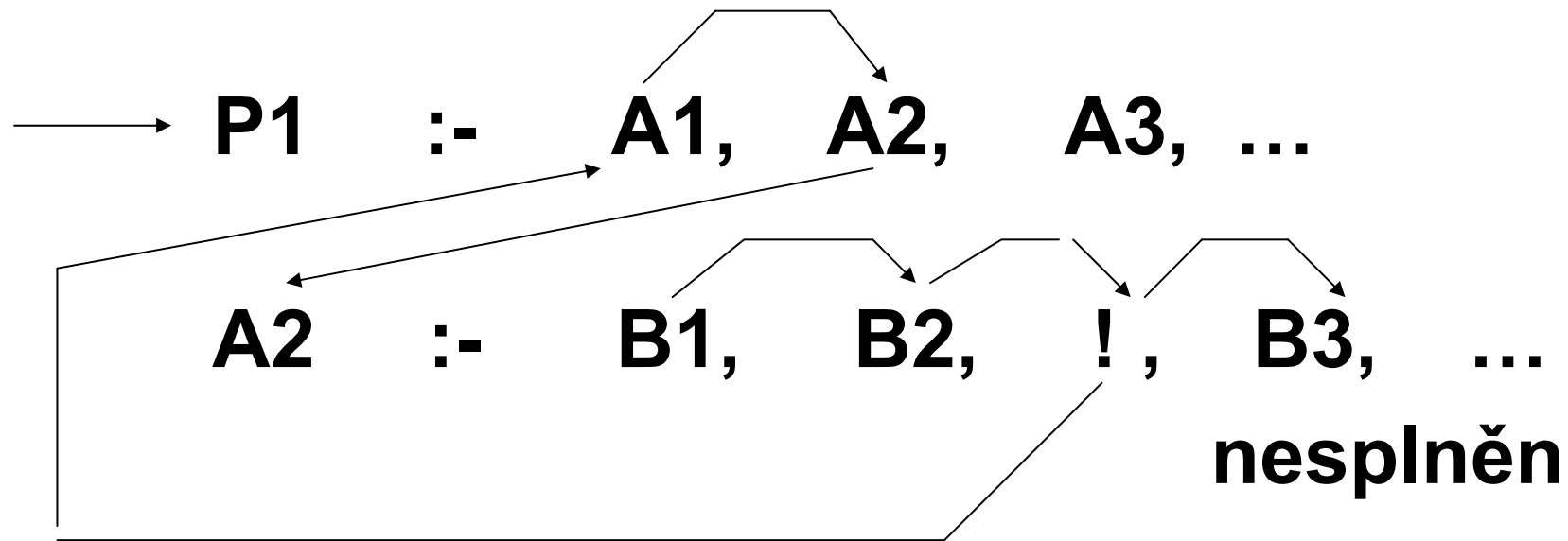


Logické programování pokračování

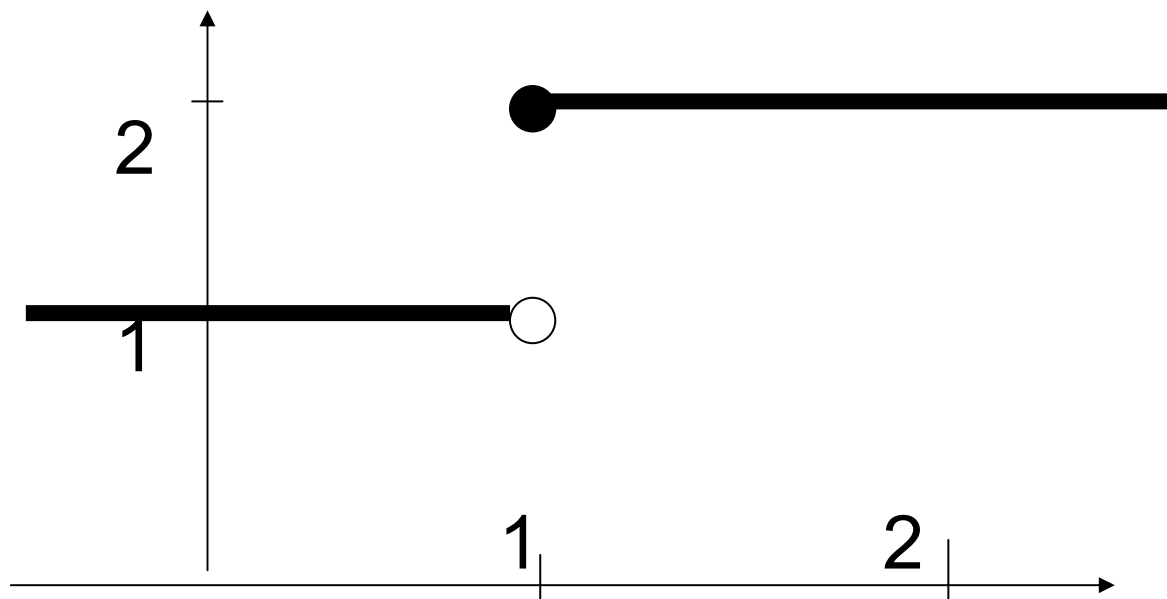
Ovlivnění mechanismu návratu

Predikát řezu !



- **Zelený řez** znemožní návrat, který by stejně skončil neúspěchem (nemění deklarativní smysl)

Př. $f(X, Y) :- X < 1, !, Y = 1.$ %je zelený
 $f(X, 2) :- X \geq 1.$



- **Červený řez** znemožní návrat, který by našel jiné řešení (mění deklarativní smysl)

`f(X, Y) :- X < 1, !, Y = 1. %je červený`

`f(X, 2) .`

protože

`f(X, Y) :- X < 1, Y = 1.`

`f(X, 2) . %Produkuje alternativní nesprávné řešení`

`?-f(0,Y).`

`Y=1;`

`Y=2;`

`no`

Predikát řezu

- Použijeme, když chceme zabránit hledání jiné alternativy
- Odřízne další provádění cílů z hlavy pravidla
- Je bezprostředně splnitelným cílem, který nelze opětovně při návratu splnit
- Projeví se pouze, když má přes něj dojít k návratu
- Změní mechanismus návratu tím, že znepřístupní ukazatele vlevo od něj ležících cílů (přesune je na konec Db)

Př. použití řezu

fakt(N,1) :-N=0,!.

fakt(N,F) :- M is N-1,
 fakt(M,G),
 F is G * N.

?-fakt(1,X).

X=1;

