

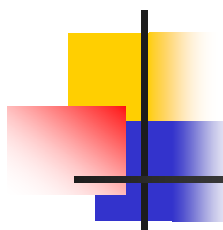


Logické programovanie 1

(pokus o úvod #1)

Peter Borovanský, KAI, I-18,
borovan(a)ii.fmph.uniba.sk

- je dielom sedemdesiatok, teda dekády '70...
- najprv vznikol programovací jazyk (Prolog), až potom teória LP ('80)
- **Fifth Generation** Computer Systems (FGCS) bol v 1982 projekt Japonského min. priemyslu zameraný na paralelné výpočty a logické programovanie
- Ehud Shapiro: Trip Report 1983, "Fifth Generation computers will be built around the concepts of concurrent logic programming.
- jeho cieľom bolo začiatkom '90 zvládnuť Umelú inteligenciu
- trochu s nadsádzkou sa hovorilo, že *programovanie je za 10 rokov mŕtva disciplína* (to sa už robiť nebude, všetko zvládne AI)
- vzniklo mnoho nesmrteľných konceptov, algoritmov, ktoré sa učíte do dnes



Logické programovanie 1

(pokus o úvod #2)

Peter Borovanský, KAI, I-18,
borovan(a)ii.fmph.uniba.sk

- logické programovanie je o rozmýšľaní v predikátovom počte
- neprogramujete metódy (čo niečo robia), ani funkcie (ktoré niečo počítajú)
- programujete relácie, vzťahy medzi objektami
- kým každá funkcia je relácia, naopak to neplatí - objavíte nedeterministické programovanie
- nedeterministické programovanie je simulované zabudovaným backtrackom
- Logické programovanie veľmi úzko súvisí s Constraint Logic Programming (LP s obmedzeniami)
- CLP(Z), CLP(Finite domain) je skoro to, čo poznáte ako SatSolver CLP(Bool)



Logické programovanie 1

(pokús o úvod #3)

programovanie v predikátovom počte

- Robinson, 1965

rezolučný princíp

$$(A \vee B) \wedge (\neg A \vee C) \Rightarrow (B \vee C)$$

- Robert Kowalski

Algorithms = Logic + Control

- Alain Colmerauer, 1972

jazyk Prolog

Klasická (až historická) literatúra, ktorú asi nebudete čítať:

Lloyd, J.W.: *Foundations of Logic Programming*, 1987

Apt, K.R.: *Introduction to Logic Programming*, 1988

Sterling, L., Shapiro, E.: *The Art of Prolog – Advanced Programming Techniques*, 1986

Clocksin, W.F., Melish, C.S.: *Programming in Prolog*, 1984



Tutorialy na nete

- J.R.Fisher: prolog :- tutorial
https://www.cpp.edu/~jrfisher/www/prolog_tutorial/pt_framer.html
- Prolog Tutorial (??)
<http://www.lix.polytechnique.fr/~liberti/public/computing/prog/prolog/prolog-tutorial.html>
- J. Wielemaker: SWI-Prolog for MS-Windows
<http://www.swi-prolog.org/windows.html>
- T.Plachetka (SK):
<http://www.dcs.fmph.uniba.sk/~plachetk/TEACHING/RLDB2012/prolog.pdf>
- Tutorialspoint, online interpreter
https://www.tutorialspoint.com/execute_prolog_online.php

Softvér

(čo používame)



- Prolog je beztypový jazyk
 - ani len náznak typovej inferencie a-la Haskell nebude
- Prolog je interpretovaný jazyk
 - s možnosťou kompilácie do vlastného bajt kódu
 - native nehrozí...
- SWI-Prolog – <http://www.swi-prolog.org/>
- Tutorialspoint – https://www.tutorialspoint.com/execute_prolog_online.php
- EclipseCLP – <https://eclipseclp.org/>

```
1 :- initialization(main).
2 %-- protekcie
3
4 pozna(jano, fero).
5 pozna(fero, jozo).
6 pozna(jano, ludovit).
7 pozna(zuzka, amalka).
8 pozna(zuzka, maria).
9 pozna(maria, fero).
10 pozna(cyril, metod).
11
12 protekcia(X,X).
13 protekcia(X,Y) :- protekcia(Y,X).
14 protekcia(X,Y) :- protekcia(X,Z), pozna(Z,Y).
15 protekcia(X,Y) :- protekcia(Z,Y), pozna(X,Z).
```

```
$gprolog --consult-file main.pg
GNU Prolog 1.4.4 (64 bits)
Compiled Feb 10 2017, 19:52:45 with gcc
By Daniel Diaz
Copyright (C) 1999-2013 Daniel Diaz
compiling /home/cg/root/4472213/main.pg for byte code...
/home/cg/root/4472213/main.pg compiled, 250 lines read - 33364 bytes writt
jano,fero
fero,jozo
jano,ludovit
zuzka,amalka
zuzka,maria
maria,fero
cyril,metod
warning: /home/cg/root/4472213/main.pg:1: user directive failed
| ?-
```

