FMI, Info, Anul II, 2019-2020
Programare logică

Seminar 5
Unificare şi arborele de execuţie. Forma prenex. Skolemizare.

(S5.1) Folosind algoritmul de unificare, indicaţi dacă următoarea listă de termeni are unificator:

$$U = \{p(x, f(y, z)), p(x, a), p(x, g(h(k(x))))\}$$

(p, f sunt simboluri de funţii de aritate 2, g, h, k sunt simboluri de funcţii de aritate 1, a este simbol de constantă, x, y, z sunt variabile).

Demonstraţie: Lista de termeni nu are unificator. Să observăm că, după o descompunere, ar trebui să unificăm termenii a şi $f(y, z)$ (sau a şi $g(h(k(x)))$, ceea ce este imposibil.

□

(S5.2) Găsiţi răspunsul dat de Prolog la următoarele întrebări:

1. `'Luke' = luke.`
2. `Luke = luke.`
3. `jedi(luke) = luke.`
4. `jedi(luke) = X.`
5. `jedi(luke) = jedi(X).`
6. `father(vader, A) = father(B, luke).`
7. `heroes(han, X, luke) = heroes(Y, ben, X).`
8. `heroes(han, X, luke) = heroes(Y, ben).`
9. `jedi(X) = X.`
10. `characters(hero(luke), villain(vader)) = characters(X, Y).`
11. `characters(hero(luke), X) = characters(X, villain(vader))`

Demonstraţie:

1. `false` (niciuna dintre acestea nu este variabilă, iar constantele sunt diferite). Compară cu: `luke=luke.` (true) şi `'Luke'='Luke'.` (true).
2. `Luke = luke`
3. `false` (semne diferite pentru funcţii).
4. `X = jedi(luke)`
5. `X= luke`

6. `A=luke, B=vader`
7. `false`
8.`heroes(han, X, luke) = heroes(Y, ben)`. Funcţiile sunt diferite, având aritate diferită.
9. Swish prolog: `X = jedi(X)`. (după trace, *execution aborted*)
10. `X = hero(luke), Y = villain(vader)`
11. `false`

□

(S5.3) În această problemă W,R,O,N,G,I,H,T reprezintă nişte cifre distincte de la 0 la 9. Scrieţi un program în Prolog care găseşte valori pentru aceste variabile astfel încât expresia

$$\text{WRONG} + \text{WRONG} = \text{RIGHT}$$

să fie adevărată (unde WRONG şi RIGHT sunt numere cu câte 5 cifre).

De exemplu, dacă W = 1, R = 2, O = 7, N = 3, G = 4, I = 5, H = 6, T = 8 atunci avem
12734 + 12734 = 25468

Demonstraţie:

□

```
is_digit(X) :- member(X,[0,1,2,3,4,5,6,7,8,9]).
all_distinct([]).
all_distinct([Head | Tail]) :- not(member(Head,Tail)), all_distinct(Tail).
find_solution(W,R,O,N,G,I,H,T) :-
    is_digit(W), is_digit(R), is_digit(O), is_digit(N),
    is_digit(G), is_digit(I), is_digit(H), is_digit(T),
    W > 0, R > 0,
    all_distinct([W,R,O,N,G,I,H,T]),
    constraint(W,R,O,N,G,I,H,T).
constraint(W,R,O,N,G,I,H,T) :-
    Num1 is W * (10 ** 4) + R * (10 ** 3) + O * (10 ** 2) + N * 10 + G,
    Num2 is R * (10 ** 4) + I * (10 ** 3) + G * (10 ** 2) + H * 10 + T,
    Num1 + Num1 =:= Num2.
```

