

## III. DEL: REŠITVE NALOG

### 1 SINTAKTIČNI IN SEMANTIČNI PROBLEMI

1.1.

- a) igra(sonja,klavir).
- b) igra(irena,Instrument) :- ima(Instrument,klaviatura).
- c) igra\_instrument(Oseba) :- instrument(I), igra(Oseba,I).  
?- igra\_instrument(igor).
- d) ?- igra(igor,I),instrument(I).
- e) igra(boris,Instrument) :-  
    igra(sonja,Instrument),  
    igra(mojca,Instrument),  
    instrument(Instrument).
- f) igra(boris,Instrument) :-  
    instrument(Instrument),  
    (igra(sonja,Instrument);  
    igra(mojca,Instrument)).
- g)?-igra(Kdo,kitara),igra(Kdo,harmonika).
- h) igra\_kit\_har :-  
    igra(Nekdo,kitara),  
    igra(Nekdo,harmonika).  
?- igra\_kit\_har.

1.2. Definirati je treba drugačno ime procedure za vsako relacijo, npr.:

otrok(leo,lili).	% Leo je Lilin otrok.
otroci([leo,tea],lili).	% Leo in Tea sta Lilina otroka.
stev_otrok(lili,2).	% Lili ima 2 otroka.

1.3.

- a) Ana ima rada Nikota.
- b) '=>' je znak za implikacijo.
- c) '&' je znak za konjunkcijo.
- d) Karkoli je znak za disjunkcijo. (V je spremenljivka!)
- e) Zvezdica je znak za karkoli. (Zvezdica je spremenljivka!)
- f) Nekdo lahko porabi 10000 din, če je njegov osebni dohodek večji kot 10000 din.  
(Ana je spremenljivka!)
- g) Niko lahko porabi manj, kot je njegova plača. (\_100 in \_200 sta spremenljivki!)
- h) Vsakdo ima rad vsakogar. (Ana in Niko sta spremenljivki!)
- i) Kdo ima rad koga?
- j) Kdo ima rad nekoga?
- h) Ali nima nihče rad Nikota? (V negaciji vprašanja so spremenljivke univerzalno kvantificirane!)

a) stara\_mama(X,Y) :-  
    mama(X,Z),  
    roditelj(Z,Y).

roditelj(X,Y) :-  
    mama(X,Y);  
    oce(X,Y).

b) premagal(X,Y) :-  
    predal(Y,X);  
    vec\_tock(X,Y).

vec\_tock(X,Y) :-  
    tocke(X,N),  
    tocke(Y,M),  
    N > M.

c) med(C,A,B) :-  
    levo\_od(A,C),  
    desno\_od(B,C).  
med(C,B,A) :-  
    levo\_od(A,C),  
    desno\_od(B,C).

d) ima\_rad(Fant,Dekle) :-  
    lepa(Dekle),  
    bogata(Dekle).  
ima\_rad(Fant,Dekle) :-  
    dobra(Dekle),  
    inteligentna(Dekle).

e) pot\_domov(Pisarna,Dom):-  
    pes(Pisarna,Postaja),  
    prevoz(Postaja,Postaja1),  
    pes(Postaja1,Dom).

prevoz(Postaja,Postaja1) :-  
    vlak(Postaja,Postaja1);  
    avtobus(Postaja,Postaja1).

	glava	rep
a)	a	[b,c,d]
b)	a	[b X]
c)	X	Y
d)	X	[Y]
e)	a	b

<input type="radio"/>	x	<input type="radio"/>	
<input checked="" type="radio"/>	se ne ve	<input type="radio"/>	se ne ve
<input type="radio"/>	[a,b,c d]	<input type="radio"/>	[e,f,g x]
<input type="radio"/>	je ni	<input type="radio"/>	ja ni
<input type="radio"/>	a	<input type="radio"/>	[b,c,d]



1.7.

- a)  $G = a, R = [b,c,d]$
  - b)  $L = [a,b,c,d]$
  - c)  $L = [[a],a]$
  - d)  $G = [], R = []$
  - e)  $G = a, R = [c,d]$
  - f)  $L = [a|b]$
  - g)  $L = [a,b,c], T = [b,c], T1 = [c]$

18.