no

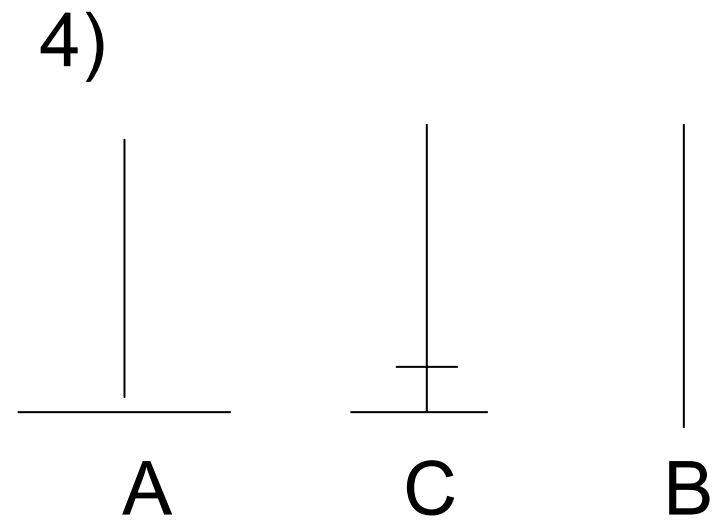
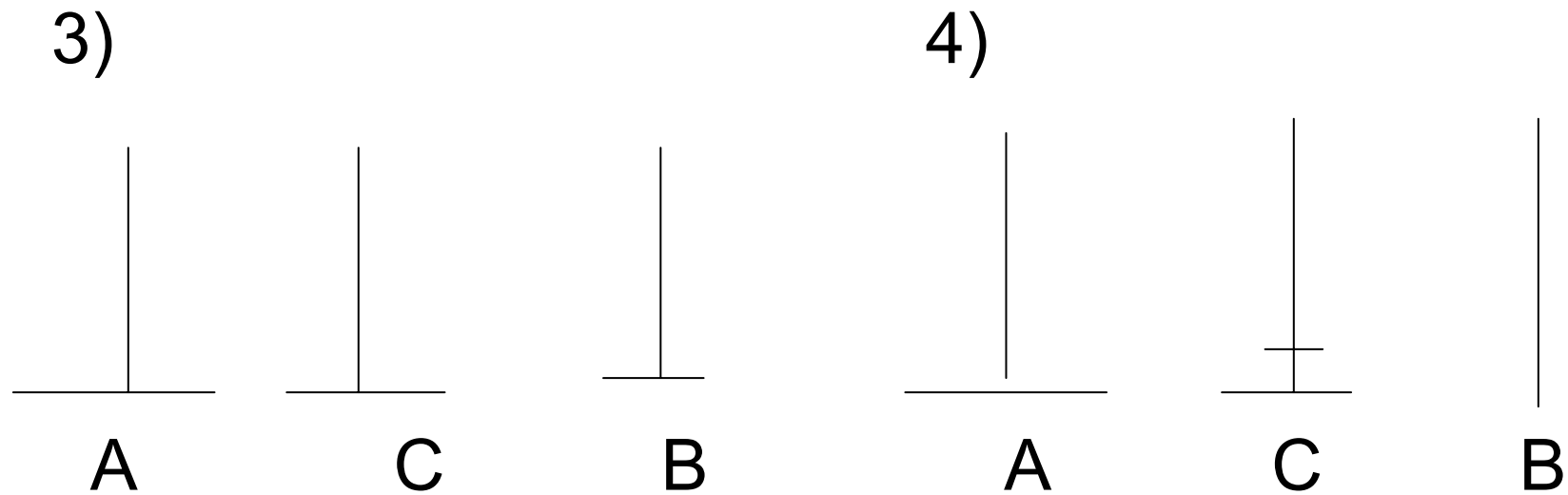
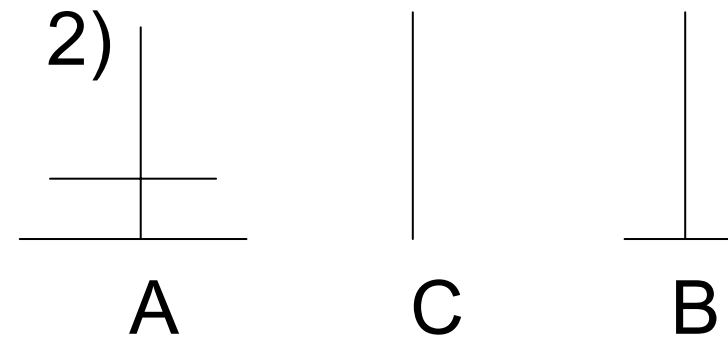
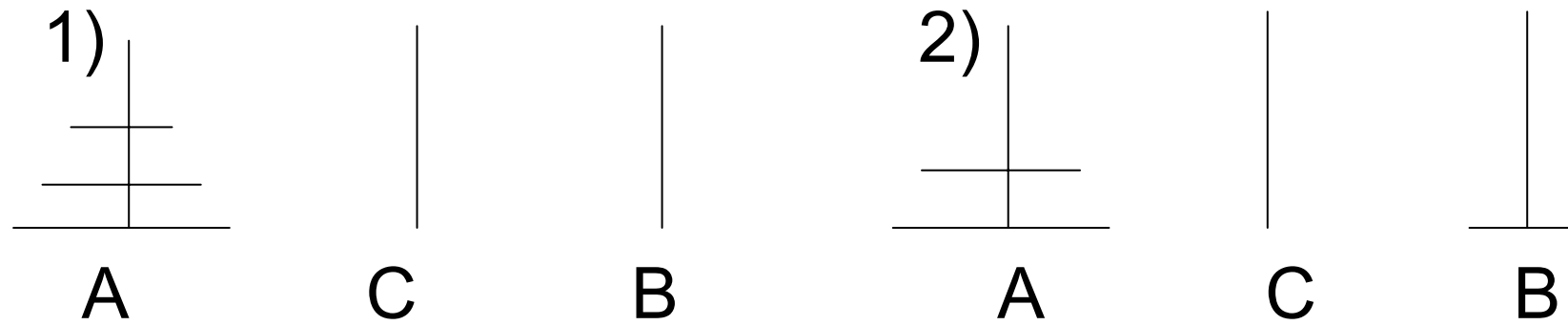
Př.Hanoiské věže

Jsou dány 3 trny A, B, C.