idea: program je logická formula

Príklady definícií predikátov

- podmnožina

$$\forall X \forall Y \text{ subset}(X,Y) \Leftrightarrow \forall U (U \in Y \Leftrightarrow U \in X)$$

- protekcia – reflexívny, symetrický a tranzitívny uzáver relácie
pozna(X,Y)

$$\begin{aligned} \forall X \forall Y \text{ protekcia}(X,Y) \Leftrightarrow \\ & X=Y \vee \\ & \text{protekcia}(Y,X) \vee \\ & \exists Z (\text{protekcia}(X,Z) \wedge \text{pozna}(Z,Y)) \vee \\ & (\text{protekcia}(Z,Y) \wedge \text{pozna}(X,Z)) \end{aligned}$$

Tu definitívne opustíme syntax predikátového počtu – teda kvantifikátory



Rodinné vzťahy

(definícia relácie ekvivalenciou)

- známe sú fakty v tvare $\text{rodic}/2$, $\text{zena}/1$, $\text{muz}/1$
predikátový_symbol/arita

$\text{otec}(X,Y) \Leftrightarrow \text{rodic}(X,Y) \wedge \text{muz}(X)$

$\text{dedo}(X,Y) \Leftrightarrow \text{otec}(X,Z) \wedge \text{rodic}(Z,Y)$

$\text{brat}(X,Y) \Leftrightarrow \text{rodic}(Z,X) \wedge \text{rodic}(Z,Y) \wedge \text{muz}(X)$

$\text{predok}(X,Y) \Leftrightarrow \text{rodic}(X,Y)$

$\text{predok}(X,Y) \Leftrightarrow \text{rodic}(X,Z) \wedge \text{predok}(Z,Y)$

$\text{predok}(U,V) ?$

$(U,V) = \{(jano,zuzka),(jano,palko),(zuzka,miska),(jano,miska)\}$

$\text{rodic}(jano, zuzka)$
 $\text{rodic}(jano, palko)$
 $\text{rodic}(zuzka, miska)$
 $\text{muz}(jano)$
 $\text{muz}(palko)$
 $\text{zena}(zuzka)$
 $\text{zena}(miska)$

$\text{muz}/1$, $\text{zena}/1$ sú unárne
 $\text{rodic}/2$ je binárny pred.symb

$\text{otec}(jano,zuzka)$

$\text{dedo}(jano,miska)$

$\text{brat}(palko,zuzka)$

$\text{predok}(X,miska) ?$
 $X = \{zuzka, jano\}$

Toto niekomu môže pripomínať úvod do Datalogu, alebo záver Mat-4



Rodinné vzťahy

(pôsobí to ako tvrdenia vo výrokovom počte)

rodic(jano, zuzka)
rodic(jano, palko)
rodic(zuzka, miska)
muz(jano)
muz(palko)
zena(zuzka)
zena(miska)

$\text{vrstovnik}(X, X) \Leftrightarrow \text{true}$

$\text{vrstovnik}(X, Y) \Leftrightarrow \text{rodic}(X_r, X) \wedge \text{rodic}(Y_r, Y) \wedge \text{vrstovnik}(X_r, Y_r)$

$\text{vrstovnik}(X, Y) ?$

$(X, Y) = \{(X=Y), (\text{zuzka}, \text{zuzka}), (\text{zuzka}, \text{palko}), (\text{palko}, \text{zuzka}), (\text{palko}, \text{palko}),$
 $(\text{miska}, \text{miska}), (\text{miska}, \text{miska})\}$

