PENOPORA:

Prolog výpisky

© 2011 Petr Nohejl, [jestrab.net](http://jestrab.net), aktualizace: 13. 5. 2011

Zdroje: S3rvacovy příklady, opora, cvičení, slajdy, zkoušky z předmětu FLP @ FIT BUT

# Základy

## SWI Prolog

* spuštění: pl -s file.pl
* znovunačtení: reconsult(’file.pl’).
* nápověda: help(append).
* ukončení: halt.

## Syntaktické základy

Zdrojový program tvoří databáze predikátů, záleží na pořadí klauzulí. Prolog nemá líné vyhodnocování. Základní pojmy:

* **term**
  + **atom**: a, ahoj, ’ahoj ’, ’Ahoj ’, ’a’, ’@’
  + **číslo**: 3, 3.1415
  + **proměnná**: A, Ahoj, Ahoj\_123
* **složený term**: ahoj(cau(neco),dalsi),jiny(X,term)
  + **seznam**: [], [ahoj,cau,nazdar]
  + **řetězec**: "ahoj", [97,104,111,106]
* **cíl** - dotaz při spuštění programu
* **klauzule**, **predikát**
* **fakt** - klauzule bez těla
* **pravidlo**: A :- B, C, D. (cíl A je splněn, když jsou splněny podcíle)
* **instanciace**: X = term (z volné proměnné udělá vázanou)
* **synonimum**: X = Y (jiné jméno stejné proměnné)

## Operátory

|  |  |
| --- | --- |
| :- | definice pravidla |
| ?- | otázka |
| ; | or |
| -> | if-then-else |
| , | and |
| =, \= | porovnání |
| is | vyčíslení |
| <, >, =<, >= | porovnání |
| ==, \== | porovnání bez přiřazení |
| =:=, =\= | porovnání s vyhodnocením |
| +, -, \*, /, //, mod, div, ^ | matematické funkce |
| not, \+ | negace |

## Predikáty

Predikáty se v dokumentačním zápise značí jako: *jméno/arita* (např. *faktorial/2*).   
Zápis hlavičky: *predikat(+Vstupni, -Vystupni, ?VstupneVystupni).*

% radkovy komentar

\\* blokovy komentar \*\

velka\_hlava(lada). % fakt ze lada ma velkou hlavu

cepice(lada). % fakt ze lada ma cepici

velka\_cepice(X) :- cepice(X), velka\_hlava(X). % pravidlo s konjunkci - nizsi prior.

rostlina(X) :- strom(X); ker(X); bylina(X). % pravidlo s disjunkci - vyssi prior.

% faktorial/2, pouziti: faktorial(5, Vysledek).

faktorial(0, 1).

faktorial(N, Vysledek) :-

N > 0,

Predchozi is N-1,

faktorial(Predchozi, PredchoziVysledek),

Vysledek is PredchoziVysledek \* N.

% vypocet clenu Fibonacciho posloupnosti

fib(0, 0).

fib(1, 1).

fib(2, 1).

fib(X, R) :-

X > 2,

X1 is X - 1,

X2 is X - 2,

fib(X1, R1),

fib(X2, R2),

R is R1 + R2.

% nejvetsi spolecny delitel 2 cisel

gcd(X, X, X).

gcd(X, Y, Z) :-

X > Y, X1 is X - Y, gcd(X1, Y, Z).

gcd(X, Y, Z) :-

Y > X, Y1 is Y - X, gcd(X, Y1, Z).

% test prvocisel

isPrime(X) :- testPrime(X, 2).

testPrime(X, X) :- !.

testPrime(X, Y) :-

Z is X mod Y,

Z \= 0,

YY is Y + 1,

testPrime(X, YY).

## Fakty

Předpoklad uzavřeného světa (databáze), vyjádření pouze pozitivních informací.

% databaze rodinnych vztahu

muz(jan).

muz(pavel).

muz(robert).

muz(tomas).

muz(petr).

zena(marie).

zena(jana).

zena(linda).

zena(eva).

otec(tomas,jan).

otec(jan,robert).

otec(jan,jana).

otec(pavel,linda).

otec(pavel,eva).

matka(marie,robert).

matka(linda,jana).

matka(eva,petr).

rodic(X,Y) :- otec(X,Y),!; matka(X,Y),!.

sourozenec(X,Y) :- X\=Y, rodic(Z,X), rodic(Z,Y).

sestra(X,Y) :- zena(X), sourozenec(X,Y).

deda(X,Y) :- muz(X), otec(T,Y), otec(X,T).

je\_matka(X) :- zena(X), matka(X,\_).

teta(X,Y) :- sestra(X, M), matka(M, Y).

## Vestavěné predikáty

true. % vzdy uspesny predikat (mimo navraceni)

fail. % vzdy neuspesny predikat

repeat. % vzdy uspesny a znovusplnitelny predikat

not(X). % test na neuspech

X is 1+2. % vyhodnoceni aritmetickoho vyrazu, <promenna> is <vyraz>

X is X+1. % error!

