Poznámky z přednášek
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

Neprocedurální programování

Peter Černo, 2010
petercerno@gmail.com

Garant: RNDr. Rudolf Kryl
E-mail: Rudolf.Kryl@mff.cuni.cz
Domácí stránka: `http://ksvi.mff.cuni.cz/~kryl/`

Anotace: Přednáška je věnována neprocedurálnímu programování. Většina semestru je věnována programování v jazyku Prolog, ve kterém studenti i ladí zápočtové programy. Informativně se studenti seznámí i s jazykem LISP a neprocedurálními částmi programovacích systémů.

Sylabus:

1. Tvar programu v Prologu a jeho interpretace, unifikace, backtracking. Deklarativní a operační sémantika programu.

2. Práce se seznamy, různé variace základních predikátů pro práci s nimi, efektivní obracení seznamu.

3. Aritmetika v Prologu, vyčíslení kontra unifikace.

4. Základní standardní predikáty (atom, atomic, number, name, func, arg, ...) predikáty pro práci s databází (assert, retract), predikáty grupování termů (bagof, setof) a jejich použití v programech). predikáty.

5. Vstup a výstup, definice operátorů.

6. Řez a jeho vliv na semantiku programů a jejich efektivitu. Definice negace a její vlastnosti.

7. Efektivita programů v prologu, neúplně definované datové struktury (např. rozdílové seznamy).

8. Logické programování a Prolog.

9. Funkcionální programování, základy jazyka LISP a programování v něm.

10. Haskell: zakladní syntax programů, definice typů a datových struktur, polymorfismus, porovnávání se vzorem (pattern matching), rekurze, líné vyhodnocování, funkce vyšších řádů, typové třídy a instance

Literatura:

1. Bratko I.: PROLOG Programming for Artificial Intelligence Addison-Wesley, Reading, Massachussets, 1986 ISBN 0-201-14224-4

2. Harold Abelson, Gerald Jay Sussman, Julie Sussman : Structure and Interpretation of Computer Programs Mc Graw-Hill Book Company 1985 ISBN 0-07-000-422-6

3. Paul Hudak, Joseph H. Fasel: A Gentle Introduction to Haskell, `http://www.haskell.org/`

This page is intentionally left blank.

06.10.2006 Fr
PETER ČERNO

PRG005 NEPROCEDURÁLNÍ PROGRAMOVÁNÍ

ZÁPOČET + SKÚŠKA (SKÔR PROGRAMOVANIE)

NEEXISTUJE POJEM PREMENNEJ AKO MIESTA V PAMÄTI

NEPROC. PROG. ⊂ LOGICKÉ PROG. — PROLOG (70. ROKY)
FUNKCIONÁLNE PR. — LISP (50.-60. ROKY)
HASKEL

DOPORUČENÁ LITERATURA

P. ŽIRKŮ : PROGRAMOVÁNÍ V JAZYKU PROLOG

SWI PROLOG

```
?- muz(eva).
no
?- zena(X).
X = eva ↵
;
X = helena
;
no
```

```
?- rodic(X, hugo).
X = adam ;
X = eva ;
no
?- rodic(eva, X).
X = jozef ↵
?- rodic(X, hugo), muz(X).
X = adam ↵
```

```
otec(Otec, Dite) :- rodic(Otec, Dite),
                    muz(Otec).
```

```
?- otec(X, hugo).
?- rodic(X,Y).
     X=adam, Y=josef.

?- rodic(X,_).
     X=adam ;
     X=adam
```

```
rodic(X) :- rodic(X,_).

bab(Bab, Vnouce) :- matka(Bab, Rodic),
    rodic(Rodic, Vnouce).
matka(M,D) :- rodic(M,D), zena(M).
sestra(Kdo, Ci) :- rodic(X, Kdo), rodic(X, Ci), zena(Kdo),
      Kdo \= Ci.
```

NEGACIA = NIEČO SA NEDÁ ODVODIŤ

06.10.2006 Fr
PETER ČERNO

PRG005 NEPROC. PROG.

JEDINÁ DAT. ŠTRUKTÚRA

ZOZNAM < PRÁZDNY [] / NEPRÁZDNY . (Hlava, Ocas)

Hlava ↑ PRVÝ PRVOK; Ocas ↑ ZVYŠOK ZOZNAMU

.(a,.(b,.(c,[])))

PÍŠE SA TO INAK [Hlava | Ocas]

KONVENCIE:

1) PRED | MÔŽE BYŤ AJ VIAC PRVKOV OD ZAČIATKU ZOZNAMU (ALE NIE JE TO ZOZNAM)

2) ZA | MOŽNO VYNECHAŤ PRÁZDNY ZOZNAM

PR.:

[a | [b | [c | []]]]

[a, b | [c]]

[a, b, c]

[a | [b, c]]

[[a, b], c | [d, e]]

JE TO ZOZNAM = [[a, b], c, d, e]

[a, b | c, d]

TOTO NIE JE ZOZNAM

ZA | MUSÍ ÍSŤ ZOZNAM

```
prvek(Co, [Co|_]).
prvek(Co, [_|X]) :- prvek(Co, X).

prvyprvok(Co, [Co|_]).

posledny_prvok(Co, [Co]).
posledny_prvok(Co, [_|X]) :-
    posledny_prvok(Co, X).

lbnl(Co, [Co,_]).
lbnl(Co, [_|X]) :- lbnl(Co, X).

spoj([], L, L).
spoj([X|V], L, [X|W]) :- spoj(V, L, W).

?- spoj([a,b], [1,2], X).
    X = [a,b,1,2] ;
    no

?- prvok(X, [a,b,c]).
    X = a ;
    X = b ;
    X = c ;
    no

?- prvok(a, X).
    X = [a|_] ;
    X = [_, a|_]
```

06.10.2006 Fr
PETER ČERNO

PRG005 NEPROC. PROG.

```
% prvok2(Co, Ceho)
prvok2(X, Y) :- spoj(_, [X|_], Y)

% perm(S, Perm)
perm([], []).
perm([X|V], S) :- perm(V, PV), insert(X, PV, S).

% insert(Co, Kam, Vysl)
insert(X, T, [X|T]).
insert(X, [Y|T], [Y|W]) :- insert(X, T, W).
```

ÚLOHA: SPRAVIŤ PREDIKÁT prostredný

```
otoc([], [])

otoc([X|V], S) :- otoc(V, OV),
        spoj(OV, [X], S).
```

