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

- Formulare precondiţii şi postcondiţii
- Identificare proprietăţi invariante
- Demonstrarea corectitudinii

Problema 1 Demonstraţi că algoritmul `identic6` returnează $True$ dacă toate elementele din $x[1..n]$ sunt identice şi $False$ 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
```

Indicaţie. Se stabileşte starea algoritmului înainte de intrarea în ciclul `while` (precondiţia) şi postcondiţia specificată în enunţ. Se arată că afirmaţia "dacă $x[1..i]$ are toate elementele identice atunci rez este $True$ altfel rez este $False$" satisface cerinţele specifice unui invariant.

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

Indicaţie. Se calculează $a^{|p|} = \prod_{i=1}^{|p|} a$ şi 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` şi se identifică o proprietate invariantă.

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

`conversie_10_q`(**integer** n, q)	`conversie_q_10`(**integer** $c[0..k], q$)
integer $c[0..k]$,k, m	**integer** i, n
1: $m = n$	1: $i = k$
2: $k = 0$	2: $n = c[i]$
3: $c[k] = m$ MOD q	3: **while** $i > 0$ **do**
4: $m = m$ DIV q	4: $i = i - 1$
5: **while** $m > 0$ **do**	5: $n = n * q + c[i]$
6: $k = k + 1$	6: **end while**
7: $c[k] = m$MODq	7: **return** n
8: $m = m$DIVq	
9: **end while**	
10: **return** $c[0..k]$	

Rezolvare. In cazul primului algoritm, k reprezintă numărul de cifre în reprezentarea în baza q a numărului n iar tabloul $c[0..k]$ conţine cifrele reprezentării începând cu cea mai puţin semnificativă. Cu aceste notaţii, precondiţiile sunt $P = \{n \in \mathbb{N}, q \in \mathbb{N}, q \geq 2\}$, iar postcondiţia este $n = c[k]q^k + c[k-1]q^{k-1} + \ldots + c[1]q + c[0]$.

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

După execuţia liniilor 7 şi 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 postcondiţia. Intr-adevăr, la ieşirea din **while**, valoarea lui m este 0 deci proprietatea devine: $n = \sum_{i=0}^{k} c[i] \cdot q^i$ adică tocmai postcondiţia.

In ceea ce priveşte finitudinea, se observă uşor că funcţia $t(p) = m_p$ (m_p este valoarea variabilei m la a p-a execuţie a ciclului) satisface proprietăţile unei funcţii 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 $\sum_{i=0}^{k} c[i]q^i$. Precondiţia se rezumă la $P = \{k > 0\}$ iar postcondiţia este $Q = \{n = \sum_{i=0}^{k} c[i]q^i\}$. După execuţia 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. Intrucât la intrarea în ciclu este adevărată rămâne să arătăm că rămâne adevărată după execuţia corpului ciclului şi că la final implică postcondiţia.

După execuţia 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 execuţia liniei 5 valoarea variabilei n se modifică astfel că devine din nou adevărată relaţia $n = \sum_{j=i}^{k} c[j]q^{j-i}$. La ieşirea din ciclu variabila i are valoarea 0, astfel că se obţine $n = \sum_{j=0}^{k} c[j]q^j$ adică postcondiţia. Finitudinea rezultă imediat remarcând faptul că $t(p) = i_p$ are proprietăţile unei funcţii de terminare.

Problema 4 Demonstraţi punând în evidenţă un invariant şi o funcţie de terminare că după execuţia algoritmului de mai jos variabila n va conţine valoarea în baza 10 corespunzătoare şirului binar $b[0..k]$ ($b[i] \in \{0, 1\}$, $i = \overline{0, k}$).

```
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
```

Indicaţie. 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 funcţie de terminare.

Problema 5 Să se demonstreze corectitudinea algoritmului de determinare a valorii obţinute prin inversarea ordinii cifrelor unui număr natural. Considerând că $n = c_k c_{k-1} \ldots c_1 c_0$ este numărul iniţial, valoarea căutată este $m = c_0 c_1 \ldots c_k$. Prin urmare precondiţia este: $n = \sum_{i=0}^{k} c_i 10^i$ iar postcondiţia 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 uşurează demonstrarea corectitudinii. Să considerăm relaţiile: $\{n = \sum_{i=p}^{k} c[i]10^{i-p},\ m = \sum_{i=0}^{p-1} c[i]10^{p-1-i}\}$. Se observă că precondiţia implică prima relaţie (pentru $p = 0$) iar a doua este evident adevărată pentru $m = 0$ şi $p = 0$ (m este descrisă în acest caz ca o sumă vidă). După execuţia liniei 4 relaţiile 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 execuţia liniei 5, a doua relaţie 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 execuţia liniei 6 obţinem 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 postcondiţia.

Finitudinea este asigurată de faptul că oricare dintre funcţiile $t(p) = n_p$ sau $t(p) = k + 1 - p$ satisface proprietăţile unei funcţii 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
```

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

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

Finitudinea se demonstrează folosind ca funcţie 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 = 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]
```

Indicaţie. Fie $x_0[1..n]$ conţinutul iniţial al tabloului. Cu această notaţie precondiţia poate fi specificată prin $P = \{x[i] = x_0[i], i = \overline{1,n}\}$ iar postcondiţia prin $Q = \{x[i] = x_0[n + 1 - i], i = \overline{1,n}\}$. Se poate demonstra că următoarele relaţii $\{i \leq n + 1 - i$, $x[j] = x_0[n + 1 - j]$, $x[n + 1 - j] = x_0[j]$, $j = \overline{1,i}\}$ sunt invariante în raport cu ciclul **while** iar la ieşirea din ciclu ($i = \lfloor (n + 1)/2 \rfloor$) implică postcondiţia.

Problema 8 Identificaţi proprietăţi invariante pentru prelucrările repetitive din algoritmii `alg1` şi `alg2` descrişi în continuare şi stabiliţi ce returnează fiecare dintre ei când este apelat pentru valori naturale nenule.

