Aritmetika, seznamy, řez

Aritmetika

Důležitý rozdíl ve vestavěných predikátech `is/2` vs. `=/2` vs. `=:=/2`

- `is/2`

  < konstanta nebo proměnná > is < aritmetický výraz >

  výraz na pravé straně je nejdříve aritmeticky vyhodnocen

  a pak unifkován s levou stranou

- `=/2`

  < libovolný term > = < libovolný term >

  levá a pravá strana jsou unifikovány

- `"=:="/2` `"=\="/2` `">="/2` `"=<"/2`

  < aritmetický výraz > =:= < aritmetický výraz >

  < aritmetický výraz > =\= < aritmetický výraz >

  < aritmetický výraz > =< < aritmetický výraz >

  < aritmetický výraz > >= < aritmetický výraz >

  levá i pravá strana jsou nejdříve aritmeticky vyhodnoceny a pak porovnány

Aritmetika: příklady

Jak se liší následující dotazy (na co se kdy ptáme)? Které uspějí (kladná odpověď), které neuspějí (záporná odpověď), a které jsou špatně (dojde k chybě)? Za jakých předpokladů by ty neúspěšné případně špatné uspěly?

- X = Y + 1
- X is Y + 1
- X = Y
- X == Y
- 1 + 1 = 2
- 2 = 1 + 1
- 1 + 1 = 1 + 1
- 1 + 1 is 1 + 1
- 1 + 2 =:= 2 + 1
- X \== Y
- X =\= Y
- 1 + 2 =\= 1 - 2
- 1 <= 2
- 1 =< 2
- sin(X) is sin(2)
- sin(X) = sin(2+Y)
- sin(X) =:= sin(2+Y)

Seznamy a append

```
append( [], S, S ).
append( [X|S1], S2, [X|S3] ) :- append( S1, S2, S3).
```

Napište následující predikáty pomocí append/3:

- ```
  last( X, S ) :-
  append([3,2], [6], [3,2,6]).        X=6, S=[3,2,6]
  ```
  ```

Seznamy a append

```
append( [], S, S ).
append( [X|S1], S2, [X|S3] ) :- append( S1, S2, S3).
```

Napište následující predikáty pomocí append/3:

- ```
  last( X, S ) :- append( _S1, [X], S).
  append([3,2], [6], [3,2,6]).        X=6, S=[3,2,6]
  ```
  ```

Seznamy a append

```
append( [], S, S ).
append( [X|S1], S2, [X|S3] ) :- append( S1, S2, S3).
```

Napište následující predikáty pomocí append/3:

- ```
  last( X, S ) :- append( _S1, [X], S).
  append([3,2], [6], [3,2,6]).        X=6, S=[3,2,6]
  ```
- ```
  prefix( S1, S2 ) :-
  ```

Seznamy a append

```
append( [], S, S ).
append( [X|S1], S2, [X|S3] ) :- append( S1, S2, S3).
```

Napište následující predikáty pomocí append/3:

- ```
  last( X, S ) :- append( _S1, [X], S).
  append([3,2], [6], [3,2,6]).        X=6, S=[3,2,6]
  ```
- ```
  prefix( S1, S2 ) :- append( S1, _S3, S2).
  ```

Seznamy a append

```
append( [], S, S ).
append( [X|S1], S2, [X|S3] ) :- append( S1, S2, S3).
```

Napište následující predikáty pomocí append/3:

- ```
  last( X, S ) :- append( _S1, [X], S).
  append([3,2], [6], [3,2,6]).        X=6, S=[3,2,6]
  ```
- ```
  prefix( S1, S2 ) :- append( S1, _S3, S2).
  ```
  DÚ: suffix(S1,S2)

Seznamy a append

```
append( [], S, S ).
append( [X|S1], S2, [X|S3] ) :- append( S1, S2, S3).
```

Napište následující predikáty pomocí append/3:

- ```
  last( X, S ) :- append( _S1, [X], S).
  append([3,2], [6], [3,2,6]).        X=6, S=[3,2,6]
  ```
- ```
  prefix( S1, S2 ) :- append( S1, _S3, S2).
  ```
  DÚ: suffix(S1,S2)
- ```
  member( X, S ) :-
  append([3,4,1], [2,6], [3,4,1,2,6]).    X=2, S=[3,4,1,2,6]
  ```
  DÚ: adjacent(X,Y,S)
  ```

Seznamy a append

