Domény

Petr Štěpánek

S využitím materialu Krysztofa R. Apta

2006

Typy programů v čistém Prologu je možné uspořádat podle různých pohledů. Zajímavá je charakteristika podle domén, které programy ke svému životu potřebují. Nejprve musíme vysvětlit pojem domény.

Definice. (Domény)

Je-li L jazyk nějakého programu, množinu všech základních termů označíme HU_L a nazveme Herbrandovo univerzum.

Množinu všech základních atomů jzyka L označíme HB_L a nazveme Herbrandova báze.

Z praktických důvodů budeme Herbrandovo univerzum nazývat *doménou* jazyka L (případně i *doménou programu*, ke kterému tento jazyk patří.

Prázdná doména

Je nejjednodušší doménou. Jazyk, kterému prázdná doména odpovídá neobsahuje žádné funkční symboly. Proto také nejsou žádné termy bez proměnných a všechny predikáty mají četnost nula. Jsou to vlastně výrokové konstanty.

Nicméně lze napsat legální Prologovské programy, které počítají nad touto doménou.

Příkladem takového programu je program SUMMER.

```
summer.
warm ← summer.
warm ← sunny.
happy ← summer, warm.
```


Programu SUMMER můžeme klást jednoduché dotazy. Protože atomy (v Herbrandově bázi) neobsahují proměnné, vypočtené odpovědní substituce jsou prázdné.

```
?-  happy.
yes
?- sunny.
no
```

Prolog nabízí tři vestavěné predikáty četnosti nula: `true/0`, `fail/0` a `repeat/0`.

Predikát `true/0` je vnitřně definován jedinou klauzulí

```
true.
```

takže dotaz `true` je vždy úspěšný.

`fail/0` má prázdnou definici, proto dotaz `fail` je vždy neúspěšný.

`repeat/0` je vnitřně definován pomocí dvou klauzulí:

```
repeat.
repeat ← repeat.
```

Vestavěné predikáty nelze předefinovat, takže klauzule, jejichž hlavy se shodují s některým vestavěným predikátem jsou ignorovány.

Cvičení.

(i) Nakreslete LD-strom a Prologvský strom pro dotazy `repeat` a `fail.`

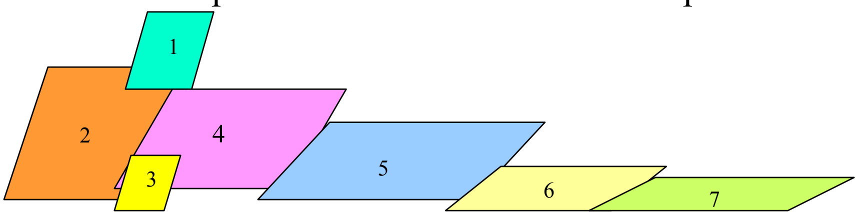
(ii) Příkaz Prologu `write(´a ´)` zobrazí řetězec `a` a příkaz `nl` znamená přechod na nový řádek.

Jaký efekt má dotaz `repeat, write(´a´), nl, fail?`

Konečné domény

Užitečnější jsou již konečné domény. Každý prvek domény odpovídá nějaké konstantě v daném jazyce. Předpokládáme, že takových konstant je konečně mnoho a žádné funkční symboly četnosti větší než nula v jazyce nejsou.

Uvažujme příklad jednoduché databáze, která obsahuje informaci o tom, které země spolu sousedí. Znázorníme to na příkladu Střední Ameriky.



