
Algoritmi și structuri de date (I). Seminar 4: Verificarea corectitudinii algoritmilor.

- Formularea precondiții și postcondiții
 - Identificare proprietăți invariante
 - Demonstrarea corectitudinii
-

Problema 1 Demonstrați că algoritmul `identic6` returnează *True* dacă toate elementele din $x[1..n]$ sunt identice și *False* în caz contrar.

```
identic6(x[1..n])
rez ← True
i ← 1
while i < n do
  i ← i + 1
  if x[i - 1] = x[i] then
    rez ← False
  end if
end while
return rez
```

Indicație. Se stabilește starea algoritmului înainte de intrarea în ciclul `while` (precondiția) și postcondiția specificată în enunț. Se arată că afirmația "dacă $x[1..i]$ are toate elementele identice atunci *rez* este *True* altfel *rez* este *False*" satisface cerințele specifice unui invariant.

Problema 2 Scrieți un algoritm pentru calculul puterii întregi a unei valori reale nenule (a^p cu $a \in \mathbb{R}^*$ și $p \in \mathbb{Z}$) și demonstrați corectitudinea algoritmului.

Indicație. Se calculează $a^{|p|} = \prod_{i=1}^{|p|} a$ și se analizează semnul lui p pentru a decide dacă se returnează $a^{|p|}$ sau $1/a^{|p|}$. Pentru demonstrarea corectitudinii se descrie prelucrarea repetitivă folosind `while` și se identifică o proprietate invariantă.

Problema 3 Să se analizeze corectitudinea algoritmilor `conversie_10_q` și `conversie_q_10` care realizează conversia unui număr din baza 10 în baza q ($q \geq 2$) respectiv din baza q în baza 10.

<code>conversie_10_q(integer n, q)</code>	<code>conversie_q_10(integer c[0..k], q)</code>
<code>integer c[0..k], k, m</code>	<code>integer i, n</code>
1: $m \leftarrow n$	1: $i \leftarrow k$
2: $k \leftarrow 0$	2: $n \leftarrow c[i]$
3: $c[k] \leftarrow m \text{ MOD } q$	3: while $i > 0$ do
4: $m \leftarrow m \text{ DIV } q$	4: $i \leftarrow i - 1$
5: while $m > 0$ do	5: $n \leftarrow n * q + c[i]$
6: $k \leftarrow k + 1$	6: end while
7: $c[k] \leftarrow m \text{ MOD } q$	7: return n
8: $m \leftarrow m \text{ DIV } q$	
9: end while	
10: return $c[0..k]$	

Rezolvare. În cazul primului algoritm, k reprezintă numărul de cifre în reprezentarea în baza q a numărului n iar tabloul $c[0..k]$ conține cifrele reprezentării începând cu cea mai puțin semnificativă. Cu aceste notații, precondițiile sunt $P = \{n \in \mathbb{N}, q \in \mathbb{N}, q \geq 2\}$, iar postcondiția este $n = c[k]q^k + c[k-1]q^{k-1} + \dots + c[1]q + c[0]$.

După execuția primelor patru instrucțiuni, starea algoritmului este $\{n = m \cdot q + c[k], k = 0\}$ adică $\{n = m \cdot q^{k+1} + c[k] \cdot q^k\}$. Intuim că proprietatea invariantă este: $n = m \cdot q^{k+1} + \sum_{i=0}^k c[i] \cdot q^i$. Se observă imediat că pentru $k = 0$ este echivalentă cu starea algoritmului la intrarea în ciclul **while**.

După execuția liniei 6 proprietatea devine $n = m \cdot q^k + \sum_{i=0}^{k-1} c[i] \cdot q^i$.