If -> Then ; Else. % podmineny prikaz

max(X,Y,M) :- % vrati maximum

(X>Y -> M=X ; M=Y).

X=Y. % je možné termy unifikovat?

X\=Y. % není možné termy unifikovat?

X==Y. % jsou termy stejné?

X\==Y. % nejsou termy stejné?

integer(X). % test zda je promenna cislo

atom(X). % test zda je promenna atom

atomic(X). % test zda je promenna cislo nebo atom

var(X). % test zda neni promenna navazana na nejakou hodnotu

nonvar(X). % test zda je promenna navazana na nejakou hodnotu

arg(2,color(r,g,b),R). % N-ty argument termu, R = g

functor(color(r,g,b),color,3). % overeni nazvu/arity termu

## Unifikace

Při volání procedur zadáním cíle nebo explicitně pomocí =. Příklady využití unifikace:

X = 1. % navazani hodnoty na promennou

1 = X.

1 = 1. % test na unifikovatelnost (rovnost)

pes \= kocka.

List = [1|[2,3,4]]. % selektor polozek datovych struktur/seznamu

Term = barvy(X, Y).

List = [1,2,3]. % vytvareni datovych struktur/seznamu

Term = barvy(zelena, modra).

## Operátor řezu

% a :- b ; c % b selze, rizne, ale pokracuje se v c

% b :- d, !, e. % operator rezu ! reze jen v ramci predikatu

% c :- f.

% priklad sdeleni, ze jiz byla nalezena spravna varianta podcile

sum\_to(1,1) :- !. % soucet prvku od 1 do N, rez zabrani dalsi unifikaci

sum\_to(N,Res) :-

N1 is N-1,

sum\_to(N1,Res1),

Res is Res1+N.

% priklad kdy dalsi prohledavani nevede k reseni

not(P) :- call(P), !, fail. % operator not

% priklad ukonceni generovani alternativnich reseni

is\_integer(0). % test zda je promenna cislo (pro nasled. priklad)

is\_integer(X) :-

is\_integer(Y),

X is Y+1.

divide(\_, 0, \_) :- !, fail. % celociselne deleni

divide(N1, N2, Result) :-

is\_integer(Result),

Product1 is Result\*N2,

Product2 is (Result+1)\*N2,

Product1 =< N1, Product2 > N1, !. % bez ! by pri navraceni program zacyklil

# Seznam

Nehomogenní struktura, konstruktorem je *./2* (např. L = .(H, T), pozor na: *L=.(H,T)*). Nepotřebná proměnná se značí pomocí \_.

X = .(jan, .(tomas, [])). % konstrukce seznamu

neprazdny([\_|\_]) :- true. % neprazdnost seznamu

hlavicka([H|\_], H). % hlavicka seznamu

posledni([H], H) :- !. % posledni prvek seznamu

posledni([\_|T], Res) :- posledni(T, Res).

delka([],0). % delka seznamu

delka([\_|T],D) :- delka(T,TD), D is TD+1.

obsahuje(X, [X|\_]). % seznam obsahuje prvek

obsahuje(X, [\_|T]) :- obsahuje(X, T).

spoj([],L,L). % spojeni dvou seznamu do tretiho

spoj([H|T],L,[H|TT]) :- spoj(T,L,TT).

obrat([],[]) :- !. % reverzace seznamu

obrat([H|T], Res) :- obrat(T,TT), spoj(TT,[H],Res).

sluc(L,[],L). % slouceni dvou serazenych seznamu do tretiho sluc([],L,L).

sluc([X|XS], [Y|YS], [X|T]) :- X @< Y, sluc(XS, [Y|YS], T).

sluc([X|XS], [Y|YS], [Y|T]) :- X @>= Y, sluc([X|XS], YS, T).

serad([],[]). % serazeni seznamu

serad([H|T], SL) :- serad(T, S), sluc([H], S, SL).

isPalindrom([]). % test palindromu

isPalindrom([\_]) :- !.

isPalindrom([H|T]) :-

last(T, H),

reverse(T, [\_|RTT]),

reverse(RTT, TR),

isPalindrom(TR).

slice(\_, \_, [], []). % rez seznamu, vrati seznam od Nteho do Kteho prvku

slice(1, 1, [H|\_], [H]).

slice(1, Y, [H|T], [H|Res]) :-

Y >= 2,

YY is Y - 1,

slice(1, YY, T, Res).

slice(X, Y, [\_|T], Res) :-

X >= 2,

Y >= 2,

XX is X - 1,

YY is Y - 1,

slice(XX, YY, T, Res).

?- slice(2,5,[a,b,c,d,e,f],R). % uziti predikatu slice, vrati [b,c,d,e]

% je dan seznam predikatu a seznam jejich parametru

% test zda alespon N z nich je splnenych pro seznam zadanych parametru

a(1).

b(2).

c(1).

d(2).

atleast(Pr,Pa,N) :- N > -1, test(Pr,Pa,N).

test(A,B,0) :- !, isList(A), isList(B).

test([],\_,N) :- N>0, !, fail.

test([P|PS],Pa,N) :-

CC =.. [P|Pa],

call(CC), !,

N1 is N - 1,

test(PS,Pa,N1).

test([\_|Pr],Pa,N) :-

test(Pr,Pa,N).

isList([]).

isList([\_|\_]).

