Liste

Lista je niz uređenih elemenata proizvoljne dužine. Element liste može biti bilo koji term pa čak i druga lista.

Formalnija definicija:

Lista je:

- **prazna lista u oznaci [];**
- **struktura .(G,R) gde je G ma koji term, a R lista.**

Ovo je rekurzivna definicija. Funktor u strukturi liste je tačka (.), prvi argument naziva se glava, a drugi rep. Drugi argument je uvek lista.

Zbog rekurzivne definicije liste, rekurzija je najpogodniji način za obradu listi.

Ako pođemo od definicije liste, imamao:

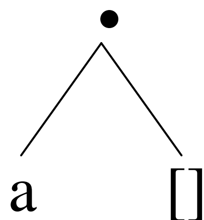
[] je prazna lista (lista koja ne sadrži ni jedan element)

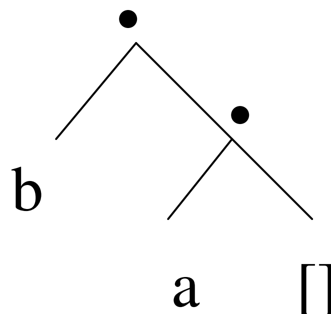
.(a, []) je jednočlana lista gde je a nekakav term.

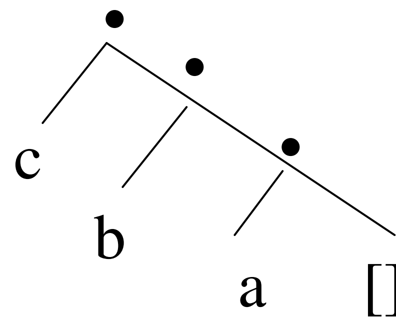
.(b, .(a,[])) je dvočlana lista (a i b su termi).

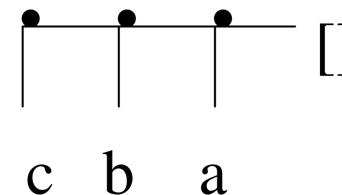
.(c, .(b, .(a,[]))) je tročlana lista (a, b i c su termi).

Ovaj postupak možemo nastaviti i kreirati listu proizvoljne dužine. Svaka od ovih listi (kao i svaka struktura) može se predstaviti preko drvoidne strukture.









Zapis liste **.(a,.(b, .(c, [])))** je glomazan i nepregledan. Umesto ovog, zgodnije je napisati: **[a,b,c].** Ovo je zapis liste pomoću zagrada. Svaka lista se na ovaj način može zapisati: **[], [a], [a,b], [a, b,c], [a,b,c,d],** … Prvi element u listi predstavlja glavu, a ostali elementi čine listu koja predstavlja rep.

Primeri:

Lista	**Glava**	**Rep**
[mira, mara, dara]	**mira**	**[mara, dara]**
[[voz, tramvaj],trolejbus]	**[voz, tramvaj]**	**[trolejbus]**
[a]	**a**	**[]**
[]	**-**	**-**
[to, [jabuka, so]]	**to**	**[[jabuka,so]]**
[X+Y, a+b]	**X+Y**	**[a+b]**

Sa listama se najčešće operiše tako što se razdvaja glava od repa. Stoga je pogodno imati zapis liste u kojem su jasno razdvojene ove dve komponente. Takav zapis postoji u PROLOG-u i to je:

[G|R]

U ovoj strukturi G se unifikuje sa glavom liste, a R sa repom.

Pogledati primer Lista1.pl

Unifikacija listi

Lista1	Lista2	Konkretizacija
[X, Y, Z]	[pera, mika, zika]	X=pera, Y=mika, Z=zika
[racunar]	[X|Y]	X=racunar, Y=[]
[X,Y|Z]	[aca, mara, ana]	X=aca, Y=mara, Z=[ana]
[G|kruska]	[jabuka|R]	G=jabuka , R= kruska
[X,Y|Z,V]	nekorektan zapis liste	
[[ah,X]|Y]	[[uh,Z]|V]	Nema unifikacije

Pripadnost elementa listi

Kada je zadata lista, jedan od važnijih zadataka je ispitivanje da li dati term pripada listi.

To se može uraditi pomoću predikata: pripada(X, Lista):

pripada(X, [Y|_]) :- X=Y.

pripada(X, [_|Y]) :- pripada(X,Y).

ili

pripada(X, [X|_]).

pripada(X, [_|Y]) :- pripada(X,Y).

Ovde se primenjuje rekurzivni postupak. Izlaz iz postupka traženja je

- nađen je element u listi i odgovor je yes   ili
- došlo se do kraja listi i ispituje se da li element pripada praznoj listi. Posto prazna lista nema elemenata, odgovor je: no.

Primer:

?- pripada(m, [a,b,c,d]).

m <> a, ide se na 2. pravilo:

?- pripada(m, [b,c,d]).

m <> b, ide se na 2. pravilo:

?- pripada(m, [c,d]).

m <> c, ide se na 2. pravilo:

?- pripada(m, [d]).

m <> d, ide se na 2. pravilo:

?- pripada(m, []).

Generiše se odgovor: no

Nekorektna primena rekurzije

Rekurzija je ‘prirodan’ način rešavanja problema u PROLOG-u.

Treba biti oprezan u primeni rekurzije, tj. izbegavati beskonačne cikluse (kada nema osnovnog slučaja, tj. nema izlaza iz kruga) poput sledećeg:

```
roditelj(A,B) :- dete(B, A).

dete(X,Y) :- roditelj(Y,X).
```

No, nisu dozvoljene ni tzv. levostrane rekurzije (javlja se kada pravilo generiše podcilj istovetan polaznom cilju):

```
covek(X) :- covek(Y), majka(X,Y).

covek(adam).
```

Nakon upita:

```
?- covek(X).
```

generiše se greška koja nastaje usled prepunjenja steka.

