Prolog

Tag:veza,putanja,lista

(1)Neka je zadan usmjereni graf kao na slici:

Implementirajte:

1. (2 boda) Predikat veza/2 koji ce uspjevati ako postoji veza od cvora zadanog u prvom parametru do cvora zadanog u drugom parametru.

Primjeri korištenja:
|?- v e za (a , X) . X
= b ;

X = c ;

```
gr(a,b).
gr(a,c).
gr(b,c).
gr(b,d).
gr(c,e).
gr(c,f).
gr(d,g).
gr(e,d).
gr(e,g).
gr(f,h).
gr(g,h).

veza(X,Y):-gr(X,Y).
```

2. (2 boda) Predikat putanja/2 koji ce provjeravati je li postoji usmjerena putanja izmedu _cvorova.

Primjeri korištenja:

|?-p u t a n j a (d , X) .
X = g ;

X = h ;
no

```
putanja(X,Y):-veza(X,Y).
putanja(X,Y):-veza(X,Z),putanja(Z,Y).
```

3. (3 boda) Predikat dostupno/2 koji ce vracati listu onih cvorova koji su dostupni iz zadanog cvora.

Primjeri koristenja |?-
dostupno (c , L) . L =
[d , e , f , g , h] ; No.

```
dostupno(X,L):-setof(C,putanja(
        X,C),L).
```

setof (varijabla, cilj, lista) – pronalazi sve vrijednosti koje zadovoljavaju cilj i skuplja ih u listu. Nema ponavljanja i sortirana je. (findall – prazna lista, bagof –ako [], cilj ne uspjeva).

4. (3 boda) Predikat put od do/3 koji _ce za zadane _cvorove u listi vratiti putanju izmedu njih, ako ona postoji.

Radi mogucnoti beskonacne petlje, tablirajte predikat direktivom:
:- t a b l e put od do / 3 .

Primjeri koristenja

|?- put od do (a , d , P) .
P = [a , c , e , d] ;

P = [a , b , d] ;
P = [a , b , c , e , d] ;

```
put_od_do(A,A,[A]).
put_od_do(A,B,[A,B]):-veza(A,B).

put_od_do(A,B,[A|Put]):-veza(A,C),put_od_do(C,B,Put).
```

2 KOLOKVIJ

Zadatak 1) *Zadana je XML datoteka na adresi: http://arka.foi.hr/~mschatten/lp/vozila.xml Bez lokalnog pohranjivanja XML datoteke, implementirajte predikat vozila_do_godine/2 koji*

će za zadanu godinu u prvom argumentu uz drugi argument vezati listu čiji elementi imaju sljedeću strukturu:

vozilo(Marka, Boja)
I to samo onih vozila koja imaju godinu proizvodnje manju ili jednaku od zadane.

Pri implementaciji smiju se koristiti moduli XSB-a. Za pretvorbu znakovnog niza u broj, možete se koristiti predikatom atom_to_term/2.

Primjeri korištenja:
| ?- vozila_do_godine(2001, L).

L = [vozilo(mercedes,bijela),vozilo(opel,plava),vozilo(skoda,plava)];
no

| ?- vozila_do_godine(2000, L).

L = [vozilo(opel,plava),vozilo(skoda,plava)];
no

……..
/*

Implementirajte infiksne operatore '+' i 'je' koji će omogućiti spajanje proizvoljnih atomarnih elemenata u listu. Pri implementaciji se smiju koristiti ugrađeni predikati.

Primjeri korištenja:

| ?- a + 1 je X.
X = [a,1];

no

| ?- a + b + d + 4 je Y.
Y = [a,b,d,4];

no

| ?- a + f(1) + 3 je Z.
No.

*/

```
:- op( 500, xfy, + ). :-
op( 600, xfx, je ).

X + Y je [ X, Y ]
  :-atomic( X ),
  atomic( Y ).



X + Y je [ X | Z ]
  :-Y je Z,
  compound( Y ),
  atomic( X ).
```

/*tag:usporedba, uspoređivanje listi,duža

Bez korištenja ugrađenih predikata implementirajte predikat veca/3 koji će usporediti dvije liste (prva dva argumenta) te uz treći argument vezati vrijednost 'istina', ako je prva lista duža od druge, odnosno vrijednost 'laz' u suprotnom.

Primjeri korištenja:

```
| ?- veca( [a,b,c], [1,2], X
).

X = istina;
no

| ?- veca( [a,b], [1,2,3], X ).
X = laz;

no

| ?- veca( [], [], X ).
X = laz;

no
*/


veca( [ _ | _ ], [], istina ).
veca( [], _, laz ).

veca( [ _ | R1 ], [ _ | R2 ], X )
  :-veca( R1, R2, X ).

/*tag:setof, lista
Neka je zadana baza činjenica kao što slijedi:


osoba( ivek, 1990 ).
osoba( joza, 1991 ).
osoba( bara, 1989 ).
osoba( stef, 1992 ).
osoba( stefica, 1994 ).
```

Implementirajte predikat godine/1 koji će uz svoj argument vezati listu osoba i odgovarajućih starosti oblika osoba(ime, starost).

Primjeri korištenja:

```
| ?- godine(L).
```

L = [osoba(bara,23),osoba(ivek,22),osoba(joza,21),osoba(stef,20),osoba(stefica,18)]

yes

*/

```
osoba( ivek, 1990 ).
osoba( joza, 1991 ).
osoba( bara, 1989 ).
osoba( stef, 1992 ).
osoba( stefica, 1994 ).


starost( X, Y )
  :-osoba( X, Z ),
  Y is 2012 - Z.

godine( L ) :-
  setof( osoba(X,Y), starost(X,Y), L ).
```

/*

Bez korištenja ugrađenih operatora implemetirajte predikat zadnji/2 koji će za prvi argument primati listu, a uz drugi vezati zadnji element te liste umanjen za jedan.

Primjer korištenja

| ?- zadnji([4,2,3,1,3], X).

X = 2;

*/

```
zadnji( [ X ], Y )
  :-Y is X - 1.

zadnji( [ _ | R ], X )
  :-zadnji( R, X ).
```

/*

Implementirajte infiksne operatore + i je koji omogućiti konkatenaciju (spajanje) lista. Dopušteno je korištenje ugrađenih predikata, npr. putem direktive:

```
?- import append/3 from lists.

 Primjeri korištenja:

| ?- [1,2,3] + [4,5,6] je L.

L = [1,2,3,4,5,6];

no
| ?- [1,2] + [3,4] + [5,6] + [7,8] je L.

L = [1,2,3,4,5,6,7,8];

 */



:- op( 500, xfy, + ). :-
op( 600, xfx, je ).

?- import append/3 from lists.
?- import is_list/1 from lists.

X + Y je Z :-is_list(
  Y ), append( X, Y,
  Z ).

X + Y je Z :-

  not( is_list( Y ) ),
  Y je T,

  X + T je Z.

 /*tag:graf, putanja
  Neka je zadana baza činjenica kao što slijedi (jednostavni usmjereni graf):


 put(a,   b).
 put(b,   c).
 put(b,   d).
 put(c,   e).
 put(d,   e).
 put(e, f).
```

Bez korištenja ugrađenih predikata implementirajte predikat putanja/3 koji će biti zadovoljen ako su prva dva argumenta čvorovi, a treći lista koja predstavlja putanju između ta dva čvora.

Primjeri korištenja:

| ?- putanja(a, f, P).
P = [a,b,c,e,f];

P = [a,b,d,e,f];

no

| ?- putanja(b, C, P).
C = c

P = [b,c];
C = d

P = [b,d];
C = e

P = [b,c,e];
C = f

P = [b,c,e,f];
C = e

P = [b,d,e];
C = f
P = [b,d,e,f];

no

*/

```
put(a,  b).
put(b,  c).
put(b,  d).
put(c,  e).
put(d,  e).
put(e, f).

putanja(X,Y,[X,Y])
 :-put(X,Y).
```

```
 putanja(X,Y,[X|R])
   :-put(X,Z),
   putanja(Z,Y,R).
```
%6)

%Bez korištenja ugrađenih predikata implementirajte predikat zadnji/2 koji će za prvi argument primati listu,a za drugi vezati zadnji element te liste umanjen za jedan

%pr
%zadnji(4,2,3,1,3],X).
%X=2

definiramo listu i povratnu vrijednost koja je jednaka zadnjem elementu
zadnji([X],Y):-Y is X-1.

element tijela liste koji je x dobivamo rekurzivno.
zadnji([_|R],X):-zadnji(R,X).

%može i ovako
zadnji([W|R],X):-zadnji(R,X).

……..

```
%  1) Implementrajte predikate pocetak/1 (početni čvor), kraj/1 (završni čvor),
%     boja/2 (boja čvora) i put/2 (veza izmedu dva čvora). (2 boda)
|?- poc e t ak ( P ) , k r a j ( K ) .


pocetak(1).
kraj(11).
boja(2, crna).
boja(6, crna).
boja(9, crna).

boja(_X, bijela)
    :-not(boja(_X,
    crna)).


put(1,  2).
put(1,  3).
put(2,  3).
put(3,  4).
put(3,  7).
put(4,  5).
put(4, 6).
```

```
put(5, 8). put(6,
7).   put(6,   8).
put(7,          9).
put(8,10).
put(9,10).
put(9,11).
put(10, 11). % na grafu nije oznacena, ali u rezultatima se pojavljuje

%  2) Implementirajte predikat bijeli_put/2 koji će za dva čvora provjeriti jesu
%     li oba bijela i povezana putem. (2 boda)
|?-b i j e l i p u t ( X, Y ) .


bijeli_put(X, Y)
   :-put(X, Y), boja(X,
   bijela), boja(Y,
   bijela).



%  3) Implementirajte predikat bijela_putanja/2 koji će provjeriti postoji li
%     izmedu dva čvora putanja u kojoj su svi među-čvorovi bijeli. (3 boda)
|?- b i j e l a p u t a n j a ( 1 , X ) .


     bijela_putanja(X, Y)
       :-bijeli_put(X, Y).
     bijela_putanja(X, Y)
       :-bijeli_put(X, Z),

   bijela_putanja(Z, Y).


%  4) Implementirajte predikat bijela_putanja/3 koja će se ponašati isto kao i
%     predikat u predhodnom zadatku uz nadopunu da će u trećem argumentu
%     prikupiti listu elemenata koji su na tom putu. (3 boda)
|¨?- b i j e l a p u t a n j a ( 1 , 11 , P ) .


bijela_putanja(X, X, [ X ])
   :-boja(X, bijela).

bijela_putanja(X, Y, [ X, Y ])
   :-bijeli_put(X, Y).

bijela_putanja(X, Y, [ X | Putanja ])
   :-bijeli_put(X, Z),
```

bijela_putanja(Z, Y, Putanja).

29.06.2015.

1. (2 boda) Zadan je xml dokument na sljedecoj adresi:
http://arka.foi.hr/~mschatten/lp/djelovi.xml

Potrebno je implementirati predikat dio/2 koji iz zadanog dokumenta izvlaci sifru i naziv dijela. Pri tome je dopusteno koristiti ugradene predikate i tabliranje.

Primjeri koristenja: j ?-
d i o (S , N) . S = 1

N = p r o p e l e r ; S =
2

N = k u c i s t e ; S =
3

N = b a t e r i j a ; S =
4

N = p r o c e s o r ; S =
5

N = mat icna p l o c a ; S =
6

N = b e z i c n i k o n t r o l e r ; S =
7

N = mi k r o c i p ; S =
8

N = krak ; S =
9

N = quadcopt e r ; no

j ?- d i o (S , b a t e r i j a) . S =
3 ;

No
:- auto_table.