?- atleast([a,b,c,d],[1],2). % uziti predikatu atleast, vraci true

% ze seznamu seznamu vybere ten, jehoz delka se nejvic blizi cislu v polovine mezi

% delkou nejkratsiho a nejdelsiho seznamu na vstupu

selHalf([],\_) :- !, fail.

selHalf([H|T],L) :-

mapl([H|T],LL),

%minmax(MI,MA,LL),

minList(LL,MI),

maxList(LL,MA),

Half is (MI+MA) // 2,

selll(Half,[H|T],L).

mapl([],[]).

mapl([H|T],[HL|TL]) :-

length(H,HL),

mapl(T,TL).

selll(\_,[L],L) :- !.

selll(Ha,[H|T],L) :-

selll(Ha,T,LL),

length(H,HL),

length(LL,LLL),

HD is Ha - HL,

LLD is Ha - LLL,

abs(HD,AHD), abs(LLD,ALLD),

(AHD<ALLD -> L = H ; L = LL).

maxList([H], H).

maxList([H|T], M2) :-

maxList(T, M),

M2 is max(H, M).

minList([H], H).

minList([H|T], M2) :-

minList(T, M),

M2 is min(H, M).

## Vestavěné predikáty u seznamů

member(2,[1,2,3]). % test zda je prvek v seznamu

append([1,2,3],[4,5,6],L). % spojeni dvou seznamu

last([1,2,3],X). % posledni prvek seznamu

permutation([1,2,3],[3,2,1]). % test zda P2 je permutaci P1

reverse([1,2,3],R). % reverzace seznamu

barvy(zelena, zluta) =.. X. % prevod termu na seznam, X = [barvy, zelena, zluta]

Y =.. [barvy, zelena, zluta]. % prevod seznamu na term, Y = barvy(zelena, zluta)

% ukazka volani predikatu

aplikuj(Funkce,Parametr,Vysledek) :-

Predikat =.. [Funkce,Parametr,Vysledek],

call(Predikat).

inkrement(X,Y) :- Y is X+1.

main :- aplikuj(inkrement,3,V).

## Operátor řezu u seznamů

delete(\_, [], []). % odstraneni prvku ze seznamu

delete(X, [X|L], M) :-

!, delete(X, L, M). % bez ! bychom pri navraceni ziskavali seznamy,

delete(X, [Y|L1], [Y|L2]) :- % kde nejsou vsechny vyskyty X zcela odstraneny

delete(X, L1, L2).

remove(A,[A|L], L) :- !. % odstraneni prvku ze seznamu

remove(A,[B|L],[B|M]) :-

remove(A,L,M).

remove(\_,[],[]).

intersection([], X, []). % prunik dvou mnozin

intersection([X|R], Y, [X|Z]) :-

member(X,Y),

!, intersection(R,Y, Z). % bez ! by vysledna mnozina pri navraceni postupne

intersection([X|R], Y, Z) :- % ztracela cleny az do prazdna

intersection(R, Y, Z).

union([],X,X) :- !. % sjednoceni dvou mnozin

union([X|R],Y,Z) :-

member(X,Y),!,union(R,Y,Z).

union([X|R],Y,[X|Z]) :-

union(R,Y,Z).

range(S, S, [S]) :- !. % generator posloupnosti cisel od S do E

range(S, E, [S|T]) :- % ! zastavuje rekurzivni vypocet

S < E, SS is S+1,

range(SS, E, T), !.

range(\_, \_, []).

take(\_, [], []) :- !. % vrati prvnich N prvku seznamu

take(N, [H|T], [H|TT]) :-

N > 0,

!, % bez ! bychom meli vzdy dale mensi pocet prvku

NN is N-1, % v ruznych kombinacich

take(NN, T, TT).

take(N, [\_|\_], []) :-

N =< 0.

takeWhile(\_,[],[]). % vrati prvnich N prvku seznamu dokud plati dana funkce

takeWhile(P,[H|T],[H|TT]) :-

PP =.. [P,H], call(PP), !, takeWhile(P,T,TT).

takeWhile(\_,\_,[]).

?- takeWhile(odd,[1,3,5,6,7,9],L). % uziti predikatu takeWhile, L = [1,3,5,7].

dropWhile(\_,[],[]). % vrati poslednich N prvku jakmile prestane platit funkce

dropWhile(P,[H|T],TT) :-

PP =.. [P,H], call(PP), !, dropWhile(P,T,TT).

dropWhile(\_,L,L).