Ako imamo:

```
lista([A|B]):- lista(B).
lista([]).
```

Problem se rešava navođenjem činjenice pre pravila:

```
lista1([]).
lista1([A|B]) :- lista1(B).
```

ili:

```
lista2([]).
lista2([_|_]).
```

Razgovor.pl

3.3. Stringovi

String (niz slova između navodnike) je lista ASCII-kodova tih slova.

“system” = [115,121,115, 116, 101, 109]

“$3*Ara+=“ =[36, 51, 42, 65, 114, 97, 43, 61]

Primeri programa sa listama

```
?- duzina(Lista, Duzina).

duzina([],0).

duzina([G|R],D) :- duzina(R,D1), D  is D1+1.

?- suma(Lista, ZbirElemenataListe).

suma([], 0).

suma([Glava|Rep],S) :-suma(Rep,S1), S is Glava+S1.

?- poslednji(PoslednjiElemetListe, Lista).

poslednji(X,[X]).

poslednji(X,[_|Y]):- poslednji(X,Y).
```

```
?- spoj(L1,L2,L).

spoj([],X,X).

spoj([Xg|Xr],Y,[Xg|Z]) :- spoj(Xr,Y,Z).
```

Program, osim spajanja, omogućava i razdvajanje listi (deluje u inverznom smeru).

```
?-obrni(Prava, Inverzna).

obrni([],[]).

obrni([Xg|Xr],Y):- obrni(Xr,Y1), spoj(Y1,[Xg],Y).
```

4. Način rešavanja problema u PROLOG-u. Stablo pretraživanja.

Kako PROLOG rešava postavljeni problem?

Svaki upit PROLOG tretira kao cillj koji treba dostići (ostvariti, ispuniti).

To PROLOG-mašina čini pokušavajući da dokažee saglasnost cilja sa bazom znanja. U tom procesu baza znanja se pregleda od vrha ka dnu i moguće su 2 situacije:

- pronađeno je tvrđenje koje se uparuje sa postvljenim ciljem (uspešno je ostavaren cilj - uspeh) ili

- nije pronađeno tvrđenje koje se uparuje sa postavljenim ciljom (cilj nije ispunjen- neuspeh).

U slučaju uspeha, korisnik može zahtevati da se ponovo dokaže saglasnost cilja sa bazom podataka.

Primer1:

```
→otac(marko, petar).          ?-otac(_,X).
→otac(milos, janko).           X=petar;
 otac(marija, pavle).          X=janko;
 otac(milan, pavle).           ...
```

Primer2:

```
3.  2. 1.
        →momak(marko).                          ?-moguci_par(M,D).
   →     momak(janko).                          M=marko, D=ana;
         momak(petar).                          M=marko, D=mara;
   →    →devojka(ana).                          M=marko, D=sonja;
   →    →devojka(mara).                         M=janko, D=ana;
    .   →devojka(sonja).                        ..........
         moguci_par(M,D):-momak(M), devojka(D).
```

4.1. Stablo pretraživanja

Stablo pretraživanja omogućava slikovit prikaz načina rešavanja problema u PROLOG-u.

Neka je zadat program P nizom tvrđenja i cilj C.

1. Koren stabla pretraživanja je cilj C.

?-C.

2. Ako je

?-C_i,C_{i+1},…,C_n

čvor stabla pretraživanja i tačka C_i pozitivan podcilj, tada čvor ima po jednog potomka za za svaki podcilj. Neka je za C_i taj potomak G:-T_1,T_2, …T_k iz programa P. Sa njim se može unifikovati C_i preko opšteg unifikatora S. Potomok čvora tada glasi:

?-(T_1,T_2,…,T_k, C_{i+1},…,C_n)*S

3. Ako je

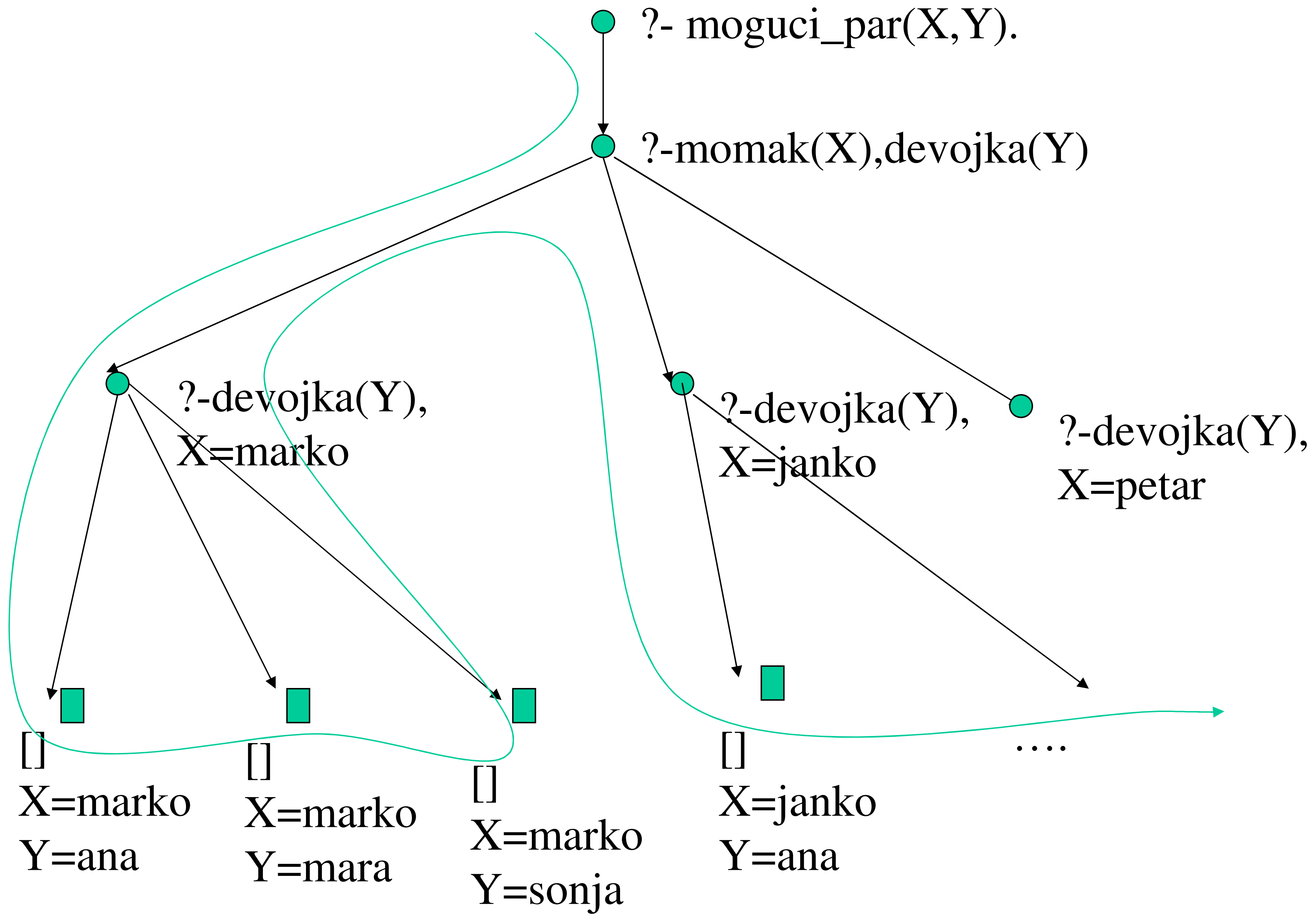
$?-C_i, C_{i+1}, \ldots, C_n$

čvor stabla pretraživanja gde je C_i negativan potcilj (oblika not(C_i')) i ako C_i' ima konačno stablo pretraživanja sa granom uspeha, onda C_i nema potomaka. Ako C_i' ima konačno stablo pretraživanja bez ijedne grane uspeha, onda je potomak prethodnog čvora:

$?- C_{i+1}, \ldots, C_n.$

4. Potomak koji nema potcilja (bez elemenata) , naziva se prazan cilj i označava se sa **[].**

Čvorovi uspeha su čvorovi koji sadrže prazan cilj. Čvorovi sa nepraznim ciljem, ali bez potomaka su čvorovi neuspeha.


?- moguci_par(X,Y).
?-momak(X),devojka(Y)
?-devojka(Y),
X=marko
?-devojka(Y),
X=janko
?-devojka(Y),
X=petar
[]
X=marko
Y=ana
[]
X=marko
Y=mara
[]
X=marko
Y=sonja
[]
X=janko
Y=ana
....

4.1.1. Primeri stabla pretraživanja

Primer 1:

lekar(marko).

lekar(stanko).

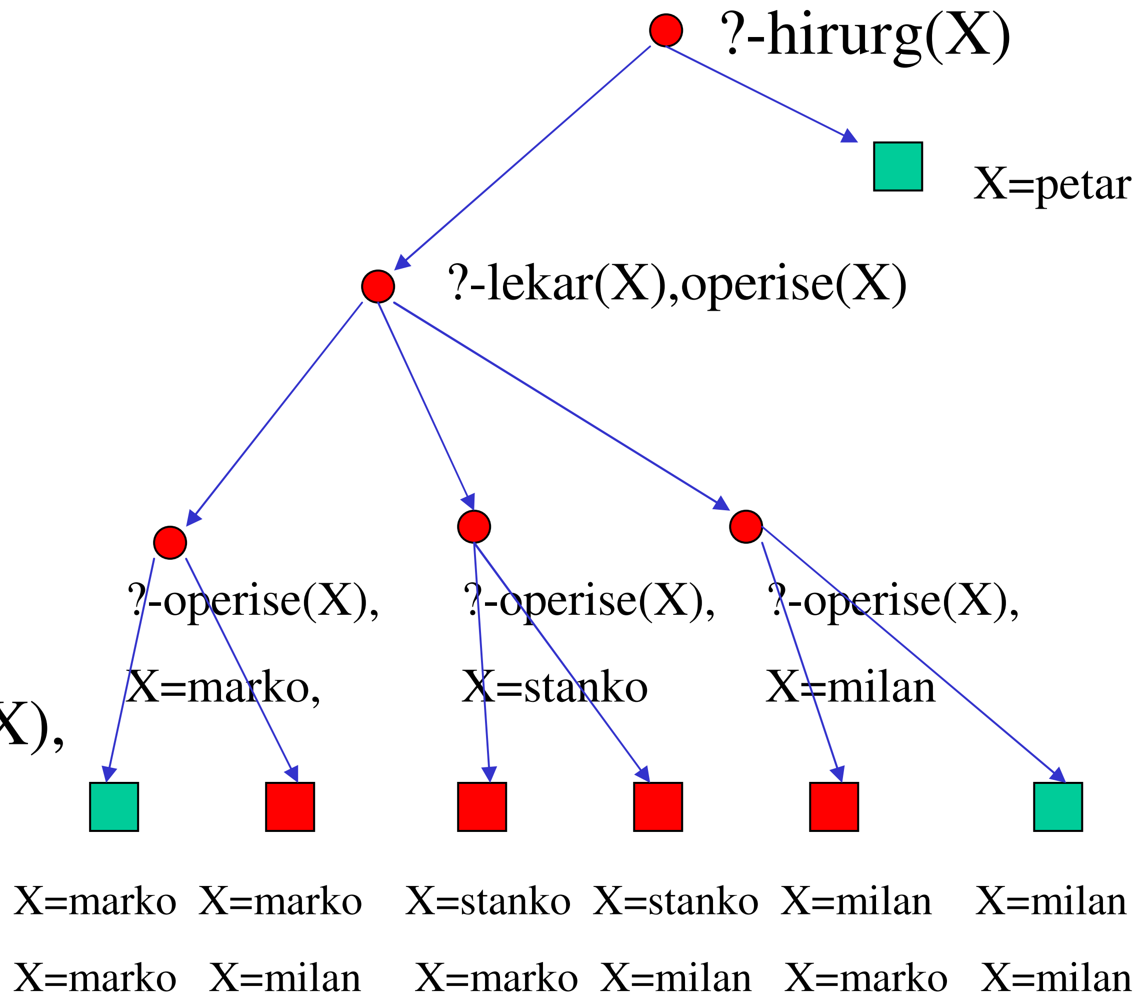
lekar(milan).

operise(marko).

operise(milan).