După execuția liniilor 7 și 8 vechea valoare a lui m (m_p) poate fi exprimată prin noua valoare a lui m (m_{p+1}) astfel: $m_p = m_{p+1} \cdot q + c[k]$ motiv pentru care proprietatea analizată devine iar $n = m \cdot q^{k+1} + \sum_{i=0}^k c[i] \cdot q^i$. Rămâne să arătăm că la finele ciclului **while** această proprietate implică postcondiția. Intr-adevăr, la ieșirea din **while**, valoarea lui m este 0 deci proprietatea devine: $n = \sum_{i=0}^k c[i] \cdot q^i$ adică tocmai postcondiția.

În ceea ce privește finitudinea, se observă ușor că funcția $t(p) = m_p$ (m_p este valoarea variabilei m la a p -a execuție a ciclului) satisface proprietățile unei funcții de terminare.

Algoritmul **conversie_q_10** se bazează pe faptul că determinarea valorii în baza 10 corespunzătoare reprezentării în baza q , $c[0..k]$, este echivalentă cu calculul sumei $\sum_{i=0}^k c[i]q^i$. Precondiția se rezumă la $P = \{k > 0\}$ iar postcondiția este $Q = \{n = \sum_{i=0}^k c[i]q^i\}$. După execuția primelor două linii, starea algoritmului este $n = c[k] = c[k] \cdot q^{k-i} = \sum_{j=i}^k c[j]q^{j-i}$. Intuim că $n = \sum_{j=i}^k c[j]q^{j-i}$ este proprietate invariantă. Întrucât la intrarea în ciclu este adevărată rămâne să arătăm că rămâne adevărată după execuția corpului ciclului și că la final implică postcondiția.

După execuția liniei 4 proprietatea devine (în raport cu noua valoare a variabilei i): $n = \sum_{j=i+1}^k c[j]q^{j-i-1}$. Prin execuția liniei 5 valoarea variabilei n se modifică astfel că devine din nou adevărată relația $n = \sum_{j=i}^k c[j]q^{j-i}$. La ieșirea din ciclu variabila i are valoarea 0, astfel că se obține $n = \sum_{j=0}^k c[j]q^j$ adică postcondiția. Finitudinea rezultă imediat remarcând faptul că $t(p) = i_p$ are proprietățile unei funcții de terminare.

Problema 4 Demonstrați punând în evidență un invariant și o funcție de terminare că după execuția algoritmului de mai jos variabila n va conține valoarea în baza 10 corespunzătoare șirului binar $b[0..k]$ ($b[i] \in \{0, 1\}$, $i = \overline{0, k}$).

alg(integer $b[0..k]$)

integer n, i
 1: $i \leftarrow 0$
 2: $n \leftarrow 0$
 3: **while** $i \leq k$ **do**
 4: $n \leftarrow n * 2 + b[k - i]$
 5: $i \leftarrow i + 1$
 6: **end while**
 7: **return** n

Indicație. Se arată că $n = \sum_{j=0}^{i-1} b[k-j]2^{i-j-1}$ este proprietate invariantă iar $t(p) = (k+1) - i_p$ este funcție de terminare.

Problema 5 Să se demonstreze corectitudinea algoritmului de determinare a valorii obținute prin inversarea ordinii cifrelor unui număr natural. Considerând că $n = c_k c_{k-1} \dots c_1 c_0$ este numărul inițial, valoarea căutată este $m = c_0 c_1 \dots c_k$. Prin urmare preconditionia este: $n = \sum_{i=0}^k c_i 10^i$ iar postcondiția este $m = \sum_{i=0}^k c_i 10^{k-i}$. Metoda de calcul a lui m este descrisă în algoritmul **inversare** descris în continuare.

