Neprocedurální programování

Cvičení 27.2.2023

Cvičení

- Napište predikát `` `same_length(?L1, ?L2)` ``,

který uspěje ve chvíli, kdy jsou oba seznamy stejně dlouhé.

- Nepoužívejte aritmetiku.

Řešení

```
same_length([], []).
same_length([_ | Tail1], [_ | Tail2]) :-
same_length(Tail1, Tail2).
```

- Prázdný seznam je stejně dlouhý pouze jako prázdný seznam (je to jediný seznam délky 0).
- Seznamy jsou stejně dlouhé, pokud jsou jejich ocásky stejně dlouhé

Cvičení

- Napište predikát `middle(+L, ?M)`, který uspěje, pokud je `M` prvek uprostřed seznamu.

- Nepoužívejte aritmetiku!

Nápověda

- Vyrobte si `` `middle(+L1, +L2, ?M)` ``, kde v jednom seznamu skáčete po dvou prvcích a v druhém po jednom prvku.

- Poté volejte následovně:

  `` `middle(L, M) :- middle(L, L, M)` ``.

Řešení

```
middle(List, Mid) :- middle(List, List, Mid).
middle([Mid | _], [], Mid).
middle([Mid | _], [_], Mid).
middle([_ | Tail1], [_, _ | Tail2], Mid) :-
                                middle(Tail1, Tail2, Mid).
```

- Pokud v druhém seznamu nic nezbylo, pak začátek prvního seznamu je prostředek.
- Pokud v druhém seznamu zbyl jediný prvek, pak začátek prvního seznamu je prostředek.
- V prvním seznamu skáčeme po jednom prvku, v druhém po dvou prvcích a voláme rekurzivně zbytek :)

Cvičení

- Napíšeme si predikát `delka`, který vrátí délku seznamu v Peanově aritmetice (viz předchozí přednáška). Čísla v Peanove aritmetice jsme definovali následovně:

  num(X) :- X je 0 nebo X je následníkem jiného čísla.

```
num(0).
num(s(X)) :- num(X).

add(0, X, X) :- num(X).
add(s(X), Y, s(Z)) :- add(X, Y, Z).
```

Řešení

- `delka` vrací délku seznamu v Peanově aritmetice

```
delka(Xs, R) :- delka_(Xs, 0, R).
% delka_(+Xs, +Acc, -R)
delka_([], Acc, Acc).
% delka_([_|Xs], Acc, Result) :- delka_(Xs, Acc + 1,
Result).
delka_([_|Xs], Acc, Result) :- delka_(Xs, s(Acc),
Result).
```

Cvičení

- Napište predikát `soucet`, který sečte všechny členy seznamu (předpokládáme, že elementy seznamu jsou kódovány pomocí čísel z Peanovy aritmetiky).

Řešení

```
soucet_peano(L,N) :- soucet_peano(L,0,N).
soucet_peano([], A, A).
soucet_peano([Head|Tail], A, Result) :-
  add(Head, A, Result1),
     soucet_peano(Tail, Result1, Result).
```

Cvičení

- Napište predikát `smaz(?P, ?L, ?NL)`, který ze seznamu L, který obsahuje alespoň jeden prvek P, vytvoří seznam NL smazáním tohoto prvku.

Řešení

```
smaz(P, [P|L], L).
smaz(P, [X|L], [X|NL]) :- smaz(P, L, NL).
```

Cvičení

- Naprogramujte predikát `nahrad(?P, ?NP, ?L, ?NL)` splnitelný, jestliže L obsahuje P a seznam NL má stejné prvky jako L, až na jeden prvek NP nahrazující prvek P.
- Predikát tedy nahradí právě jeden výskyt prvku P za NP.

Řešení

```
nahrad(P, NP, [P|L], [NP|L]).
nahrad(P, NP, [X|L], [X|NL]) :-
     nahrad(P, NP, L, NL).
```

Cvičení

- Naprogramujte predikát `smazVice(?P, ?L, ?NL)` splnitelný, jestliže NL je seznam vzniklý z L vypuštěním libovolné podmnožiny výskytu prvků P.

Řešení

```
smazVice(_, [], []).
smazVice(P, [P|L], NL) :- smazVice(P, L, NL).
smazVice(P, [X|L], [X|NL]) :- smazVice(P, L, NL).
```

Cvičení

- Naprogramujte predikát `smazVsechny(?P, ?L, ?NL)` splnitelný, jestliže NL je seznam vzniklý z L vypuštěním všech výskytu prvků P.

