Fangli Andrei
grupa 231
14 aprilie 2014

A.23 Calculați diferența a două mulțimi.

Completarea enunțului

O mulțime este o secvență liniară de numere reale. Să se dea o funcție care preia două mulțimi și returnează diferența dintre prima mulțime și cea de a doua.

Specificare

Diferența(prima, aDoua, rezultat)

- Date de intrare:
  - prima: o secvență de numere reale, deține un câmp denumit Lungime egal cu numărul total de elemente conținute.
  - aDoua: o secvență de numere reale, deține un câmp denumit Lungime egal cu numărul total de elemente conținute.
- Date de ieșire:
  - rezultat: o secvență de numere reale, deține un câmp denumit Lungime egal cu numărul total de elemente conținute. Secvența este dată de mulțimea $\{prima_i \mid i = 1..\ prima.Lungime$ și $prima_i \neq aDoua_j, j = 1..aDoua.Lungime\}$ iar rezultat.Lungime este egal cu valoarea cardinalului mulțimii respective.

Dezvoltare din specificații

Specificare

Φ: A = $(a)_{A.Lungime} \in \mathbb{R}$ ^ B = $(b)_{B.Lungime} \in \mathbb{R}$

Ψ: C = A / B, unde A / B = $\{(c)_n \leq$ A.Lungime, oricare ar fi i ∈ **N** < A.Lungime, exista $c_i = a_j$,
j ∈ **N** < A.Lungime si nu exista $c_i = b_j$, ∈ < B.Lungime, n = (i maxim + 1)

Program 0

[A = $(a)_{A.Lungime} \in \mathbb{R}$ ^ B = $(b)_{B.Lungime} \in \mathbb{R}$; C= A/B]

Program 1 (compunere)

[A = $(a)_{A.Lungime} \in \mathbb{R}$ ^ B = $(b)_{B.Lungime} \in \mathbb{R}$; (C U D) / S = A/B]
[(C U D) / S = A/B; C= A/B]

Program 2 (atribuire)

(C.Lungime, D, S) ← (0, A, B)
[(C U D) / S = A/B; C= A/B]

Program 3

(C.Lungime, D, S) ← (0, A, B)
[(C U D) / S = A/B; ((C U D) / S = A/B) ^ (D.Lungime = 0)]

Program 4 (iteratie)

(C.Lungime, D, S) ← (0, A, B)
while D.Lungime ≠ 0 do
    [D.Lungime ≠ 0; (C U D) / S = A / B]
end

Program 5 (compunere)

(C.Lungime, D, S) ← (0, A, B)
while D.Lungime ≠ 0 do
    [D.Lungime ≠ 0; index = 0]
    [index = 0; (C U D) / S = A / B]
end

Program 6 (compunere)

(C.Lungime, D, S) ← (0, A, B)
while D.Lungime ≠ 0 do
    [D.Lungime ≠ 0; index = 0]
    [index = 0; (index = 0) ^ (index = B.Lungime) ^ (b_{index} ≠ $a_{(D.Lungime\ -1)}$)]
    [(index = 0) ^ (index = B.Lungime) ^ (b_{index} ≠ $a_{(D.Lungime\ -1)}$); (C U D) / S = A / B]
end

Program 7 (atribuire)

(C.Lungime, D, S) ← (0, A, B)
while D.Lungime ≠ 0 do
    index = 0
    [index = 0; (index = 0) ^ (index = B.Lungime) ^ (b_{index} ≠ $a_{(D.Lungime\ -1)}$)]
    [(index = 0) ^ (index = B.Lungime) ^ (b_{index} ≠ $a_{(D.Lungime\ -1)}$); (C U D) / S = A / B]
end

Program 8 (iteratie)

(C.Lungime, D, S) ← (0, A, B)
while D.Lungime ≠ 0 do
    index = 0
    while ¬((index = B.Lungime) ^ (b_{index} ≠ $a_{(D.Lungime\ -1)}$) do
        index = index + 1
    end
    [(index = 0) ^ (index = B.Lungime) ^ (b_{index} ≠ $a_{(D.Lungime\ -1)}$); (C U D) / S = A / B]
end

