ALGORITHMS. Seminar 2: Operations with sets and polynomials. Correctness verification. Loop invariants.

Exercise 1 (S+L) Let $A = \{a_1, \ldots, a_n\}$ and $B = \{b_1, \ldots, b_m\}$ be two sets of integers. Write algorithms to:

(a) check if an element belongs to a set or not.

(b) compute the union of two sets ($R = A \cup B$ is the set of elements belonging to at least one of the sets).

(c) compute the intersection of two sets ($C = A \cap B$ is the set of common elements of A and B).

Solution. The sets can be represented either by the arrays of their distinct elements or by using an array with presence flags (in this case the array will have as many elements as there are in the universal set containing all sets used as input instances). In the first case the set A will be represented as an array $a[1..n]$ and the set B as an array $b[1..m]$. The elements of the array are in correspondence with the set elements (e.g. $a[i] = a_i$, $i = \overline{1,n}$).

In the second case, the set will be represented as an array of k elements (k is the number of elements of the universal set S). Supposing that $S = \{s_1, \ldots, s_k\}$ the element on position i in $a[1..n]$ is 1 if s_i belongs to the set and 0 otherwise.

(a) If the set is represented by using the array of its elements then checking if an element belong to the set is equivalent with the problem of searching a value in an array.

```
contains(integer a[1..n], e)
integer i
boolean found
found ← False
i ← 1
while i ≤ n AND found = False do
    if a[i] = e then found = True
        else i ← i + 1
    endif
endwhile
return found
```

If the set is represented as an array of presence flags then checking if an element e is in the set or not means comparing $a[e]$ with 1.

(b) In the first representation variant the array corresponding to the union is initialized with one of the sets. Then the array is extended by adding the elements of the second set which does not belong to the first one.

```
union (integer a[1..n], b[1..m])
integeri, r[1..k], k for i = 1, n do
    r[i] ← a[i]
endfor
k ← n
for i = 1, m do
    if contains(a[1..n], b[i]) = False then
        k ← k + 1
        r[k] ← b[i]
    endif
endfor
return r[1..k]
```

When the sets are represented using presence flags the array corresponding to the union should be initialized with 0 then all positions $i \in \{1, \ldots, k\}$ for which $a[i] = 1$ or $b[i] = 1$ are set on 1.

(c) In the first representation variant the array is initialized such that it corresponds to the empty set (the number of elements is 0). Then for all elements of the first set one have to check if they belong to the second set also. All elements which belong to both sets are added to the intersection.

```
intersection (integer a[1..n], b[1..m])
integeri, c[1..k], k k ← 0
for i = 1, m do
    if contains(a[1..n], b[i]) = True then
        k ← k + 1
        r[k] ← b[i]
    endif
endfor
return r[1..k]
```

When the sets are represented by using presence flags the array, $r[1..k]$ corresponding to the intersection will contain the value 1 for all positions i for which $a[i] = 1$ and $b[i] = 1$.

Exercise 2 (S+L) Let $A = a_n X^n + a_{n-1} X^{n-1} + \ldots + a_1 X + a_0$ and $B = b_m X^m + b_{m-1} X^{m-1} + \ldots + b_1 X + b_0$ be two polynomials with real coefficients. Write algorithms for:

(a) evaluating the polynomial A for a given argument x.

(b) computing the sum of the polynomials ($S = A + B$).

(c) computing the product of the polynomials ($P = A * B$).

Solution. There exist several ways of representing the polynomials each one having some particularities. Two of the most used ones are:

(i) *Sequence of all coefficients.* In order to represent a polynomial $A = a_n X^n + a_{n-1} X^{n-1} + \ldots + a_1 X + a_0$ we can use an array $a[0..n]$ where $a[i]$ corresponds to the coefficient a_i. The size of the array depends on the polynomial degree not on the number of non-zero terms. For instance, the polynomial $X^4 - 2X^2 + 5$ is represented by the sequence of coefficients $(5, 0, -2, 0, 1)$ while the polynomial $X^{10} - 2$ is represented as $(-2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1)$.