$\text{skutocnyVrstovnik}(X, Y) \Leftrightarrow \text{vrstovnik}(X, Y) \wedge \neg(X=Y)$

$\text{skutocnyVrstovnik}(X, Y) ?$

$(X, Y) = \{ (\text{zuzka}, \text{palko}), (\text{palko}, \text{zuzka}) \}$



Klauzálna forma

(z ekvivalencie píšeme len jednu implikáciu)

$A \leftarrow$

$\forall A$

$A \leftarrow B_1, B_2, \dots, B_n$

$\forall A \Leftarrow B_1 \wedge B_2 \wedge \dots \wedge B_n$

Klauzuly (Hornove):

$\text{protekcia}(X,Y) \leftarrow X=Y$

$\text{protekcia}(X,Y) \leftarrow \text{protekcia}(Y,X)$

$\text{protekcia}(X,Y) \leftarrow \text{protekcia}(X,Z) \wedge \text{pozna}(Z,Y)$

$\text{protekcia}(X,Y) \leftarrow \text{protekcia}(Z,Y) \wedge \text{pozna}(X,Z)$

odteraz už len prologovská syntax

Prolog:

$\text{protekcia}(X,X).$

$\text{protekcia}(X,Y) \text{ :- } \text{protekcia}(Y,X).$

$\text{protekcia}(X,Y) \text{ :- } \text{protekcia}(X,Z), \text{pozna}(Z,Y).$

$\text{protekcia}(X,Y) \text{ :- } \text{protekcia}(Z,Y), \text{pozna}(X,Z).$

- predikátové a funkčné symboly začínajú malým písmenom
- premenné začínajú veľkým písmenom
- `_` je anonymná premenná
- konjunkcia sa píše ako čiarka
- implikácia \leftarrow sa píše `:-`
- za vetou je bodka.

(trochu formálnejší pohľad)

- %- podčiarknuté sú funkčné symboly

Aritmetika "naozaj"

(výraz a hodnota/interpretácia výrazu)

- `times(0,Y,0).`
`times(X+1, Y, Z) :- times(X,Y,W), add(W,Y,Z).`

$$0*y = y$$
$$(\mathbf{x+1})*y = x*y + y$$

%-- zle

- `fact(0,s(0)).`
`fact(N+1,F) :- fact(N,F1), times(N+1, F1, F).`

$$0! = 1$$
$$(\mathbf{n+1})! = (n+1)*n!$$

%-- zle

X is Výraz je predikát v infixovej notácii, ktorý vyhodnotí aritmetický Výraz, a priradí do X, resp. porovná s hodnotu X.

- `times(0,_,0).`
`times(X, Y, Z) :- X>0, X1 is X-1, times(X1,Y,W), Z is W+Y.`

$$0*_ = y$$
$$x*y = (x-1)*y + y$$

- `fact(0,1).`
`fact(N,F) :- N>0, N1 is N-1, fact(N1,F1), F is N*F1.`

$$0! = 1$$
$$n! = n*(n-1)!$$

Haskell	Prolog
<code>[]</code>	<code>[]</code>
<code>H:T</code>	<code>[H T]</code>

Zoznamy

(nemá typy)

- konštruktory: `[]`, `[H | T]`
- konvencie: $[a_1, a_2, \dots, a_n | T] = [a_1 | [a_2 | \dots | [a_n | T]]]$
 $[a_1, a_2, \dots, a_n] = [a_1 | [a_2 | \dots | [a_n | []]]]$
- heterogénne $[1, "abc", janko] = [1, [97, 98, 99], janko]$
- nachádza sa v zozname (štandardný predikát member)
`member(X, [X|_]).`
`member(X, [_|Ys]) :- member(X,Ys).`

`?- member(X,[1,2,3]).`
`X = { 1,2,3 }`
- dĺžka zoznamu (štandardný predikát length)
`len([], 0).`
`len([_|Xs], L1) :- len(Xs,L), L1 is L+1.`

`?- len([1,2,3],X).`
`X = s(s(s(0)))`
- začiatok zoznamu
`prefix([], _).`
`prefix([X|Xs], [X|Ys]) :- prefix(Xs, Ys).`

