Olomoucký informatický korespondenční seminář – Olinx

Programovací jazyk PROLOG, 3. díl

Miroslav Kolařík

olinx.inf.upol.cz

Úvod

V tomto dílu se budeme zabývat problémem osmi dam. Demonstrujeme si na něm, že k danému problému lze přistupovat různými způsoby, třeba změnou reprezentace tohoto problému, kde se často zavádí redundance, která ulehčí výpočet, důsledkem čehož se ušetří čas. Uvidíme také, že klíčovým krokem k řešení je často zobecnění výchozího problému. Paradoxně, pokud vezmeme v úvahu obecnější problém, řešení lze pak jednodušeji formulovat.

Problém osmi dam

Problém osmi dam je šachový kombinatorický problém (uveřejněný v roce 1848), jehož cílem je umístit osm dam na šachovnici takovým způsobem, aby se podle pravidel šachu navzájem neohrožovaly. Je tedy třeba vybrat na šachovnici osm polí tak, aby žádná dvě nebyla na stejném řadku, ve stejném sloupci a ani v jedné ze dvou diagonálních linií. Zobecněním tohoto problému je problém `N` dam (uveřejněn v roce 1850), jehož cílem je rozmístit `N` dam na šachovnici velikosti `N`×`N` tak, aby se vzájemně neohrožovaly. Zajímavou otázkou je, jak vypadají všechna možná taková rozmístění a jaký je jejich počet.

Podívejme se pro zajímavost na počet řešení (`P`) pro šachovnice do velikosti `N`×`N` polí, kde $1 \leq \mathtt{N} \leq 14$.

N	1	2	3	4	5	6	7	8	9	10	11	12	13	14
P	1	0	0	2	10	4	40	92	352	724	2680	14200	73712	365596

Problém osmi dam byl poprvé zveřejněn v berlínském časopise Schachzeitung v roce 1848. Autorem článku byl Max Bezzel. V dalších letech se problému věnovalo mnoho slavných matematiků včetně Karla Friedricha Gausse. Zobecnění problému na `N` dam navrhl v roce 1850 Franz Nauck, který také správně stanovil počet všech řešení původního problému s osmi dámami.
V dnešní době není znám rychlý[a] algoritmus, který by problém dokázal řešit i pro obrovské šachovnice, řekněme o rozměrech 1000 × 1000 polí. Zde jsou výpočty natolik komplikované, že je ani dnešní počítače nezvládají. V souvislosti s tím nabízí Americký institut Clay Mathematics Institute jeden milion dolarů tomu, kdo dokáže napsat rychlý algoritmus, nebo dokáže, že takový algoritmus neexistuje.[b]

[a] Pojem rychlý algoritmus zde nebudeme přesně definovat.
[b] Anglicky více o tom třeba zde: https://www.alphr.com/artificial-intelligence/1006866/win-million-eight-queens-puzzle.

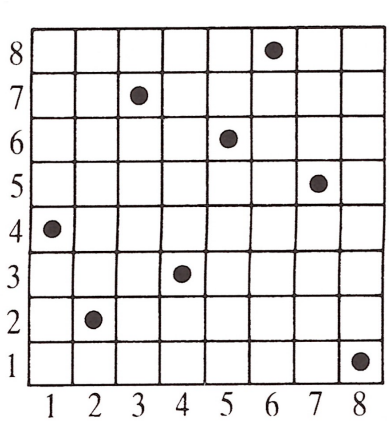
Podívejme se, jak lze problém osmi dam řešit v PROLOGu. Řešení bude naprogramováno jako unární relace

```
solution(Pos),
```

která je pravdivá, pokud `Pos` reprezentuje pozici s osmi dámami, které se navzájem neohrožují. Jistě bude zajímavé porovnávat různé nápady pro programování tohoto problému z pohledu odlišných reprezentací, a proto si dále uvedeme tři různé programy.