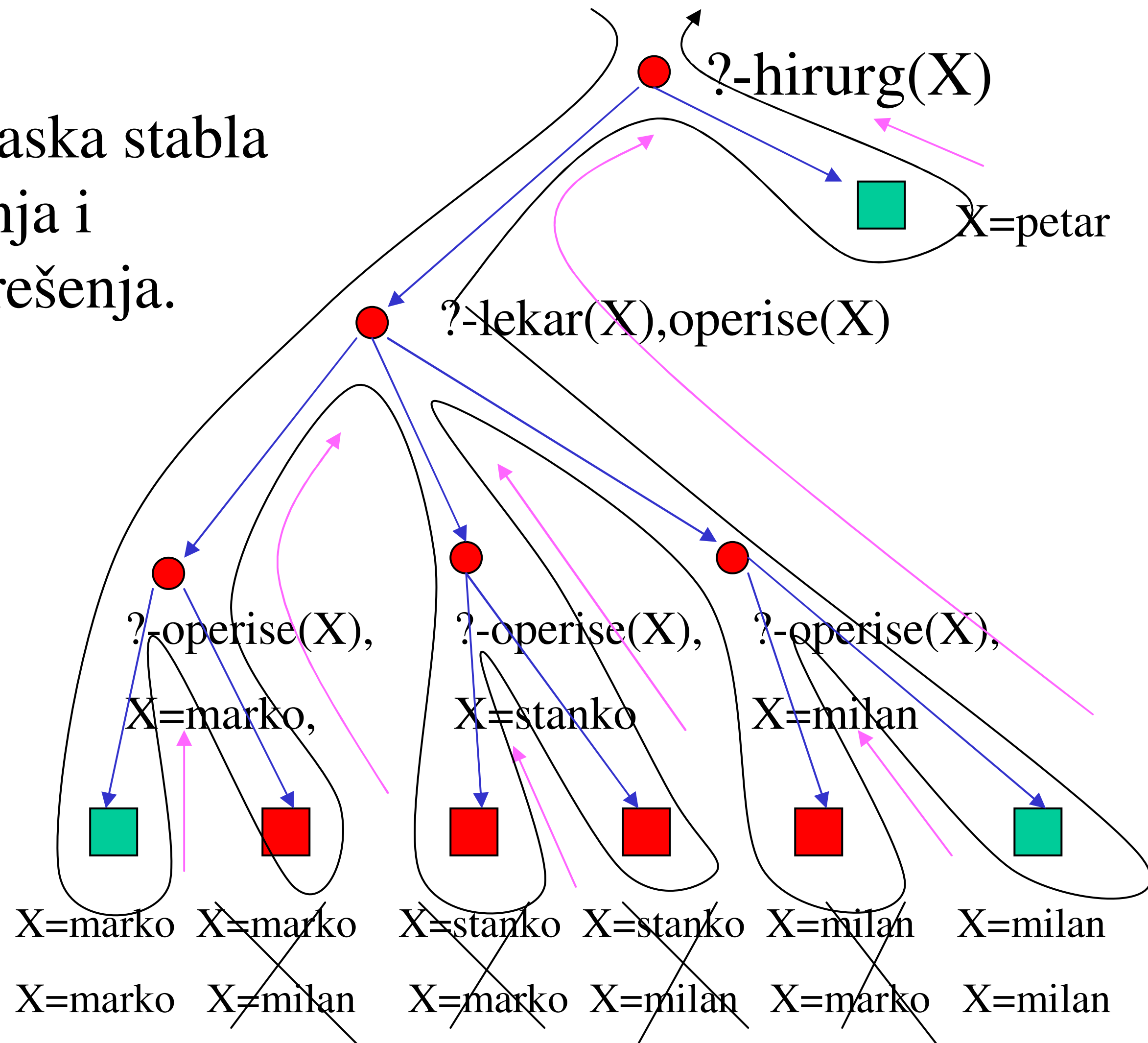
hirurg(X):- lekar(X), operise(X).

hirurg(petar).