?- dropWhile(odd,[1,3,4,5,7,9],R). % uziti predikatu dropWhile, L = [4,5,7,9].

less\_than\_3(X) :- X<3. % test < 3 (pro nasled. priklad)

split(\_,[],([],[])) :- !. % rozdeli seznam kdyz funkce prestane platit

split(P,L,R) :- split(P,L,[],R).

split(\_,[],W,(RW,[])) :- reverse(W,RW).

split(P,[H|T],W,R) :-

PP =.. [P,H], call(PP), !, split(P,T,[H|W],R).

split(\_,R,L,(RL,R)) :- reverse(L,RL).

?- split(less\_than\_3,[1,2,3,4,5,6],L). % uziti predikatu split, L=([1,2],[3,4,5,6]).

## Predikáty vyššího řádu

map(\_, [], []). % predikat map

map(F, [H|T], [NH|NT]) :- P =.. [F,H,NH],

call(P),

map(F,T,NT).

inc(X,Y) :- var(Y), Y is X+1, !. % inkrement (pro nasled. priklad)

inc(X,Y) :- nonvar(Y), Z is Y-1, Z=X.

?- map(inc,[1,2,3],X). % pouziti map, vysledek [2,3,4]

?- map(inc,X,[2,3,4]). % pouziti map, vysledek [1,2,3]

filter(\_, [], []) :- !. % predikat filter

filter(P, [H|T], [H|TT]) :-

PP =.. [P,H], call(PP), % zde by mohlo byt i zkracene jen: call(P, [H])

!, % bez ! bychom pri navraceni postupne dostavali

filter(P, T, TT). % seznam s mensim a mensim poctem spravnych prvku

filter(P, [\_|T], TT) :-

filter(P, T, TT).

even(V) :- X is V mod 2, X = 0. % test sudych cisel

odd(X) :- % test lichych cisel

Y is X // 2,

YY is Y \* 2,

YY \= X.

?- filter(even,[1,2,3,4,5,6],R). % pouziti filter, vysledek [2,4,6]

foldr(\_, B, [], B). % predikat foldr

foldr(F, B, [H|T], BB) :-

foldr(F, B, T, BT),

P =.. [F,H,BT,BB], call(P).

foldl(\_, A, [], A). % predikat foldl

foldl(F, A, [H|T], AA) :-

P =.. [F,A,H,AT], call(P),

foldl(F, AT, T, AA).

add(X,Y,Z) :- ZZ is Y+X, ZZ=Z. % soucet (pro nasled. priklad)

conS(T,H,[H|T]). % prohozeni (pro nasled. priklad)

sum(L, S) :- foldr(add, 0, L, S). % pouziti foldr, suma seznamu

rev(L, RL) :- foldl(conS, [], L, RL). % pouziti foldl, reverzace seznamu

## Řazení

% razeni typu generuj a testuj

gtsort(L1,L2) :- % serazeni seznamu

mypermutation(L1,L2), sorted(L2), !.

sorted([]). % test serazeni seznamu

sorted([\_]).

sorted([A,B|L]) :-

A =< B, sorted([B|L]).

mypermutation([],[]). % generovani permutaci

mypermutation(L,[H|T]) :-

append(V,[H|U],L),

append(V,U,W),

mypermutation(W,T).

mypermutation2([], []). % jine generovani permutaci

mypermutation2([H|T], P) :-

mypermutation2(T, TP),

insert(H, TP, P).

insert(X,[],[X]). % vlozeni prvku na zacatek seznamu

insert(X,[Y|Y1],[X,Y|Y1]).

insert(X,[Y|Y1],[Y|Z1]) :- insert(X,Y1,Z1).

% razeni na principu vkladani

insort(L1,L2) :- insort(L1,[],L2). % serazeni seznamu

insort([],X,X).

insort([X|X1],Y,Z) :-

insertSorted(X,Y,Z1),

insort(X1,Z1,Z).

insertSorted(X,[],[X]). % vlozeni prvku do seraz. seznamu

insertSorted(X,[Y|Y1],[X,Y|Y1]) :- X =< Y, !.

insertSorted(X,[Y|Y1],[Y|Z1]) :-

insertSorted (X,Y1,Z1).

% bubble sort

bubble(L1,L2) :- % serazeni seznamu

append(X, [A,B|Y], L1), A>B,

append(X, [B,A|Y], Z), bubble(Z,L2).

bubble(L,L).

% quick sort

quick([],[]). % serazeni seznamu

quick([H|T],S) :-

split(T,H,A,B),

quick(A,A1), quick(B,B1),

append(A1,[H|B1],S).

split([],\_,[],[]). % rozdeleni seznamu na 2 casti

split([X|X1],Y,[X|Z1],Z2) :-

X<Y, split(X1,Y,Z1,Z2).

split([X|X1],Y,Z1,[X|Z2]) :-

X>=Y, split(X1,Y,Z1,Z2).

% nejdelsi neklesajici posloupnost ze vsech neklesajicich posloupnosti

maxNondecreasingSeq([], []).

maxNondecreasingSeq([H|T], Res) :-

nondecreasingSeq([H|T], HTRes),

maxNondecreasingSeq(T, TRes),

longestList(HTRes, TRes, Res).