- a)  $X = 46$
  - b) javil bo napako, ker  $X$  ni opredeljen
  - c) no
  - d) yes
  - e) yes
  - f) glej (b)
  - g) yes
  - h)  $X = 1 + 2$
  - i) no
  - j) no ( ker je  $X^*3 = (1+2)^*3$ )
  - k)  $X = 5 + 3, Y = 16$
  - l)  $A = X + Y, Z = B + C$
  - m)  $X = a + (b + (c + d)), Y = b + (c + d), Z = c + d$

1.9.

- a) X = metka;
- b) X = metka, Y = tone
- c) X = metka, Y = tone
- d) X = metka, Y = tone, Z = metka, W = tone

1.10.

- a) X = [d,e|Y], Z = [a,b,c,d,e|Y]-Y
- b) X = [b,c,d,e|Z], Y = [b,c,d,e|Z]-Z
- c) X = [3,4,5|W], Y = [5|W], Z = [1,2,3,4,5|W]-[5|W],  
R = [1,2,3,4,5|W]-W

1.11.

Z vprašanjem "?- Cilj = pot\_do(Cilj)." dosežemo neskončni izpis:

Cilj = pod\_do(pot\_do(pot\_do(pot\_do(pot\_do( ...

1.12.

- a) da
- b) da
- c) ne
- d) ne
- e) da
- f) ne

1.13.

(V vprašanju so spremenljivke eksistencialno kvantificirane, razen znotraj predikata 'not'!)

- a)?-not((oseba(X),ima\_rad(X,vino))).
- b)?-not((oseba(X),not(ima\_rad(X,vino)))).
- c)vse\_instrumente :-  
oseba(X),  
not((instrument(I),not(igra(X,I)))).
- ?-vse\_instrumente.
- d)vsi\_igrajo :-  
instrument(I),  
not((oseba(X),not(igra(X,I)))).
- ?-vsi\_igrajo.
- e)?-not((instrument(I),oseba(X),not(igra(X,I)))).

1.14.

- a)  $X = 1 + (1 + 0)$ ; b) no; c)  $X = 1 + (1 + 0)$ ; d) no

1.15.

Premakne ničlo na desno stran izraza. Dovoljeni vhodi so izrazi z eno ali več enic in z eno ničlo, ki pa mora biti prva ali druga cifra z leve. Dovoljen operator je samo '+'. Z uporabo oklepajev je lahko ničla tudi drugje v izrazu. V obratni smeri program ne deluje (razen za a(X,1+0)).

1.16.

- a)  $X = 2, Y = 2$
- b)  $L = [1,2,3,4]$
- c)  $L = [2^*2, 2^*3, 2^*4]$

1.17.

- a)  $X = s(s(s(s(nil))))$
- b)  $L = [A,B,C]$

1.18.

$$Z = 3 + 2 + 1$$

$$Z = 2 + 3 + 1$$

$$Z = 1 + (2 + 3)$$

1.19.

- a)  $X = 15$
- b)  $X = 13$
- c)  $\max(X, Y, X) :- X \geq Y.$   
 $\max(X, Y, Y) :- X < Y.$

1.20.

- a) `clovek(joze).`
- b) `smrten(joze).`
- c) `smrten(X) :- clovek(X).`
- d) `med(nina,irena,igor).`
- e) `ima_rad(janez,X) :-`  
`ima_rad(X,vino).`
- f) `ima_rad(M,Z) :-`  
`moski(M),`  
`zenska(Z).`
- g) `ima_rad(M,P) :-`  
`moski(M),`  
`posebna_zenska(P).`
- h) `ima_rad_eno_zensko(M) :-`  
`moski(M).`

i) V prologu se eksistenčnega kvantifikatorja ne da direktno izraziti. Spodnje rešitve so samo (neuporabni) približki.

Trivialna rešitev je:

`nekateri_ljudje_so_pametni.`

Bolj smiselna rešitev je, če eksplicitno podamo nekatere ljudi, ki so pametni, npr.:

`clovek(einstein).`  
`pameten(einstein).`

Lahko pa tudi direktno zapišemo relacijo eksistence:  
`obstaja(X) :- clovek(X), pameten(X).`

j) Negativnega znanja se v prologu ne da izraziti. Zopet so spodnje rešitve samo približki. Postopkovna rešitev je:

```
dobro in slab(X) :- fail.
```

Direktno preveden stavek bi bil:

nic(X):-dober(X),slab(X).