ČASOVÁ ZLOŽITOSŤ: $O(N^2)$

DÁ SA TO SPRAVIŤ V ČASE $O(N)$

13.10.2006 Fr
PETER ČERNO

PRG005 NEPROCEDURÁLNE PROGRAMOVANIE

PROSTREDNÝ PRÍKLAD:

+ VSTUPNÝ
- VÝSTUPNÝ
? ĽUBOVOĽNÝ

```
% pul (+Seznam, -SezDelkyPul)

pul([_,_|T], [a,Z]) :- pul(T,Z).
pul([_], []).
pul([], []).

?- pul([0,b,c,d,e], X).
   X = [a,a] ↵

%umaz (+Odkud,+Kolik, -Zbytek)
umaz([_|T], [_|U], [_|W]) :- umaz(T,U,W).
umaz(K, [], K).

% drpul(+Seznam, -Dr)
drpul(S,D) :- pul(S,P), umaz(S,P,D).
```

2. RIEŠENIE:

```
% rozpul (+Seznam, -Pru, -Dr).
rozpul(S,P,D) :- r(S,S,P,D).
r([_,_|T], [X|V], [X|PA],D) :- r(T,V,PA,D)
r([_], D, [],D).
r([], D, [],D).
```

ROZMYSLIEŤ: ROZDELIŤ NA TRI ČASTI

OTÁČANIE V LIN. ČASE

TECHNIKA AKUMULÁTOR

```
% ot(+S, -OS)        ↙ AKUMULÁTOR
ot(S, OS) :- o(S, [], OS).
ot2([X|T], A, OS) :- ot2(T, [X|A], OS).
ot2([], A, A).
```

ČO ZNAMENÁ, ŽE DVA TERMY MATCHUJÚ ?
PROCES UNIFIKÁCIE ?

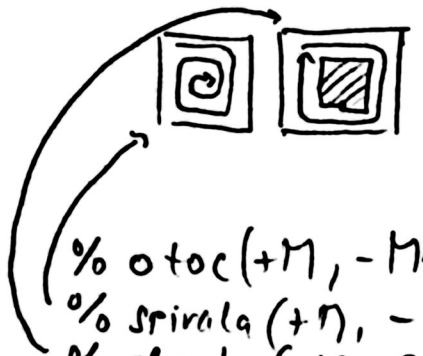
ZADOST' O UNIF.

```
X = f(a).
```

```
svisla(usecka(bod(X, _), bod(X, _))).
vodorovna(usecka(bod(_, Y), bod(_, Y))).
```

MATICA ... ZOZNAM ZOZNAMOV

$\begin{bmatrix} a & b \\ c & d \\ e & f \end{bmatrix}$ .. [[a,b], [c,d], [e,f]]



```
% otoc(+M, -M1)
% spirala(+M, -S)
% skrutka(+M, -S, -P)
```

```
% obdlznik (+ Matica)
% stvorec (+ Matica)
% diag (+ Stvorcova Matica, - Diag)
diag([[H|R]|M], [H|D]) :- urls(M, M1), diag(M1, D).
diag([], [])
urls(M, M1) :- ur1(M, M1, S).
ur([[H|T]|Z], [T|M], [H|S]) :-
    ur(Z, M, S).
ur(M, [], []) :- praz(M)
```

```
praz([]).
praz([]|T) :-
    praz(T)
```

20.10.2006 Fr
PETER ČERNO

PRG005 NEPROCEDURÁLNE PROGRAMOVANIE

```
?- X = 1+2.
   X = 1+2
```

1) POKÚSIM SA SPOČÍTAŤ HODNOTU V
AK SA TO NEPODARÍ ⇒ BEHOVÁ CHYBA

ARITMETICKÝ VÝRAZ SA SKLADÁ Z OPERÁCIÍ, ČÍSIEL A PREMENNÝCH VIAZANÝCH NA ČÍSLO

AK SA TO PODARÍ ⇒ TJ. V DÁVA W

2) AK JE X VOĽNÁ, POTOM SA SUBSTITUUJE W ZA X
AK NIE JE X VOĽNÁ, POTOM SA OTESTUJE X = W. < yes / no

is

X is V
↑ VÝRAZ
PREMENNÁ

```
?- X is 7, Y is X+3.
   X = 7, Y = 10

?- X is Y+3, Y is 5.
   BEHOVÁ CHYBA

?- X+1 = 3+Y.        (ŽIADOSŤ O UNIFIKÁCIU)
   X = 3, Y = 1

?- X+1 is 7.
   SYNTAX ERROR

?- Y is 5, X is Y+3, X is X+1.
   NO

?- X is X+1
   BEHOVÁ CHYBA
```

$V_1 \; \square \; V_2$

$\square$ ::= ARITMETICKÉ =
=\= ARITMETICKÉ ≠
<
>
<=
>=

1) VYHODNOTÍM
$V_1 \to w_1$
MH. $V_2 \to w_2$

KEĎ SA JEDNO Z TOHO NEPODARÍ ⇒ BEHOVÁ CHYBA

2) $w_1 \; \square \; w_2$

```
% gcd(C1, C2, nsd(C1, C2))
gcd(X,X,X).
gcd(X,Y,D):- X>Y, X1 is X-Y, gcd(X1,Y,D).
gcd(X,Y,D):- Y>X, gcd(Y,X,D).
```

NEMOŽNO NAPÍSAŤ gcd(X-Y, Y, D)
↳ BUDE UNIFIKOVAŤ, ALE NEBUDE POČÍTAŤ

```
% min(A,B,C)
min(A,B,A):- A<B.
min(A,B,B):- B<=A.

% min(+Zoznam, -Minimum)
min([X|L], M):- minzoz(X,L,M).
minzoz(X,[],X).
minzoz(X,[Y|T],M):- Y<X, min(Y,T,M).
minzoz(X,[Y|T],M):- X<=Y, minzoz(X,T,M).
```

20.10.2006 Fr
PETER ČERNO

PRG005 NEPROCEDURÁLNE PROGRAMOVANIE

```
% len (+Zoznam, -Dlzka).
len ([], 0).
len ([_|T], D):- len (T, D1), D is D1 +1.
```

KEĎ SME ZADEFINOVALI TIEŽ 2 KLAUZULE ⇒ BEHOVÁ CHYBA

KEĎ SME NAPÍSALI D = D1+1,
POTOM
```
?- len ([a,b,c], D).
D= 0 + 1 +1 +1
```

LEXIKOGRAFICKÉ USPORIADANIE

6 4 1 5 3 2
6 4 2 1 3 5
6 4 2 1 5 3
6 4 2 3 1 5
...

```
naslperm (Perm, NPerm):-
    otoc (Perm, Of Perm),
    rozloz (Of Perm, RastZac, X, Zvysok),
    zarad (RastZac, X, Y, NRastZac)
    concrev (Zvysok, [Y|NRastZac], NPerm).
```