(ii) *Sequence of non-zero terms.* Each element of the sequence contains information corresponding to a non-zero term in the polynomial: the degree of the term and its coefficient. Thus each element of the array is a pair of values (degree, coefficient) or one can use two distinct arrays: one to store the terms degrees and one the corresponding coefficients. The terms can be stored in an arbitrary order. For instance the polynomial $X^4 - 2X^2 + 5$ is represented as $((1, 4), (-2, 1), (5, 0))$ and the polynomial $X^{10} - 2$ as $((1, 10), (-2, 0))$. This representation is advantageous in the case of polynomials with high degree but a small number of non-zero terms.

Let us consider first the case when the polynomial is represented by using the sequence of all coefficients.

(a) The simplest and most efficient method to evaluate a polynomial given by the sequence of its coefficients is that based on the Horner scheme. The main idea of this method is to write the polynomial as:

$$A = a_n X^n + a_{n-1} X^{n-1} + \ldots + a_1 X + a_0 = (\ldots((a_n X + a_{n-1})X + a_{n-2})X \ldots + a_1)X + a_0$$

This form of the polynomial suggests that to evaluate the polynomial for $X = x$ it is enough if the variable which will contain the polynomial value is initialized with a_n and for each $i = \overline{n-1, 0, -1}$ one multiplies it by x and add a_i.

```
polynomial_evaluation(real a[0..n], x)
integer i
real v
v ← a[n]
for i ← n − 1, 0, −1 do
    v ← v ∗ x + a[i]
endfor
return v
```

(b) The polynomial obtained by adding the polynomials A and B will have the degree $p = \max\{m, n\}$ and its coefficients will be:

$$s_i = \begin{cases} a_i + b_i & 0 \leq i \leq \min\{m, n\} \\ b_i & n+1 \leq i \leq m(\text{ if } m > n) \\ a_i & m+1 \leq i \leq n(\text{ if } n > m) \end{cases}$$

The algorithm can be described as (where maximum and minimum denotes simple algorithms which return the maximum and the minimum of two values, respectively):

```
polynomial_sum(integer a[0..n],b[0..m])
integer i,max,min
real s[0..max]
max ←maximum(n, m)
min ←minimum(n, m)
for i ← 0, min do
    s[i] ← a[i] + b[i]
endfor
if n > m then
    for i ← min + 1, n do
        s[i] ← a[i]
    endfor
endif
if m > n then
    for i ← min + 1, m do
        s[i] ← b[i]
    endfor
endif
return s[0..max]
```

c) The product of the polynomials will have the degree $q = m + n$ and the corresponding coefficients will be:

$$p_k = \sum_{i=\overline{0,n}, j=\overline{0,m}, i+j=k} a_i b_j$$

To compute the coefficients of the product it is enough if all elements of the $p[0..q]$ are initialized with 0 and for all $i = \overline{0, n}$ and for all $j = \overline{0, m}$) the product $a[i] * b[j]$ is added to $p[i + j]$. The algorithm can be described as:

```
polynomial_product(integer a[0..n],b[0..m])
integer i,j,q
real p[0..q]
q ← n + m
for i ← 0, q do
    p[i] ← 0
endfor
for i ← 0, n do
    for j ← 0, m do
        p[i + j] ← p[i + j] + a[i] ∗ b[j]
    endfor
endfor
return p[0..q]
```

Exercise 3 (S+L) Write an algorithm to compute the factorial of a natural number and prove its correctness.

Solution. For a natural number n, the value of the factorial $n! = 1 \cdot 2 \cdots n$ can be computed by following one of the following algoprithms (`factorial1` or `factorial2`).

```
factorial1(integer n)
1: f ← 1
2: i ← 1
3: while i < n do
4:     i ← i + 1
5:     f ← f ∗ i
6: end while
7: return f
```

```
factorial2(integer n)
1: f ← 1
2: i ← 1
3: while i ≤ n do
4:     f ← f ∗ i
5:     i ← i + 1
6: end while
7: return f
```

The problem precondition is $P = \{n \in \mathbb{N}\}$, while the postcondition is $Q = \{f = \prod_{i=1}^{n} i\}$.

After the execution of the first two instructions the algorithm state is $\{f = 1, i = 1\}$ which means $\{f = \prod_{j=1}^{i} j\}$. Let us analyze the algorithm `factorial1` and let us prove that $f = \prod_{j=1}^{i} j$ is invariant with respect to the **while** loop. This property is obviously satisfied before the loop. Let us analyze what happens during the execution of the instructions in the loop.

