Logické programování

Verze zima 1998

Literatura

Jirků, P.: Logické programování I. Programovací jazyk Prolog. Praha, VŠE 1995.

Kopie promítaných fólií, na lokální síti.

Jirků, P. a kol.: Programování v jazyku Prolog. Praha, SNTL 1991.

Polák, J.: Prolog. Praha, Grada 1992.

Internet newsgroup comp.lang.prolog
(viz soubor PROLOG.FAQ).

Clocksin, W. F. - Mellish, C. S.: Programming in Prolog. Berlin, Springer-Verlag 1981.

Lloyd, J. W.: Foundations of logic programming. Berlin, Springer-Verlag 1987.

Bratko, I: Prolog programming for artificial intelligence. Reading, Addison-Wesley 1986.

Flach, P.: Simply logical. Intelligent Reasoning by Example. John Wiley & Sons 1994.

Lavrač, N. - Džeroski, S.: Inductive Logic Programming. Ellis Horwood 1994.

... a mnoho dalších

Historie Prologu

30.-60.léta Výzkum problematiky strojového dokazování teorémů (Herbrand, Robinson, Kowalski...).

1972 První verze jazyka Prolog (Marseille - Colmerauer, Roussel).

1974 Teoretický model Prologu (Kowalski).

1977 Edinburská implementace (Warren) - již zhruba současná syntaxe.

1981 Prolog zahrnut do japonského projektu počítačů 5.generace.

1983 Abstraktní model překladače Prologu (Warren Abstract Machine)

1984 Začíná vycházet Journal of Logic Programming.

2.pol. 80.let Prolog běžným programovacím nástrojem; propojování s jinými jazyky.

90.léta Rozsáhlé průmyslové aplikace (viz konference PAP - od r.1993).

Prolog vs. imperativní jazyky

- Nerozlišení programů/příkazů a dat na syntaktické úrovni (Hornovy klauzule).
- Mechanismus navracení při průchodu programem.
- Absence deklarací typů; invertibilita vstupu a výstupu.
- Tradiční implementace pomocí interpretů.

Rod Jana Lucemburského

program PROGRAM.PL

```
/*
Prvni priklad logickeho programu ...
*/
% muzi
muz(jan'lucembursky).
muz(karel'IV).
...
% zeny
zena(eliska'premyslovna).
zena(blanka'z'valois).
...
% vztah manzelstvi
manzelka(eliska'premyslovna,jan'lucembursky).
manzelka(blanka'z'valois,karel'IV).
manzelka(anna'falcka,karel'IV).
...
% vztah otcovstvi
otec(karel'IV,vaclav'IV).
otec(karel'IV,zikmund).
...
% vztah materstvi
matka(eliska'premyslovna,marketa).
matka(eliska'premyslovna,judita).
...
% vztahy definovane pomoci pravidel
bratr(X,Y) :-
  muz(X),otec(O,X),matka(M,X),otec(O,Y),matka(M,Y).
...
```

Druhy klauzulí

- **Pravidlo:** *hlava* :- *tělo*.
- **Fakt:** *hlava*.
- **Dotaz:** :- *tělo*.

Základní klauzule (fakt, dotaz, ev. pravidlo) - neobsahuje proměnné.

Příklady faktů a pravidel

```
muz(jan'lucembursky).

manzelka(eliska'premyslovna,jan'lucembursky).

bratr(X,Y) :-
    muz(X),
    otec(O,X),matka(M,X),
    otec(O,Y),matka(M,Y).

deda(X,Y) :-
    muz(X),otec(X,Z),
    (otec(Z,Y);matka(Z,Y)).

soucin(X,1,X).
```

Predikáty

Arita predikátu - počet jeho argumentů:

- **0** - nulární - obvykle vyjadřuje stav "světa" (např. prsi/0, is_error/0)
- **1** - unární - obvykle vyjadřuje vlastnost objektu (např. muz/1, zena/1)
- **2** - binární - obvykle vyjadřuje vztah dvou objektů (např. manzelka/2, otec/2)
- **3** - ternární atd.

Definice predikátu - souhrn všech klauzulí (faktů a pravidel), které ho mají v hlavě.

Predikát (s aritou ≥ 2) $\sim$ DB relace!

Proměnné v Prologu

- Začínají velkým písmenem nebo podtržítkem (x predikátová logika).
- Stejné proměnné v různých klauzulích na sobě nezávislé.
- Kvantifikace:
  - proměnné v hlavě univerzálně
  - proměnné jen v těle existenčně

```
bratr(X,Y) :-
    muz(X),otec(O,X),otec(O,Y).
```

∀ x,y
muz(x) ∧ (∃ o otec(o,x) ∧ otec(o,y)) ⇒ bratr(x,y)

Tři pohledy na logický program

Syntaxe - z jakých a jak uspořádaných prvků se skládá (klauzule, atomické formule, predikátové symboly, proměnné...).

(Deklarativní) sémantika - co jednotlivé prvky vyjadřují (z hlediska predikátové logiky - implikace, konjunkce, disjunkce, kvantifikace... pravdivost tvrzení, logické vyplývání...).

Pragmatika - jak je program Prologem zpracováván.

SWI Prolog

- vyvíjen na University of Amsterdam
- public domain - FTP na swi.psy.uva.nl, adresář pub/SWI-Prolog
- primární platforma UNIX, dále Mac, MS-DOS, MS-Windows, Windows95...
- inkrementální kompilátor (s možností optimalizační kompilace)
- založen na WAM; syntax odpovídá Edinburskému standardu
- nestandardní predikáty většinou kompatibilní s Quintus Prologem (nejběžnější komerční implementace)
- napsán v Prologu a C; distribuovány i zdrojové texty
- rozhraní na C

Pracovní prostředí Prologu

- Programové klauzule ("data" i "instrukce", s výjimkou systémových - vestavěných predikátů) jsou umístěny v pracovní paměti - tzv. **databázi**.
- Databáze "v klidovém stavu" obsahuje jen *fakty* a *pravidla* (tzv. definitní klauzule); přidání *dotazu* aktivuje odvozovací mechanismus Prologu.
- Komunikace s uživatelem pomocí zadávání "příkazů" - dotazů.

Komunikace s Prologem - vestavěné predikáty (I)

- halt - ukončení činnosti Prologu
- shell - odskok do OS (UNIX)
- consult(Soubor) - načtení (kompilace) programu do databáze; varianty:

```
?- consult('program.pl').
?- consult(program).
?- [program].
?- consult(['program.pl','program1.pl']).
?- consult([program,program1]).
?- [program,program1].
```

Komunikace s Prologem - vestavěné predikáty (II)

- listing - vypsání obsahu databáze
- listing(PSym) - vypsání klauzulí všech predikátů s predikátovým symbolem PSym, bez ohledu na aritu
- listing(PSym/Arita) - vypsání klauzulí predikátu PSym/Arita

```
?- listing(muz/1).
```

Nápověda ve SWI Prologu

(nestandardní vestavěné predikáty!)

- help - přehled nápovědy
- help(Pred), help(Pred,Arita) - nápověda ke konkrétnímu predikátu/ům
- help(Sekce) - výpis příslušné sekce manuálu (1-1, 1-2 atd.)
- apropos(Text) - vypsání jmen predikátů (a čísel sekcí), v jejichž jednořádkových nápovědách se vyskytuje text Text
- explain(Objekt) - poskytnutí "vysvětlení" k datovému objektu (typu, vazba na jiné objekty)

Práce s historií příkazů

- Vyvolání předchozích dotazů pomocí kurzorových kláves
- Speciální sekvence zastupující předchozí dotazy - viz tabulka:

!!.	Zopakuj poslední dotaz
!nr.	Zopakuj dotaz č.¡nr¿
!str.	Zopakuj poslední dotaz začínající na ¡str¿
!?str.	Zopakuj poslední dotaz obsahující ¡str¿
^old^new.	Zopakuj poslední dotaz a zaměň v něm ¡new¿ místo ¡old¿
!nr^old^new.	Totéž pro dotaz č.¡nr¿
!str^old^new.	Totéž pro dotaz začínající na ¡str¿
!?str^old^new.	Totéž pro dotaz obsahující ¡str¿
h.	Vypiš historii příkazů
!h.	Vypiš tuto tabulku

Různé typy dotazů (cílů)

- Základní dotaz

  ?- muz(karel_IV).

- Využití proměnných

  ?- manzelka(X,karel_IV).
  ?- manzelka(Zena,Muz).

- Využití anonymní proměnné

  ?- manzelka(_,karel_IV).
  ?- manzelka(_,_).

- Vícenásobný výskyt proměnné

  ?- manzelka(X,X).

Různé pohledy na komunikaci pomocí dotazů

- **Příkaz**, co má Prolog udělat (typicky např. halt nebo consult).
- **Cílové tvrzení**, které má Prolog potvrdit nebo vyvrátit (typicky základní dotazy).
- **Dotaz na hodnoty** obsažené v databázi, případně vypočtené (typicky dotazy s proměnnými).

Předpoklad uzavřeného světa

Při zodpovídání dotazů Prolog využívá tzv. **předpoklad uzavřeného světa** (closed-world assumption, CWA):

"Tvrzení, která jsou obsažena v databázi (nebo z ní odvoditelná) jsou pravdivá, všechna ostatní tvrzení jsou nepravdivá."

Příklad:

?- muz(eliska_premyslovna).

No
?- muz(vaclav_II).

No

Fakty vs. pravidla

Definice predikátů:

- extenzionální - pomocí základních faktů
- intenzionální - pomocí pravidel (příp. faktů) s proměnnými
- smíšené - základní fakty postihují výjimky z pravidel

Nevýhody extenzionálních definic:

- velký rozsah
- obtížná aktualizace

bratr(karel_IV,jan_jindrich).
bratr(jan_jindrich,karel_IV).
bratr(karel_IV,vaclav_lucembursky).
bratr(vaclav_lucembursky,karel_IV).
bratr(karel_IV,marketa).
...

Unifikace

Substituce - zobrazení z konečné množiny proměnných do množiny termů.

Unifikace - nalezení takové substituce ve dvou formulích, aby vznikly dvě *totožné* formule.

Příklady unifikace cíle s databázovou klauzulí

Cíl Substituce	DB Klauzule
?- muz(karel_IV). {}	muz(karel_IV).
?- manzelka(X,karel_IV). {X ← blanka }	manzelka(blanka,karel_IV).
?- bratr(karel_IV,X). {X ← karel_IV, Y ← X }	bratr(X,Y) :- …

Rekurze

programy REKURZE.PL a REKURZE1.PL

Rekurze (přímá) - v těle klauzule se vyskytuje tentýž predikátový symbol jako v hlavě.

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

Dotazy:

```
?- predek(karel_IV,vaclav_IV).
?- predek(jan_lucembursky,vaclav_IV).
?- predek(X,vaclav_IV).
```

```
predek1(X,Y) :- rodic(X,Y).
predek1(X,Y) :- rodic(Z,Y),predek1(X,Z).
```

```
predek2(X,Y) :- rodic(X,Y).
predek2(X,Y) :- rodic(X,Z),predek2(Z,Y).
```

Rekurzivní cyklus

program REK_CYKL.PL

```
pis_rodokmen(X) :-
        otec(Y,X),
        write(Y),nl,
        pis_rodokmen(Y).
pis_rodokmen(_).
```

Vestavěné predikáty

- write(Term) - vypíše term Term na standardní výstup
- nl - odřádkuje na standardním výstupu

Oba jsou vždy splněny, jako "vedlejší efekt" svého vyhodnocení vykonají akci; při navracení vždy nesplněny.