Direktno preveden stavki bi bili:

```
nic(X):-dober(X),slab(X).
```

Uporabna rešitev je, če imamo podane dobre stvari z dejstvi oblike "dober(X)." in definiramo relacijo slab s pravilom:

```
slab(X):-not(dober(X)).
```

1.21.  $\langle X \rangle = \text{konst}(\bar{X})$  % X je konstanta (atom ali število)

a) term(X) :- atomic(X).

term(X) :- var(X).

term(X) :-

$$X = ..[F|Argi],$$

funkcijski\_simbol(F,N),

length(Argi,N),

termi(Argi).

% X je konstanta (atom ali število)

% X je neopredeljena spremenljivka

termi([]).

```
termi([Arg|Argi]) :-
```

term(Arg),

termi(Argi).

b) atomicna\_formula(A) :-

$$A = .. [P | Termi],$$

predikat(P,N),

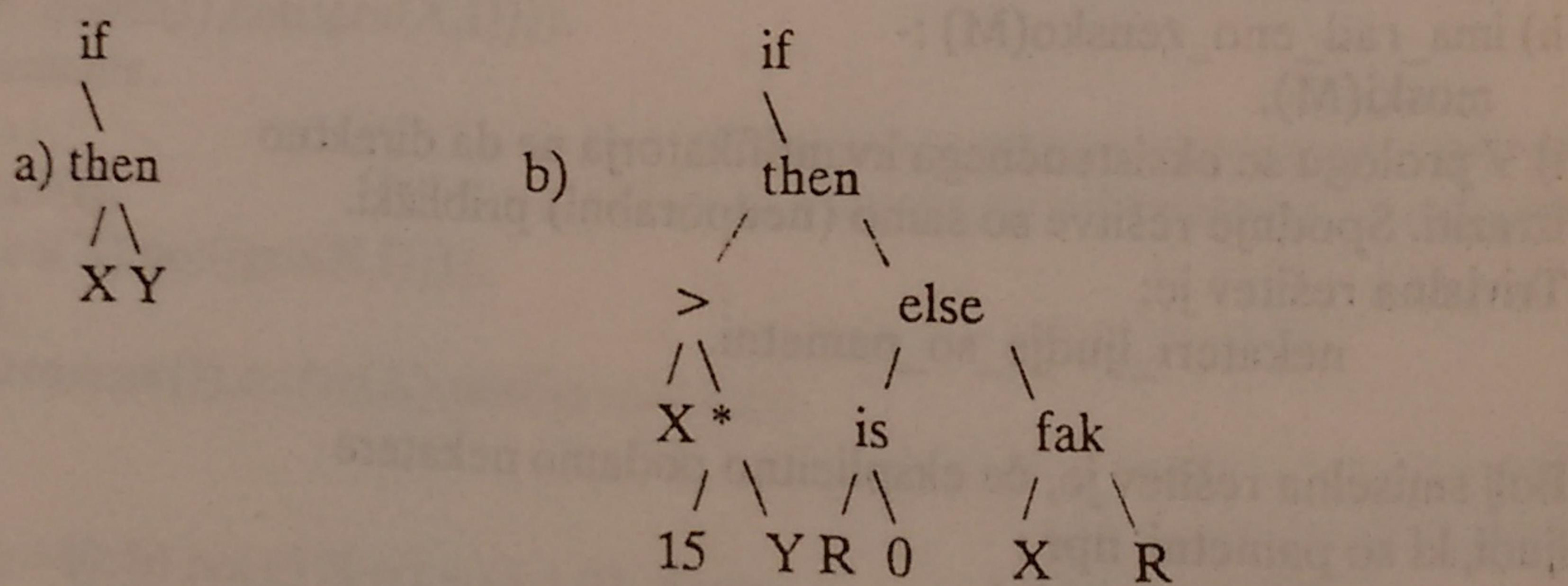
length(Termi,N),

### % N mestni predikat

c) literal(not(A)) :- atomicna formula(A).

literal(A) :- atomicna\_formula(A).