inversare(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 m
```

Rezolvare. Utilizarea variabilei p nu este necesară pentru descrierea algoritmului însă introducerea ei ușurează demonstrarea corectitudinii. Să considerăm relațiile: $\{n = \sum_{i=p}^k c[i]10^{i-p}, m = \sum_{i=0}^{p-1} c[i]10^{p-1-i}\}$. Se observă că precondiția implică prima relație (pentru $p = 0$) iar a doua este evident adevărată pentru $m = 0$ și $p = 0$ (m este descrisă în acest caz ca o sumă vidă). După execuția liniei 4 relațiile devin: $\{n = \sum_{i=p-1}^k c[i]10^{i-p+1}, m = \sum_{i=0}^{p-2} c[i]10^{p-2-i}\}$.

După execuția liniei 5, a doua relație devine $m = \sum_{i=0}^{p-2} c[i]10^{p-1-i} + c[p-1] = \sum_{i=0}^{p-1} c[i]10^{p-1-i}$. După execuția liniei 6 obținem că $n = \sum_{i=p}^k c[i]10^{i-p}$. Când n este 0 înseamnă că $p = k + 1$ astfel că $m = \sum_{i=0}^{p-1} c[i]10^{p-1-i}$ implică postcondiția.

Finitudinea este asigurată de faptul că oricare dintre funcțiile $t(p) = n_p$ sau $t(p) = k + 1 - p$ satisface proprietățile unei funcții de terminare.

Problema 6 Să se demonstreze că algoritmul produs calculează corect produsul a două numere naturale nenule.

produs(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
```

Indicație. Se observă că algoritmul corespunde metodei de înmulțire descrisă "à la russe". Pornim de la ipoteza că valoarea a poate fi reprezentată în baza 2 prin $k + 1$ cifre binare $c_k c_{k-1} \dots c_1 c_0$ specificate în tabloul $c[0..k]$.

Cu aceste notații precondiția este $a = c_k 2^k + c_{k-1} 2^{k-1} \dots c_1 2 + c_0$ iar postcondiția $s = ab$. Proprietatea invariantă este constituită din relațiile: $\{s = (\sum_{i=0}^{p-1} c_i 2^i)b, x = \sum_{i=p}^k c_i 2^{i-p}, y = b2^p\}$. Este ușor de verificat că aceste relații sunt satisfăcute înainte de intrarea în ciclu. Prin execuția liniei 6 relațiile devin: $\{s = (\sum_{i=0}^{p-2} c_i 2^i)b, x = \sum_{i=p-1}^k c_i 2^{i-p+1}, y = b2^{(p-1)}\}$. Prin execuția liniei 7 se modifică prima relație, devenind: $s = (\sum_{i=0}^{p-1} c_i 2^i)b$. Prin execuția liniei 8, a doua relație devine $x = \sum_{i=p}^k c_i 2^{i-p}$ iar prin execuția liniei 9 a treia relație devine $y = b2^p$. Prin urmare relațiile sunt invariante în raport cu ciclul. La părăsirea ciclului valoarea lui x este 0 iar $p = k + 1$, astfel că prima relație implică postcondiția.

Finitudinea se demonstrează folosind ca funcție de terminare pe $t(p) = x_p$.

Problema 7 Să se demonstreze că algoritmul `inversare_sir` descris în continuare realizează inversarea ordinii elementelor din tabloul $x[1..n]$ primit ca parametru.

`inversare_sir`($x[1..n]$)

```

1:  $i \leftarrow 0$ 
2: while  $i < \lfloor (n+1)/2 \rfloor$  do
3:    $i \leftarrow i + 1$ 
4:    $x[i] \leftrightarrow x[n+1-i]$ 
5: end while
6: return  $x[1..n]$ 

```

Indicație. Fie $x_0[1..n]$ conținutul inițial al tabloului. Cu această notație preconditionia poate fi specificată prin $P = \{x[i] = x_0[i], i = \overline{1, n}\}$ iar postcondiția prin $Q = \{x[i] = x_0[n+1-i], i = \overline{1, n}\}$. Se poate demonstra că următoarele relații $\{i \leq n+1-i, x[j] = x_0[n+1-j], x[n+1-j] = x_0[j], j = \overline{1, i}\}$ sunt invariante în raport cu ciclul `while` iar la ieșirea din ciclu ($i = \lfloor (n+1)/2 \rfloor$) implică postcondiția.