Řešení

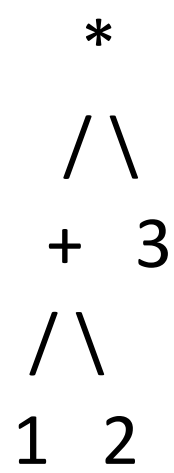
```
smazVsechny(_, [], []).
smazVsechny(P, [P|L], NL) :- smazVsechny(P, L, NL).
smazVsechny(P, [X|L], [X|NL]) :- P \= X,
     smazVsechny(P, L, NL).
```

Aritmetika

- Nejprve si rozmyslete, že syntakticky je následující výraz:

(1 + 2) * 3

v podstatě jen tenhle strom:



Aritmetika

- Když do swipl-u napíšete

  ?- X = 1 + 2

- tak vám odpoví

  X = 1 + 2

- Protože Prolog podporuje uživatelsky definované operátory, tak `1 + 2` vlastně není nic jiného než `+(1, 2)`.

  ?- +(1, 2) = 1 + 2

  true.

- Podobně:

  ?- X = (1 + 2) * 3

  X = (1 + 2) * 3

- To je proto, že Prologovské rovnítko `=` je unifikace.

Aritmetika

- My ale chceme operaci "vyhodnoť a ulož do".
- Na to se dá použít operátor `is`!

```
?- X is 1 + 2
X = 3

?- X is (1 + 2) * 3
X = 9
```

Aritmetika

- Co když tedy budeme chtít napsat součet s "opravdovou" aritmetikou?
- Nejprve zapomeňme na to, že existuje `is`.

```
soucetBezIs(Xs, Result) :- soucetBezIs_(Xs, 0, Result).
```

- zase používáme akumulátor

```
soucetBezIs_([], Acc, Acc).
soucetBezIs_([H|T], Acc, Result) :- NewAcc = H + Acc,
        soucetBezIs_(T, NewAcc, Result).
```

Aritmetika

- Nyní když zavoláme `soucetBezIs`, tak se `NewAcc` bude nastavovat jen syntakticky, bez vyhodnocení. Tedy:

```
?- soucetBezIs([1, 2, 3], X).
X = (3 + (2 + (1 + 0)))
```

- To ale nechceme :(

Aritmetika

- Zkusme tedy napsat verzi s `is`:

```
soucet(Xs, Result) :- soucet_(Xs, 0, Result).
soucet_([], Acc, Acc).
soucet_([H|T], Acc, Result) :- NewAcc is H + Acc,
        soucet_(T, NewAcc, Result).
```

```
:- soucet([1, 2, 3], X).
X = 6
```

- Mnohem lepší! :)

Cvičení

- Napište predikát, které vezme číslo v Peanově aritmetice a vrátí číslo v běžné aritmetice, třeba: s(s(s(0))) ~~~> 3

```
paeano_to_number(+P, -N)
```

Řešení

```
paeano_to_number(P, N) :- paeano_to_number_acc(P, 0, N).
paeano_to_number_acc(0, A, A).
paeano_to_number_acc(s(P),A,N) :- A1 is A + 1,
    paeano_to_number_acc(P,A1,N).
```

- pokud zkusime

  ```
  paeano_to_number(P, 3).
  ```

- P = s(s(s(0))) ;
- a cyklime ;)

Cvičení

- Napište opačný predikát.

```
number_to_peano(+N, -P)
```

Řešení

```
number_to_peano(N, P) :-
    number_to_peano_acc(N, 0, P).
number_to_peano_acc(0, A, A).
number_to_peano_acc(N, A, P) :- N > 0, N1 is N - 1,
    number_to_peano_acc(N1, s(A), P).
```

Cvičení

- Napište predikát `fakt(+N,-F)`, který je splněn, když F je faktoriál N

Řešení

```
fakt(X, Y) :- fakt_(X, 1, Y).
fakt_(1, Y, Y).
fakt_(X, Acc, Y) :- X > 1, NewAcc is Acc * X,
     NewX is X - 1,
     fakt_(NewX, NewAcc, Y).
```

- Defenzivně kontrolujeme, že nám někdo nedal číslo menší než 1 :)

Cvičení

- Napište predikát delka_seznamu(?L, ?N) :- Seznam L má N prvků.

Řešení

% delka_seznamu(?L, @A, ?N) :- L má délku N, zkrácením seznamu zvýší A o 1.

```
delka_seznamu(L, N) :- delka_seznamu(L, 0, N).
delka_seznamu([], N, N).
delka_seznamu([_|L], A, N) :- A1 is A + 1,
    delka_seznamu(L, A1, N).
```