## Sumar



### Exemplul 1. Împărțire întreagă (cât şi rest)

Specificare
φ: $(x\geq 0) \wedge (y>0)$
ψ: $(x=q*y+r) \wedge (0\leq r<y)$

| | |
|---|---|
| $A_0$: | Subalgoritmul ImpInt (x,y,q,r) este:<br>[φ, ψ ]<br>sfImpInt |

Fie η: $(x=q*y+r) \wedge (0\leq r)$ un predicat intermediar (middle predicate). Prin aplicarea regulii compunerii secvențiale:

| | |
|---|---|
| $A_1$: | Subalgoritmul ImpInt (x,y,q,r) este:<br>[φ, η]<br>[η,ψ ]<br>sfImpInt |

Predicatul η devine true prin atribuirea (q,r):= (0,x).

| | |
|---|---|
| $A_2$: | Subalgoritmul ImpInt (x,y,q,r) este:<br>(q,r)⟵(0,x)<br>[η, η ∧ r<y]<br>sfImpInt |

Predicatul η este un predicat invariant. Prin aplicarea regulii iterației:

| | |
|---|---|
| $A_3$: | Subalgoritmul ImpInt (x,y,q,r) este:<br>(q,r) ⟵ (0,x)<br>DO r≥y ---><br>[r≥y ∧ η, η ∧ TC]<br>OD<br>sfImpInt |

**Dezvoltarea algoritmilor corecţi din specificaţii**

Pentru ca DO să se termine, r trebuie să scadă (să descrească); deoarece r≥y, putem reduce valoarea lui r cu y, adică r⟵r-y.

η trebuie să rămână true şi în post-condiție, deci este necesar ca:

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

Astfel, r şi q îşi modifică valoarea.

| | |
|---|---|
| $A_4$: | Subalgoritmul ImpInt (x,y,q,r) este:<br>(q,r) ⟵ (0,x)<br>DO r≥y -><br>(q,r) ⟵ (q+1,r-y)<br>OD<br>sfImpInt |

### Exemplul 2. Rădăcină pătrată

- r = [sqrt(n)]; se ştie că r ≤ sqrt(n) < r+1;
- post-condiția este $r^2 \le n < (r+1)^2$;

Specificare:

φ: n>1

ψ: $r^2 \le n < (r+1)^2$

$A_0$

Subalgoritmul RadPatrata(n, r) este:
[φ, ψ]
endRadPatrata

Rescriem predicatul de ieşire în forma: $(r^2 <= n < q^2) \wedge (q=r+1)$.

Folosim predicatul intermediar (middle predicate) η ::= $(r^2 \le n < q^2)$.

$A_1$

Subalgoritmul RadPatrata (n, r) este:
[φ, η]
[η,ψ]
sfRadPatrata

Predicatul η devine true în $A_1$ pentru r=0 şi q=n+1.

$A_2$

Subalgoritmul RadPatrata (n, r) este:
(q,r) ⟵ (n+1,0)
[η,η ∧ (q=r+1)]
{ψ}
sfRadPatrata

Pentru $A_2$ se pote aplica regula iterației.

$A_3$

Subalgoritmul RadPatrata (n,r) este:
(q,r) ⟵ (n+1,0)
DO q≠r+1 ->
[η∧ q≠r+1,η ∧ TC]
OD
sfRadPatrata

Pentru ca DO să se termine, este necesar ca r sau q să descrească; q-r trebuie să devină 1 la final.

Expresia p=(q+r)/2 satisface condiția r<p<q; iar (q-r) se actualizează prin modificarea intervalului [r,q] la [r,p] sau [p,q].
Dar η trebuie să rămână true în post-condiție, deci este necesar ca:

dacă ($p^2 \le n$) atunci atribuirea r ⟵ p satisface invariantul η;
dacă ($p^2 > n$) atunci atribuirea q ⟵ p satisfice invariantul η.

$A_4$

```
Subalgoritmul RadPatrata (n,r) este:
    (q,r) ⟵ (n+1,0)
    DO q>r+1 ->
        p⟵ (q+r)/2
        IF p²≤n → r⟵p
        □p²<n→  q⟵p
        FI
    OD
sfRadPatrata
```

## Exemplul 3. Înmulțire prin adunări repetate

Specificare:

φ : (x≥0) ∧ (y≥0)
ψ : z = x*y

$A_0$

```
Subalgoritmul Produs(x,y,z) este:
    [φ, ψ]
sfProdus
```

Post-condiția ψ este satisfacută dacă se utilizează un predicat intermediar η:
η::= (z+u*v = x*y ) ∧ (v≥0).
De asemenea, se aplică regula compunerii secvențiale:

$A_1$

```
Subalgoritmul Produs (x,y,z) este:
    [φ, η]
    [η,ψ]
sfProdus
```

Programul abstract $A_1$ devine true prin atribuirea (u,v,z) ⟵ (x,y,0).

