Developing correct algorithms from specification

Example 1

Integer division (quotient and remainder)

Specification

φ: $(x\geq 0) \wedge (y>0)$

ψ: $(x=q*y+r) \wedge (0\leq r<y)$

A_0:	Subalgorithm IntegerDivision(x,y,q,r) is: [φ, ψ] endIntegerDivision

Let η: $(x=q*y+r) \wedge (0\leq r)$ a middle predicate and we can use Sequential rule:

A_1:	Subalgorithm IntegerDivision (x,y,q,r) is: [φ, η] [η, ψ] endIntegerDivision

The predicate η becomes true if we do the assignment (q,r):= (0,x).

A_2:	Subalgorithm IntegerDivision (x,y,q,r) is: (q,r)←(0,x) [η, η and r<y] endIntegerDivision

The predicate η is an invariant predicate and we can apply the iteration rule:

A_3:	Subalgorithm IntegerDivision (x,y,q,r) is: (q,r) ← (0,x) DO r≥y -> [r≥y and η, η and TC] OD endIntegerDivision

For the DO to terminate we must decrease r, and because r≥y, we can use r←r-y.
But η should hold also in the post-condition, so we must have:

$q*y+r = q*y+r-y +y=(q+1)*y + (r-y)$.

So, both r and q must be changed.

A_4:	Subalgorithm IntegerDivision (x,y,q,r) is: (q,r) ← (0,x) DO r≥y -> (q,r) ← (q+1,r-y) OD endIntegerDivision

Example 2
Square root. r = [sqrt(n)]
But we know that: $r \le sqrt(n) < r+1$
⇨ The output predicate is $r^2 \le n < (r+1)^2$
Specification:
φ: $n>1$
ψ: $r^2 \le n < (r+1)^2$
A_0

Subalgorithm RADICALINT(n,r) is:
[φ, ψ]
endSquare

We rewrite the output predicate: $(r^2 <= n < q^2) \wedge (q=r+1)$
We use the middle predicate: $\eta ::= (r^2 \le n < q^2)$

A_1

Subalgorithm Square(n,r) is:
[φ, η]
[η,ψ]
endSquare

The predicate η becomes True (in the first abstract program) for r=0 şi q=n+1.
A_2

Subalgorithm Square(n,r) is:
$(q,r) \leftarrow (n+1,0)$
[η,$\eta \wedge (q=r+1)$]
{ψ}
endSquare

For the second abstract program we can apply the iteration rule.
A_3

Subalgorithm Square (n,r) is:
$(q,r) \leftarrow (n+1,0)$
DO $q \ne r+1$ ->
[$\eta \wedge q \ne r+1$,$\eta \wedge TC$]
OD
endSquare

For the DO to terminate we must decrease r or q (loop condition, q-r must become 1 at the end). The value of p=(q+r)/2 satisfies r<p<q, and (q-r) is changed by changing the interval [r,q] to [r,p] or [p,q].
But η should hold also in the post-condition, so we must have:
If $(p^2 \le n)$ then the assignment $r \leftarrow p$ keeps η invariant.
If $(p^2 > n)$ then the assignment $q \leftarrow p$ keeps η invariant.
A_4

Subalgorithm Square (n,r) is:
$(q,r) \leftarrow (n+1,0)$
DO $q > r+1$ ->
$p \leftarrow (q+r)/2$
If $p^2 \le n \rightarrow r \leftarrow p$
□ $p^2 < n \rightarrow q \leftarrow p$

fi
OD
endSquare

Example 3

Multiplication by repeated additions

Specification:

φ : $(x\geq 0) \wedge (y\geq 0)$

ψ : $z = x*y$

A_0

Subalgorithm Product(x,y,z) is:
[φ, ψ]
endProduct

The postcondition ψ may be satisfied if we use the following predicate:

η::= $(z+u*v = x*y) \wedge (v\geq 0)$

We use this predicate as a middle predicate and the sequential composition rule:

A_1

Subalgorithm Product (x,y,z) is:
[φ, η]
[η,ψ]
endProduct

The first abstract program becomes true with the assignment $(u,v,z) \leftarrow (x,y,0)$

A_2

Subalgorithm Product (x,y,z) is:
$(z,u,v) \leftarrow (0,x,y)$
[η,ψ]
endProduct

The abstract program [η,ψ] may be rewritten as [η,$\eta \wedge (v=0)$] and now we can apply the iteration rule:

A_3

Subalgorithm Product (x,y,z) is:
$(z,u,v) \leftarrow (0,x,y)$
DO $v\neq 0$ ->
[$\eta \wedge v\neq 0$,$\eta \wedge$ TC]
OD
endProduct

For the DO to terminate we must decrease v:

First possibility: $v \leftarrow v-1$. But η should hold also in the post-condition, so we must have: $z+u*v = z + u + u*(v-1)$. So also the assignment $z \leftarrow z+u$ is needed.

Second possibility: $v \leftarrow v/2$, if v is even. . But η should hold also in the post-condition, so we must have: $z+u*v = z + (u*2)* v/2$So also the assignment (u,v):=(u+u, v/2) is needed.

