IB013 Logické programování I

Hana Rudová

jaro 2012

Hodnocení předmětu

- **Průběžná písemná práce:** až 30 bodů (základy programování v Prologu)
  - pro každého jediný termín: **22.března**
  - alternativní termín pouze v případech závažných důvodů pro neúčast
  - vzor písemky na webu předmětu
- **Závěrečná písemná práce:** až 150 bodů
  - vzor písemky na webu předmětu
  - opravný termín možný jako ústní zkouška
- **Zápočtový projekt:** celkem až 40 bodů
- Hodnocení: součet bodů za projekt a za obě písemky
  - známka A za cca 175 bodů, známka F za cca 110 bodů
  - známka bude zapsána pouze těm, kteří dostanou zápočet za projekt
- Ukončení předmětu zápočtem: zápočet udělen za zápočtový projekt

Základní informace

- **Přednáška:** účast není povinná, nicméně ...
- **Cvičení:** účast povinná
  - individuální doplňující příklady za zmeškaná cvičení
    - nelze při vysoké neúčasti na cvičení
  - skupina 01, sudý pátek, první cvičení **24.února**
  - skupina 02, lichý pátek, první cvičení **2.března**
- **Web předmětu: interaktivní osnova v ISu**
  - průsvitky dostupné postupně v průběhu semestru
  - harmonogram výuky, předběžný obsah výuky pro jednotlivé přednášky během semestru
  - elektronicky dostupné materiály
  - informace o zápočtových projektech

Rámcový obsah předmětu

Obsah přednášky

- základy programování v jazyce Prolog
- teorie logického programováni
- logické programování s omezujícími podmínkami
- implementace logického programováni

Obsah cvičení

- zaměřeno na praktické aspekty, u počítačů
- programování v Prologu
  - logické programování
  - DCG gramatiky
  - logické programování s omezujícími podmínkami

Literatura

- Bratko, I. **Prolog Programming for Artificial Intelligence.** Addison-Wesley, 2001.
  - prezenčně v knihovně
- Clocksin, W. F. – Mellish, Ch. S. **Programming in Prolog.** Springer, 1994.
- Sterling, L. – Shapiro, E. Y. **The art of Prolog : advanced programming techniques.** MIT Press, 1987.
- Nerode, A. – Shore, R. A. **Logic for applications.** Springer-Verlag, 1993.
  - prezenčně v knihovně
- Dechter, R. **Constraint Processing.** Morgan Kaufmann Publishers, 2003.
  - prezenčně v knihovně

\+ Elektronicky dostupné materiály (viz web předmětu)

Průběžná písemná práce

- Pro každého jediný termín **22. března**
- Alternativní termín pouze v závažných důvodech pro neúčast
- Celkem až 30 bodů (150 závěrečná písemka, 40 projekt)
- 3 příklady, 40 minut
- Napsat zadaný predikát, porovnat chování programů
- Obsah: první čtyři přednášky a první dvě cvičení
- Oblasti, kterých se budou příklady zejména týkat
  - unifikace
  - backtracking
  - řez
  - seznamy
  - optimalizace posledního volání
  - aritmetika
- Ukázka průběžné písemné práce na webu

Zápočtové projekty

- Týmová práce na projektech, až 3 řešitelé
  - zápočtové projekty dostupné přes web předmětu
- Podrobné **pokyny k zápočtovým projektům** na webu předmětu
  - bodování, obsah předběžné zprávy a projektu
  - typ projektu: LP, CLP, DCG
    - CLP a LP: **Adriana Strejčková**
    - DCG: **Miloš Jakubíček, Vojtěch Kovář**
- **Předběžná zpráva**
  - podrobné zadání
  - v jakém rozsahu chcete úlohu řešit
  - které vstupní informace bude program používat a co bude výstupem programu
  - scénáře použití programu (tj. ukázky dvojic konkrétních vstupů a výstupů)