% nejdelsi neklesajici posloupnost zacinajici prvnim prvkem seznamu

nondecreasingSeq([], []).

nondecreasingSeq([X], [X]) :- !.

nondecreasingSeq([X1,X2|T], [X1|Res]) :-

X1 =< X2,

nondecreasingSeq([X2|T], Res).

nondecreasingSeq([X1,X2|\_], [X1]) :-

X1 > X2.

% vrati delsi ze dvou seznamu

longestList(L1, L2, L1) :-

length(L1, L1L),

length(L2, L2L),

L1L > L2L.

longestList(L1, L2, L2) :-

length(L1, L1L),

length(L2, L2L),

L1L =< L2L.

## Množina

% vypocet mnoziny vsech podmnozin, podmnoziny vraci v seznamu

subbags([], [[]]).

subbags([X|XS], XSS) :- subbags(XS, XX),

addOneToAll(X, XX, XXX),

append(XX, XXX, XSS).

% pridej prvek do vsech mnozin

addOneToAll(\_, [], []).

addOneToAll(E, [L|LS], [[E|L]|T]) :- addOneToAll(E, LS, T).

% vypocet mnoziny vsech podmnozin, podmnoziny vraci postupne prohledavanim prostoru

subbags2([],[]).

subbags2([H1|T1],[H1|T2]) :-

subbags2(T1,T2).

subbags2([\_|T1],T2) :-

subbags2(T1,T2).

% test zda vsechny prvky z druhe mnoziny jsou podmnozinou prvni

mysubset([],[]).

mysubset([\_|\_],[]).

mysubset([H|T],[HH|TT]) :- myelem(HH,[H|T]), mysubset([H|T],TT), !.

myelem(H,[H|\_]) :- !.

myelem(V,[\_|T]) :- myelem(V,T).

?- mysubset([1,2,3,4,5],[2,4,3]). % uziti mysubset

% mnozinovy rozdil

rozdil(M1,M2,RES) :- permutation(RES,P), rozdil2(M1,M2,P), !.

rozdil2([],\_,[]) :- !.

rozdil2([H|T], L, [H|R]) :- not(element(H,L)), !, rozdil2(T,L,R).

rozdil2([H|T], L, R) :- element(H,L), rozdil2(T,L,R).

element(X,[X|\_]) :- !.

element(X,[\_|T]) :- element(X,T).

## Strom

hloubkaStromu(list,0).

hloubkaStromu(uzel(\_,L,P),H) :-

hloubkaStromu(L,LH), hloubkaStromu(P,PH),

max(LH,PH,MAX),

H is MAX+1.

% pouziti predikatu hloubkaStromu

?- hloubkaStromu(

uzel(2,uzel(1,list,list),

uzel(4,uzel(3,list,list),list)),

H).

empty\_tree(leaf). % prazdny strom

add2tree(K, V, leaf, node(K,V,leaf,leaf)). % pridani stromu

add2tree(K, \_, node(K,V,X,Y), node(K,V,X,Y)) :-

!.

add2tree(Kn, Vn, node(K,V,L,R), node(K,V,LL,R)) :-

Kn < K, !,

add2tree(Kn, Vn, L, LL).

add2tree(Kn, Vn, node(K,V,L,R), node(K,V,L,RR)) :-

add2tree(Kn, Vn, R, RR).

?- add2tree(6,petr,leaf,R), add2tree(10,jana,R,RR). % uziti predikatu add2tree

inOrder(leaf, []). % inorder pruchod stromem

inOrder(node(\_,Value,L,R), List) :-

inOrder(L, LL),

inOrder(R, RL),

append(LL, [Value|RL], List).

?- inOrder(Tree, List). % uziti predikatu inOrder

preOrder(leaf, []). % preorder pruchod stromem

preOrder(node(\_,Value,L,R), List) :-

preOrder(L, LL),

preOrder(R, RL),

append([Value|LL], RL, List).

list2tree([],leaf). % prevod seznamu na strom

list2tree([I|IS], NewTree) :-

list2tree(IS, Tree),

add2tree(I, I, Tree, NewTree).

?- list2tree([1,2,3],T). % uziti predikatu list2tree

search(\_, leaf, \_) :- % vyhledani hodnoty podle klice

!, fail.

search(Key, node(Key,Value,\_,\_), Value) :-

!.

search(Key, node(KeyT,\_,L,\_), Value) :-

Key < KeyT, !,

search(Key, L, Value).

search(Key, node(\_,\_,\_,R), Value) :-

search(Key, R, Value).