```
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
```

Indicaţie. Notăm cu p contorul prelucrării iterative (p va avea valoarea 0 înainte de intrarea în ciclu şi este incrementat cu 1 la fiecare execuţie a ciclului). Folosind această notaţie (chiar dacă p nu este variabilă explicită în cadrul algoritmului) se observă că pentru `alg1` relaţiile $\{y = b - p$, $z = px$, $x = a\}$ sunt invariante în raport cu prelucrarea repetitivă. La ieşirea din ciclul **while** variabila y are valoarea 0, prin urmare $p = b$ iar $z = px = ba$. Deci algoritmul `alg1` returnează produsul ab. In cazul algoritmului `alg2` relaţiile $\{x = a - pb$, $y = b$, $z = p\}$ sunt invariante. La ieşirea din ciclu $a = z \cdot b + x$ şi $x < y = b$. Aceasta înseamnă că z va conţine câtul împărţirii lui a la b, iar x restul aceleiaşi împărţiri.

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

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

Indicaţie. 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 ieşirea din ciclu implică $c_{0..(n+1)} = a_{0..n} + b_{0..n}$ adică tabloul $c[0..n+1]$ conţine suma valorilor corespunzătoare reprezentărilor binare din $a[0..n]$ şi $b[0..n]$.

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

```
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]
```

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

Problema 11 Propuneţi un algoritm care transformă un tablou $a[1..n]$ prin interschimbări de elemente vecine astfel încât valoarea minimă ajunge pe prima poziţie a tabloului. Demonstraţi corectitudinea algoritmului identificând un invariant şi o funcţie de terminare.

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

Problema 12 Să se demonstreze că algoritmii `cmmdc1` şi `cmmdc2` descrişi în continuare determină cel mai mare divizor comun al numerelor nenule a şi b.

```
cmmdc1 (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
```

```
cmmdc2 (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
```

Indicaţie. Dacă notăm cu a_0 şi b_0 valorile iniţiale ale variabilelor a respectiv b atunci precondiţiile sunt $P = \{a = a_0, b = b_0\}$ iar postcondiţia este $Q = \{d = \text{cmmdc}(a_0, b_0)\}$. Pentru ambele variante de algoritm proprietatea invariantă este cmmdc(a, b)=cmmdc(a_0, b_0).

Demonstrarea proprietăţii de invarianţă se bazează pe proprietăţi cunoscute ale celui mai mare divizor comun şi anume, cmmdc(a, b)=cmmdc$(a$ MOD $b, b)$=cmmdc$(a, b$ MOD $a)$ respectiv cmmdc(a, b)=cmmdc$(a - b, b)$ (dacă $a > b$) şi cmmdc(a, b)=cmmdc$(a, b - a)$ (dacă $a < b$). In ceea ce priveşte demonstrarea terminării aceasta se bazează pe doua şiruri strict descrescătoare de numere naturale: $t(p) = \min(a_p, b_p)$ şi $t(p) = |a_p - b_p|$.

Problema 13 (S) Se consideră un tablou $x[1..n]$ care conţine valoarea x_0. Să se demonstreze că: (a) algoritmul `cauta1` determină prima poziţie pe care se află valoarea x_0;(b) algoritmul `cauta2` determină toate poziţiile pe care se află x_0 în tabloul x.

`cauta1` ($x[1..n]$,$x0$)	`cauta2` ($x[1..n]$,$x0$)
$i = 1$	$i = 1$
while $x[i]! = x0$ **do**	$m = 0$
$i = i + 1$	**while** $i \leq n$ **do**
end while	**if** $x[i] == x0$ **then**
return i	$m = m + 1$
	$poz[m] = i$
	end if
	$i = i + 1$
	end while
	return $poz[1..m]$

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

Probleme suplimentare.

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

   Indicaţie. Numărul de scimbări ale stării uşii cu numărul i depinde de numărul de divizori ai lui i. Dacaă i are un număr impar de divizori atunci uşa va fi deschisă, iar dacă i are un număr par de divizori atunci uşa i va fi închisă. Rămâne de stabilit care dintre valorile cuprinse între 1 şi n au un număr par de divizori şi 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 paşi de 1 sau 2 trepte.

   Indicaţie. Dacă $n = 1$ atunci există un singur mod. Dacă $n = 2$ atunci există două moduri (se fac doi paşi de câte o treaptă sau un pas de două trepte). Dacă $n = 3$ atunci sunt 3 variante: (1,1,1), (1,2) şi (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 $K(1) = 2$, $K(2) = 2$.