:- import parse_xpath/4 from xpath.

```
:- import load_xml_structure/3 from sgml. :-
import member/2 from lists.

:- import ith/3 from basics. :-
import str_cat from string.

parse( X, R ) :-

      parse_xpath( file( '/home/markus/Dropbox/myCourses/Logicko programiranje
(2014)/pismeni/djelovi.xml' ), X, R, [] ), !.


dio( Sifra, Naziv ) :-parse(
   '//dio/sifra', S ), parse(
   '//dio/naziv', N ),

   load_xml_structure(  string(  S  ),  SS,  _  ),
   load_xml_structure(  string(  N  ),  SN,  _  ),
   member( ES, SS ),

   ith( I, SS, ES ), ith(
   I, SN, EN ),

   get_element( ES, Sifra ),
   get_element( EN, Naziv ).
```

2. (2 boda) Potrebno je implementirati predikat ima/1 koji uspjeva ukoliko zadani dio ima poddijelova.

Primjeri koristenja: j ?-
ima (krak) . no

j ?- ima (quadcopt e r) . y e
s

j ?- ima (p r o c e s o r) .
no

```
komponente( Dio, Kom ) :-
   findall( K, komponenta( Dio, K ), Kom ).

nema( Dio ) :-
```

```
    str_cat( '//dio[ naziv="', Dio, I1 ),
    str_cat( I1, '" ]/sastav', XPath ),
    parse( XPath, K ),

    load_xml_structure( string( K ), SK, _ ),
    member( EK, SK ),

    get_element( EK, '0' ).

ima( Dio ) :-
    not( nema( Dio ) ).
```

3. (4 boda) Potrebno je implementirati predikat komponenta/2 koji ce za zadani dio izlistati sve njegove komponente, ako takve postoje. Ako ne postoje, predikat ne uspijeva.

Primjeri koristenja:

```
j ?- komponenta ( quadcopt e r , K ) . K
= p r o c e s o r ;

K = maticna p l o c a ;

K = b e z i c n i k o n t r o l e r ; K
= krak ;

K = mi k r o c i p ; K
= b a t e r i j a ; K =
k u c i s t e ; K = p r
o p e l e r ; no

j ?- komponenta ( krak , K ) .
no
```

```
komponenta( Dio, Kom )
    :-ima( Dio ),

    str_cat( '//dio[ naziv="', Dio, I1 ),

    str_cat( I1, '" ]/sastav/komponenta', XPath ),
    parse( XPath, K ),
```

```
    load_xml_structure( string( K ), SK, _ ),
    member( EK, SK ),

    get_element( EK, Koma ),
    dio( Koma, Kom ).
```

4. (2 boda) Potrebno je implementirati predikat komponente/2 koji _ce za zadani dio vratiti listu svih njegovih

komponenti. Ako dio nema komponenti, predikat vra_ca praznu listu.
Primjeri kori_stenja:

```
j ?- komponente ( ' mi k roc ip ' , K ) .

K = [ b e z i c n i k o n t r o l e r , mat icna ploca , p r o c e s o r ] ;
no

j ?- komponente ( krak , K ) .
K = [ ] ;

no
2

komponenta( Dio, Kom )
    :-ima( Dio ),

    komponenta( Dio, Kom1 ),
    komponenta( Kom1, Kom ).

get_element( element( _, _, [ X ] ), X ).
```

FLORA

Tag:moduli

Neka je zadan modul s.flr koji daje interpretaciju za odredeni svijet (relacija I) te definiciju mogucnosnih relacija za agente A i B (relacija k).

I (p1) . I
(p2) .

A[k->s1 , k->s2] .
B[k->s2] .

Logicki naziv modula u koji ce se ucitati modul s.flr predstavljat ce naziv svijeta u kojem on vrijedi. Konkretno, ako se modul ucita u logicki modul s1, to znaci da u svijetu s1 vrijede propozicije p1 i p2, da agent A u svijetu s1 percipira svijetove s1 i s2, a agent B u svijetu s1 percipira svijet s2. Vas zadatak odnosi se na implementaciju logickog programa za rezoniranje u Kripkeovim strukturama.

1. (2 boda) Potrebno je implementirati modul agenti.flr te pri pokretanju modula dva puta ucitati modul s.flr jednom u logicki modul s1, a drugi put u logicki modul s2.

Primjer koristenja:
f l o r a 2 ?- [a g e n t i] .
. . . u c i t a v a n j e modula . . .
Yes

f l o r a 2 ?- A[k->? v i d i]@?u .
? v i d i = s1

?u = s1

Učitavanje modula s u s1 i s2 (oni nisu nigdje definirani).
?- [s>>s1].

?- [s>>s2].

2. (2 boda) Potrebno je a_zurirati bazu znanja tako da se u logicki modul s1 doda cinjenica: B[k->s1] . te obrise cinjenica (iz istog modula s1):

I (p1) .

Oba azuriranja treba provesti pri pokretanju glavnog modula agenti.r
Primjeri koristenja:

1
f l o r a 2 ?- [a g e n t i] .
. . . u c i t a v a n j e modula . . .
Yes
f l o r a 2 ?- B[k->? v i d i]@?u .

? v i d i = s1
?u = s1

? v i d i = s2
?u = s1

? v i d i = s2
?u = s2

?- insert { B[k->s1]@s1 }.
?- delete { I(p1)@s1 }.

3. (3 boda) Potrebno je implementirati predikat percipira(?agent, ?svijet, ?svijetovi) koji ce za zadanog agenta i zadani svijet uz varijablu ?svijetovi vezati listu svih svijetova koje agent percipira putem svoje mogucnosne relacije k u zadanom svijetu.

Primjeri koristenja:

f l o r a 2 ?- p e r c i p i r a (A, s1 , ? s) .
? s = [s1 , s2]

f l o r a 2 ?- p e r c i p i r a (B, s2 , ? s) .
? s = [s2]

```
percipira( ?agent, ?svijet, ?svjetovi ) :-
        ?svjetovi = setof { ?_vidi | ?agent[ k->?_vidi ]@?svijet }.
```

4. (3 boda) Potrebno je implementirati predikat zna(?agent, ?propozicija, ?svijet) koji _ce za zadanog agenta,

zadanu propoziciju i zadani svijet provjeriti je li agent zna propoziciju u tom svijetu. (Agent zna propoziciju p

akko ne postoji niti jedan svijet koji agent percipira, a da u njemu p ne vrijedi).
Primjeri kori_stenja:

f l o r a 2 ? zna (A, p1 , s1) .
No

f l o r a 2 ? zna (A, p2 , s2) . El
aps ed time 0.0000 s e c onds
Yes

```
vrijedi_u_svjetovima( ?_, [] ). vrijedi_u_svjetovima(
?propozicija, [ ?svijet | ?r ] ) :-
```

```
        I( ?propozicija )@?svijet,
        vrijedi_u_svjetovima( ?propozicija, ?r
        ).


zna( ?agent, ?propozicija, ?svijet )
        :-percipira( ?agent, ?svijet, ?_svjetovi
        ),

        vrijedi_u_svjetovima( ?propozicija, ?_svjetovi ).
```

PAZI: dok radiš sa transakcijama, upit mora također sadržavati %

Zadan je modul kuca.flr koji definira objekte koji predstavljaju sobe u kuci te trenutnu poziciju igrača.

```
hodnik[vrata->'dnevni boravak',vrata->'spavaca
soba']. 'spavaca soba'[vrata->kupaona].

'dnevni
boravak'[vrata->kuhinja].
kuhinja[sadrzi->pizza].
igrac[pozicija->hodnik].
```

//1. zad. Potrebno je implementirati pravilo po kojem ce vrijediti da ako iz jedne sobe postoje vrata u drugu, tada postoje i vrata iz druge u prvu.

Znači, pravilo nije upit, nego se piše u .flr datoteku.

```
?_prostorija[vrata->?soba]:-?soba[vrata->?_prostorija]
. ili

?_p1[vrata->?_p2]:-?_p2[vrata->?_p1].
```

//2. zad. Potrebno je implementirati predikat put/2 koji će provjeravati je li između dvije sobe postoji put.

```
?put(?_prostorija,?soba):-?_prostorija[vrata->?soba].
?put(?_prostorija,?soba):-?_prostorija[vrata->?y],?put(?y,?soba).
```

//3.zad. potrebno je implementirati metodu idi/1 objekta igrac koja ce provjeriti je li za igraca postoji put u zadanu sobu te promijeniti njegovu poziciju.

```
igrac[idi(?_prostorija)]:-igrac[pozicija->
?_trenutna],?put(?_trenutna,?_prostorija),delete{igrac[pozicija->
?_trenutna]},insert{igrac[pozicija->?_prostorija]}.
```

Transakcijsko:

```
igrac[ %idi(?_s) ]
:-

        igrac[pozicija->?_p], put(?_p,
        ?_s),
        t_delete{igrac[pozicija->?_p]},
        t_insert{igrac[pozicija->?_s]}.
```

//4.zad. Potrebno je implementirati metodu uzmi/1 objekta igrac koja će provjeriti je li se zadana stvar nalazi u trenutnoj sobi

```
igrac[uzmi(?_stvar)]:-?_prostorija[sadrzi->?_stvar],insert{igrac[ima->
?_stvar]},delete{?_prostorija[sadrzi->?_stvar]}.
```

Ili sva 4 na naš način:

```
?_p1[vrata->?_p2] :- ?_p2[vrata -> ?_p1].

put(?s1, ?s2) :- ?s1[vrata->?s2].
put(?s1, ?s2) :- ?s1[vrata->?_s], put(?_s, ?s2).

igrac[ %idi(?_s) ] :-

        igrac[pozicija->?_p], put(?_p, ?_s),
        t_delete{igrac[pozicija->?_p]},
        t_insert{igrac[pozicija->?_s]}.

igrac[%uzmi(?stvar)]:-igrac[
        pozicija->?x],
        ?x[sadrzi->?stvar].
```

Ispit 16.07.2013.

Neka je zadana baza znanja koja opisuje usmjereni graf:

```
a:cvor[brid->{b,c,d,x}].

b:cvor[brid->{c,e}].
c:cvor[brid->{e,d}].
```

```
d:cvor[brid->{f}].
e:cvor[brid->{f}].
f:cvor.
```

// 1. zadatak Implementirajte metodu put/0 klase cvor koja vraca sve cvorove kojima je trenutni cvor povezan usmjerenom putanjom.

```
?c1:cvor[ put->?c2 ] :- ?c1[ brid->?c2 ].
?c1:cvor[ put->?c2 ] :-

                ?c1[ brid->?cc ],?cc[ put->?c2 ].
```

// 2. zadatak Implementirajte metodu duljina/1 klase cvor koja za primljeni cvor vraca sve udaljenosti od trenutnog cvora.

```
?c1:cvor[   duljina(?c2)->1   ]
        :-?c1[ brid->?c2 ].

?c1:cvor[ duljina(?c2)->?x ] :-
        ?c1[ brid->?c ], ?c[ duljina(?c2)->?y ], ?x is ?y + 1.
```

// 3. zadatak Implementirajte klasu obojeni cvor koja naslijeduje iz klase cvor te ima dodatni atribut boja. Dodajte tri obojena cvora (x, y, z) koji imaju boje zuta, plava i zelena i imaju bridove k cvorovima a, b i c respektivno.

Definiramo da je klasa obojeni_cvor podklasa klase cvor i definiramo unutar te podklase instancu boja. Dolje niže definiramo tri obojana čvora sa svojim bojama i k kojim bridovima teže.

```
obojeni_cvor :: cvor[
        boja=>string
].

x : obojeni_cvor[ boja -> zuta, brid -> a ]. y :
obojeni_cvor[ boja -> plava, brid -> b ].

z : obojeni_cvor[ boja -> zelena, brid -> c ].
```

// 4. zadatak Izmjenite metodu put iz prvog zadatka tako da ona vrijedi samo za neobojene cvorove. Novu petodu nazovite put2/0.

Ovo je upitno, radi samo dio, ne radi za neobojene čvorove. @_prolog pozivamo da bi omogućili komandu write(writeln) (jer kao takve je nema u Flori.

```
?c1[ put2 -> ?c2 ]
        :-?c1[ brid->?c2
        ],

        write(?c2:obojeni_cvor)@_prolog.


/*?c1:cvor[ put2 -> ?c2 ]
        :-?c1[ brid->?cc ],
        ?cc[ put2->?c2 ],

        not( ?c1:obojeni_cvor ),
        not( ?c2:obojeni_cvor ),
        not( ?cc:obojeni_cvor ).*/
```

ISPIT 20.02.2013.