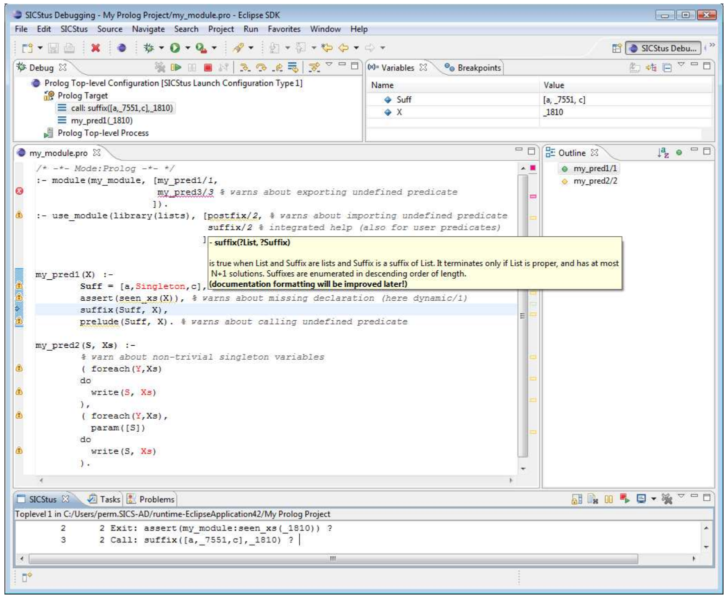
Časový harmonogram k projektům

- Zveřejnění zadání (většiny) projektů: **27. února**
- Zahájení registrace řešitelů projektu: **7. března, 19:00**
- Předběžná analýza řešeného problému: **13. dubna**
- Termín pro odevzdání projektů: **18. května**
- Předvádění projektů (po registraci): **21.května – 22.června**

Software: SICStus Prolog

- Doporučovaná implementace Prologu
- Dokumentace: `http://www.fi.muni.cz/~hanka/sicstus/doc/html`
- Komerční produkt
  - licence pro instalace na domácí počítače studentů
- Nové IDE pro SICStus Prolog SPIDER
  - dostupné až od verze SICStus 4.1.3
  - `http://www.sics.se/sicstus/spider`
  - používá Eclipse SDK
- Podrobné informace dostupné přes web předmětu
  - stažení SICStus Prologu (sw + licenční klíče)
  - pokyny k instalaci (SICStus Prolog, Eclipse, Spider)

SICStus IDE SPIDER



převzato z `http://www.sics.se/sicstus/spider`

Úvod do Prologu

Prolog

- PROgramming in LOGic
  - část predikátové logiky prvního řádu
- Deklarativní programování
  - specifikační jazyk, jasná sémantika, nevhodné pro procedurální postupy
  - **Co dělat** namísto **Jak dělat**
- Základní mechanismy
  - unifikace, stromové datové struktury, automatický backtracking

Logické programování

Historie

- Rozvoj začíná po roce 1970
- Robert Kowalski – teoretické základy
- Alain Colmerauer, David Warren *(Warren Abstract Machine)* – implementace
- SICStus Prolog vyvíjen od roku 1985
- Logické programování s omezujícími podmínkami – od poloviny 80. let

Aplikace

- rozpoznávání řeči, telekomunikace, biotechnologie, logistika, plánování, data mining, business rules, ...

- SICStus Prolog — the first 25 years, Mats Carlsson, Per Mildner. Theory and Practice of Logic Programming, 12 (1-2): 35-66, 2012. `http://arxiv.org/abs/1011.5640`.

Program = fakta + pravidla

- **(Prologovský) program** je **seznam programových klauzulí**
  - programové klauzule: fakt, pravidlo
- **Fakt:** deklaruje vždy pravdivé věci
  - `clovek( novak, 18, student ).`
- **Pravidlo:** deklaruje věci, jejichž pravdivost závisí na daných podmínkách
  - `studuje( X ) :- clovek( X, _Vek, student ).`
  - **alternativní (obousměrný) význam pravidel**

pro každé X,	pro každé X,
X studuje, jestliže	X je student, potom
X je student	X studuje

  - `pracuje( X ) :- clovek( X, _Vek, CoDela ), prace( CoDela ).`
- **Predikát:** seznam pravidel a faktů se stejným **funktorem** a **aritou**
  - značíme: `clovek/3`, `student/1`; analogie **procedury** v procedurálních jazycích,

Komentáře k syntaxi

- Klauzule ukončeny tečkou
- Základní příklady argumentů
  - **konstanty**: (`tomas, anna`) . . . začínají malým písmenem
  - **proměnné**
    - `X, Y` . . . začínají velkým písmenem
    - `_, _A, _B` . . . začínají podtržítkem (nezajímá nás vracená hodnota)
- Psaní komentářů

```
clovek( novak, 18, student ).        % komentář na konci řádku
clovek( novotny, 30, ucitel ).         /* komentář */
```


Dotaz

- **Dotaz**: uživatel se ptá programu, zda jsou věci pravdivé

```
?- studuje( novak ).        % yes    splnitelný dotaz
?- studuje( novotny ).      % no     nesplnitelný dotaz
```