```
append( [], S, S ).
append( [X|S1], S2, [X|S3] ) :- append( S1, S2, S3).
```

Napište následující predikáty pomocí append/3:

- ```
  last( X, S ) :- append( _S1, [X], S).
  append([3,2], [6], [3,2,6]).        X=6, S=[3,2,6]
  ```
- ```
  prefix( S1, S2 ) :- append( S1, _S3, S2).
  ```
  DÚ: suffix(S1,S2)
- ```
  member( X, S ) :- append( S1, [X|S2], S ).
  append([3,4,1], [2,6], [3,4,1,2,6]).    X=2, S=[3,4,1,2,6]
  ```
  DÚ: adjacent(X,Y,S)
  ```

Seznamy a append

```
append( [], S, S ).
append( [X|S1], S2, [X|S3] ) :- append( S1, S2, S3).
```

Napište následující predikáty pomocí append/3:

- ```
  last( X, S ) :- append( _S1, [X], S).
  append([3,2], [6], [3,2,6]).        X=6, S=[3,2,6]
  ```
- ```
  prefix( S1, S2 ) :- append( S1, _S3, S2).
  DÚ: suffix(S1,S2)
  ```
- ```
  member( X, S ) :- append( S1, [X|S2], S ).
  append([3,4,1], [2,6], [3,4,1,2,6]).    X=2, S=[3,4,1,2,6]
  DÚ: adjacent(X,Y,S)
  ```
- ```
  % sublist(+S,+ASB)
  sublist(S,ASB) :-
  ```

Seznamy a append

```
append( [], S, S ).
append( [X|S1], S2, [X|S3] ) :- append( S1, S2, S3).
```

Napište následující predikáty pomocí append/3:

- ```
  last( X, S ) :- append( _S1, [X], S).
  append([3,2], [6], [3,2,6]).        X=6, S=[3,2,6]
  ```
- ```
  prefix( S1, S2 ) :- append( S1, _S3, S2).
  DÚ: suffix(S1,S2)
  ```
- ```
  member( X, S ) :- append( S1, [X|S2], S ).
  append([3,4,1], [2,6], [3,4,1,2,6]).    X=2, S=[3,4,1,2,6]
  DÚ: adjacent(X,Y,S)
  ```
- ```
  % sublist(+S,+ASB)
  sublist(S,ASB) :- append( AS, B, ASB ),
                    append( A, S, AS ).
  ```

  POZOR na efektivitu, bez append lze často napsat efektivněji

Akumulátor a `sum_list(S,Sum)`

```
?- sum_list( [2,3,4], Sum ).
```

bez akumulátoru:

Akumulátor a sum_list(S,Sum)

```
?- sum_list( [2,3,4], Sum ).
```

bez akumulátoru:

```
sum_list( [], 0 ).
sum_list( [H|T], Sum ) :- sum_list( T, SumT ),
                          Sum is H + SumT.
```

s akumulátorem:

```
sum_list( S, Sum ) :-  sum_list( S, 0, Sum ).

sum_list( [], Sum, Sum ).
sum_list( [H|T], A, Sum ) :- A1 is A + H,
                             sum_list( T, A1, Sum).
```

Výpočet faktoriálu `fact(N,F)`

s akumulátorem:

