

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

Prolog: historie a současnost

- Rozvoj začíná po roce 1970
 - Robert Kowalski – teoretické základy
 - Alain Colmerauer, David Warren (*Warren Abstract Machine*) – implementace
 - pozdější rozšíření Prologu o logické programování s omezujícími podmínkami
 - Prolog v současnosti
 - zavedené aplikační oblasti, nutnost přidání inteligence
 - hypotéky; pediatrický sw; konfigurace a pravidla pro stanovení ceny objednávky; testovací nástroje, modelové testování; ...
 - náhrada procedurálního kódu Prologem vede k
 - desetinásobnému zmenšení kódu, řádově menšímu času na vývoj, jednodušší údržbě
 - efektivita Prologu?
 - zrychlení počítačů + výrazné zvětšení nároků sw
- ⇒ ve prospěch kompaktnosti i rychlosti Prologu

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:** množina 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 vrácená 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

- pozitivní – **dotaz je splnitelný a uspěl**
- negativní – **dotaz je nesplnitelný a neuspěl**

Proměnné jsou během výpočtu **instanciovány** (= nahrazeny objekty)

- ?- clovek(novak, 18, Prace).
- 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 skládá 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

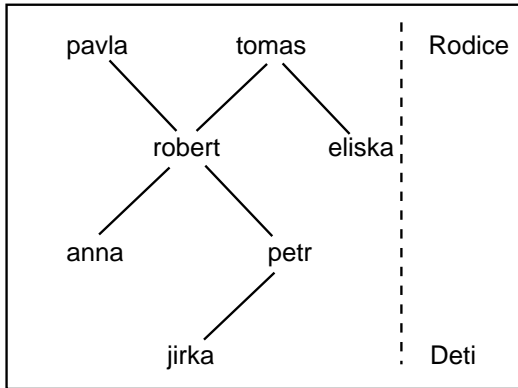
```
predek( X, Z ) :- rodic( X, Z ).           % (1)
```

```
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 ).
```

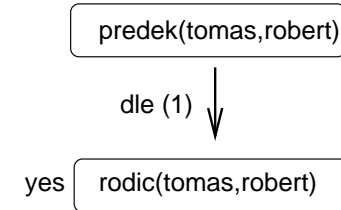


```
predek( X, Z ) :- rodic( X, Z ).      % (1)
```

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

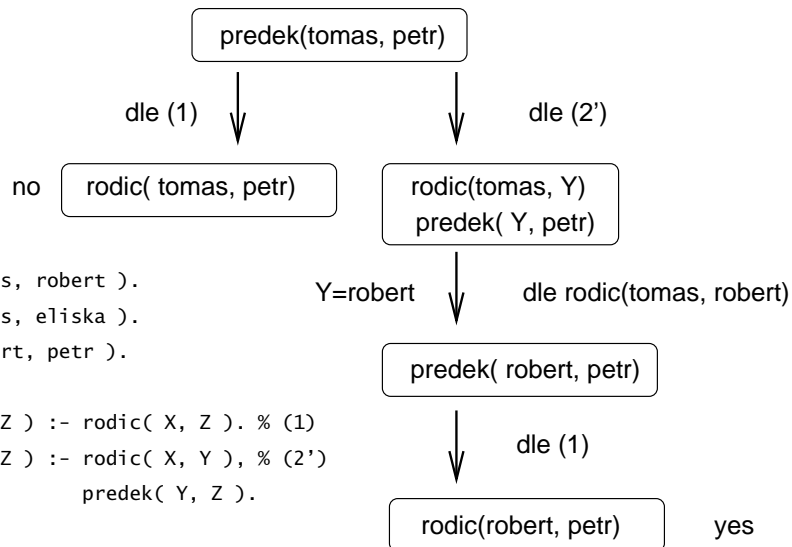
Výpočet odpovědi na dotaz ?- predek(tomas,robert)

```
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)

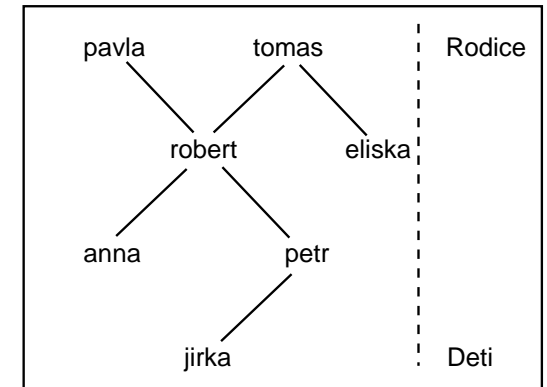


```
rodic( tomas, robert ).
rodic( tomas, eliska ).
rodic( robert, petr ).
```

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

Odpověď na dotaz ?- predek(robert, Potomek)

```
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(robert, Potomek) --> ???

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 \Rightarrow význam mají logické relace
- Procedurální: **Jak** vypočítáme výstup programu?
 - p vyřešíme tak, že **nejprve** vyřešíme q a **pak** r \Rightarrow kromě logických relací je významné i pořadí cílů
- výstup
 - indikátor yes/no určující, zda byly cíle splněny
 - instancie proměnných v případě splnění cílů

Deklarativní význam programu

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.