After the execution of instruction 4 (modification of i) the property can be written as $f = \prod_{j=1}^{i-1} j$ (since f contains the product of values less than the value of i before its modification). By executing the line 5, f is multiplied with the current value of i, thus it becomes again $f = \prod_{j=1}^{i} j$. Thus, this property is invariant with respect to the loop. We only have to prove that at the end of the loop this property implies the postcondition. At the loop and the value of i is n, thus the property becomes $f = \prod_{j=1}^{n} j$, which is just the postcondition. Thus the algorithm is partially correct.

Let us prove now its finiteness. It is enough to find a termination function $t : \mathbb{N} \to \mathbb{N}$ (with respect to the implicit loop counter, p) which is strictly decreasing and which is 0 when the continuation condition is false. Such a function is $t(p) = n - i_p$ where i_p denotes the value of variable i corresponding to the pth execution of the loop.

In the case of the algorithm `factorial2` the correctness can be proved in a similar way by using as loop invariant $f = \prod_{j=1}^{i-1} j$ and as termination function $t(p) = (n + 1) - i_p$.

Exercise 4 (S) Prove the correctness of the algorithms `conversion_10_q` and `conversion_q_10` which correspond to the conversion of a number from base 10 to base q and from base q to base 10, respectively.

```
conversion_10_q(integer n, q)
integer c[0..k],k,m
 1: m ← n
 2: k ← 0
 3: c[k] ← m MOD q
 4: m ← m DIV q
 5: while m > 0 do
 6:     k ← k + 1
 7:     c[k] ← mMODq
 8:     m ← mDIVq
 9: end while
10: return c[0..k]
```

```
conversion_q_10(integer c[0..k], q)
integer i, n
1: i ← k
2: n ← c[i]
3: while i > 0 do
4:     i ← i − 1
5:     n ← n ∗ q + c[i]
6: end while
7: return n
```

Solution. In the case of the first algorithm k denotes the number of digits in the representation of the number in base q and $c[0..k]$ contains the digits of this representation starting with the least significant one. Using these notations the precondition is $P = \{n \in \mathbb{N},\ q \in \mathbb{N},\ q \geq 2\}$, and the postcondition is $n = c[k]q^k + c[k-1]q^{k-1} + \ldots + c[1]q + c[0]$. After the execution of the first four instructions the state of the algorithm is $\{n = m \cdot q + c[k], k = 0\}$ i.e. $\{n = m \cdot q^{k+1} + c[k] \cdot q^k\}$. We can guess that the invariant is: $n = m \cdot q^{k+1} + \sum_{i=0}^{k} c[i] \cdot q^i$. It is easy to see that for $k = 0$ this property is equivalent with the state of the algorithm just before the **while** loop.

After the execution of the instruction on line 6 the property can be written as $n = m \cdot q^k + \sum_{i=0}^{k-1} c[i] \cdot q^i$.

By executing the instructions on lines 7 and 8 the old value of m (m_p) can be described by using the new value of m (m_{p+1}) as follows: $m_p = m_{p+1} \cdot q + c[k]$. Thus the analyzed property becomes again $n = m \cdot q^{k+1} + \sum_{i=0}^{k} c[i] \cdot q^i$. It is enough now to prove that at the end of the **while** loop this property implies the postcondition. Indeed, when the **while** loop is terminated the value of m is 0 thus the property becomes $n = \sum_{i=0}^{k} c[i] \cdot q^i$, which is the postcondition.

Concerning the finiteness of the algorithm, one can easy see that the function $t(p) = m_p$ (m_p is the value of variable m corresponding to the p-th execution of the loop) satisfies the properties of a termination function.

The main idea of the algorithm `conversion_q_10` is that in order to compute the value in base 10 corresponding to a representation in base q, $c[0..k]$, one have to compute the sum $\sum_{i=0}^{k} c[i]q^i$. The precondition is $P = \{k > 0\}$ and the postcondition is $Q = \{n = \sum_{i=0}^{k} c[i]q^i\}$. After the execution of the instructions in the first two lines the state of the algorithm becomes $n = c[k] = c[k] \cdot q^{k-i} = \sum_{j=i}^{k} c[j]q^{j-i}$. Let consider the property $n = \sum_{j=i}^{k} c[j]q^{j-i}$ and let us to prove that it is a loop invariant. It is easy to see that the property is true just before the **while** loop.