Výpočet faktoriálu `fact(N,F)`

s akumulátorem:

```
fact( N, F ) :- fact ( N, 1, F ).

fact( 1, F, F ) :- !.
fact( N, A, F ) :-  N > 1,
                    A1 is N * A,
                    N1 is N - 1,
                    fact( N1, A1, F ).
```

```
r(X):-write(r1).
r(X):-p(X),write(r2).
r(X):-write(r3).

p(X):-write(p1).
p(X):-a(X),b(X),!,
      c(X),d(X),write(p2).
p(X):-write(p3).

a(X):-write(a1).
a(X):-write(a2).

b(X):- X > 0, write(b1).
b(X):- X < 0, write(b2).

c(X):- X mod 2 =:= 0, write(c1).
c(X):- X mod 3 =:= 0, write(c2).

d(X):- abs(X) < 10, write(d1).
d(X):- write(d2).
```

```
r(X):-write(r1).
r(X):-p(X),write(r2).
r(X):-write(r3).

p(X):-write(p1).
p(X):-a(X),b(X),!,
      c(X),d(X),write(p2).
p(X):-write(p3).

a(X):-write(a1).
a(X):-write(a2).

b(X):- X > 0, write(b1).
b(X):- X < 0, write(b2).

c(X):- X mod 2 =:= 0, write(c1).
c(X):- X mod 3 =:= 0, write(c2).

d(X):- abs(X) < 10, write(d1).
d(X):- write(d2).
```

```
| ?- X=1,r(X).
r1
X = 1 ? ;
p1r2
X = 1 ? ;
a1b1r3
X = 1 ? ;
no
```

```
r(X):-write(r1).
r(X):-p(X),write(r2).
r(X):-write(r3).

p(X):-write(p1).
p(X):-a(X),b(X),!,
      c(X),d(X),write(p2).
p(X):-write(p3).

a(X):-write(a1).
a(X):-write(a2).

b(X):- X > 0, write(b1).
b(X):- X < 0, write(b2).

c(X):- X mod 2 =:= 0, write(c1).
c(X):- X mod 3 =:= 0, write(c2).

d(X):- abs(X) < 10, write(d1).
d(X):- write(d2).
```

```
| ?- X=1,r(X).
r1
X = 1 ? ;
p1r2
X = 1 ? ;
a1b1r3
X = 1 ? ;
no

| ?- X=0,r(X).
r1
X = 0 ? ;
p1r2
X = 0 ? ;
a1a2p3r2
X = 0 ? ;
r3
X = 0 ? ;
no
```

```
r(X):-write(r1).
r(X):-p(X),write(r2).
r(X):-write(r3).

p(X):-write(p1).
p(X):-a(X),b(X),!,
      c(X),d(X),write(p2).
p(X):-write(p3).

a(X):-write(a1).
a(X):-write(a2).

b(X):- X > 0, write(b1).
b(X):- X < 0, write(b2).

c(X):- X mod 2 =:= 0, write(c1).
c(X):- X mod 3 =:= 0, write(c2).

d(X):- abs(X) < 10, write(d1).
d(X):- write(d2).
```

```
| ?- X=1,r(X).
r1
X = 1 ? ;
p1r2
X = 1 ? ;
a1b1r3
X = 1 ? ;
no

| ?- X=0,r(X).
r1
X = 0 ? ;
p1r2
X = 0 ? ;
a1a2p3r2
X = 0 ? ;
r3
X = 0 ? ;
no
```

```
| ?- X=3,r(X).
r1
X = 3 ? ;
p1r2
X = 3 ? ;
a1b1c2d1p2r2
X = 3 ? ;
d2p2r2
X = 3 ? ;
r3
X = 3 ? ;
no
```

```
r(X):-write(r1).
r(X):-p(X),write(r2).
r(X):-write(r3).

p(X):-write(p1).
p(X):-a(X),b(X),!,
      c(X),d(X),write(p2).
p(X):-write(p3).

a(X):-write(a1).
a(X):-write(a2).

b(X):- X > 0, write(b1).
b(X):- X < 0, write(b2).

c(X):- X mod 2 =:= 0, write(c1).
c(X):- X mod 3 =:= 0, write(c2).

d(X):- abs(X) < 10, write(d1).
d(X):- write(d2).
```

```
| ?- X=1,r(X).
r1
X = 1 ? ;
p1r2
X = 1 ? ;
a1b1r3
X = 1 ? ;
no

| ?- X=0,r(X).
r1
X = 0 ? ;
p1r2
X = 0 ? ;
a1a2p3r2
X = 0 ? ;
r3
X = 0 ? ;
no
```

```
| ?- X=3,r(X).
r1
X = 3 ? ;
p1r2
X = 3 ? ;
a1b1c2d1p2r2
X = 3 ? ;
d2p2r2
X = 3 ? ;
r3
X = 3 ? ;
no
```

```
| ?- X= -6, r(X).
r1
X = -6 ? ;
p1r2
X = -6 ? ;
a1b2c1d1p2r2
X = -6 ? ;
d2p2r2
X = -6 ? ;
c2d1p2r2
X = -6 ? ;
d2p2r2
X = -6 ? ;
r3
X = -6 ? ;
no
```

```
r(X):-write(r1).
r(X):-p(X),write(r2).
r(X):-write(r3).

p(X):-write(p1).
p(X):-a(X),b(X),!,
      c(X),d(X),write(p2).
p(X):-write(p3).

a(X):-write(a1).
a(X):-write(a2).

b(X):- X > 0, write(b1).
b(X):- X < 0, write(b2).

c(X):- X mod 2 =:= 0, write(c1).
c(X):- X mod 3 =:= 0, write(c2).

d(X):- abs(X) < 10, write(d1).
d(X):- write(d2).
```