Řízení běhu programu

- fail - vždy nesplněn

- true - vždy splněn, ale při navracení nesplněn; implicitně tělem každého faktu:

  ```
  muz(karel_IV) :- true.
  soucin(X,1,X) :- true.
  ```

- repeat - vždy splněn, a to i při navracení

  ```
  repeat.
  repeat :- repeat.
  ```

Využití predikátu true
program TRUE.PL

- zabránění neúspěchu při nesplnění dílčího cíle

```
pis˙rod1(X) :-
  (
    (otec(O,X),write(O),nl,pis˙rod1(O))
  ;
    true
  ).

pis˙rodice(X) :-
        (
                (
                otec(O,X),
                write('Otec: '),write(O),nl
                )
        ;
                true
        ),
        (
                (
                matka(M,X),
                write('Matka: '),write(M),nl
                )
        ;
                true
        ).
```

Cyklus s navracením typu DO-WHILE
(s využitím fail) - program FAIL.PL

```
vsechny˙manzelky(X) :-
        manzelka(M,X),
        write(M),nl,
        fail.
vsechny˙manzelky(˙).
```

Cyklus s navracením typu DO-UNTIL
(bez fail) - program UNTIL.PL

```
zenich :-
   muz(X),
   write('Zkoumam muze jmenem '),
   write(X),
   write('...'),nl,
   not(manzelka(˙,X)),
   write('Slava! '),
   write(X),
   write(' neni zenaty! Toho si vezmu!').
```

Cyklus s navracením typu DO-UNTIL
(s využitím repeat) - program REPEAT.PL

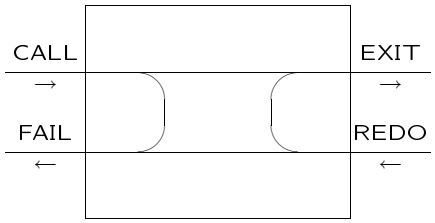
```
najdi˙osobu :-
  repeat,
  write('Zadej jmeno osoby: '),
  read(X),
  (
    (
    muz(X),
    write(X),
    write(' je muz.'),nl
    )
  ;
    (
    zena(X),
    write(X),
    write(' je zena.'),nl
    )
  ;
    (
    write('Chybne jmeno. Osoba '),
    write(X),
    write(' neni v databazi.'),nl,
    fail
    )
  ).
```

Přerušení běhu programu

- při přerušení v důsledku chyby se automaticky přejde do režimu trasování
- při přerušení uživatelem (Ctrl-Break) jsou dostupné volby:
  - a (abort) - ukončení běhu programu
  - b (break) - spuštění vyšší úrovně rozhraní
  - c (continue) - pokračování běhu programu
  - e (exit) - ukončení práce Prologu
  - g (goals) - zobrazení posloupnosti cílů
  - h (help) - přehled voleb
  - t (trace) - přechod do režimu trasování

Blokový model Prologu

Blokový (*box*, tj. schránkový, krabičkový...) model:



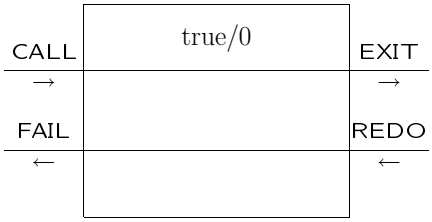
CALL - první volání cíle

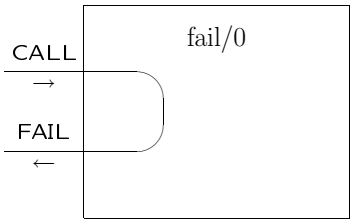
EXIT - úspěch (splnění cíle)

REDO - další volání cíle (při navracení)

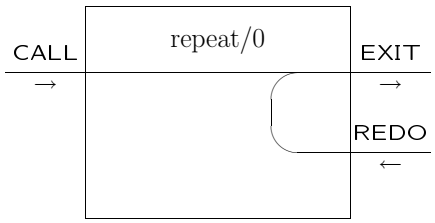
FAIL - neúspěch (nesplnění cíle)

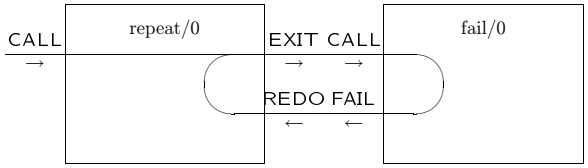
Blokové modely predikátů (I)





Blokové modely predikátů (II)





Trasování - vestavěné predikáty

- trace - aktivace trasování pro následující dotaz

```
?- trace.
?- vnucka(anna,X).
```

- leash(Ports) - na kterých portech se trasování zastavit

- visible(Ports) - které porty mají být při trasování viditelné

```
?- leash([-exit,-fail]).
?- visible(-exit).
?- leash(+full).
?- visible(+full).
```

Trasování - volby (výběr)

A (alternatives) - zobrazení cílů s altern. řešením

a (abort) - ukončení běhu programu

b (break) - spuštění vyšší úrovně rozhraní

c (creep) - pokračování v trasování - také Enter nebo mezerník

e (exit) - ukončení práce Prologu

f (fail) - vnucený neúspěch cíle (C,R,E)

g (goals) - zobrazení posloupnosti cílů

h (help) - přehled voleb

i (ignore) - ignorování cíle, jako by uspěl (C,R,F)

n (no debug) - pokračování běhu bez trasování

r (retry) - návrat ke call portu daného cíle (R,E,F)

s (skip) - pokračování, nezobrazují se podcíle (C,R)

u (up) - pokračování, zobrazuje se až nadřazený cíl

Řízení běhu programu - call

Predikát call - volání libovolného cíle (obvykle s proměnnými).

```
menu :-
   repeat,nl,nl,
   write('**********************'),nl,
   write('      M E N U'),nl,
   write('**********************'),nl,
   write('Zadejte kod polozky ukonceny teckou: '),nl,
   write('l.    Vypis stavu databaze (listing)'),nl,
   write('h.    Napoveda k napovede Prologu (help)'),nl,
   write('k.    Konec prace s menu'),nl,
   write('**********************'),nl,nl,
   write('Kod'),
   read(X),call(X),X=k.
l :- listing.
h :- help.
k.

vsechna_res(Cil,X) :-
        call(Cil),
        write(X),nl,
        fail.
vsechna_res(_,_).

?- vsechna_res(manzelka(X,karel_IV),X).
```

Řízení běhu programu - řez

Při navracení se vždy prochází zpětně celá posloupnost cílů v těle klauzule; výjimkou je použití predikátu "!" (řez):

- při volání vždy splněn
- při navracení vyvolá nesplnění celého *nadřazeného cíle* (Prolog se nenavrací k předcházejícím cílům v téže klauzuli, ani se nepokouší unifikovat nadřazený cíl s dalšími klauzulemi téhož predikátu!)
- narušuje deklarativní sémantiku programu...

Příklad použití řezu

Bez řezu

```
naslednici :-
        muz(X),
        write(X),write(':'),nl,
        naslednik(N,X),
        write('naslednik '),write(N),nl,nl,
        fail.
naslednici.

naslednik(N,X) :-       otec(X,N),muz(N).
naslednik(N,X) :-       manzelka(N,X).
naslednik(N,X) :-       otec(X,N),zena(N).
naslednik('neexistuje',_).
```

S řezem

```
naslednici1 :-
        muz(X),
        write(X),write(':'),nl,
        naslednik1(N,X),
        write('naslednik '),write(N),nl,nl,
        fail.
naslednici1.

naslednik1(N,X) :-      otec(X,N),muz(N),!.
naslednik1(N,X) :-      manzelka(N,X),!.
naslednik1(N,X) :-      otec(X,N),zena(N),!.
naslednik1('neexistuje',_).
```

Řízení běhu programu - not

Predikát not - negace *neúspěchem*:

- jen v těle klauzule
- cíl not(X) uspěje právě tehdy, když cíl X neuspěje
- definovatelná jako

```
not(X) :- call(X),!,fail.
not(_).
```

- příklad použití:

```
bratranec(X,Y) :-
    vnuk(X,Z),vnuk(Y,Z),not(bratr(X,Y)).
```

Problémy s negací neúspěchem

1. Problém dvojí negace - např.

```
a(X) :- b(X).
c(X) :- not(not(d(X))).
b(1).
d(1).

?- a(X).
X=1       Yes
?- c(X).
X= ¡volna promenna¿       Yes
```

2. Ověření tautologie $(non(P) \Rightarrow P) \Rightarrow P$ vede na nekonečný výpočet!

```
p :- not(p).
?- p.
```

Práce s databází (I)
Klauzulární ("vnitřní") databáze

V klauzulární databázi jsou uloženy zkompilované klauzule:

- consult(Soubor) načte (zkompiluje) klauzule ze souboru Soubor
- assert(Term) vloží Term jako klauzuli na konec definice příslušného predikátu; asserta(Term) ukládá na začátek; predikáty nemohou být splněny při navracení
- retract(Term) odstraní z databáze první klauzuli unifikovatelnou s cílem Term; při navracení odstraňuje další unifikovatelné klauzule
- retractall(Term) odstraní všechny unifikovatelné klauzule (v cyklu s navracením...)
- abolish(PSym,Arita) odstraní všechny klauzule daného predikátu
- clause(Head,Body) unifikuje Head s hlavou a Body s tělem první unifikovatelné klauzule; při navracení unifikuje s dalšími klauzulemi

Práce s databází (II)
Záznamová ("vnější") databáze

V záznamové databázi jsou uloženy posloupnosti údajů identifikované klíčem:

- recordz(Key,Term) vloží Term na konec posloupnosti identifikované klíčem Key; recorda(Key,Term) ukládá na začátek; predikáty nemohou být splněny při navracení
- recorded(Key,Value) vyhledá první údaj v posloupnosti identifikované klíčem Key, a vrátí ho jako Value; při navracení se vyhledávají další hodnoty v posloupnosti
- recordz(Key,Term,Ref), recorda(Key,Term,Ref) a recorded(Key,Value,Ref) navíc vracejí referenční číslo Ref (paměťová adresa)
- erase(Ref) odstraní údaj s adresou Ref

Referenční čísla a erase lze využívat i pro *klauzulární* databázi, pomocí

```
assert(Term,Ref)
asserta(Term,Ref)"
clause(Head,Body,Ref)
```

Práce s databází (III)
Příklady

```
assert(muz(vojtech˙svatek)).
asserta(muz(caesar)).
retract(muz(X)).
retractall(muz(X)).
retractall(manzelka(X,karel˙IV)).
abolish(zena,1).
clause(X,Y).
clause(otec(X,Y),Body).
clause(bratr(X,Y),Body).

recordz(muz,petr).
recordz(muz,jiri).
recorda(muz,michal).
recorded(muz,X).
recoded(muz,jiri,Ref),erase(Ref).
```

Práce s databází (IV)
Program ASSERT.PL

Předání dat při volání

```
pis˙rodinu(Muz) :-
  muz(Muz),najdi(Muz,Zena,Syn,Dcera),
  tiskni(Muz,Zena,Syn,Dcera).

najdi(Muz,Zena,Syn,Dcera) :-
  (manzelka(Zena,Muz);Zena='neexistuje'), ...

tiskni(Muz,Zena,Syn,Dcera) :-
  ... write('Zena: '),write(Zena),nl, ...
```

Předání dat pomocí databáze

```
pis˙rodinu1(Muz) :-
  muz(Muz),najdi1(Muz),tiskni1.

najdi1(Muz) :-
  ... (manzelka(Zena,Muz);Zena='neexistuje'),
  assert(r˙zena(Zena)), ...

tiskni1 :-
  ... retract(r˙zena(Zena)),
  write('Zena: '),write(Zena),nl, ...
```

Anatomie Prologu
Druhy termů

Term - univerzální datová struktura Prologu; rozlišují se:

- jednoduché (atomické) termy
  - konstanty
    - atomy
    - čísla
    - řetězce (jen v některých implementacích)
  - proměnné
- složené termy (struktury)

Atomy

Atomy - buď začínají malým písmenem, nebo jsou sestaveny pouze ze speciálních znaků, nebo v apostrofech.

```
a        a˙1          toto˙je˙atom
muz      karel˙IV     'Karel IV'
a324     'nazdar!'    =
?-       :-           !
a=       a-           a+
```

Příklady výrazů, které nejsou atomy

```
324a     nazdar!      =a
-a       ˙a           muz(karel˙IV)
```

Proměnné

Proměnné - začínají malým písmenem nebo podtržítkem.

X _123 XYZ
Xyz X_ _
Promenna _promenna _abc
Toto_neni_konstanta_ale_promenna
_toto_je_take_promenna

Příklady výrazů, které nejsou proměnnými:

1_X 1X x
toto_neni_promenna_ale_konstanta
A! _a%

Struktury (I)

Struktury - skládají se z *funktoru* (atom) a argumentů (libovolné termy). Např.:

- predikátové struktury (atomické formule)
- klauzule
- funkční struktury (→ v argumentech predikátů)

muz(karel_IV)
muz(X)
bratr(X,Y) :- muz(X),otec(O,X),otec(O,Y)
:-(bratr(X,Y),(muz(X),otec(O,X),otec(O,Y)))
trojuhelnik(v(0,0),v(1,0),v(0,1))

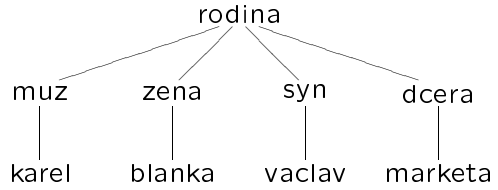
Využití funkčních struktur (STRUKTUR.PL)

...
uloz(Muz,Zena,Syn,Dcera) :-
  assert(rodina(muz(Muz),zena(Zena),
                syn(Syn),dcera(Dcera))).