Na A je navléknuto N kotoučů s různými průměry tak, že vytváří tvar pagody (menší je nad větším).

Přemístěte kotouče z A na C s využitím B jako pomocného trnu tak, aby stále platilo, že nikdy není položen větší kotouč na menší

Př.Hanoiské věže



Př.Hanoiské věže (Hanoi.pro)

```
hanoi(N) :- presun(N,levy,stred,pravy), !.  
presun(0,_,_,_) :- !.  
presun(N,A,B,C) :-    %presun z A na B za pouziti  
                      M is N-1,%pomocneho C  
                      presun(M,A,C,B),  
                      inform(A,B),  
                      presun(M,C,B,A).  
inform(A,B) :- write([presun,disk,z,A,na,B]),  
               nl.
```

Standardní predikáty jazyka Prolog

- I/O operace
- Řídící predikáty a testy
- Predikáty pro práci s aritmetickými výrazy
- Predikáty pro manipulaci s databází
- Predikáty pro práci se strukturami

I/O operace

write(X) zápis termu do výstupního proudu

nl odřádkování

tab(X) výstup X mezer

display(X) výstup jako write, ale ve funkční notaci

read(X) čtení termu ze vstupního proudu

put(X) výstup znaku na jehož ASCII kod je X
instalováno

get(X) vstup ASCII kódu zobrazitelného znaku
(32 a výše)

get0(X) jako get, ale i pro nezobrazitelné znaky

Př.

?-write(a+b*c), nl, display(a+b*c).

a+b*c

+(a,*(b,c))

yes

?- consult(user). *zahájí vstup z klavesnice*

| write-retezec([]).

| write-retezec([H|T]):-put(H),
write-retezec(T).

| quit. *vstup z klavesnice ukončí*

?- write-retezec([65,66,67]).

ABC

yes

?-

I/O operace

- skip(X)** přeskakuje vstupující znaky, dokud úspěšně neporovná znak s X
- tell(X)** přepnutí výstupu do souboru X
- told** uzavření souboru a přepnutí výstupu na user
- see(X)** přepnutí vstupu na soubor X
- seen** současný vstupní soubor je uzavřen a vstup přepnut na user
- setdrive(X)** nastavení driveru X
- getdrive(X)** zjištění jména aktuálního driveru
- chdir(X)** nastavení pracovního adresáře

Př. Zápis do souboru a čtení ze souboru

```
?-setdrive(d), tell(vystupni),  
    write(hokuspokus), write('.'), told.
```

yes

```
?- see(vystupni), read(X), seen.
```

X = hokuspokus

yes

```
?-chdir("\\tmp04\\Kuchar",Chyba).
```

Chyba = 0 znamená bezchybné provedení

Řídící predikáty a testy

true vždy splněný cíl

fail vždy nesplněný cíl

var(X) splněno, je-li X volnou proměnnou

nonvar(X) splněno, neplatí-li var(X)

atom(X) splněno, je-li X instalováno na atom

integer(X) splněno, je-li X instalováno na integer

atomic(X) splněno, je-li X instalováno na atom
nebo integer

not(X) X musí být interpretovatelné jako cíl.
Uspěje, pokud X není splněn

call(X) X musí být interpretovatelné jako cíl.
Uspěje, pokud X je splněn

Řídící predikáty a testy

halt	ukončí výpočet
$X = Y$	pokus o porovnání X s Y
$X \neq Y$	opak =
$X == Y$	striktní rovnost
$X \neq Y$	úspěšně splněn, neplatí-li $==$
!	změna mechanismu návratu
repeat	nekonečněkrát splnitelný cíl
X, Y	konjunkce cílů
$X ; Y$	disjunkce cílů

Predikáty pro práci s aritmetickými výrazy

X is E E musí být aritm. výraz, který se vyhodnotí a porovná s X

E1 + E2 při instalovaných argumentech (pod. −, *, /, mod)