Program 1

Pozice dam na šachovnici budeme reprezentovat seznamem osmi položek odpovídajících poloze každé dámy. Konkrétní pozici na šachovnici bude určovat dvojice souřadnic (`X` a `Y`), což budou přirozená čísla menší nebo rovna osmi. Takovou dvojici budeme v programu zapisovat `X/Y`. Na obrázku 1 je zobrazeno jedno z možných řešení a jeho odpovídající reprezentace seznamem.



Obrázek 1: Řešení problému osmi dam. Pozice dam je specifikována seznamem `[1/4,2/2,3/7,4/3,5/6,6/8,7/5,8/1]`.

S touto reprezentací stačí k vyřešení problému najít seznam ve tvaru `[X1/Y1,X2/Y2,X3/Y3,...,X8/Y8]`, který splňuje vzájemné neohrožování dam. Potřebujeme tedy najít konkrétní hodnoty proměnných `X1,Y1,X2,Y2,...,X8,Y8`. Vzhledem k tomu, že všechny dámy musí být v různých sloupcích (aby se vertikálně neohrožovaly) můžeme hledání řešení zjednodušit. Zafixujeme `X`-ové souřadnice, čímž seznam s řešením dostane specifičtější podobu: `[1/Y1,2/Y2,3/Y3,...,8/Y8]`.

Zajímá nás řešení na šachovnici 8 krát 8 polí. Nicméně (při programování) vede mnohokrát cesta ke klíčovému řešení skrze úvahy o obecnějším problému. Paradoxně se často stává, že když uvažujeme o obecnějším řešení, je jeho formulace jednodušší než pro konkrétnější (výchozí) problém. Původní problém je pak snazší vyřešit jako speciální případ obecnějšího problému.

Kreativní částí řešení problému je najít správné zobecnění původního problému. V našem případě je dobrou myšlenkou zobecnit počet dam (počet sloupců v seznamu) z osmi na libovolné nezáporné celé číslo, tedy včetně nuly. Relace `solution` může být formulována užitím dvou případů:

(1) Seznam dam je prázdný. Prázdný seznam je jistě řešením, neboť nula dam se nemůže ohrožovat.
(2) Seznam dam je neprázdný. Pak vypadá takto: `[X/Y|Others]`.

Ve druhém případě je první dáma na pozici `X/Y` a ostatní dámy jsou na pozicích daných seznamem `Others`. Pokud má být tento případ řešením, musí platit následující podmínky:

(1) Dámy v seznamu `Others` se nesmí ohrožovat, tedy `Others` musí být samo o sobě řešením.
(2) `X,Y` náleží do množiny `{1,2,3,4,5,6,7,8}`
(3) Dáma na pozici `X/Y` nesmí ohrožovat žádnou z dam v seznamu `Others`.

K naprogramování první podmínky můžeme jednoduše využít rekurzi. U druhé podmínky se nemusíme zabývat hodnotami pro `X`, neboť `X`-ové souřadnice specifikuje podoba řešení (viz výše). Stačí tedy zařídit, aby `Y` bylo celé číslo mezi 1 a 8. Třetí podmínku můžeme vyřešit novou relací `noattack`. Vše lze v PROLOGu zapsat takto:

```
solution([X/Y|Others]) :-
  solution(Others),
  member(Y,[1,2,3,4,5,6,7,8]),
  noattack(X/Y,Others).
```

Nyní zbývá implementovat relaci `noattack`:

```
noattack(Q,Qlist).
```

I tuto relaci můžeme rozložit na dva případy:

(1) Jestliže je seznam `Qlist` prázdný, pak je relace určitě pravdivá, jelikož žádná dáma nemůže být atakována.
(2) Jestliže je `Qlist` neprázdný, pak je ve tvaru `[Q1,Qlist1]` a musí být splněny dvě podmínky:
   (a) dáma na pozici `Q` nesmí atakovat dámu na `Q1`,
   (b) dáma na `Q` nesmí atakovat žádnou z dam z `Qlist1`.