`?- prefix([a,b],[a,b,b,a]).`
`yes`



Zoznamy - pokračovanie

- zoznam usporiadaný vzostupne
vzostupne([]).
vzostupne([_]).
vzostupne([X1,X2|Xs]) :- X1=<X2, vzostupne([X2|Xs]).
- zoznam usporiadaný zostupne
zostupne([]).
zostupne([_]).
zostupne([X1,X2|Xs]) :- X1>=X2, zostupne([X2|Xs]).
- usporiadaný zoznam
usporiadany(X) :- zostupne(X) ; vzostupne(X).

operátor **;** predstavuje disjunkciu (alebo)
operátor **,** predstavuje konjunkciu (a zároveň)

```
1200 xfx -->, :-  
1200 fx  :-, ?-  
1000 xfy ,  
990 xfx  :=  
900 fy   \+  
700 xfx  <, =, =<, ==, =\=,  
        >, >=, is  
600 xfy   :  
500 yfx   +, -, /\, \/, xor  
500 fx     ?  
400 yfx   *, /, //, div, <<,  
        >>, mod  
200 xfx   **  
200 xfy   ^  
200 fy    +, -, \
```



Zoznamová rekurzia

spojenie zoznamov, rekurzívna definícia predikátu append/3

- `append([], Ys, Ys).`
`append([X | Xs], Ys, [X | Zs]) :- append(Xs, Ys, Zs).`

- `?- append([1,2],[a,b],[1,2,a,b]).`

yes

- `?- append([1,2,3],[4,5],V).`

`V = [1,2,3,4,5]`

- `?- append(X, Y, [1,2,3]).`

`X = [], Y = [1,2,3] ;`

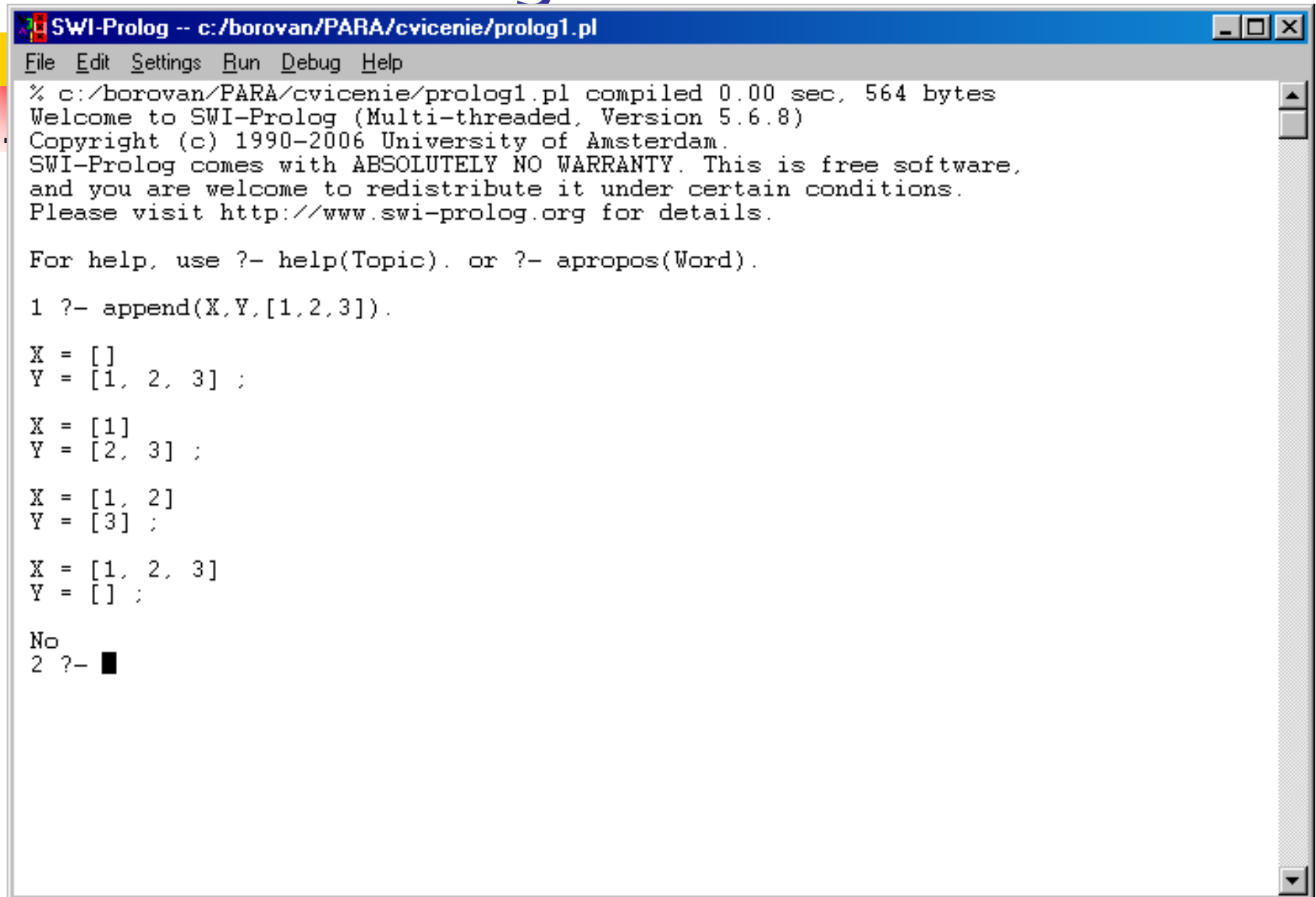
`X = [1], Y = [2,3] ;`

`X = [1,2], Y = [3] ;`

`X = [1,2,3], Y = [] ;`

no

SWI-Prolog



```
SWI-Prolog -- c:/borovan/PARA/cvicenie/prolog1.pl
File Edit Settings Run Debug Help
% c:/borovan/PARA/cvicenie/prolog1.pl compiled 0.00 sec, 564 bytes
Welcome to SWI-Prolog (Multi-threaded, Version 5.6.8)
Copyright (c) 1990-2006 University of Amsterdam.
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to redistribute it under certain conditions.
Please visit http://www.swi-prolog.org for details.

For help, use ?- help(Topic). or ?- apropos(Word).

