

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: 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,

X studuje, jestliže

X je student

pro každé X,

X je student, potom

X studuje

● `pracuje(X) :- clovek(X, _Vek, CoDe1a), prace(CoDe1a).`

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,

X studuje, jestliže

X je student

pro každé X,

X je student, potom

X studuje

● `pracuje(X) :- clovek(X, _Vek, CoDe1a), prace(CoDe1a).`

● **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 vracená hodnota)
- Psaní komentářů

```
clovek( novak, 18, student ).  
clovek( novotny, 30, ucitel ).
```

```
% komentář na konci řádku  
/* 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**

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**

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),
 rodic(Y, Z). % (2)

Rekurzivní pravidla

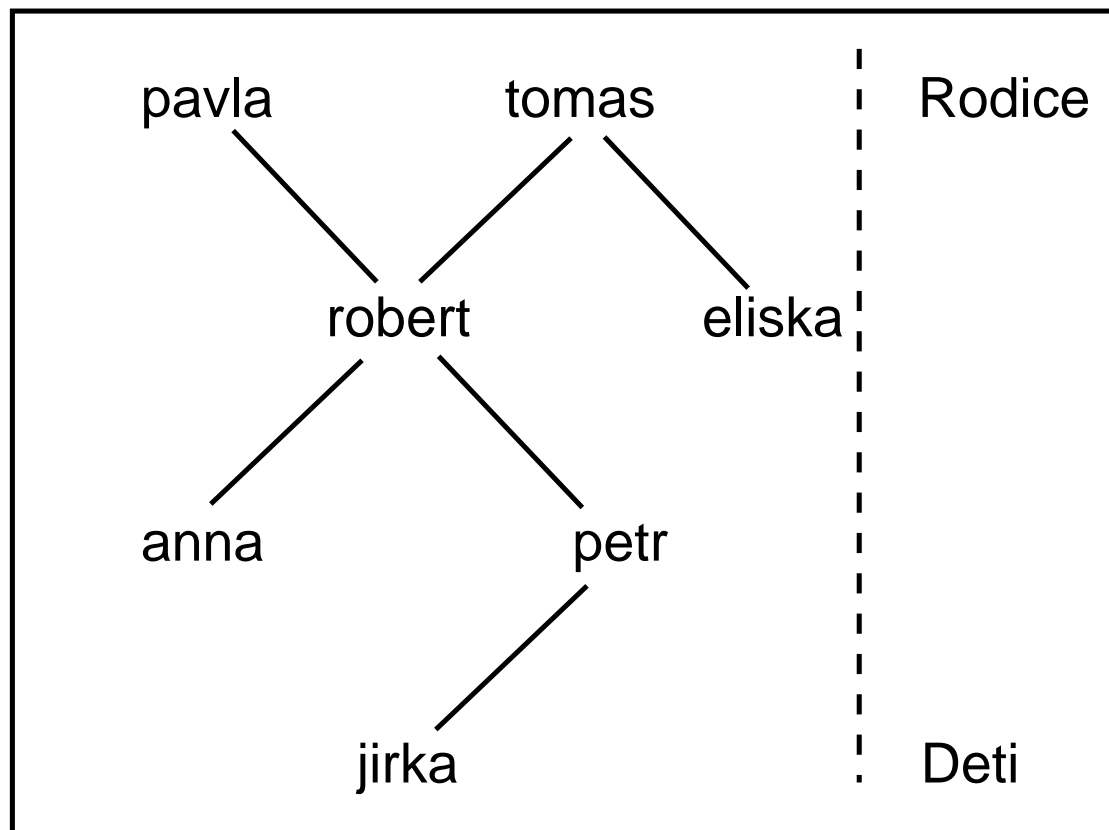
predek(X, Z) :- rodic(X, Z). % (1)

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

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

Příklad: rodokmen

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

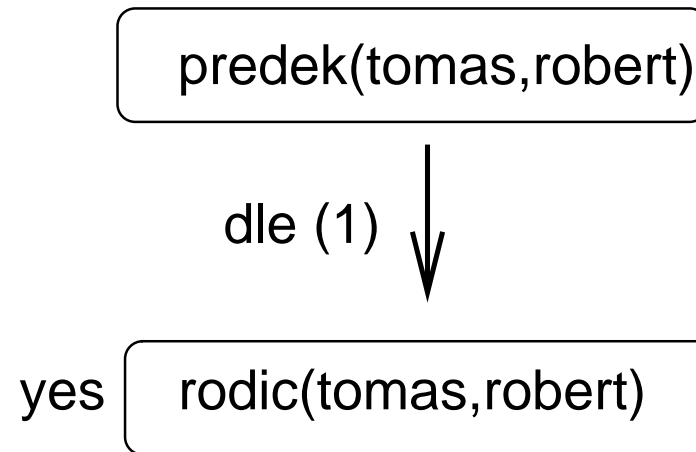