E1 > E2 při instalovaných argumentech (pod. >=, <, <=, \=, =)

E1 ::= E2 uspěje, jsou-li si hodnoty E1, E2 rovny

E1 != E2 uspěje, nejsou-li si hodnoty E1, E2 rovny

Predikáty k manipulaci s databází a klauzulemi

listing(X) výpis všech klauzulí na jejichž jméno je X instalováno

listing výpis celého programu

clause(X, Y) porovnání X a Y s hlavou a s tělem klauzule

asserta(X) přidání klauzule instalované na X na začátek databáze

assertz(X) totéž, ale přidává se na konec databáze

retract(X) odstranění prvního výskytu klauzule X z databáze

findall(X,Y,Z) všechny výskyty termu X v databázi, které splňují cíl Y jsou vloženy do seznamu Z

Predikáty pro práci se strukturami

functor(T, F, A) -vytvoří strukturu T s funktorem F a aritou A, nebo rozdělí strukturu T na její funktor a aritu

arg(N, T, A) porovná A s N-tým argumentem struktury T

name(A, L) je-li A instalováno, převede jméno atomu A na seznam znaků a ten porovná s L. Není-li A instalováno, instaluje je znaky podle seznamu L

length(L, A) zjistí délku seznamu a porovná ji s A

X =.. L tzv."univ". Provádí porovnání termu X se seznamem L, který je složen z funktoru termu X, následovaném argumenty X

op(Prec, Asociat{xfx,xfy,yfx,yfy,fx,fy,xf,yf}, Op)

definuje operator Op s precedencí Prec a asociativitou A. Op může být
seznam operatorů stejné precedence a asociativity

y argument může obsahovat operátor stejné nebo nižší priority

x " musí " " nižší priority

f určuje pozici operátoru

op(1200, xfx, [:-, -->])

op(1200, fx, [?- , :-])

op(1100, xfy, ';') pravoasociativní

op(1000, xfy, ', ')

op(900, fy, not)

op(700, xfx, [=, \=, is, =.., ==, \==, :=, =\=, <, >, =<, >=]) neasociativn

op(500, yfx, [+ , -]) levoasociativní

op(500, fx, [+ , -])

op(400, yfx, [/ , *])

op(300, xfx, mod)

op(200, xfy, ^)

Paralelismus ve výpočtu logického programu

- AND paralelismus
cíl1, cíl2, ..., cíln v deklarativní sémantice
- OR paralelismus
cíl1, cíl2, ..., cílm „ „
- Unifikační paralelismus
souběžné porovnání odpovídajících si
komponent složených termů

Př. Quicksort.pro

```
qsort([],[]).  
qsort([A],[A]).  
qsort([A|X],Y):-pod(A,X,POD),  
    qsort(POD,S1),  
    nad(A,X,NAD),  
    qsort(NAD,S2),  
    append(S1,[A|S2],Y).
```

Př. hledání cesty grafem – Path.pro

vertex(1, 4).

vertex(1, 5).

vertex(2, 1).

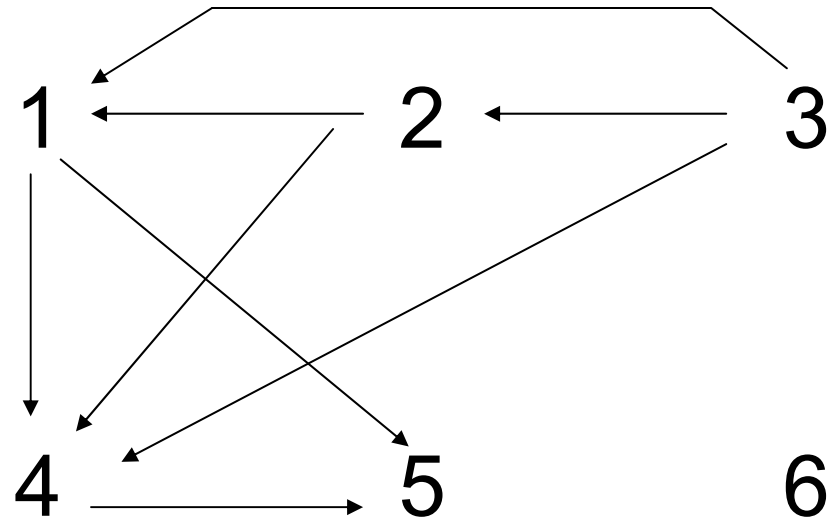
vertex(2, 4).

vertex(3, 1).

vertex(3, 2).

vertex(3, 4).

vertex(4, 5).