Struktury (II)

Stromová reprezentace struktur:

rodina(muz(karel),zena(blanka),
  syn(vaclav),dcera(marketa))



bratr(X,Y) :- muz(X),otec(O,X),otec(O,Y)

Operátory

Funktor deklarovaný jako *operátor* umožňuje jiný zápis struktury než prefixový se závorkami:

+(1,2) :-(a,',',(b,c)) -(5) ¿(X,5)
1+2 a :- b,c -5 X¿5

Role operátorů

- Konstrukce klauzulí (:- ?-)
- Funktor ve funkční struktuře (aritmetické, . a ^) - vyhodnocením obvykle získáme číslo/strukturu
- Predikátový symbol - vyhodnocením získáme pravdivostní hodnotu

Typy předdefinovaných operátorů

1. Logické operátory , ; not :- ?-
2. Aritmetické operátory

   + - * / ^ mod //
   /“ “/ “ xor ¿¿ ¡¡ is
3. Relační operátory

   ¡ ¿ ¿= =¡ ...
4. Unifikační operátory = “=
5. Seznamový operátor .
6. Konverzní operátor =.. (“univ”)
7. Existenční kvantifikátor ^
8. Podmínkový operátor -¿

Podmínkový operátor

Podmínkový operátor -¿ (“if-then-else”) - v novějších implementacích; definovatelný jako:

```
X -¿ Y ; ' :-      X, !, Y.
X -¿ ' ; Y :-      !, Y.
X -¿ Y :-          X, !, Y.
```

Příklady - program IFTHEN.PL:

```
pis'rod2(X) :-
        otec(O,X) -¿
        (write(O),nl,pis'rod2(O)).

pis'rodice1(X) :-
        otec(O,X) -¿
        (write('Otec: '),write(O),nl),
        matka(M,X) -¿
        (write('Matka: '),write(M),nl).
```

Tabulka předdefinovaných operátorů

1200	xfx	–¿, :-
1200	fx	:-, ?-
1150	fx	dynamic, multifile, module'transparent, discontiguous
1100	xfy	;, —
1050	xfy	-¿
1000	xfy	,
954	xfy	“‘
900	fy	“+, not
700	xfx	¡, =, =.., =@=, =:=, =¡, ==, =“=, ¿, ¿=, @¡, @=¡, @¿, @¿=, “=, “==, is
600	xfy	:
500	yfx	+, -, /“, “/, xor
500	fx	+, -, ?, “
400	yfx	*, /, //, ¡¡, ¿¿
300	xfx	mod
200	xfy	^

1. priorita (čím nižší číslo tím vyšší)
2. typ (unární/binární, prefix/infix/postfix)
3. asociativita:
   - x: operátory uvnitř musejí mít vyšší prioritu
   - y: operátory uvnitř mohou mít i stejnou prioritu
4. seznam jmen operátorů

Deklarace nových operátorů

Vestavěný predikát op(Priorita,TypAsoc,Op); Op je jméno nového operátoru nebo jejich seznam (v [ ]).

Obvykle jako *direktiva* (dotaz umístěný v programu) - volá se při načtení - viz program OPERATOR.PL:

```
:- op(500,yfx,je'bratrem).
:- op(500,xf,je'muzem).

X je'bratrem Y :- bratr(X,Y).
X je'muzem :- muz(X).
```

Unifikační a relační operátory (I)

Operátor unifikace = uspěje, jestliže se podaří unifikovat výrazy na levé a na pravé straně (analogicky k unifikaci cíle s hlavou klauzule).

```
?- 1=1.
?- X=1.
?- 1+2=3.
?- 1+2=2+1.
?- X=1+2.
?- 1+2=1+2.
?- X=1,X=2.
?- X=Y,X=1.
```

Operátor neunifikovatelnosti "= uspěje, když se unifikace nezdaří; jakoby X "= Y :- not(X=Y).

```
bratr(X,Y) :-
    X"=Y,muz(X),otec(O,X),otec(O,Y).
```

Unifikační a relační operátory (II)

Operátor identity termů == uspěje při porovnávání stejných termů; operátor "== uspěje, když termy stejné nejsou.

```
?- 1==1.
?- X=a,Y=a,X==Y.
?- 1+2==3.
?- X==1.
```

Operátory porovnání termů podle standardního pořadí:

@¡ @=¡ @¿ @¿=

(pomocí standardního pořadí lze porovnávat libovolné typy termů!)

(Nestandardní!) operátor strukturní identity =@= (a jeho negace "=@=) - vyžaduje stejnou stromovou strukturu a stejný "vzor" proměnných.

Unifikační a relační operátory (III)

Operátory porovnávání numerických výrazů

=:= ="= ¡ ¿ ¿= =¡

- před porovnáním se oba výrazy *vyhodnotí!*

```
?- 1+2=:=3.
?- 2*3=:=5+1.
?- a=:=a.
?- X=:=1+1.
```

Operátor is - "přiřazovací příkaz" - vyhodnotí pravou stranu, a pak ji unifikuje s levou; na levé straně musí být volná proměnná, případně číslo.

```
?- X is 1+2.
?- 3 is 1+2.
?- 1+2 is 1+2.
?- X is 1, X is X+1.
```

Aritmetické funkce

Predikáty, které lze vyhodnocovat pomocí is a relačních operátorů. Vestavěné:

```
+ - * / ^ mod // /" ¡¡ ¿¿ " "/ xor
sqrt abs ceil floor integer max min
e exp log log10
pi sin cos tan asin acos atan
cputime random
```

Definice nové aritmetické funkce - vestavěný predikát arithmetic_function(PSym/Arity) (program ARITM.PL):

```
:- arithmetic_function(mean/2).

mean(A, B, C) :-
        C is (A+B)/2.
```

(hodnota C se nevrací argumentem, ale jako výsledek vyhodnocení struktury)

Numerické výpočty v cyklu (I)

Rekurzivní cyklus

```
predek´v(X,Y,1) :-
    rodic(X,Y).
predek´v(X,Y,N) :-
    rodic(Z,Y),predek´v(X,Z,N0), N is N0+1.

faktorial(1,1) :- !.
faktorial(N,F) :-
    N1 is N-1, faktorial(N1,F1),F is F1*N.

faktor1(N,F) :-
    faktor11(N,F,1,1).
faktor11(N,F,N,F) :- !.
faktor11(neexistuje,F,´,F0) :-
    F0 ¿, F, !.
faktor11(N,F,N0,F0) :-
    N1 is N0+1,F1 is F0*N1,faktor11(N,F,N1,F1).
```

Numerické výpočty v cyklu (II)

Cyklus s navracením

```
p´manzelek(Muz,´) :-
    assert(mem(0)),
    manzelka(´,Muz),
    retract(mem(N)),N1 is N+1,assert(mem(N1)),
    fail.
p´manzelek(´,N) :-
    retract(mem(N)).

kolikbodu(Student,Test,´) :-
   assert(zatim(0)),
   body(Student,Test,B),
   retract(zatim(Zatim)),
   Zatim1 is Zatim+B,
   assert(zatim(Zatim1)),
   fail.
kolikbodu(´,´,Body) :-
   retract(zatim(Body)).
```

Seznamy

Definice

1. Prázdný seznam [] (tj. atom!) je seznam.
2. Binární struktura s funktorem (operátorem) ., jejíž
   - 1.argument (hlava seznamu) je libovolný term, a
   - 2.argument (tělo seznamu) je seznam,

   je seznam.

.(a,[]) .(a,.(b,[])) .(a,.(b,.(c,[])))

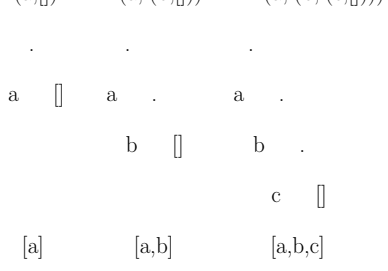
.(.(a,[]),[]) .(.(a,.(b,[])),.(c,.(d,[])))

.(f(a,b),.(g(c,d),[])) .([],[])

Seznamy

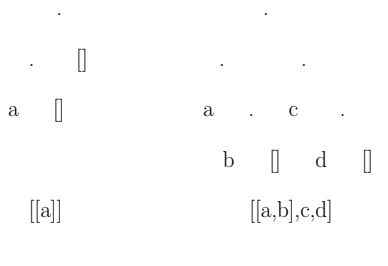
Alternativní zápisy

.(a,[]) .(a,.(b,[])) .(a,.(b,.(c,[])))



[a] [a,b] [a,b,c]

.(.(a,[]),[]) .(.(a,.(b,[])),.(c,.(d,[])))

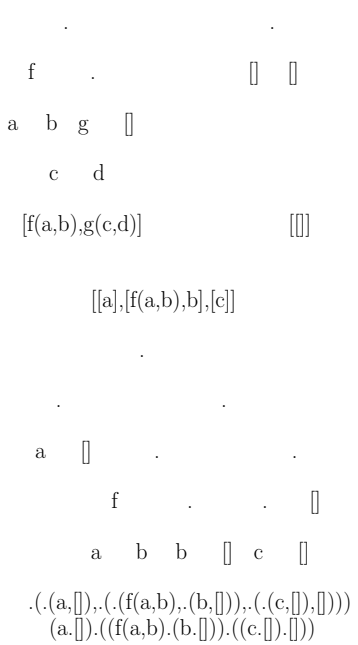


[[a]] [[a,b],c,d]

Seznamy

Alternativní zápisy - pokr.

.(f(a,b),.(g(c,d),[])) .([],[])



Seznamy s proměnnými

Seznamy se známým počtem prvků

.(X,[]) .(X,.(Y,[])) .(X,.(Y,.(Z,[])))

[X] [X,Y] [X,Y,Z]

Seznamy s neznámým počtem prvků

.(X,Y) .(X,.(Y,Z)) .(X,.(Y,.(Z,W)))

[X—Y] [X,Y—Z] [X,Y,Z—W]

Operátor — - oddělovač hlavy a těla seznamu v "[]" notaci.

Predikáty pro práci se seznamy (I)

Program SEZNAM.PL

Prvek seznamu - member

```
prvek(H,[H—_]).
prvek(X,[_—T]) :-
        prvek(X,T).
```

Spojování (a rozpojování) seznamů - append

```
spojeni([],S,S).
spojeni([H—T],S,[H—U]) :-
        spojeni(T,S,U).
```

Obrácení seznamu - reverse - 1.varianta

```
obraceni0([],[]).
obraceni0([H—T],S) :-
        obraceni0(T,T1), spojeni(T1,[H],S).
```

Obrácení seznamu - reverse - 2.varianta

```
obraceni(S,S1) :-
        obraceni1(S,[],S1).
obraceni1([],S,S).
obraceni1([H—T],S0,S) :-
        obraceni1(T,[H—S0],S).
```

Predikáty pro práci se seznamy (II)

Pořadí prvku v seznamu - nth1, nth0

```
prvni(X,[X—_]).
druhy(X,[_,X—_]).
treti(X,[_,_,X—_]).

n_ty(1,[H—_],H).
n_ty(N,[_—T],X) :-
    n_ty(N0,T,X), N is N0+1.
```

Poslední prvek seznamu - last

```
posledni(X,[X]).
posledni(X,[_—Y]) :- posledni(X,Y).
```

Délka seznamu - length

```
delka([],0).
delka([_—T],D) :- delka(T,D0), D is D0 + 1.
```

Zploštění ("rozplynutí" vnořených seznamů) - flatten

```
zplosteni([],[]) :- !.
zplosteni([H—T],Seznam) :-
  rozklad(H,S),
  zplosteni(T,T1),
  append(S,T1,Seznam).
rozklad([H—T],Listy) :- !,zplosteni([H—T],Listy).
rozklad(X,[X]).
```

Predikáty pro práci se seznamy (III)