STROM

nil ATOM PRE PRÁZDNY STROM

t(L, X, R)


```
show(T) :- show2(T, 0)
show2(nil, _).
show2(t(L, X, R), Ind) :-
    Ind2 is Ind + 2,
    show2(R, Ind2),
    tab(Ind), write(X), nl,
    show2(L, Ind2).
```

% in(X, T): X JE V STROME T.

```
in(X, t(_, X, _)).
in(X, t(L, _, _)) :- in(X, L).
in(X, t(_, _, R)) :- in(X, R).
```

20.10.2006 Fr
PETER ČERNO

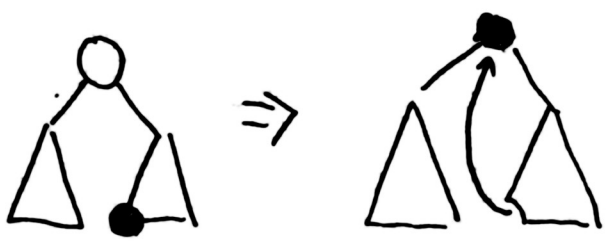
PRG005 NEPROCEDURÁLNE PROGRAMOVANIE

```
% vloz (+Co, +kam, -Vysl)
vloz (X, nil, t(nil, X, nil)).
vloz (X, t(L,X,R), t(L,X,R)).
vloz (X, t(L,V,R), t(L1,V,R)) :-
    X < V, vloz (X, L, L1).
vloz (X, t(L,V,R), t(L,V,R1)) :-
    X > V, vloz (X,R, R1).

? vypust (Co, Odkial, Vysl) :- vloz (Co, Vysl, Odkial).
```

NEFUNGUJE - 2. KLAUZULA TO NEDOVOLÍ

2. PROBLÉM: PROGRAM VKLADÁ VŽDY DO LISTU
=) BEZ 2. KLAUZULE DOKÁŽE VYPUSTIŤ IBA LIST !!!

VYPÚŠŤANIE INAK

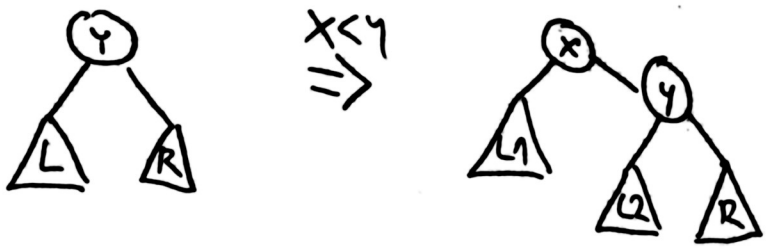


```
del(t(nil,X,R), X, R).
del(t(L,X,nil), X, L).

del(t(L,X,R), X, t(L,Y,R1)) :-
    delmin(R,Y,R1).
del(t(L,K,R), X, t(L1,K,R)) :-
    X<K, del(L,X,L1).
del(t(L,K,R), X, t(L,K,R1)) :-
    X>K, del(R,X,R1).

delmin(t(nil,Y,R), Y, R).
delmin(t(L,K,R), Y, t(L1,K,R)) :-
    delmin(L,Y,L1).
```

AKO SPRAVIŤ VKLADANIE TAK, ABY BOLO MOŽNÉ MYSŤAŤ ?

- VLOŽIŤ DO KORENA
- VLOŽIŤ DO ĽAVÉHO
- VLOŽIŤ DO PRAVÉHO

OR (NEDETERMINIZMUS)



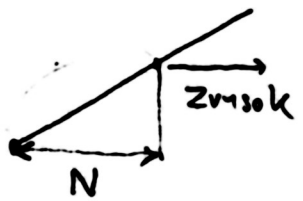
20.10.2006 Fr
PETER ČERNO

PRG005 NEPROCEDURÁLNE PROGRAMOVANIE

```
add(D,X,D1) :- addroot(D,X,D1).
add(t(L,Y,R), X, t(L1,X,R)) :-
        X<Y, add(L,X,L1).
...
addroot(nil, X, t(nil, X, nil)).
addroot(t(L,X,R), X, t(L,X,R)).
addroot(t(L,Y,R), X, t(L1,X,t(L2,Y,R1))) :-
        X<Y, addroot(L,X,t(L1,X,L2)).
addroot(t(L,Y,R), X, t(L1,X,t(L2,Y,R1))) :-
        X>Y, addroot(R,X,t(R1,X,R2)).
```

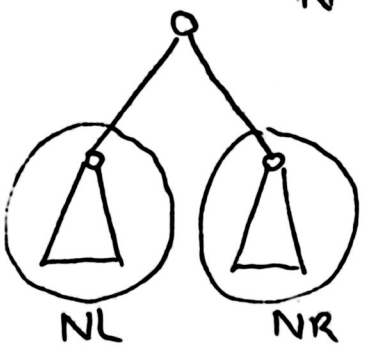
POKONALÉ VYVÁŽENÍ STROM

```
% vytvor(+Zoznam, +N, -DobreVyvStrom, -Zvysok)
% +Zoznam je zoznam RAJTSCICH prvkov
```



CHCEME SPRAVIT STROM

N PRVKOV



N1 = N - 1
NL = N1 / 2
NR = N1 - NL

```
vytvor(Z, 0, nil, Z).
vytvor(Z, N, t(L,V,R), Z) :- N1 is N-1, NL is N1 div 2,
  NR is N1 - NL,
  vytvor(Z, NL, L, [V|Z1]),
  vytvor(Z1, NR, R, Z).
```

GRAF

```
% vrch(Vrchol, Zoznam susedov)
% priechod(+Graf, -Zoznam)
  priechod
```

NA DOMA!

```
% listy(+Strom, -ZoznamListov)
  listy(nil, []).
  listy(t(nil, X, nil), [X]).
  listy(t(L, _, R), S) :- listy(L, S1), listy(R, S2),
      conc(S1, S2, S).
```

! SPRAVIT TO BEZ conc

27.10.2006 Fr
PETER ČERNO

PRG005 NEPROC. PROGRAMOVANIE

ÚLOHA: SPRAVIŤ quicksort BEZ ZRETAZENIA !!!