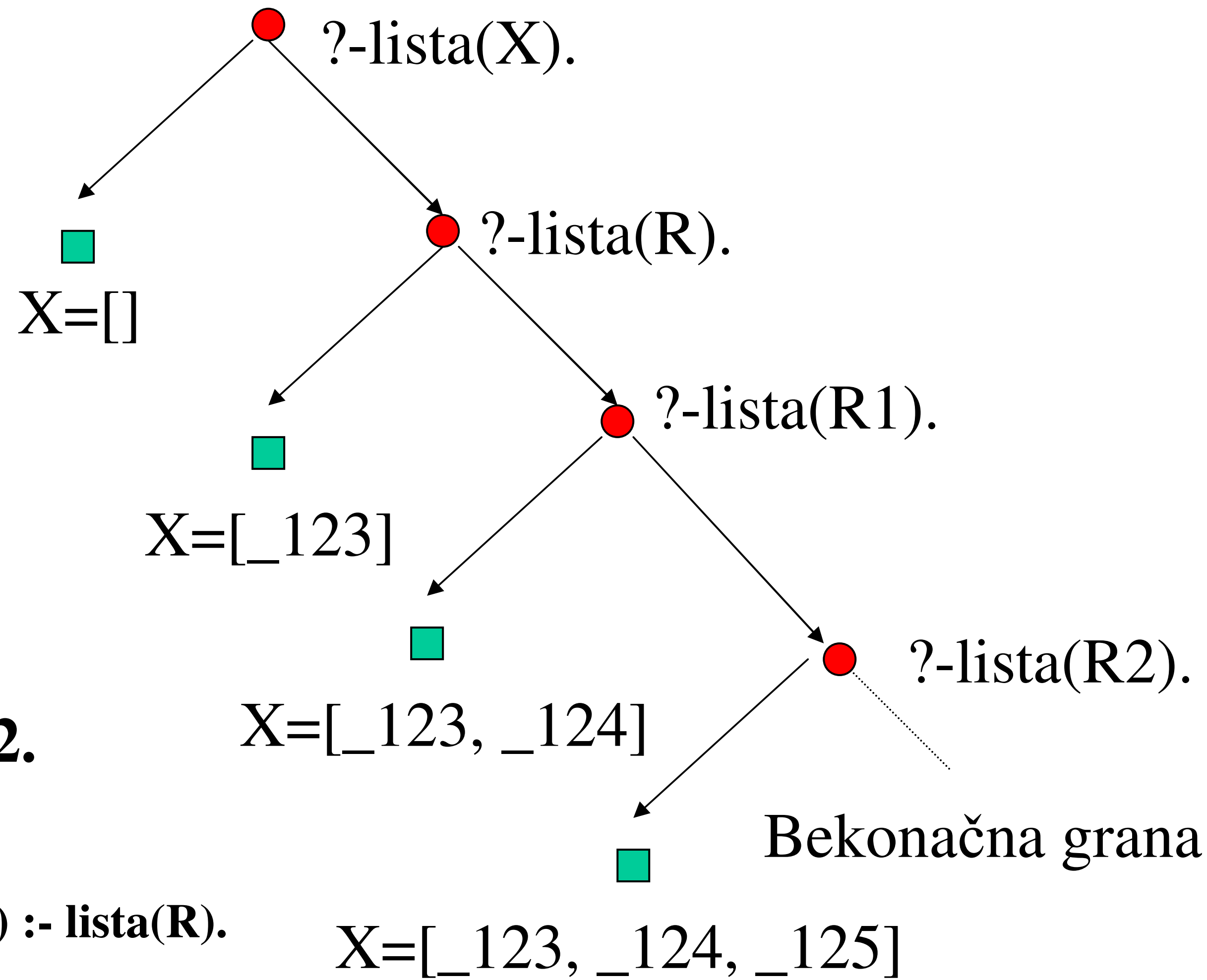


Način obilaska stabla pretraživanja i nalaženja rešenja.

hirurg.pl



Pomoću stabla pretraživanja potpuno je određen prostor pretraživanja za jedan cilj. Njega čine svi putevi koji vode od korena stabla pretraživanja do njegovih čvorova.



Primer 2.

```
lista([]).
lista([G|R]) :- lista(R).
```

4.1.2. Terminologija u vezi sa stablom pretraživanja

1. Stablo pretraživanja se sastoji iz grana i čvorova. Grana u stablu pretraživanja može biti konačna ili beskonačna.

2. Stablo pretraživanja je konačno ako su sve grane u njemu konačne, u suprotnom je beskonačno.

2. Čvorovi koji su označeni upitom (ciljem) nazivaju se nezavršnim. (Iz njih se izvode sledeći čvorovi.)

3. Listovi u stablu pretraživanja nazivaju se završnim čvorovima. Grane koje vode do završnih čvorovo su konačne. Grane koje nemaju završne čvorove su beskonačne.

4. Ako zavšni čvor daje rešenje, naziva se se čvor uspeha (označen zelenom bojom), a odgovarajuća grana je grana uspeha. Završni čvor koji ne predstavlja rešenje je čvor neuspeha (označen crvenom bojom), a odgovarajuća grana je grana neuspeha.

5. Rez-predikat (Predikat odsecanja, Cut-predikat)

To je sistemski predikat koji omogućava:

- brže izvršavanje programa
- uštedu memorijskog prostora

Rez-predikat odseca pojedine grane na stablu pretraživanja (otud i njegov naziv!), pa samim tim smanjuje prostor pretraživanja. To omogućava brže nalaženje rešenja. U isto vreme ne moraju se pamtiti mnogobrojne tačke prilikom traženja sa vraćanjem, što dovodi do uštede memorijskog prostora.

Rez-predikat se označava sa: ! (znakom uzvika) i uvek uspeva.

To je kontraverzan predikat (poput GOTO-naredbe u proceduralnim jezicima). Narušava deklarativni stil programiranja i može da proizvede neželjene efekte (greške).

5.1. Dejstvo Rez-predikata

Neka imamo tvrđenje:

A :- B1, B2, …, Bk, !, …, Bm

Kada se neki cilj G unifikuje sa A, aktivira se prethodno tvrđenje. Tada se cilj G naziva roditeljski cilj za A. Kada se dođe do reza, uspeli su potciljevi: B1, B2, …,Bk. Rez uspeva i rešenje B1, B2, …, Bk se "zamrzava", tj. Rez-predikat onemogućava traženje alternativnih rešenja. Takođe se onemogućava ujedinjavanje cilja G sa glavom nekog drugog predikata koji je u bazi podataka iza navedenog tvrđenja.

Ako imamo:

A:- P,Q,R, !, S, T.

A :- U.

G :- L, A, D.

?- G.

Traženje sa vraćanjem moguće je samo između: P,Q i R. Kad uspe R uspeva i Cut i alternativna rešenja se više ne traže. Alternativno rešenje A:-U, takođe se ne razmatra. Alternativna rešenja su moguća između S i T. Roditeljski cilj je A.