```
predek( X, Z ) :- rodic( X, Z ).           % (1)
```

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

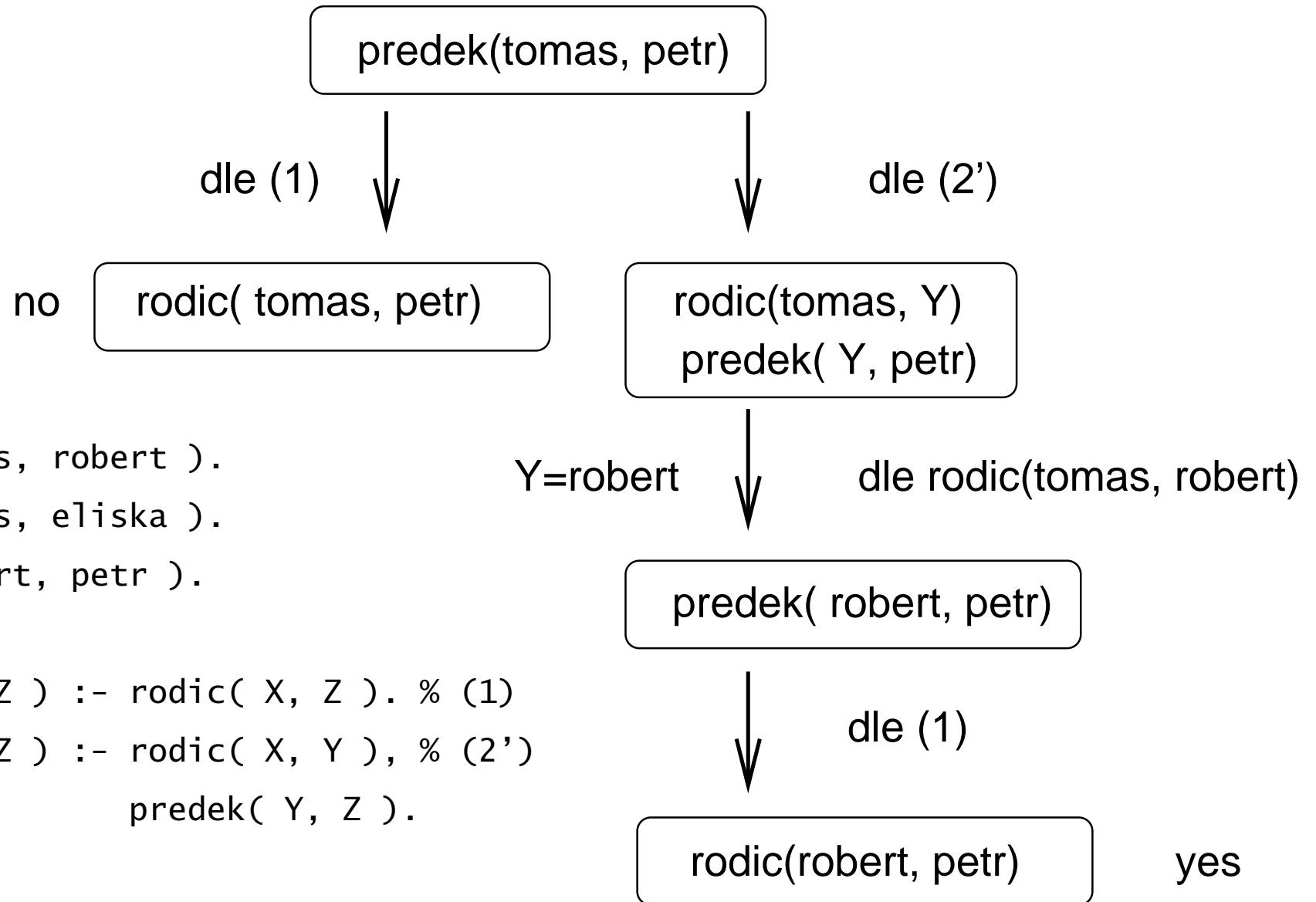
Výpočet odpovědi na dotaz ?- predek(tomas,robert)

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