Je jednoduché určit, zda dáma na nějakém poli neohrožuje jiné pole. Zřejmě tato dvě pole nesmí ležet ve stejném řádku, ve stejném sloupci ani na stejné diagonále (nahoru směřující či dolů směřující). Tvar našeho řešení zaručuje, že všechny dámy budou v různých sloupcích. Zbývá zajistit, aby

- `Y`-ové souřadnice dam byly různé,
- dámy nebyly na stejné diagonále, tedy, aby rozdíl mezi poli ve směru osy `X` byl jiný než rozdíl mezi poli ve směru osy `Y`.

Nyní již vypíšeme celý program.

```
solution([]).

%prvni dama na X/Y, ostatni damy na Others
solution([X/Y|Others]) :-
  solution(Others),
  member(Y,[1,2,3,4,5,6,7,8]),
  %prvni dama neohrozuje ostatni damy
  noattack(X/Y,Others).

%bez ohrozovani
noattack(_,[]).

noattack(X/Y,[X1/Y1|Others]) :-
%dama na X/Y neohrozuje damu na pozici X1/Y1
  %ruzne Y-souradnice
  Y=\=Y1,
  %ruzne diagonaly
  Y1-Y=\=X1-X,
  Y1-Y=\=X-X1,
  %dama na X/Y neohrozuje damy ze seznamu Others
  noattack(X/Y,Others).

%relaci member (byt prvkem seznamu) zname ze 2. dilu
member(Item,[Item|_Rest]).

member(Item,[_First|Rest]) :-
  member(Item,Rest).

%podoba reseni
template([1/Y1,2/Y2,3/Y3,4/Y4,5/Y5,6/Y6,7/Y7,8/Y8]).
```

K programu byla přidána podoba řešení ve tvaru seznamu `template`. Tento seznam můžeme použít při generování řešení jako součást dotazu. Ptáme se takto:

```
?- template(S), solution(S).
```

a program nám generuje následující řešení:

```
S=[1/4,2/2,3/7,4/3,5/6,6/8,7/5,8/1];
S=[1/5,2/2,3/4,4/7,5/3,6/8,7/6,8/1];
S=[1/3,2/5,3/2,4/8,5/6,6/4,7/7,8/1];
...
```

Program 2

V šachovnicové reprezentaci programu 1 mělo každé řešení tvar

```
[1/Y1,2/Y2,3/Y3,4/Y4,5/Y5,6/Y6,7/Y7,8/Y8],
```

přičemž dámy byly jednoduše umístěny do příslušných sloupců. Pokud odstraníme `X`-ové souřadnice, tak žádnou informaci neztratíme. Ponecháním jen `Y`-ových souřadnic dam získáme úspornější reprezentaci pozic na šachovnici:

```
[Y1,Y2,Y3,Y4,Y5,Y6,Y7,Y8].
```

Nutně přitom musí být čísla `Y1`,`Y2`,...,`Y8` permutací množiny {1,2,...,8}, neboť jinak by se dámy horizontálně ohrožovaly. Aby permutace `S` seznamu `[1,2,3,4,5,6,7,8]` byla řešením, musí být dámy v bezpečí (vzájemně neatakovány). Můžeme psát:

```
solution(S) :-
  perm([1,2,3,4,5,6,7,8],S),
  safe(S).
```

Permutacím jsme se věnovali ve druhém dílu, zbývá nám implementovat relaci `safe`. Její definici rozdělíme na dva případy:

(1) `S` je prázdný seznam: to je určitě bezpečné, protože nemůže dojít k atakování.
(2) `S` je neprázdný seznam ve tvaru `[Queen|Others]`. To je bezpečná pozice, je-li seznam `Others` sám o sobě bezpečný a navíc dáma `Queen` neatakuje žádnou dámu ze seznamu `Others`.