PRECHOD STROMOM < DO HĹBKY / DO ŠÍRKY

```
prejdi(nil, []).
prejdi(Strom, Vysl):-
    spracuj([Strom], Vysl).
spracuj([], []).
spracuj([t(L,V,R)|Z], [V|Kon]):-
    pridaj(L, R, Z, Z1), spracuj(Z1, Kon).
```

DO HĹBKY: pridaj(L,R,Z,Z1) NA ZAČIATOK
ŠÍRKY: NA KONIEC

```
pzac(nil, nil, Z, Z).
pzac(nil, R, Z, [R|Z]).
pzac(L, nil, Z, [L|Z]).
pzac(L, R, Z, [L,R|Z]).

pkon(nil, nil, Z, Z).
pkon(nil, R, Z, Zn):-
        conc(Z, [R], Zn)
        ...
```

DEKLARATÍVNA A PROCEDURÁLNA SÉMANTIKA

DEKLARATÍVNA

DEKLARATÍVNY VÝZNAM PROGRAMU JE KONJUNKCIA FORMÚL PRE JEDNOTLIVÉ KLAUZULE

$p :- r_1, r_2, \ldots, r_n.$	$r_1 \wedge r_2 \wedge \ldots \wedge r_n \Rightarrow p$
$p.$	p

⚠ KONEČNOSŤ
PARCIÁLNA SPRÁVNOSŤ - KEĎ PROG. SKONČÍ, DÁ SPRÁVNU ODPOV.

```
predok (Pred, Pot) :- rodic (Pred, Pot).
predok (Pred, Pot) :- rodic (Rod, Pot),
                      predok (Pred, Rod).
```

} DEKLARATÍVNE SPRÁVNY PROGRAM

```
predok1 (Pred, Pot) :- predok (Pred, Rod),
                       rodic (Rod, Pot).
predok1 (Pred, Pot) :- rodic (Pred, Pot).
```

} TAKTIEŽ DEKLARATÍVNE SPRÁVNY PROGRAM

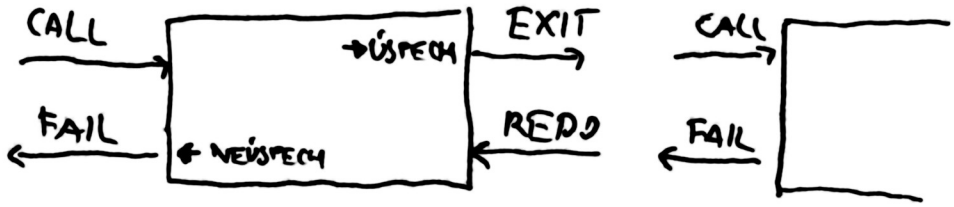
(NIKDY NESKONČÍ)

2/3 281 913 925
724 051 203 HASLEROVA

27.10.2006 Fr
PETER ČERNO

PRG005 NEPROC. PROGRAMOVANIE

KRABICKOVÝ MODEL
KAŽDÉMU VOLANIU ZODPOVEDÁ JEDNA KRABICKA



?- consult (subor)

.-:-consult (subor) % V PROGRAME

?- halt

ARITA

trace /0
notrace /0

spy /1
nospy /1

AK SME ROZLIŠILI otec/2 a otec/3

spy (otec/2)

HELP ABORT
DEBUGER : ↵, h↵, a↵

RADY: LADIŤ KAŽDÚ PROCEDÚRU ZVLÁŠŤ
LADIŤ OD JEDNODUCHÝCH PROC. K ZLOŽITEJŠÍM
LADIŤ NA MALÝCH VSTUPNÝCH DATACH
LADIŤ V KLASICKOM TXT REŽIME → MÁME HISTÓRIU!

ŠTANDARDNÉ PREDIKÁTY :

name (A, L)

+	−
−	L
A	L

A atom
L zoznam ASCII kódov

functor (X, Y, Z)

+	−	−
?	A	E

X je term
Y jeho funktor
Z jeho arnosť

arg (N, Term, A)

E	+	?

A je n-tý argument

=..

Term	=..	L
+		L
+		−
−		L

L je zoznam
hlava je funktor
telo je zoznam argumentov

% subst (+PodTerm1, +Term1, +PodTerm2, -Term2)
všetky výskyty ↑ v ↑ nahrad ↑ a výsledok ↑

27.10.2006 Fr
PETER ČERNO

PRLOOS NEPROC. PROGRAMOVANIE

ROVNOSTI:

= ŽIADOSŤ O UNIFIKÁCIU

\= NEDÁ SA UNIFIKOVAŤ

=:= ARITM. ROVNOSŤ

=\= -II- NEROVNOSŤ

== TOTOŽNOSŤ

\==

?- X == Y.

no

?- X is 7, Y is 5+2, X == Y.

yes

=.. PREVOD term ↔ zoznam

ÚLOHA ELIZA WEISENBAUM [1966]

PREČÍTAJ VSTUP

DOKEDY VSTUP NIE JE bye

VYBER DUDÍCU podnet - reakcia

NÁJDI KORESPONDENCIU VSTUP - podnet

03.11.2006 Fr
PETER CERNO

PRG005 NEPROC. PROGRAMOVANIE

REZ !

P, Q, R, !, S, T, U

```
manzel(kto, koho) :- plesatoy(koho), !, fail.
manzel(kto, koho) :- muz(koho).
```

```
bubble(S, S1) :- vymen(S, NS), !, bubble(NS, S1).
bubble(S, S).

vymen([X,Y|T], [Y,X|T]) :- X>Y, !.
vymen([Z|T], [Z|T1]) :- vymen(T, T1).
```

```
min(X, Y, X) :- X<=Y.
min(X, Y, Y) :- X>Y.
```

```
min(X, Y, X) :- X=<Y, !.
min(X, Y, Y).
```

ALE!!

```
?- min(3,5,5).
   yes.          % !!!
```

OPRAVA:

```
min(X,Y,Z) :- X=<Y, !, Z=X.
min(X,Y,Y).
```

```
p :- q, r.
p :- s.
```