After the execution of line 4 the property becomes (with respect to the new value of the variable i): $n = \sum_{j=i+1}^{k} c[j]q^{j-i-1}$. By executing the line 5 the value of the variable n is changed such that the analyzed property becomes $n = \sum_{j=i}^{k} c[j]q^{j-i}$. When the loop is terminated the variable i has the value 0, thus one obtains that $n = \sum_{j=0}^{k} c[j]q^j$, i.e. the postcondition. The finiteness follows immediately from the fact that $t(p) = i_p$ has the properties of a termination function.

Exercise 5 (S) Prove that by the execution of the following algorithm, n will contain the value in base 10 corresponding to the binary sequence $b[0..k]$ ($b[i] \in \{0, 1\}$, $i = \overline{0,k}$).

```
alg(integer b[0..k])
integer n, i
 1: i ← 0
 2: n ← 0
 3: while i ≤ k do
 4:     n ← n * 2 + b[k − i]
 5:     i ← i + 1
 6: end while
 7: return n
```

Hint. One can prove that $n = \sum_{j=0}^{i-1} b[k-j]2^{k-j}$ is a loop invariant and $t(p) = (k+1) - i_p$ is a termination function.

Exercise 6 (S) Prove that the following algorithm reverses the digits of a natural value. Let us consider that $n = c_k c_{k-1} \dots c_1 c_0$ is the initial number and the new value is $m = c_0 c_1 \dots c_k$. Thus the precondition is $n = \sum_{i=0} c_i 10^i$ and the postcondition is $m = \sum_{i=0} c_i 10^{k-i}$. The method to compute m is described in the algorithm `reverse` described in the following.

```
reverse(integer n)
integer m, p
 1: m ← 0
 2: p ← 0
 3: while n > 0 do
 4:     p ← p + 1
 5:     m ← m * 10 + n MOD 10
 6:     n ← n DIV 10
 7: end while
 8: return n
```

Solution. The variable p is not necessary to describe the algorithm. It was introduced just to help the correctness proof. Let us consider the following properties: $\{n = \sum_{i=p}^{k} c[i]10^{i-p}$, $m = \sum_{i=0}^{p-1} c[i]10^{p-1-i}\}$. One can see that the precondition implies the first property (when $p = 0$) while the second one is true for $m = 0$ and $p = 0$ (m is described in this case as an empty sum). By executing the instructions in line 4 the properties become: $\{n = \sum_{i=p-1}^{k} c[i]10^{i-p+1}$, $m = \sum_{i=0}^{p-2} c[i]10^{p-2-i}\}$.
After the execution of line 5, the second property becomes $m = \sum_{i=0}^{p-2} c[i]10^{p-1-i} + c[p-1] = \sum_{i=0}^{p-1} c[i]10^{p-1-i}\}$.
After the execution of line 6 one have that $n = \sum_{i=p}^{k} c[i]10^{i-p}$. When n is 0 it follows that $p = k+1$ thus $m = \sum_{i=0}^{p-1} c[i]10^{p-1-i}$ which implies the postcondition.

The finiteness is ensured by the existence of a termination function ($t(p) = n_p$ or $t(p) = k + 1 - p$).

Exercise 7 Prove that the following algorithm computes the product of two nonzero natural numbers.

```
product(integer a,b)
integer s
 1: x ← a
 2: y ← b
 3: p ← 0
 4: s ← 0
 5: while x > 0 do
 6:    p ← p + 1
 7:    if x MOD 2=1 then
 8:       s ← s + y
 9:    end if
10:    x ← x DIV 2
11:    y ← 2 * y
12: end while
13: return s
```

Solution. It is easy to see that this algorithm corresponds to the method of multiplication "à la russe". Let us suppose that a can be described in base 2 by using $k+1$ binary digits $c_k c_{k-1}...c_1 c_0$ stored in an array, $c[0..k]$.