Sumarno, Cut-predikat ima sledeće efekte:

1. Ako uspe u nekom tvrđenju, seče tvrđenja koja imaju istu glavu i arnost sa tvrđenjem u kojem je rez-predikat i koja su ispod tog tvrđenja.

Primer:

t(X,Y,Z) :- u(X,Y), !, …

t(X,Y,Z) :- v(X), w(Y, Z) …

t(2, jabuka, zora).

...

2.Kojunkcija potciljeva u tvrđenju levo do Cut-predikata uspeva samo jedanput. (Cut-predikat deluje kao brana!).

A :- b, c, …, m, !, p, q, …,z.

Konjinkcija potciljeva: b,c,…,m uspeva samo jedanput.

3. Cut-predikat ne utiče na podciljeve desno do njega, tj. tamo je omogućen backtracking.

A :- b, c, …, m, !, p, q, …,z.

Ako u procesu traženja rešenja konjunkcija potciljeva: p,q,…,z ne uspe jedanput, pokušava se ponovo zadovoljenje ove konjunkcije.

4. Cut-predikat ne utiče na tvrđenja izvan grana stabla pretraživaja koje polaze iz čvora u kojem je Cut- predikat.

t(X,Y,Z) :- p(X),q(Y), r(Z).   (*)

t(X,Y,Z) :- u(X,Y), !, …

t(X,Y,Z) :- v(X), w(Y, Z) …

t(2, jabuka, zora).