Potrebno je implementirati apstraktni tip podataka rjecnik (asocijativni niz) u Flori-2. Obratite pozornost na to da akcije azuriranja trebaju biti transakcijske, kako bi se izbjegli nezeljeni rezultati (korištenje prefiksa '%' kod metoda objekta i metoda ažuriranja s prefiksom

't ').

1. (2 boda) Implementirajte metodu dodaj/2 koja ce primati dva parametra (kljuc i vrijednost) i dodavati odgovarajuce vrijednosti u rijecnik.

Naredba naf bi zamijenila not, jer not baš ne radi na način da negira, nego više služi za to da se nešto konstatira. Provjeravamo da li već postoji neka vrijednost pod ključem, ako postoji – nećemo je dodati (to nam omogućuje naf). Kljuc mora biti drugaciji (?kljuc) da bi mogli dodavati nove vrijednosti.

```
?r:rjecnik[%dodaj(?kljuc,?vrijednost)]:-
        naf(?r[?kljuc->?_]),
        t_insert{?r[?kljuc->?vrijednost]}
        .
```

Proba: a[%dodaj(1,2)].

2. (2 boda) Implementirajte metodu brisi/1 koja ce primati kljuc rijecnika i obrisati vrijednost pod zadanim kljucem.

```
?r:rjecnik[%brisi(?kljuc)]
        :-t_delete{?r[?kljuc->?_]
        }.
```

Provjera: a[%brisi(2)].
a[?x->?y].

3. (3 boda) Implementirajte metodu azuriraj/2 koja ce primati postojeci kljuci za njega promijeniti trenutnu vrijednost na prosljedenu.

```
?r:rjecnik[%azuriraj(?kljuc,?vrijednost)]
        :-t_delete{?r[?kljuc->?_]},

        t_insert{?r[?kljuc->?vrijednost]}.
```

Provjera: a[%azuriraj(1,3)].
a[?x->?y].

4.(3boda) implementirajte metodu prikazi/1 koja ce prikazati sadrzaj rjecnika.
Proba: [r(1,2)]., a[%dodaj(3,4)], a[%prikazi(?x)].

```
?r:rjecnik[%prikazi(?x)] :-
                ?x=collectset{?z | ?r[?kljuc->?vrijednost], ?z = r(?kljuc, ?vrijednost)}.
```

ROK 11.02.2014.

Prvi zadatak sa učitavanjem xml.

2. (2 boda) Rjesenje prethodnog zadatka pohranite u datoteku dohvat.P te ju koristite kao modul unutar Flora-2 stroja. Potrebno je implementirati klasu rok koja sadrzi atribute kolegij, datum, vrijeme i dvorana.
Primjer koristenja:
f l o r a 2 ?→? x : rok [k o l e g i j →?k , datum→?d , v r i j eme →?v , dvorana→?dv] .

```
?- [dohvat].

load_rokovi :-

        rok( ?id, ?kolegij, ?datum, ?vrijeme, ?dvorana )@_prolog,
        insert { ?id : rok [

                kolegij -> ?kolegij,
                datum -> ?datum,
```

```
            vrijeme -> ?vrijeme,
            dvorana -> ?dvorana

        ] }.
?- load_rokovi.
```

3.Implementirajte predikat rokovi prema dvoranama/2 koji _ce uz svoje parametre vezati dvorane te

odgovaraju_ce liste rokova koji se odr_zavaju u toj dvorani.

```
rokovi_prema_dvoranama( ?dvorana, ?kolegiji ) :-

        ?_ : rok [ dvorana -> ?dvorana ],

        ?kolegiji = setof { ?_id | ?_id : rok [ dvorana -> ?dvorana ] }.
```

4.(2 boda) Implementirajte predikat rokovi prema datumima/3 koji ce uz svoje parametre vezati datume, vremena i odgovarajuce liste rokova koji se odrzavaju u tim terminima. Primjeri koristenja:

```
f l o r a 2 ?- rokovi prema datumima ( ?datum , ? v r i j eme , ? r o k o v i ) .
rokovi_prema_datumima( ?datum, ?vrijeme, ?rokovi ) :-

        ?_ : rok [ datum -> ?datum, vrijeme -> ?vrijeme ],

        ?rokovi = setof { ?_id | ?_id : rok [ datum -> ?datum, vrijeme -> ?vrijeme ] }.
```

Zadana je baza znanja:

```
a : cvor [b r i d −>{b, c, d } ].
b : cvor [b r i d −>{c, e } ].

c : cvor [b r i d −>{e, d } ].
d : cvor [b r i d −>f].

e : cvor [b r i d
−>f]. f : cvor .
```

1.Implementirajte metodu put/0 klase cvor koja vraca sve cvorove kojima je trenutni cvor povezan usmjerenom putanjom.

Primjeri korištenja:

```
flora2 ?− c [ put −>?x ]
. ?x = d
```

?x = e
?x = f

3 solution (s) in 0.0000
seconds Yes

flora2 ?− a [put −>?x]
. ?x = b

?x = c
?x = d
?x = e
?x = f
Rj.:

?c:cvor[put->?x]:-?c[brid->?x].
?c:cvor[put->?x]:-?c[brid->?y],?y[put->?x
]

2.Implementirajte metodu duljina/1 klase cvor koja za primljeni cvor vraca sve udaljenosti od trenutnog cvora.

Primjeri koristenja:

flora2 ?− a [duljina (c)−>?x]
. ?x = 1

?x = 2

2 solution (s) in 0.0040
seconds Yes

flora2 ?− a [duljina (e)−>?x]
. ?x = 2

Rj.:
?c:cvor[duljina(?x)->1]:-?c[brid->?x]
.

?c:cvor[duljina(?x)->?d]:-?c[brid->?y],?y[duljina(?x)->?d1],?d is ?d1+1.

3.Implementirajte klasu obojeni cvor koja naslijeduje iz klase cvor te ima dodatni atribut boja. Dodajte tri obojena cvora (x, y, z) koji imaju boje zuta, plava i zelena i imaju bridove k cvorovima a, b i c respektivno.
Primjeri koristenja:

flora2 ?− ?C : obojeni_cvor [boja−>?boja ,brid−>? brid]
. ?C = x

? boja = zuta
? brid = a

?C = y
? boja = plava
? brid = b
?C = z
? boja = zelena
? brid = c

Rj.:
obojeni_cvor::cvor.

x:obojeni_cvor[boja->zuta, brid->a].
y:obojeni_cvor[boja->zelena, brid->b].
z:obojeni_cvor[boja->zelena, brid->c].

4.Izmjenite metodu put iz prvog zadatka tako da ona vrijedi samo za neobojene cvorove. Novu petodu nazovite put2/0.

Primjeri koristenja:
flora2 ?− x [put −>?y] .
?y = a

?y = b
?y = c
?y = d
?y = e
?y = f

6 solution (s) in 0.0000 seconds
Yes

flora2 ?− x [put2 −>?y] .
No

Rj.:
?c:cvor[put2->?x]:-?c[brid->?x],not(?c:obojeni_cvor),not(?x:obojeni_cvor).
?c:cvor[put2->?x]:-?c[brid->?y],?y[put2->?x],not(?c:obojeni_cvor),not(?x:o
bojeni_cvor),not(?y:obojeni_cvor).

……

Neka je zadana baza znanja:
zaposlenik::osoba.
kupac::osoba.
menadzer::zaposlenik.
direktor::menadzer.
stefica:kupac.
ivek:zaposlenik.
joza:menadzer.
bara:direktor.

ivek[placa->100].
joza[placa->200].
bara[placa->300].

1) *Implementirajte metodu ukupne_place koja će za zadanu klasu vratiti zbroj svih plaća objekata u toj klasi.*

Primjeri korištenja:

flora2 ?- osoba[ukupne_place->?p]. ?p =
600

1 solution(s) in 0.0100 seconds Yes

flora2 ?- zaposlenik[ukupne_place->?p]. ?p =
600

1 solution(s) in 0.0000 seconds Yes

flora2 ?- menadzer[ukupne_place->?p]. ?p =
500

1 solution(s) in 0.0000 seconds Yes

flora2 ?- direktor[ukupne_place->?p]. ?p =
300

1 solution(s) in 0.0000 seconds Yes

Rj.:

```
osoba[ukupne_place => integer].
zaposlenik::osoba[placa => integer].
osoba[uk -> ?u] :-
?u= sum{? _ g | ? _ :?osoba[placa->?_g]}.
```

Iliti moje (bez definicije u bazi):

```
osoba[uk->?p]:-?p=sum{?g|?_:osoba[placa->?g]}.
```

2) *Implementirajte meta-predikat (naziv/funktor predikata je varijabilan) koji će omogućiti vraćanje liste objekata koji su u klasi zadanom nazivom predikata.*

Primjeri korištenja: flora2 ?-
osoba(?x).

?x = [bara, ivek, joza, stefica] 1 solution(s) in 0.0100 seconds Yes

flora2 ?- kupac(?x). ?x =
[stefica]

1 solution(s) in 0.0000 seconds Yes

flora2 ?- zaposlenik(?x). ?x =
[bara, ivek, joza]

1 solution(s) in 0.0000 seconds Yes

flora2 ?- menadzer(?x). ?x =
[bara, joza]

1 solution(s) in 0.0000 seconds Yes

flora2 ?- direktor(?x). ?x =
[bara]

1 solution(s) in 0.0000 seconds Yes

Rj.

```
?p(?x) :- ?x = collectset{?_g| ?_g:?p}.
```

……..

Neka je zadana sljedeća baza znanja u F-logici:
Joza:osoba[roditelj->{ Stef, Bara }, spol->musko].
Stef:osoba[roditelj->{ Stefica, Ivek }, spol->musko].
Bara:osoba[roditelj->{ Slavek, Marica }, spol->zensko].
Ivek:osoba[spol->musko].
Stefica:osoba[spol->zensko].

Slavek:osoba[spol->musko].
Marica:osoba[spol->zensko].

Implementirajte metode baka i predak za klasu osoba. Primjeri korištenja:
flora2 ?- ?x[baka->?y].

?x = Joza ?y =
Marica ?x = Joza
?y = Stefica

2 solution(s) in 0.0000 seconds Yes

```
?x[baka->?y]:-?x:osoba[roditelj->?z]
        , ?z:osoba[roditelj->?y],
        ?y:osoba[spol->zensko].
```

flora2 ?- Joza[predak->?y]. ?y =
Bara

?y = Ivek ?y =
Marica ?y =
Slavek ?y = Stef

?y = Stefica

6 solution(s) in 0.0000 seconds Yes

flora2 ?-

```
?ja[predak->?oni]:-?ja:osoba[roditelj->?oni].

?ja[predak->?oni]:-?ja:osoba[roditelj->?o],
                   ?o:osoba[roditelj->?oni].
```

Rj.:

osoba [roditelj = > osoba, spol = > string, baka = > osoba, predak = > osoba]. ?osoba[baka -> ?baka] :- ?osoba[roditelj->?roditelj], ?roditelj[roditelj->?baka], ?baka[spol->zensko].
?osoba[predak -> ?predak] :- ?osoba[roditelj->?predak].
?osoba[predak -> ?predak] :- ?osoba[predak -> ?x], ?x[roditelj->?predak].

Implementirajte sljedeći UML dijagram u F-logici:
http://arka.foi.hr/~mschatten/lp/uml_manageri.png

Metoda podređeni prima naziv odjela i vraća listu podređenih datog menadžera.
Primjerice neka je zadana sljedeća baza znanja:

ivek:zaposlenik[ime->Ivan, prezime->Presvetli, nadredjeni->bara, odjel->Marketing].
joza:zaposlenik[ime->Josip, prezime->Prikratki, nadredjeni->bara, odjel->Marketing].
bara:menadzer[ime->Barica, prezime->Jambrek].