?- search(6, Tree, Data). % uziti predikatu search

% vyrazy ve stromu

val(3). % hodnota 3

op(plus, val(3), val(8)). % operace 3+8

op(sub, % vyraz 3\*5-2

op(mul, val(3), val(5)),

val(2)

)

eval(val(X),X). % vyhodnoceni hodnoty

eval(op(plus,L,R,),X) :- % vyhodnoceni souctu

eval(L,LX), eval(R,RX),

X is LX+RX.

eval(op(minus,L,R),X) :- % vyhodnoceni rozdilu

eval(L,LX), eval(R,RX),

X is LX-RX.

eval(op(mul,L,R),X) :- % vyhodnoceni nasobeni

eval(L,LX), eval(R,RX),

X is LX\*RX.

eval(op(div,L,R),X) :- % vyhodnoceni deleni

eval(L,LX), eval(R,RX),

X is LX//RX.

## Lambda kalkul

% reprezentace lambda kalkulu

lambdaExpr( % promenna

var(\_) ).

lambdaExpr(

appl( lambdaExpr(\_), lambdaExpr(\_) ) ). % aplikace

lambdaExpr(

abstr( var(\_), lambdaExpr(\_) ) ). % abstrakce

% vraci seznam s polozkami prvniho bez duplicit

removeDups([], []).

removeDups([H|T], [H|TRes]) :-

not(member(H, T)),

removeDups(T, TRes).

removeDups([H|T], TRes) :-

member(H, T),

removeDups(T, TRes).

?- removeDups([3,1,2,3,2,3,1],R). % uziti removeDups, vraci [2,3,1]

% vraci seznam vsech volnych promennych v lambda vyrazu

unboundedVars(E, RDRes) :-

uvImpl(E, [], Res),

removeDups(Res, RDRes).

uvImpl(lVar(X), L, [X]) :-

not(member(X, L)).

uvImpl(lVar(X), L, []) :-

member(X, L).

uvImpl(lAppl(X, Y), L, Res) :-

uvImpl(X, L, XRes),

uvImpl(Y, L, YRes),

append(XRes, YRes, Res).

uvImpl(lAbstr(lVar(X), Y), L, Res) :-

uvImpl(Y, [X|L], Res).

% uziti unboundedVars, vraci [b]

?- unboundedVars(lAbstr(lVar(a), lAppl(lVar(a), lVar(b))), R).

# Databáze

assert(barva(zelena)). % ulozeni predikatu do databaze

retract(barva(cervena)). % unifikace termu a odstraneni predikatu

retractall(barva(\_)). % odstraneni vsech predikátu s hlavickou

listing(barva). % zobrazeni predikatu v databazi

clause(barva(zelena),X). % vyber klauzule z databaze podle hlavicky

## Prohledávání stavového prostoru

% overeni kroku, prostor 9x9, zadan vychozi a koncovy bod, uhlopricny pohyb

nextStep(X, Y, XX, YY) :-

XX is X+1, YY is Y+1, test(XX, YY).

nextStep(X, Y, XX, YY) :-

XX is X+1, YY is Y-1, test(XX, YY).

nextStep(X, Y, XX, YY) :-

XX is X-1, YY is Y+1, test(XX, YY).

nextStep(X, Y, XX, YY) :-

XX is X-1, YY is Y-1, test(XX, YY).

test(X, Y) :- X>0, Y>0, X<10, Y<10.

% hledani cesty L z bodu S do pozice E, pos (navstivena mista) zamezi chozeni v kruhu

:- dynamic pos/2.

searchPathL(start(X,Y), end(X,Y), [p(X,Y)]) :- % S a E se shoduji

!.

searchPathL(start(X,Y), E, [p(X,Y)|T]) :- % realizace 1 kroku

assert(pos(X,Y)),

nextStep(X, Y, XX, YY),

not( pos(XX,YY) ),

searchPathL(start(XX,YY), E, T).

searchPathL(start(X,Y), \_, \_) :- % krok nelze ucinit, navraceni

pos(X, Y),

retract(pos(X,Y)),

fail.

?- searchPathL(start(1,1),end(2,2),L). % uziti predikatu searchPathL

% smazani predikatu pos z databaze, slo by pouzit i retractall

clearPos :-

pos(X, Y),

retract( pos(X,Y) ),

!, clearPos.

% hledani cesty z bodu S do pozice E, vysledky uklada do databaze (pos)

searchPath(start(X,Y), end(X,Y)) :-

assert( pos(X,Y) ).

searchPath(start(X,Y), end(X,Y)) :-

pos(X, Y),

retract( pos(X,Y) ),

!, fail.

searchPath(start(X,Y), E) :-

assert( pos(X,Y) ),

nextStep(X, Y, XX, YY),

not( pos(XX,YY) ),

searchPath(start(XX,YY), E).

searchPath(start(X,Y), \_) :-

pos(X, Y),

retract( pos(X,Y) ),

fail.

?- searchPath(start(1,1),end(6,4)). % uziti predikatu searchPath

?- listing(pos).

% ulozeni cesty z databaze (pos) do seznamu

listPos(L) :-

listPos(1,L).

listPos(N,[X,Y|T]) :-

nth\_clause(pos(\_,\_),N,R),

clause(pos(X,Y),\_,R),

NN is N+1,

listPos(NN,T), !.

listPos(\_,[]) :- !.