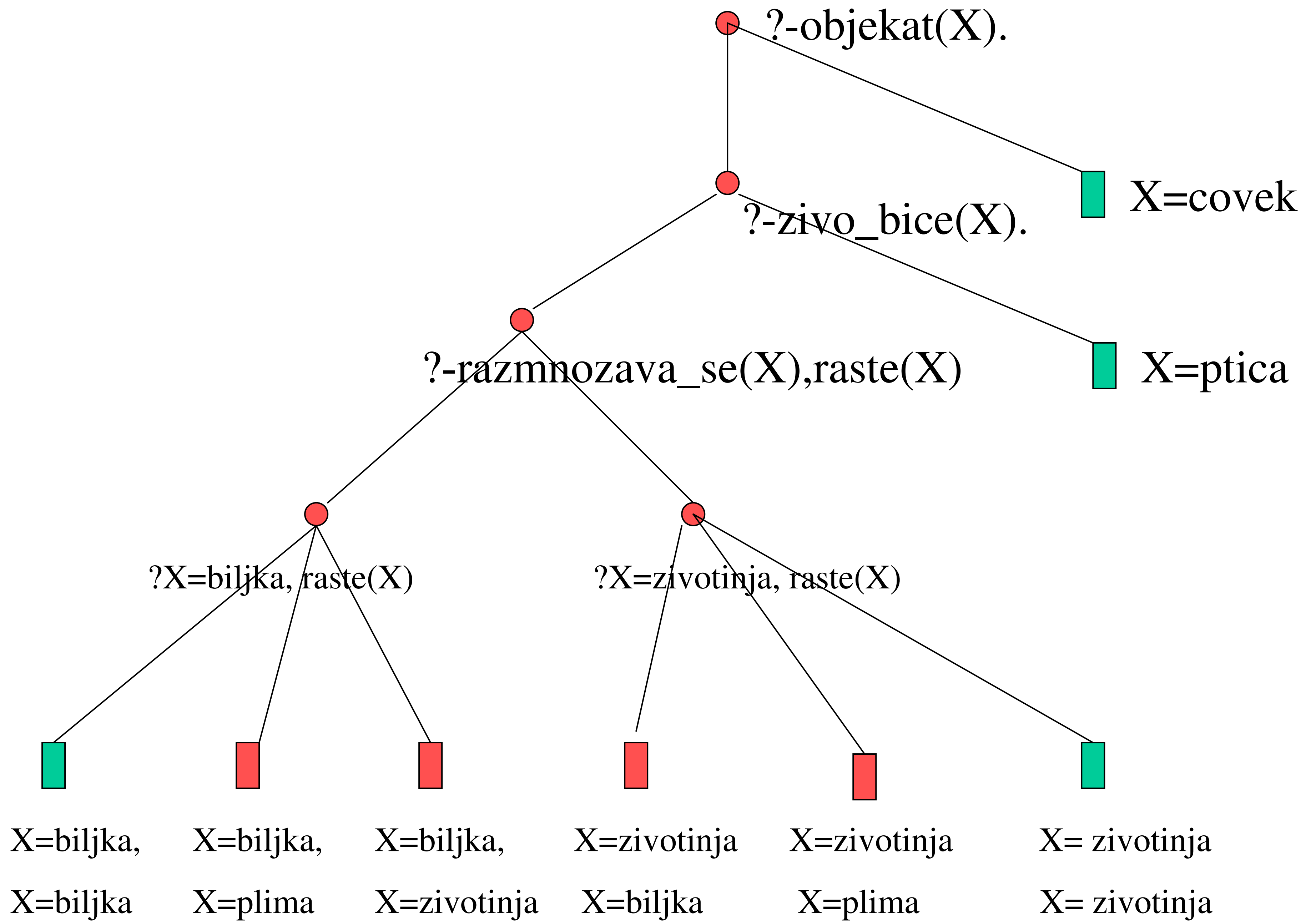
…

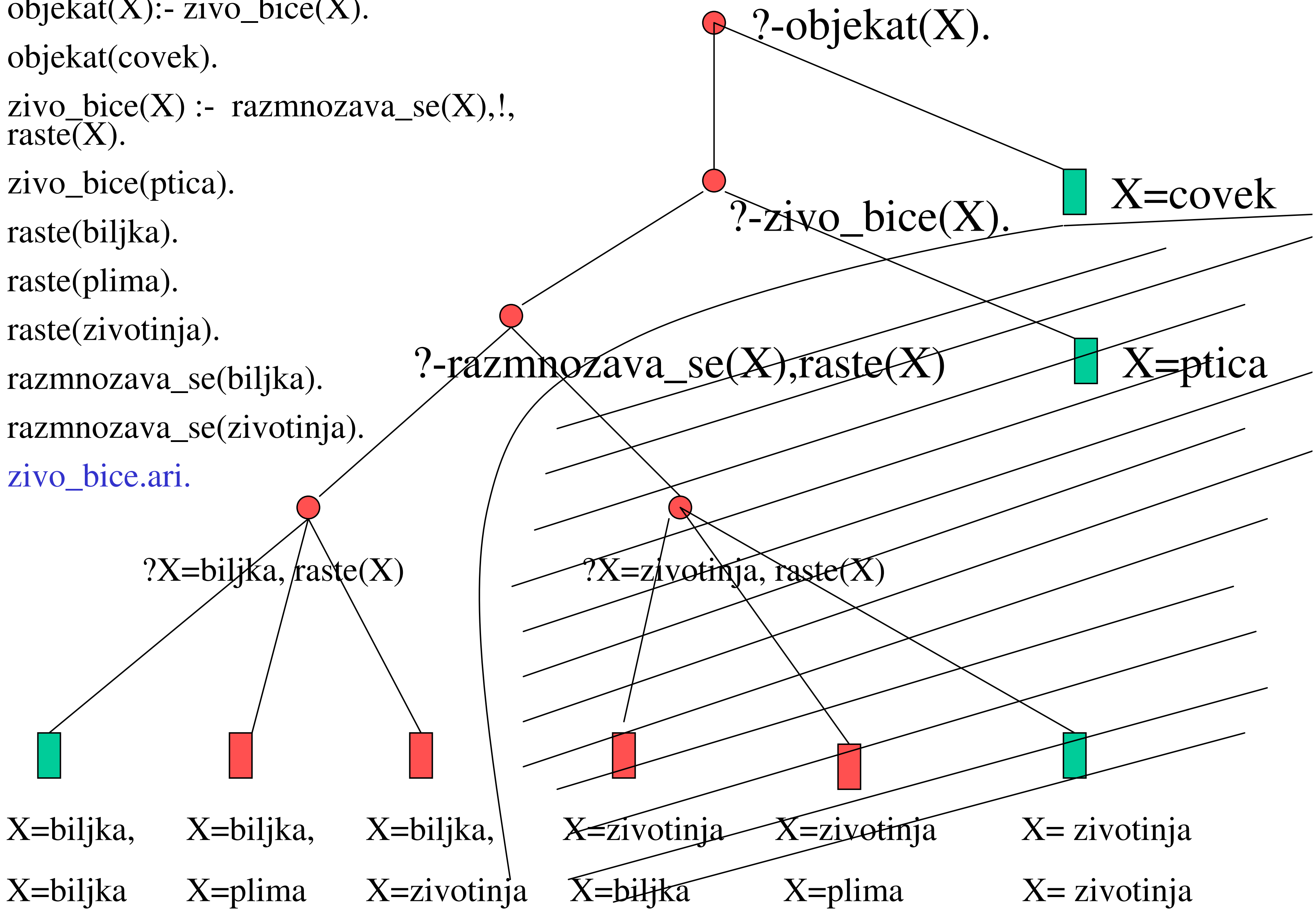
Prilikom traženja rešenja, prvo tvrđenje (označeno sa *) se razmatra.

5.2. Efekti prikazani na stablu pretraživanja

Primer:

```
objekat(X):- zivo_bice(X).

objekat(covek).

zivo_bice(X) :-  razmnozava_se(X), raste(X).

zivo_bice(ptica).

raste(biljka).

raste(plima).

raste(zivotinja).

razmnozava_se(biljka).

razmnozava_se(zivotinja).
```


?-objekat(X).
X=covek
?-zivo_bice(X).
X=ptica
?-razmnozava_se(X),raste(X)
?X=biljka, raste(X)
?X=zivotinja, raste(X)
X=biljka,
X=biljka
X=biljka,
X=plima
X=biljka,
X=zivotinja
X=zivotinja
X=biljka
X=zivotinja
X=plima
X= zivotinja
X= zivotinja



objekat(X):- zivo_bice(X).
objekat(covek).
zivo_bice(X) :- razmnozava_se(X),!,
raste(X).
zivo_bice(ptica).
raste(biljka).
raste(plima).
raste(zivotinja).
razmnozava_se(biljka).
razmnozava_se(zivotinja).
zivo_bice.ari.
?-objekat(X).
X=covek
?-zivo_bice(X).
X=ptica
?-razmnozava_se(X),raste(X)
?X=biljka, raste(X)
?X=zivotinja, raste(X)
X=biljka,
X=biljka
X=biljka,
X=plima
X=biljka,
X=zivotinja
X=zivotinja
X=biljka
X=zivotinja
X=plima
X= zivotinja
X= zivotinja

5.3.Tipične primene Cut-predikata

1. Kada želimo da saopštimo PROLOG-u: “Nađeno je potrebno rešenje, ne treba dalje tražiti!”

bibilioteka.pl

razgovorCut.pl (izmenjen sa Cut-predikatom)

2. Kada želimo da saopštimo PROLOG-u: “Nađeno je jedinstveno rešenje i ne treba dalje tražiti!”

kolicnik.pl

3.Kada želimo da saopštmo PROLOG-u: “Na pogrešnom si putu, završiti pokušaj zadovoljenja cilja!”. Koristi se u kombinaciji sa fail-predikatom. Može samo da se koristi za proverne svrhe, ne i za generisanje rešenja.

inostrani.ari

5.4. Problemi sa Cut-predikatom?

Pogrešna upotreba Cut-predikata najčešći je uzrok grešaka u PROLOG-programima. Cut-predikat ponekad doprinosi da PROLOG-program daje kontradiktorne odgovore.