- **Odpověď** na dotaz
  - positivní – **dotaz je splnitelný a uspěl**
  - negativní – **dotaz je nesplnitelný a neuspěl**
- Proměnné jsou během výpočtu **instanciovány** (= nahrazeny objekty)
  - `?- clovek( novak, 18, Prace ).`
    `Prace = student`
  - výsledkem dotazu je **instanciace proměnných** v dotazu
  - dosud nenainstanciovaná proměnná: **volná proměnná**
- Prolog umí generovat více odpovědí, pokud existují

```
?- clovek( novak, Vek, Prace ).        % všechna řešení přes ";"
```

Klauzule = fakt, pravidlo, dotaz

- **Klauzule** se skláda z **hlavy** a **těla**
- Tělo je **seznam cílů** oddělených čárkami, čárka = konjunkce
- **Fakt**: pouze hlava, prázdné tělo
  - `rodic( pavla, robert ).`
- **Pravidlo**: hlava i tělo
  - `upracovany_clovek( X ) :- clovek( X, _Vek, Prace ), prace( Prace, tezka ).`
- **Dotaz**: prázdná hlava, pouze tělo
  - ```
    ?- clovek( novak, Vek, Prace ).
    ?- rodic( pavla, Dite ), rodic( Dite, Vnuk ).
    ```


Rekurzivní pravidla

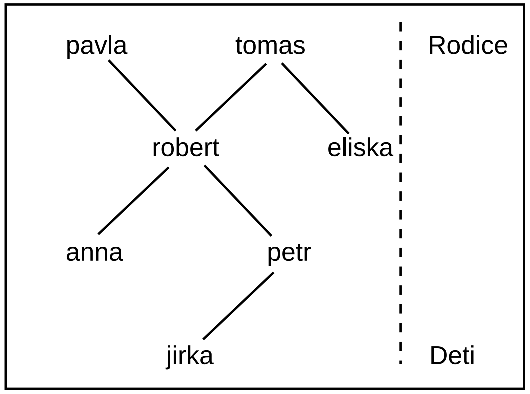
```
predek( X, Z ) :- rodic( X, Z ).            % (1)

predek( X, Z ) :- rodic( X, Y ),            % (2)
                  rodic( Y, Z ).

predek( X, Z ) :- rodic( X, Y ),            % (2’)
                  predek( Y, Z ).
```


Příklad: rodokmen

```
rodic( pavla, robert ).
rodic( tomas, robert ).
rodic( tomas, eliska ).
rodic( robert, anna ).
rodic( robert, petr ).
rodic( petr, jirka ).
```

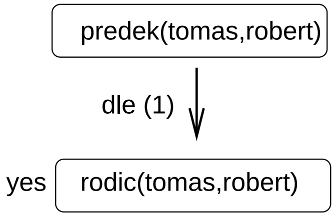



```
predek( X, Z ) :- rodic( X, Z ).      % (1)

predek( X, Z ) :- rodic( X, Y ),      % (2’)
                  predek( Y, Z ).
```

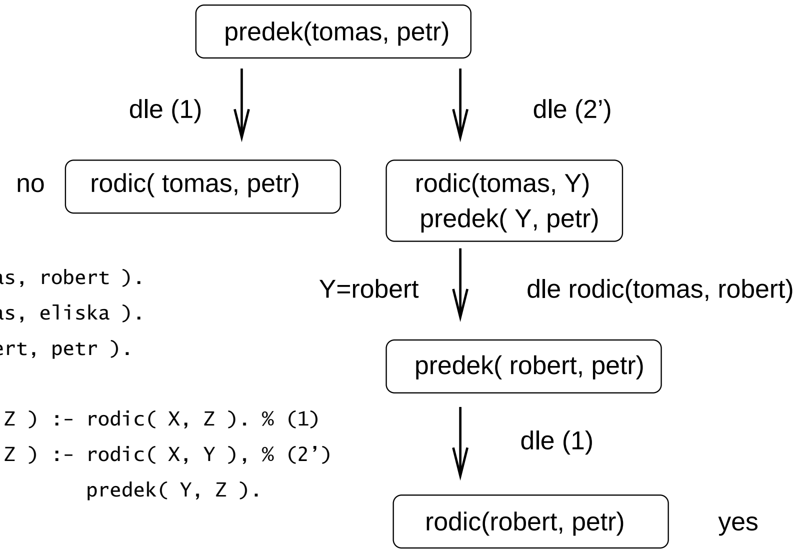

Výpočet odpovědi na dotaz ?- predek(tomas,robert)