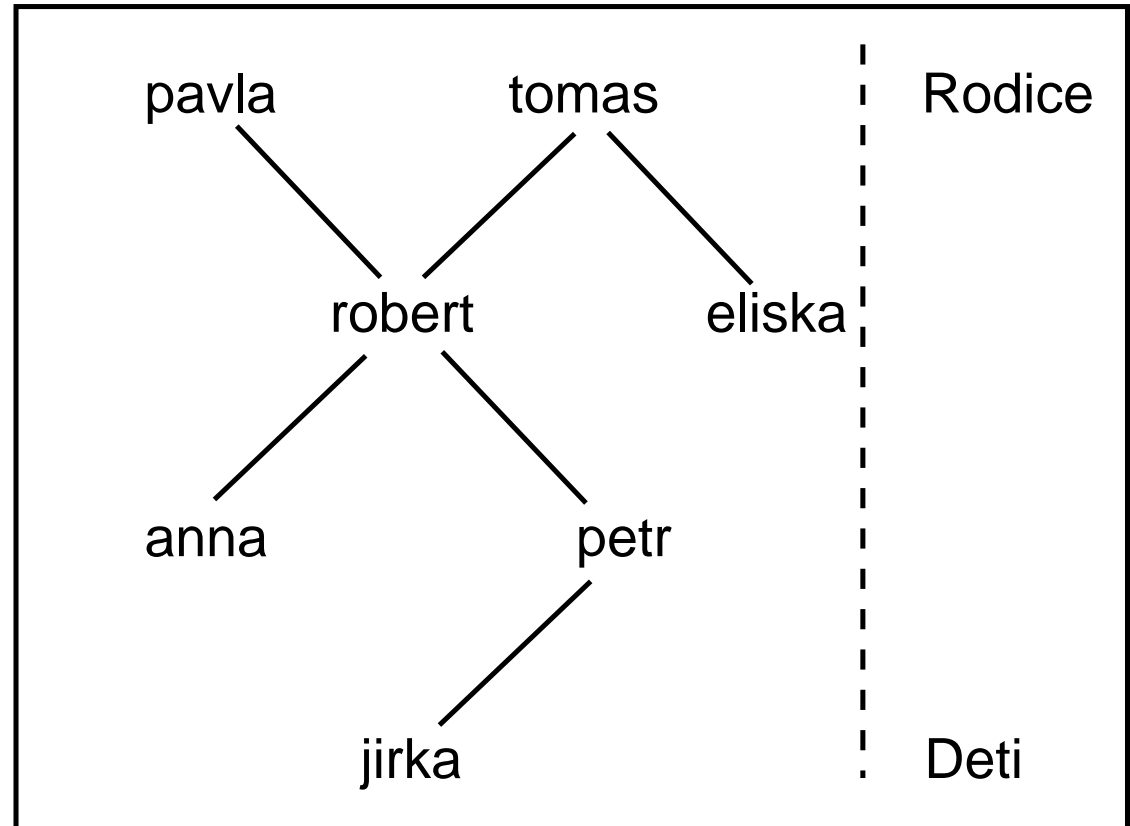
```
predek( X, Z ) :- rodic( X, Z ).           % (1)  
predek( X, Z ) :- rodic( X, Y ),          % (2')  
                  predek( Y, Z ).
```

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



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