Prozkoumejte trasy výpočtu a navracení např. pomocí následujících dotazů (vždy si středníkem vyžádejte navracení):

```
(1) X=1,r(X).      (2) X=3,r(X).
(3) X=0,r(X).      (4) X= -6,r(X).
```

```
r(X):-write(r1).
r(X):-p(X),write(r2).
r(X):-write(r3).

p(X):-write(p1).
p(X):-a(X),b(X),!,
      c(X),d(X),write(p2).
p(X):-write(p3).

a(X):-write(a1).
a(X):-write(a2).

b(X):- X > 0, write(b1).
b(X):- X < 0, write(b2).

c(X):- X mod 2 =:= 0, write(c1).
c(X):- X mod 3 =:= 0, write(c2).

d(X):- abs(X) < 10, write(d1).
d(X):- write(d2).
```

Prozkoumejte trasy výpočtu a navracení např. pomocí následujících dotazů (vždy si středníkem vyžádejte navracení):

```
(1) X=1,r(X).      (2) X=3,r(X).
(3) X=0,r(X).      (4) X= -6,r(X).
```

- řez v predikátu p/1 neovlivní alternativy predikátu r/1
- dokud nebyl proveden řez, alternativy predikátu a/1 se uplatňují, př. neúspěch b/1 v dotazu (3)
- při neúspěchu cíle za řezem se výpočet navrací až k volající proceduře r/1, viz (1)
- alternativy vzniklé po provedení řezu se zachovávají - další možnosti predikátu c/1 viz (2) a (4)

Řez: maximum

Je tato definice predikátu max/3 korektní?

```
max(X,Y,X):-X>=Y,!.
max(X,Y,Y).
```

Řez: maximum

Je tato definice predikátu max/3 korektní?

```
max(X,Y,X):-X>=Y,!.
max(X,Y,Y).
```

Není, následující dotaz uspěje: `?- max(2,1,1).`

Uveďte dvě možnosti opravy, se zachováním použití řezu a bez.

Řez: maximum

Je tato definice predikátu max/3 korektní?

```
max(X,Y,X):-X>=Y,!.
max(X,Y,Y).
```

Není, následující dotaz uspěje: `?- max(2,1,1).`

Uveďte dvě možnosti opravy, se zachováním použití řezu a bez.

```
max(X,Y,X):-X>=Y.
max(X,Y,Y):-Y>X.
```

```
max(X,Y,Z):-X>=Y,!,Z=X.
max(X,Y,Y).
```

Problém byl v definici, v první verzi se tvrdilo: X=Z $\wedge$ X>=Y => Z=X

správná definice je: X>=Y => Z=X

Při použití řezu je třeba striktně oddělit vstupní podmínky

od výstupních unifikací a výpočtu.