1.22. Glej sliko 7.1.



## Slika 7.1 Rešitev naloge 1.22.

## III. DEL: REŠITVE NALOG

1.23.

- a) :- op(800,xfx,je).  
 :- op(750,xf,poklic).  
 :- op(750,xfx,za).

njegov poklic je mehanik za akumulatorje

- b) :-op(800,xfx, // ).  
 :-op(750,xfy, ':').  
 :-op(740,xfx,do).  
 :-op(730,xfx,od).  
 :-op(700,fx,polet).

polet Koda\_poleta od Start do Cilj : Dan : Ura // Prevoznik

1.24.

- a) if P then Q :- call(P),!,call(Q).  
 if \_ then \_ :- !.  
 b) if P then Q else \_ :- call(P),!,call(Q).  
 if \_ then \_ else R :- call(R).

1.25.

- a) se\_lahko\_prilagodita(X,Y) :- not(not(X = Y)).  
 b) ista\_spremenljivka(prilagodi,Y) :-  
     var(Y), !, fail. % če je Y še vedno neopredeljena  
     ista\_spremenljivka(X,Y) :-  
     var(X),var(Y). % obe morata biti neopredeljeni

## 2 OPERACIJE NAD SEZNAMI IN MNOŽICAMI

2.1.

- |  |                     |
|--|---------------------|
| a) prvi([X _],X).  | b) tretji([_ _],X). |
| c) zadnji([X],X).  |                     |
| d) zadnji([_ Rep],X) :-<br>zadnji(Rep,X).                |                     |
| e) n_ti([X _],1,X).                                      |                     |
| f) n_ti([_ Rep],N,X) :-<br>N1 is N-1,<br>n_ti(Rep,N1,X). |                     |

2.2.

- desni([X,D|\_],X,D).  
 desni([\_|Rep],X,D) :-  
 desni(Rep,X,D).

.06

2.3.

razbij([Glava|Rep],Glava,Rep).

2.4.

podseznam([],\_).

podseznam([Gl|Rep],[Gl|Rep1]):-

ostanek(Rep,Rep1).

podseznam(Podsez,[\_|Rep]):-

podseznam(Podsez,Rep).

ostanek([],\_).

ostanek([G|R],[G|R1]):-

ostanek(R,R1).

5.

dodaj\_z(A,Sez,[A|Sez]).

% dodaj\_k(A,Sez,Sez1) :-

% stik(Sez,[A],Sez1).

dodaj\_k(A,[],[A]).

dodaj\_k(A,[Gl|Rep],[Gl|Rep1]):-

dodaj\_k(A,Rep,Rep1).

6.

brisi1(A,[],[]).

% če elementa ni v seznamu

brisi1(A,[A|Rep],Rep) :- !.

brisi1(A,[B|Rep1],[B|Rep]) :-

brisi1(A,Rep1,Rep).

brisi\_vse(A,[],[]).

brisi\_vse(A,[A|Rep1],Rep) :- !,

brisi\_vse(A,Rep1,Rep).

brisi\_vse(A,[B|Rep1],[B|Rep]):-

brisi\_vse(A,Rep1,Rep).

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

daljsi([\_|Rep1],[\_|Rep]):-

daljsi(Rep1,Rep).

dolzina([],0).

dolzina([\_|Rep],N1):-

dolzina(Rep,N),

N1 is N + 1.

stevilo(A,[],0).

stevilo(A,[A|Rep],N1) :-

```

stevilo(A,Rep,N),
N1 is N + 1.
stevilo(A,[_|Rep],N) :-  

    stevilo(A,Rep,N).

```

2.10.

```

sestej([],0).
sestej([I|Rep],N1) :-  

    integer(I),!,  

    sestej(Rep,N),  

    N1 is N + I.
sestej([_|Rep],N) :-  

    sestej(Rep,N).

```

2.11.

```

perm([],[]).
perm([A|Rep],Perm) :-  

    perm(Rep,Rep1),
    vstavi(A,Rep1,Perm).

vstavi(A,Sez,[A|Sez]).
vstavi(A,[G1|Rep],[G1|Rep1]):-
    vstavi(A,Rep,Rep1).

```

2.12.

```

je_seznam([]).
je_seznam([_|Rep]):-
    je_seznam(Rep).