$q \wedge r \vee s \Rightarrow p$

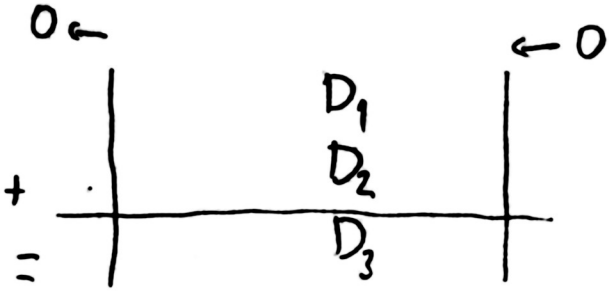
```
p :- q, !, r.
p :- s.
```

$q \Rightarrow (r \Rightarrow p)$

$\neg q \Rightarrow (s \Rightarrow p)$

ALGEBROGRAM

```
  DONALD
+ GERALD
--------
  ROBERT
```



$D_3 = (D_1 + D_2 + C_1) \bmod 10$

$C = (D_1 + D_3 + C_1) \operatorname{div} 10$

var(X) .. USPEJE VTEDY, KEĎ X JE VOĽNÁ PREM.

nonvar(X) .. USPEJE KEĎ X NIE JE PREM. ALEBO JE ALE NIE VOĽNÁ

03.11.2006 Fr
PETER ČERNO

PRG005 NEPROC. PROGRAMOVANIE

ROZDIELOVÉ ZOZNAMY

[] L-L var(L)
[a,b,c] [a,b,c|L]-L var(L)

conc(A-B, B-C, A-C).

PREMENNÁ JE OZNAČENIE MIESTA, KAM MOŽNO NIEČO SUBSTITUOVAŤ!

```
rozd2list(L-L, []) :- var(L), !.
rozd2list([X|T]-L, [X,S]) :- var(L), !,
    rozd2list(T-L, S).
```

% spirala(+Matica, -Spirala).

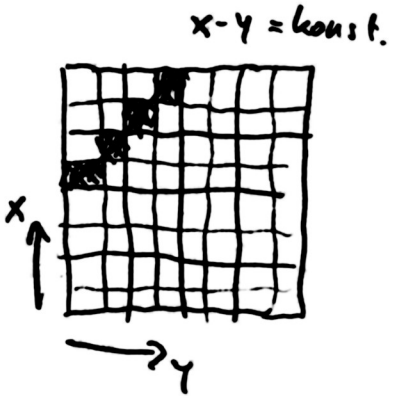
% invspirala(+M, +N, +Spirala, -Matica).

```
% anvektor (+N, -V).
anvektor(0, []).
anvektor(N, [_|T]) :- N>0, N1 is N-1,
    anvektor(N1, T).

% anmatica (+M, +N, -Matica).
    anmatica(0, _, []).
    anmatica(M, N, [R|T]) :- M>0, M1 is M-1,
        anvektor(N, R), anmatica(M1, N, T).

 invsp(M, N, Spirala, Matica) :-
    anmatica(M, N, Matica),
    spirala(Matica, Spirala).
```

PROBLÉM 8 DÁM

```
res(Ylist, Dx, Dy, Du, Dv)
```

VOĽNÉ RIADKY
VOĽNÉ STĹPCE
HLAV. DIAG. x-y
KLESAJÚCE DIAG. x+y




```
res(Ylist) :- res(Ylist, [1,2,...,8], [1,2,...,8],
        [-7,...,7], [2,...,16]).
```

ČÍM ZAPLATÍME, KEĎ NEPOUŽÍVAME POLIA

03.11.2006 Fr
PETER ČERNO

TRIEDENIE ZLIEVANÍM

```
% merge(+X,+Y,-Z).
merge([X|XL],[Y|YL],[X|ZL]):-
  X<Y, merge(XL,[Y|YL],ZL).
  ...

merge([],[X|XL],[X|XL]).
merge(XL,[],XL).
```

→ KEBY SME NAPÍSALI `merge([],X,X).`
DOSTALI BY SME NEDETERMINISTICKÝ merge.

KEĎ JE PROCEDÚRA DETERMINISTICKÁ,
KOMPILÁTOR JE SCHOPNÝ AKOBY ODSTRÁNIŤ REKURZIU.

```
% subst(+Podterm1,+Term1,+Podterm2,-Term2)
subst(T,T,T1,T1).
subst(_,T,_,T):- atomic(T),!.
subst(Sub,Term,Sub1,Term1):-
  Term=..[F|Arg],
  substlist(Sub,Arg,Sub1,Arg1),
  Term1=..[F,Arg1].
```

```
substlist(_, [], _, []).
substlist(Sub, [Term|Rest], Sub1, [Term1|Rest1]):-
    subst(Sub, Term, Sub1, Term1),
    substlist(Sub, Rest, Sub1, Rest1).
```

INSERT SORT

```
ins_sort([], []).
ins_sort([X|T], S):-
    ins_sort(T, ST),
    vloz(X, ST, S).

vloz(X, [Y|S], [Y|S1]):-
    X > Y, !, vloz(X, S, S1).
vloz(X, S, [X|S]).
```

10.11.2006 Fr
PETER ÖRNY

PRG005 NEPROC. PROGRAMOVANIE

VSTUPNÝ / VÝSTUPNÝ STREAM — STANDARDNÝ / AKTUÁLNY — VSTUP/VÝSTUP

STANDARDNÝ VSTUP/VÝSTUP ... user

see (+ Stream) .. OTVORÍ SÚBOR, Stream je atom

seen .. UZAVRIE SÚBOR, KT. SLÚŽIL AKO VSTUPNÝ PRÚD, PREPÍNA VSTUP NA STANDARDNÝ VSTUP

seeing (X) .. AKTUÁLNY VST. PRÚD

tell(Subor) .. OTVORÍ SÚBOR PRE VÝSTUP

telling (X)

ČÍTANIE VSTUPU < ZNAK PO ZNAKU / PO TERMOCH

read (Term)

ČÍTA term ZO VSTUPU .. UNIFIKUJE!
read VŽDY SPOTREBUJE 1 VSTUP
USPEJE IBA VTEDY, KEĎ SA MU PODARÍ ZUNIFIKOVAŤ VSTUP S Term-om
read NEBACKTRACKUJE

write (t(a,b,c)), write('.')
PÍŠE BEZ BODIEK

VSTUP ZNAKOV

get0(X) .. ČÍTA ZNAK, UNIFIKUJE JEHO ASCII HODNOTU S X

get(X) .. TO ISTÉ, IGNORUJE NETLAČITEĽNÉ ZNAKY

skip(X) .. PREČÍTA AJPOŇ 1 ZNAK A POKRAČUJE AŽ PO OKAMŽIKU PREČÍTANIA ZNAKU S KÓDOM X.

put(X) .. VYPÍŠE ZNAK X

tab(N) .. N MEDZIER

nl .. NEW LINE

PRÍKLADY (VID SKRIPTÁ)

repeat, ...

