10.11.2022

Mihăilă Andra
gr. 224
Subiect 109

Colocviu FLF

A.
```
f([], -1).
f([H|T], S):- f(T, S1),
              aux([H|T], S, S1).

aux([H|T], S, S1):- S1>0,
                    !,
                    S is S1+H.

aux([_|T], S, S1):- S is S1.
```

Explicație: am definit un predicat auxiliar pentru a evita apelul repetat al funcției f(T, S).

Mihăilă Andra
gr. 224
S 109
~~f~~

B. progresie$(l_1, l_2, \dots, l_n) =$

$$\begin{cases} \text{adevărat}, & n = 2 \\ \text{fals}, & n > 2 \text{ și } abs(l_1 - l_2) \mathrel{!=} abs(l_2 - l_3) \\ \text{progresie}(l_2, l_3, \dots, l_n), & \text{altfel} \end{cases}$$

progresie : l - list -> lista pe care o verificăm
model de flux: (i) - determinist.

Cod PROLOG:

```
progresie([_, _]):-!.
progresie([L1, L2, L3 | T]):-
            D1 is abs(L1-L2),
            D2 is abs(L2-L3),
            D1 =:= D2,
            progresie([L2, L3 | T]).
```

Mihăilă Andra
gr 224
S 109
[illegible]

$$impare(l_1, l_2, ..., l_n) = \begin{cases} adevărat, & n = 0 \\ fals, & n > 0 \text{ și } l_1 \% 2 = 0 \\ impare(l_2, ..., l_n), & altfel \end{cases}$$

impare: l - list -> lista pe care o verificăm
model de flux: (i) - determinist.

Cod PROLOG:

```
impare([]) :- !.
impare([H|T]) :-
            H mod 2 =:= 1,
            impare(T).
```

$$conditie(l) = \begin{cases} fals, & pare(l) \% 2 = 1 \text{ sau } impare(l) \% 2 = 0 \\ adevărat, & pare(l) \% 2 = 0 \text{ și } impare(l) \% 2 = 1 \end{cases}$$

conditie: l - list -> lista pe care o verificăm
model de flux: (i) - determinist.

Mihăilă Andra
gr 224
S 109
[illegible]

Cod PROLOG:

```
conditie(L):-
        impare(L),
        progresie(L).
```

combinari ($l_1, l_2, \dots, l_n$, k) =

$$\begin{cases} 1.\ l_1, & \text{dacă } k=1 \\ 2.\ \text{combinari}(l_2, \dots, l_n, k) & \\ 3.\ l_1 + \text{combinari}(l_2, \dots, l_n, k-1), & k>1 \end{cases}$$

combinari: l - list → lista pe care o verificam
k - integer → numărul de numere
c - list → lista căutată

model de flux: (i, i, o) - nedeterminist.
(poate să fie și (i, i, i) - determinist)

Cod PROLOG:

4/10

Mihăilă Andra
gr 224
S 109
[illegible]

```
combinari([H | _], 1, [H]).
combinari([_ | T], K, C) :-
        combinari(T, K, C).

combinari([H | T], K, [H | C]) :-
        K > 1,
        K1 is K - 1,
        combinari(T, K1, C).
```

combinariCuConditie(l, k) =
{ 1. combinari(l, k), dacă conditie(combinari(l, k)) este adevarat

combinariCuConditie:
l - list -> lista pe care o verificam
k - integer -> numarul de elemente

model de flux: (i, i, o) - nedeterminist.
(poate fi, si (i, i, i) sau (i, i, o) ambele deterministe) :)

5/10

Mihăilă Andra
gr 223
S 109
[illegible]

Cod PROLOG:

```
combinariCuConditie(L, K, C):-
        combinari(L, K, C),
        conditie(C).
```

inserează$(l_1, l_2, ..., l_n, el) =$

$$\begin{cases} [el], & n=0 \\ el \oplus l_1 \ldots l_n, & n>0 \text{ și } el <= l_1 \\ l_1 \oplus \text{insereaza}(l_2, ..., l_n, el), & n>0 \text{ și } el > l_1 \end{cases}$$

inserează: l - list -> lista pe care o verificam
e - integer -> elementul de inserat
r - list -> lista după inserare)

model de flux: (i, i, o) - determinist
(poate fi și (i,i,i) sau (i,i,o) ambele deterministe)

cod PROLOG:

Mihăilă Andra
gr 224
S 109
[illegible]

```
insert([], E, [E]) :- !.
insert([H|T], E, R) :-
        E =< H,
        !,
        R = [E, H | T].
in~~sert(T, E, R1~~)
insert([H|T], E, R) :-
        E > H,
        R = [H | R1],
        insert(T, E, R1).
```

$$sortare(l_1, l_2, \dots, l_n) = \begin{cases} [], & n = 0 \\ insert(sortare(l_2 \dots l_n), l_1), & \text{altfel } (n>0) \end{cases}$$

7/10

Mihăilă Andra
gr 224
S 109
~~4~~

sortare : l - list
r - list -> lista sortata

model de flux : (i, o) - det.
(poate fi și (i, i) $\overset{\text{det.}}{\text{~~sau (i, o) ambele det.~~}}$)

```
sortare([], []) :- !.
sortare([H|T], R) :-
        sortare(T, R1),
        insereaza(R1, H, R).
```

main(L, K) = U combinari Cu Conditie(L, K) V)
model de flux :(i, i, o) - det
poate fi și(i, i, i) - det.

```
main(L, K, LC) :-
        sortare(L, L1),
        findall(O1, combinariCuConditie(L1, K, O1), LC)
```

8/10

c. inloc(l, niv) = l + 1, l e atom, număr
l % 2 = 1, niv % 2 = 0

l , l e atom, nr. , niv % 2 = 1

l , l e atom, ~~nr~~, nu e nr.

inlocuire(l_1, niv + 1) U ... U (inlocuire l_n, niv + 1)
altfel

inlocuire : l - list
niv - integer

```
(defun inlocuire (l niv)
    ( cond
      (
        (AND (atom l) ( ~~nub~~ numberp l) (equal 1
              (mod l 2)) (equal 0 (mod niv 2)))
              (+ l 1)
      )
      (
        (AND (atom l) (numberp l) (equal 1
                       (mod niv 2)))
      ) l
(
  (AND (atom l)( not( numberp l)))
    l
)
(T
  (mapcar #'(lambda(x)
              (
                inlocuire x (+niv 1)
              )
        )
        l
      )
    )
  )
)

main (l) = inlocuire (l, 0)
main: l- list
 (defun main
    (inlocuire l 0)
)
```

10/10