Program 9 (alternanta)

(C.Lungime, D, S) ← (0, A, B)
while D.Lungime ≠ 0 do
    index = 0
    while ¬((index = B.Lungime) ^ (b_{index} ≠ $a_{(D.Lungime\ -1)}$) do
        index = index + 1
    end
    if index = B.Lungime then
        $C_{C.Lungime}$ ← $a_{(D.Lungime\ -\ 1)}$
        C.Lungime ← C.Lungime + 1
        D.Lungime ← D.Lungime – 1
    else
        D.Lungime ← D.Lungime -1
    end
end

Demonstrarea corectitudinii

Functia u = A.Lungime * D.Lungime – index

Predicatul invariant: (C ∩ B = Ø) ^ (C ⊂ A)

B.16 Determinați cel mai lung subșir de rădăcini ale unui polinom.

Completarea enunțului

Șirul de numere este format din numere reale iar polinomul este un șir de numere reale care conține coeficienții polinomului în ordine descrescătoare după grad. De exemplu polinomul $3x^3 - x$ are șirul de coeficienții (3, 0, -1, 0).

Specificare

Subșir(șir, polinom, rezultat)

- Date de intrare:
  - șir: un șir de numere reale, deține un câmp denumit Lungime egal cu numărul total de elemente conținute.
  - polinom: un șir de numere reale, deține un câmp denumit Lungime egal cu numărul total de elemente conținute.
- Date de ieșire:
  - rezultat: un șir de numere reale, deține un câmp denumit Lungime egal cu numărul total de elemente conținute.

    rezultat = ($șir_{i+p}$, $șir_{i+p+1}$, …, $șir_{i+p+n}$), 0 ≤ p < șir.Lungime, 0 ≤ n ≤ șir.Lungime – i,

    1 ≤ i ≤ șir.Lungime, n maxim,

$$polinom_{polinom.Lungime} + \sum_{k=1}^{polinom.Lungime-1} \text{polinom}_k \text{șir}_j^{polinom.Lungime-k} = 0$$

    când j = p + 1..p + n.

    rezultat.Lungime = n.

Dezvoltare din specificații

Specificare

Φ: S = $(s)_{S.Lungime} \in \mathbb{R}$; C = $(c)_{C.Lungime} \in \mathbb{R}$

Ψ: R = SirDorit(S, C), unde SirDorit(S, C) = (s_{+p}, s_{+p+1}, …, s_{+p+n}), 0 ≤ p < S.Lungime,

0 ≤ n ≤ S.Lungime – i, 1 ≤ i ≤ S.Lungime, n maxim,

$$C_{C.Lungime} + \sum_{k=1}^{C.Lungime-1} C_k s_j^{C.Lungime-k} = 0$$

când j = p + 1..p + n.

Program 0

[S = $(s)_{S.Lungime} \in \mathbb{R}$; C = $(c)_{C.Lungime} \in \mathbb{R}$; R = SirDorit(S, C)]

Program 1 (compunere)

[S = $(s)_{S.Lungime} \in \mathbb{R}$; C = $(c)_{C.Lungime} \in \mathbb{R}$; R = SirDorit($(s)_{1...index}$, C)]
[R = SirDorit($(s)_{1...index}$, C); R = SirDorit(S, C)]

Program 2 (atribuire)

(index, S.Lungime, T.Lungime) <- (0, 0, 0)
[R = SirDorit($(s)_{1...index}$, C); R = SirDorit(S, C)]

Program 3

(index, S.Lungime, T.Lungime) <- (0, 0, 0)
[R = SirDorit($(s)_{1...index}$, C); R = SirDorit($(s)_{1..index}$, C) ^ (index = S.Lungime)]

Program 4 (iteratie)

(index, S.Lungime, T.Lungime) <- (0, 0, 0)
while index ≠ S.Lungime do
    [index ≠ S.Lungime; R = SirDorit($(s)_{1...index}$, C); R = SirDorit($(s)_{1..index}$, C) ^ (index = S.Lungime)]
end

Program 5 (compunere)