?- listPos(L). % uziti predikatu listPos

% ulozeni cesty z databaze (pos) do seznamu s pouzitim zip

slistPos(L) :-

bagof(X, Ye^pos(X,Ye),Xs),

bagof(Y, Xe^pos(Xe,Y),Ys),

zip(Xs,Ys,L).

zip([],\_,[]) :- !.

zip(\_,[],[]).

zip([X|XS],[Y|YS],[X,Y|XYS]) :-

zip(XS,YS,XYS).

?- slistPos(L). % uziti predikatu slistPos

## Predikát bagof a setof

Predikát bagof nalezne všechny unifikace dané proměnné, které splní daný cíl: bagof(Vzor, Cil, Bag). Setof nevrací duplicity.

* Vzor - proměnná, kterou chci unifikovat
* Cíl - cíl s proměnnou, pro který se unifikace hledají
* Bag - výsledný seznam všech navázání

foo(a,b,c).

foo(a,b,c).

foo(a,b,d).

foo(b,c,e).

foo(b,c,f).

foo(c,c,g).

?- bagof(C, foo(A,B,C), BAG). % uziti bagof, vrati [c,c,d];[e,f];[g]

?- bagof(C, A^foo(A,B,C), BAG). % A^ znaci ze nas A nezajima, vrati [c,c,d];[e,f,g]

?- setof(C, foo(A,B,C), SET). % uziti setof, vrati [c,d];[e,f];[g]

?- setof(C, A^foo(A,B,C), SET). % A^ znaci ze nas A nezajima, vrati [c,d];[e,f,g]

## Praktické příklady

% Roboti svet: pozice dana celym cislem, pohyb doleva/doprava, na 1 pozici max 1 robot

:- dynamic robot/2, dira/1. % deklarace dynamickych predikatu

% databaze

robot(KRYTON, 0).

robot(R2D2, 1).

dira(2).

obsazeno(POS) :- robot(\_, POS) ; dira(POS). % obsazeno robotem nebo dirou

vytvor(ID, POS) :- not(obsazeno(POS)), assert( robot(ID, POS) ). % vytvori robota

vytvor(POS) :- not(obsazeno(POS)), assert( dira(POS) ). % vytvori diru

odstran(POS) :- ( dira(POS), retract(dira(POS)) ) ; % odstrani prvek

( robot(ID,POS), retract(robot(ID,POS)) ).

obsazene\_pozice(X) :- bagof(POS, obsazeno(POS), X). % seznam obsazenych pozic

% seznam pozic kde jsou roboti, ID^ znamena ze nas ta promenna nezajima

obsazene\_roboty(X) :- bagof(POS, ID^robot(ID, POS), X).

% pohyb robotu

inkrementuj(X, Y) :- Y is X+1.

dekrementuj(X, Y) :- Y is X-1.

doleva(ID) :- pohni(ID, dekrementuj).

doprava(ID) :- pohni(ID, inkrementuj).

pohni(ID, Operace) :- robot(ID, POS),

call(Operace, POS, NEWPOS),

retract(robot(ID, POS)),

(

obsazeno(NEWPOS) ->

( robot(\_,NEWPOS) -> odstran(NEWPOS); true ) ;

assert(robot(ID,NEWPOS))

).

armageddon :- forall( robot(\_,POS), vybuch(POS) ). % vybuch vsech robotu

vybuch(POS) :- odstran(POS), vytvor(POS).

% Problem N dam: rozestaveni na sachovnici aby se neohrozovaly, kodovani reseni - seznam N cisel, udavajici pozici postupne ve sloupcich, generovani vsech reseni - vygenerovani jednoho a pouziti predikatu permutation

sequence(0, []) :- !.

sequence(N, [N|T]) :- NN is N-1, sequence(NN,T). % sekvence cisel od N do 1

queens(Solution) :- queens(8, Solution). % pro sachovnici 8x8

queens(N, Solution) :- sequence(N, List), permutation(List, Solution), test(Solution).

test([]) :- !. % test neohrozovani dam

test([H|T]) :- test(H,1,T), test(T).

test(\_, \_, []) :- !.

test(Pos, Dist, [H|T]) :-

Pos \= H,

X is abs(Pos-H),

X \= Dist,

Dn is Dist+1,

test(Pos, Dn, T).

:- dynamic size/2, pos/2.