V PROLOGu je to:

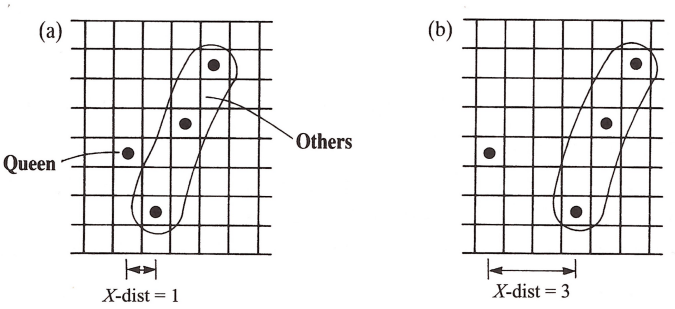
```
safe([]).

safe([Queen|Others]) :-
  safe(Others),
  noattack(Queen,Others).
```

Potřebujeme se vypořádat s relací `noattack` a to bez použití `X`-ových souřadnic, které nyní nemáme explicitně k dispozici. Tento problém můžeme obejít malým zobecněním relace `noattack`. Cílem relace

```
noattack(Queen,Others)
```

je zajistit, aby dáma `Queen` neatakovala dámy v seznamu `Others`, když je `X`-ová vzdálenost mezi `Queen` a `Others` rovna 1, viz obrázek 2. Je potřeba zobecnit právě `X`-ovou vzdálenost mezi `Queen` a `Others`. Proto přidáme vzdálenost `Xdist` jako třetí argument relace `noattack`:

```
noattack(Queen,Others,Xdist)
```



Obrázek 2: (a) `X`-ová vzdálenost mezi `Queen` a `Others` je 1.
(b) `X`-ová vzdálenost mezi `Queen` a `Others` je 3.

V souladu s tím musí být změněna relace `safe` na:

```
noattack(Queen,Others,1)
```

Relaci `noattack` lze zavést podle dvou případů, v závislosti na seznamu `Others`:

- pokud je `Others` prázdný, pak zde určitě nedojde k ohrožování,
- pokud je seznam `Others` neprázdný, pak dáma `Queen` nesmí napadnout první dámu v `Others` (která je `Xdist` sloupců od `Queen`) a také ostatní dámy v seznamu `Others` ve vzdálenosti `Xdist+1`.

Výše uvedené vede k následujícímu PROLOGovskému programu, v němž v unární relaci `solutions(Queens)`, představuje `Queens` seznam `Y`-ových souřadnic osmi neohrožujících se dam.

```
solution(Queens) :-
  perm([1,2,3,4,5,6,7,8],Queens),
  safe(Queens).

%permutace zname ze 2. dilu
perm([],[]).

perm([Head|Tail],PermList) :-
  perm(Tail,PermTail),
  del(Head,PermList,PermTail).

%smazani polozky ze seznamu zname ze 2. dilu
del(Item,[Item|List],List).

del(Item,[First|List],[First|List1]) :-
  del(Item,List,List1).

safe([]).

safe([Queen|Others]) :-
  safe(Others),
  noattack(Queen,Others,1).

noattack(_,[],_).

noattack(Y,[Y1|Ylist],Xdist) :-
  Y1-Y =\= Xdist,
  Y-Y1 =\= Xdist,
  Dist1 is Xdist+1,
  noattack(Y,Ylist,Dist1).
```