```

2.13.

palindrom(Sez) :- obrni(Sez,Sez). % glej pogl. 3.2

2.14.

```

mnozica([],[]).
mnozica([G1|Rep],Mno) :-  

    element(G1,Rep),!, % glej pogl. 3.2  

    mnozica(Rep,Mno).
mnozica([G1|Rep1],[G1|Rep]):-
    mnozica(Rep1,Rep).

```

2.15.

Glej pogl. 3.2.

2.16.

```

dodaj_el(E,Mno,Mno) :- element(E,Mno),!.
dodaj_el(E,Mno,[E|Mno]).
```

b)  $1 + 2 + 3 + 4$

2.17.

```
podmnozica([], _).
podmnozica([A | Rep], Mno) :-  
    element(A, Mno),  
    podmnozica(Rep, Mno).
```

2.18.

```
je_mnozica(Mno) :- mnozica(Mno, Mno). % glej nal. 2.14
```

2.19.

```
unija([], Mno, Mno).
unija([E | Rep], Mno, Unija) :-  
    element(E, Mno), !,  
    unija(Rep, Mno, Unija).
unija([E | Rep], Mno, [E | Rep1]) :-  
    unija(Rep, Mno, Rep1).
```

2.20.

```
presek([], _, []).
presek([E | Rep1], Mno, [E | Rep]) :-  
    element(E, Mno), !,  
    presek(Rep1, Mno, Rep).
presek([_| Rep], Mno, Presek) :-  
    presek(Rep, Mno, Presek).
```

2.21.

```
brisi_el(E, Mno1, Mno) :- brisi1(E, Mno1, Mno). % glej nalog 2.6
```

2.22.

```
brez(Mno, [], Mno).
brez(Mno1, [E | Rep], Mno) :-  
    brisi_el(E, Mno1, Mno2),  
    brez(Mno2, Rep, Mno).
```

2.23.

```
enaki(Mno, Mno).
enaki([E | Rep], Mno1) :-  
    element(E, Mno1),  
    brisi_el(E, Mno1, Mno),  
    enaki(Rep, Mno).
```

2.24.

```
moc([], 0).
moc([_| Rep], N1) :-  
    moc(Rep, N),  
    N1 is N + 1.
```

2.25.

```

potenca([],[]).
potenca([E|Rep],Pot1):- 
    potenca(Rep,Pot), % za vsako množico iz Pot dodaj
    razsiri(E,Pot,Pot1). % isto množico z dodanim elementom E

razsiri(_,[],[]).
razsiri(E,[Mn|Rep],[Mn,[E|Mn]|Rep1]):-
    razsiri(E,Rep,Rep1).

```

2.26.

```

:- op(600,xfy,in).

stik(konec,Sez,Sez).
stik(Gl in Rep,Sez,Gl in Rep1) :- 
    stik(Rep,Sez,Rep1).

element(X, X in _).
element(X, _ in Rep) :-
    element(X,Rep).

```

2.27.

```

isci([X|_],F,X) :-
    P = .. [F,X],
    call(P), !.
isci(_|Rep],F,X) :-
    isci(Rep,F,X).

```

2.28.

(a) izlusci(Sez,Stev,Rez) :- izlusci(Sez,Stev,1,Rez).

```

izlusci(_,[],[],[]).
izlusci([X|Sez],[N1|Stev],N1,[X|Rep]) :-
    !,
    N2 is N1 + 1,
    izlusci(Sez,Stev,N2,Rep).

izlusci(_|Sez],[N|Stev],N1,Rez) :-
    N > N1,
    N2 is N1 + 1,
    izlusci(Sez,[N|Stev],N2,Rez).

```

(b) izlusci(Sez,Stev,Rez) :-

```

hitri(Stev,Sort), % glej nalog 2.30
izlusci(Sez,Sort,1,Rez). % glej nalog (a)

```

2.29. razlika(Sez,Raz-X) :- konec(X,Sez,Raz).

konec(X,[],X).  
 konec(X,[G|Rep],[G|Rep1]) :-  
 konec(X,Rep,Rep1).

vrni\_sez(X-[],X).