Př. hledání cesty grafem – Path.pro

```
path:-read(X),read(Y),  
      go(X,Y,[X]).
```

```
go(X,X,T):-write(T).  
go(X,Y,T):-vertex(X,Z),  
             go(Z,Y,[Z|T]).
```

?- path.

3. 4.

[4,1,3]

yes

?-

Př. hledání cesty grafem – Path.pro

```
lookforall(X,Y):-go(X,Y,[X]),  
    fail.
```

```
lookforall(_,_).
```

```
?-lookforall(3,1).
```

```
[1,3][1,2,3]
```

```
yes
```

```
?-
```

Př. hledání cesty grafem – Path.pro

Při vyhodnocení nejdivšechny způsobí cyklus v grafu nekonečný výpočet.. Jak napravit?

```
go(X,X,T):-write(T).  
go(X,Y,T):-vertex(X,Z),  
               not(member(Z,T)),  
               go(Z,Y,[Z|T]).
```

Závody gymnastek –závod.pro

Urcete jmena vitezek disciplin z techto informaci:

- 1) Dvorakova ani Sobotkova nevyhraly preskok ani bradla.
- 2) V preskoku nezvitezila Vera ani Ludmila.
- 3) Sobotkova se nejmenuje ani Vera ani Jirina a kamaradi
- 4) s Beckovou
- 5) Junkova neni ani Monika ani Jirina
- 6) Vera nevyhrala na bradlech, Monika nevyhrala v prostnych.
- 7) Jednou z disciplin byla kladina.
- 8) V kazde ze ctyr disciplin zvitezila jina zavodnice.

Závody gymnastek –závod.pro

```
vitpres(X,Y):-pojmen(X,Y),          %pozitivní cíl = důležité
    not((Y=sobotkova; Y=dvorakova)), %1
    not((X=vera; X=ludmila)).        %2

vitbrad(X,Y):-pojmen(X,Y),
    not((Y=sobotkova; Y=dvorakova)), %1
    not(X=vera).                    %6

vitpros(X,Y):-pojmen(X,Y),
    not(X=monika).                  %6

vitklad(X,Y):-pojmen(X,Y).          %7
```

Závody gymnastek –závod.pro

pojmen(X,junkova):-jmeno(X),

X\=monika, X\=jirina.

%5

pojmen(X,sobotkova):-jmeno(X),

X\==vera, X\==jirina.

%3

pojmen(X,beckova):- jmeno(X).

%4

pojmen(X,dvorakova):- jmeno(X).

jmeno(monika).

jmeno(jirina).

jmeno(vera).

jmeno(ludmila).

ruzne(A1,A2,A3,A4):-

A1\=A2,A1\=A3,A1\=A4,A2\=A3,A2\=A4,A3\=A4.

%8

} jsou
taková
jména

Závody gymnastek –závod.pro

vitez(X1,Y1,X2,Y2,X3,Y3,X4,Y4):-

vitpres(X1,Y1), vitbrad(X2,Y2),

vitpros(X3,Y3), vitklad(X4,Y4),

ruzne(X1,X2,X3,X4), ruzne(Y1,Y2,Y3,Y4). %8

go:- vitez(JPR,PPR,JBR,PBR,JPROS,
PPROS,JKLAD,PKLAD),
write(JPR),write(PPR),write(JBR),write(PBR),nl,
write(JPROS),write(PPROS),
write(JKLAD),write(PKLAD).