1 Belize
2 Guatemala
3 El Salvador
4 Honduras
5 Nicaragua
6 Costa Rica
7 Panama

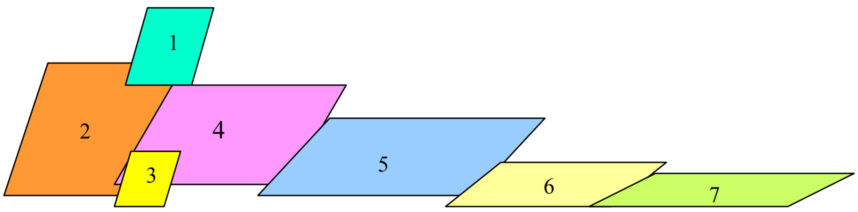
Program *CENTRAL_AMERICA* je jednoduchou databází, která popisuje, které země spolu sousedí.

```
% neighbour(X,Y) ← X sousedí s Y.

neighbour(belize,guatemala).
neighbour(guatemala, belize).
neighbour(guatemala,el_salvador).
neighbour(guatemala,honduras).
neighbour(el_salvador,guatemala).
neighbour(el_salvador,honduras).
neighbour(honduras,guatemala).
neighbour(honduras,el_salvador).
neighbour(honduras,nicaragua).
neighbour(nicaragua,honduras).
neighbour(nicaragua,costa_rica).
neighbour(costa_rica,panama).
neighbour(panama,costa_rica).
```


Program odpovídá na jednoduché otázky: "jsou Honduras a El Salvador sousední státy ?"

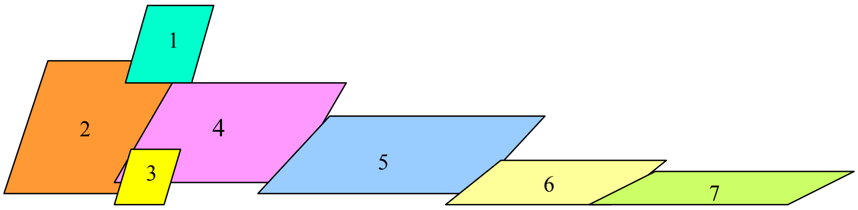
```
?- neighbour(honduras,el_salvador).
yes
```



1 Belize
2 Guatemala
3 El Salvador
4 Honduras
5 Nicaragua
6 Costa Rica
7 Panama

“které státy jsou sousedy Nicaragui ?”

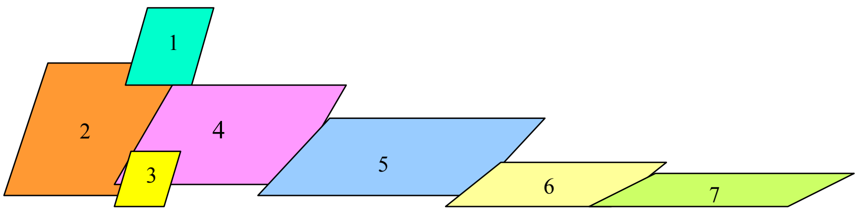
```
?- neighbour(nicaragua,X).
X = honduras ;
X = costa_rica ;
no
```



1 Belize
2 Guatemala
3 El Salvador
4 Honduras
5 Nicaragua
6 Costa Rica
7 Panama

„Které státy sousedí s oběma státy Hondurasem a Costa Ricou ?”

```
?- neighbour(X,honduras), neighbour(X,costa_rica).
X = nicargua ;
no
```



1 Belize
2 Guatemala
3 El Salvador
4 Honduras
5 Nicaragua
6 Costa Rica
7 Panama

Složitější dotazy vyžadují rozšíření programu o další pravidla. Například pro otázku

"které země lze dosáhnout z Guatemaly přejitím jediné země"

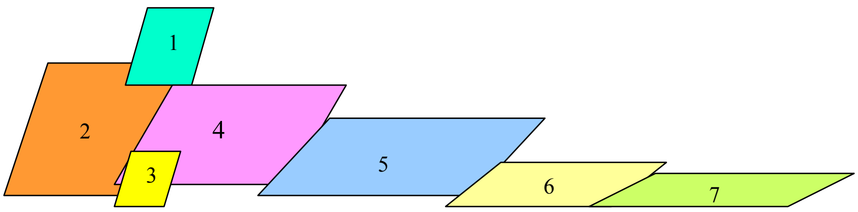
je třeba doplnit program o pravidlo

```
one_crossing(X,Y) ← neighbour(X,Z),
                    neighbour(Z,X), diff(X,Y).
```

Nyní dostáváme


```
?- one_crossing(guatemala,Y).

Y = honduras ;
Y = el_salvador ;
Y = nicaragua ;
no
```



1 Belize
2 Guatemala
3 El Salvador
4 Honduras
5 Nicaragua
6 Costa Rica
7 Panama

Toto pravidlo obsahuje lokální proměnnou `Z`. Proměnnou v klauzuli, která se vyskytuje jen těle klauzule nazýváme *lokální*.