Program SEZNAM1.PL

Spojování s využitím rozdílových seznamů

```
spojeni˙r(X-Y,Y,X).

?- spojeni˙r([a,b,c—P]-P, [d,e,f], C)
?- spojeni˙r([a,b,c,d,e,f]-[d,e,f], [d,e,f], C).
?- spojeni˙r([a,b,c,d,e,f]-[d,e,f], [d,e,f],
             [a,b,c,d,e,f]).
```

Odstranění všech výskytů prvku ze seznamu - delete

```
odstran([],˙,[]).
odstran([X—T],X,S) :- !, odstran(T,X,S).
odstran([H—T1],X,[H—T2]) :- odstran(T1,X,T2).
```

Odstranění jednoho výskytu prvku ze seznamu - select

```
odstran1([X—T],X,T).
odstran1([H—T1],X,[H—T2]) :- odstran1(T1,X,T2).
```

Permutace seznamu

```
permutace([],[]).
permutace(S,[P—S2]) :-
        select(S,P,S1),
        permutace(S1,S2).
```

Predikáty pro práci se seznamy (IV)

Začátek, úsek a podseznam

```
zacatek([],˙).
zacatek([H—T1],[H—T2]) :- zacatek(T1,T2).

usek(S1,S2) :- zacatek(S1,S2).
usek(S,[˙—T]) :- usek(S,T).

podseznam([],˙).
podseznam([H—T1],[H—T2]) :- podseznam(T1,T2).
podseznam(S,[˙—T]) :- podseznam(S,T).
```

Redukce seznamu na množinu - list_to_set

```
sez˙mnoz(S,M) :-
        sez˙mnoz1(S,[],M0),
        reverse(M0,M).
sez˙mnoz1([],M,M).
sez˙mnoz1([H—T],M0,M) :-
        member(H,M0),!,
        sez˙mnoz1(T,M0,M).
sez˙mnoz1([H—T],M0,M) :-
        sez˙mnoz1(T,[H—M0],M).
```

Predikáty pro práci se seznamy (V)

Sjednocení množin - union

```
sjednoceni([],X,X).
sjednoceni([X—Y],U,V) :-
   member(X,U),!,sjednoceni(Y,U,V).
sjednoceni([X—Y],U,[X—V]) :-
   sjednoceni(Y,U,V).
```

Průnik množin - intersection

```
prunik([],X,[]).
prunik([X—U],Y,[X—V]) :-
   member(X,Y),!,prunik(U,Y,V).
prunik([X—U],Y,V) :-
   prunik(U,Y,V).
```

Podmnožina - subset

```
podmnozina([H—T],X) :-
   member(H,X),podmnozina(T,X).
podmnozina([],X).
```

Rozdíl množin - subtract

```
rozdil˙mnozin([],X,[]).
rozdil˙mnozin([H—T],X,[H—Y]) :-
   not(member(H,X)),!,rozdil˙mnozin(T,X,Y).
rozdil˙mnozin([H—T],X,Y) :- rozdil˙mnozin(T,X,Y).
```

Slučování a třídění seznamů (I)

Rozpoznání setříděného seznamu

```
setrideno([]).
setrideno([˙]).
setrideno([X,Y—T]) :-
        X @¡ Y,
        setrideno([Y—T]).
```

Slučování setříděných seznamů - merge

```
sluc([],Seznam,Seznam).
sluc(Seznam,[],Seznam).
sluc([X—T1],[Y—T2],[X—T3]) :-
   X == Y,!,
   sluc(T1,T2,T3).
sluc([X—T1],[Y—T2],[X—T3]) :-
   X @¡ Y,!,
   sluc(T1,[Y—T2],T3).
sluc(S1,[Y—T2],[Y—T3]) :-
   sluc(S1,T2,T3).
```

Naivní třídění

```
tridit(S,SS) :-
        permutace(S,SS),
        setrideno(SS).
```

Slučování a třídění seznamů (II)

Bublinové třídění (bubble-sort)

```
bubble_sort(S,SS) :-
        append(S0,[X,Y|T],S),
        Y @< X,!,
        append(S0,[Y,X|T],S1),
        bubble_sort(S1,SS).
bubble_sort(SS,SS).
```

"Rychlé" třídění (quick-sort)

```
quick_sort([],[]).
quick_sort([H|T],S) :-
        rozdel(H,T,Men,Vet),
        quick_sort(Men,Men1),
        quick_sort(Vet,Vet1),
        append(Men1,[H|Vet1],S),!.
rozdel(X,[H|T],[H|Men],Vet) :-
        H @< X,rozdel(X,T,Men,Vet),!.
rozdel(X,[H|T],Men,[H|Vet]) :-
        rozdel(X,T,Men,Vet).
rozdel(_,[],[],[]).
```

Slučování a třídění seznamů (III)

Quick-sort s využitím rozdílových seznamů

```
quick_sort_r([H|T],SS-T1) :-
   rozdel(H,T,U,V),
   quick_sort_r(U,SS-[H|T2]),
   quick_sort_r(V,T2-T1).
quick_sort_r([],T-T).

qs(Seznam,SetridenySeznam) :-
   quick_sort_r(Seznam,SetridenySeznam-[]),!.
```

Měření rychlosti třídících algoritmů - využití aritmetických funkcí cputime a random - program CPUTIME

```
cas(Cil,Cas) :-
        X is cputime, call(Cil),
        Y is cputime, Cas is Y-X.
test_n(N,Cas) :-
        nesetr(N,S0),cas(tridit(S0,_),Cas).
test_b(N,Cas) :-
        nesetr(N,S0),cas(bubble_sort(S0,_),Cas).
test_q(N,Cas) :-
        nesetr(N,S0),cas(qs(S0,_),Cas).
nesetr(0,[]).
nesetr(N,[X|S]) :-
        X is random(100),N1 is N-1,nesetr(N1,S).
```

Konverze seznam-databáze

program SEZ_DAT.PL

Ukládání prvků seznamu do klauzulí

```
sez_dat([]).
sez_dat([H|T]) :-
    asserta(sez_kl(H)),
    sez_dat(T).
```

"Sbírání" hodnot z klauzulí do seznamu

```
dat_sez(_) :-
    assert(mem([])),
    retract(sez_kl(H)),
    retract(mem(T)),
    assert(mem([H|T])),
    fail.
dat_sez(S) :-
    retract(mem(S)).
```

Řetězce

- atomické termy - posloupnosti znaků
- zápis v "", např. "abc"
- častá konverze na seznam ASCII kódů; SWI Prolog automaticky na vstupu (pokud není nastaveno style_check(+string)), např.:

  ?- X="abc", Y="def", append(X,Y,Z).

  X = [97,98,99]
  Y = [100,101,102]
  Z = [97,98,99,100,101,102]
  Yes

- práce s nimi se liší podle implementace!

Klasifikace termů

Ve většině implementací:

- atom(+Term)
- string(+Term)
- integer(+Term) - celé číslo
- float(+Term) - číslo s desetinnou tečkou
- number(+Term) - číslo (celé nebo s desetinnou tečkou)
- atomic(+Term) - atomický term (atom, řetězec nebo číslo)
- var(+Term) - volná proměnná (viz program VAR.PL)
- nonvar(+Term) - cokoliv kromě volné proměnné

Obvyklé použití - kontroly dat na vstupu.

Analýza a konverze termů

Ve většině implementací:

- name(?Atom,?Seznam) - konvertuje atom na seznam ASCII hodnot a naopak (viz program NAME.PL)
- functor(?Term,?Funktor,?Arita) - zjistí funktor a aritu struktury, nebo zkonstruuje strukturu obsahující volné proměnné
- arg(?ArgNo,?Term,?Value) - zjistí hodnotu argumentu struktury, nebo konkretizuje argument hodnotou
- =.. (operátor "univ") - konvertuje strukturu na seznam a naopak, např.:

  f(a,b) =.. [f,a,b]

Nestandardní predikáty pro práci s termy (SWI Prolog)

Klasifikační:

- ground(+Term) - základní term
- is_list(+Term) - seznam (struktura s funktorem '.')
- proper_list(+Term) - seznam, jehož tělo není proměnná

Konverzní:

- string_to_atom(?Ret,?Atom) - řetězec na atom a naopak
- string_to_list(?Ret,?Seznam) - řetězec na seznam ASCII hodnot a naopak
- int_to_atom(?Cis,?Atom) - celé číslo na atom a naopak
- term_to_atom(?Struk,?Atom) - struktura na atom a naopak

Jiné:

- atom_length(+Atom,-Delka) - délka atomu
- concat(?A1,?A2,?A3) - spojení (ev. rozdělení) atomů
- string_length(+Ret,-Delka) - délka řetězce
- substring(+Ret,+Start,+Delka,-Podret) - vyjmutí podřetězce délky Delka z řetězce Ret, počínaje pozicí Start

Vstup a výstup - přehled (I)
Vstup a výstup s otevřeným souborem

Např. predikáty:

- nl(+Id) - odřádkování
- get(+Id,+Char) - načtení znaku
- put(+Id,+Char) - zapsání znaku (zadaného atomem nebo ASCII hodnotou)
- tab(+Id,+N) - zapsání N mezer
- read(+Id,+Term) - načtení termu
- write(+Id,+Term) - zapsání termu
- display(+Id,+Term) - zapsání termu ve standardní notaci

Id je identifikátor otevřeného souboru (kanálu); pokud není uveden, použije se aktuální V/V soubor (obvykle user).

Vstup a výstup - přehled (II)

Otvírání a zavírání souborů

1. Přesměrování standardního V/V (viz program TELL.PL)
   - see(+Soubor) - přesměruje vstup do Soubor
   - seeing(-Soubor) - vrátí jméno aktuálního vstupního souboru
   - seen - uzavření aktuálního vstupního souboru
   - tell(+Soubor) - přesměruje výstup do Soubor
   - append(+Soubor) - přesměruje výstup do Soubor (s připisováním na konec)
   - telling(-Soubor) - vrátí jméno aktuálního výstupního souboru
   - told - uzavření aktuálního výstupního souboru

2. Práce s pojmenovanými kanály
   - open(+Soubor,+Mod,?Id) - otevření souboru Soubor; Mod je jedno z read, write, append; Id (identifikátor kanálu) je buď zadaný atom nebo proměnná konkretizovaná systémem
   - close(Id) - uzavření kanálu

Další prvky SWI Prologu

- formátovaný výstup
- možnost předefinovat systémové predikáty
- optimalizační kompilace (do spustitelných souborů)
- dekompozice na moduly
- obousměrné propojení s jazykem C
- správa paměti
- prostředky pro analýzu efektivity programů
- správa souborů
- DDE komunikace
- integrace s OO prostředím pro vývoj GUI
- ...

Imperativní jazyky a Prolog Srovnání

datových a programových struktur

Imperativní jazyky	Prolog
lokální proměnné globální proměnné pole se sekvenčním přístupem pole s přímým přístupem řetězec struktura	proměnné DB klauzule seznam DB klauzule atom / řetězec struktura
procedura funkce parametry přiřazení	predikát aritmetická funkce argumenty unifikace / operátor is
sekvence cyklus podmínka	konjunkce navracení / rekurze disjunkce+řez / podmínkový operátor

Relační databáze

Soubor DB.PL

```
p(j(jan,novak),adr(praha,vodickova,10),
  muz,54,89,176,diabetes,?,v(10.3,pos,pos)).
p(j(josef,novak),adr(praha,vodickova,12),
  muz,18,70,178,diabetes,astma,v(?,?,?)).
p(j(jan,novacek),adr(brno,ceska,36),
  muz,57,78,172,astma,?,v(10.2,neg,neg)).
p(j(jana,novakova),adr(brno,janackova,1231),
  zena,67,70,167,diabetes,?,v(?,?,?)).
```

Selektory - predikáty, které vybírají záznamy podle hodnot položek

```
astmatik(Jmeno,Prijmeni) :-
   p(j(Jmeno,Prijmeni),_,_,_,_,_,astma,_);
   p(j(Jmeno,Prijmeni),_,_,_,_,_,_,astma,_).

diabetik(Jmeno,Prijmeni) :-
   p(j(Jmeno,Prijmeni),_,_,_,_,_,Hl_dg,Vedl_dg,_),
   (Hl_dg = diabetes;Vedl_dg = diabetes).

obezni(Jmeno,Prijmeni) :-
   p(j(Jmeno,Prijmeni),_,_,_,Hmotn,Vyska,_,_,_),
   B_index is Hmotn/(Vyska-100),B_index ¿ 1.1.

obezni_diabetik(Jmeno,Prijmeni) :-
   diabetik(Jmeno,Prijmeni),
   obezni(Jmeno,Prijmeni).
```

Zápočtová úloha I

Soubor KIZI.PL

K níže uvedené relační databázi připojte program, který bude v menu (vyvolaném predikátem menu/0) nabízet následující funkce:

1. Výpis všech předmětů ve tvaru ident - název
2. Výpis všech předmětů učitele (zadaného příjmením) - ident a název
3. Výpis všech učitelů předmětu (zadaného identem) - jméno a příjmení
4. Zjištění počtu osob se zadaným titulem
5. Zjištění celkového počtu kreditů na bakalářském (identy s 2xx) a inženýrském (identy se 4xx) stupni.