```
% (1)  
% (2')
```

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

Syntaxe a význam Prologovských programů

Syntaxe Prologovských programů

● Typy objektů jsou rozpoznávány podle syntaxe

● Atom

- řetězce písmen, čísel, „_” začínající malým písmenem: `pavel`, `pavel_novak`, `x25`
- řetězce speciálních znaků: `<-->`, `====>`
- řetězce v apostrofech: `'Pavel'`, `'Pavel Novák'`

● Celá a reálná čísla: `0`, `-1056`, `0.35`

● Proměnná

- řetězce písmen, čísel, „_” začínající velkým písmenem nebo „_”
- **anonymní proměnná**: `ma_dite(X) :- rodic(X, _)`.
- hodnotu anonymní proměnné Prolog na dotaz nevrací: `?- rodic(X, _)`
- lexikální rozsah proměnné je pouze jedna klauzule:

`prvni(X,X,X) .`

`prvni(X,X,_) .`

Termy

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

Unifikace

- Termíny jsou **unifikovatelné**, jestliže
 - jsou identické nebo
 - proměnné v obou termínech mohou být instanciovány tak, že termíny jsou po substituci identické
 - $\text{datum}(D1, M1, 2003) = \text{datum}(1, M2, Y2)$ **operátor =**
 $D1 = 1, M1 = M2, Y2 = 2003$

Unifikace

- Termíny jsou **unifikovatelné**, jestliže
 - jsou identické nebo
 - proměnné v obou termínech mohou být instanciovány tak, že termíny jsou po substituci identické
 - $\text{datum}(D1, M1, 2003) = \text{datum}(1, M2, Y2)$ **operátor =**
 $D1 = 1, M1 = M2, Y2 = 2003$
- Hledáme **nejobecnější unifikátor** (*most general unifier (MGU)*)
 - jiné instanciaci? ... $D1 = 1, M1 = 5, Y2 = 2003$... není MGU
 - ?- $\text{datum}(D1, M1, 2003) = \text{datum}(1, M2, Y2), D1 = M1.$

Unifikace

• Termy jsou **unifikovatelné**, jestliže

- jsou identické nebo

- proměnné v obou termech mohou být instanciovány tak, že termy jsou po substituci identické

- $\text{datum}(D1, M1, 2003) = \text{datum}(1, M2, Y2)$ **operátor =**
D1 = 1, M1 = M2, Y2 = 2003

• Hledáme **nejobecnější unifikátor** (*most general unifier (MGU)*)

- jiné instanciaci? ... D1 = 1, M1 = 5, Y2 = 2003 ... není MGU

- ?- $\text{datum}(D1, M1, 2003) = \text{datum}(1, M2, Y2), D1 = M1.$

• **Test výskytu** (*occurs check*)

?- $X=f(X).$

$X = f(f(f(f(f(f(f(f(f(\dots))))))))))$

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:

$k = k \dots$ yes, $k1 = k2 \dots$ no,

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:

$k = k \dots \text{yes}$, $k1 = k2 \dots \text{no}$, $A = k(2,3) \dots \text{yes}$, $k(s,a,l(1)) = A \dots \text{yes}$

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:

$k = k \dots \text{yes}$, $k1 = k2 \dots \text{no}$, $A = k(2,3) \dots \text{yes}$, $k(s,a,l(1)) = A \dots \text{yes}$
 $s(sss(2),B,ss(2)) = s(sss(2),4,ss(2),s(1)) \dots$

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:

$k = k \dots \text{yes}$, $k1 = k2 \dots \text{no}$, $A = k(2,3) \dots \text{yes}$, $k(s,a,l(1)) = A \dots \text{yes}$
 $s(sss(2),B,ss(2)) = s(sss(2),4,ss(2),s(1)) \dots \text{no}$

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:

$k = k \dots \text{yes}$, $k1 = k2 \dots \text{no}$, $A = k(2,3) \dots \text{yes}$, $k(s,a,l(1)) = A \dots \text{yes}$

$s(sss(2),B,ss(2)) = s(sss(2),4,ss(2),s(1)) \dots \text{no}$

$s(sss(A),4,ss(3)) = s(sss(2),4,ss(A)) \dots$

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:

$k = k \dots \text{yes}$, $k1 = k2 \dots \text{no}$, $A = k(2,3) \dots \text{yes}$, $k(s,a,l(1)) = A \dots \text{yes}$

$s(sss(2),B,ss(2)) = s(sss(2),4,ss(2),s(1)) \dots \text{no}$

$s(sss(A),4,ss(3)) = s(sss(2),4,ss(A)) \dots \text{no}$

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:

$k = k \dots \text{yes}$, $k1 = k2 \dots \text{no}$, $A = k(2,3) \dots \text{yes}$, $k(s,a,l(1)) = A \dots \text{yes}$

$s(sss(2),B,ss(2)) = s(sss(2),4,ss(2),s(1)) \dots \text{no}$

$s(sss(A),4,ss(3)) = s(sss(2),4,ss(A)) \dots \text{no}$

$s(sss(A),4,ss(C)) = s(sss(t(B)),4,ss(A)) \dots$

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:

$k = k \dots \text{yes}$, $k1 = k2 \dots \text{no}$, $A = k(2,3) \dots \text{yes}$, $k(s,a,l(1)) = A \dots \text{yes}$

$s(sss(2),B,ss(2)) = s(sss(2),4,ss(2),s(1)) \dots \text{no}$

$s(sss(A),4,ss(3)) = s(sss(2),4,ss(A)) \dots \text{no}$

$s(sss(A),4,ss(C)) = s(sss(t(B)),4,ss(A)) \dots A=t(B),C=t(B) \dots \text{yes}$

Deklarativní a procedurální význam programů

- $p \text{ :- } q, r.$
 - Deklarativní: **Co** je výstupem programu?
 - p je pravdivé, jestliže q a r jsou pravdivé
 - Z q a r plyne p
- ⇒ význam mají logické relace

Deklarativní a procedurální význam programů

- $p :- q, r.$

- Deklarativní: **Co** je výstupem programu?

- p je pravdivé, jestliže q a r jsou pravdivé

- Z q a r plyne p

- ⇒ význam mají logické relace

- Procedurální: **Jak** vypočítáme výstup programu?

- p vyřešíme tak, že **nejprve** vyřešíme q a **pak** r

- ⇒ kromě logických relací je významné i pořadí cílů

- výstup

- indikátor yes/no určující, zda byly cíle splněny

- instanciací 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
```

