FLP šesté cvičení
Prolog: náročnější příklady

Martin Hyrš, Marek Milkovič, Lukáš Zobal

Faculty of Information Technology

Brno University of Technology

Funkcionální a logické programování, 2019/2020

1. úkol: Rozdíl množin

Bez použití vestavěných predikátů vytvořte predikát **`rozdil/3`**, který najde rozdíl dvou množin reprezentovaných seznamy:

```
?- rozdil([a,b,c,d], [c,d,e,f], R).
R=[a,b].
```

Řešení

```
prvek(H, [H|_]) :-
prvek(H, [_|T]) :-

rozdil([], _, []).
rozdil([H|T], S, R) :-
rozdil([H|T], S, [H|P]) :-
```

2. úkol: Problém *n*-dam (*n*-Queens problem)

Vytvořte predikáty `queens/1` a `queens/2`, které určí všechna řešení pro rozestavení N dam na šachovnici $N \times N$ tak, aby se vzájemně neohrožovaly. Predikát `queens/1` počítá úlohu pro šachovnici 8x8. Predikát `queens/2` má navíc parametr určující rozměr šachovnice.

Postup:

1. vhodné kódování řešení
2. generování všech řešení
3. testování

2. úkol: Problém *n*-dam (*n*-Queens problem)

Postup řešení

1. vhodné kódování řešení – seznam *N* čísel, udávají pozici postupně ve sloupcích
2. generování všech potenciálních řešení – vygenerování jednoho a použití predikátu `permutation/2`
3. testování – zda je vygenerované řešení platné

Potřebujeme vygenerovat seznam čísel od jedné do *N*, v libovolném pořadí.

Řešení

```
sequence(0, []) :- !.
sequence(N, [N|T]) :-
```

Poté můžeme všechna řešení generovat predikátem `permutation/2`.

Kostra predikátů pak může vypadat například takto:

Kostra

```
queens(Solution) :- queens(8, Solution).
queens(N, Solution) :-
test(Solution) :- true.     % zatím
```

Musíme zajistit, aby se dámy navzájem neohrožovaly – porovnat všechny navzájem, tzn. každou dámu vůči zbytku.

Řešení

```
test([]) :- !.
test([H|T]) :- ???
```

Musíme zajistit, aby se dámy navzájem neohrožovaly – porovnat všechny navzájem, tzn. každou dámu vůči zbytku.

Řešení

```
test([]) :- !.
test([H|T]) :-
test(_, _, []) :- !.
test(Pos, Dist, [H|T]) :-
```

3. úkol: Cesty koně

Vytvořte predikát **`cesty/7`**, který zjistí počet **acyklických** cest (N), kterými se může šachový kůň dostat z počáteční pozice (XS, YS) do cílové pozice (XE, YE) na šachovnici o zadaných rozměrech (XR, YR):

```
cesty(XR, YR, XS, YS, XE, YE, N)
```

Postup

1. jeden skok
2. jedna cesta
3. celková struktura

3. úkol: Cesty koně (celková struktura – úvod)

```
:- dynamic velikost/2, ...

cesty(XR, YR, XS, YS, XE, YE, N) :-

  assert(velikost(XR, YR)),
  ...,
  findall(...),
  ...,
  retract(velikost(_,_)).
```

3. úkol: Cesty koně (jeden skok)

Testování pozice

```
testPoz(X,Y) :-
```

Pohyb

```
skok(X,Y,XN,YN) :-
skok(X,Y,XN,YN) :-
skok(X,Y,XN,YN) :-
skok(X,Y,XN,YN) :-
skok(X,Y,XN,YN) :-
skok(X,Y,XN,YN) :-
skok(X,Y,XN,YN) :-
skok(X,Y,XN,YN) :-
```

3. úkol: Cesty koně (jedna cesta)

```
cesta(X,Y,X,Y,[X:Y]) :- !.
cesta(X,Y,XE,YE,[X:Y|T]) :-
```

```
cesta(X, Y, XE, YE, _) :-
```

3. úkol: Cesty koně (celková struktura)

```
:- dynamic velikost/2, poz/2.

cesty(XR, YR, XS, YS, XE, YE, N) :-
  assert(velikost(XR, YR)),


  retract(velikost(_,_)).
```

3. úkol: Cesty koně (celková struktura)

```
:- dynamic velikost/2, poz/2.

cesty(XR, YR, XS, YS, XE, YE, N) :-
  XR > 0, YR > 1,
  assert(velikost(XR, YR)),




  retract(velikost(_,_)).
```

3. úkol: Cesty koně (celková struktura)

```
:- dynamic velikost/2, poz/2.

cesty(XR, YR, XS, YS, XE, YE, N) :-
  XR > 0, YR > 1,
  assert(velikost(XR, YR)),




  retract(velikost(_,_)).
```

3. úkol: Cesty koně (celková struktura)

```
:- dynamic velikost/2, poz/2.

cesty(XR, YR, XS, YS, XE, YE, N) :-
  XR > 0, YR > 1,
  assert(velikost(XR, YR)),



  retractall(poz(_,_)),
  retract(velikost(_,_)).
```

4. úkol: Asociativní paměť

Vytvořte predikát **`slovnik/3`**, který bude implementovat funkcionalitu asociativní paměti. Podle typu argumentu bude položky vyhledávat, vkládat a modifikovat.

- `slovnik(s, klic, hodnota)` – vložení/modifikace
- `slovnik(s, klic, Hodnota)` – vyhledání hodnoty
- `slovnik(s, Klic, hodnota)` – vyhledání klíčů

Testování typu termu

`var(V)` – uspěje, je-li **`V`** volná proměnná

4. úkol: Asociativní paměť (cont.)

Řešení

```
% kontroly (co může být volná proměnná?)
slovnik(D, _, _) :-
slovnik(_, K, V) :-
% vyhledani hodnoty
slovnik(D, K, V) :-
% vyhledani klicu
slovnik(D, K, V) :-

% modifikace
slovnik(D, K, V) :-


% vlozeni
slovnik(D, K, V) :-
```

- BFS – prohledávání do šířky (Breadth-first search)
- DFS – prohledávání do hloubky (Depth-first search)
- DLS – prohledávání do hloubky s omezením (Depth-limited search)
- IDS – iterativní prohledávání do hloubky (Iterative deepening search)

backtracking ??
mají tyto metody nějaké problémy ??

\+ zbytečné chyby

- testovací výpisy
- formální náležitosti (jméno souboru...)

je to jen projekt za 8 bodů

dřívější odevzdání: teoreticky ano