VŽDY USPEJE

repeat, ..., fail ≡ NEKONEČNÝ CYKLUS

DEFINÍCIA OPERÁTORU

:- op(600, xfx, ma_rad).

PRIORITA — ČÍM MENŠIA, TÝM VIAC VIAŽE

ZAKLÍNADLO → PREFIX fx, fy
POSTFIX xf, yf
INFIX BIN. xfx, xfy, yfx

X A Y UDÁVAJÚ ASOCIATIVITU

(a+b)+c JE xfy. xfx NIEDOVOLUJE ANI F BOLO 2x

10.11.2006 Fr
PETER ČERNO

PRG005 NEPROC. PROGRAMOVANIE

```
zjedn (V1, V2) :- zoj (V1, 0, V2).
zj (V1+A, N, V2) :- integer(A), !,
    M is N+A, zj (V1, N1, V2).
    ...
```

```
bagof ( Vzor, Ciel, Bag).
```

Bag JE ZOZNAM VŠETKÝCH INŠTANCIÍ termu VZOR, PRE KTORÉ JE CIEĽ SPLNITEĽNÝ

! KEĎ NEEXISTUJE ŽIADNA INŠT., VRÁTI no.

```
setof (Vzor, Ciel, Set)
```

TO ISTÉ AKO bagof
Set JE MNOŽINA

```
bagof ( X, manad(Y, X), L),
Y = petr    L = [vera, eva, hana, miro];
Y = ivan    L = [vodka, eva, hana];
Y = eva     L = [ivan];
no

bagof (r(Y,S), bagof (X, manad(X,Y), S), L ).
```

↳ MOŽE BYŤ IBA V setof, bagof

bagof(X, Y^ manad(Y, X), L).

↓

EXISTENČNÝ KVANTIFIKÁTOR

L = [reva, eva, hana, pivo, vodka, eva, hana]

findall (Vzor, Ciel, Z)

PODOBNE AKO Bag, VŽDY USPEJE, NIE VRÁTIŤ []

VŠETKY PREMENNÉ BY MALI BYŤ VIAZANÉ ^

NEGÁCIA

not(P)
not P
\+

\+ P :- P, !, fail
\+P.

24.11.2006 Fr
PETER ČERNO

PRG005 NEPROC. PROGRAMOVANIE

IVAN KOLAŇ INÉ PROGRAMOVANIE
STRETNUTIE S JAZYKOM LISP
ALFA 1990

SPÔSOB VYHODNOCOVANIA VÝRAZU

NORMAL ORDER = EXPANDUJ A REDUKUJ
APPLICATIVE ORDER = VYHODNOŤ NAJSKÔR ARGUMENTY

```
(define (abs x)
  (cond ((> x 0) x)
        ((= x 0) 0)
        ((< x 0)(-x))))
```

nil JE VŽDY ~~FALSE~~ false , = PRÁZDNY ZOZNAM
OSTATNÉ JE true

```
(define (abs x)
  (cond ((x < 0) (-x))
        (else x)  ))
```

STUDIJNÁ: JAK SE JMENUJE ~~PATO~~ TAHLE KNÍŽKA

08.12.2006 Fr
PETER ČERNO

PRG005 NEPROC. PROG.

ŠPECIÁLNE TYPY PATTERNOV

PREMENNÁ (FORM. PARAMETER MATCHUJE VŽDY)

ŽOLÍK _ (MATCHUJE VŽDY)

DÁTOVÝ KONŠTRUKTOR ← MATCHUJE ZLYHÁVA TO (NAPR. ZOZNAM DVOJíC)

ARITMETICKÝ ARGUMENT $f\ n+1 = f\ n * n$

AS-PATTERNS

$$f\ (x:xs) = x:x:xs$$

$$f\ @\ (x:xs) = x:s$$

LAZY PATTERNS

(\ ~(x,y) -> 0) bot ⇒ 0

(\ (x,y) -> 0) bot ⇒ ⊥

(\ ~[x] -> 0) [] ⇒ 0

(\ ~[x] -> x) [] ⇒ ⊥

(\ ~[x, ~(a,b)] -> x) [0,bot] ⇒ 0

(\ ~[x, (a,b)] -> x) [0,bot] ⇒ ⊥

(\ (x:xs) -> x:x:xs) bot ⇒ ⊥

(\ ~(x:xs) -> x:x:xs) bot ⇒ ⊥:⊥:⊥

```
let a = x + y + z in a*a
```

POSTA $(x+y+z) * (x+y+z)$

```
let f i j k = i + j - k
in  f a b c * f x y d
```

$(a+b-c) * (x+y-d)$

INDENTACE

```
let y = a * b
    f x = (x+y) / y
in  f c + f d
```

ZNAMÉNKA

```
let { y = a * b
    ; f x = (x+y)/y
    }
in f c + f d
```

```
let y = a*b ; z = a/b
    f x = (x+y)/z
in  f c + f d
```

08.12.2006 Fr
PETER ČERNO

PRG005 NEPROC. PROG.

LET EXPR.:

```
let { e1 ; ... ; en }
in  e
```

```
f a + f b   where { a = 5; b = 4;
                    f x = x + 1
```

```
f a + f b   where  a = 5; b = 4
                   f x = x + 1
```

```
sumsquares :: Int -> Int -> Int

sumsquares n m
    = sqN + sqM
    where
      sqN = n*n
      sqM = m*m
```

```
maxsq x y
  | sqx > sqy = sqx
  | otherwise = sqy
      where
        sqx = sq x
        sqy = sq y
        sq :: Int -> Int
        sq z = z*z
```

CONDITIONAL:

if e_1 then e_2 else e_3

$\equiv$ case e_1 of True -> e_2
False -> e_3

```
map :: (a -> a) -> [a] -> [a]
map _ [] = []
map f (a:x) = f a : map f x

map f list = [f a | a <- list]

sumlist :: [Int] -> Int
sumlist [] = 0
sumlist (x:s) = x + sumlist s

fold :: (a -> a -> a) -> [a] -> a
fold f [a] = a
fold f (a:b:x) = f a (fold f (b:x))

sumlist l = fold (+) l
maxlist l = fold max l
```

08.12.2006 Fr
PETER ČERNO

PRG 005 NEPROG. PROG.

```
bag :: Int -> [a] -> [a]
bag 0 _ = []
beg n (x:s) = x : beg (n-1) s

foldr :: (t -> t -> t) -> t -> [t] -> t
foldr f s [] = s
foldr f s (a:r) = f a (foldr f s r)

concat :: [[t]] -> [t]
concat xs = foldr (++) [] xs
```

concat [[1,2,3], [4,5], [], [1,2]] = [1,2,3,4,5,1,2]