(index, S.Lungime, T.Lungime) <- (0, 0, 0)
while index ≠ S.Lungime do
    [index ≠ S.Lungime ^ R = SirDorit($(s)_{1...index}$, C); suma = 0]
    [suma = 0; R = SirDorit($(s)_{1..index}$, C) ^ (index = S.Lungime)]
end

Program 6 (compunere)

(index, S.Lungime, T.Lungime) <- (0, 0, 0)
while index ≠ S.Lungime do
    [index ≠ S.Lungime ^ R = SirDorit($(s)_{1...index}$, C); suma = 0]
    [suma = 0; coefIndex = 0]
    [coefIndex = 0; R = SirDorit($(s)_{1..index}$, C) ^ (index = S.Lungime)]
end

Program 7 (compunere)

(index, S.Lungime, T.Lungime) <- (0, 0, 0)
while index ≠ S.Lungime do
    [index ≠ S.Lungime ^ R = SirDorit($(s)_{1...index}$, C); suma = 0]
    [suma = 0; coefIndex = 0]
    [coefIndex = 0; (coefIndex =0) ^ (coefIndex = C.Lungime)]
    [(coefIndex = 0) ^ (coefIndex = C.Lungime); R = SirDorit($(s)_{1..index}$, C) ^ (index = S.Lungime)]
end

Program 8 (atribuire)

(index, S.Lungime, T.Lungime) <- (0, 0, 0)
while index ≠ S.Lungime do
    suma <- 0
    [suma = 0; coefIndex = 0]
    [coefIndex = 0; (coefIndex =0) ^ (coefIndex = C.Lungime)]
    [(coefIndex = 0) ^ (coefIndex = C.Lungime); R = SirDorit($(s)_{1..index}$, C) ^ (index = S.Lungime)]
end

Program 9 (atribuire)

(index, S.Lungime, T.Lungime) <- (0, 0, 0)
while index ≠ S.Lungime do
    suma <- 0
    coefIndex <- 0
    [coefIndex = 0; (coefIndex =0) ^ (coefIndex = C.Lungime)]
    [(coefIndex = 0) ^ (coefIndex = C.Lungime); R = SirDorit($(s)_{1..index}$, C) ^ (index = S.Lungime)]
end

Program 10 (iteratie)

(index, S.Lungime, T.Lungime) <- (0, 0, 0)
while index ≠ S.Lungime do
    suma <- 0
    coefIndex <- 0
    while coefIndex = C.Lungime do
        suma <- suma + $c_{coefIndex} * s_{index}^{C.Lungime - coefIndex - 1}$
    end
    [(coefIndex = 0) ^ (coefIndex = C.Lungime); R = SirDorit($(s)_{1..index}$, C) ^ (index = S.Lungime)]
end

Program 11 (alternanta)

(index, S.Lungime, T.Lungime) <- (0, 0, 0)
while index ≠ S.Lungime do
    suma <- 0
    coefIndex <- 0
    while coefIndex = C.Lungime do
        suma <- suma + $c_{coefIndex} * s_{index}^{C.Lungime - coefIndex - 1}$
    end
    if (suma = 0 then
        [(suma = 0) ^ (coefIndex = C.Lungime); R = SirDorit($(s)_{1..index}$, C)]
    else
        [(suma ≠ 0) ^ (coefIndex = C.Lungime); R = SirDorit($(s)_{1..index}$, C)]
    end
end

Program 12 (alternanta)

```
(index, S.Lungime, T.Lungime) <- (0, 0, 0)
while index ≠ S.Lungime do
        suma <- 0
        coefIndex <- 0
        while coefIndex = C.Lungime do
                suma <- suma + c_coefIndex * s_index^(C.Lungime – coefIndex – 1)
        end
        if (suma = 0 then
                TT.Lungime <- sindex
                T.Lungime <- T.Lungime + 1
                if T.Lungime > S.Lungime then
                        S <- T
                end
                index <- index + 1
        else
                T.Lungime = 0
                Index = index + 1
        end
end
```

Demonstrarea corectitudinii

Functia u = index

Predicatul invariant: S = SirDorit($(s)_{i..index}$, C)