Tada će se program ponašati na sljedeći način:
flora2 ?- bara[podredjeni(Marketing) -> ?x].
?x = [ivek, joza]

1 solution(s) in 0.0000 seconds
Yes

Rj.:

zaposlenik [ime => _string, prezime => _string, nadredjeni => menadzer, odjel => _string].
menadzer::zaposlenik[podredjeni(_string)=>zaposlenik].
?menadzer[podredjeni(?odjel)->?podredjeni] :-

?podredjeni = collectset{?_g| ?_g:zaposlenik[nadredjeni->?menadzer, odjel-> ?odjel]}.
ivek:zaposlenik[ime->Ivan, prezime->Presvetli, nadredjeni->bara, odjel->Marketing].
joza:zaposlenik[ime->Josip, prezime->Prikratki, nadredjeni->bara, odjel->Marketing].
bara:menadzer[ime->Barica, prezime->Jambrek].

Zadana je baza znaja o automobilima:
a1:vozilo[boja->plava,godina_proizvodnje->2005].
a2:vozilo[boja->zuta,godina_proizvodnje->1977].
a3:vozilo[boja->zelena,godina_proizvodnje->1980].
a4:vozilo[boja->crvena,godina_proizvodnje->2013].
a5:vozilo[boja->zuta,godina_proizvodnje->2014].
a6:vozilo[boja->plava,godina_proizvodnje->2002].
a7:vozilo[boja->zelena,godina_proizvodnje->1990].
a8:vozilo[boja->crvena,godina_proizvodnje->2003].

1. Potrebno je implementirati atribut je_oldtimer klase vozilo koji pokazuje je li neko vozilo starije od 30godina.

   Primjer korištenja:

   ?x[je_oldtimer]
   . ?x=a2

   ?x=a3.

?vozilo[je_oldtimer(?sada)->?je_oldtimer]:-?vozilo[godina_proizvodnje
->?godina_proizvodnje],

?je_oldtimer \is ?sada-?godina_proizvodnje.
?x[je_oldtimer]:-?x:vozilo[je_oldtimer(2015)->?je_oldtimer],
?je_oldtimer>30.

2.Potrebno je implementirati atribut opis koji se izračunava na temelju atributa boja i to tako da se makne zadnje slovo vrijednosti atributa i doda nastavak o_vozilo. Pri implementaciji se smiju koristiti moduli.
Primjeri korištenja:

?x[opis->?o].
?x=a1
?o=plavo_vozilo
?x=a2
?o=zuto_vozilo
?x=a3
?o=zeleno_vozilo
?x=a4

?o=crveno_vozil
o

..
?x[opis->?o]:-?x:vozilo[boja->?o].
????????

3.Potrebno je implementirati predikat vozila_prema_rasponu_godina/3 koji će za prva 2 argumenta primati godine te uz zadnji argument vezati ona vozila koja su proizvedena u zadanom rasponu.

Primjeri korištenja:
vozila_prema_rasponu_godina(2000,2015,?v).
?v=a1

?v=a4
?v=a5

?v=a6
?v=a8

vozila_prema_rasponu_godina(?x,?y,?v):-?v[godina_proizvodnje->?v1], ?prva \is ?x, ?druga \is ?y, ?v1>?prva, ?v1<?druga.

4.Potrebno implementirati predikat vozila_po_boji/2 koji će agregirati vozila i njihove godine proizvodnje u listu čiji su elementi oblika v(vozilo,godina) i grupirati ih prema boji.

Primjeri korištenja:
vozila_po_boji(plava,?v).
?v=[v(a1,2005),v(a6,2002)
]

vozila_po_boji(?_boja,?v):-?v=collectset{?v[?_boja]|?v:vozilo[boja->?_boja]}.

……..

SVE

PROLOG ISPIT 08.07.2015.
https://www.cpp.edu/~jrfisher/www/prolog_tutorial/contents.html

cesta(a,b,5,130,100). cesta(a,c,10,50,0).
cesta(b,d,5,130,100).
cesta(b,e,5,130,100). cesta(c,e,10,50,0).
cesta(c,f,15,80,0). cesta(d,g,5,130,100).
cesta(e,g,15,80,0). cesta(f,h,15,80,0).
cesta(g,h,10,50,0). cesta(g,i,5,130,100).
cesta(h,i,15,80,0). predikat: cesta (ili
dionica) | ?- dionica(c,X,D,B,C).

http://flora.sourceforge.net/release_notes.html
?c:cvor[duljina(?x)->?d]:-

?c[brid->?x].
?c:cvor[duljina(?x)->?d]:-?c[brid->?y],
?y[duljina(?x)->?d1],

?d is ?d1 + 1.

?obojeni_cvor::cvor.
x:obojeni_cvor[boja->zuta,brid->a].

```
y:obojeni_cvor[boja->plava,brid->b].
c:obojeni_cvor[boja->zelena,brid->c].



?c:cvor [put2 ->
?x]:-?c[brid ->?x],
not(?c:obojeni_cvor),
not(?x:obojeni_cvor).


?c:cvor[put->?x]:-?c[bri
d->?y], ?y[put2->?x],
not(?c:obojeni_cvor),
not(?x:obojeni_cvor),
not(?y:obojeni_cvor).
--------------------------------------------------------------------------



?r:rijecnik[%dodaj(?k,?v)]:-n
ot(?r[?k->?_]).
t_insert{?r[?k->?v]}.
?r:rijecnik[%brisi(?k)]:-t_dele
te{?r[?k->?_]}.


?r:rijecnik[%azuriraj(?k,?v)]:-?r
[%brisi(?k)],
?r[%dodaj(?k,?v)].
?r:rijecnik[$prikazi(?x)]:-

?x=collectset{?z | ?r[?k->?v],?z = r(?k,?v)}.
```

XML učitavanje:

```
:- import parse_xpath/4 from xpath.

:- import load_xml_structure/3 from sgml.
:- import member/2 from lists.

:- import ith/3 from basics.


rok(Id,Kolegij,Datum, Vrijeme,
        Dvorana):-parse_xpath(file('http://arka.foi.hr/~mschatten/lp/rok.xml'),'/rokovi/rok/id',I
        deovi,[]),

      parse_xpath(file('http://arka.foi.hr/~mschatten/lp/rok.xml'),'/rokovi/rok/kolegij',
Kolegiji,[]),
```

```
        parse_xpath(file('http://arka.foi.hr/~mschatten/lp/rok.xml'),'/rokovi/rok/datum',
Datumi,[]),

        parse_xpath(file('http://arka.foi.hr/~mschatten/lp/rok.xml'),'/rokovi/rok/vrijeme',
Vremena,[]),

        parse_xpath(file('http://arka.foi.hr/~mschatten/lp/rok.xml'),'/rokovi/rok/dvorana',
Dvorane,[]),


        get_triple(Ideovi,Kolegiji,Datumi,Vremena,Dvorane,
Id,Kolegij,Datum,Vrijeme,Dvorana).


get_triple(Ideovi,Kolegiji,Datumi,Vremena,Dvorane,
        Id,Kolegij,Datum,Vrijeme,Dvorana):-load_xml_structure(string(Ideovi),
        Id_struct,_), load_xml_structure(string(Kolegiji),Kolegij_struct,_),
        load_xml_structure(string(Datumi),Datum_struct,_),
        load_xml_structure(string(Vremena),Vrijeme_struct,_),
        load_xml_structure(string(Dvorane),Dvorana_struct,_),

member( Id_elem, Id_struct ),
                ith(I, Id_struct, Id_elem),

                ith(I, Kolegij_struct, Kolegij_elem),
                ith(I, Datum_struct, Datum_elem),
                ith(I, Vrijeme_struct, Vrijeme_elem),
                ith(I, Dvorana_struct, Dvorana_elem),


                get_element(Id_elem,Id),
                get_element(Kolegij_elem,Kolegij),

            get_element(Datum_elem,Datum),
                get_element(Vrijeme_elem,Vrijeme),
                get_element(Dvorana_elem,Dvorana).

get_element(element(_,_,[X]),X).
```

Za učitavanje osobe/osoba (ime i prezime) iz XML-a::::::::::::::::::::::::::::::::::::::

```
:- import parse_xpath/4 from xpath.

:- import load_xml_structure/3 from
sgml. :- import member/2 from lists.

:- import ith/3 from basics.


osoba( Ime, Prezime ) :-
```

```
    parse_xpath( file( '/home/natalija/Desktop/pro.xml' ), '/osobe/osoba/ime', Imena, [] ),
    parse_xpath( file( '/home/natalija/Desktop/pro.xml' ), '/osobe/osoba/prezime', Prezimena, []
),
    get_triple( Imena, Prezimena, Ime, Prezime ).


get_triple( Imena, Prezimena, Ime, Prezime )
    :-load_xml_structure( string( Imena ), Imena_struct, _ ),
    load_xml_structure( string( Prezimena ), Prezimena_struct, _
    ), ith( I, Imena_struct, Ime_elem ),

    ith( I, Prezimena_struct, Prezime_elem ),
    get_element( Ime_elem, Ime ),
    get_element( Prezime_elem, Prezime ).

get_element( element( _, _, [ X ] ), X ).
```

:::

::::::::::::::::
----

https://sites.google.com/site/prologsite/prolog-problems/1

http://autopoiesis.foi.hr/wiki.php?name=Logi%C4%8Dko%20programiranje%20-%20FOI&parent=NULL&page=hetinfo

11.2.2014.

1) Zadan je xml dokument na sljede_coj adresi: http://arka.foi.hr/~mschatten/lp/rok.xml Potrebno je implementirati predikat rok/5 koji iz zadanog dokumenta izvla_ci id, kolegij, datum, vrijeme i dvoranu roka. Pri tome je dopu_steno koristiti ugradene predikate.

Za ovo je potrebno koristiti module sgml i xpath.

http://arka.foi.hr/~isvogor/lp/isvogor_lp.pdf

PROLOG:

and	*,*
If	*:-*
Or	*;*
Not	*not*

```
vrijeme(krizevci,ljeto,toplo).
vrijeme(varazdin,zima,hladno
).
vrijeme(australija,zima,vruce)
. toplije_od(G1,G2):-

        vrijeme(G1,vruce,ljeto),
        vrijeme(G2,toplo,ljeto),
        write(G1), write(' je topliji od '), write(G2),nl.
```

Bez korištenja ugrađenih predikata i modula implementirajte predikat zip/3 koji će u prva dva parametra primati dvije liste jednake duljine, te u treći argument vezati listu koja se

sastoji od elemenata oblika par(X1, X2) u kojem su X1 i X2 odgovarajući elementi prve i druge liste respektivno.

Primjeri korištenja:

| ?- zip([1, 2, 3], [a, b, c], L).

L = [par(1,a),par(2,b),par(3,c)];

no
| ?- zip(L1, L2, [par(a, b), par(c, d)]).

L1 = [a,c]
L2 = [b,d];

no

kolokvij(prolog G1) 1)

zip([X|Xs], [Y|Ys], [par(X,Y)|Zs]) :- zip(Xs, Ys, Zs). zip([A], [B], [par(A,B)]).
Implementirajte predikate postavi/1 i povecaj/1 koji će uz pomoć ažuriranja baze znanja omogućiti postavljanje globalne vrijednosti i njezino povećavanje za jedan. Predikat postavi/1 će primati za argument broj koji će postaviti za globalnu vrijednost. Predikat povećaj/1 će uz argument vezati globalnu vrijednost koja se svakim pozivom povećava za jedan.

Primjeri korištenja:
| ?- postavi(3).

yes
| ?- povecaj(X).

X = 4;

no
| ?- povecaj(X).

X = 5;

no
| ?- povecaj(X).

X = 6;

no

| ?- postavi(1).

yes

| ?- povecaj(X).

X = 2;

no
| ?- povecaj(X).

X = 3;

no

2) :- dynamic broj/1. broj(0). postavi(X):-retract(broj(Y)),assert(broj(X)).
povecaj(X):-broj(Y),X is Y+1,postavi(X).