% pocet vsech acyklickych cest kone

% parametry: rozmer sachovnice, vychozi misto, cilove misto, pocet

cesty(XR,YR,XS,YS,XE,YE,Num) :-

XR > 0, YR > 0,

assert(size(XR,YR)),

testPos(XS,YS),

testPos(XE,YE),

setof(P,search(XS,YS,XE,YE,P),LRes),

length(LRes,Num),

retractall(pos(\_,\_)),

retractall(size(\_,\_)).

testPos(X,Y) :- X>0, Y>0, size(SX,SY), X=<SX, Y=<SY.

nextStep(X,Y,XN,YN) :- XN is X+1, YN is Y+2, testPos(XN,YN).

nextStep(X,Y,XN,YN) :- XN is X-1, YN is Y+2, testPos(XN,YN).

nextStep(X,Y,XN,YN) :- XN is X+1, YN is Y-2, testPos(XN,YN).

nextStep(X,Y,XN,YN) :- XN is X-1, YN is Y-2, testPos(XN,YN).

nextStep(X,Y,XN,YN) :- XN is X+2, YN is Y+1, testPos(XN,YN).

nextStep(X,Y,XN,YN) :- XN is X-2, YN is Y+1, testPos(XN,YN).

nextStep(X,Y,XN,YN) :- XN is X+2, YN is Y-1, testPos(XN,YN).

nextStep(X,Y,XN,YN) :- XN is X-2, YN is Y-1, testPos(XN,YN).

search(X,Y,X,Y,[X:Y]).

search(X,Y,X,Y,\_) :- !, fail.

search(X,Y,XE,YE,[X:Y|RST]) :-

assert(pos(X,Y)),

nextStep(X,Y,XN,YN),

not(pos(XN,YN)),

search(XN,YN,XE,YE,RST).

search(X,Y,\_,\_,\_) :-

pos(X,Y),

retract(pos(X,Y)),

!, fail.

:-dynamic id/2.

% asociativni pamet assocMem(id, key, value), kdyz se zada key tak hleda value,

% kdyz se zada value tak hleda klic a kdyz se zada oboje tak pridava do databaze

assocMem(A,K,V) :- var(A), !, fail.

assocMem(A,K,V) :- var(K), var(V), !, fail.

assocMem(A,K,V) :- var(V), P =.. [A,K,V], call(P).

assocMem(A,K,V) :- var(K), P =.. [A,Y,V], setof(Y,P,K).

assocMem(A,K,V) :-

not(var(K)),

not(var(V)),

P =.. [A,K,\_],

call(P), !,

retract(P),

PP =.. [A,K,V],

assert(PP).

assocMem(A,K,V) :-

not(var(K)),

not(var(V)),

P =.. [A,K,V],

assert(P).

?- assocMem(id,mykey,myval).

?- assocMem(id,mykey,X).

:- dynamic size/2, pos/2.

% nalezeni nejkratsi cesty v 2D prostoru, uhlopricny pohyb

% na miste Switch mozno chodit i vodorovne a svisle, na misto Barrier nelze vstoupit

% parametry: rozmer sachovnice, vychozi misto, cilove misto, cesta

shortest(W,H,XS,YS,XE,YE,Path) :-

retractall(size(\_,\_)),

retractall(pos(\_,\_)),

W > 0, H > 0,

assert(size(W,H)),

test(XS,YS),

test(XE,YE),

setof(P,search(XS,YS,XE,YE,P),Paths),

shortest(Paths,Path).

test(X,Y) :- X>0, Y>0, size(W,H), X =< W, Y =< H.

nextStep(X,Y,XN,Y) :- switch(X,Y), XN is X + 1, test(XN,Y), not(barrier(XN,Y) ).

nextStep(X,Y,XN,Y) :- switch(X,Y), XN is X - 1, test(XN,Y), not(barrier(XN,Y) ).

nextStep(X,Y,X,YN) :- switch(X,Y), YN is Y + 1, test(X,YN), not(barrier(X,YN) ).

nextStep(X,Y,X,YN) :- switch(X,Y), YN is Y - 1, test(X,YN), not(barrier(X,YN) ).

nextStep(X,Y,XN,YN) :- XN is X+1, YN is Y+ 1, test(XN,YN), not(barrier(XN,YN) ).

nextStep(X,Y,XN,YN) :- XN is X-1, YN is Y+1, test(XN,YN), not(barrier(XN,YN) ).

nextStep(X,Y,XN,YN) :- XN is X+1, YN is Y-1, test(XN,YN), not(barrier(XN,YN) ).

nextStep(X,Y,XN,YN) :- XN is X-1, YN is Y-1, test(XN,YN), not(barrier(XN,YN) ).

search(X,Y,X,Y,[X:Y]).

search(X,Y,X,Y,\_) :- !, fail.

search(X,Y,XE,YE,[X:Y|RST]) :-

assert(pos(X,Y)),

nextStep(X,Y,XN,YN),

not(pos(XN,YN)),

search(XN,YN,XE,YE,RST).

search(X,Y,\_,\_,\_) :-

pos(X,Y),

retract(pos(X,Y)),

!, fail.

shortest([],\_) :- !, fail.

shortest([H|T],P) :- length(H,L), spath(H,L,T,P).

spath(S,\_,[],S).

spath(\_,L,[H|T],P) :- length(H,HL), HL < L, !, spath(H,HL,T,P).

spath(S,L,[\_|T],P) :- spath(S,L,T,P).

barrier(2,2).

switch(2,3).

switch(1,1).

?- shortest(5,5,1,1,3,3,R). % uziti predikatu shortest, vrati [1:1,1:2,2:3,3:3]