Using these notations the precondition is $a = c_k 2^k + c_{k-1}2^{k-1} \dots c_1 2 + c_0$ and the postcondition is $s = ab$. The loop invariant consists of the following properties: $\{s = (\sum_{i=0}^{p-1} c_i 2^i)b,\ x = \sum_{i=p}^{k} c_i 2^{i-p},\ y = b2^p\}$. It is easy to check that these properties are satisfied just before the loop. By executing the line 6 the properties become: $\{s = (\sum_{i=0}^{p-2} c_i 2^i)b,\ x = \sum_{i=p-1}^{k} c_i 2^{i-p+1},\ y = b2^{(p-1)}\}$. By executing the line 7 the first property becomes: $s = (\sum_{i=0}^{p-1} c_i 2^i)b$. By executing the line 8, the second property becomes $x = \sum_{i=p}^{k} c_i 2^{i-p}$ and by executing the line 9 the third property becomes $y = b2^p$. Thus these properties are invariant with respect to the loop. When the loop is terminated the value of x is 0 and $p = k+1$, thus the first property implies the postcondition..

The finiteness can be proved by using $t(p) = x_p$ as termination function.

Exercise 8 (S) Prove that the following algorithm corresponds to the addition of two natural numbers represented in base 2 using arrays of $n+1$ binary values.

```
addition(integer a[0..n], b[0..n])
integer c[0..n + 1], s, cd
 1: c[0..n + 1] ← 0
 2: cd ← 0
 3: i ← 0
 4: while i ≤ n do
 5:    s ← a[i] + b[i] + cd
 6:    c[i] ← s MOD 2
 7:    cd ← s DIV 2
 8:    i ← i + 1
 9: end while
10: c[n + 1] ← cd
11: return c[0..n + 1]
```

Hint. Let us denote by $x_{0..i}$ the value corresponding to the binary sequence $x[0..i]$. One can prove that $\{c_{0..i} = a_{0..i-1} + b_{0..i-1}\}$ is a loop invariant, and when the loop is terminated it implies $c_{0..(n+1)} = a_{0..n} + b_{0..n}$ i.e. the array $c[0..n+1]$ contains the sum of all digits in $a[0..n]$ and $b[0..n]$.

Exercise 9 (S) Find a loop invariant for the loop specified in the following algorithm and establish which is the effect of this algorithm.

```
alg(real a[1..n])
integer i
1: i ← 1
2: while i ≤ n − 1 do
3:     if a[i] > a[i + 1] then
4:         a[i] ↔ a[i + 1]
5:     end if
6:     i ← i + 1
7: end while
8: return a[1..n]
```

Solution. After the first execution of the loop body the property $a[1] \leq a[2]$ is satisfied. After the second execution we shall have $a[2] \leq a[3]$ (we should remark that the value of the second element is not necessarily the same as after the first step, therefore one cannot say that $a[1] \leq a[2] \leq a[3]$ but only that $a[3] \geq a[2]$ and $a[3] \geq a[1]$). These remarks suggest that after the execution of the ith step one have $a[i] = \max_{j=\overline{1,i}} a[j]$. Let us prove that this property is a loop invariant. For $i = 1$ the property is implicitly satisfied. After the execution of line 3 the property becomes $a[i + 1] = \max_{j=\overline{1,i+1}} a[j]$. By the execution of the line 4 the property $a[i] = \max_{j=\overline{1,i}} a[j]$. Thus it is a loop invariant. When the loop is terminated $i = n$, thus $a[n] = \max_{j=\overline{1,n}} a[j]$. Thus the effect of this algorithm is that it will put the maximum on the last position of the array.

Exercise 10 (S) Prove that the following algorithm reverse the order of elements in the array specified as parameter.

```
reverse_sequence(a[1..n])
1: i ← 0
2: while i ≤ ⌊(n + 1)/2⌋ do
3:     i ← i + 1
4:     x[i] ↔ x[n + 1 − i]
5: end while
6: return x[1..n]
```

Hint. Let $x_0[1..n]$ denotes the initial content of the array. Thus the precondition can be described as $P = \{x[i] = x_0[i], i = \overline{1,n}\}$ while the postcondition is $Q = \{x[i] = x_0[n+1-i], i = \overline{1,n}\}$. One can prove that the following assertions $\{i \leq n + 1 - i,\ x[j] = x_0[n + 1 - j],\ x[n + 1 - j] = x_0[j],\ j = \overline{1,i}\}$ are invariant with respect to the **while** loop and when the loop is terminated ($i = \lfloor (n + 1)/2 \rfloor$) they imply the postcondition.