Problema 8 Identificați proprietăți invariante pentru prelucrările repetitive din algoritmi `alg1` și `alg2` descriși în continuare și stabiliți ce returnează fiecare dintre ei când este apelat pentru valori naturale nenule.

`alg1`(`integer` a, b)

```

integer  $x, y, z$ 
1:  $x \leftarrow a$ 
2:  $y \leftarrow b$ 
3:  $z \leftarrow 0$ 
4: while  $y > 0$  do
5:    $z \leftarrow z + x$ 
6:    $y \leftarrow y - 1$ 
7: end while
8: return  $z$ 

```

`alg2`(`integer` a, b)

```

integer  $x, y, z$ 
1:  $x \leftarrow a$ 
2:  $y \leftarrow b$ 
3:  $z \leftarrow 0$ 
4: while  $x \geq y$  do
5:    $x \leftarrow x - y$ 
6:    $z \leftarrow z + 1$ 
7: end while
8: return  $z, x$ 

```

Indicație. Notăm cu p contorul prelucrării iterative (p va avea valoarea 0 înainte de intrarea în ciclu și este incrementat cu 1 la fiecare execuție a ciclului). Folosind această notație (chiar dacă p nu este variabilă explicită în cadrul algoritmului) se observă că pentru `alg1` relațiile $\{y = b-p, z = px, x = a\}$ sunt invariante în raport cu prelucrarea repetitivă. La ieșirea din ciclul `while` variabila y are valoarea 0, prin urmare $p = b$ iar $z = px = ba$. Deci algoritmul `alg1` returnează produsul ab . În cazul algoritmului `alg2` relațiile $\{x = a - pb, y = b, z = p\}$ sunt invariante. La ieșirea din ciclu $a = z \cdot b + x$ și $x < y = b$. Aceasta înseamnă că z va conține câtul împărțirii lui a la b , iar x restul aceleiași împărțiri.

Problema 9 Să se demonstreze că algoritmul `adunare` descris în continuare corespunde adunării în baza 2 a două numere reprezentate în binar pe $n+1$ poziții.

`adunare`(`integer` $a[0..n], b[0..n]$)

```

integer  $c[0..n+1], s, report$ 
1:  $c[0..n+1] \leftarrow 0$ 
2:  $report \leftarrow 0$ 
3:  $i \leftarrow 0$ 
4: while  $i \leq n$  do
5:    $s \leftarrow a[i] + b[i] + report$ 
6:    $c[i] \leftarrow s \text{ MOD } 2$ 
7:    $report \leftarrow s \text{ DIV } 2$ 
8:    $i \leftarrow i + 1$ 
9: end while
10:  $c[n+1] \leftarrow report$ 
11: return  $c[0..n+1]$ 

```

Indicație. Notând cu $x_{0..i}$ valoarea corespunzătoare tabloului de cifre binare $x[0..i]$ se poate arăta că proprietatea $\{c_{0..i} = a_{0..i-1} + b_{0..i-1}\}$ este invariantă în raport cu ciclul, iar la ieșirea din ciclu implică $c_{0..(n+1)} = a_{0..n} + b_{0..n}$ adică tabloul $c[0..n+1]$ conține suma valorilor corespunzătoare reprezentărilor binare din $a[0..n]$ și $b[0..n]$.

Problema 10 Se consideră algoritmul **alg** descris în continuare. Să se identifice o proprietate invariantă corespunzătoare ciclului și să se stabilească care este efectul algoritmului.

```

alg(real  $a[1..n]$ )


---


integer  $i$ 
1:  $i \leftarrow 1$ 
2: while  $i \leq n - 1$  do
3:   if  $a[i] > a[i + 1]$  then
4:      $a[i] \leftrightarrow a[i + 1]$ 
5:   end if
6:    $i \leftarrow i + 1$ 
7: end while
8: return  $a[1..n]$ 

