Beskonačna petlja

p:-p.

- P je tačno, ako je p tačno

?- p.

Program je deklarativno ispravan, a proceduralno neispravan

U pojedinim slučajevima promenom redosleda klauza i ciljeva može se izbeći beskonačna petlja.

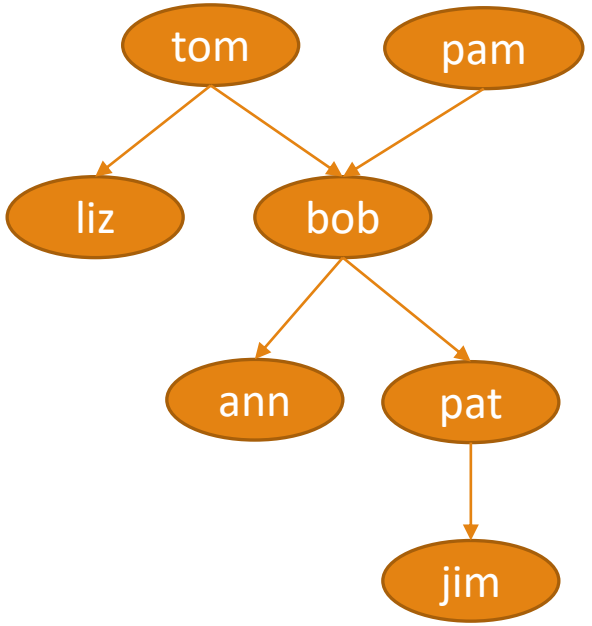
Izbegavanje beskonačne petlje

```
anc1(X,Z):-parent(X,Z).
anc1(X,Z):-parent(X,Y),
           anc1(Y,Z).
```

```
anc2(X,Z):-parent(X,Y),
           anc2(Y,Z).
anc2(X,Z):-parent(X,Z).
```

```
anc3(X,Z):-parent(X,Z).
anc3(X,Z):-anc3(X,Y),
           parent(Y,Z).
```

```
anc4(X,Z):-anc4(X,Y),
           parent(Y,Z).
anc4(X,Z):-parent(X,Z).
```




```
28 ?- anc1(tom,pat).
true .

29 ?- anc2(tom,pat).
true .

30 ?- anc3(tom,pat).
true .

31 ?- anc4(tom,pat).
ERROR: Stack limit (1.0Gb
ERROR:    Stack sizes: loc
```

Liste

Reprezentacija listi

[ann, tennis, tom,skiing]

- ann – glava liste
- [tennis, tom,skiing] – rep liste

[Head|Tail]

```
36 ?- [a,b,c] = [a|[b|[c|[]]]].
true.

36 ?- List1 = [a,b,c], List2 = [a|[b|[c|[]]]].
List1 = List2, List2 = [a, b, c].

37 ?- Hobbies1 = [tennis|[music|[]]],
      Hobbies2 = [skiing,food],
      L = [ann,Hobies1,tom,Hobies2].
Hobbies1 = [tennis, music],
Hobbies2 = [skiing, food],
L = [ann, Hobies1, tom, Hobies2].
```

L = [ann,[tennis,music],tom,[skiing,food]]

Element liste

X je element liste L ako je

- X glava liste L ili je
- X element repa liste L

```
member(X,[X|Tail]).
member(X,[Head|Tail]):-
                member(X,Tail).
```

```
member(X,[X|_]).
member(X,[_|Tail]):-
                member(X,Tail).
```

```
41 ?- member(b,[a,b,c]).
true .

42 ?- member(X,[a,b,c]).
X = a ;
X = b ;
X = c ;
false.

43 ?- member(a,L).
L = [a|_370] ;
L = [_1028, a|_1036] ;
L = [_1028, _1694, a|_1702] ;
L = [_1028, _1694, _2360, a|_2368] .

44 ?- member(a,L),member(b,L),member(c,L).
L = [a, b, c|_4632] ;
L = [a, b, _5290, c|_5298] ;
L = [a, b, _5290, _5956, c|_5964] ;
L = [a, b, _5290, _5956, _6622, c|_6630] ;
L = [a, b, _5290, _5956, _6622, _7288, c|_7296] .

45 ?- L=[_,_,_],member(a,L),member(b,L),member(c,L).
L = [a, b, c] ;
L = [a, c, b] ;
L = [b, a, c] ;
L = [c, a, b] ;
L = [b, c, a] ;
L = [c, b, a] ;
false.
```