Závody gymnastek (permutacemi) –závod1.pro

vypiseme seznam jmen a seznam prijmeni vitezů
soutěže v poradi: vitezka Bradel, vitezka Kladiny,
vitezka Preskoku, vitezka Prostnych

```
perm([],[]).
```

```
perm(L,[X|P]) :- del(X,L,L1), perm(L1,P).
```

```
del(X,[X|T],T).
```

```
del(X,[Y|T],[Y|T1]) :- del(X,T,T1).
```

```
%hledá poradové číslo N, prvku E, v zadaném seznamu
```

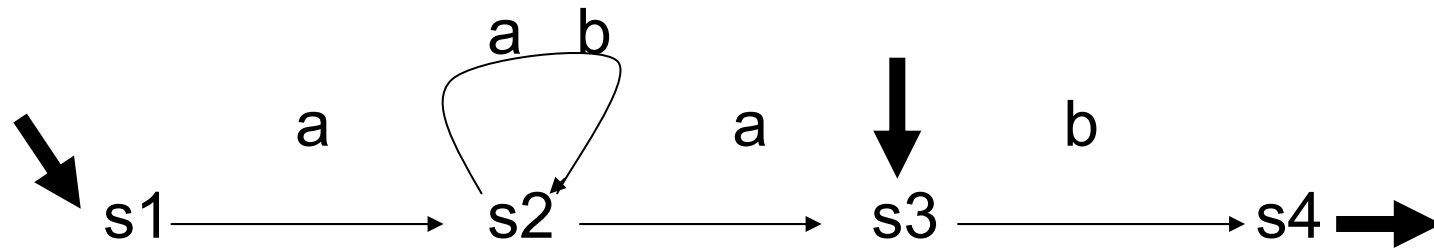
```
poradi(E,[E|T],1) :-!.
```

```
poradi(E,[X|T],N) :- poradi(E,T,M), N is M+1.
```

Závody gymnastek (permutacemi) –závod1.pro

```
%permutuj Jmena a Prijmeni
vitez :- perm([j,l,m,v],J), perm([b,d,j,s],P),
    J=[J1,J2,J3,J4], P=[P1,P2,P3,P4],
% sob se nejmenuje ver    sob se nejmenuje jir
    poradi(s,P,N1),poradi(v,J,N2), N1\=N2, poradi(j,J,N3),
    N1\=N3,
%jun neni mon    jun neni jir
    poradi(j,P,N4),poradi(m,J,N5), N4\=N5, N4\=N3,
%bradla nevyhrala vera ani dvorak. ani sobotk
    J1\=v, P1\=d, P1\=s,
%preskok nevyhrala vera ani lud. ani dvorak, ani sobotk
    J3\=v, J3\=l, P3\=d, P3\=s,
%prostna nevyhrala monika
    J4\=m,
    write(J),write(P).
```


Nedeterministický automat



koncovy(s4).

prechod(s1, a, s2).

prechod(s2, a, s2).

prechod(s2, b, s2).

prechod(s2, a, s3).

prechod(s3, b, s4).

Nedeterministický automat

akceptuje(S, []) :- koncovy(S).

akceptuje(S, [H|Z] :- prechod(S, H, S1),
akceptuje(S1, Z).

go(PocStavy, Vstup) :- member(S, PocStavy),
akceptuje(S, Vstup).

Akceptuje slovo abab ?

?-go([s1, s3], [a,b,a,b]).

Akceptuje z s1 třípísmenové slovo ?

?- akceptuje(s1,[X,Y,Z]).

Z jakého stavu lze akceptovat slovo ab ?

?-akceptuje(S, [a,b]).

Skákání koně

Heuristika – jdi nejdříve do špatně přístupných míst

1. Zjisti místa, kam lze táhnout
2. Bylo-li provedeno 63 tahů, vypiš cestu a skonči
3. Nelze-li nikam táhnout vrať se a zkus další alternativu
4. Zjisti, kolik alternativ tahů je z každého z těchto míst
5. Jdi do místa s nejmenším počtem dalších alternativ
6. Opakuj od 1. bodu

Cestou bude seznam tahů tvaru:

[x_1/y_1 , x_2/y_2 , ... , x_{63}/y_{63}]

Skákání koně - – skokkone.pro

go(X,Y) :-skoky([X/Y],63).

skoky([_/_|Z],0) :-write(Z).

skoky([X/Y|Z],J) :-
 findall(X1/Y1,gen(X1,Y1,X,Y,Z),L),
 min(L,XM,YM,Z),
 [X/Y|Z]=Z1,
 I is J-1,
 skoky([XM/YM|Z1],I).