A_4

Subalgorithm Product (x,y,z) is:
$(z,u,v) \leftarrow (0,x,y)$
DO v>0}

```
            Do  (v even) →
                   (u,v) ← (u+u,v div 2)
            OD
            (z,v) ← ( z+u, v-1)
       OD
endProduct
```

Example 4

Raising a number to a power by multiplications

Compute $z = x^y$ by multiple multiplications

Specification:

A0:	φ : $(x>0) \wedge (y\geq 0)$ ψ : $z = x^y$

The predicate $\eta ::= (z*u^v = x^y) \wedge (v\geq 0)$ implies ψ if v=0. Using it as a middle predicate we can apply the sequential composition rule:

A1:	Subalgorithm RaisingPower(x,y,z) is: [φ , η] [η , ψ] endRaisingPower

The η becomes true if (z,u,v) = (1,x,y) (in the first abstract program):

A2:	Subalgorithm RaisingPower(x,y,z) is: (z,u,v) ← (1,x,y) [η,$\eta \wedge$ v=0] endRaisingPower

The predicate η is invariant, we can apply the iteration rule.

A3:	Subalgorithm Putere(x,y,z) is: (z,u,v) ← (1,x,y) DO v≠0 → [$\eta \wedge$v≠0,$\eta \wedge$ TC] OD endRaisingPower

For the DO to terminate we must decrease v:

First possibility: v←v-1. But η should hold also in the post-condition, so we must have: $z*u^v = z * u*u^{v-1}$. So also the assignment (z,v)←(z*u,v-1) is needed.

Second possibility: v←v/2, if v is even. . But η should hold also in the post-condition, so we must have: $z*u^v = z* (u*u)^{v/2}$ .So also the assignment (u,v) ← (u*u,v/2) is needed.

A4	Subalgorithm RaisingPower1 (x,y,z) is: (z,u,v) ← (1,x,y) DO v≠0 (z,v) ← (z*u,v-1)

	OD endRaisingPower

A4	Subalgorithm RaisingPower2 (x,y,z) is: (z,u,v) ← (1,x,y) DO v≠0 DO (v even) →(u,v) ← (u*u,v/2) OD (z,v) ← (z*u,v-1) OD endRaisingPower

Example 5

Greatest common divisor

Specification:

φ : x>0, y > 0

ψ : d=gcd(x,y)

A_0

Subalgorithm GCD(x,y,d) is:
[φ, ψ]
endGCD

The middle predicate η::= gcd(d,s)=gcd(x,y) is used for the sequential composition rule.

A_1

Subalgorithm GCD(x,y,d) is:
[φ, η]
[η,ψ]
endGCD

The first abstract program becomes true if (d,s)=(x,y), so we use assignement rule:

A_2

Subalgorithm GCD(x,y,d) is:
(d,s) ← (x,y)
[η,ψ]
endGCD

If d=s then η implies ψ. So, we can write:

A_3

Subalgorithm GCD(x,y,d) is:
(d,s) ← (x,y)
[η,η ∧ (d=s)]
endGCD

We can apply now the iteration rule:

A_2

Subalgorithm GCD(x,y,d) is:
(d,s) ← (x,y)
DO d≠s →
[η∧d≠s,η ∧ TC]
OD
endGCD

For d≠s we have d>s or d<s. We know that for d>s we have gcd(d,s)=gcd(d-s,s) and the assignment d ←d-s keepsη invariant.

A_3

```
Subalgorithm GCD (x,y,d) este:
    (d,s) ← (x,y)
    DO d≠s →
        IF d>s  → d←d-s
         □ d<s → s←s-d
        FI
    OD
    gcd←s
endGCD
```

Example 6

Insertion

$A = (a_1, a_2,...,a_n)$ an array with n components ordered in decrease order and x a value. Insert x in A such that A remains ordered and A containes a new value x.

The predicate ORD is define by:

$ORD(n,A) ::= (\forall i,j: 1\le i,j\le n,\ i\le j \Rightarrow a_i\le a_j)$

Specification:

$\varphi ::= ORD(n,A) \wedge (n \text{ natural})$

$\psi ::= ORD(n+1,A)$ and (A contains the initial elements and a new element x)

A_0:	$[\varphi, \psi]$

There are two possibilities ($x<a_n$ and $n\ne 0$) or ($x\ge a_n$ or $(n=0)$):

A_1:	Subalgorithm Insert(n,A,x) is:
	  Dacă $x<a_n$ şi $n\ne 0$
	    atunci $[\varphi \wedge (x<a_n) \wedge n\ne 0, \psi]$
	    altfel $[\varphi \wedge ((x\ge a_n)\vee(n=0)), \psi]$
	  sfdacă
	endInsert

A doua propoziţie nestandard se rafinează printr-o atribuire

A_2:	Subalgorithm Insert(n,A,x) is:
	  Dacă $x<a_n$ şi $n\ne 0$
	    atunci $[\varphi \wedge (x<a_n) \wedge n\ne 0, \psi]$
	    altfel $(n,a_{n+1}) \leftarrow (n+1,x)$
	  sfdacă
	endInsert