Řez: member

Jaký je rozdíl mezi následujícími definicemi predikátů member/2. Ve kterých odpovědích se budou lišit? Vyzkoušejte např. pomocí member(X, [1,2,3]).

```
mem1(H,[H|_]).
mem1(H,[_|T]) :- mem1(H,T).
```

```
mem2(H,[H|_]) :- !.
mem2(H,[_|T]) :- mem2(H,T).
```

```
mem3(H,[K|_]) :- H==K.
mem3(H,[K|T]) :- H\==K, mem3(H,T).
```

Řez: member

Jaký je rozdíl mezi následujícími definicemi predikátů member/2. Ve kterých odpovědích se budou lišit? Vyzkoušejte např. pomocí member(X, [1,2,3]).

```
mem1(H,[H|_]).
mem1(H,[_|T]) :- mem1(H,T).
```

```
mem2(H,[H|_]) :- !.
mem2(H,[_|T]) :- mem2(H,T).
```

```
mem3(H,[K|_]) :- H==K.
mem3(H,[K|T]) :- H\==K, mem3(H,T).
```

- mem1/2 vyhledá všechny výskyty, při porovnávání hledaného prvku s prvky seznamu může dojít k vázání proměnných (může sloužit ke generování všech prvků seznamu)

Řez: member

Jaký je rozdíl mezi následujícími definicemi predikátů member/2. Ve kterých odpovědích se budou lišit? Vyzkoušejte např. pomocí member(X, [1,2,3]).

```
mem1(H,[H|_]).
mem1(H,[_|T]) :- mem1(H,T).
```

```
mem2(H,[H|_]) :- !.
mem2(H,[_|T]) :- mem2(H,T).
```

```
mem3(H,[K|_]) :- H==K.
mem3(H,[K|T]) :- H\==K, mem3(H,T).
```

- mem1/2 vyhledá všechny výskyty, při porovnávání hledaného prvku s prvky seznamu může dojít k vázání proměnných (může sloužit ke generování všech prvků seznamu)
- mem2/2 najde jenom první výskyt, taky váže proměnné

Řez: member

Jaký je rozdíl mezi následujícími definicemi predikátů member/2. Ve kterých odpovědích se budou lišit? Vyzkoušejte např. pomocí member(X, [1,2,3]).

```
mem1(H,[H|_]).
mem1(H,[_|T]) :- mem1(H,T).
```

```
mem2(H,[H|_]) :- !.
mem2(H,[_|T]) :- mem2(H,T).
```

```
mem3(H,[K|_]) :- H==K.
mem3(H,[K|T]) :- H\==K, mem3(H,T).
```

- mem1/2 vyhledá všechny výskyty, při porovnávání hledaného prvku s prvky seznamu může dojít k vázání proměnných (může sloužit ke generování všech prvků seznamu)
- mem2/2 najde jenom první výskyt, taky váže proměnné
- mem3/2 najde jenom první výskyt, proměnné neváže (hledá pouze identické prvky)

Dokážete napsat variantu, která hledá jenom identické prvky a přitom najde všechny výskyty?

Řez: member

Jaký je rozdíl mezi následujícími definicemi predikátů member/2. Ve kterých odpovědích se budou lišit? Vyzkoušejte např. pomocí member(X, [1,2,3]).

```
mem1(H,[H|_]).
mem1(H,[_|T]) :- mem1(H,T).
```

```
mem2(H,[H|_]) :- !.
mem2(H,[_|T]) :- mem2(H,T).
```

```
mem3(H,[K|_]) :- H==K.
mem3(H,[K|T]) :- H\==K, mem3(H,T).
```

- mem1/2 vyhledá všechny výskyty, při porovnávání hledaného prvku s prvky seznamu může dojít k vázání proměnných (může sloužit ke generování všech prvků seznamu)
- mem2/2 najde jenom první výskyt, taky váže proměnné
- mem3/2 najde jenom první výskyt, proměnné neváže (hledá pouze identické prvky)

Dokážete napsat variantu, která hledá jenom identické prvky a přitom najde všechny výskyty?

```
mem4(H,[K|_]) :- H==K.   mem4(H,[K|T]) :- mem4(H,T).
```

Seznamy: `intersection(A,B,C)`

DÚ: Napište predikát pro výpočet průniku dvou seznamů.

Nápověda: využijte predikát member/2

DÚ: Napište predikát pro výpočtu rozdílu dvou seznamů. Nápověda: využijte predikát member/2