$A_2$

```
Subalgoritmul Produs (x,y,z) este:
    (z,u,v) ⟵ (0,x,y)
    [η,ψ]
sfProdus
```

Programul abstract [η,ψ] se poate rescrie prin [η,η ∧ (v=0)], ceea ce permite aplicarea regulii iterației:

$A_3$

```
Subalgoritmul Produs (x,y,z) este:
    (z,u,v) ⟵ (0,x,y)
    DO v≠0 ->
      [η∧v≠0,η ∧ TC]
    OD
sfProdus
```

**Dezvoltarea algoritmilor corecţi din specificaţii**

Pentru ca DO să se termine, este necesar să micşorăm pe v:

- prima posibilitate:
  - v←v-1; dar η trebuie satisfăcut şi în postcondiție, deci este necesar ca:
    - z+u*v = z + u + u*(v-1) şi atribuirea z←z+u trebuie să aibă loc;
- a doua posibilitate:
  - v←v/2, dacă v este par; dar η trebuie satisfăcut şi în post-condiție, deci este necesar ca:
    - z+u*v = z + (u*2)* v/2 şi atribuirea (u,v):=(u+u, v/2)trebuie realizată.

$A_4$

```
Subalgoritmul Produs (x,y,z) este:
      (z,u,v) ⟵ (0,x,y)
      DO v>0}
            Do (v % 2 == 0) →
                  (u,v) ⟵ (u+u,v div 2)
            OD
            (z,v) ⟵ ( z+u, v-1)
      OD
sfProdus
```

## Exemplul 4. Cel mai mare divizor comun al două numere naturale

Specificare:

φ : x>0, y > 0

ψ : d=cmmmdc(x,y)

$A_0$

```
Subalgoritmul CMMDC(x,y,d) este:
      [φ, ψ]
sfCMMDC
```

Predicatul intermediar η::= cmmdc(d,s)=cmmdc(x,y) este utilizat pentru a aplica regula compunerii secvențiale.

$A_1$

```
Subalgoritmul CMMDC(x,y,d) este:
      [φ, η]
      [η,ψ]
sfCMMDC
```

Programul abstract $A_1$ devine true prin atribuirea (d,s)=(x,y), folosind regula atribuirii:

$A_2$

```
Subalgoritmul CMMDC(x,y,d) este:
      (d,s) ⟵ (x,y)
      [η,ψ]
sfCMMDC
```

Dacă d=s atunci η implică pe ψ. Astfel, se poate scrie următorul program abstract:

$A_3$

```
Subalgoritmul CMMDC(x,y,d) este:
      (d,s) ⟵ (x,y)
      [η,η ∧ (d=s)]
sfCMMDC
```

Prin aplicarea regulii iterației se obține:

$A_2$

```
Subalgoritmul CMMDC(x,y,d) este:
```

**Dezvoltarea algoritmilor corecţi din specificaţii**

```
                (d,s) ← (x,y)
                DO d≠s →
                        [η∧d≠s,η ∧ TC]
              OD
        sfCMMDC
```

Pentru d≠s avem condiţiile d>s şi d<s. Se ştie că pentru d>s avem cmmdc(d,s)=cmmdc(d-s,s) şi atribuirea d ←d-s păstrează predicatul η invariant.

$A_3$

```
        Subalgoritmul CMMDC(x,y,d) este:
                (d,s) ← (x,y)
                DO d≠s →
                        IF d>s → d←d-s
                          □d<s → s←s-d
                   FI
                OD
                CMMDC←s
        sfCMMDC
```

## Example 5. Raising a number to a power by multiplications

Compute $z = x^y$ by multiple multiplications

Specification:

| | |
|---|---|
| A0: | φ : (x>0) ∧ (y≥0)<br>ψ : $z = x^y$ |

The predicate η ::= ($z*u^v = x^y$) ∧ (v≥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:<br>[φ , η]<br>[η , ψ ]<br>endRaisingPower |

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

| | |
|---|---|
| A2: | Subalgorithm RaisingPower(x,y,z) is:<br>(z,u,v) ← (1,x,y)<br>[η,η∧ v=0 ]<br>endRaisingPower |

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

| | |
|---|---|
| A3: | Subalgorithm Putere(x,y,z) is:<br>(z,u,v) ← (1,x,y)<br>DO v≠0 →<br>[η∧v≠0,η∧ TC ]<br>OD<br>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.