Konjunce ",", vs. disjunkce ";" cílů

● **Konjunce** = nutné splnění **všech cílů**

● $p :- q, r.$

● **Disjunkce** = stačí splnění **libovolného cíle**

● $p :- q; r.$ $p :- q.$

$p :- r.$

● priorita středníku je vyšší:

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

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

$p :- q, r.$

$p :- s, t, u.$

Pořadí klauzulí a cílů

(a) `a(1).`

`?- a(1).`

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

`b(1,1).`

Pořadí klauzulí a cílů

(a) a(1).

?- a(1).

a(X) :- b(X,Y), a(Y).

b(1,1).

(b) a(X) :- b(X,Y), a(Y).

% změněné pořadí klauzulí v programu vzhledem k (a)

a(1).

b(1,1).

Pořadí klauzulí a cílů

(a) `a(1).` `?- a(1).`

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

`b(1,1).`

(b) `a(X) :- b(X,Y), a(Y).` `% změněné pořadí klauzulí v programu vzhledem k (a)`

`a(1).`

`b(1,1).`

`% nenalezení odpovědi: nekonečný cyklus`

Pořadí klauzulí a cílů

(a) `a(1).` `?- a(1).`

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

`b(1,1).`

(b) `a(X) :- b(X,Y), a(Y).` `% změněné pořadí klauzulí v programu vzhledem k (a)`

`a(1).`

`b(1,1).`

`% nenalezení odpovědi: nekonečný cyklus`

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

`b(1,1).`

`c(2).`

`c(1).`

Pořadí klauzulí a cílů

(a) `a(1).` `?- a(1).`

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

`b(1,1).`

(b) `a(X) :- b(X,Y), a(Y).` `% změněné pořadí klauzulí v programu vzhledem k (a)`

`a(1).`

`b(1,1).`

`% nenalezení odpovědi: nekonečný cyklus`

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

`b(1,1).`

`c(2).`

`c(1).`

(d) `a(X) :- c(Y), b(X,Y).` `% změněné pořadí cílů v těle klauzule vzhledem k (c)`