Să notăm prin η următorul predicat

$$ORD(n,A) \wedge [\, (x<a_1) \wedge (p=1) \vee (a_{p-1} \leq x < a_p) \wedge (1<p\leq n)]$$

Care este o postconditie pentru o problemă de căutare şi să folosim regula secvenţei. Ajungem la:

A_3:	Subalgorithm Insert(n,A,x) is:
	  IF $x<a_n$ and $n\neq 0 \rightarrow$
	        $[\varphi \wedge (x<a_n) \wedge n\neq 0, \eta]$
	        $[\eta , \psi]$
	   □ (not $x<a_n$ and $n\neq 0$) $\rightarrow (n,a_{n+1}) \leftarrow (n+1,x)$
	  FI
	endInsert

Vom satisface postcondiţia η în urma apelului subalgoritmului de căutare, astfel că ajungem la:

A_4:	Subalgorithm Insert(n,A,x) is:
	    IF $x<a_n$ and $n\neq 0 \rightarrow$
	        CALL SEARCH(x,n,A,p)
	        $[\eta , \psi]$
	    □ (not $x<a_n$ and $n\neq 0$) $\rightarrow (n,a_{n+1}) \leftarrow (n+1,x)$
	  FI
	endInsert

After the search we know that x is between a_{p-1} and a_p , so x must be inserted on position p, so we have

$a'_{i+1} \leftarrow a_i$, for i=n,n-1,...,p

and $a'_p \leftarrow x$.

$n' \leftarrow n+1$

We use the assignments:

```
i←n;
DO i≥p →
      a[i+1]←a[i]
      i←i-1
OD
```

A_5:	Subalgorithm Insert(n,A,x) is:
	    IF $x<a_n$ and $n\neq 0$ $\rightarrow$
	        CALL SEARCH(x,n,A,p)
	          $i\leftarrow n$
	          DO $i\geq p$ $\rightarrow$
	              $a_{i+1}\leftarrow a_i$
	              $i\leftarrow i-1$
	          OD
	          $a_p\leftarrow x$
	          $n\leftarrow n+1$
	      ☐ (not $x<a_n$ and $n\neq 0$) $\rightarrow (n,a_{n+1}) \leftarrow (n+1,x)$
	  FI
	endInsert

Another refinement regardi8nt the $n\leftarrow n+1$ assignment:

A_5:	Subalgorithm Insert(n,A,x) is:
	    IF $x<a_n$ and $n\neq 0$ $\rightarrow$
	        CALL SEARCH(x,n,A,p)
	          $i\leftarrow n$
	          DO $i\geq p$ $\rightarrow$
	              $a_{i+1}\leftarrow a_i$
	              $i\leftarrow i-1$
	          OD
	          $a_p\leftarrow x$
	      ☐ (not $x<a_n$ and $n\neq 0$) $\rightarrow (n,a_{n+1}) \leftarrow (n+1,x)$
	   FI
	   $n\leftarrow n+1$
	endInsert

Example 7

InsertionSort

Let $A = (a_1, a_2,...,a_n)$ be an array with n integer components. The problem requires to order the components of A.

Specification

φ::= $n\geq 2$, A has integer components

ψ ::= ORD(n,A) and A has the same elements as in the precondition

A_0:	[φ, ψ]

We use the middle predicate ORD(k,A) and apply the sequential composition rule:

A_1:	Subalgorithm InsertSort(n,A) is:
	  [φ, ORD(k,A)]
	  [ORD(k,A), ψ]
	endInsertSort

The first abstract program may be refined to an assignment

A_2:	Subalgorithm InsertSort(n,A) is:
	  k←1
	  [ORD(k,A), ψ]
	endInsertSort

Wer can rewrite the remained abstract program remarking that

$$ORD(k,A) \wedge (k=n) \Rightarrow \psi$$

A_3:	Subalgorithm InsertSort(n,A) is:
	  k←1
	  [ORD(k,A), ORD(k, A) şi (n=k)]
	endInsertSort

We now can apply the iteration rule

A_4:	Subalgorithm InsertSort(n,A) is:
	  k←1
	  DO k<n →
	    [ORD(k,A) and k<n, ORD(k,A) and TC]
	  OD
	endInsertSort

For the DO to terminate we must increase k:
First possibility: k←k+1. But η(k)::=ORD(k,A) invariant – by modifying k by k+1 the predicate η(k|k+1) must be true.

A_5:	Subalgorithm InsertSort(n,A) is:
	  k←1
	  DO k<n →
	    [k<n and η(k), η(k|k+1)]
	  OD
	endInsertSort

The abstract program

$$[\ (k<n) \wedge\ ORD(k,A), ORD(k+1,A)]$$

Corresponds to the following subproblem:

If ORD(k,A) (the first k elements in A are orderes) then modify the A such that the first k+1 elements to be ordered. This can be achieve by calling a subalgorithn that inserts the a_{k+1} component such that after insertion the postcondition ORD(k+1,A) is true.

A_4:	Subalgorithm InsertSort(n,A) is:
	  k←1
	  DO k<n →
	      CALL INSERT(k,A, a_{k+1})
	   OD
	endInsertSort