Zadana je XML datoteka na adresi:

http://arka.foi.hr/~mschatten/lp/vozila.xml

Bez lokalnog pohranjivanja XML datoteke, implementirajte predikat vozila_do_godine/2 koji

će za zadanu godinu u prvom argumentu uz drugi argument vezati listu čiji elementi
imaju sljedeću strukturu:

vozilo(Marka, Boja)

I to samo onih vozila koja imaju godinu proizvodnje manju ili jednaku od zadane.

Pri implementaciji smiju se koristiti moduli XSB-a. Za pretvorbu znakovnog niza u broj, možete se koristiti predikatom atom_to_term/2.

Primjeri korištenja:

| ?- vozila_do_godine(2001, L).

L = [vozilo(mercedes,bijela),vozilo(opel,plava),vozilo(skoda,plava)];

no
| ?- vozila_do_godine(2000, L).

L = [vozilo(opel,plava),vozilo(skoda,plava)];

no

-shell

Implementirajte objekt shell u Flori-2 koji ima metodu cmd/1 koja za argument prima shell naredbu te uz izlaz veže ispis te naredbe.

Primjeri korištenja:

flora2 ?- shell[cmd(ls) -> ?sadrzaj].

?sadrzaj = [['k1.P'], ['k1.xwam'], ['K2.flr'], ['k2.P'], ['k2.xwam'], ['kn.K1'], ['vozila.xml']]

1 solution(s) in 0.0010 seconds

Yes

flora2 ?- shell[cmd(ps) -> ?procesi].

?procesi = [[PID, TTY, TIME, CMD], [17994, 'pts/12', '00:00:00', bash], [18163, 'pts/12', '00:00:00', zsh], [19211, 'pts/12', '00:00:00', runflora], [19216, 'pts/12', '00:00:00', xsb], [19277, 'pts/12', '00:00:00', sh], [19278, 'pts/12', '00:00:00', ps]]

1 solution(s) in 0.0010 seconds

Yes

flora2 ?- shell[cmd(who) -> ?korisnici].

Riješenje:

```
shell[
  cmd(string)=>string
].
?s:shell[cmd(?x)->?rezultat]:-shell_to_list(?x,?rezultat,?KodPogreske)@_prolog.
```

% - komentari

halt. - zatvara prolog

trace. - ulazi u mod za pokretanje programa notrace.
- izlazi iz moda za pokretanje programa

true. -cilj koji uvijek uspjeva false. -
cilj koji nikad ne uspjeva
consult(ime_modula). - uciava modul, može biti datoteka,path,libray

Aritmetičke operacije: X > Y, X < Y, X =< Y, X >= Y

Metalogički predikati - Uspjevaju ukoliko je X još neznana varijabla. Dakle istinit ako ne znamo _X.

var(_X). -> true
_X = 3, var(_X). -> false jer smo dali vrijednosti _X=3.
var(_X), _X = 3. -> true jer prvo pitamo _X koliko je a onda dajemo vrijednost.

Kontrola jednakosti :

[X,Y|R] = [a,b,c]. -> true
[X,Y,Z] = [a,b]. -> false

Kontrola izvođenja :

call(X). -> poziva ako je X uspješno izveden

! -> reže stablo rješavanja, dakle vraća prvi točan rezultat preraživanja. osoba(
joza).

osoba(barica).

?- osoba(X). ?-
osoba(X), !.

?- osoba(X), !, osoba(Y).

Negacije

negacija treba vratiti true ako je tvrdnja neistina.

not(true). -> false
not(false). -> true

Ažuriranje baze znanja:
Baza znanja čita se slijeva na desno.

asserta(R) -> ubacuje rečenicu na početak baze znanja.

?- asserta(proba(1)). ?-
proba(X).

?- asserta((druga_proba(X) :- proba(X))). -> dodajemo postojeći array u novi kao element

?- druga_proba(X).

retract -> briše iz baze znanja
retract(proba(2)).

?- asserta(proba(1)). ?-
proba(X).

?- asserta((druga_proba(X) :- proba(X))). ?-
druga_proba(X).

Vezanje logičke varijable uz rezulat logičke operacije.

?- X = 2 + 2. -> ne veže rezultat ?- X is 2
+ 2. -> veže rezutat

?- Y = 2 + 2, X is Y. -> također veže rezultat

LISTE

http://autopoiesis.foi.hr/wiki.php?name=Logi%C4%8Dko%20programiranje%20-%20FOI&parent=NULL&page=liste

- atom ili složeni term s funkcijskim simbolom.
- Dva argumenta - prvi -glava, drugi -rep liste.

1. We can specify lists in Prolog by enclosing the elements of the list in square brackets (that is, the symbols [and]). The elements are separated by commas.

2. we learn that all sorts of Prolog objects can be elements of a list. The first element of this list is mia , an atom; the second element is robber(honey_bunny) , a complex term; the third element is X , a variable; the fourth element is 2 , a number. Moreover, we also learn that the same item may occur more than once in the same list: for example, the fifth element of this list is mia , which is same as the first element.

   [mia, robber(honey_bunny), X, 2, mia]

3. The third example shows that there is a special list, the empty list. The empty list (as its name suggests) is the list that contains no elements. What is the length of the empty list? Zero, of course (for the length of a list is the number of members it contains, and the empty list contains nothing). **[]**

4. The fourth example teaches us something extremely important: lists can contain other lists as elements. For example, the second element of

   [mia, [vincent, jules], [butch,girlfriend(butch)]
   is **[vincent,jules]** . The third is **[butch,girlfriend(butch)]** .
   What is the length of the fourth list? The answer is: three. If you thought it was five

   (or indeed, anything else) you're not thinking about lists in the right way. The elements of the list are the things between the outermost square brackets separated by commas. So this list contains three elements: the first element is **mia** , the second element is **[vincent, jules]** , and the third element is **[butch, girlfriend(butch)]** .

The empty list has no internal structure; for Prolog, [] is a special, particularly simple, list.

Prolog has a special built-in operator | which can be used to decompose a list into its head and tail. It is important to get to know how to use | , for it is a key tool for writing Prolog list manipulation programs.

```
| ?- [Head|Tail]=[lovro,mario,zoran,denis,david].

Head = lovro
Tail = [mario,zoran,denis,david].

no
```

konačan skup nekih itema

```
| ?- [_,_,_,_,_,[X|Y]]= [[], dead(z), [2, [b, c]], [], Z, [2, [b, c]]].

X = 2
Y = [[b,c]]

Z = _h196
```

notacija još može biti i

[G|R] ili [a,b | R]

što izgleda ovako :

ili

Predikati s listama:
dodavanje elementa:

dodaj(E,Lu,Li):- Li = [E|Lu]. -> u listu Li gdje je glava E,rep Lu,

Suma vrijednosti u listi
suma([],0).

suma([G|R],S):-suma(R,S1),S is S1+G. -> prvi argument je lista a drugi zbroj vrijednosti u listi.

Broj elemenata u listi
broj_el([],0).

broj_el([_|R],N):-broj_el(R,N1),N is N1+1.

Rekurzije
http://www.learnprolognow.org/lpnpage.php?pagetype=html&pageid=lpn-htmlse9

Operatori

Infiksni

+(1, 2) ili

Svaki operator ima svoju jačinu od 1 do 1200. Prednosti se ostvaruju zagradama.

Pr.

a+(b/c) ili

x-> operator niže prednosti

xfx – ne asocijativni – oba argumenta operatora moraju biti izrazi s glavnim operatorom niže prednosti od samog operatora ili biti u zagradi (zagrada automatski daje prednost 0)

xfy – desno asocijativni – samo lijevi argument mora biti izraz s glavnim operatorom niže prednosti dok desni može biti jednake prednosti

yfx – lijevo asocijativni – obrnuto, desni argument mora biti izraz s glavnim operatorom niže prednosti dok lijevi može imati jednaku prednost

:- op(Prednost, Tip, Naziv)

Prefiksni

Postfiksni

http://pastebin.com/snswMJQ3

Vježba----------

%1)

%implementirajte infiksne operatore '+' i 'je' koji će
%omogućti spajanje proizvoljnih atomarnih elemenata u listu
%Pri implementaiji se smiju koristiti ugrađeni predikati

%koristenje:
% a + 1 je X.
% X = [a,1];

% a + b +d +4 je Y.
% Y = [a,b,d,4].

% a + f(z) + 3 je Z.
% no

:-op(500,xfy,+).
:-op(600,xfx,je).

%kažmo da su element X i Y lista

%u kojoj su X i Y atomarne vrijednosti

X+Y je [X,Y]:-atomic(X),atomic(Y).

%Y je Z mi služi za rekurziju, spajam glavu s tijelom.
%compound mi stvara HiLog compound term,spaja zagrade kojima rekurzija ide

% atomic definira glavu liste kao atomarnu vrijednost i mora biti tako ako hocu spojiti

X+Y je [X|Z]:-Y je Z,compound(Y),atomic(X).

%2)

%Bez koristenja ugrađenih predikata implementiraje predikat veca/3 koji ce usporediti dvije liste(prva dva argumenta) te uz trei argument vezati 'istina ako je prva lista duza od druge te 'laz' u suprotnom slucaju

%pr:
%veca([a,b,c],[1,2],X).
%X = istina.
%veca([a,b],[a,b,c],X).
%X=laz.

%neman pojma ako ovo
radi **veca([_|_],[],istina).**
veca([],_,laz).

veca([_|R1],[_|R2],X):-veca(R1,R2,X).

%3)

Implementirajte infiksne operatore + i je koji će omogućiti konkatenaciju lista. Dopušteno je korištenje ugrađenih predikata,npr. putem direktive.

import append/3 from lists.
%pr:

%[1,2,3] + [4,5,6] je
L. %L = [1,2,3,4,5,6].

%

%[1,2]+[3,4]+[5,6]+[7,8] je
L. %L = [1,2,3,4,5,6,7,8]

:-import append/3 from lists.
:-import is_list/1 from lists.

%xfy mi kaže da lijevo mora biti argument niže prednosti

:-op(400,xfy,+).
:-op(500,xfx,je).
%moramo ga nekako definirati,vrijednot smo praktički odredili s prethodnim

X + Y je L:-is_list(Y),append(X,Y,L).

%drugi dio mi triba jer moram ici rekurzivno da bi mogao obuhvatiti sve vrijdnosti koliko god da ih je

X + Y je L:- not(is_list(Y)),Y je Z, X+Z je L.

%4)
%Neka je zadana baza činjenica kako slijedi:

%osoba(ivek,1990).
%osoba(joza,1991).

%osoba(bara,1989).
%osoba(stef,1992).
%osoba(stefica,1994).

%Implementirajte predikat godine/1 koji će uz svoj atgument vezati litu osoba i odgovaaućih staroti oblika

%osoba(ime,starost).

%godine(L).
%L=[osoba(bara,23),osoba(ivek,22),osoba(joza,21)].

osoba(ivek,1990).
osoba(joza,1991).
osoba(bara,1989).
osoba(stef,1992).
osoba(stefica,1994).

%predikat starost koji od definirane osobe uzima Z i računa godine
starost(X,Y):-osoba(X,Z),Y is 2014-Z.

%godine su jednostavna lista kojoj dodajemo setof/bagof za računanje jedinstvenih
godine(L):-bagof(osoba(X,Y),starost(X,Y),L).

%5)

%Bez korištenja ugrađenih predikata implementirajte predikat putanja/3 koji će biti zadovoljen ako su prva

%dva argumenta čvorovi a treći lista koja predtavlja putanju između ta dva čvora.
%

%primjeri:
%pitanja(a,f,P).
%

%P = [a,b,c,e,f]; %P
= [a,b,d,e,f];

%
%

%putanja(b,C,P).
%C = c

%P= [b,c];
%

%C=d
%P=[b,d];
%
%C=e
%P=[b,c,e]
%
%C=f
%P=[b,c,e,f]
%

%C=e

%P=[b,d,e];
%
%

put(a,b).
put(b,c).
put(b,d)
.
put(d,e).
put(c,e).
put(e,f).