1 ?- append(X,Y,[1,2,3]).
X = []
Y = [1, 2, 3] ;

X = [1]
Y = [2, 3] ;

X = [1, 2]
Y = [3] ;

X = [1, 2, 3]
Y = [] ;

No
2 ?- ■
```



reverse

- **reverse rekurzívny**

`reverse([], []).`

`reverse([X|Xs], Y) :- reverse(Xs, Ys), append(Ys, [X], Y).`

- **akumulátorová verzia**

`reverse(X, Y) :- reverse(X, [], Y).`

`reverse([], Acc, Acc).`

`reverse([X | Xs], Acc, Z) :- reverse(Xs, [X | Acc], Z).`

`reverse([1,2,3],Acc)`

`reverse([1,2,3],[],Acc)`

`reverse([2,3],[1],Acc)`

`reverse([3],[2,1],Acc)`

`reverse([], [3,2,1], Acc)`

`Acc = [3,2,1]`



Kvíz o zoznamoch

- neprázdny zoznam
`neprazdnyZoznam([_|_]).`
- aspoň dvojprvkový zoznam
`asponDvojPrvkovyZoznam([_,_|_]).`
- tretí prvok
`tretiPrvok([_,_,T|_],T).`
- posledný prvok zoznamu
`posledny([X],X).`
`posledny([_,Y|Ys],X):-posledny([Y|Ys],X).`
- tretí od konca
`tretiOdKonca(Xs,T) :- reverse(Xs,Ys),tretiPrvok(Ys,T).`
- prostredný prvok zoznamu, ak existuje
`prostredny(Xs,T):-append(U,[T|V],Xs),length(U,L), length(V,L).`



Rekurzia vs. iterácia

- vygeneruj zoznam čísel od 1 po N pre zadané N
- prvý pokus (rekurzívne riešenie) – $[N, \dots, 1]$
`nAzJedna(0,[]).`
`nAzJedna(N,[N|X]):-N>0,N1 is N-1,nAzJedna(N1,X).`

?- nAzJedna(4,L).
L = [4, 3, 2, 1] ;

- druhý pokus (iteratívne riešenie) – $[1, \dots, N]$
`jednaAzN(N,Res):-jednaAzN(N,[],Res).`
`jednaAzN(0,Acc,Res):-Acc=Res.`
`jednaAzN(N,Acc,Res):-N>0,N1 is N-1,jednaAzN(N1,[N|Acc],Res).`
... alebo počítajme *dohora*

?- jednaAzN(5,L).
L = [1, 2, 3, 4, 5] ;



Konverzia zoznamu cifier na číslo

- konverzia zoznamu cifier na číslo

zoznamToInt([],0).

zoznamToInt([X|Xs],C) :- zoznamToInt(Xs,C1), C **is** 10*C1+X.

