

## 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

## Program = fakta + pravidla

### 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>.

- **(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,

X studuje, jestliže

X je student

pro každé X,

X je student, potom

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 vrácená hodnota)

### Psaní komentářů

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

## 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 ).`

## 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 ).`  
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 ";"
```

## Rekurzivní pravidla

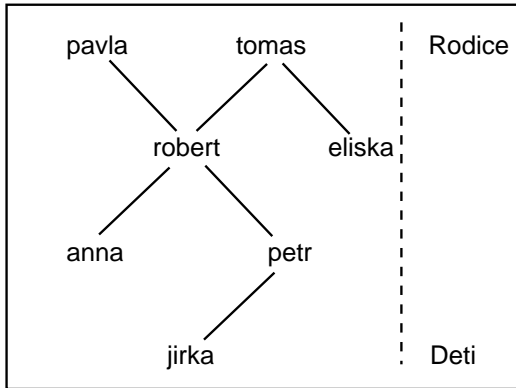
```
predek( X, Z ) :- rodic( X, Z ).      % (1)
```

```
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 ).
```

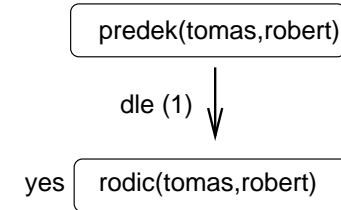


```
predek( X, Z ) :- rodic( X, Z ).      % (1)
```

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

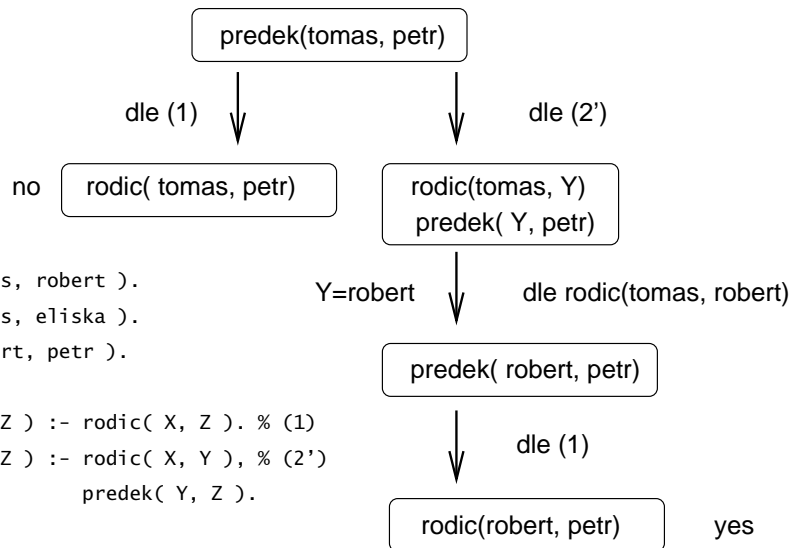
## Výpočet odpovědi na dotaz ?- predek(tomas,robert)

```
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)

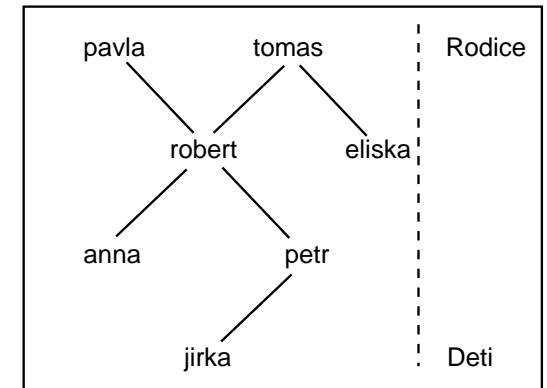


```
rodic( tomas, robert ).
rodic( tomas, eliska ).
rodic( robert, petr ).
```

```
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) --> ???

predek(robert, P) --> ???

Potomek=jirka

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,_).`

Hana Rudová, Logické programování I, 25. února 2013

14

Syntaxe a význam Prologovských programů

## 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) )`

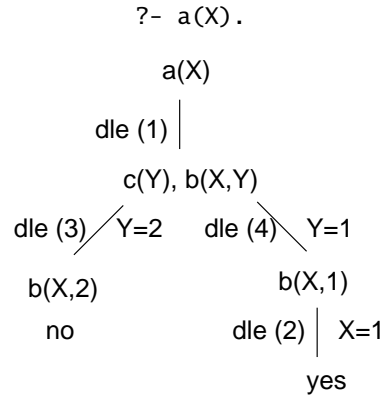
## Unifikace

- 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 instancie? `...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(...))))))))))`



## Pořadí klauzulí a cílů II.

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



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.

## Operátory, aritmetika

### Operátory

- Infixová notace:  $2*a + b*c$
- Prefixová notace:  $+( *(2,a), *(b,c) )$       priorita +: 500, priorita \*: 400
  - prefixovou notaci lze získat predikátem display/1
  - $:- display((a:-s(0),b,c)).$        $:- (a, ,(s(0), ,(b,c)))$
- **Priorita operátorů:** operátor s **nejvyšší** prioritou je hlavní funktor
- Uživatelsky definované operátory: zna      petr zna alese.      zna( petr, alese).
- Definice operátoru:  $:- op( 600, xfx, zna ).$       priorita: 1..1200
  - $:- op( 1100, xfy, ; ).$       nestrukturované objekty: 0
  - $:- op( 1000, xfy, , ).$
  - $p :- q,r; s,t.$        $p :- (q,r) ; (s,t).$       ; má vyšší prioritu než ,
  - $:- op( 1200, xfx, :- ).$       :- má nejvyšší prioritu
- Definice operátoru není spojena s datovými manipulacemi (kromě spec. případů)

# Aritmetika

## Typy operátorů

### Typy operátorů

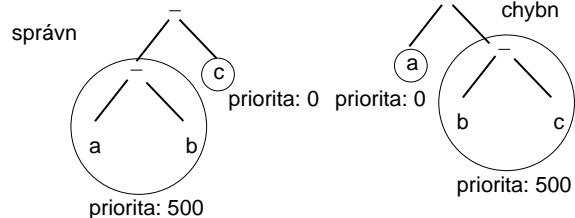
- infixové operátory:  $xfx$ ,  $xfy$ ,  $yfx$
- prefixové operátory:  $fx$ ,  $fy$
- postfixové operátory:  $xf$ ,  $yf$

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  $yfx$



### 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)
- výraz na pravé straně je nejdříve aritmeticky vyhodnocen a pak unifikován s levou stranou  
 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:  $?- A == B. \dots \text{no}$   $?- A=B, A==B. \dots B = A \text{ yes}$

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

porovnej:  $?- A \backslash == B. \dots \text{yes}$   $?- A=B, A \backslash == B. \dots A \text{ no}$

$X \text{ is } Y$  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 alfabetického n. aritmetického uspořádání

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

zleva podle argumentů

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