```
rodic( pavla, robert ).
rodic( tomas, robert ).
rodic( tomas, eliska ).
rodic( robert, anna ).
rodic( robert, petr ).
rodic( petr, jirka ).
```




```
predek( X, Z ) :- rodic( X, Z ).      % (1)
predek( X, Z ) :- rodic( X, Y ),      % (2’)
                  predek( Y, Z ).
```
```

Výpočet odpovědi na dotaz ?- predek(tomas, petr)

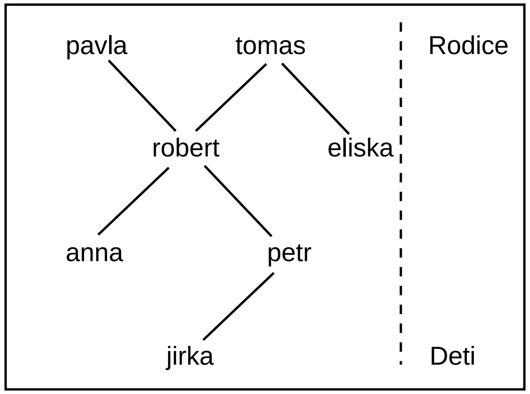



```
rodic( tomas, robert ).
rodic( tomas, eliska ).
rodic( robert, petr ).

predek( X, Z ) :- rodic( X, Z ). % (1)
predek( X, Z ) :- rodic( X, Y ), % (2')
                  predek( Y, Z ).
```


Odpověď na dotaz s proměnnou

```
rodic( pavla, robert ).
rodic( tomas, robert ).
rodic( tomas, eliska ).
rodic( robert, anna ).
rodic( robert, petr ).
rodic( petr, jirka ).
```




```
predek( X, Z ) :- rodic( X, Z ).     % (1)
predek( X, Z ) :- rodic( X, Y ),     % (2')
                  predek( Y, Z ).

predek(petr,Potomek) --> ???                          Potomek=jirka
predek(robert,P) --> ???               1. P=anna, 2. P=petr, 3. P=jirka
```


Syntaxe a význam Prologovských programů

Syntaxe Prologovských programů

- **Typy objektů jsou rozpoznávány podle syntaxe**
- **Atom**
  - řetězce písmen, čísel, „_" začínající malým písmenem: `pavel, pavel_novak, x25`
  - řetězce speciálních znaků: `<-->, ====>`
  - řetězce v apostrofech: `'Pavel', 'Pavel Novák'`
- Celá a reálná **čísla**: `0, -1056, 0.35`
- **Proměnná**
  - řetězce písmen, čísel, „_" začínající velkým písmenem nebo „_"
  - **anonymní proměnná**: `ma_dite(X) :- rodic( X, _ ).`
    - hodnotu anonymní proměnné Prolog na dotaz nevrací: `?- rodic( X, _ )`
  - lexikální rozsah proměnné je pouze jedna klauzule:

```
prvni(X,X,X).
prvni(X,X,_).
```

Termy

- **Term** – datové objekty v Prologu: `datum( 1, kveten, 2003 )`
  - **funktor**: `datum`
  - **argumenty**: `1, kveten, 2003`
  - **arita** – počet argumentů: 3
- Všechny strukturované objekty v Prologu jsou **stromy**
  - `trojuhelnik( bod(4,2), bod(6,4), bod(7,1) )`
- **Hlavní funktor** termu – funktor v kořenu stromu odpovídající termu
  - `trojuhelnik` je hlavní funktor v `trojuhelnik( bod(4,2), bod(6,4), bod(7,1) )`

Unifikace

- Termy jsou **unifikovatelné**, jestliže
  - jsou identické nebo
  - proměnné v obou termech mohou být instanciovány tak, že termy jsou po substituci identické
  - `datum( D1, M1, 2003 ) = datum( 1, M2, Y2)` **operátor** =
    `D1 = 1, M1 = M2, Y2 = 2003`
- Hledáme **nejobecnější unifikátor** *(most general unifier (MGU)*
  - jiné instanciace? ...`D1 = 1, M1 = 5, Y2 = 2003` ... není MGU
  - `?- datum( D1, M1, 2003 ) = datum( 1, M2, Y2), D1 = M1.`
- **Test výskytu** *(occurs check)*

```
?- X=f(X).
X = f(f(f(f(f(f(f(f(f(f(...))))))))))
```