Primer1: **Zaposlen.pl**

Primer2:

```
spoj([],X,X) :- !.
spoj([G|R],X,[G|Y]) :- spoj(R,X,Y).
```

Upiti:

?-spoj([a,b,c],[d,e],X).

?- spoj([a,b,c],X,Y).

daju korektna rešenja.

Upit:

?- spoj(X,Y,[a,b,c]).

X=[], Y=[a,b,c];

No

daje samo jedno rešenje.

Primer3:

```
broj_roditelja(adam, 0) :-!.
broj_roditelja(eva,0) :- !.
broj_roditelja(X,2).
```

```
?-broj_roditelja(eva,X).
X=0;
no
```

```
?- broj_roditelja(eva, 2).
yes
```

Kako se razrešava problem?

```
broj_roditelja(adam, N) :-!, N=0.
broj_roditelja(eva, N) :- !, N=0.
broj_roditelja(X,2).
```

Ili:

```
broj_roditelja(adam, 0).
broj_roditelja(eva, 0).
broj_roditelja(X,2) :-  X \= adam, X \= eva.
```

5.5. Zeleni i crveni Cut-predikat

Ako se Cut-predikat upotrebljava tako da ne narušava deklarativno svojstvo predikata (već samo pojačava njegova deterministička svojstva i ubrzava proces izračunavanja), onda se naziva zeleni Cut- predikat. U suprotnom, reč je o crvenom Cut-predikatu.

Primer:Definišimo predikat: max(X,Y,Maximum):

max(X,Y, X) :- X >= Y. **(1)**

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

Radi ubrzanja postupka traženja rešenja, nameće se ideja korišćenja Cut-predikata:

max(X, Y, X) :- X >= Y,!. **(2)**

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

Sada se nameće ideja o daljem uprošćenju programa:

max(X, Y, X) :- X >= Y,!. **(3)**

max(X, Y, Y).

U primeru (2) ne narušavaju se deklarativna svojstva predikata, tj. tu je reč o zelenom Cut-predikatu (Cut-predikat ističe determinizam i optimizuje proces nalaženja rešenja).

Nasuprot tome, u primeru (3) narušavaju se deklarativna svojstva predikata. Cut-predikat je upotrebljen za izostavljanje uslova. Promenjeno je dekarativno svojstvo i glasi: za svako X i Y, Y je maksimalno. Ovde je reč o crvenom Cut-predikatu.

Po pravilu, crveni Cut-predikat izaziva greške (dovodi do kontradiktornih rešenja).

Ako primenimo program (3) i postavimo sledeći upit, dobijamo:

?- max(5,4,4).

yes

Odgovor je očigledno nekorektan i nije teško uočiti kako se dolazi do ovog odgovora.

5.6. Kako (i da li) izbegavati Cut-predikat?

Da li se može izbeći Cut-predikat?

Primer:

```
zbir(1,1) :- !.
zbir(N, R) :- N1 is N-1, zbir(N1, R1), R is R1+N.
```

može se zameniti sa:

```
zbir(1,1).
zbir(N, R) :- not(N=1), N1 is N-1, zbir(N1, R1), R is R1+N.
```

ili sa:

```
zbir(1,1).
zbir(N, R) :- N \= 1, N1 is N-1, zbir(N1, R1), R is R1+N.
```

Opšte pravilo: Cut-predikat se može izbeći korišćenjem NOT-operatora ili operatora poređenja.

Prethodni primer se može korigovati (ne javlja se greška za negativan broj, ali rezultat nije korektan za negativan broj) na sledeći način:

zbir(N,1) :- N=<1,!.

zbir(N, R) :- N1 is N-1, zbir(N1, R1), R is R1+N.

odnosno:

zbir(N,1) :- N=<1.

zbir(N, R) :- N>1, N1 is N-1, zbir(N1, R1), R is R1+N.

Ako imamao:

A :- B, !, C.

A :- D.

možemo izvršiti zamenu:

A :- B, C.

A :-not (B), D.

Ako je baza sa tvrđenjuma B velika, njeno pregledanje se tada vrši 2 puta!

5.7. Not-predikat

Not-predikat se može koristiti i kao operator.

not(X) je saglasan sa bazom ako X nije saglasan sa bazom (uspeva, ako ne uspeva X).

Ova definicija nije potpuno u skladu sa matematičkom (što ponekad stvara problema).

?- not(covek(petar)).

Odgovor na ovaj upit je yes ako ne postoji činjenica covek(petar).

Pretpostavka je da je svet zatvoren i da je sve sadržano u bazi znanja. (U stvarnosti, svet nije zatvoren. Ako ne raspolažemo činjenicom da je petar čovek, ne znači da petar nije čovek.)

```
vozac(janko).
vozac(marko).
prodavac(petar):-fail.
```

```
?- not (vozac(X).
  false
```

?- not(vozac(janko)).

false

?- not(prodavac(X)).

true

Teskoće sa not-predikatom:

vozac(janko).

vozac(marko).

vozac(petar).

pije(janko).

pije(marko).

dobar_vozac(X) :- not(pije(X)), vozac(X).

?-dobar_vozac(X).

no

?-dobar_vozac(petar).

yes

Kako se ovo može razrešiti? Ako napišemo:

dobar_vozac1(X) :- vozac(X), not(pije(X)).

?-dobar_vozac1(X).

X=petar;

no

?-dobar_vozac1(petar).

yes

Pravilo: not-predikat se bez problema koristi ako se operiše sa konkretizovanim promenljivm. Ukoliko pravila treba koristiti u generativnom smislu (za generisanje rešenja), a u njima se javlja not-predikat, komponenta sa not-predikatom treba da stoji na kraju.