Nadovezivanje lista

```
conc([],L,L).
conc([X|L1],L2,[X|L3]):-
                    conc(L1,L2,L3).
```

```
?- conc(L1,L2,[a,b,c]).
L1 = [],
L2 = [a, b, c] ;
L1 = [a],
L2 = [b, c] ;
L1 = [a, b],
L2 = [c] ;
L1 = [a, b, c],
L2 = [] ;
false.
```

```
?- conc([a,b],[c,d],X).
X = [a, b, c, d].
```

```
?- conc([a,b,c],[1,2,3],L).
L = [a, b, c, 1, 2, 3].
```

```
?- conc([a,[b,c],d],[a,[],b],L).
L = [a, [b, c], d, a, [], b].
```

```
?- conc(Before,[may|After],
|    [jan,feb,mar,apr,may,jun,jul,avg,sep,oct,nov,dec]).
Before = [jan, feb, mar, apr],
After = [jun, jul, avg, sep, oct, nov, dec] .
```

```
?- L1=[a,b,z,z,c,z,z,z,d,e],conc(L2,[z,z,z|_],L1).
L1 = [a, b, z, z, c, z, z, z, d|...],
L2 = [a, b, z, z, c] .
```

```
member1(X,L):-conc(_,[X|_],L).
```

Operacije sa listama

Spajanje lista

Dodavanje elemenata

Brisanje elemenata

Podliste

Permutacije

Dužina liste

Aritmetika

Aritmetika

Operacije

- +, -, *, /,
- ** (stepenovanje),
- // (celobrojno deljenje),
- mod

Relacije

- >, <, >=, =<
- =:=
- =\=

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

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

```
49 ?- 1+2 =:= 2+1.
true.

50 ?- 1+2=2+1.
false.

51 ?- 1+A=B+2.
A = 2,
B = 1.
```

```
14 ?- X is 2, Y is X+2,Y\=X.
X = 2,
Y = 4.

15 ?- X is 2, Y is X+2,Y=\=X.
X = 2,
Y = 4.

16 ?- X = a(1,2), Y = a(2,3), X\=Y.
X = a(1, 2),
Y = a(2, 3).

17 ?- X = a(1,2), Y = a(2,3), X=\=Y.
ERROR: Arithmetic: `a/2' is not a funct
ERROR: In:
ERROR:   [11] a(1,2)=\=a(2,3)
```

Rebusi

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

```
donald([D,O,N,A,L,G,E,R,B,T]):-
    assign([0,1,2,3,4,5,6,7,8,9],[D,O,N,A,L,G,E,R,B,T]),
   100000*D + 10000*O + 1000*N + 100*A + 10*L + D +
   100000*G + 10000*E + 1000*R + 100*A + 10*L + D =:=
   100000*R + 10000*O + 1000*B + 100*E + 10*R + T.
```

```
assign(_,[]).
assign(Digs,[D|Vars]):-
    del(D,Digs,Digs1),
   assign(Digs1,Vars).
```

```
del(X,[X|Tail],Tail).
del(X,[Y|Tail],[Y|Tail1]):-
    del(X,Tail,Tail1).
```

Rebusi

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

```
2 ?- donald([D,O,N,A,L,G,E,R,B,T]),
     L1 = [D,O,N,A,L,D], L2 = [G,E,R,A,L,D], L3 = [R,O,B,E,R,T].
D = 5,
O = 2,
N = 6,
A = 4,
L = 8,
G = 1,
E = 9,
R = 7,
B = 3,
T = 0,
L1 = [5, 2, 6, 4, 8, 5],
L2 = [1, 9, 7, 4, 8, 5],
L3 = [7, 2, 3, 9, 7, 0] .
```

