \wedge i = 0\}\ \texttt{while}\ i < p\ \texttt{do}\ i := i + 1;\ rez := rez * a\ \{Q\}}}{\{P\} \ \ rez := 1;\ i := 0;\ \texttt{while}\ i < p\ \texttt{do}\ i := i + 1;\ rez := rez * a \ \ \{Q\}}$$

We synthesize a suitable invariant I for the loop above. We have the following: We conjecture

#iter	i	rez
0	0	1
1	1	$1 * a = a$
2	2	$a * a = a^2$
...	...	...
k	k	$a^{k-1} * a = a^k$
$k+1$	$k+1$	$a^k * a = a^{k+1}$
...	...	...
p	p	$a^{p-1} * a = a^p$

that the loop invariant is $rez = \prod_{j=1}^{i} a$. We have the following:

We have:

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\checkmark}{(P\wedge rez{=}1\wedge i{=}0)\Rightarrow 1{=}1}}{(P\wedge rez{=}1\wedge i{=}0)\Rightarrow 1{=}\prod\limits_{j=1}^{0}a}}{(P\wedge rez{=}1\wedge i{=}0)\Rightarrow rez{=}\prod\limits_{j=1}^{i}a}
\quad
\dfrac{\dots}{\{rez{=}\prod\limits_{j=1}^{i}a\wedge i{<}p\}i:=i{+}1;\, rez{:=}rez{*}a\{rez{=}\prod\limits_{j=1}^{i}a\}}
\quad
\dfrac{\dfrac{\dfrac{\checkmark}{\left(rez=\prod\limits_{j=1}^{p}a\wedge\neg(i{=}p)\right)\Rightarrow Q}}{\left(rez=\prod\limits_{j=1}^{i}a\wedge\neg(i{=}p)\right)\Rightarrow Q}}{\left(rez=\prod\limits_{j=1}^{i}a\wedge\neg(i{<}p)\right)\Rightarrow Q}}
{\{P\wedge rez=1\wedge i=0\}\ \texttt{while}\ i<p\ \texttt{do}\ i:=i+1;\ rez:=rez*a\ \{Q\}}}
{\{P\}\ \ rez:=1;\ i:=0;\ \texttt{while}\ i<p\ \texttt{do}\ i:=i+1;\ rez:=rez*a\ \ \{Q\}}
$$

We take, by notation $\prod\limits_{i=1}^{0} z = 1$.

We continue the middle tree below:

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\checkmark}{rez{=}\prod\limits_{j=1}^{i}a\wedge i{<}p\Rightarrow rez{=}\prod\limits_{j=1}^{i}a}}{rez{=}\prod\limits_{j=1}^{i}a\wedge i{<}p\Rightarrow rez*a{=}\prod\limits_{j=1}^{i}a*a}}{rez{=}\prod\limits_{j=1}^{i}a\wedge i{<}p\Rightarrow rez*a{=}\prod\limits_{j=1}^{i+1}a}}{rez{=}\prod\limits_{j=1}^{i}a\wedge i{<}p\Rightarrow wp(i{:=}i{+}1, rez*a{=}\prod\limits_{j=1}^{i}a)}}{rez{=}\prod\limits_{j=1}^{i}a\wedge i{<}p\Rightarrow wp(i{:=}i{+}1, wp(rez{:=}rez{*}a, rez{=}\prod\limits_{j=1}^{i}a))}}{\{rez{=}\prod\limits_{j=1}^{i}a\wedge i{<}p\}i:=i{+}1;\, rez{:=}rez{*}a\{rez{=}\prod\limits_{j=1}^{i}a\}}
$$

Example 3

Consider the following algorithm finding the smallest index r of an occurrence of value x in array a ($r = -1$, if x does not occur in a).

```
i := 0; r := -1; n = len(a);
while i < n && r = -1 do
    if a[i] = x
        then r := i
    else i := i + 1
return r
```

Write for the algorithm a suitable specification, derive a loop invariant and prove the partial correctness of the algorithm.