2.30. pretvori(Sez,N) :- pretvori(Sez,\_N).

pretvori([C],0,C).  
 pretvori([C|Cifre],K,N) :-  
 pretvori(Cifre,K1,N1),  
 K is K1 + 1,  
 pot(10,K,P), % potenca  
 N is N1 + C\*P.

pot(P,0,1) :- !.  
 pot(P,K,Pot) :-  
 K1 is K - 1,  
 pot(P,K1,P1),  
 Pot is P1 \* P.

2.31.

(a) poberi(Izraz,[Izraz|Sez]) :-  
 Izraz = .. [ \_ | Args],  
 poberivse(Args,Sez).

poberivse([],[]).  
 poberivse([Izraz|Rep],Sez) :-  
 poberi(Izraz,Sez1),  
 poberivse(Rep,Sez2),  
 stik(Sez1,Sez2,Sez). % glej pogl. 3.2

(b)  
 poberi(Izraz,Sez) :- poberi(Izraz,Sez-[]).

poberi(Izraz,[Izraz|Sez]-X) :-  
 Izraz = .. [ \_ | Args],  
 poberivse(Args,Sez-X).

poberivse([],X-X).  
 poberivse([Izraz|Rep],Sez-X) :-  
 poberi(Izraz,Sez-X1),  
 poberivse(Rep,X1-X).

2.32.

```
ok(X) :- ok(X,_).
```

```
ok([X],X).
```

```
ok([G|Rep],G2) :-  
    ok(Rep,G),  
    G2 is 2*G2.
```

2.33.

Seznam z dvema repoma je binarno drevo.

### 3 PODATKOVNE BAZE

3.1.

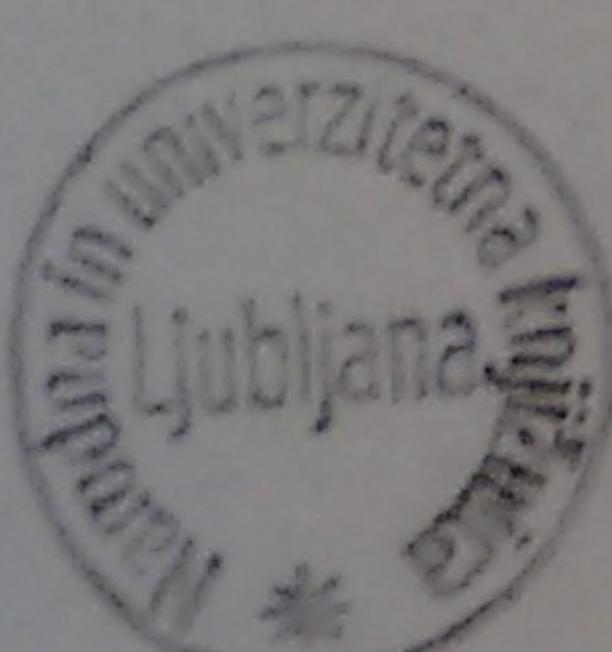
- a)?-predavanja(ponedeljek,\_,\_,Predmet).
- b)?-predavanja(torek,Od,Do,Predavanje);vaje(torek,Od,Do,Vaje).
- c)?-predavanja(Dan,Od,Do,programiranje\_2).
- d)?-vaje(petek,\_,\_,\_).

3.2.

- a)?-profesor(Profesor,programske\_jeziki\_2).
- b)?-profesor(polak,Predmet).
- c)?-profesor(delak,Predmet),predavanja(Dan,Od,Do,Predmet).
- d)?-profesor(Ucitelj,programske\_jeziki\_2);  
asistent(Ucitelj,programske\_jeziki\_2).
- e)?-asistent(kolar,Predmet),vaje(petek,\_,\_,Predmet).

3.3.