NEKONEČNÉ VÝRAZY

```
ones = 1 : ones

nums_from n = n : nums_from (n+1)
squares = [ x ** 2 | x ← nums_from 0 ]
```

[1..10] ~ [1,2,3,4,5,6,7,8,9,10]
[1,3..10] ~ [1,3,5,7,9]
[1..] ~ IN
[1,3..] ~ NEPÁRNE IN

```
zip (x:xs) (y:ys)  =  (x,y) : zip xs ys
zip  xs    ys      =  []
```

```
fib = 1 : 1 : [ a + b | (a, b) <- zip fib (tail fib) ]
```

```
pythag = [ (a, b, c) |  c <- [2..], b <- [2..c-1],
                        a <- [2..b-1], a*a + b*b == c*c ]
```

```
primes :: [Int]
primes = sieve [2..]
sieve (a:x) = a : sieve [y | y <- x, y `mod` a > 0]
```

```
class Num a where
  (+)      :: a -> a -> a
  negate   :: a -> a

(Num a) => a -> a -> a

instance Num Int where
  x + y    =  addInt x y
  negate x =  negateInt x

instance Num Float where
  x + y    =  addFloat x y
  negate x =  negateFloat x
```

15.12.2006 Fr
PETER ČERNO

PRG005 NEPROC. PROG.

```
[ f x | x <- xs ]
```

GENERÁTOR

= ZOZNAM VŠETKÝCH TERMOV fx, KDE x∈xs

```
[ (x,y) | x <- xs, y <- ys ]
```

= KARTÉZSKY SÚČIN

```
[ (x,y) | x <- [1,2], y <- [3,4] ] =
    = [ (1,3), (1,4), (2,3), (2,4) ]
```

```
g :: a -> [a] -> [a]
g x y = [ succ z | z <- y, z < x ]
```

STRÁŽ

```
g 9 [2,4,7,8,10,11] = [3,5,8,9]
```

```
[ a+b | a <- [1,2], isEven a, b <- [a .. 2*a] ]
```
⇒ [4,5,6]

```
[ (a,b) | a <- [1..3], b <- [1..a] ]
```
⇒ [(1,1), (2,1), (2,2), (3,1), (3,2), (3,3)]

```
perm :: [a] -> [[a]]

perm []  = [[]]
perm x   = [a:p | a<- x, p <- perm (x \\ [a])]

perm []  = [[]]
perm (a:x) = [ p ++ [a] ++ q |
      r <- perm x ,  (p, q) <- split r ]

split :: [t] -> [([t], [t])]
split []  = [([],[])]
split (a:x) = ([], a:x) :
    [(a:p, q) | (p,q) <- split x ]

(\\) : [a] -> [a] -> [a]
x \\ [] = x
x \\ (a:w) = vyhod x a \\ w

vyhod [] _ = []
vyhod a:x b
   | a == b  = vyhod x b
   | a /= b  = a: vyhod x b
```

~~vyhod a:x a~~ NEZMYSEL V HASKELLU

15.12.2006 Fr
PETER ČERNO

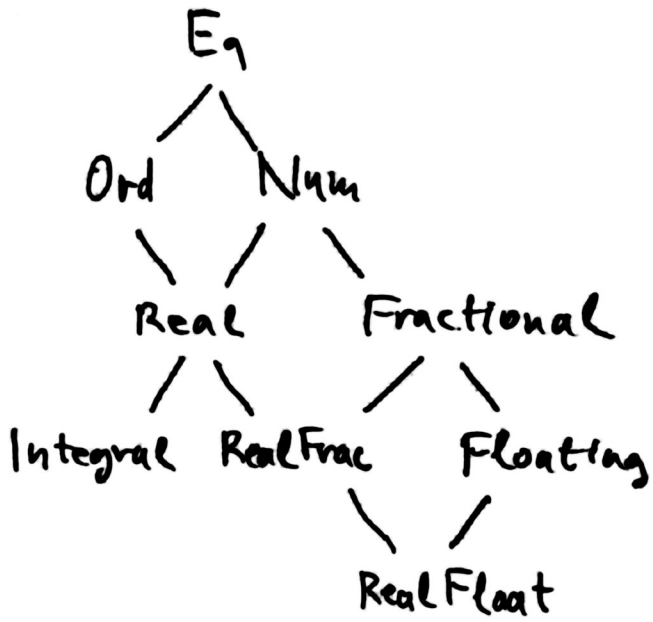
PRG 005 NERROC. PROG.

The Evolution of a Haskell Programmer

```
member x []     = False
member x (y:ys) = x==y || member x ys

:t member
(Eq a) => a -> [a] -> Bool
```

TYPE CLASSES

```
class Num a where
  (+) :: a -> a -> a
  negate :: a -> a

instance Num Int where
  x + y = addInt x y
  negate x = negateInt x

double :: (Num a) => a -> a
double x = x+x

doubles :: (Num a, Num b) => (a,b) -> (a,b)
doubles (x,y) = (double x, double y)
```



DERIVED INSTANCES

```
data Bool = False | True  deriving (Eq, Ord)
```

POLE VO FUNKCIONÁLNYCH JAZYKOCH

INKREMENTÁLNE
- KONŠTRUKTOR PRÁZDNEHO POĽA
- update op pole, index, hodnota → pole

MONOLITICKÉ
- POLE AKO CELOK

```
fib n = array (0, n)
  ([0:=1, 1:=1] ++ [i:= a!(i-2)+a!(i-1) | i ← [2..n]])
```

04.01.2007 Th
PETER ČERNO

PRLOOS NEPROC PROG.

SKÚŠKA:

3HOD. 4-5 PRÍKLADOV

VYŽADUJE SA NEPROCEDURÁLNY PRÍSTUP

ÚSTNA ČASŤ - O PRÍKLADOCH (V EXTR. PAÍA. NEBUDE)

EFEKTIVITA V PROLOGU:

VEĽKÉ NÁROKY NA PAMÄŤ, SEKUNDÁRNE NA ČAS

PROSTRIEDKY NA ZLEPŠENIE EFEKTIVITY:

AKUMULÁTOR

NEÚPLNE DEF. DÁT. ŠTRUKTÚRY

REZ

DÁTA V DATABÁZE /AKO PARAM.