Skákání koně - – skokkone.pro

gen(X1,Y1,X,Y,Z) :-

```
((X1 is X+2,Y1 is Y+1,X1=<8,Y1=<8);  
(X1 is X+2,Y1 is Y-1,X1=<8,Y1>0);  
(X1 is X-2,Y1 is Y+1,X1>0,Y1=<8);  
(X1 is X-2,Y1 is Y-1,X1>0,Y1>0);  
(X1 is X+1,Y1 is Y+2,X1=<8,Y1=<8);  
(X1 is X-1,Y1 is Y+2,X1>0,Y1=<8);  
(X1 is X+1,Y1 is Y-2,X1=<8,Y1>0);  
(X1 is X-1,Y1 is Y-2,X1>0,Y1>0)),  
not(member(X1/Y1,Z)).
```

Skákání koně - – skokkone.pro

```
min(L,XM,YM,Z)    :- member(X/Y,L),  
                    findall(X1/Y1,gen(X1,Y1,X,Y,Z),L1),  
                    length(L1,N),  
                    asserta(uloz(X/Y/N)),  
                    fail.
```

```
min(L,XM,YM,Z) :- sdruz([],M),!,  
                  qsort(M,M1),  
                  member(XM/YM/_,M1).
```

```
sdruz(U,V)    :- dalsi(X),!,sdruz([X|U],V).  
sdruz(U,U).
```

```
dalsi(X/Y/N) :- retract(uloz(X/Y/N)),!.
```

Skákání koně - – Qsortspe.pro

```
qsort([],[]).  
qsort([A],[A]).  
qsort([A|X],Y):-pod(A,X,POD),qsort(POD,S1),  
                nad(A,X,NAD),qsort(NAD,S2),  
                append(S1,[A|S2],Y).
```

```
pod(A,[],[]).  
pod(A,[B|X],[B|Y]):- B=_/_/B1, A=_/_/A1,B1=<A1, pod(A,X,Y).  
pod(A,[B|X],Y):- B=_/_/B1, A=_/_/A1, B1>A1, pod(A,X,Y).
```

```
nad(A,[],[]).  
nad(A,[B|X],[B|Y]):- B=_/_/B1, A=_/_/A1, A1<B1, nad(A,X,Y).  
nad(A,[B|X],Y):- B=_/_/B1, A=_/_/A1, A1>=B1, nad(A,X,Y).
```

```
append([],X,X).  
append([A|B],X,[A|C]):-append(B,X,C).
```

Symbolické derivování – deriv.pro

Derivace x podle x je 1

$d(X,X,1).$

$d(T,X,0) \text{ :- atom}(T) ; \text{number}(T).$

$d(U+V,X,DU+DV) \text{ :- } d(U,X,DU), d(V,X,DV).$

$d(U-V,X,DU+ (-DV)) \text{ :- } d(U,X,DU), d(V,X,DV).$

$d(-T,X,-R) \text{ :- } d(T,X,R).$

$d(K*U,X,K*W) \text{ :- number}(K), d(U,X,W).$

Symbolické derivování – deriv.pro

$d(U*V,X,B*U+A*V) \text{ :- } d(U,X,A), d(V,X,B).$

$d(U/V,X,W) \text{ :- } d(U*V^{\text{(-1)}},X,W).$

$d(U^V,X,V*W*U^{\text{(V+(-1))}}) \text{ :- } \text{number}(V), d(U,X,W).$

$d(U^V,X,Z*\log(U)*U^V+V*W*U^{\text{(V+(-1))}}) \text{ :- } d(U,X,W),$
 $d(V,X,Z).$

$d(\log(T),X,R*T^{\text{(-1)}}) \text{ :- } d(T,X,R).$

Symbolické derivování – deriv.pro

$d(\exp(T), X, R * \exp(T)) :- d(T, X, R).$

$d(\sin(T), X, R * \cos(T)) :- d(T, X, R).$

$d(\cos(T), X, -R * \sin(T)) :- d(T, X, R).$

$d(\tan(T), X, W) :- d(\sin(T)/\cos(T), X, W).$