Solution. The precondition is $P :\iff \top$ and the postcondition is

$$Q :\iff ((r = -1 \wedge \underset{0\le i<len(a)}{\underset{i}{\forall}} a[i] \neq x) \vee (0 \le r < len(a) \wedge a[r] = x \wedge \underset{0\le i<r}{\underset{i}{\forall}} a[i] \neq x))$$

The invariant is

$$I :\iff n = len(a) \land 0 \leq i \leq n \land \underset{0 \leq j < i}{\overset{}{\underset{j}{\forall}}} \; a[j] \neq x \land (r = -1 \lor (r = i \land i < n \land a[r] = x))$$

We have

$$\dfrac{\dfrac{...}{(P \land ...) \Rightarrow I} \qquad \dfrac{...}{\{I \land (i{<}n\&\&r{=}{-1})\}\ \texttt{if}\ ...\ \texttt{then}\ ...\ \texttt{else}\ ...\ \{I\}} \qquad \dfrac{...}{I \land \neg(i{<}n\&\&r{=}{-1})\} \Rightarrow Q}}{\dfrac{\{P \land i{=}0 \land r{=}{-1} \land n{=}len(a)\}\ \texttt{while}\ (i{<}n\&\&r{=}{-1})\texttt{do if}\ a[i]{=}x\ \texttt{then}\ r{:=}i\ \texttt{else}\ i{:=}i+1\ \{Q\}}{\{P\}\ i{:=}0; r{:=}{-1}; n{=}len(a);\ \texttt{while}\ (i{<}n\&\&r{=}{-1})\texttt{do if}\ a[i]{=}x\ \texttt{then}\ r{:=}i\ \texttt{else}\ i{:=}i+1\ \{Q\}}}$$

The inner branch of the tree must be split again, while the other 2 branches are already verification conditions (FOL formulae) which must be proved.

Example 4

Write an algorithm computing the sum of the first n natural numbers and prove its partial correctness.

Hint. A suitable invariant for the loop invariant is:

$$I :\iff s = \sum_{j=1}^{i-1} j \land 1 \leq i \leq n+1$$

.

Example 5

Let $P(X) = a_n X^n + a_{n-1} X^{n-1} + \ldots + a_1 X + a_0$ be a polynomial with real coefficients. Write an algorithm computing the value of the polynomial in the point X_0, that is $P(X_0)$. Prove the partial correctness of the algorithm.

Hint. There are several ways of representing a polynomial, each one being more suitable for different types of problems. For our problem, a suitable representation is the *list of coefficients*. For example, for the polynomial above, we will use the array $a[0..n]$ where $a[i]$ represents the value of the coefficient a_i. The length of a is given by the degree of the polynomial. For example, the polynomial $X^4 - 2X^2 + 5$ is represented by the array $(5, 0, -2, 0, 1)$ and the polynomial $X^{10} - 2$ by $(-2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1)$.

The simplest and most efficient method for polynomial evaluation in a point when the polynomial is represented by the list of coefficients is inspired by Horner scheme. We have:

$$\begin{aligned} P(X) &= a_n X^n + a_{n-1} X^{n-1} + \ldots + a_1 X + a_0 \\ &= \left(a_n X^{n-1} + a_{n-1} X^{n-2} + \ldots + a_1\right) X + a_0 \\ ... \\ &= (\ldots((a_n X + a_{n-1}) X + a_{n-2}) X \ldots + a_1) X + a_0 \end{aligned}$$

The rewriting above suggests that for evaluating the polynomial in a point X_0 it suffices to initialize the variable which contains the result with a_n and for every $i = \overline{n-1...0}$ to multiply the current value with x and add a_i.
An algorithm implementing the idea above is:

```
v := a[n]; i := n;
while i>0 do
     i := i-1;
     v := v*x + a[i]
return v
```

The precondition is : $\top$ and the postcondition is $v = \sum_{i=0}^{n} a_i x^i$. The loop invariant is $v = \sum_{j=i}^{n} a_j x^{j-i}$.

Example 6

What is the output of the following algorithm. Prove its partial correctness.

```
i := 0; s := 0;
while i < n do
     i := i + 1;
     s := s + x[i]*y[i]
return s
```

Example 7

What is the output of the following algorithm. Prove its partial correctness.

```
i := n; s := 0;
while i > 0 do
     s := s + i
     i := i - 1
return s
```