```

Indicație. După prima execuție a corpului ciclului va fi satisfăcută proprietatea $a[1] \leq a[2]$. După a două execuție a ciclului va fi satisfăcută proprietatea $a[2] \leq a[3]$ (de remarcat faptul că valoarea elementului de pe poziția 2 nu este neapărat aceeași cu cea obținută după prima etapă a algoritmului ceea ce înseamnă că nu e în mod necesar adevărată afirmația că $a[1] \leq a[2] \leq a[3]$ ci doar afirmația că $a[3] \geq a[2]$ și $a[3] \geq a[1]$). Acestea sugerează că după execuția pasului i al ciclului este adevărată afirmația $a[i] = \max_{j=\overline{1,i}} a[j]$. Să demonstrăm că această afirmație este într-adevăr un invariant al ciclului. Pentru $i = 1$ este implicit satisfăcută. După execuția liniei 3 devine $a[i + 1] = \max_{j=\overline{1,i+1}} a[j]$. După execuția liniei 4 devine iar adevărată afirmația $a[i] = \max_{j=\overline{1,i}} a[j]$ prin urmare aceasta este o proprietate invariantă a ciclului. La părăsirea ciclului, i va fi n deci se obține că $a[n] = \max_{j=\overline{1,n}} a[j]$, adică efectul algoritmului este că aduce valoarea maximă a tabloului pe ultima poziție.

Problema 11 Propuneți un algoritm care transformă un tablou $a[1..n]$ prin interschimbări de elemente vecine astfel încât valoarea minimă ajunge pe prima poziție a tabloului. Demonstrați corectitudinea algoritmului identificând un invariant și o funcție de terminare.

Indicație. Se parcurge tabloul pornind de la ultimul element și se compară fiecare element cu predecesorul său. Dacă elementul curent este mai mic decât predecesorul atunci se interschimbă elementele.

Problema 12 Se consideră un polinom cu coeficienți numere reale, $a_n X^n + a_{n-1} X^{n-1} + \dots + a_1 X + a_0$, specificat prin tabloul coeficienților $a[0..n]$. Demonstrați (prin stabilirea precondiției, postcondiției, identificarea unui invariant și verificarea proprietăților acestuia) că algoritmul de mai jos determină valoarea polinomului pentru un argument x .

```

alg(real  $a[0..n]$ , real  $x$ )


---


integer  $i$ ; real  $v$ 
1:  $i \leftarrow n$ ;  $v \leftarrow a[n]$ 
2: while  $i \geq 0$  do
3:    $v \leftarrow v * x + a[i]$ 
4:    $i \leftarrow i - 1$ 
5: end while
6: return  $v$ 

```

Indicație. Invariant: $v = \sum_{j=i}^n a[j] x^{j-i}$

Problema 13 Să se demonstreze că algoritmi **cmmdc1** și **cmmdc2** descriși în continuare determină cel mai mare divizor comun al numerelor nenule a și b .

cmmdc1 (integer a, b)	cmmdc2 (integer a, b)
1: while $a! = 0$ and $b! = 0$ do	1: while $a! = b$ do
2: $a \leftarrow a \text{ MOD } b$	2: if $a > b$ then
3: if $a! = 0$ then	3: $a \leftarrow a - b$
4: $b \leftarrow b \text{ MOD } a$	4: else
5: end if	5: $b \leftarrow b - a$
6: end while	6: end if
7: if $a = 0$ then	7: end while
8: $d \leftarrow a$	8: $d \leftarrow a$
9: else	9: return d
10: $d \leftarrow b$	
11: end if	
12: return d	

Indicație. Dacă notăm cu a_0 și b_0 valorile inițiale ale variabilelor a respectiv b atunci condițiile sunt $P = \{a = a_0, b = b_0\}$ iar postcondiția este $Q = \{d = \text{cmmdc}(a_0, b_0)\}$. Pentru ambele variante de algoritmi proprietatea invariantă este $\text{cmmdc}(a, b) = \text{cmmdc}(a_0, b_0)$.