- a)?-profesor(Profesor,r\_4\_C),uci(Profesor,matematika).
- b)?-uci(Profesor,slovenscina).
- c)ucenca\_uci(Profesor,Ucenec) :-  
 profesor(Profesor,Razred),  
 ucenec(Ucenec,Razred).  
?-ucenca\_uci(mavec,Ucenec).
- d)ima(Razred,Predmet) :-  
 profesor(Profesor,Razred),  
 uci(Profesor,Predmet).  
?-ima(r\_2\_A,kemija).
- e)koliko(Stevilo,Predmet) :-  
 findall(Ucenec,  
 (ima(Razred,Predmet), % glej (d)  
 ucenec(Ucenec,Razred)),  
 Sez),  
 length(Sez,Stevilo). % vgrajena procedura  
?-koliko(Koliko,glasbena\_vzgoja).
- f)vec\_kot(Profesor,N) :-  
 uci(Profesor,\_), % rezultat se bo večkrat ponovil, če  
 % en profesor uči več predmetov



b)  $1 + 2 + 3 = \dots$

findall(Razred,  
profesor(Profesor,Razred),  
Sez),  
length(Sez,M),  
M > N.  
?-vec\_kot(Profesor,5).

3.4.

pouk(Dan,Ura):-  
(vaje(Dan,Od,Do,\_);  
predavanja(Dan,Od,Do,\_)),  
Ura < Do,  
Ura >= Od.

3.5.

zveza(\_ , Iz, Odhod, V, Prihod):-  
vsak\_dan(Iz, Odhod, V, Prihod).  
zveza(nedelja, Iz, Odhod, V, Prihod):-  
nedelja(Iz, Odhod, V, Prihod).  
zveza(delavnik, Iz, Odhod, V, Prihod):-  
delavnik(Iz, Odhod, V, Prihod).  
zveza(nedelja, Iz, Odhod, V, Prihod):-  
(nedelja(Iz, Odhod, Vmes, PrihVm);  
vsak\_dan(Iz, Odhod, Vmes, PrihVm)),  
zveza(nedelja, Vmes, OdhVm, V, Prihod),  
OdhVm >= PrihVm.  
zveza(delavnik, Iz, Odhod, V, Prihod):-  
(delavnik(Iz, Odhod, Vmes, PrihVm);  
vsak\_dan(Iz, Odhod, Vmes, PrihVm)),  
zveza(delavnik, Vmes, OdhVm, V, Prihod),  
OdhVm >= PrihVm.

3.6.

zveza1(Vrsta\_dneva, Mesto1, Odhod, Mesto2, Prihod):-  
zveza(Vrsta\_dneva, Mesto1, Odhod, Mesto2, Prihod, 0).

zveza(\_ , Iz, Odhod, V, Prihod, Prej):-  
vsak\_dan(Iz, Odhod, V, Prihod),  
Odhod >= Prej.  
zveza(nedelja, Iz, Odhod, V, Prihod, Prej):-  
nedelja(Iz, Odhod, V, Prihod),  
Odhod >= Prej.  
zveza(delavnik, Iz, Odhod, V, Prihod, Prej):-  
delavnik(Iz, Odhod, V, Prihod),  
Odhod >= Prej.  
zveza(nedelja, Iz, Odhod, V, Prihod, Prej):-  
(nedelja(Iz, Odhod, Vmes, PrihVm);  
vsak\_dan(Iz, Odhod, Vmes, PrihVm)),

```

Odhod > = Prej,
zveza(nedelja,Vmes,OdhVm,V,Prihod,PrihVm).
zveza(delavnik,Iz,Odhod,V,Prihod,Prej) :- 
  (delavnik(Iz,Odhod,Vmes,PrihVm));
  vsak_dan(Iz,Odhod,Vmes,PrihVm)),
Odhod > = Prej,
zveza(delavnik,Vmes,OdhVm,V,Prihod,PrihVm).

```

3.7.

- a)?-zvezal(delavnik,celje,Odhod,murska\_sobota,Prihod).
- b)?-zvezal(nedelja,ljubljana,Kdaj,maribor,Prihod), Prihod = < 13.
- c)?-zvezal(delavnik,maribor,Odhod,celje,Prihod), Odhod > = 14.

3.8.

```

gostota(Drzava,Preb_na_kv_km) :- 
  populacija(Drzava,Preb_v_mio),
  povrsina(Drzava,Kv_km_v_mio),
  Preb_na_kv_km is Preb_v_mio/Kv_km_v_mio.