GRAF, KT. JE VEĽKÝ, JE LEPŠIE DÁŤ V DATABÁZE

12.01.2007 Fr
PETER ČERNO

PRG005 NEPROC. PROG.

```
deli(X,Z) :- deli(X,Y), deli(Y,Z).
deli(3,12).
deli(12,48).

?- deli(3,X).
   X = 12 ;
   X = 48 ;
   No
```

KLAUZULA

$$U_1 \vee U_2 \vee \ldots \vee U_n$$

↓ LITERÁL

$$A :- B_1, \ldots, B_n$$

$$B_1 \wedge \ldots \wedge B_n \Rightarrow A$$

$$A \Rightarrow B. \equiv \neg A \vee B$$

$$\neg B_1 \vee \neg B_2 \vee \ldots \vee \neg B_n \vee A$$

= HORNOVA KLAUZULE .. MAJÍ NEJVÍCE 1 ~~LITERÁL~~ POZITIVNÍ LITERÁL

$$\frac{A \quad A \Rightarrow B}{B}$$ MODUS PONENS

$$\frac{A \quad \neg A \vee B}{B}$$

PRAVIDLO REZU

$$\frac{\Delta \vee A \quad \Gamma \vee \neg A}{\Delta \vee \Gamma}$$

LOGICKY PRAVDIVÁ FORMULA. PRÍKLAD:

$$\neg \exists x\, A(x) \Leftrightarrow \forall x\, \neg A(x)$$

INTERPRETÁCIA NEJAKEJ FORMULE

TEÓRIA:

ZOBERIEM SI NEP. UNIVERZUM U

$A(\,)$ .. PODMNOŽINA

LOGICKÁ FORMULA JE LOGICKY PRAVDIVÁ FORMULA
$\Leftrightarrow$ PLATÍ VO VŠETKÝCH INTERPRETÁCIACH

LOG. FORMULA JE LOGICKY SPLNITEĽNÁ
$\Leftrightarrow$ $\exists$ INTERPRETÁCIA, V KT. JE PRAVDIVÁ

LOG. FORMULA JE PRAVDIVÁ $\Leftrightarrow$ JEJ NEGÁCIA
NIE JE SPLNITEĽNÁ

FORMULA V <u>KLAUZULÁRNEJ FORME</u>

$\forall x_1 \dots \forall x_n : C_1 \wedge \dots \wedge C_n$

C_i JE KLAUZULA, T.J. DISJUNKCIA LITERÁLOV

$x_1, \dots, x_n$ SÚ PREMENNÉ

12.01.2007 Fr
PETER ČERNO

PRG005 NEPROC. PROG.

SKOLEMOVA VETA:

KU KAŽDEJ FORMULE D EXISTUJE FORMULA D'
V KLAUZULÁRNEJ FORME:

D JE SPLNITEĽNÁ ⇔ D' JE SPLNITEĽNÁ

DÔKAZ: A) SPRAVÍME EXISTENČNÝ UZÁVER

1) VYLÚČIME ZBYTOČNÉ KVANTIFIKÁTORY

$\forall x \exists y \forall z \; x = z$

2) PREMENUJEME VIAZ. PREM.

$\forall x A(x) \wedge \exists x B(x)$

$\forall x A(x) \wedge \exists y B(y)$

3) ELIM. ⇔

$(A \Leftrightarrow B) \equiv (A \Rightarrow B) \wedge (B \Rightarrow A)$

4) NEGÁCIE DOVNÚTRA

$\neg(\forall x B(x)) \equiv \exists x (\neg B(x))$

5) KVANTORY DOPRAVA

$\exists x (A \vee B) \rightarrow A \vee \exists x B$ × NIE JE V A

6) ELIM. EX. KVANT. (SKOLEMIZÁCIA)

$\exists y B(y)$ ↺ VOĽNÉ V..

$B(f(x_1, \ldots, x_n))$

7) UNIVERZÁLNY KVANT. DOĽAVA

8) DISTRIBUTÍVNE ZÁK.

9) ZJEDNODUŠENIE

11.11.2006 Sa
PETER ČERNO

PRG005 NEPROCED. PROG. - ÚLOHA

```
variations(0, Ls, []):- !.
variations(K, Ls, [Z|Zs]) :-
    K1 is K-1,
    select(Z, Ls, Ks),
    variations(K1, Ks, Zs).

combinations(K, As, Bs) :-
    length(As, N),
    combinations(K, N, As, Bs).
combinations(0, N, As, []):- !.
combinations(K, N, As, _ ):- K>N, !, fail.
combinations(K, N, [A|As], [A|Bs]):-
    K1 is K-1, N1 is N-1,
    combinations(K1, N1, As, Bs).
combinations(K, N, [A|As], Bs ):-
    N1 is N-1,
    combinations(K, N1, As, Bs).
```

(VARIÁCIE / KOMBINÁCIE) S OPAKOVANÍM
→ STAČÍ NEMINIMALIZOVAŤ

19.11.2006 S4
PETER ČERNO

PRG007 NEPROC. PROGRAMOVANIE - ÚLOHA

```
prvystlpec([],[],[]).
prvystlpec([[]|_],[],[]):-!.
prvystlpec([[R|Rs]|M1s],[R|Cs],[Rs,M2s]):-
  prvystlpec(M1s,Cs,M2s).

skalarnysucin([],[],0).
skalarnysucin([H1|C1s],[H2|C2s],Scalar):-
  H3 is H1 * H2,
  skalarnysucin(C1s,C2s,H4),
  Scalar is H3 + H4.

vektormatica(_,[],[]).
vektormatica(_,[[]|_],[]):-!.
vektormatica(R1,M,[H|T]):-
  prvystlpec(M,C1,M2),
  skalarnysucin(R1,C1,H),
  vektormatica(R1,M2,T).

maticamatica([],_,[]).
maticamatica([[]|_],_,[]):-!.
maticamatica([R1|M1],M2,[R3|M3]):-
  vektormatica(R1,M2,R3),
  maticamatica(M1,M2,M3).

maticamaticaT([],_,[]).
maticamaticaT([[]|_],_,[]):-!.
maticamaticaT([R1|M1],M2,[R3|M3]):-
  vektormaticaT(R1,M2,R3),
  maticamaticaT(M1,M2,M3).
```

```
vektormaticaT(_,[],[]).
vektormaticaT(_,[[]|_],[]):-!.
vektormaticaT(R1,[R2|M2],[H|T]):-
  skalarnysucin(R1,R2,H),
  vektormaticaT(R1,M2,T).
```

```
transponuj([],[]).
transponuj([[]|_],[]):-!.
transponuj(M1,[R3|M3]):-
  prvystlpec(M1,R3,M2),
  transponuj(M2,M3).
```