`b(1,1).`

`c(2).`

`c(1).`

Pořadí klauzulí a cílů

(a) $a(1).$ $?- a(1).$

$a(X) :- b(X,Y), a(Y).$

$b(1,1).$

(b) $a(X) :- b(X,Y), a(Y).$ % změněné pořadí klauzulí v programu vzhledem k (a)

$a(1).$

$b(1,1).$

% nenalezení odpovědi: nekonečný cyklus

(c) $a(X) :- b(X,Y), c(Y).$ $?- a(X).$

$b(1,1).$

$c(2).$

$c(1).$

(d) $a(X) :- c(Y), b(X,Y).$ % změněné pořadí cílů v těle klauzule vzhledem k (c)

$b(1,1).$

$c(2).$

$c(1).$

% náročnější nalezení první odpovědi než u (c)

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

Pořadí klauzulí a cílů II.

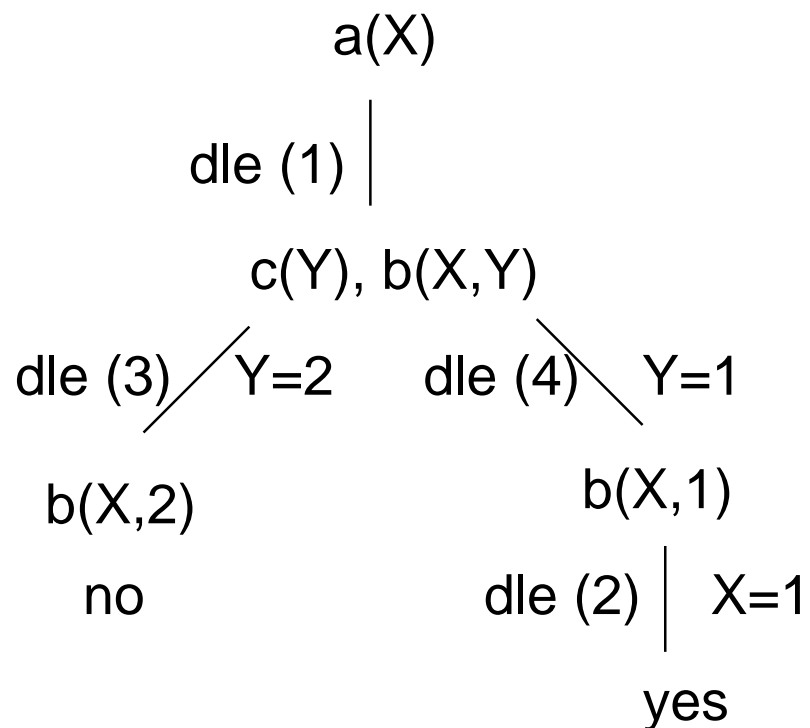
(1) $a(X) :- c(Y), b(X, Y).$

(2) $b(1, 1).$

(3) $c(2).$

(4) $c(1).$

?- $a(X).$



Pořadí klauzulí a cílů II.