Unifikace

Termy S a T jsou unifikovatelné, jestliže

1. S a T jsou konstanty a tyto konstanty jsou identické;
2. S je proměnná a T cokoliv jiného – S je instanciována na T;
   T je proměnná a S cokoliv jiného – T je instanciována na S
3. S a T jsou termy
   - S a T mají stejný funktor a aritu a
   - všechny jejich odpovídající argumenty jsou unifikovatelné
   - výsledná substituce je určena unifikací argumentů

Příklady:

```
k = k ... yes, k1 = k2 ... no, A = k(2,3) ... yes, k(s,a,l(1)) = A ... yes
s(sss(2),B,ss(2)) = s(sss(2),4,ss(2),s(1))... no
s(sss(A),4,ss(3)) = s(sss(2),4,ss(A))... no
s(sss(A),4,ss(C)) = s(sss(t(B)),4,ss(A))... A=t(B),C=t(B)... yes
```


Deklarativní a procedurální význam programů

- `p :- q, r.`
- Deklarativní: **Co** je výstupem programu?
  - `p` je pravdivé, jestliže `q` a `r` jsou pravdivé
  - Z `q` a `r` plyne `p`

  ⇒ význam mají logické relace
- Procedurální: **Jak** vypočítáme výstup programu?
  - `p` vyřešíme tak, že **nejprve** vyřešíme `q` a **pak** `r`

  ⇒ kromě logických relací je významné i pořadí cílů
  - výstup
    - indikátor yes/no určující, zda byly cíle splněny
    - instanciace proměnných v případě splnění cílů

Deklarativní význam programu

Instance klauzule: proměnné v klauzuli jsou substituovány termem

```
ma_dite(X) :- rodic( X, Y ).            % klauzule
ma_dite(petr) :- rodic( petr, Z ).      % instance klauzule
```

Máme-li program a cíl G, pak **deklarativní význam** říká:

cíl G je splnitelný právě tehdy, když — cíl `?- ma_dite(petr).`

existuje klauzule C v programu taková, že

existuje instance I klauzule C taková, že

hlava I je identická s G a

všechny cíle v těle I jsou pravdivé.

Konjunce "," vs. disjunkce ";" cílů

- **Konjunce** = nutné splnění **všech cílů**
  - `p :- q, r.`
- **Disjunkce** = stačí splnění **libovolného cíle**
  - `p :- q; r.`

```
p :- q.
p :- r.
```

  - priorita středníku je vyšší (viz ekvivalentní zápisy):

```
p :- q, r; s, t, u.

p :- (q, r) ; (s, t, u).

p :- q, r.
p :- s, t, u.
```


Pořadí klauzulí a cílů

```
(a) a(1).                                                ?- a(1).
    a(X) :- b(X,Y), a(Y).
    b(1,1).

(b) a(X) :- b(X,Y), a(Y).   % změněné pořadí klauzulí v programu vzhledem k (a)
    a(1).
    b(1,1).                 % nenalezení odpovědi: nekonečný cyklus
```

```
(c) a(X) :- b(X,Y), c(Y).                                ?- a(X).
    b(1,1).
    c(2).
    c(1).

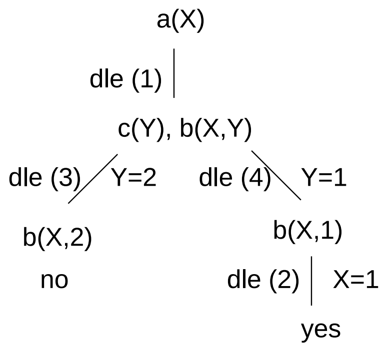
(d) a(X) :- c(Y), b(X,Y).   % změněné pořadí cílů v těle klauzule vzhledem k (c)
    b(1,1).
    c(2).
    c(1).                   % náročnější nalezení první odpovědi než u (c)
```

V obou případech **stejný deklarativní ale odlišný procedurální význam**

Pořadí klauzulí a cílů II.

```
(1) a(X) :- c(Y), b(X,Y).
(2) b(1,1).
(3) c(2).
(4) c(1).
```

`?- a(X).`



Vyzkoušejte si:

```
(1) a(X) :- b(X,X), c(X).
(3) a(X) :- b(Y,X), c(X).
(4) b(2,2).
(5) b(2,1).
(6) c(1).
```

Cvičení: průběh výpočtu

```
a :- b,c,d.
b :- e,c,f,g.
b :- g,h.
c.
d.
e :- i.
e :- h.
g.
h.
i.
```

Jak vypadá průběh výpočtu pro dotaz `?- a.`