Program bude zahrnovat kontrolu správnosti vstupu od uživatele.

Při tvorbě programu použijte pouze ty z vestavěných predikátů SWI Prologu, které jsou ve výkladu uváděny jako standardní.

```
% p(Ident,Nazev,Kredity,Hodnoceni,Vyucujici).
p(izi211,metody˙zpracovani˙informaci,2,zap,
  [rauch,ivanek]).
...
% u(Prijmeni,Jmeno,Tituly,Pozice˙na˙katedre).
u(berka,petr,[ing,csc,doc],clen).
...
```

Expertní systém (EXSYS.PL)

```
rozpoznej(X):-
  assert(fakt(nul,nul)),pravidlo(N,X,Y,Z),
  over(Z),zaver(X,Y,N),!,rozpoznej(Y).
rozpoznej(X):-
  write('Vic nelze odvodit. Zaver: '),write(X),
  retractall(fakt(˙,˙)),nl.

over([]).
over([H—T]):-dotaz(H),over(T).
dotaz(X):-fakt(X,ano),!.
dotaz(X):-fakt(X,ne),!,fail.
dotaz(X):-text(X,Text),write(Text),nl,!,odpoved(X).
odpoved(C) :- read(X),asserta(fakt(C,X)),ano˙ne(X).
ano˙ne(X):- (X==ano ; X==ne),!,X==ano.
ano˙ne(˙):-write('Dovoleny tvar odpovedi je: ano./ne.'),
  nl,write('Odpovezte znovu.'),read(X),ano˙ne(X).

zaver(X,Y,N):-
  nl,write('Je to '),write(X),write('-'),write(Y),
  write(' podle pravidla '),write(N),nl.

pravidlo(1,zvire,savec,[c1]).
pravidlo(2,zvire,savec,[c2]).
...

text(c1,'Ma srst?').
text(c2,'Dava mleko?').
...
```

Grafy (GRAFY.PL)

```
cesta(X,X).
cesta(X,Y) :- h(X,Z),cesta(Z,Y).

hled˙cesty(Pocatek,Konec,Cesta) :-
   zmena˙uzlu([Pocatek],Konec,Cesta0),
   reverse(Cesta0,Cesta).
zmena˙uzlu([Konec—T],Konec,[Konec—T]) :- !.
zmena˙uzlu([H—T],Konec,Cesta) :-
   h(H,H1),not(member(H1,[H—T])),
   zmena˙uzlu([H1,H—T],Konec,Cesta).

nejkr˙cesta(Pocatek,Konec,˙,˙) :-
   assert(nej([neexistuje],100000000000)),
   zmena˙uzlu([Pocatek],Konec,Cesta0),
   reverse(Cesta0,Cesta),nova˙nej(Cesta),
   fail.
nejkr˙cesta(˙,˙,Cesta,Delka) :-
   retract(nej(Cesta,Delka)).

nova˙nej(Cesta) :-
  length(Cesta,Delka0),
  Delka is Delka0-1,
  retract(nej(CestaNej,DelkaNej)),
  ((Delka¡DelkaNej,
   write('Nalezena cesta delky '),write(Delka),nl,
   assert(nej(Cesta,Delka)))
  ;
   assert(nej(CestaNej,DelkaNej))),
  !.          % aby nemohlo byt splneno znova
```

Řešení úloh (I)

Farmář (FARMAR.PL)

```
farmar(DelkaRes) :-
  pocatecni(F,V,K,Z,L),
  zmena˙stavu([stav(F,V,K,Z,L)],Reseni),
  vypis˙reseni(Reseni),length(Reseni,DelkaRes0),
  DelkaRes is DelkaRes0-1.

pocatecni(l,l,l,l,l).
cilovy(p,p,p,p,˙).
zakazany(F,VK,VK,˙,˙) :-      % vlk zere kozu
  F "= VK.
zakazany(F,˙,KZ,KZ,˙) :-      % koza zere zeli
  F "= KZ.

zmena˙stavu([stav(F,V,K,Z,L)—MinStavy],Reseni) :-
  cilovy(F,V,K,Z,L),
  reverse([stav(F,V,K,Z,L)—MinStavy],Reseni).
zmena˙stavu([stav(FL,V,K,Z,FL)—MinStavy],Reseni) :-
  zmena(FL,FL1),
  zmena˙max˙n([V,K,Z],1,[V1,K1,Z1]),
  not(zakazany(FL1,V1,K1,Z1,FL1)),
  not(member(stav(FL1,V1,K1,Z1,FL1),MinStavy)),
  zmena˙stavu([stav(FL1,V1,K1,Z1,FL1),
               stav(FL,V,K,Z,FL)—MinStavy],Reseni).

zmena(l,p).
zmena(p,l).
```

Řešení úloh (II)

Hanojské věže (HANOI.PL)

```
hanoi(N,Delka) :-
  pocatecni(N,A,B,C),zmena_stavu(N,[stav(A,B,C)],Reseni),
  vypis_reseni(Reseni),
  length(Reseni,Delka0),Delka is Delka0-1.

hanoi_nej(N,_) :-
  pocatecni(N,A,B,C),assert(nej([neexistuje],1000000000)),
  zmena_stavu(N,[stav(A,B,C)],Reseni),nove_nej(Reseni),
  fail.
hanoi_nej(_,Delka) :-
  retract(nej(Reseni,Delka)),vypis_reseni(Reseni).

nove_nej(Reseni) :-
  length(Reseni,Delka0),Delka is Delka0-1,
  retract(nej(ReseniNej,DelkaNej)),
  ((Delka¡DelkaNej,
   write('Nalezeno reseni delky '),write(Delka),nl,
   assert(nej(Reseni,Delka)))
  ;
   assert(nej(ReseniNej,DelkaNej))),
  !.          % aby nemohlo byt splneno znova

pocatecni(N,A,[],[]) :- postav_vez(N,A).
cilovy(N,[],[],C) :- postav_vez(N,C).
postav_vez(0,[]) :- !.
postav_vez(N,[N—T]) :- N1 is N-1, postav_vez(N1,T).

zakazany(_,[X,Y—_],_,_) :- X¡Y.
zakazany(_,_,[X,Y—_],_) :- X¡Y.
zakazany(_,_,_,[X,Y—_]) :- X¡Y.

zmena_stavu(N,[stav(A,B,C)—MinStavy],Res) :-
 cilovy(N,A,B,C),!,reverse([stav(A,B,C)—MinStavy],Res).
zmena_stavu(N,[stav(A,B,C)—MinStavy],Res) :-
 zmena(_,_,A,B,C,A1,B1,C1),
 not(zakazany(_,A1,B1,C1)),
 not(member(stav(A1,B1,C1),MinStavy)),
 zmena_stavu(N,[stav(A1,B1,C1),stav(A,B,C)—MinStavy],Res).

zmena('A','B',[HA—TA],B,C,TA,[HA—B],C).    % z A na B
zmena('A','C',[HA—TA],B,C,TA,B,[HA—C]).    % z A na C
zmena('B','A',A,[HB—TB],C,[HB—A],TB,C).    % z B na A
zmena('B','C',A,[HB—TB],C,A,TB,[HB—C]).    % z B na C
zmena('C','A',A,B,[HC—TC],[HC—A],B,TC).    % z C na A
zmena('C','B',A,B,[HC—TC],A,[HC—B],TC).    % z C na B
```

Efektivní algoritmus řešení (HANOI_R.PL)

```
hanoj(N) :-
  presun(N,a,b,c),!.

presun(0,_,_,_) :- !.
presun(N,X,Y,Z) :-
  M is N-1,
  presun(M,X,Z,Y),informuj(X,Y),presun(M,Z,Y,X).

informuj(X,Y) :-
  write('Presun disk z tyce '),write(X),
  write(' na tyc '),write(Y),nl.
```

Deduktivní databáze (I)

```
:- op(1150,xfy,'=¿').

let(1,7:30,praha,8:45,paris).
...

spojeni0(X,CX,Y,CY,[X—Mista],[N—Spoje]) :-    % Prolog
  let(N,CX1,X,CZ,Z),pred(CX,CX1),
  spojeni0(Z,CZ,Y,CY,Mista,Spoje),not(member(X,Mista)).
spojeni0(X,CX,Y,CY,[X,Y],[N]) :-
  let(N,CX1,X,CY1,Y),pred(CX,CX1),pred(CY1,CY).

let(N,CX,X,CZ,Z),spojeni(Z,CZ1,Y,CY,Mista,Spoje),
pred(CZ,CZ1),not(member(X,Mista))
  =¿ spojeni(X,CX,Y,CY,[X—Mista],[N—Spoje]).   % DDB
let(N,CX,X,CY,Y) =¿ spojeni(X,CX,Y,CY,[X,Y],[N]).

dedukce(_) :-
  assert(pocet(0)),(X =¿ Y),X,not(Y),assert(Y),
  retract(pocet(P)),P1 is P+1,assert(pocet(P1)),fail.
dedukce(P) :-
  retract(pocet(P)),write('Vygenerovano '),write(P),
  write(' novych faktu.'),nl.
uplna_dedukce :-  repeat,dedukce(Novych),Novych = 0.
uklid :-
  (_ =¿ Zaver),functor(Zaver,F,A),abolish(F,A),fail.
uklid.

pred(X,X) :- !.
pred(H1:_,H2:_) :- H1 ¡ H2.
pred(H:M1,H:M2) :- M1 ¡ M2.
```

Deduktivní databáze (II)

Prohledávání stavového prostoru do šířky
(na rozdíl od logického programování!)
→ odstranění rizika nekonečných cyklů

Bottom-up mechanismus

Odvozování závěrů pravidel, u nichž jsou splněny předpoklady

Top-down mechanismus

Odvozování odpovědí na zadané dotazy, pomocí metapravidel:

query(X),(Y=¿X),answer(Y) =¿ fact(X).
query(X),(Y=¿X) =¿ query(Y).
query((X,_)) =¿ query(X).
query((X,Y)),answer(X) =¿ query(Y).
query(X),fact(X) =¿ answer(X).
query((X,Y)),answer(X),answer(Y) =¿ answer(X,Y).

