FLP čtvrté cvičení
Prolog: úvod

Martin Hyrš, Marek Milkovič, Lukáš Zobal

Faculty of Information Technology

Brno University of Technology

Funkcionální a logické programování, 2019/2020

Osnova cvičení – Prolog

Lukáš Zobal, izobal@fit.vutbr.cz

1. Cvičení 4 (16. 3. / 19. 3. / 23. 3. / 26. 3.)
   - Úvod, syntaxe
   - Seznamy
2. Cvičení 5 (30. 3. / 2. 4. / 6. 4. / 9. 4.)
   - Dynamické predikáty
   - Stavový prostor
3. Cvičení 6 (13. 4. / 16. 4. / 20. 4. / 23. 4.)
   - Náročnější příklady
   - Prohledávání stavového prostoru

- 6 variant (nebo vlastní)
- Registrace: 26. 3. 2020
- Deadline: 26. 4. 2020
- Můžete použít cokoliv z:
  - z přednášek
  - ze cvičení
- **vstup** načtěte jakýmkoliv fungujícím způsobem (`input2.pl`)
- Zbytek musí být vaše vlastní práce

- Prolog
  - první logický jazyk (1972)
  - dodnes populární
- deklarativní ($\times$ imperativní $\times$ funkcionální)

Syntaxe

- Komentáře:
  - `% komentar do konce radku`
  - `/* blokovy komentar */`
- Syntaktické stavební kameny:
  - termy
  - cíle
  - klauzule, predikáty
- Zdrojový program $\approx$ databáze predikátů (záleží na pořadí klauzulí)
- Spuštění programu $\approx$ dotaz (ve formě cíle/cílů)
- Výsledkem je navázání[1] volných proměnných na individua, které odpovídá zadanému dotazu.

[1] V logickém programování tzv. instanciace.

Term:

- jednoduchý:
  - atom: `a`, `ahoj`, `'ahoj'`, `'Ahoj'`, `'a'`, `'@'`
  - číslo: `3`, `3.1415`
  - proměnná: `A`, `Ahoj`, `Ahoj_123`
- složený term: obecně **`funktor(arg1, arg2, ...)`**
  - například: `ahoj(cau(neco),dalsi)`
  - seznam: `[]`, `[ahoj,cau,nazdar]`
  - řetězec: `"ahoj"` ∼ `[97,104,111,106]`

- fakt = predikát bez těla: a(b,c,...).
  - vztahy mezi objekty: a(b,c). – objekty b a c jsou v relaci a
  - vlastnosti objektů: a(b). – objekt b má vlastnost a

Jednoduchá fakta

```
velka_hlava(lada).      % Láďa má velkou hlavu
cepice(lada).           % Láďa má čepici
```

- pravidlo: A :- B, C, D.
  - cíl A je splněn, když jsou splněny podcíle B, C a D.
  - **čárka** ≈ **konjunkce**, středník ≈ disjunkce[2], závorky ≈ priorita apod.

Jednoduchá pravidla

```
% Kdo má čepici a velkou hlavu,
% ten má velkou čepici
velka_cepice(X) :- cepice(X), velka_hlava(X).

% Je-li něco strom, keř nebo bylina,
% pak je to rostlina
rostlina(X) :- strom(X); ker(X); bylina(X).
```

[2] `;` má vyšší prioritu než `,`, ale menší než `:-`.

Unifikace

Unifikace[3] se provádí při výběru klauzule nebo v podcíli explicitně operátorem =, což **přiřazení** či porovnání dle kontextu.

- Instanciace: Z volné proměnné udělá vázanou (např. `X = term`)
- Alias/Synonyma: Jiné jméno stejné proměnné (např. `X = Y` nebo `a(X, X)`)
- Kontrola rovnosti: Vázaná proměnná musí mít stejný obsah jako druhá vázaná proměnná nebo term.
- Rekurzivní unifikace: `term(subterm(X, atom)) = term(Y)`

Pomocí unifikace lze provádět předávání argumentů hodnotou (term) i odkazem (proměnná).

[3] Prolog nemá líné vyhodnocování; volné proměnnné unifikuje při první příležitosti.

- Jméno/arita prediktátu:
  `faktorial/2` (přetěžování přes počet parametrů)
- Dokumentační zápis[4] hlavičky predikátu:
  `faktorial(+N, ?Vysledek)`
  - `+` značí “vstupní” parametr (vyžaduje plnou instanciaci)
  - `-` značí “výstupní” parametr (nevázaný/neinstanciovaný)
  - `?` značí vstupně/výstupní parametr (výlučně nebo zároveň)

Příklad

```
faktorial(0, 1).
faktorial(N, Vysledek) :-
    N > 0,
    N1 is N - 1,
    faktorial(N1, PredchoziVysledek),
    Vysledek is PredchoziVysledek * N.
```

[4] Jakože zápis v dokumentaci / zadání úkolu

Uživatel zadá cíle ke splnění v zadané/načtené databáze klauzulí:

1. vyberu predikát (podle jména) pro daný podcíl
2. vybírám klauzuli predikátu (podle unifikace parametrů) v pořadí shora dolů
3. podcíle těla klauzule je snaha splnit do hloubky zleva doprava

Uvědomme si, že:

- provedenou instanciaci proměnné lze změnit pouze jinou unifikací při navracení (backtracking)
- operátor řezu `!` slouží pro oříznutí možnosti navracení

Rovnost a nerovnost v Prologu

Existuje několik možností porovnávání:

- `=` (`\=`) – je (není) možné termy unifikovat?
  - `X=Y` $\rightarrow$ `true` – proměnné lze unifikovat
- `==`, `\==` – jsou (nejsou) termy stejné?
  - nepokouší se o unifikaci
  - `X==Y` $\rightarrow$ `false` – proměnné nejsou stejné

- Spuštění: `swipl -s file.pl` – načte soubor `file.pl`
- Znovunačtení: `reconsult('file.pl').` – znovunačte soubor `file.pl`
- Nápověda: `help(append).`, `help(.).`
- Ukončení: `halt.`
- Sledování: `trace.`, `notrace.`, `debug.`, `nodebug.`
- Kompilace z příkaz. řádku: `swipl -q -o <executable> -c <filename>`

1. úkol

Na základě predikátů o rodinných vztazích (`flp-cv4.pl`) vytvořte tyto predikáty:

- `sourozenec(X,Y)` – X je sourozenec Y
- `sestra(X,Y)` – X je sestra Y
- `deda(X,Y)` – X je děda Y
- `teta(X,Y)` – X je těta Y

Řešení

```
rodic(R,X) :-
sourozenec(X,Y) :-
sestra(X,Y) :-
deda(X,Y) :-
je_matka(X) :-
teta(X,Y) :-
```

- nehomogenní, ale jinak podobné jako v Haskellu :)
- konstruktor[5]: operátor `./2`
- prázdný seznam: `[]`
- neprázdný seznam: `[H|T]`
  - `H` – hlavička
  - `T` – zbytek

Příklady: Neprázdnost, hlavička, poslední prvek

```
X = .(jan, .(tomas, [])).
neprazdny([_|_]) :- true.
hlavicka([H|_], H).
posledni([H], H) :- !.
posledni([_|T], Res) :- posledni(T, Res).
```

[5] Pozor: `Z = .(X,Y)` není to samé jako `Z =.(X,Y)`

Časté chyby
Které si už nedovolíte udělat

```
R = rodic(X,Y)
```

Predikát není funkce, nemá návratou hodnotu, nejde přiřadit.
(Maximálně lze považovat za funkci typu `boolean` –
uspěl/neuspěl = „vrací“ true/false.)

```
rodic(R,Y)
```

```
pred(X) :- X is [a,b|[c,d]].
```

`is` je numerické vyhodnocení. Přiřazuje se normálně **`=`** (rovná se).

```
pred(X) :- X = [a,b|[c,d]].
```

V prologu se přiřazení používá dost zřídka.

```
pred([a,b|[c,d]]).
```

2. úkol

Vytvořte predikát `spoj/3`, který spojí dva seznamy do třetího. Umožněte, ať dopočítá kterýkoliv jeden vynechaný parametr.

Řešení

```
spoj([],L,L).
spoj( , , ) :- ...
```

Zkuste `spoj([1,2], X, [1,2,3,4]).`

S využitím predikátu `spoj/3` vytvořte predikát `obrat/2`, který reverzuje seznam.

Řešení

```
obrat([],[]).
obrat([H|T], Res) :-
```

4. úkol

Vytvořte predikát `sluc/3`, který sloučí dva seřazené seznamy do seznamu třetího.

Porovnání:

- aritmetické: `<`, `=<`, `>`, `>=`
- pro jakékoliv atomy: `@<`, `@=<`, `@>`, `@>=`

Řešení

```
sluc(L, [], L).
sluc([], L, L).
sluc([X|XS], [Y|YS], [X|T]) :- ...
sluc([X|XS], [Y|YS], [Y|T]) :- ...
```

5. úkol

Vytvořte predikát `serad/2`, který s pomocí predikátu `sluc/3` seřadí seznam.

Řešení

```
serad([], []).
serad([H|T], SL) :-
```

6. úkol

Vytvořte predikát `split/2`, který rozdělí zadaný seznam na podseznamy. Určený prvek zdrojového seznamu (mezera `' '`) je oddělovač – odděluje vznikající podseznamy.

Příklad

```
?- split([a,h,o,j,' ',s,v,e,t,e], S).
S = [[a, h, o, j], [s, v, e, t, e]].
```

Řešení

Dynamická konstrukce predikátů

```
C =.. [a,b,c]
call(C) ~ call(a,b,c) ~ a(b,c)
```

```
plus(X,Y,Z) :- Z is X + Y.
```

Ukázka

```
?- plus(1,2,X).
X = 3.

?- call(plus,1,2,X).
X = 3.

?- C =.. [plus,1,2,X], call(C).
C = plus(1,2,3),
X = 3.

?- help(=..).
```

7. úkol

Naimplementujte predikát `zipWith/4`, který bere jako vstup `Pred/3` a dva seznamy. Výsledkem bude nový seznam spojující prvky na odpovídajících pozicích seznamů pomocí predikátu `Pred/3`. Nekorespondující prvky delšího vstupního seznamu ignorujte.

Příklad

```
plus(X,Y,Z) :- Z is X + Y.
?- zipWith(plus, [1,2,3,4], [10,20,30,40,50], L).
 L = [11,22,33,44].
```

Řešení

```
read_line(L,C) :-
   get_char(C),
   (isEOFEOL(C), L = [], !;
   read_line(LL,_),
   [C|LL] = L ).

isEOFEOL(C) :-
   C == end_of_file;
   (char_code(C,Code), Code==10).

read_lines(Ls) :-
   read_line(L, C),
   (C == end_of_file, Ls=[] ;
   (read_lines(LLs), [L|LLs] = Ls)).
```