?- zoznamToInt([1,2,3,4],X).
X = 4321 ;

- konverzia čísla na zoznam cifier

intToZoznam(0,[]).

intToZoznam(C,[X|Xs]) :- C > 0,

X **is** C **mod** 10, C1 **is** C // 10, intToZoznam(C1,Xs).

?- intToZoznam(4321,X).
X = [1, 2, 3, 4] ;



Konverzia s akumulátorom

- akumulátorová verzia konverzie zoznamu cifier na číslo

zoznamToInt2(X,Res) :- zoznamToInt2(X,0,Res).

zoznamToInt2([],C,Res) :- Res = C.

zoznamToInt2([X|Xs],C,Res) :- C1 is 10*C+X,
zoznamToInt2(Xs,C1, Res).

?- zoznamToInt2([1,2,3,4],X).

X = 1234 ;

- akumulátorová verzia konverzie čísla na zoznam cifier

intToZoznam2(X,Res) :- intToZoznam2(X,[],Res).

intToZoznam2(0,Res,Res).

intToZoznam2(C,Xs,Res) :- C > 0,

X is C mod 10, C1 is C // 10, intToZoznam2(C1,[X|Xs],Res).

?- intToZoznam2(1234,X).

X = [1, 2, 3, 4] ;



flat alias splošti

Sploštenie heterogénneho zoznamu s viacerými úrovňami do jedného zoznamu všetkých prvkov

- **naivné riešenie**

```
flat([X|Xs],Ys):flat(X,Ys1),flat(Xs,Ys2), append(Ys1,Ys2,Ys).  
flat(X,[X]):atomic(X),X \= [].  
flat([],[]).
```

```
?- flat([1,[2,[],[3,[4]]]], X).  
X = [4, 3, 2, 1] ;
```

- **akumulátorové riešenie (odstraňujeme append):**

```
flat1(X,Y):flat(X,[],Y).  
flat1([X|Xs],Ys1,Ys):flat1(X,Ys1,Ys2),flat1(Xs,Ys2,Ys).  
flat1(X,Ys,[X|Ys]):atomic(X),X \= [].  
flat1([],Ys,Ys).
```

```
?- flat1([1,[2,[],[3,[4]]]], X).  
X = [4, 3, 2, 1] ;
```



Prefix a sufix zoznamu

- začiatok zoznamu, napr. `?-prefix([1,a,3],[1,a,3,4,5])`

`prefix([], _).`

`prefix([X|Xs], [Y|Ys]) :- X = Y, prefix(Xs, Ys).`

`prefix([X|Xs], [X|Ys]) :- prefix(Xs, Ys).`

- koniec (chvost) zoznamu `?-sufix([3,4,5],[1,2,3,4,5])`

`sufix(Xs,Xs).`

`sufix(Xs,[_|Ys]):-sufix(Xs,Ys).`

- koniec zoznamu, ak už poznáme reverse

`sufix(Xs,Ys):-reverse(Xs,Xs1), reverse(Ys,Ys1), prefix(Xs1,Ys1).`



Podzoznam zoznamu

- **súvislý podzoznam**, napr. `?-sublist([3,4,5],[1,2,3,4,5,6])`

```
sublist1(X,Y) :- append(_,X,V),append(V,_,Y).  
sublist2(X,Y) :- append(V,_,Y),append(_,X,V).
```

- **ešte raz súvislý podzoznam**, keď poznáme sufix, prefix, ...

```
sublist3(Xs,Ys):-prefix(W,Ys),sufix(Xs,W).  
sublist4(Xs,Ys):-sufix(W,Ys),prefix(Xs,W).
```

- **nesúvislý podzoznam**, tzv. vybratá podpostupnosť

```
subseq([X|Xs],[X|Ys]):-subseq(Xs,Ys).  
subseq(Xs,[_|Ys]) :- subseq(Xs,Ys).  
subseq([],_).
```



Práca so zoznamom

- definujte predikát **index**(X,Xs,I), ktorý platí, ak $Xs_i = X$
index(X,[X|_],1).
index(X,[_|Ys],I):-index(X,Ys,I1),I is I1+1.