Lokální proměnné se zpravidla používají k přenášení mezivýsledků.

Lokální proměnné nelze zaměňovat s anonymními proměnnými. Kdybychom v uvedené klauzuli zaměnili oba výskyty `Z` anonymními proměnnými, změnili bychom její význam. Každý výskyt anonymní proměnné “_“ by označoval jinou proměnnou.

Zatím co konjunkce je obsažena v definici klauzule, s disjunkcí máme v čistém Prologu větší problém.

Uvažujme dotaz

“které země sousedí s Hondurasem nebo s Costa Ricou ?”

V čistém Prologu je třeba definovat nový predikát pomocí dvou klauzulí, které přidáme k programu.

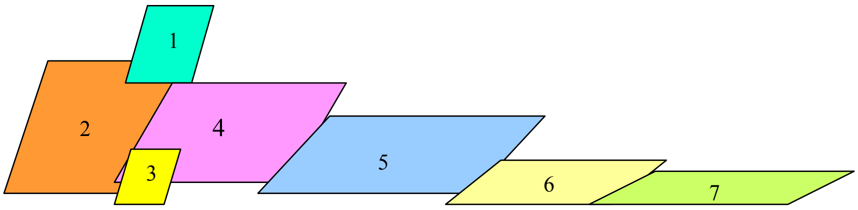
```
neighbour_h_or_c(X) ← neighbour(X,honduras).
neighbour_h_or_c(X) ← neighbour(X,costa_rica).
```

Potom dostáváme

```
?- neighbour_h_or_c(X).

X = guatemala ;
X = el_salvador ;
X = nicaragua ;
X = nicaragua ;
X = panama ;
no
```

V odpovědích je `nicaragua` uvedena dvakrát, protože je sousedem obou zemí.



1 Belize
2 Guatemala
3 El Salvador
4 Honduras
5 Nicaragua
6 Costa Rica
7 Panama

Reprezentace relace `neighbour` v programu *CENTRAL AMERICA* je příliš jednoduchá.

Otázky

"Která země má nejvíce sousedů ?"

"Které země je třeba přejít na cestě ze země `X` do země `Y` ?"

nelze jednoduše zodpovědět.

Vhodnější by byla reprezentace, kde by pro každou zemi byl vytvořen seznam sousedních zemí. K tomuto problému se vrátíme až zavedeme pojem seznamu.

Numerály

Již jsme je použili reprezentaci přirozených čísel v programu *SUMA*.

Formálně se numerály definují indukcí. Používáme jazyk, ve kterém je konstanta 0 (nula) a unární funkční symbol *s* (následník). Podle následující definice vytvořené základní termy nazýváme *numerály*.

- 0 je numerál,
- je-li x numerál, potom $s(x)$, následník x je také numerál.

Tuto definici lze bezprostředně vyjádřit programem *NUMERAL* .

```
% num(X) ← X je numerál.

num(0).
num(s(X)) ← num(X).
```


Snadno se ověří

- pro numerál `s`^{n}`(0)`, a `n` $\geq$ `0` dotaz `num(s`^{n}`(0))` končí úspěšně,
- pro základní term `t` , který není numerálem, dotaz `num(t)` konečně selhává.

Předchozí program je rekurzivní, predikát `num` je v těle i v hlavě druhé klauzule. V obecném případě rekurse připouští možnost nekonečných výpočtů.

Stačí vyměnit pořadí první a druhé klauzule a vytvořit tak program *NUMERAL1*.

Dotaz `num(Y)` s proměnnou `Y` , která se unifikuje s hlavou první (rekurzivní) klauzule `s(X)`, dává `num(X)` jako rezolventu `num(Y)`. Je zřejmé, že opakování tohoto postupu vede na nekonečný výpočet počínající dotazem `num(Y)`.