Zpracování přirozeného jazyka (I)

Rozklad věty pomocí append - JAZYK1.PL

```
?- sentence([a,man,loved,a,woman]).
?- sentence([the,woman,fell]).

sentence(S) :-
   append(NP,VP,S),
   noun˙phrase(NP),verb˙phrase(VP).
noun˙phrase(NP) :-
   append(A,Subst,NP),
   article(A),substantive(Subst).
verb˙phrase(VP) :-
   verb1(VP).
verb˙phrase(VP) :-
   append(V,VP,NP),verb2(V),noun˙phrase(NP).

article([a]).
article([the]).

substantive([man]).
substantive([woman]).
substantive([pie]).

verb2([loved]).
verb2([ate]).
verb1([fell]).
verb1([ate]).
```

Zpracování přirozeného jazyka (II)

Analýza gramatikou definitních klauzulí (DCG) - JAZYK2.PL, JAZYK3.PL

```
?- sentence([a,man,loved,a,woman],[]).
```

Bez vestavěného mechanismu gramatik

```
sentence(X,Y) :- noun˙phrase(X,Z),verb˙phrase(Z,Y).
noun˙phrase(X,Y) :-  article(X,Z),substantive(Z,Y).
verb˙phrase(X,Y) :-  verb1(X,Y).
verb˙phrase(X,Y) :-  verb2(X,Z),noun˙phrase(Z,Y).

article([a—T],T).
...
substantive([man—T],T).
...
```

S vestavěným mechanismem gramatik

```
sentence --¿ noun˙phrase,verb˙phrase.
noun˙phrase --¿ article,substantive.
verb˙phrase --¿ verb1.
verb˙phrase --¿ verb2,noun˙phrase.

article --¿ [a].
...
substantive --¿ [man].
...
```

Zprac. přirozeného jazyka (III)

Analýza s výstupem - JAZYK4.PL, JAZYK5.PL

```
?- sentence(Subject,Action,[a,man,loved,a,woman],[]).

sentence(Actor,Action,X,Y) :-
 noun˙phrase(Actor,X,Z),animate(Actor),
 verb˙phrase(Action,Z,Y).
noun˙phrase(Subst,X,Y) :-
 article(X,Z),substantive(Subst,Z,Y).
verb˙phrase(Action,X,Y) :- verb1(Action,X,Y).
verb˙phrase(Action,X,Y) :-
 verb2(Action,X,Z),noun˙phrase(˙,Z,Y).

substantive(man,[man—T],T).
...
verb2(loving,[loved—T],T).
...
animate(man).
animate(woman).
```

```
sentence(Actor,Action) --¿
 noun˙phrase(Actor),–animate(Actor)″,verb˙phrase(Action).
noun˙phrase(Subst) --¿ article,substantive(Subst).
verb˙phrase(Action) --¿ verb1(Action).
verb˙phrase(Action) --¿ verb2(Action),noun˙phrase(˙).
...
substantive(man) --¿ [man].
...
verb2(loving) --¿ [loved].
...
```

Zprac. přirozeného jazyka (IV)

Struktura na výstupu - JAZYK6.PL

```
?- sentence(Struc,[a,man,loved,a,woman],[]).
```

Struc = loved(man,woman)

```
sentence(Struc) --¿
   noun˙phrase(Subject),–animate(Subject)″,
   verb˙phrase(Action),–reduce(Action,Subject,Struc)″.
noun˙phrase(Subst) --¿ article,substantive(Subst).
verb˙phrase(Action)--¿ verb1(Action).
verb˙phrase(Action) --¿ verb2(Act0),noun˙phrase(Object),
                –reduce(Act0,Object,Action)″.
...
verb2(X^Y^loved(Y,X))--¿ [loved].
verb2(X^Y^ate(Y,X)) --¿ [ate].
verb1(X^fell(X))--¿ [fell].
verb1(X^ate(X)) --¿ [ate].

reduce(Arg^Expr,Arg,Expr).
...
```

Některé teoretické pojmy logického programování

Upraveno podle *Lloyd, J. W.: Foundations of logic programming. Berlin, Springer-Verlag 1987.*

- Termy, formule a klauzule (→ Logika).
- Substituce a unifikace, unifikační algoritmus.
- Princip vyvracení, rezoluční princip (→ Logika), SLD rezoluce.

Termy, formule a klauzule (I)

Každá abeceda 1.řádu se skládá ze sedmi tříd symbolů:

- proměnné
- predikátové symboly
- funkční symboly
- konstanty (funkční symboly arity 0)
- spojky
- kvantifikátory
- pomocné symboly

Termy, formule a klauzule (II)

Term je jedno z:

- konstanta
- proměnná
- funkční symbol s argumenty (funkční struktura).

Predikátový symbol s argumenty je **atomická formule** (atom).

Literál je buď atomická formule (pozitivní literál), nebo její negace (negativní literál).

$$A \qquad \neg A \quad non\, A \quad \bar{A}$$

Termy, formule a klauzule (III)

Klauzule je formule ve tvaru

$$\forall X_1 \forall X_2 \ldots \forall X_s \quad (L_1 \vee L_2 \vee \ldots \vee L_m)$$

kde každé L_i je literál, a $X_1, X_2, \ldots, X_s$ jsou všechny proměnné v $L_1 \vee L_2 \vee \ldots \vee L_m$ se vyskytující.

Klauzule je i (kvantifikovaný) literál (m=1) a prázdná klauzule (m=0).

Klauzuli

$$\forall X_1 \ldots \forall X_s \quad (A_1 \vee \ldots \vee A_k \vee \neg B_1 \vee \ldots \vee \neg B_n)$$

kde všechny A_i, B_j jsou atomy, a $X_1, \ldots, X_s$ jsou všechny proměnné v těchto atomech se vyskytující, budeme zapisovat jako

$$A_1, \ldots, A_k \leftarrow B_1, \ldots, B_n.$$

Termy, formule a klauzule (IV)

Příklad klauzule (Prolog):

$\mathrm{bratr(X,Y)}$:- $\mathrm{muz(X)}$,$\mathrm{otec(Z,X)}$,$\mathrm{otec(Z,Y)}$.

$\forall$ x,y ((M(x) $\wedge$ ($\exists$ z O(z,x) $\wedge$ O(z,y))) $\Rightarrow$ B(x,y))

$\forall$ x,y $\exists$ z ((M(x) $\wedge$ O(z,x) $\wedge$ O(z,y)) $\Rightarrow$ B(x,y))

$\forall$ x,y $\exists$ z ($\neg$ (M(x) $\wedge$ O(z,x) $\wedge$ O(z,y)) $\vee$ B(x,y))

$\forall$ x,y $\exists$ z $\neg$ M(x) $\vee$ $\neg$ O(z,x) $\vee$ $\neg$ O(z,y) $\vee$ B(x,y)

$\forall$ x,y $\neg$ M(x) $\vee$ $\neg$ O(f(x,y),x) $\vee$ $\neg$ O(f(x,y),y) $\vee$ B(x,y)

Termy, formule a klauzule (V)

Hornova klauzule obsahuje nejvýše jeden pozitivní literál.

Programová (definitní) klauzule obsahuje právě jeden pozitivní literál:

$$A \leftarrow B_1, \ldots, B_n.$$

Cílová klauzule (cíl) obsahuje pouze negativní literály:

$$(false) \leftarrow B_1, \ldots, B_n.$$

Jednotková klauzule (fakt) obsahuje pouze pozitivní literál:

$$A \leftarrow (true).$$

Prázdná klauzule (spor - $\square$) :

$$(false) \leftarrow (true).$$

Substituce a unifikace (I)

Substituce θ je konečná množina tvaru

$$\{u_1/s_1, \ldots, u_m/s_m\}$$

kde každé u_i je proměnná, každé s_i je term různý od u_i, a proměnné $u_1, \ldots, u_m$ jsou navzájem různé
($\rightarrow$ zobrazení z množiny proměnných do množiny termů!).

Výraz je term, literál, nebo konjunkce/disjunkce literálů; **jednoduchý výraz** je term nebo atom.

Nechť E je výraz a $\theta = \{u_1/s_1, \ldots, u_m/s_m\}$ je substituce. Potom $E\theta$, **instance** výrazu E pomocí θ, je výraz, který vznikne z E simultánním nahrazením všech výskytů každé proměnné u_i v E termem s_i.

Příklad: $E = p(X, Y, f(a))$, $\theta = \{X/b, Y/X\}$. Potom $E\theta = p(b, X, f(a))$.

Substituce a unifikace (II)

Nechť $\theta = \{u_1/s_1, \ldots, u_m/s_m\}$
a $\sigma = \{v_1/t_1, \ldots, v_n/t_n\}$ jsou substituce. Potom **složení** $\theta\sigma$ substitucí θ a σ je substituce, která vznikne z množiny

$$\{u_1/s_1\sigma, \ldots, u_m/s_m\sigma, v_1/t_1, \ldots, v_n/t_n\}$$

vynecháním vazeb $u_i/s_i\sigma$, pro něž $u_i = s_i\sigma$
($\rightarrow$ dosazení z σ "vrací zpět" dosazení z θ);
a vazeb v_j/t_j, pro něž $v_j \in \{u_1, \ldots, u_m\}$
($\rightarrow$ dosazení z θ odstraňuje proměnnou, za kterou má dosazovat σ).

Příklad: $\theta = \{X/f(Y), Y/Z\}$, $\sigma = \{X/a, Y/b, Z/Y\}$.
Potom $\theta\sigma = \{X/f(b), Z/Y\}$.

Substituce ϵ určená prázdnou množinou se nazývá **identická substituce**.

Substituce a unifikace (III)

Unifikace (Herbrand 1930):

Nechť S je konečná množina jednoduchých výrazů. Substituce θ se nazývá **unifikátor** pro S, jestliže $S\theta$ je jednoprvková množina.

θ je **nejobecnější unifikátor** pro S, $nu(S)$, jestliže ke každé σ, která je unifikátorem pro S, existuje substituce γ taková, že $\sigma = \theta\gamma$.

Příklad: Pro $S = \{p(f(X), Z), p(Y, a)\}$ je unifikátorem např. $\{Y/f(a), X/a, Z/a\}$;
$nu(S) = \{Y/f(X), Z/a\}$.

Pro každé S je $nu(S)$ určen *jednoznačně*, až na přejmenování proměnných.

Substituce a unifikace (IV)

Množina neshod v konečné množině jednoduchých výrazů S:

Nalezněte první pozici zleva, v níž existuje neshoda, a vyčleňte z každého výrazu v S podvýraz začínající symbolem na této pozici. Množina všech takových podvýrazů je množina neshod.

Příklad: Pro $S = \{p(f(X), h(Y), a),$
$p(f(X), Z, a),$
$p(f(X), h(Y), b)\}$
je množinou neshod $\{h(Y), Z\}$.

Substituce a unifikace (V)

Unifikační algoritmus pro konečnou množinu jednoduchých výrazů S:

1. Polož $k = 0$, $\sigma_0 = \epsilon$.
2. Jestliže je $S\sigma_k$ jednoprvková množina, pak STOP; σ_k je $nu(S)$.
   Jinak nechť D_k je množina neshod v $S\sigma_k$.
3. Jestliže D_k obsahuje taková v a t, že v je proměnná *nevyskytující se* v t, polož $\sigma_{k+1} = \sigma_k\{v/t\}$, zvětši k o 1, a jdi na 2.
   Jinak STOP; S není unifikovatelná.

Unifikační teorém: Jestliže S je unifikovatelná, pak se unifikační algoritmus zastaví a vydá $nu(S)$. Jestliže S není unifikovatelná, pak se rovněž zastaví a vydá o tom zprávu.

Substituce a unifikace (VI)

Příklad: Nechť $S = \{p(f(a), g(X)), p(Y, Y)\}$.

0. $\sigma_0 = \epsilon$.
1. $D_0 = \{f(a), Y\}$, $\sigma_1 = \{Y/f(a)\}$,
   $S\sigma_1 = \{p(f(a), g(X)), p(f(a), f(a))\}$.
2. $D_1 = \{g(X), f(a)\}$.
   Tudíž S není unifikovatelná.

Příklad: Nechť $S = \{p(a, X), p(Y, h(Y))\}$.

0. $\sigma_0 = \epsilon$.
1. $D_0 = \{a, Y\}$, $\sigma_1 = \{Y/a\}$,
   $S\sigma_1 = \{p(a, X), p(a, h(a))\}$.
2. $D_1 = \{X, h(a)\}$, $\sigma_2 = \{Y/a, X/h(a)\}$,
   $S\sigma_2 = \{p(a, h(a)), p(a, h(a))\} = \{p(a, h(a))\}$.
   Tudíž S je unifikovatelná a σ_2 je $nu(S)$.

Substituce a unifikace (VII)

Příklad: Nechť $S = \{p(X, X), p(Y, f(Y))\}$.

0. $\sigma_0 = \epsilon$.
1. $D_0 = \{X, Y\}$, $\sigma_1 = \{X/Y\}$,
   $S\sigma_1 = \{p(Y, Y), p(Y, f(Y))\}$.
2. $D_1 = \{Y, f(Y)\}$.
   Protože se Y vyskytuje v $f(Y)$, S není unifikovatelná!

Kontrola výskytu ("occur-check") v unifikačním algoritmu ovšem může (v nejhorším případě) vést na exponenciální složitost vzhledem k délce vstupu!

→ v implementacích Prologu se většinou vypouští
→ možnost "unifikačního" nekonečného cyklu!
Např.:

?- X = f(X).

Rezoluce a vyvracení (I)

Rezoluční princip: odvozovací pravidlo se schématem

$$\frac{X \vee Z, \;\; Y \vee \neg Z}{X \vee Y}$$

V logickém programování je Z zpravidla atom; $\{Z, \neg Z\}$ se nazývá **doplňkový pár** literálů; klauzule $X \vee Z$ a $Y \vee \neg Z$ se nazývají **rodičovský pár**, a $X \vee Y$ je jejich **rezolventa**.
Rezoluce je **korektní** - rezolventou dvou pravdivých klauzulí je opět pravdivá klauzule.