%moramo definirati da za putanju mora postojati
put **putanja(X,Y,[X,Y]):-put(X,Y).**

%rekurzivno moramo ii zato ide putanja za a put nam je
sidro. **putanja(X,Y,[X|R]):-put(X,Z),putanja(Z,Y,R).**

%6)

%Bez korištenja ugrađenih predikata implementirajte predikat zadnji/2 koji će za prvi argument primati listu,

%a za drugi vezati zadnji element te liste umanjen za jedan

%pr
%zadnji(4,2,3,1,3],X)
. %X=2

%definiramo listu i povratnu vrijednost koja je jednaka zadnjem
elementu **zadnji([X],Y):-Y is X-1.**

%element tijela liste koji je x dobivamo
rekurzivno. %zadnji([_|R],X):-zadnji(R,X).

%može i ovako
zadnji([W|R],X):-zadnji(R,X).

-

NEKA PITANJA

1. Definicija abecede računa predikata
   Abededa računa sastoji se od:
   - logičkih veznika (^,->,<->...)
   - zagrada ({})
   - konstanti C (a,b,c,d)
   - varijabli (X,Y,Z)
   - funkcijskih znakova f
   - kvantifikatora
2. Term je svaka konstanta ili varijabla. Također, term je i svaka funkcija f(t1...tn) ako su t1,...,tn termi
3. Atomarne formule - ako su t1...tn termi onda je predikat P(t1,...,tn) atomarna formula
4. Konstruiranje formula računa predikata
   - svaka atomarna formula je formula
   - ako je F formula onda je to i negirano F
   - ako su F i G formule onda je to (F and G), (F or G), (F->G), (F<->G)

- ako je F formula, a X varijabla onda je VxF također formula i €xF formula (V - onaj univerzalni kvantifikator, € je egzistencijalni)

5. Varijabla je vezana u formuli ako joj je svaki nastup uz neki kvantifikator. Slobodna je ako postoji takav nastup varijable u formuli koji nije uz kvantifikator.

6. Interpretacija računa sudova:

   ako imamo sud F(P1,...,PN) i interpretaciju i:{P1,...,Pn) koja može poprimiti vrijednosti {0,1}. Tada je sud F lažan ako je i(F)=0 ili istinit ako je i(F)=1.

7. Interpretacija terma

   ako je D domena i t term tada je:

   - t: C (ako je term konstanta)
     i(C) = d€D

   - t: X (ako je term varijabla)
     i(X) = d' € D

   - f(t1, …,tn) (ako je term funkcijski)
     preslikavamo i(F): D^n -> D

8. Model

   Uređeni par M(D, i) zovem modelom. Ako je formula F istinita u modelu M(D,i) tako da je i(F) = 1 onda kažemo da je M model za formulu F.

9. Neka je H={F1,...,Fn) skup formula i G isto formula. Tada kažemo da je G logička posljedica skupa formula H ako za svaki model M za formulu iz skupa H je to model ujedno i za formulu G.

Bez korištenja ugrađenih predikata i modula implementirajte predikat zbroji/3 koji će u prva dva agumenta primati dvije liste brojeva jednakih duljina, a uz treći argument vezati listu koja se sastoji od elemenata oblika zbroj(Z) u kojem je Z suma odgovarajućih elemenata u listama iz prva dva argumenta.

Primjeri korištenja:

| ?- zbroji([1, 2, 3], [6, 5, 4], Z).

Z = [zbroj(7),zbroj(7),zbroj(7)];

no
| ?- zbroji([2, 1, 4], [1, 5, 6], Z).

Z = [zbroj(3),zbroj(6),zbroj(10)];

no

Implementirajte predikate postavi/1 i smanji/1 koji će uz pomoć ažuriranja baze znanja omogućiti postavljanje globalne vrijednosti i njezino smanjivanje za jedan. Predikat postavi/1 će primati za argument broj koji će postaviti za globalnu vrijednost. Predikat smanji/1 će uz argument vezati globalnu vrijednost koja se svakim pozivom smanjuje za jedan.

Primjeri korištenja:

```
| ?- postavi(5).

yes
| ?- smanji(X).

X = 4;

no
| ?- smanji(X).

X = 3;

no
| ?- postavi(1).

yes
| ?- smanji(X).

X = 0;

no
| ?- smanji(X).

X = -1;

no
```

Kupac na raspolaganju ima određeni broj kuna < 20. Želi kupiti voće za voćnu salatu. Cijena voća po komadu je kao što slijedi:

Jabuka 3 kn Banana
2 kn
Šljiva 1 kn
Nar 4 kn

Implementirajte predikat salata/2 koji će za zadanu količinu novca u prvom parametru uz drugi parametar vezati sve moguće kombinacije gore navedenih vrsta voća tako da ukupna cijena bude jednaka prvom parametru.

Primjer korištenja:

| ?- salata(10, [Jabuka, Banana, Sljiva, Nar]).

Jabuka = 0
Banana = 0
Sljiva = 2
Nar = 2;

Jabuka = 0

Banana = 0

Sljiva = 6
Nar = 1;

Jabuka = 0

Banana = 0

Sljiva = 10
Nar = 0;

Jabuka = 0

Banana = 1

Sljiva = 0
Nar = 2;

Jabuka = 0

Banana = 1

Sljiva = 4
Nar = 1;

Jabuka = 0

Banana = 1
Sljiva = 8
Nar = 0;

Jabuka = 0

Banana = 2
Sljiva = 2
Nar = 1;

Jabuka = 0

Banana = 2
Sljiva = 6
Nar = 0;

Jabuka = 0

Banana = 3
Sljiva = 0
Nar = 1;

Jabuka = 0

Banana = 3

Sljiva = 4
Nar = 0;

Jabuka = 0

Banana = 4

Sljiva = 2
Nar = 0;

Jabuka = 0

Banana = 5

Sljiva = 0
Nar = 0;

Jabuka = 1

Banana = 0

Sljiva = 3
Nar = 1;

Jabuka = 1

Banana = 0

Sljiva = 7
Nar = 0;

Jabuka = 1

Banana = 1

Sljiva = 1
Nar = 1;

Jabuka = 1

Banana = 1
Sljiva = 5
Nar = 0;

Jabuka = 1

Banana = 2
Sljiva = 3
Nar = 0;

Jabuka = 1

Banana = 3
Sljiva = 1

Nar = 0;

Jabuka = 2

Banana = 0
Sljiva = 0
Nar = 1;

Jabuka = 2

Banana = 0

Sljiva = 4
Nar = 0;

Jabuka = 2

Banana = 1

Sljiva = 2
Nar = 0;

Jabuka = 2

Banana = 2

Sljiva = 0
Nar = 0;

Jabuka = 3

Banana = 0

Sljiva = 1
Nar = 0;

no

GRAMATIKE

produkcijska pravila
S0--->S1, s2, sn.

terminale pišemo unutar [ ] i njih ne izlazemo na manje dijelove
non-terminale mozemo razlagati

pocetni simbol → ono sto zelimo prepoznati
recenica → subject, predikat. (rec. se sastoji od subj. i pred.)

recenica([ivek, jede],[]) → gramatika prepoznaje je li ovdje riječ o rečenici. Ako stavimo nešto što ne spada u našu gramatiku, dobit ćemo No.

primjer:

recenica--> subjekt,predikat. subjekt
--> [ivek].

subjekt --> [joza]. predikat
--> [uci]. predikat --> [jede].
predikat --> [spava].

recenica([ivek,jede],[]). YES

drugi parametar je ostatak.ne mora biti jedna recenica vec moze biti vise razlicit..Pronaliazi prvu zadovoljivu recenicu a ostatak zapisuje u X kao ostatak.

recenica([ivek,spava,joza], X).
X=[joza]

Nemamo info sto je subjekt a sto predikat i sto je dovelo da je recenica prepoynata kao recenica.

Mogu se dodavati argumenti koji omogucavaju da kreiramo sintaksna stabla, parse tree, govori od kojeg se dijela sastoji recenica, sto je S,P itd…

recenica(recenica(S, P))--> subjekt(S), predikat(S).
ako je rezultat subjekt(S), predikat(P) u rezultat upisi recenica(S,P) → bit ce dio Parse

Tree Primjer:

recenica(recenica(S,P))-->subjekt(S),predikat(P).

subjekt(subjekt(ivek))-->[ivek].
subjekt(subjekt(joza))-->[joza].

predikat(predikat(ico))-->[uci].

member -> provjerava dal je neki element clan liste

:-auto_table. omogucava tabliranje, pamti međurezultate prilikom pretrazivanja,znatno ubrzava postupak pretraživanja

select_klauzula(sk(K,T))-->[select],kolone(K),[from],tablice(T),[';'].
zapisi rezulrate selecta pod K i T te ; je tu jer zavrsava na ;

http://autopoiesis.foi.hr/wiki.php?name=Logi%C4%8Dko%20programiranje%20-%20FOI&parent=NULL&page=bnf

#ako postoji jedna napravi ovo
kolone(kolone([X])) --> spec_kolone(X).

#ako je vise kolona zapisi prvu kako treba a iduce zapisuj rekurzivno opet u glavu kolone(kolone([X | Y])) -->

spec_kolone(X),
[','],

kolone(kolone(Y)).

{ sve unutar vitičastih zagrada se poziva kao običan Prolog, ne gleda se sematika unutar riječi}

{ nije_kljucna(K) }. → provjerava da naziv stupca nije kljucna rijec u SQL-u (FROM, WHERE i sl.)

S tablicama je ista stvar kao i sa kolonama, spremamo ih u listu, ako ih je više procesuiramo ih rekurzivno

spec_tablice(spec_tablice(T)) --> [T], { nije_kljucna(T) }.

spec_tablice(spec_tablice(T, A)) --> [T], [A], { nije_kljucna(T), nije_kljucna(A) }. Provjera da niti naziv tablice niti alias nisu kljucne rijeci

mozemo imati niz ogranicenja koja su razdvojena operatorom, moze biti samo jedna ili situacijia gdje ih je vise.

To opet može biti ako imamo operator or ili and. Svaki od njih ima svoja pravila. Primjer:

```
ogranicenja( ogranicenja( [ X ] ) ) --> spec_ogranicenja( X ).

ogranicenja( ogranicenja( [ X | Y ] ) ) --> spec_ogranicenja( X ), [and],
ogranicenja( ogranicenja( Y ) ).

ogranicenja( ogranicenja( [ X | Y ] ) ) --> spec_ogranicenja( X ), [or],
ogranicenja( ogranicenja( Y ) ).
```

```
spec_ogranicenja(spec_ogranicenja(K1,O,K2))-->
kolona_ili_operand(K1), operator(O),kolona_ili_operand(K2)).

kolona_ili_operand(kolona(K))-->spec_kolone(K).
kolona_oili_operand(broj(B))-->[B]....
```

niz - moze biti samo jedna rijec pri cemu mora biti atom i ne smije biti znak navodnika...ili

moze biti vise rijeci no onda imamo glavu i rep pri cemu prvu rijec stavljamo u glacu a ostale u rep i glava mora biti atom i ne smije biti navodnik.

Opet ovdje rekurzivno dodajemo elemente u glavu).

Specifikacija operatora :

```
operator( operator( '=' ) ) --> ['='].
operator( operator( '<' ) ) --> ['<'].
operator( operator( '>' ) ) --> ['>'].
operator( operator( '<=' ) ) --> ['<'], ['='].
operator( operator( '>=' ) ) --> ['>'], ['='].
```

pokretanje parsera:

```
stablo:-read_line(L),select_klauzula(St,L,[]),
write(St).
```

pokrecemo i ispisujemo stablo sintakse
notrace - trace => debugiranje

F-logika

-transakcije
-f molekule - šire jedinice znanja

naziv_objekta : naziv_klase[naziv_atributa_1
-> vrijednost_atr_1

naziv_metode_m(m)].

klasa_p::klasa_n - > klasa _p naslijeđuje sve od klase_n
varijable - počinju sa ?