Demonstrarea proprietății de invarianță se bazează pe proprietăți cunoscute ale celui mai mare divizor comun și anume, $\text{cmmdc}(a, b) = \text{cmmdc}(a \text{ MOD } b, b) = \text{cmmdc}(a, b \text{ MOD } a)$ respectiv $\text{cmmdc}(a, b) = \text{cmmdc}(a - b, b)$ (dacă $a > b$) și $\text{cmmdc}(a, b) = \text{cmmdc}(a, b - a)$ (dacă $a < b$). În ceea ce privește demonstrarea terminării aceasta se bazează pe două șiruri strict descrescătoare de numere naturale: $t(p) = \min(a_p, b_p)$ și $t(p) = |a_p - b_p|$.

Problema 14 (S) Se consideră un tablou $x[1..n]$ care conține valoarea x_0 . Să se demonstreze că: (a) algoritmul `cauta1` determină prima poziție pe care se află valoarea x_0 ; (b) algoritmul `cauta2` determină toate pozițiile pe care se află x_0 în tabloul x .

<code>cauta1</code> ($x[1..n], x_0$)	<code>cauta2</code> ($x[1..n], x_0$)
$i \leftarrow 1$	$i \leftarrow 1$
while $x[i]! = x_0$ do	$m \leftarrow 0$
$i \leftarrow i + 1$	while $i \leq n$ do
end while	if $x[i] == x_0$ then
return i	$m \leftarrow m + 1$
	$\text{poz}[m] = i$
	end if
	$i \leftarrow i + 1$
	end while
	return $\text{poz}[1..m]$

Indicație. Pentru algoritmul `cauta1` se poate folosi ca invariant relația $\{x[j] \neq x_0, j = \overline{1, i-1}\}$ iar pentru algoritmul `cauta2` $\{\text{poz}[k] = x_{i_k}, k = \overline{1, m}\}$ (unde $\{i_k, k = \overline{1, m}\}$ reprezintă pozițiile din tablou pe care se află valoarea căutată, x_0).

Probleme suplimentare.

1. Se consideră un hol pe care sunt n uși numerotate de la 1 la n . La început toate ușile sunt închise. Se trece pe hol de n ori de la prima ușă către ultima. La prima parcurgere se deschid toate ușile. La a doua parcurgere se închid ușile numerotate cu valori pare. La a treia parcurgere se schimbă starea ușilor numerotate cu valori care sunt multiplu de 3 s.a.m.d. (la trecerea cu numărul i se schimbă starea ușilor care au număr un multiplu de i). A schimba starea unei uși înseamnă a o deschide dacă este închisă și a o închide dacă este deschisă. Să se stabilească starea ușii i după n treceri.

Indicație. Numărul de schimbări ale stării ușii cu numărul i depinde de numărul de divizori ai lui i . Dacă i are un număr impar de divizori atunci ușa va fi deschisă, iar dacă i are un număr par de divizori

atunci ușa i va fi închisă. Rămâne de stabilit care dintre valorile cuprinse între 1 și n au un număr par de divizori și care au un număr impar de divizori.

2. Se consideră o scară cu n trepte. Să se stabilească numărul de moduri în care poate fi urcată scara efectuând pași de 1 sau 2 trepte.

Indicație. Dacă $n = 1$ atunci există un singur mod. Dacă $n = 2$ atunci există două moduri (se fac doi pași de câte o treaptă sau un pas de două trepte). Dacă $n = 3$ atunci sunt 3 variante: $(1,1,1)$, $(1,2)$ și $(2,1)$. Fie $K(n)$ numărul de variante de a urca scara. Ultimul pas efectuat poate fi de o treaptă (în acest caz există $K(n-1)$ variante pentru a se ajunge la treapta $n-1$) sau de două trepte (în acest caz există $K(n-2)$ variante pentru a se ajunge la treapta $n-2$). Prin urmare $K(n) = K(n-1) + K(n-2)$ iar $K(1) = 2$, $K(2) = 2$.