Princip vyvracení: Uzavřená formule ϕ vyplývá z teorie T právě tehdy když $T \cup \{\neg\phi\}$ je nesplnitelná (kontradiktorická).

Rezoluce a vyvracení (II)

Příklad: Chceme dokázat, že formule u vyplývá z teorie

$$T = \{v \Rightarrow u, \; w \Rightarrow v, \; w\}$$

Vyvracení: Převedeme T na klauzulární tvar, a snažíme se dokázat spornost teorie

$$T' = \{\neg v \vee u, \; \neg w \vee v, \; w \vee \mathbf{false}, \; \neg u \vee \mathbf{false}\}$$

(ke konjunkci lze vždy přidat **true** a k disjunkci **false**!)

Rezoluce: Rezolvujeme např. $\neg v \vee u$, $\neg u \vee \mathbf{false}$ na $\neg v \vee \mathbf{false}$:

$$T'' = \{\neg w \vee v, \; w \vee \mathbf{false}, \; \neg v \vee \mathbf{false}\}$$

(rezolventu umístíme na konec seznamu klauzulí)

a dále postupně:

$$T''' = \{w \vee \mathbf{false}, \neg w \vee \mathbf{false}\}$$

$$T'''' = \{\mathbf{false}\}$$

Odvodili jsme spor a tím dokázali, že $T \models u$.

Rezoluce a vyvracení (III)

Herbrandova věta: Množina formulí v klauzulárním tvaru je nesplnitelná právě tehdy, když existuje *konečná* množina jejich *základních* instancí, která je logicky sporná.
→ při rezoluci lze používat (unifikační) substituce.

SLD rezoluce (v Prologu): rezoluce na definitních klauzulích, s výběrovou (selection) funkcí, s lineární strategií.

S → pracuje se vždy s 1 rodičovským párem

L → jedním z členů páru je rezolventa z minulého kroku (→ prohledávání do hloubky); výběr první unifikovatelné klauzule z databáze

D → doplňkovým literálem druhého členu je jeho hlava

SLD rezoluce je *korektní*

$$\forall F, G \quad F \models G \quad if \quad F \vdash_{SLD} G$$

ale není úplná (problém nekonečného cyklu).

Rezoluce a vyvracení (IV)

Příklad: dotaz ?-syn(X,Y) a databáze
syn(X,Y) :- otec(Y,X),muz(X).
muz(karel˙IV).
otec(jan˙lucembursky,karel˙IV).

$$\{s(X_1, Y_1) \vee \overline{o(Y_1, X_1)} \vee \overline{m(X_1)},\ m(k),\ o(j, k)\} \cup \{\overline{s(k, X_0)}\}$$

$$\theta_0 = \{X_1/k,\ Y_1/X_0\}$$

$$\{s(k, X_0) \vee \overline{o(X_0, k)} \vee \overline{m(k)},\ m(k),\ o(j, k),\ \overline{s(k, X_0)}\}$$

$$\{\overline{o(X_0, k)} \vee \overline{m(k)},\ m(k),\ o(j, k)\}$$

$$\theta_1 = \{X_0/j\}$$

$$\{\overline{o(j, k)} \vee \overline{m(k)},\ m(k),\ o(j, k)\}$$

$$\{\overline{m(k)},\ m(k)\}$$

$$\square$$

Induktivní logické programování

Verze zima 1998

Induktivní logické programování

Upraveno podle *Lavrač, N. - Džeroski, S.: Inductive Logic Programming. Ellis Horwood 1994.*

ILP lze chápat v kontextu několika disciplín...

Logické programování : ILP je automatické *generování programů* (intenzionálních definic predikátů) z příkladů (extenzionálních definic).

Strojové učení (ML): ILP je strojové učení v *jazyce 1.řádu*.

Získávání znalostí z databází (KDD): ILP je získávání znalostí z relačních databází *s několika propojenými tabulkami*.

ILP – příklad

Hledáme intenzionální definici predikátu dcera/2 na základě **příkladů**

dcera(marketa,eliska_premyslovna).	$\oplus$
dcera(katerina,karel_IV).	$\oplus$
dcera(karel_IV,eliska_premyslovna).	$\ominus$
dcera(katerina,eliska_premyslovna).	$\ominus$

a **apriorních znalostí**

rodic(eliska_premyslovna,marketa).
rodic(eliska_premyslovna,karel_IV).
rodic(karel_IV,katerina).
rodic(karel_IV,vaclav_IV).
zena(eliska_premyslovna).
zena(marketa).
zena(katerina).

Správný tvar je:

dcera(X,Y) :- zena(X), rodic(Y,X).

ILP – obecný tvar úlohy

Dáno:

- množina *trénovacích příkladů* $\mathcal{E}$, složená z pozitivních příkladů $\mathcal{E}^+$ a z negativních příkladů $\mathcal{E}^-$; všechny příklady jsou základními fakty neznámého predikátu p;
- jazyk $\mathcal{L}$, určující *syntaktická omezení*, která musí definice predikátu p splňovat;
- množina *apriorních znalostí* ("background knowledge") $\mathcal{B}$, tj. množina definic predikátů q_i (různých od p); tyto predikáty mohou být využity v definici p.

Hledáme:

definici $\mathcal{H}$ pro predikát p, vyjádřenou pomocí $\mathcal{L}$; $\mathcal{H}$ musí být *úplná* a *konzistentní* vzhledem k příkladům $\mathcal{E}$ a apriorním znalostem $\mathcal{B}$.

ILP – pokrytí, úplnost, konzistence

Hypotéza $\mathcal{H}$ **pokrývá** příklad $e \in \mathcal{E}$ vzhledem k apriorním znalostem $\mathcal{B}$, jestliže $\mathcal{B} \cup \mathcal{H} \models e$ (tj. pravdivost příkladu logicky vyplývá ze současné pravdivosti hypotézy a apriorních znalostí).

Hypotéza $\mathcal{H}$ je **úplná** vzhledem k množině pozitivních a negativních příkladů $\mathcal{E} = \mathcal{E}^+ \cup \mathcal{E}^-$ a apriorním znalostem $\mathcal{B}$, jestliže pokrývá všechny pozitivní příklady $e \in \mathcal{E}^+$ vzhledem k apriorním znalostem $\mathcal{B}$.

Hypotéza $\mathcal{H}$ je **konzistentní** vzhledem k množině pozitivních a negativních příkladů $\mathcal{E} = \mathcal{E}^+ \cup \mathcal{E}^-$ a apriorním znalostem $\mathcal{B}$, jestliže nepokrývá žádný z negativních příkladů $e \in \mathcal{E}^+$ vzhledem k apriorním znalostem $\mathcal{B}$.

θ-subsumpce (Plotkin 1969)

Nechť c a c' jsou dvě programové klauzule. Klauzule c θ-**zahrnuje** klauzuli c' jestliže existuje substituce θ taková, že $c\theta \subseteq c'$.

θ-subsumpce jako relace **generalizace**: jestliže c θ-zahrnuje c', pak c je generalizací c' (tj. c je alespoň tak obecná jako c').

Vlastnosti θ-subsumpce:

- jestliže c θ-zahrnuje c', pak $c \models c'$;
- θ-subsumpce vytváří na množině *redukovaných* klauzulí (generalizační) *svaz*; tudíž každé dvě klauzule mají jednoznačně určenou *nejmenší horní mez* a *největší dolní mez*.

θ-subsumpce – příklad

Klauzule

$$\begin{aligned} c &= dcera(X,Y) \leftarrow rodic(Y,X) \\ &= \{dcera(X,Y), \overline{rodic(Y,X)}\} \end{aligned}$$

θ-zahrnuje klauzule:

$$\begin{aligned} c' &= dcera(X,Y) \leftarrow zena(X), rodic(Y,X) = \\ &= \{dcera(X,Y), \overline{zena(X)}, \overline{rodic(Y,X)}\} \end{aligned}$$

za identické substituce $\theta = \emptyset$;

$$c'' = dcera(X,X) \leftarrow zena(X), rodic(X,X)$$

za substituce $\theta = \{Y/X\}$;

$$\begin{aligned} c''' = \; & dcera(marketa, eliska) \leftarrow zena(marketa), \\ & rodic(eliska, marketa), rodic(eliska, karel) \end{aligned}$$

za substituce $\theta = \{X/marketa, Y/eliska\}$;

$$c'''' = dcera(X,Y) \leftarrow rodic(Y,X), rodic(W,V)$$

opět za identické substituce.

c a c'''' se θ-zahrnují *vzájemně*, protože naopak $c''''\theta$-zahrnuje c za substituce $\theta = \{W/Y,\ V/X\}$!

Nejméně obecná generalizace (I)

Nejmenší horní mez dvou redukovaných klauzulí c a c' ve svazu daném θ-subsumpcí se nazývá **nejméně obecná generalizace** ("least-general generalization"), $lgg(c, c')$.

Výpočet lgg:

lgg **termů:**

- pro dva identické termy: $lgg(t, t) = t$;
- lgg dvou různých konstant nebo struktur s různým funktorem nebo aritou je *proměnná*;
- lgg dvou struktur se stejným funktorem a aritou je struktura, jejíž argumenty jsou lgg dvojic argumentů, které mají v původních strukturách stejné pořadí;

lgg stejných podvýrazů (termů, atomů i literálů) se nahrazují stejnou proměnnou!

Příklady:

$$lgg([a, b, c], [a, c, d]) = [a, X, Y]$$

$$lgg(f(a, b), f(b, a)) = f(V, V)$$

Nejméně obecná generalizace (II)

lgg **atomů:**

- lgg dvou atomů se stejným predikátovým symbolem a aritou je atom, jehož argumenty jsou lgg dvojic argumentů, které mají v původních atomech stejné pořadí;
- lgg dvou atomů s různým predikátovým symbolem nebo aritou *není definována*;

lgg **literálů:**

- lgg dvojice pozitivních literálů - viz lgg atomů;
- lgg dvojice negativních literálů je negace lgg jejich atomů;
- lgg pozitivního a negativního literálu *není definována*;

Příklady:

$$lgg(rodic(eliska, marketa), rodic(eliska, karel)) = \\ = rodic(eliska, X)$$

$$lgg(rodic(eliska, marketa), \overline{rodic(eliska, karel)})$$
není definováno

Nejméně obecná generalizace (III)

lgg **klauzulí** je klauzule, jejímiž literály jsou lgg *všech dvojic* literálů z původních klauzulí, pro které je lgg definována.

Příklady:

Nechť

$$\begin{aligned} c_1 &= dcera(marketa, eliska) \leftarrow \\ &\quad zena(marketa), rodic(eliska, marketa) \\ c_2 &= dcera(katerina, karel) \leftarrow \\ &\quad zena(katerina), rodic(karel, katerina) \end{aligned}$$

Potom

$$lgg(c_1, c_2) = dcera(X, Y) \leftarrow zena(X), rodic(Y, X)$$

kde

$$X = lgg(marie, katerina) \quad Y = lgg(eliska, karel)$$

Nejméně obecná generalizace (IV)

Nechť

$$\begin{aligned} c_1 &= deda(jan, vaclav) \leftarrow \\ &\quad otec(jan, karel), otec(karel, vaclav) \\ c_2 &= deda(jan, jost) \leftarrow \\ &\quad otec(jan, jan\_jindrich), otec(jan\_jindrich, jost) \end{aligned}$$

Potom

$$\begin{aligned} lgg(c_1, c_2) &= deda(jan, X) \leftarrow otec(jan, Y), otec(Z, V), \\ &\quad otec(W, T), otec(Y, X) \end{aligned}$$

kde

$$\begin{array}{ll} X = lgg(vaclav, jost) & Y = lgg(karel, jan\_jindrich) \\ Z = lgg(jan, jan\_jindrich) & V = lgg(karel, jost) \\ W = lgg(karel, jan) & T = lgg(vaclav, jan\_jindrich) \end{array}$$

?Ale jak aplikovat lgg na původní úlohu:

- příklady ve formě základních atomů
- apriorní znalosti

Relativní lgg

Mějme dva příklady (základní atomy) e_1 a e_2, a množinu apriorních znalostí (základních faktů) $\mathcal{B}$. **Relativní nejméně obecná generalizace** e_1 a e_2 vzhledem k $\mathcal{B}$ je

$$\begin{aligned} rlgg(e_1, e_2, \mathcal{B}) &= lgg((e_1 \leftarrow c(\mathcal{B})), (e_2 \leftarrow c(\mathcal{B}))) = \\ &= lgg(e_1, e_2) \leftarrow lgg(c(\mathcal{B}), c(\mathcal{B})) \end{aligned}$$

kde $c(\mathcal{B})$ označuje konjunkci všech faktů z $\mathcal{B}$.