Napraviti opštiji predikat koji rešava ma koji rebus

```
  SEND
+ MORE
-------
 MONEY
```

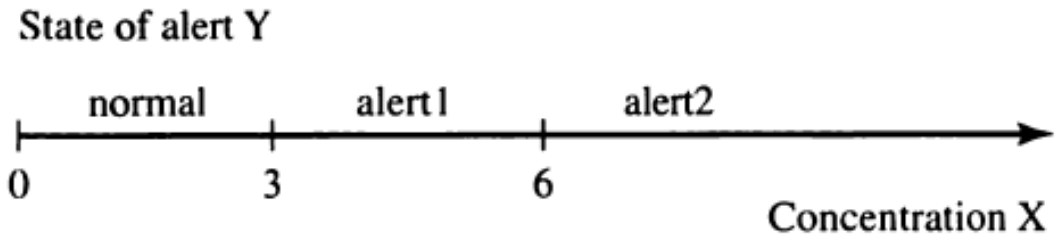
Problem 8 (ili n) dama na šahovsku tablu 8x8 (ili nxn)

Backtracking, Cut

Kontrola backtracking-a

Prolog automatski radi backtracking da bi zadovoljio dati cilj

Nekontrolisani backtracking može da dovede do neefikasnog programa



Pravilo 1: Ako je X<3 tada je Y = *normal*
Pravilo 2: Ako je 3≤X i X<6 tada je Y = *alert1*
Pravilo 3: Ako je 6≤X tada je Y = *alert2*

```
f(X,normal):- X<3.
f(X,alert1):- 3=<X, X<6.
f(X,alert2):- 6=<X.
```

Kontrola backtracking-a

```
f(X,normal):- X<3.
f(X,alert1):- 3=<X, X<6.
f(X,alert2):- 6=<X.
```

```
2 ?- f(2,alert1).
false.
```

Aktivira pravilo 2: X =2 – ciljevi u telu pravila nisu ispunjeni
Ostala pravila ne može da aktivira

```
3 ?- f(2,Y),Y=alert1.
false.
```

Aktivira pravilo 1: X =2, Y=normal

Uslov normal = alert1 nije ispunjen

Aktivira pravilo 2: X=2, Y=alert1 -- ciljevi u telu pravila nisu ispunjeni

Aktivira pravilo 3: X=2, Y=alert2 -- ciljevi u telu pravila nisu ispunjeni

Pravila su međusobno isključujuća i samo jedno od njih može biti tačno

Cut

```
f(X,normal):- X<3,!.
f(X,alert1):- 3=<X, X<6,!.
f(X,alert2):- 6=<X.
```

```
3 ?- f(2,Y),Y=alert1.
false.
```

Aktivira pravilo 1: X =2, Y=normal – ciljevi u telu, aktiviran cut (rez)

Uslov normal = alert1 nije ispunjen

Aktivacija reza ne dozvoljava traženje novih dokaza.

```
6 ?- f(7,Y).
Y = alert2.
```

Aktivira pravilo 1: Y=normal – ciljevi nisu ispunjeni, rez nije aktiviran

Aktivira pravilo 2: Y=alert1 – ciljevi nisu ispunjeni, rez nije aktiviran

Aktivira pravilo 3: Y=alert2 -- ciljevi u telu pravila ispunjeni

Cut

Ako je X<3 tada je Y = *normal*

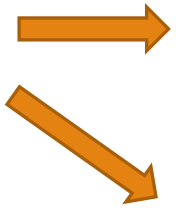
Inače ako je X<6 tada je Y = *alert1*

Inače Y = *alert2*

```
f(X,normal):- X<3,!.
f(X,alert1):- X<6,!.
f(X,alert2).
```

!

```
f(X,normal):- X<3.
f(X,alert1):- X<6.
f(X,alert2).
```



```
12 ?- f(2,alert1).
true.

13 ?- f(2,Y),Y=alert1.
false.
```

```
8 ?- f(1,Y).
Y = normal ;
Y = alert1 ;
Y = alert2.
```

```
9 ?- f(2,alert1).
true .

10 ?- f(2,Y),Y=alert1.
Y = alert1 .
```