Exercise 11 (S) Find invariants for the loops in algorithms `alg1` and `alg2`. What is the effect of each algorithm?

```
alg1(integer a, b)
integer x, y, z
1: x ← a
2: y ← b
3: z ← 0
4: while y > 0 do
5:     z ← z + x
6:     y ← y − 1
7: end while
8: return z
```

```
alg2(integer a, b)
integer x, y, z
1: x ← a
2: y ← b
3: z ← 0
4: while x ≥ y do
5:     x ← x − y
6:     z ← z + 1
7: end while
8: return z,x
```

Hint. Let us denote with p the implicit loop counter (p will have the value 0 before the loop and it is increased with 1 at each execution of the loop body). By using this notation one can see that for `alg1` the

properties $\{y = b - p,\ z = px,\ x = a\}$ are invariant. When the loop is terminated the variable y has the value 0, thus $p = b$ and $z = px = ba$. Thus the algorithm `alg1` returns the product ab. In the case of the algorithm `alg2` the properties $\{x = a - pb,\ y = b,\ z = p\}$ are invariant. When the loop is terminated one have that $a = z \cdot b + x$ and $x < y = b$. This means that z will contain the quotient of the division of a by b, and x the corresponding remainder.

Homework

1. Write algorithms for computing the union, intersection and difference of two sets represented as arrays of presence flags.

2. Write algorithms for computing the sum and the product of two polynomials represented by arrays containing information about the non-zero terms (degree and coefficient).

3. Prove that the following sequence leads to the exchange of values of variables x and y.

```
1: x ← x + y
2: y ← x − y
3: x ← x − y
```

4. Write an algorithm for the computation of a^p where $a \in \mathbb{R}^*$ and $p \in \mathbb{Z}$. Prove the correctness of the algorithm.

   Hint. First compute $a^{|p|} = \prod_{i=1}^{|p|} a$ and then use the sign of p to decide if the result is $a^{|p|}$ or $1/a^{|p|}$. The correctness can be proved in the same manner as in the case of the factorial.

5. Write an algorithm which transforms an algorithm (by interchanging neighboring elements) such that the minimum is placed on the first position. Prove the correctness of the algorithm.

   Hint. The array is scanned starting with the last element and each element is compared with it predecessor (the current element, i, is compared with the element of index $i - 1$). If the current element is smaller than its predecessor then they are interchanged.

6. Prove that the following algorithms compute the greatest common divisor of two natural numbers.

```
gcd1 (integer a, b)
1: while a ≠ 0 and b ≠ 0 do
2:     a ← a MOD b
3:     if a ≠ 0 then
4:         b ← b MOD a
5:     end if
6: end while
7: if a ≠ 0 then
8:     d ← a
9: else
10:    d ← b
11: end if
12: return d
```

```
gcd2 (integer a, b)
1: while a ≠ b do
2:     if a > b then
3:         a ← a − b
4:     else
5:         b ← b − a
6:     end if
7: end while
8: d ← a
9: return d
```

   Hint. If we denote with a_0 and b_0 the initial values of the variables a and b then the precondition is $P = \{a = a_0, b = b_0\}$ and the postcondition is $Q = \{d = \gcd(a_0, b_0)\}$. In both cases the invariant property is $\gcd(a, b) = \gcd(a_0, b_0)$.

   Proving that this property is invariant is equivalent with proving that $\gcd(a, b) = \gcd(a \text{ MOD } b, b) = \gcd(a, b \text{ MOD } a)$ and $\gcd(a, b) = \gcd(a - b, b)$ (if $a > b$) and $\gcd(a, b) = \gcd(a, b - a)$ (if $a < b$). The finiteness can be proved by using as termination functions: $t(p) = \min(a_p, b_p)$ and $t(p) = |a_p - b_p|$, respectively.