?- index(b,[a,b,c],I).
I = 2 ;

?- index(X,[a,b,c],I).
X = a, I = 1 ;
X = b, I = 2 ;
X = c, I = 3 ;
- definujte predikát **select**(X,Y,Z), ktorý vyberie všetky možné prvky X zo zoznamu Y, a výsledkom je zoznam Z
select(X,[X|Xs],Xs).
select(X,[Y|Ys],[Y|Zs]):-select(X,Ys,Zs).

?- select(X,[a,b,c],Z).
X = a, Z = [b, c] ;
X = b, Z = [a, c] ;
X = c, Z = [a, b] ;
- definujte predikát **delete**(X,Y,Z) ak Z je Y-[X]
delete(X,Y,Z):-select(X,Y,Z).
- definujte predikát **insert**(X,Y,Z), ktorý vsunie prvok X do zoznamu Y (na všetky možné pozície), výsledkom je zoznam Z
insert(X,Y,Z):-select(X,Z,Y).



Permutácie a kombinácie

■ definujte predikát `perm(X,Y)`, ktorý platí, ak zoznam `Y` je permutáciou zoznamu `X`

```
perm(Xs,[H|Hs]):-select(H,Xs,W),perm(W,Hs).  
perm([],[]).
```

```
?- perm([1,2,3,4],Xs).  
Xs = [1, 2, 3, 4] ;  
Xs = [1, 2, 4, 3] ;  
Xs = [1, 3, 2, 4] ;  
Xs = [1, 3, 4, 2]
```

- iná verzia, miesto `select/delete` robíme `insert`
`perm2([],[]).`
`perm2([X|Xs],Zs):-perm2(Xs,Ys),insert(X,Ys,Zs).`

```
?- L=[_,_,_,_],  
   comb(L,[1,2,3,4,5,6]).
```

- definujte predikát `comb(X,Y)`, ktorý platí, ak zoznam `X` je kombináciou prvkov zoznamu `Y`
- `comb([],_).`
`comb([X|Xs],[X|T]):-comb(Xs,T).`
`comb([X|Xs],[_ |T]):-comb([X|Xs],T).`
`comb(Xs,[_ |T]):-comb(Xs,T).`

```
L = [1, 2, 3, 4] ;  
L = [1, 2, 3, 5] ;  
L = [1, 2, 3, 6] ;  
L = [1, 2, 4, 5] ;  
L = [1, 2, 4, 6] ;  
L = [1, 2, 5, 6] ;  
L = [1, 3, 4, 5] ;  
L = [1, 3, 4, 6] ;  
L = [1, 3, 5, 6] ;  
L = [1, 4, 5, 6] ;  
L = [2, 3, 4, 5] ;  
L = [2, 3, 4, 6] ;  
L = [2, 3, 5, 6] ;  
L = [2, 4, 5, 6] ;  
L = [3, 4, 5, 6] ;
```

- to bolo na minulej prednáške ako `subseq` 😊



Rekapitulácia

- program je konečná množina (Hornových) klauzúl tvaru:
A.

alebo

$A:-B_1, B_2, \dots, B_n.$

- klauzula predstavuje všeobecne kvantifikovanú implikáciu,
- dotaz (cieľ) je tvaru $?-B_1, B_2, \dots, B_n$ (a obsahuje hľadané premenné),
- v programe deklarujeme tvrdenia o pravdivosti predikátov - čo sa z programu nedá odvodiť, neplatí bez toho, aby sme to deklarovali,
- premenné začínajú veľkým písmenom alebo `_`
- funkčné a predik.symboly začínajú malým písmenom,
- Prolog má zoznamy s konštruktormi `[]` a `[H|T]`
- Prolog je beztypový jazyk (pozná čísla, atómy-termu, zoznamy),
- klauzule sa skúšajú v textovom poradí (od hora dole),
- klauzule sa skúšajú všetky, ktoré môžu byť použiteľné,
- Prolog vráti všetky riešenia problému, ktoré nájde,
- Prolog nemusí nájsť riešenie, ak sa zacyklí