**Dezvoltarea algoritmilor corecţi din specificaţii**

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:<br>(z,u,v) ⟵ (1,x,y)<br>DO v≠0<br>(z,v) ⟵ (z*u,v-1)<br>OD<br>endRaisingPower |

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

## 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≤i,j≤n, i≤j $\Rightarrow$ $a_i \le a_j$)

Specification:

φ::= ORD(n,A) ∧ (n natural)

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

| | |
|---|---|
| $A_0$: | [φ, ψ] |

There are two possibilities (x<$a_n$ and n≠0) or ( x≥$a_n$ or (n=0) ):

| | |
|---|---|
| $A_1$: | Subalgorithm Insert(n,A,x) is:<br>Dacă x<$a_n$ şi n≠0<br>atunci [φ ∧ (x<$a_n$) ∧ n≠0, ψ ]<br>altfel [φ ∧ ((x≥$a_n$)∨(n=0)), ψ ]<br>sfdacă<br>endInsert |

A doua propoziție nestandard se rafinează printr-o atribuire

| | |
|---|---|
| $A_2$: | Subalgorithm Insert(n,A,x) is:<br>Dacă x<$a_n$ şi n≠0<br>atunci [φ ∧ (x<$a_n$) ∧ n≠0, ψ ]<br>altfel (n,$a_{n+1}$) ⟵ (n+1,x)<br>sfdacă<br>endInsert |

**Dezvoltarea algoritmilor corecţi din specificaţii**

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 secvenței. Ajungem la:

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

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

| | |
|---|---|
| $A_4$: | Subalgorithm Insert(n,A,x) is:<br>  IF $x<a_n$ and $n\neq 0$ → <br>    CALL SEARCH(x,n,A,p)<br>     $[\eta , \psi ]$<br>  □not $x<a_n$ and $n\neq 0$) →$(n,a_{n+1}) \leftarrow (n+1,x)$<br>  FI<br>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:<br>  IF $x<a_n$ and $n\neq 0$ → <br>    CALL SEARCH(x,n,A,p)<br>     $i \leftarrow n$<br>     DO $i\geq p$ → <br>      $a_{i+1} \leftarrow a_i$<br>      $i \leftarrow i-1$<br>     OD<br>     $a_p \leftarrow x$<br>     $n \leftarrow n+1$<br>  □(not $x<a_n$ and $n\neq 0$) →$(n,a_{n+1}) \leftarrow (n+1,x)$<br>  FI<br>endInsert |

Another refinement regardi8nt the n←n+1 assignment:

| | |
|---|---|
| $A_5$: | Subalgorithm Insert(n,A,x) is:<br>  IF $x<a_n$ and $n\neq 0$ →<br>      CALL SEARCH(x,n,A,p)<br>        i←n<br>        DO i≥p →<br>            $a_{i+1}$←$a_i$<br>            i←i-1<br>        OD<br>        $a_p$←x<br>    □(not $x<a_n$ and $n\neq 0$) →(n,$a_{n+1}$) ← (n+1,x)<br>  FI<br>  n←n+1<br>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≥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:<br>  [φ, ORD(k,A)]<br>  [ORD(k,A), ψ]<br>endInsertSort |

The first abstract program may be refined to an assignment

| | |
|---|---|
| $A_2$: | Subalgorithm InsertSort(n,A) is:<br>  k←1<br>  [ORD(k,A), ψ]<br>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:<br>  k←1<br>  [ORD(k,A), ORD(k, A) şi (n=k)]<br>endInsertSort |

We now can apply the iteration rule

**Dezvoltarea algoritmilor corecţi din specificaţii**

| | |
|---|---|
| $A_4$: | Subalgorithm InsertSort(n,A) is:<br>k←1<br>DO k<n →<br>[ORD(k,A) and k<n, ORD(k,A) and TC]<br>OD<br>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:<br>k←1<br>DO k<n →<br>[k<n and η(k), η(k\|k+1)]<br>OD<br>endInsertSort |

The abstract program

[ (k<n) ∧ 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:<br>k←1<br>DO k<n →<br>CALL INSERT(k,A, $a_{k+1}$)<br>OD<br>endInsertSort |