Funkčnost programu lze nyní otestovat položením dotazu `?- solution(Queens).`

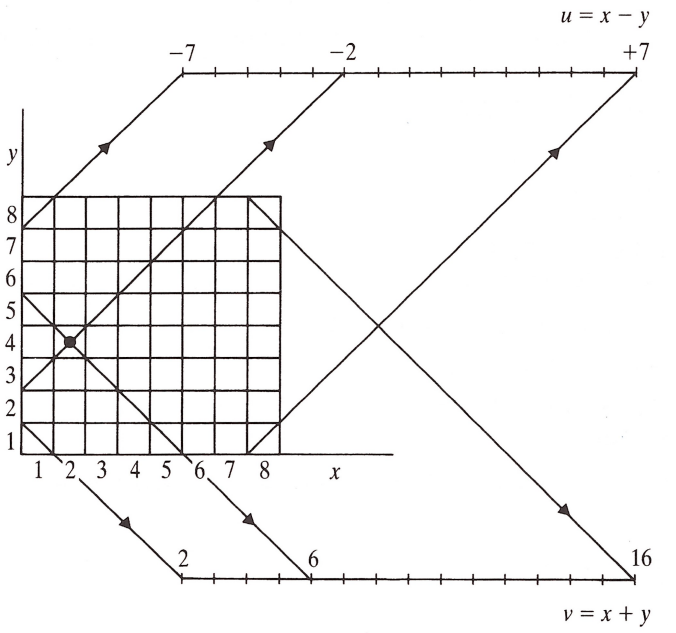
Program 3

Třetí program na řešení problému osmi dam bude založen na následující úvaze. Každá dáma musí být umístěna na nějaké pole, tedy do nějakého sloupce, nějakého řádku, nějaké nahoru směřující diagonály a nějaké dolů směřující diagonály. Aby se dámy vzájemně neohrožovaly, musí být každá z nich umístěna v jiném sloupci, jiném řádku, jiné nahoru směřující diagonále a jiné dolů směřující diagonále. Je tedy přirozené uvažovat širší reprezentaci se čtyřmi souřadnicemi:

`x` ... sloupce
`y` ... řádky
`u` ... nahoru směřující diagonály
`v` ... dolů směřující diagonály.

Souřadnice jsou závislé. Je-li dáno `x` a `y`, pak jsou souřadnice `u` a `v` jednoznačně určeny třeba takto (viz obrázek 3):

```
u=x-y
v=x+y
```



Obrázek 3: Vztahy mezi sloupci, řádky a diagonálami. Označené pole má souřadnice `x=2`, `y=4`, `u=2-4=-2`, `v=2+4=6`.

Domény pro všechny čtyři dimenze jsou

```
Dx=[1,2,3,4,5,6,7,8]
Dy=[1,2,3,4,5,6,7,8]
Du=[-7,-6,-5,-4,-3,-2,-1,0,1,2,3,4,5,6,7]
Dv=[2,3,4,5,6,7,8,9,10,11,12,13,14,15,16]
```

Problém osmi dam lze nyní uchopit takto: vybrat osm uspořádaných čtveřic `(X,Y,U,V)` z domén (`X`∈`Dx`, `Y`∈`Dy`, `U`∈`Du`, `V`∈`Dv`), přičemž se žádný stejný prvek z domény nikdy nevybere dvakrát. Samozřejmě, jsou-li vybrány `X` a `Y`, jsou tím určeny `U` a `V`. Řešení je pak (volně řečeno) následující: vybereme (vzhledem ke všem čtyřem doménám) pozici první dámy, vymažeme odpovídající položky ze všech domén a pak použijeme zbylé hodnoty ve všech čtyřech doménách k umístění ostatních dam. Program následuje.

```
%v seznamu Ylist najdeme Y-souradnice dam
solution(Ylist) :-
  sol(Ylist,
    %domena pro X-ove souradnice
    [1,2,3,4,5,6,7,8],
    %domena pro Y-ove souradnice
    [1,2,3,4,5,6,7,8],
    %domena pro nahoru smerujici diagonaly
    [-7,-6,-5,-4,-3,-2,-1,0,1,2,3,4,5,6,7],
    %domena pro dolu smerujici diagonaly
    [2,3,4,5,6,7,8,9,10,11,12,13,14,15,16]).

sol([],[],_Dy,_Du,_Dv).

sol([Y|Ylist],[X|Dx1],Dy,Du,Dv) :-
  %smazani pouziteho Y
  del(Y,Dy,Dy1),
  %vypocet U a jeho nasledne smazani
  U is X-Y,
  del(U,Du,Du1),
  %vypocet V a jeho nasledne smazani
  V is X+Y,
  del(V,Dv,Dv1),
  %pro hledani reseni pouzij zbyvajici hodnoty
  sol(Ylist,Dx1,Dy1,Du1,Dv1).

%smazani prvku ze seznamu zname ze 2. dilu
del(Item,[Item|List],List).

del(Item,[First|List],[First|List1]) :-
  del(Item,List,List1).
```