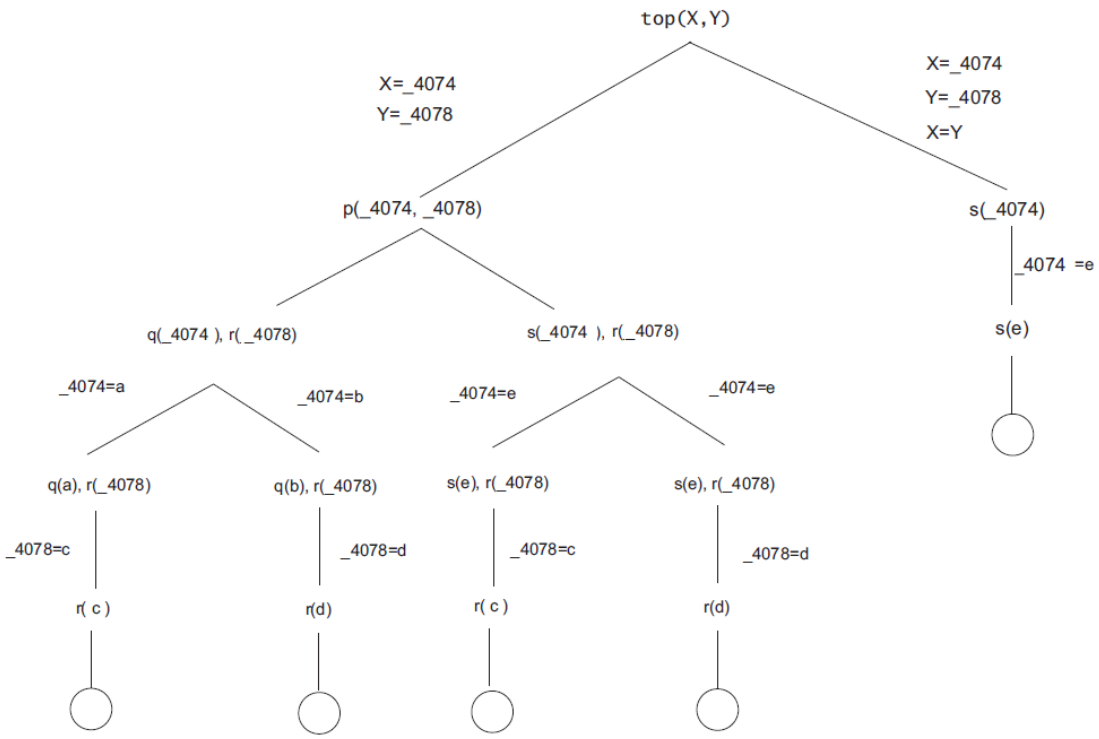
(S5.4) Fie următoarea bază de cunoştinţe:

```
q(a). q(b).  r(c).  r(d). s(e).
top(X,Y) :- p(X,Y).
top(X,X) :- s(X).
p(X,Y) :- q(X), r(Y).
p(X,Y) :- s(X), r(Y).

?- top(X,Y).
```

Desenaţi arborele de execuţie pentru întrebarea `top(X,Y)` indicând răspunsul pentru fiecare ramură.

Demonstraţie:


top(X,Y)
X=_4074
Y=_4078
X=_4074
Y=_4078
X=Y
p(_4074, _4078)
s(_4074)
_4074 =e
q(_4074), r(_4078)
s(_4074), r(_4078)
s(e)
_4074=a
_4074=b
_4074=e
_4074=e
q(a), r(_4078)
q(b), r(_4078)
s(e), r(_4078)
s(e), r(_4078)
_4078=c
_4078=d
_4078=c
_4078=d
r(c)
r(d)
r(c)
r(d)

□

(S5.5) Scrieţi în Prolog un predicat `list_poz(L,R)` care este adevărat dacă `R` este o listă de perechi care conţin elementele din `L` şi poziţia pe care se află în `L`, poziţiile fiind în ordine crescătoare:

`list_poz([a,b,c], [(1,a),(2,b),(3,c)])`

Predicatul trebuie să verifice, dar şi să instanţieze.

Demonstraţie:

□

```
list_poz(L, R) :- list_aux(L, 1, R).

list_aux([ ], _ ,[ ]).
list_aux([H | T], I, [(I,H) | RT]) :- list_aux(T, J, RT), J is I+1.
```

(S5.6) Implementaţi algoritmul *Quicksort* în Prolog.

Demonstraţie:

□

```
quicksort([X | Xs], Ys) :-
  partition(Xs, X, Left, Right),
  quicksort(Left, Ls),
  quicksort(Right, Rs),
  append(Ls, [X | Rs], Ys).
quicksort([ ], [ ]).

partition([X | Xs], Y, [X | Ls], Rs) :-
  X <= Y, partition(Xs, Y, Ls, Rs).
partition([X | Xs], Y, Ls, [X | Rs]) :-
  X > Y, partition(Xs, Y, Ls, Rs).
partition([ ], Y, [ ], [ ]).

append([ ], Ys, Ys).
append([X | Xs], Ys, [X | Zs]) :- append(Xs, Ys, Zs).
```

(S5.7) Scrieţi un predicat `flatten(L,R)` unde `L` este o listă de liste, iar `R` conţine elementele lui `L`, în aceeaşi ordine, dar elimină gruparea în liste:

```
flatten([[1,2], [3,4], [5]], [1,2,3,4,5]).
```

Predicatul trebuie să facă verificare şi să găsescă `R` cu `L` instanţiat.

Demonstraţie:

□

```
myflatten([ ],[ ]).

myflatten([Head | InTail],Out) :-
myflatten(Head,FlatHead),
myflatten(InTail,OutTail),
append(FlatHead,OutTail,Out).

myflatten([Head | InTail], [Head | OutTail]) :-
Head \= [ ],
Head \= [_|_],
        myflatten(InTail,OutTail).
```