ALT + X
flora-mode
run-flora
run-flora

ako je prvo slovo _ onda je named don't care varijabla...ne povezuje se s ničim (samo _ je don't care varijabla

?_ : osoba[ime->?ime, prezime -> ?prezime]. -- daj sve instance osobe (spremi ime i prezime u varijable ?ime i ?prezime). ?_ je don't care varijabla. Iako nemamo direktno instancu osobe, vraca sve klase koje su naslijedile klasu Osoba

?doktor[prezime->Presvetli,pacijenti->?pac],?_pac[ime->?ime,prezime->?prezime].
spajanej dvije klase pomoću varijable, kao i kod vanjskih ključeva

ivek[radno_vrijeme->od_do(?od,?do)].
radno vrijeme iveka

Agregirajuce operacije

agg{?X[?Gs] | query}

agg - operator agregacije

Gs - opcionalni dio , prema kojem možemo grupirati
pajp -> upit , u upitu se moraju pojavljivati x i varijable agregacije … ?X ne smije biti unutar ?GS

collect-bag vraća listu svih rješenja

?x = avg{ ?_g [?_s] | ?_:osoba[godine->?_g, spol->?_s] }.
prosjek godina i spol -> radimo avg nad skupom vrijednosti koji dobijemo

Moduli
učitavanje [datoteka>>modul], npr [agg>>go1]

Upiti

upit@modul.
?x:osoba @ go1

_prolog modul je direktna veza prema XSB prologu
writeln('Pozdrav')@_prolog.

Ažuriranje baze znanja

insert{...}, cilj -> ako cilj ne uspije, prvo napravi insert i to ce ostati upisano (OBICAN NACIN)

na TRANSAKCIJSKI NACIN to nece ostati upisano ako cilj faila, vec ce garbage collector pokupiti sto je insert napravio (ili ti ga zasr***)

{...} -- {izraz | upit}

Za vjezbu
OSOBA < --- STUDENT
ime:_string slusa:ag(kolegij:integer)
prezime:_string

^
|
|
NASTAVNIk
predaje: ag(kolegij,_integer)

KOLEGIJ
  naziv=>string,

  upisan(integer)=>integer,
  predavac(integer)=>integer

?k: Kolegij[upisani(?godina)->?lista]:-?lista=collectset(?s | ?s:
Student[slusa->ag(?k,?godina)]}

osoba[

        ime => string,
        prezime => string

].

nastavnik :: osoba[predaje(integer)=>integer
].

student :: osoba[

   slusa(integer)=>integer
].

kolegij[
    naziv=>string,
    upisan(?godina)=>?lista,
    predavaci(integer)=>integer
].

?k:Kolegiji[upisani(?godina)->?lista]:-
    ?lista=collectset(?s | ? s : Student[slusa->ag(?k,?godina)]}

?k:Kolegiji[predavaci(?godine)->lista?]:

?predaje = count{?_k | ?_:nastavnik[kolegij->?_k] }. ?slusa = count{?_k | ?_:student[kolegij->?_k]}.

ivek:nastavnik[ime->ivek, prezime->ivic,predaje -> { tbp, lp }]. joza:student[ime->joza, prezime->jozic,slusa->{tbp}]. bara:student[ime->bara, sprezime->baric,slusa->{lp}].

tbp:kolegij[naziv->teorijaBP].
lp:kolegij[naziv->logickoP].

ZADACI ZA KOLOKVIj

Baza znanja koja opisuje usmjereni graf:

a:cvor[brid->{b,c,d}]. b:cvor[brid->{c,e}].
c:cvor[brid->{e,d}]. d:cvor[brid->f].
e:cvor[brid->f]. f:cvor.

Rjesenje:
Metoda put

?c:cvor[put->?x]:-?c[brid->?x].
?c:cvor[put->?x]:-?c[brid->?y],?y[put->?x].

Metoda duljina koja nam govori kroz koliko cvorova moramo preci

- za susjedne cvorove je duljina 1

?c:cvor[duljina(?x)->1]:-?c[brid->?x].
?c:cvor[duljina(?x)->?y]:-?c[brid->?z],?z[duljina(?x)->?y1],?y is ?y1+1.

Klasu obojeni cvor koja nasljeduje cvor i ima atribut boja. Dodati tri cvora x,y,z koji imaju zutu, plavu i zelenu i bridove u a, b, c

x:obojeni_cvor[boja->zuta, brid->a].
y:obojeni_cvor[boja->plava, brid->b].
z:obojeni_cvor[boja->zelena,brid->c].

Izmijeniti put u put2 iz prvog zadatka koji vrijedi samo za neobojene cvorove

?c:cvor[put->?x]:-?c[brid->?x],not(?c:obojeni_cvor),not(?x:obojeni_cvor).
?c:cvor[put->?x]:-?c[brid->?y],?y[put->?x],not(?c:obojeni_cvor),not(?x:obojeni
_cvor),not(?y:obojeni_cvor).

DRUGI ZADACI ZA KOLOKVIJ

Ako zelimo INSERT, izvrsit ce se samo prvi put jer ce svaki sljedeci FLORA traziti rezultat u bazi. Zato koristimo %

1. Implelemtirati apstraktni tip podataka rijecnik. Azuriranje mora biti transakcijske Metoda dodaj/2 koja ce primati kljuc i vrijednost i dodavati odgovarajuce vrijednosti u

rijecnik
moramo prvo provjeriti i da nema nista pod tim kljucem

Moramo pokrenuti insert{a:rijecnik} u konzolu dolje

?r:rijecnik[%dodaj(?k,?v)]:- not(?r[?k->?_]),t_insert(?r[?k->v]).

Metodu brisi koja brise vrijednost za kljuc

?r:rijecnik[%brisi(?k)]:-t_delete(?r[?k->?_]).

Metoda azuriraj koja ce primiti kljuc i za njega promijeniti vrijednost

?r:rijecnik[%azuriraj(?k,?v)]:-?r[%brisi(?k)],?r[%dodaj(?k,?v)].

Metoda prikazi koja ce prikazati sadrzaj rijecnika

?r:rijecnik[%prikazi(?x)]:-?x=collectset{?z|?r[?k->?v],?z=r(?k,?v)}.

TRECI ZADACI ZA KOLOKVIJ (PROLOG)

Graf s pocetnik i zavrsnim cvorom i bojama te implementirati pocetak(P), kraj(K), put

```
pocetak(1).
kraj(12).
boja(1,bijela).
boja(2,crna).
boja(3,bijela).
boja(4,bijela).
boja(5,bijela).
boja(6,crna).
boja(7,bijela).
boja(8,bijela).
boja(9,crna).
boja(10,bijela).
boja(11,bijela).

put(1,2).
put(1,3).
put(2,3).

put(3,4).
put(3,7).
put(4,5).
put(4,6).
put(5,8).
put(6,7).
put(6,8).
put(7,9).
put(8,10).
put(9,11).
```

Metodu koja ce prikazati bilo koja dva bijela puta

```
bijeli_put(X,Y):-boja(X,bijela),boja(Y,bijela),put(X,Y).
```

Metodu bijela putanja koja ce reci jesu li izmedu dva cvora svi bijeli cvorovi

```
bijela_putanja(X,Y):-bijeli_put(X,Y).
bijela_putanja(X,Y):-bijeli_put(X,Z),bijela_putanja(Z,Y).
```

Metoda bijela putanja koja u treci argument primi listu koja je cijeli put izmedu X, Y

```
bijela_putanja(X,Y,[X,Y]):-bijeli_put(X,Y).
bijela_putanja(X,Y,[X|R]):-bijeli_put(X,Z),bijela_putanja(Z,Y,R).
```

ISPIT SA SOBAMA(09.2011)……

Zadan je modul kuca.flr koji definira objekte koji predstavljaju sobe u kuci te trenutnu poziciju igrača.

```
hodnik[vrata->'dnevni boravak',vrata->'spavaca soba'].
'spavaca soba'[vrata->kupaona].

'dnevni boravak'[vrata->kuhinja].
kuhinja[sadrzi->pizza].
igrac[pozicija->hodnik].
```

//1. zad. Potrebno je implementirati pravilo po kojem ce vrijediti da ako iz jedne sobe postoje vrata u drugu, tada postoje i vrata iz druge u prvu.

```
?_prostorija[vrata->?soba]:-?soba[vrata->?_prostorija].
```

//2. zad. Potrebno je implementirati predikat put/2 koji će provjeravati je li između dvije sobe postoji put.

```
?put(?_prostorija,?soba):-?_prostorija[vrata->?soba].
?put(?_prostorija,?soba):-?_prostorija[vrata->?y],?put(?y,?soba).
```

//3.zad. potrebno je implementirati metodu idi/1 objekta igrac koja ce provjeriti je li za igraca postoji put u zadanu sobu te promijeniti njegovu poziciju.

```
igrac[idi(?_prostorija)]:-igrac[pozicija->
?_trenutna],?put(?_trenutna,?_prostorija),delete{igrac[pozicija->
?_trenutna]},insert{igrac[pozicija->?_prostorija]}.
```

//4.zad. Potrebno je implementirati metodu uzmi/1 objekta igrac koja će provjeriti je li se zadana stvar nalazi u trenutnoj sobi

```
igrac[uzmi(?_stvar)]:-?_prostorija[sadrzi->?_stvar],insert{igrac[ima->
?_stvar]},delete{?_prostorija[sadrzi->?_stvar]}.
```

```
:- import parse_xpath/4 from xpath.

:- import load_xml_structure/3 from sgml. :-
import member/2 from lists.

:- import ith/3 from basics.


vrijednostiIzXML(I,P) :-

  parse_xpath(file('hetinfo.xml'), '/osobe/osoba/ime', Ime, []),
        parse_xpath(file('hetinfo.xml'), '/osobe/osoba/prezime', Prezime, []),
  izvuciPodatkeIzXML(Ime,Prezime,I,P).


izvuciPodatkeIzXML(Ime,Prezime,Im,Pr) :-load_xml_structure(
  string( Ime ), Imena_struct, _ ), load_xml_structure( string(
  Prezime ), Prezimena_struct, _ ), member(Ime_element,
  Imena_struct),

  ith(I, Imena_struct, Ime_element),

  ith(I, Prezimena_struct, Prezime_element),
  dohvatiElement(Ime_element,Im),
  dohvatiElement(Prezime_element,Pr).


dohvatiElement(element(_,_,[X]), X).
uzmi_ime(X,L):-read(X),findall(X,vrijednostiIzXML(X,Y),L).
```

tnx

MALO PREPRAVI

shell[

cmd(string)=>string

]. ?s:shell[cmd(?x)->?rezultat]:-shell_to_list(?x,?rezultat,?KodPogreske)@_prolog.

ISPIT(7.2013): Neka je zadana baza znanja koja opisuje usmjereni graf:
a:cvor[brid->(b, c, d }].

b:cvor[brid->(c, e }].
c:cvor[brid->(e, d }].

d:cvor[brid->(f }].
e:cvor[brid->(f }].
f:cvor.

1.) Implementirajte metodu put/0 klace cvor koja vraca sve čvorove kojima je trenutni čvor povezan usmjerenom putanjom

?c:cvor[put->?x] :- ?c[brid->?x].
?c:cvor[put->?x] :- ?c[brid->?y], ?y[put->?x].

2) implementirajte metodu duljina1 klase cvor koja za primljeni čvor vraća sve udaljenosti od trenutnog čvora

?c:cvor[duljina(?x)->1]:- ?c[brid->x].
?c:cvor[duljina(?x)->?y]:- ?c[brid -> ?z], ?z[duljina(?x) -> ?y1], ?y is ?y1 +1.

3) Implementirajte klasu obojeni_cvor koja nasljeđuje iz klase cvor te ima dodatni atribut boja. Dodajte tri bojena cvora (x,y,z) koji imaju boje zuta, plava i zelena i imaju bridove k cvorovima a b i c respektivno.

obojeni_cvor::cvor.
x:obojeni_cvor[boja->zuta, brid->a].
y:obojeni_cvor[boja->plava, brid->b].
z:obojeni_cvor[boja->zelena, brid->c].

4) izmjenite metodu put iz prvog zadatka tako da ona vrijedi samo za neobojane čvorove. Novu metodu nazovite put2/0.