Naproti tomu původní program *NUMERAL* na dotaz `num(Y)` generuje vypočtenou odpovědní substituci $\{Y/0\}$. Při navracení (hledání dalších odpovědí) postupně generuje odpovědní substituce

$$\{Y/\ s(0)\},\ \{Y/\ s(s(0))\},\ ...\ \text{atd.}$$

Program *NUMERAL* generuje nekonečně mnoho odpovědí, tedy potenciálně diverguje.

Pořadí klauzulí ovlivňuje efektivnost výpočtů. Uvažujeme-li dotaz `num(`s^n`(0))` , pro program *NUMERAL1* , jeho první klauzule se unfikuje a po `n` použitích této klauzule výpočet končí úspěšně využitím druhé klauzule. Při výpočtu bylo provedeno `n+2` pokus; o unifikaci.

Při výpočtu podle programu *NUMERAL* bude provedeno `2n+1` pokusů o unifikaci (z toho `n` neúspěšných s hlavou první klauzule). Pokud $n > 0$ platí $2n+1 \geq n+2$.

Program *SUMA*

S tímto programem pro součet numerálů jsme se již seznámili při výkladu SLD-derivací.

```
% sum(X,Y,Z) ← X,Y,Z  jsou numerály a Z je
                      součet X a Y.
sum(X,0,X).
sum(X,s(Y),s(Z)) ← sum(X,Y,Z).
```

Pro součet dostáváme například

`?- sum(s(s(0)),s(s(s(0))),Z).`

`Z = `s^5`(0)`

Můžeme také generovat rozklady součtu na dvojice sčítanců

```
?- sum(X,Y,s(s(0))).

X = s(s(s(0)))
Y = 0 ;

X = s(s(0))
Y = s(0) ;

atd.
```

Podobně dotaz `sum(s(X),s(Y),`s^5`(0))` dává všechny dvojice `X,Y,` takové, že `s(X)+s(Y) = `s^5`(0)` atd.

Také odpovědi na dotazy nemusí být základní

```
?- sum(X,s(0),Z).

Z = s(X);

no
```


Některé dotazy jako `sum(X,Y,Z)` generují nekonečně mnoho odpovědí.

Protože jsme nezabezpečili, že argumenty relace `sum` mají být termy, jejichž instance jsou numerály, můžeme dostat také odpovědi

```
?- sum(a,0,X).

X = a

...
```

nebo také

```
?- sum([a,b,c],s(0),X).

X = s([a,b,c])
```

Proti takovým případům stačí do první klauzule programu *SUM A* vložit test `num(X)` a upravit ji do tvaru

`sum(X,0,X)` $\leftarrow$ `num(X).`

S podobnými problémy se setkáme i při použití dalších programů pro numerály. Pro násobení můžeme napsat program *MULT*. Násobení je v aritmetice definováno indukcí axiomy

- $x.0 = 0$
- $x \,.\, s(y) = (x \,.\, y) + x$

Můžeme to vyjádřit programem

```
% mult(X,Y,Z) ← X,Y,Z takové, že Z je součinem X a Y .

mult(_,0,0).
mult(X,s(Y),W) ← sum(W,X,Z).
```

V první klauzuli je použita anonymní proměnná (test na numerál zde nemá smysl). Ve druhé klauzuli je lokální proměnná `W` , která přenáší mezivýsledek. Této klauzuli by lépe odpovídal druhý axiom součinu ve tvaru

- $x \,.\, s(y) = w + x$, kde $w = x \,.\, y$.

Program *LESS* .
Relace $<$ ostrého uspořádání numerálů se v aritmetice definuje pomocí dvou axiomů

- $0 < s(x)$,
- je-li $x < y$, potom $s(x) < s(y)$.

Tyto axiomy lze vyjádřit programem

```
% less(X,Y) ← X,Y jsou numerály takové, že X < Y .

less(0,s(_)).
less(s(X),s(Y)) ← less(X,Y).
```

V aritmetice se zaručuje, že relace $<$ je lineární uspořádání axiomem

- $x < y \quad x = y \quad y < x$