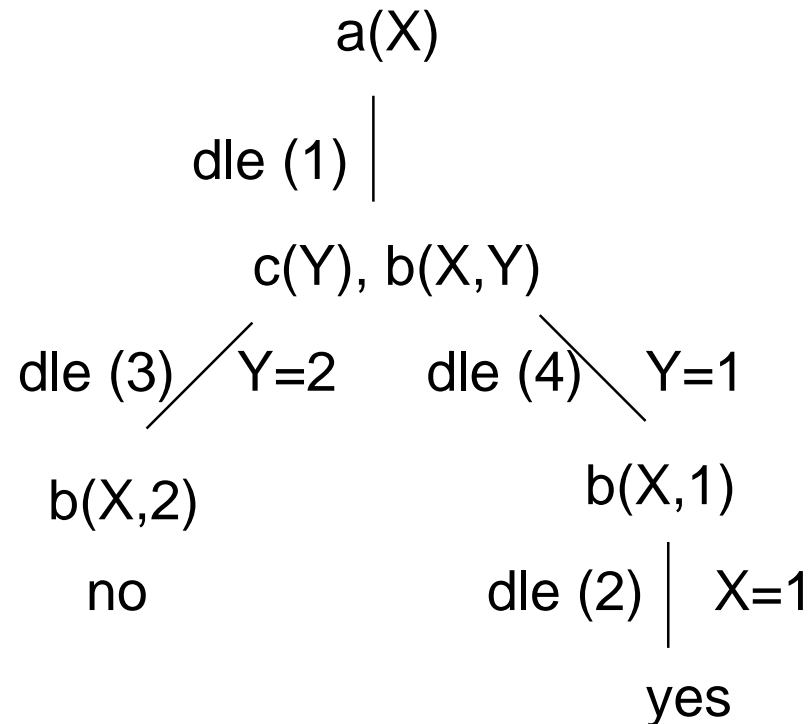
(1) $a(X) :- c(Y), b(X,Y).$

(2) $b(1,1).$

(3) $c(2).$

(4) $c(1).$

?- $a(X).$



Vyzkoušejte si:

(1) $a(X) :- b(X,X), c(X).$

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

(4) $b(2,2).$

(5) $b(2,1).$

(6) $c(1).$

Operátory, aritmetika

Operátory

- Infixová notace: $2 * a + b * c$
- Prefixová notace: $+(*(2, a), *(b, c))$ priorita +: 500, priorita *: 400
- **Priorita operátorů**: operátor s **nejvyšší** prioritou je hlavní funktor

Operátory

- Infixová notace: $2 * a + b * c$
- Prefixová notace: $+(*(2, a), *(b, c))$ priorita +: 500, priorita *: 400
- **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

Operátory

- Infixová notace: $2 * a + b * c$
- Prefixová notace: $+(*(2, a), *(b, c))$ priorita +: 500, priorita *: 400
- **Priorita operátorů**: operátor s **nejvyšší** prioritou je hlavní funktor
- Uživatelsky definované operátory: zna
petr zna a lese. zna(petr, a lese).
- Definice operátoru: $:- \text{op}(600, \text{xfx}, \text{zna})$. priorita: 1..1200
 - $:- \text{op}(1100, \text{xfy}, ;)$. nestrukturované objekty: 0
 - $:- \text{op}(1000, \text{xfy}, ,)$.
 - $p :- q, r; s, t.$ $p :- (q, r) ; (s, t).$; má vyšší prioritu než ,
 - $:- \text{op}(1200, \text{xfx}, :-)$. :- má nejvyšší prioritu

Operátory

- Infixová notace: $2 * a + b * c$
- Prefixová notace: $+(*(2, a), *(b, c))$ priorita +: 500, priorita *: 400
- **Priorita operátorů**: operátor s **nejvyšší** prioritou je hlavní funktor
- Uživatelsky definované operátory: zna
petr zna a lese. zna(petr, a lese).
- 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 speciálních případů)

