PROLOG:

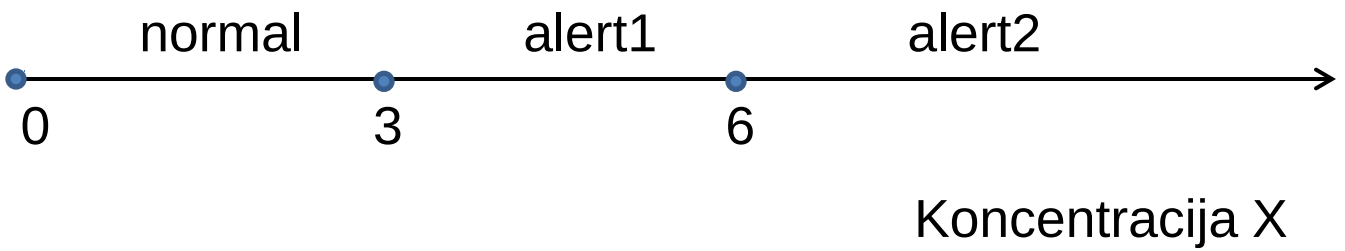
KLICAJ IN NEGACIJA

Ivan Bratko

AVTOMATSKO VRAČANJE NI VEDNO ZAŽELENO

- Avtomatsko vračanje je vgrajeno v prolog
- Pogosto je koristno in bistveno skrajša program
- Vendar pa je včasih avtomatsko vračanje nepotrebno ali celo nezaželeno
- Vračanje preprečimo s klicajem (rez; angl. cut)

PRIMER: STANJA ONESNAŽENOSTI



Stanje onesnaženosti Y je funkcija koncentracije X:

Y = f(X)

PRAVILA ZA DOLOČANJE STANJA

- *Rule 1*: if $X < 3$ then Y = normal
- *Rule 2*: if $3 \leq X$ and $X < 6$ then Y = alert1
- *Rule 3*: if $6 \leq X$ then Y = alert2

V PROLOGU: f(Koncentracija, Stanje)

```
f( X, normal)   :-   X < 3.          % Rule 1
f( X, alert1)   :-   3 =< X, X < 6.  % Rule 2
f( X, alert2)   :-   6 =< X.         % Rule 3
```

POSKUS 1

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

no

- Sled izvajanja pokaže nepotrebno vračanje, ko že vemo, da alternative ne bodo uspele

VERZIJA 2

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

Bolj učinkovito kot verzija 1,

klicaji ne vplivajo na logični pomen.

To je: Če klicaje zbrišemo, bodo rezultati programa enaki

POSKUS 2

?- f(7, Y).

Y = **alert2**

- Sled izvajanja spet pokaže nekaj odvečnega dela (preverjanje komplementarnih pogojev)

VERZIJA 3

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

Najbolj učinkovito ...

Toda: Spremenil se je logični pomen!

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

- Bolj pazljiva formulacija vprašanja:

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

no

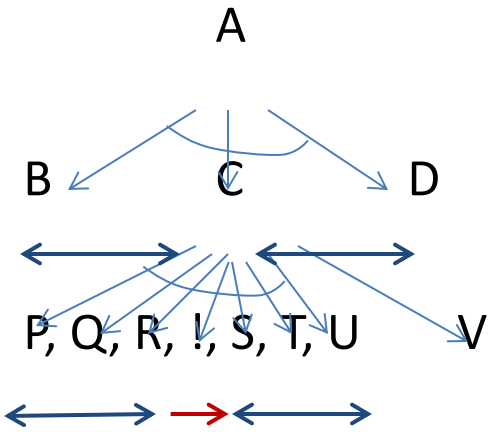
DOSEG KLICAJA

C :- P, Q, R, !, S, T, U.

C :- V.

A :- B, C, D.

?- A.



Klicaj vpliva na izvajanje cilja C;

vračanje še vedno velja med B, C, D (klicaj “ni viden” iz cilja A)

MAKSIMUM

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

% Bolj učinkovito s klicajem
max( X, Y, X)   :-   X >= Y, !.
max( X, Y, Y).

% Toda pazi!!!
?- max( 3, 1, 1).
yes
```

PREVIDNA FORMULACIJA

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

?-  max( 3, 1, 1).
no
```

“Mary likes all animals but snakes”

If X is a snake then “Mary likes X” is not true,

otherwise if X is an animal then Mary likes X.

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

likes( mary, X) :-
    animal( X).
```

KLICAJ VPLIVA NA DEKLARATIVNI POMEN

p :- a, b.

p :- c.

- Pomeni: p <===> (a & b) v c

p :- a, !, b.

p :- c.

- Pomeni: p <===> (a & b) v (~a & c)

- Če zamenjamo vrstni red stavkov:

p :- c.

p :- a, !, b.

- Pomen se spremeni:

p <===> c v (a & b)

NEGACIJA

```
not( P) :-
    P, !, fail
    ;
    true.
```

- *Negacija kot neuspeh (negation as failure)*

- not pišemo tudi kot prefiksni operator: not P

  ali kot operator \+: \+ P

- Zapis “\+” je bolj standarden in je bolj pogosto vgrajen v prolog

FORMULACIJA Z NEGACIJO

```
likes( mary, X) :-
    animal( X),
    not snake( X).
```

Izgleda boljše kot s klicajem + fail

NEGACIJA KOT NEUSPEH

- Ni točno enaka negaciji v logiki (matematiki)

- Velja le pod predpostavko zaprtega sveta (Closed World Assumption, CWA)

PREDPOSTAVKA ZAPRTEGA SVETA

- Program:

round(ball).

?- round(ball).

yes % Yes, it logically follows from program

?- round(earth).

no % I don’t know; it doesn’t logically follow from program

?- \+ round(earth).

yes % It follows from program, but only under CWA

PROBLEMI Z NEGACIJO

```
good_standard( jeanluis).        % JeanLuis is a good quality restaurant
expensive( jeanluis).            % JeanLuis is expensive
good_standard( francesco).

reasonable( Restaurant) :-       % A restaurant is reasonably priced if
    \+ expensive( Restaurant).   % it is not expensive
```

?- good_standard(X), reasonable(X).

X = francesco

?- reasonable(X), good_standard(X).

no % Surprize! What happened?

- Prolog pod negacijo spremeni običajno kvantifikacijo spremenljivk!
- V drugem vprašanju reasonable(X) uspe le, če ***za vse*** X velja reasonable(X), oz. za vse X velja \+ expensive(X)
- Uporaba negacije je varna, če so spremenljivke pod negacijo opredeljene