Instance klauzule: proměnné v klauzuli jsou substituovány termem

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

Konjunkce ";" vs. disjunkce ";" cílů

- **Konjunkce** = nutné splnění **všech cílů**
 - $p :- q, r.$
- **Disjunkce** = stačí splnění **libovolného cíle**
 - $p :- q; r.$ $p :- q.$
 - $p :- r.$
 - priorita středníku je vyšší:
 - $p :- q, r; s, t, u.$
 - $p :- (q, r) ; (s, t, u).$
 - $p :- q, r.$
 - $p :- s, t, u.$

Typy operátorů

Typy operátorů

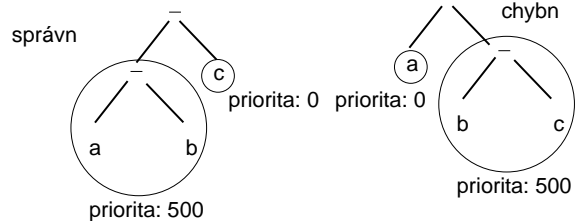
- infixové operátory: xfx , xfy , yfx
- prefixové operátory: fx , fy
- postfixové operátory: xf , yf

př. $xfx = yfx$

př. $fx ? fy$

x a y určují prioritu argumentu

- x reprezentuje argument, jehož priorita musí být **striktně menší** než u operátoru
- y reprezentuje argument, jehož priorita je **menší nebo rovna** operátoru
- a-b-c odpovídá $(a-b)-c$ a ne $a-(b-c)$: „-“ odpovídá yfx



Aritmetika

Předdefinované operátory

$+$, $-$, $*$, $/$, $**$ mocnina, $//$ celočíselné dělení, mod zbytek po dělení

?- $X = 1 + 2$. $X = 1 + 2$ = odpovídá unifikaci

?- $X \text{ is } 1 + 2$.

$X = 3$ „is“ je speciální předdefinovaný operátor, který vynutí evaluaci

porovnej: $N = (1+1+1+1)$ $N \text{ is } (1+1+1+1)$

pravá strana musí být vyhodnotitelný výraz (bez proměnné)

volání $?- X \text{ is } Y + 1$. způsobí chybu

Další speciální předdefinované operátory

$>$, $<$, $>=$, $=<$, $:=$ aritmetická rovnost, $=\backslash=$ aritmetická nerovnost

porovnej: $1+2 := 2+1$ $1+2 = 2+1$

obě strany musí být vyhodnotitelný výraz: volání $?- 1 < A + 2$. způsobí chybu

Různé typy rovností a porovnání

$X = Y$ X a Y jsou unifikovatelné

$X \backslash= Y$ X a Y nejsou unifikovatelné, (také $\backslash+ X = Y$)

$X == Y$ X a Y jsou identické

porovnej: $?- A == B$... no $?- A=B, A==B$... $B = A$ yes

$X \backslash== Y$ X a Y nejsou identické

porovnej: $?- A \backslash== B$... yes $?- A=B, A \backslash== B$... A no

$X \text{ is } Y$ Y je aritmeticky vyhodnoceno a výsledek je přiřazen X

$X := Y$ X a Y jsou si aritmeticky rovny

$X =\backslash= Y$ X a Y si aritmeticky nejsou rovny

$X < Y$ aritmetická hodnota X je menší než Y ($=<$, $>$, $>=$)

$X @< Y$ term X předchází term Y ($@<$, $@>$, $@>=$)

1. porovnání termů: podle abecedního n. aritmetického uspořádání

2. porovnání struktur: podle arity, pak hlavního funktoru a pak

zleva podle argumentů

?- $f(\text{pavel}, g(b)) @< f(\text{pavel}, h(a))$ yes

Prolog: příklady

Příklad: 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.

Příklad: věž z kostek

Příklad: postavte věž zadané velikosti ze tří různě velkých kostek tak, že kostka smí ležet pouze na větší kostce.

```
kostka(mala). kostka(stredni). kostka(velka).
```

```
vetsi(zeme,velka). vetsi(zeme,stredni). vetsi(zeme,mala).  
vetsi(velka,stredni). vetsi(velka,mala).  
vetsi(stredni,mala).
```

```
% ?- postav_vez(vez(zeme,0), vez(Kostka,0)).
```

```
% ?- postav_vez(vez(zeme,0), vez(Kostka,3)).
```

```
postav_vez( Vez, Vez ).
```

```
postav_vez( Vstup, Vystup ) :- pridej_kostku( Vstup, Pridani ),  
                             postav_vez( Pridani, Vystup ).
```

```
pridej_kostku( Vstup, Pridani ) :- Vstup = vez( Vrchol, Vyska ),  
                                  kostka( Kostka ),  
                                  vetsi( Vrchol, Kostka ),  
                                  NovaVyska is Vyska + 1,  
                                  Pridani = vez( Kostka, NovaVyska ).
```