Pozice na šachovnici je i tentokrát reprezentována seznamem `Y`-ových souřadnic. Klíčovou relací je v uvedeném programu relace

```
sol(Ylist,Dx,Dy,Du,Dv),
```

která v seznamu `Ylist` konkretizuje `Y`-ové souřadnice dam, předpokládajíce, že jsou postupně umístěny ve sloupcích z `Dx`. Všechny `Y`-ové souřadnice a odpovídající `U` a `V` souřadnice jsou brány ze seznamů `Dy`, `Du` a `Dv`. Relaci `solution` lze vyvolat dotazem

```
?- solution(S).
```

To způsobí vyvolání relace `sol` s úplnými doménami, které odpovídají problémovému prostoru pro osm dam.

Relace `sol` je obecná, ve smyslu, že může být použita pro řešení problému `N` dam (na šachovnici velikosti `N`×`N`). Je pouze nutné správně nastavit domény `Dx`, `Dy`, `Du` a `Dv`. Je proto žádoucí umět vygenerovat vhodný rozsah domén. K tomu potřebujeme relaci

```
gen(N1,N2,List),
```

která pro daná celá čísla `N1` a `N2` vyprodukuje seznam:

```
List=[N1,N1+1,N1+2,...,N2-1,N2].
```

Požadovanou relaci `gen` lze naprogramovat takto:

```
gen(N,N,[N]).

gen(N1,N2,[N1|List]) :-
  N1<N2,
  M is N1+1,
  gen(M,N2,List).
```

Výše uvedenou relaci `solution` lze odpovídajícím způsobem zobecnit na relaci

```
solution(N,S),
```

kde `N` je rozměr šachovnice a `S` je řešení reprezentované jako seznam `Y`-ových souřadnic `N` dam. Následuje zobecněná relace `solution` zapsaná v PROLOGu:

```
solution(N,S) :-
  gen(1,N,Dxy), %Dxy je spolecna domena pro X a Y
  Nu1 is 1-N,
  Nu2 is N-1,
  gen(Nu1,Nu2,Du), %vygenerovani domeny Du
  Nv2 is N+N,
  gen(2,Nv2,Dv),   %vygenerovani domeny Dv
  sol(S,Dxy,Dxy,Du,Dv).
```

Řešení problému dvanácti dam může být nyní vyvoláno dotazem:

```
?- solution(12,S).

S= [1,3,5,8,10,12,6,11,2,7,9,4];
...
```

Závěrem

Při řešení problému osmi dam jsme si ukázali, jak může být jeden problém řešen různými způsoby, přičemž byla měněna reprezentace dat. Někdy byla reprezentace úspornější, občas jasnější a také jsme viděli reprezentaci částečně redundantní. Nevýhodou úspornější reprezentace je to, že některé informace musí být přepočítány vždy, když jsou potřeba.

V několika případech bylo klíčovým krokem k řešení, zobecnění daného problému. Paradoxně, když jsme brali v potaz obecnější problém, bylo jednodušší formulovat řešení. Tento zobecňovací princip je standardně užívanou programátorskou technikou.

Třetí program nejlépe ilustroval, jak lze přistupovat ke zobecňování problému, při omezení struktury danou množinou prvků.