Problém: vznik nadbytečných literálů

V původním příkladu:

$$rlgg(e_1, e_2, \mathcal{B}) =$$

$$\mathbf{dcera}(\mathbf{X}, \mathbf{Y}) \leftarrow rodic(eliska, marketa), rodic(eliska, karel), rodic(karel, katerina), rodic(karel, vaclav), zena(eliska), zena(marketa), zena(katerina), rodic(eliska, Z), \mathbf{rodic}(\mathbf{Y}, \mathbf{X}), rodic(Y, Z), rodic(Y, V), rodic(Y, W), rodic(karel, T), zena(U), zena(S), \mathbf{zena}(\mathbf{X})$$

kde

$X = lgg(marketa, katerina)$ $\quad Y = lgg(eliska, karel)$
$Z = lgg(marketa, karel)$ $\quad V = lgg(marketa, vaclav)$
$W = lgg(karel, vaclav)$ $\quad T = lgg(katerina, vaclav)$
$U = lgg(eliska, marketa)$ $\quad S = lgg(eliska, katerina)$

Obrácení rezoluce (I)

Z klauzulí

$$\begin{aligned} b_1 &= zena(marketa) \\ b_2 &= rodic(eliska, marketa) \\ c &= dcera(X, Y) \leftarrow zena(X), rodic(Y, X) \end{aligned}$$

lze rezolučním principem s využitím substituce

$$\theta_1 = \{X/marketa\}$$

odvodit

$$\begin{aligned} res(b_1, c) &= dcera(marketa, Y) \\ &\leftarrow rodic(Y, marketa) \end{aligned}$$

a dále s využitím substituce

$$\theta_2 = \{Y/eliska\}$$

odvodit

$$res(b_2, res(b_1, c)) = dcera(marketa, eliska)$$

Obrácení rezoluce (II)

Tento postup lze obrátit: chceme nalézt hypotézu, která by společně s apriorními znalostmi

$$\begin{aligned} b_1 &= zena(marketa) \\ b_2 &= rodic(eliska, marketa) \end{aligned}$$

umožňovala odvodit příklad

$$e = dcera(marketa, eliska)$$

Pomocí **obrácené rezoluce** (Muggleton 1988) s využitím obrácené substituce

$$\theta_1^{-1} = \{eliska/Y\}$$

nalezneme hypotézu

$$c_1 = dcera(marketa, Y) \leftarrow rodic(Y, marketa)$$

a tu dále s využitím obrácené substituce

$$\theta_2^{-1} = \{marketa/X\}$$

zobecníme na

$$c = dcera(X, Y) \leftarrow zena(X), rodic(Y, X)$$

Obrácení rezoluce (III)

Problémy: obrácená rezoluce

- na rozdíl od rezoluce *není korektní*
  to platí obecně pro *induktivní* odvozovací pravidla
- je *nedeterministická*
  - výběr klauzulí
  - přiřazení proměnných při obrácené substituci

Příklad: chceme nalézt hypotézu, která by společně s apriorní znalostí $b = syn(vaclav, karel)$ umožňovala odvodit příklad $e = kral(vaclav)$. ?Jak zvolit mezi obrácenými substitucemi

$\theta_1^{-1} = \emptyset$
$res_1^{-1}(b, e) = kral(vaclav) \leftarrow syn(vaclav, karel)$

$\theta_2^{-1} = \{vaclav/X\}$
$res_2^{-1}(b, e) = kral(X) \leftarrow syn(X, karel)$

$\theta_3^{-1} = \{vaclav/X, karel/Y\}$
$res_3^{-1}(b, e) = kral(X) \leftarrow syn(X, Y)$

Specializační techniky (I)

Prohledávání stavového prostoru hypotéz (θ-subsumpčního svazu):

- **Generalizační** (bottom-up) techniky postupují zdola od příkladů;
  (nejméně obecná generalizace; obrácená rezoluce)

- **Specializační** (top-down) techniky postupují shora od nejobecnější definice cílového predikátu, a postupně ji zjemňují přidáváním literálů do těla (ev. substitucí za proměnné); typicky se přidávají literály

  – predikátů z apriorních znalostí;

  – elementárních predikátů (např. rovnost)

  s proměnnými z hlavy klauzule, nebo nejvýše 1 novou proměnnou.

a opět specializujeme. Máme k dispozici možnosti

$$c_{11} = dcera(X,Y) \leftarrow zena(X), rodic(Y,X)$$
$$c_{12} = dcera(X,Y) \leftarrow zena(X), rodic(Z,X)$$
$$c_{13} = dcera(X,Y) \leftarrow zena(X), rodic(Y,Z)$$

ovšem pouze c_{11} vyloučí zbylý $\ominus$. Výsledná úplná a konzistentní hypotéza tudíž je

$$H_2 = \{dcera(X,Y) \leftarrow zena(X), rodic(Y,X)\}$$

Specializační metody se používají zvláště při induktivním učení *z dat zatížených šumem* - pak se nepožaduje dokonalá úplnost a konzistence (namísto toho kritéria z teorie informace - entropie, informační zisk...).

Specializační techniky (II)

V příkladu začneme s nepodmíněnou klauzulí:

$$H_0 = \{c = dcera(X,Y) \leftarrow\}$$

Hypotéza pokrývá oba $\oplus$, ovšem i oba $\ominus$:

$$\begin{array}{ll} dcera(marketa, eliska) & \oplus \\ dcera(katerina, karel) & \oplus \\ dcera(karel, eliska) & \ominus \\ dcera(katerina, eliska) & \ominus \end{array}$$

Specializovat lze např. přidáním některého z literálů:

- $X = Y$, $zena(X)$, $zena(Y)$, $rodic(X,X)$, $rodic(X,Y)$, $rodic(Y,X)$, $rodic(Y,Y)$;
- $rodic(X,Z)$, $rodic(Z,X)$, $rodic(Y,Z)$, $rodic(Z,Y)$.

Z povolených minimálních specializací pouze

$$c_1 = dcera(X,Y) \leftarrow zena(X)$$
$$c_2 = dcera(X,Y) \leftarrow rodic(Y,X)$$
$$c_3 = dcera(X,Y) \leftarrow rodic(Z,X)$$
$$c_4 = dcera(X,Y) \leftarrow rodic(Y,Z)$$

stále pokrývají oba $\oplus$; c_3 a c_4 ovšem opět pokrývají oba $\ominus$, zatímco c_1 a c_2 pokrývají vždy jen jediný z $\ominus$. Zvolíme tedy např.

$$H_1 = \{c_1 = dcera(X,Y) \leftarrow zena(X)\}$$

ILP a relační databáze

Paralelní implementace LP

Zpracování se s pomocí rozvrhovacích algoritmů rozdělí mezi paralelní procesy. Základní varianty:

AND-paralelizace. Cíle, které jsou v konjunkci, se zpracovávají paralelně; pokud některý z nich neuspěje, celá konjunkce skončí neúspěchem.

- pokud paralelně zpracovávané cíle sdílejí proměnné, je nutná vzájemná synchronizace procesů
- aby bylo zřejmé, které výskyty proměnných jsou vstupní a které výstupní, jsou zpravidla uváděny V/V módy

OR-paralelizace. Cíle, které jsou v disjunkci, se zpracovávají paralelně; pokud některý z nich uspěje, celá konjunkce skončí úspěchem.

- totéž platí pro situaci, kdy je s daným cílem unifikovatelných několik klauzulí
- často se zavádí "oboustranná verze" řezu

Na nižších úrovních výpočetního stromu se může paralelizace opakovat.

Je nutno ošetřit predikáty s "vedlejším efektem" (např. vstupně/výstupní), vyskytující se v alternativních větvích

LP s omezujícími podmínkami (I)

(Constrained LP – CLP)

Podle *Mařík - Štěpánková - Lažanský a kol.: Umělá inteligence (2). Praha, Academia 1997.*

V tradičním LP lze numerické ne/rovnosti ověřovat jen pro konkretizované proměnné – nelze tedy např.:

```
?- X+Y =:= 5.

X=0
Y=5
;

X=1
Y=4
;
...
```

Rozšíření o algoritmy řešení omezujících podmínek umožňuje zadat omezení, a vyhovující hodnoty proměnných se následně vypočtou.

Cíle typu "omezení" lze v programech kombinovat s logickými cíli.

Základní varianty CLP:

Nad reálnými čísly: řešení soustavy lineárních rovnic (např. simplexovou metodou).

Nad celými čísly: iterativní propagace mezí domén; "zkusmé" dosazování hodnot s testováním.

LP s omezujícími podmínkami (II)

Příklad "brouci a pavouci v krabici":

```
hmyz(Brouci, Pavouci, Nohy) :-
    Brouci #> 0, Pavouci #> 0,
    8*Pavouci + 6*Brouci #= Nohy,
    indomain(Pavouci).

?- hmyz(P,B,46).

P = 5
B = 1
```

- První dvě omezení (každé s jedinou proměnnou) nastaví meze domén pro P i B na [0..1000000].
- Iterativní propagace mezí s využitím třetího omezení zmenší velikost domén na [2..5] pro P a [1..5] pro B.
- Protože ještě není nalezeno jednoznačné řešení, volá se cíl `indomain`, který dosazuje za `Pavouci` postupně hodnoty z domény (počínaje od nejmenší), až nalezne řešení, které třetí omezení splňuje.

Termíny (I)

Česky	Anglicky
klauzule	clause
fakt	fact
pravidlo	rule
dotaz	query
hlava klauzule	clause head
tělo klauzule	clause body
atomická formule (atom)	atomic formula
predikátový symbol	predicate symbol
argument	argument
arita	arity
definice predikátu	predicate definition
relace	relation
konstanta	constant
proměnná	variable
základní klauzule	**ground** clause
databáze	database
definitní klauzule	definite clause
vestavěný predikát	built-in predicate
cíl	goal
uspět (o cíli)	succeed
neuspět (o cíli)	fail
konkretizace	**instantiation**
navracení	**backtracking**
anonymní proměnná	anonymous variable
předpoklad uzavřeného světa	closed-world assumption

Termíny (II)

Česky	Anglicky
extenzionální (definice)	extensional
intenzionální (definice)	intensional
AND/OR strom	AND/OR tree
volná proměnná	free variable
konkretizovaná proměnná	**instantiated** variable
vazba (proměnné)	binding (of a variable)
instance (klauzule)	instance
unifikace	unification
substituce	substitution
rekurze	recursion
invertibilita	invertibility
blokový model	**box model**
brána cíle	**port** of a goal
řez	cut
negace neúspěchem	negation as failure
term	term
atomický term	atomic term
složený term	compound term
atom	atom
funktor	functor
operátor	operator
identita (shoda) termů	term equivalence
standardní pořadí	standard order
aritmetická funkce	arithmetic function
seznam	list
plochý seznam	flat list
grupování termů	term grouping

Termíny (III)

Česky	Anglicky
(jednoduchý) výraz	(simple) expression
instance (výrazu)	instance (of expression)
substituce	substitution
složení substitucí	composition of subst.
identická substituce	identity substitution
(nejobecnější) unifikátor	(most general) unifier
množina neshod	disagreement set
test výskytu	**occur check**
princip vyvracení	**refutation** principle
(SLD) rezoluce	(SLD) resolution
doplňkový pár	complementary pair
rodičovský pár	parental pair
rezolventa	resolvent
apriorní znalosti	**background** knowledge
pokrytí	covering
úplnost	completeness
konzistence	consistency
θ-subsumpce (-zahrnování)	θ-subsumption
generalizace	generalization
nejméně obecná gen.	least general gen.
obrácená rezoluce	inverse resolution
specializační graf	**refinement** graph

Standardní vestavěné predikáty

(ve většině implementací)

halt/0, consult/1, listing/0/1/2
fail/0, true/0, repeat/0
trace/0
call/1, !/0, not/1
assert/1, retract/1, clause/2
op/3
var/1, nonvar/1, atom/1, string/1,
integer/1, float/1, number/1, atomic/1
name/2, functor/3, arg/3
findall/3, bagof/3, setof/3
write/1, read/1, nl/0, tab/1

a binární operátory

=:= =“= ¡ ¿ ¿= =¡ is
= “= == “==
=.. (”univ”)