?c:cvor[put2->?x] :- ?c[brid->?x], not{ ?c:obojeni_cvor},not{ ?x:obojeni_cvor}.
?c:cvor[put2->?x] :- ?c[brid->?y], ?y[put2->?x],not{ ?c:obojeni_cvor},not{ ?x:obojeni_cvor},not{?y:obojeni_cvor}.

ISPIT (2.2013).

Potrebno je implementirati apstraktni tip podataka rječnik(asocijativni niz) u flori. Obratite pozornost na to da akcija ažuriranja trebaju biti transakcije, kako bi se izbjegli neželjeni rezultati (korištenje prefiksa % kod metoda objekta i metoda ažuriranja s prefiksom 't_').

1) *Implementirajte metodu dodaj/2 koja će primati dva parametra (ključ i vrijednost) i dodavati odgovarajuće vrijednosti u riječnik.*

```
?r:rjecnik[ %dodaj( ?k, ?v) ]:- t_insert{ ?r[ ?k->?v ]},
        not( ?r[ ?k->?_]), t_insert{ ?r[ ?k->?v]}.
```

2) *Implementirajte metodu brisi/1 koja će primati ključ riječnika i obrisati vrijednost pod zadanim ključem.*

```
?r:rjecnik[ %brisi( ?k)
        ]:-t_delete{ ?r[ ?k->?_]
        }.
```

3) *Implementirajte metodu azuriraj/2 koja ce primati postojeći ključ za njega promijeniti trenutnu vrijednost na proslijeđenu.*

```
?r:rjecnik[ %azuriraj( ?k, ?v) ]:- ?r[ %brisi(?k)], ?r[ %dodaj( ?k,?v) ].
```

4) *Implementirajte metodu prikazi/1 koja ce prikazati sadržaj rječnika.*

```
?r:rjecnik[ %prikazi( ?x) ]:- ?x = collectset{ ?z | ?r[ ?k->?v ], ?z = r( ?k, ?v) }.
```

POKRETANJE FLORA2 U PROLOGU NA VJEŽBAMA

ALT + X flora-mode ALT + X
run-flora

3.zadatak pocetak(1).

kraj(1).

boja[1, bijela]. boja[2, crna].
boja[3, bijela]. boja[4, bijela].

```
boja[5, bijela]. boja[6, crna].
boja[7, bijela]. boja[8, bijela].
boja[9, crna]. boja[10, bijela].
boja[11, bijela].



put(1,2).          put(1,3).
put(2,3).


put(3,4). put(3,7).
put(4,5). put(4,6).
put(5,8). put(6,7).
put(6,8). put(7,9).
put(8,10). put(9,10).
put(9,11).

bijeli_put(X, Y):-boja(X,bijela), boja(Y, bijela), put(X, Y).


bijela_putanja(X, Y):- bijeli_put(X, Y).
bijela_putanja(X, Y):- bijeli_put(X, Z), bijela_putanja(Z, Y).


bijela_putanja( X, Y, [ X, Y] ):- bijeli_put(X,Y).


bijela_putanja(X,Y, [ X | R]):- bijeli_put(X,Z),bijeli_put( Z, Y, R).
?c:cvor[duljina(?x)->?d]:-
?c[brid->?x].
?c:cvor[duljina(?x)->?d]:-
?c[brid->?y],
?y[duljina(?x)->?d1],

?d is ?d1 + 1.



?obojeni_cvor::cvor.
x:obojeni_cvor[boja->zuta,brid->a].
y:obojeni_cvor[boja->plava,brid->b].
c:obojeni_cvor[boja->zelena,brid->c].



?c:cvor [put2 ->
?x]:-?c[brid ->?x],
not(?c:obojeni_cvor),
not(?x:obojeni_cvor).
```

```
?c:cvor[put->?x]:-?c[b
rid->?y],
?y[put2->?x],
not(?c:obojeni_cvor),
not(?x:obojeni_cvor),
not(?y:obojeni_cvor).
```

```
?r:rijecnik[%dodaj(?k,?v)]:-
not(?r[?k->?_]).
t_insert{?r[?k->?v]}.
?r:rijecnik[%brisi(?k)]:-t_de
lete{?r[?k->?_]}.


?r:rijecnik[%azuriraj(?k,?v)]:-
?r[%brisi(?k)],
?r[%dodaj(?k,?v)].
?r:rijecnik[$prikazi(?x)]:-

?x=collectset{?z | ?r[?k->?v],?z = r(?k,?v)}.
```

http://www.isi.edu/~argos/matthew/xsb_command_line_tutorial.html /*

1) *Implementirajte infiksne operatore '+' i 'je' koji Ä‡e omoguÄ‡iti spajanje proizvoljnih atomarnih elemenata u listu. Pri implementaciji se smiju koristiti ugraÄ‘eni predikati.*

Primjeri koriĹˇtenja:

| ?- a + 1 je X.

X = [a,1];

no
| ?- a + b + d + 4 je Y.

Y = [a,b,d,4];

no
| ?- a + f(1) + 3 je Z.

no

```
*/

:- op( 500, xfy, + ). :- op(
600, xfx, je ).

X + Y je [ X, Y ]
  :-atomic( X ), atomic(
  Y ).

X + Y je [ X | Z ] :-Y
  je Z, compound( Y ),
  atomic( X ).
```

```
/*
```

2) Bez koriĹˇtenja ugraÄ‘enih predikata implementirajte predikat veca/3 koji Ä‡e usporediti dvije liste (prva dva argumenta) te uz treÄ‡i argument vezati vrijednost 'istina', ako je prva lista duĹľa od druge, odnosno vrijednost 'laz' u suprotnom.

Primjeri koriĹˇtenja:

```
| ?- veca( [a,b,c], [1,2], X ). X

= istina;

no

| ?- veca( [a,b], [1,2,3], X ). X

= laz;

no

| ?- veca( [], [], X ). X

= laz;
```

no

*/

```
veca( [ _ | _ ], [], istina ).
veca( [], _, laz ).

veca( [ _ | R1 ], [ _ | R2 ], X )
  :-veca( R1, R2, X ).
```

/*

3) Neka je zadana baza ÄŤinjenica kao Ĺˇto slijedi:

```
osoba( ivek, 1990 ).
osoba( joza, 1991 ).
osoba( bara, 1989 ).
osoba( stef, 1992 ).
osoba( stefica, 1994 ).
```

Implementirajte predikat godine/1 koji Ä‡e uz svoj argument vezati listu osoba i odgovarajuÄ‡ih starosti oblika osoba(ime, starost).

Primjeri koriĹˇtenja:

```
| ?- godine(L).

L = [osoba(bara,23),osoba(ivek,22),osoba(joza,21),osoba(stef,20),osoba(stefica,18)]

yes
```

*/

```
osoba( ivek, 1990 ).
osoba( joza, 1991 ).
osoba( bara, 1989 ).
osoba( stef, 1992 ).
osoba( stefica, 1994 ).

starost( X, Y )
  :-osoba( X, Z ),
  Y is 2012 - Z.
```

```
godine( L ) :-
  setof( osoba(X,Y), starost(X,Y), L ).

/*
```

4) Bez koriĹˇtenja ugraÄ‘enih operatora implemetirajte predikat zadnji/2 koji Ä‡e za prvi argument primati listu, a uz drugi vezati zadnji element te liste umanjen za jedan.

Primjer koriĹˇtenja

```
| ?- zadnji( [4,2,3,1,3], X ).

X = 2;

 */


zadnji( [ X ], Y ) :-Y
  is X - 1.

zadnji( [ _ | R ], X )
  :-zadnji( R, X ).

/*
```

5) Implementirajte infiksne operatore + i je koji omoguÄ‡iti konkatenaciju (spajanje) lista. DopuĹˇteno je koriĹˇtenje ugraÄ‘enih predikata, npr. putem direktive:

```
?- import append/3 from lists.
```

Primjeri koriĹˇtenja:

```
| ?- [1,2,3] + [4,5,6] je L.

L = [1,2,3,4,5,6];

no
| ?- [1,2] + [3,4] + [5,6] + [7,8] je L.

L = [1,2,3,4,5,6,7,8];
```

*/

```
:- op( 500, xfy, + ). :-
op( 600, xfx, je ).

?- import append/3 from lists. ?-
import is_list/1 from lists.

X + Y je Z :-is_list( Y
  ), append( X, Y, Z ).

X + Y je Z :-

  not( is_list( Y ) ), Y
  je T,
  X + T je Z.
```

/*

6) Neka je zadana baza ÄŤinjenica kao Ĺˇto slijedi (jednostavni usmjereni graf):

```
put(a, b).


put(b, c). put(b,
d).   put(c,   e).
put(d, e). put(e,
f).
```

Bez koriĹˇtenja ugraÄ‘enih predikata implementirajte predikat putanja/3 koji Ä‡e biti zadovoljen ako su prva dva argumenta ÄŤvorovi, a treÄ‡i lista koja predstavlja putanju izmeÄ‘u ta dva ÄŤvora.

Primjeri koriĹˇtenja:

```
| ?- putanja(a, f, P).

P = [a,b,c,e,f];

P = [a,b,d,e,f];

no
| ?- putanja(b, C, P).
```

C = c
P = [b,c];

C = d
P = [b,d];

C = e
P = [b,c,e];

C = f
P = [b,c,e,f];

C = e
P = [b,d,e];

C = f
P = [b,d,e,f];

no

*/

```
put(a, b). put(b,
c).  put(b,  d).
put(c, e). put(d,
e). put(e, f).


putanja(X,Y,[X,Y])
 :-put(X,Y).


putanja(X,Y,[X|R])
  :-put(X,Z),
  putanja(Z,Y,R).
```

baza:

a1:vozilo[boja->plava,godina_proizvodnje->2005].
a2:vozilo[boja->zuta,godina_proizvodnje->1977].
a3:vozilo[boja->zelena,godina_proizvodnje->1980].
a4:vozilo[boja->crvena,godina_proizvodnje->2013].
a5:vozilo[boja->zuta,godina_proizvodnje->2014].
a6:vozilo[boja->plava,godina_proizvodnje->2002].

a7:vozilo[boja->zelena,godina_proizvodnje->1990].
a8:vozilo[boja->crvena,godina_proizvodnje->2003].

1. Potrebno je implementirati atribut je_oldtimer klase vozilo koji pokazuje je li neko vozilo starije od 30 g.

```
vozilo[je_oldtimer(integer)=>integer,
        godina_proizvodnje=>integer].


    ?v[je_oldtimer(?sada)->?je_oldtimer]:-?v[godin
    a_proizvodnje->?godina_proizvodnje],
    ?je_oldtimer \is ?sada-?godina_proizvodnje.

?v[je_oldtimer]:-?v:vozilo[je_oldtimer(2015)->?je_oldtimer],?je_oldtimer>30.
```

2. Implementiraj atribut opis koji se izračunava na temelju atributa boja i to tako da se makne zadnje slovo vrijednosti atributa i doda nastavak o_vozilo. Mogu se korisiti moduli.

```
Tag:floraprolog,flora@prolog
?v:vozilo[opis->?x]
    :-?v[boja->?_b]
    ,
    substring(?_b,0,-2,?_nb)@_prolog, str_cat(?_nb, 'o_vozilo', ?x)@_prolog.
```

3. Predikat vozilo_prema_rasponu_godina/3 koji će za prva dva argumenta primati godine te zadnji argument vezati ona vozila koja su proizvedena u zadanom rasponu.

```
raspon(?x,?y,?z):-?z[godina_proizvodnje-
    >?godina],

        ?x =<?godina,
        ?y >=?godina.
```

4. Potrebno je implementirati predikat vozila_po_boji/2 koji će agregirati vozila i njihove godine proizvodnje u listu čiji su elementi oblika v(vozilo, godina) i grupirati ih prema boji.

vozila_boja(?boja,?v):-

?v=collectset{ ?v | ?_z:vozilo[boja->?boja, godina_proizvodnje->?godina], ?v = v(?_z, ?godina) }.