Přirozenou a podstatnou otázkou je, který ze tří uvedených programů je nejefektivnější? V tomto ohledu je program 2 mnohem horší než ostatní dva programy, které jsou si z hlediska výpočetní náročnosti podobné. Důvodem je to, že program 2 je založený na vytváření kompletních permutací, zatímco ostatní dva programy jsou schopny včas rozpoznat a odmítnout nevhodné[1] permutace, i když jsou vytvořeny pouze částečně. Zmiňme ještě, že program 3 se vyhýbá některým aritmetickým výpočtům, které jsou v podstatě zachyceny v redundantní reprezentaci šachovnice, kterou tento program používá.

[1] Máme na mysli permutace (pozice dam), které nemohou být řešením.

Úkoly k odevzdávání

Následují úkoly, které se odevzdávají přes web semináře Olinx,

https://olinx.inf.upol.cz/

Úkoly stačí odevzdat ve formě jednoho jednoduchého textového souboru, ve kterém jsou napsány řešení ke všem úkolům. Pro tvorbu řešení se předpokládá používání aplikace SWI-PROLOG.

Úkol 1 *6 bodů*

Vypište všechna možná řešení problému šesti dam.

Úkol 2 *6 bodů*

Napište, kolik řešení má problém osmi dam, je-li jedna dáma umístěna v jednom ze čtyř rohů, tedy na pozici `1/1`, nebo `1/8`, nebo `8/1`, nebo `8/8`.

Úkol 3 *13 bodů*

Předpokládejme, že jsou pole šachovnice reprezentovány dvojicemi souřadnic ve tvaru `X/Y`, kde `X` i `Y` jsou čísla mezi `1` a `8`. Nyní definujeme relaci `skok(Pole1,Pole2)` podle skoku jezdce na šachovnici. Předpokládejme, že Pole1 je vždy konkrétně zadáno, zatímco Pole2 může být nekonkrétní. Například:

```
?- skok(1/1,S).

S=3/2;
S=2/3;
false.
```

Následuje program v PROLOGu:

```
%jezdec skace z pole X/Y na pole X1/Y1
skok(X/Y,X1/Y1) :-
  %vzdalenosti ve smerech x a y nebo naopak
  (dxy(Dx,Dy); dxy(Dy,Dx)),
  X1 is X+Dx,
  nasachovnici(X1), %X1 je na sachovnici
  Y1 is Y+Dy,
  nasachovnici(Y1). %Y1 je na sachovnici

dxy(2,1).   %2 pole vpravo, 1 dopredu
dxy(2,-1).  %2 pole vpravo, 1 dozadu
dxy(-2,1).  %2 pole vlevo, 1 dopredu
dxy(-2,-1). %2 pole vlevo, 1 dozadu

%souradnice na sachovnici
nasachovnici(Souradnice) :-
  0 < Souradnice,
  Souradnice < 9.
```

V programu se vyskytl zajímavý řádek `(dxy(Dx,Dy); dxy(Dy,Dx))`. V něm má středník (`;`) význam logické spojky nebo. Jezdcův skok tak zahrnuje všech osm možností, přičemž se dále berou v potaz jen ty skoky, které zůstanou na šachovnici.
A nyní již konečně úkoly. Samozřejmě můžete použít binární relaci `skok`.

(a) Definujte relaci `cestajezdce(Cesta)`, kde `Cesta` je seznam polí, které reprezentují korektní cestu jezdce na prázdné šachovnici.
(b) S využitím relace `cestajezdce(Cesta)` napište (v PROLOGu) dotaz pro nalezení cesty délky 4 skoky z pole `2/1` na druhou stranu šachovnice (`Y=8`) tak, aby procházela přes pole `5/4` po druhém skoku.

Literatura

[1] I. Bratko, 2011. *PROLOG Programming for Artificial Intelligence (4th Edition)*. Pearson Education Canada. ISBN 9780321417466.

Katedra informatiky
Univerzita Palackého v Olomouci
17. listopadu 12
771 46 Olomouc