Typy operátorů

Typy operátorů

infixové operátory: xfx , xfy , yfx

př. $xfx = yfx -$

prefixové operátory: fx , fy

př. $fx ?- fy -$

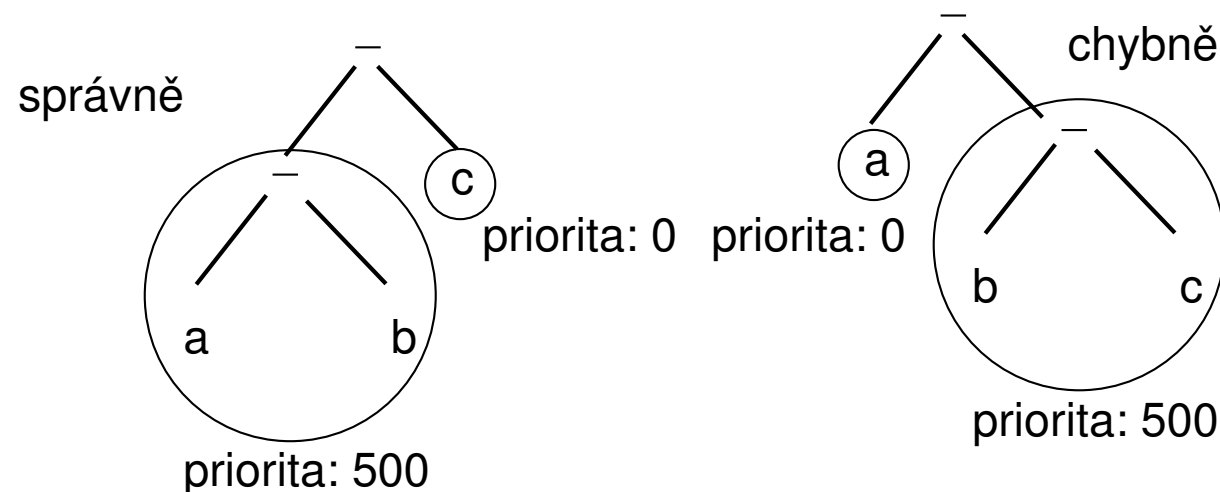
postfixové operátory: xf , yf

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

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

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

● pravá strana musí být vyhodnotitelný výraz (bez proměnné)

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

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

● 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: $1+2 := 2+1$ $1+2 = 2+1$

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

● 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

$X \neq Y$ X a Y nejsou unifikovatelné, (také $\neg X = Y$)

Různé typy rovností a porovnání

- $X = Y$ X a Y jsou unifikovatelné
 - $X \neq Y$ X a Y nejsou unifikovatelné, (také $\neq X = Y$)
 - $X == Y$ X a Y jsou identické
- porovnej: $?- A == B. \dots$ no $?- A=B, A==B.$

Různé typy rovností a porovnání

$X = Y$	X a Y jsou unifikovatelné
$X \neq Y$	X a Y nejsou unifikovatelné, (také $\neg X = Y$)
$X == Y$	X a Y jsou identické porovnej: ?- A == B. ... no ?- A=B, A==B. ... B = A yes
$X \neq Y$	X a Y nejsou identické porovnej: ?- A \neq B. ... yes ?- A=B, A \neq B. ... A no

Různé typy rovností a porovnání

- $X = Y$ X a Y jsou unifikovatelné
- $X \neq Y$ X a Y nejsou unifikovatelné, (také $\neq X = Y$)
- $X == Y$ X a Y jsou identické
porovnej: ?- A == B. ... no ?- A=B, A==B. ... B = A yes
- $X \neq Y$ X a Y nejsou identické
porovnej: ?- A \neq B. ... yes ?- A=B, A \neq B. ... A no
- $X is Y$ Y je aritmeticky vyhodnoceno a výsledek je přiřazen X
- $X ::= Y$ X a Y jsou si aritmeticky rovny
- $X \neq Y$ X a Y si aritmeticky nejsou rovny
- $X < Y$ aritmetická hodnota X je menší než Y ($=<$, $>$, $>=$)

Různé typy rovností a porovnání

- $X = Y$ X a Y jsou unifikovatelné
- $X \neq Y$ X a Y nejsou unifikovatelné, (také $\neq X = Y$)
- $X == Y$ X a Y jsou identické
porovnej: ?- A == B. ... no ?- A=B, A==B. ... B = A yes
- $X \neq Y$ X a Y nejsou identické
porovnej: ?- A \neq B. ... yes ?- A=B, A \neq B. ... A no
- $X is Y$ Y je aritmeticky vyhodnoceno a výsledek je přiřazen X
- $X ::= Y$ X a Y jsou si aritmeticky rovny
- $X \neq Y$ 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

Různé typy rovností a porovnání

- $X = Y$ X a Y jsou unifikovatelné
- $X \neq Y$ X a Y nejsou unifikovatelné, (také $\neq X = Y$)
- $X == Y$ X a Y jsou identické
porovnej: ?- A == B. ... no ?- A=B, A==B. ... B = A yes
- $X \neq Y$ X a Y nejsou identické
porovnej: ?- A \neq B. ... yes ?- A=B, A \neq B. ... A no
- $X is Y$ Y je aritmeticky vyhodnoceno a výsledek je přiřazen X
- $X ::= Y$ X a Y jsou si aritmeticky rovny
- $X \neq Y$ 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(pavel, g(b)) @< f(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.

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)).
```


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( Vrcho1, Vyska ),  
                                   kostka( Kostka ),  
                                   vetsi( Vrcho1, Kostka ),  
                                   NovaVyska is Vyska + 1,  
                                   Pridani = vez( Kostka, NovaVyska ).
```