Maksimum dva broja

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

```
15 ?- max(3,1,1).
false.
```

```
max(X,Y,X):-X>=Y,!.
max(X,Y,Y).
```

```
17 ?- max(3,1,1).
true.
```

```
max(X,Y,Max):-
     X>=Y,!,Max=X
    ;
    Max=Y.
```

```
19 ?- max(3,1,1).
false.
```

```
26 ?- max(4,8,X).
X = 8.
```

Element liste s jednim rešenjem

```
member(X,[X|_]):-!.
member(X,[_|Tail]):-member(X,Tail).
```

```
?- member(X,[a,b,c]).
X = a.
```

fail

"Mary likes all animals but snakes."

```
likes(mary,X):-animal(X),X\=snake.
```

If X is a snake then "Mary likes X" is not true

otherwise if X is an animal then Mary likes X.

```
likes(mary,snake):-!,fail.
likes(mary,X):-animal(X).
```

```
33 ?- likes(mary,cat).
true.

34 ?- likes(mary,snake).
false.
```

fail

Da li su X i Y različiti?

```
different(X,X):-!,fail.
different(X,Y).
```

```
different(X,Y):-
      X=Y,!,fail
     ;
     true.
```

```
different(X,Y):-not(X=Y).
```

```
39 ?- different(3,3).
false.

40 ?- different(a,b).
true.
```

Rez

Pomoću reza moguće je povećati efikasnost programa

- Prologu se ekspilicitno naglašava da ne pokušava da dokaže alternative, jer će biti neuspešne

Korišćenjem reza definišu se uzajamno isključiva pravila, pa je moguće definisati pravila oblika

IF uslov P THEN zaključak Q

OTHERWISE zaključak R

Ovime se povećava izražajnost jezika

Rez

```
p:-a,b.
p:-c.
```

$p \Leftrightarrow (a \wedge b) \vee c$

```
p:-a,!,b.
p:-c.
```

$p \Leftrightarrow (a \wedge b) \vee (\neg a \wedge c)$

```
p:-c.
p:-a,!,b.
```

$p \Leftrightarrow c \vee (a \wedge b)$

Pojava reza može da promeni deklarativno značenje programa

Zeleni rez – nema uticaj na deklarativno značenje programa. Pri čitanju programa može se ignorisati.

Crveni rez – menja deklarativno značenje programa, programi su teži za tumačenje i mora se pažljivo koristiti.

Često se umesto `cut-fail` kombinacije koristi `not`

Negacija neuspeha

```
round(ball).
```

```
45 ?- round(ball).
true.
```

Da, iz programa sledi da je lopta okrugla

```
46 ?- round(earth).
false.
```

Nije moguće iz programa izvesti da je Earth okrugla, tako da ne znam

```
47 ?- not(round(earth)).
true.
```

not(round(earth)) nije logička posledica programa, već je rezultat negacije neuspeha

Pretpostavke zatvorenog sveta

- Izvođenje zaključaka negacijom neuspeha je bazirano na pretpostavci zatvorenih svetova
- U svakom programu se podrazumeva da je opisano sve što je tačno u svetu koji dat programom
- Sve što nije dato u programu se podrazumeva da nije tačno (da je pogrešno)
- Ovo podrazumeva posebnu pažnju je svakodnevni život nije zatvoreni svet

Problem sa not

```
good_standard(jeanluis).
expensive(jeanluis).
good_standard(francesco).
reasonable(Restaurant):-
      not(expensive(Restaurant)).
```

```
49 ?- good_standard(X),reasonable(X).
X = francesco.

50 ?- reasonable(X),good_standard(X).
false.
```

```
51 ?- expensive(X).
X = jeanluis.

52 ?- not(expensive(X)).
false.
```

not(expensive(X))

↔ not(postoji X takav da je expensive(X))

↔za SVE x: not(expensive(X)) – ni jedan restoran nije skup?!