```

3.9.

a) preveri1(Sez) :-  
 findall(El,(elementaren(El),del(El,\_)),Sez1),  
 mnozica(Sez1,Sez). % glej nal. 2.14.

b) sestavni\_dele(Del,Sez) :-  
 findall(El,(del1(Del,El),elementaren(El)),Sez1),  
 mnozica(Sez1,Sez). % glej nal. 2.14.

del1(Del,El) :-  
 del(Del,Sez),  
 element(El,Sez). % glej pogl. 3.2

del1(Del,El) :-  
 del(Del,Sez),  
 element(Vmes,Sez),  
 del1(Vmes,El).

c) preveri2(Sez) :-  
 findall(El,  
 (del1(\_El),not(del(El,\_))),not(elementaren(El))),  
 Sez1),  
 mnozica(Sez1,Sez). % glej nal. 2.14.

d) izdelki(Sez) :-  
 findall(Izd,(del(Izd,\_),not(del1(\_Izd))),Sez1),  
 mnozica(Sez1,Sez). % glej nal. 2.14.

e) Nepotrebna so vse dejstva elementaren(Del).

3.10.

```
ok(Cas) :-  
    naklada(Cas,Nak),  
    prodaja(Cas,Sez),  
    sestej(Sez,Nak).
```

```
sestej([],0).  
sestej([Kraj/St|Sez],N) :-  
    sestej(Sez,N1),  
    N is N1 + St.
```

3.11.

```
preveri :-  
    prev_prof,  
    prev_pred,  
    prev_let.
```

```
prev_prof :-  
    not(urnik(_,P1,Prof,Pr1,Dan,Ura),  
        urnik(_,P2,Prof,Pr2,Dan,Ura),  
        (P1 \= P2; Pr1 \= Pr2)).
```

```
prev_let :-  
    not(urnik(L,P1,Prof1,Pr1,Dan,Ura),  
        urnik(L,P2,Prof2,Pr2,Dan,Ura),  
        (P1 \= P2; Pr1 \= Pr2; Prof1 \= Prof2)).
```

```
prev_pred :-  
    not(urnik(_,P1,Prof1,Pr,Dan,Ura),  
        urnik(L,P2,Prof2,Pr,Dan,Ura),  
        (P1 \= P2; Prof1 \= Prof2)).
```

3.12.

- a) ?- delavec(\_,\_,Kv,\_TOZD).
- b) ?- delavec(\_,Ime,\_,\_TOZD).
- c) ?- delavec(Del,Ime,'VK',OD,TOZD).
- d) ?- delavec(\_,Ime,'VK',\_,TOZD).

e) ime(Ime,Meja,Kv) :-  
 delavec(\_,Ime,Kv,OD,\_),  
 OD > Meja.

?- ime(Ime,10000,'VK').

f) ?- delavec(\_,Ime,\_,\_4);delavec(\_,Ime,\_,\_7).

g) ime(Ime,Mesto) :-

    delavec(\_,Ime,\_,\_TOZD),  
     tozd(TOZD,\_,Mesto).

?- ime(Ime,'LJUBLJANA').

h) ?- ime(Ime,Mesto). % glej (g)

i) sel(Ime,Mesto,St\_pr,Fun,Kv,Mejasr) :-  
     delavec(Del,Ime,Kv,\_TOZD),  
     tozd(TOZD,\_,Mesto),

```

vloga(Del,St_pr,Fun),
projekt(St_pr,_,Sred),
Sred > Mejasr.
?-sel(Ime,Mesto,St_pr,Fun,'VK',950000).
j)sel(Naziv,Mesto,MejaOD,Fun,Mejasr) :-
    tozd(TOZD,Naziv,Mesto),
    delavec(Del,_,_,OD,TOZD),
    OD > MejaOD,
    vloga(Del,St_pr,Fun),
    projekt(St_pr,_,Sred),
    Sred < Mejasr.
?-sel(Naziv,Mesto,10000,'IZVAJALEC',5000000).

```

## 3.13.

Bazo podatkov lahko sestavimo iz dejstev oblike:

```

druzina(Ime,St_otrok,OD_na_clana).

sola_v_naravi(Ime_druzine,St_otrok).

```

Pravilo:

```

prispevek(Ime_druzine,Prispevek) :-
    sola_v_naravi(Ime_druzine,St_otrok),
    osnovni(Ime_druzine,Osnova),
    dodatek(Osnova,St_otrok,Dodatek),
    Prispevek is (Osnova + Dodatek) * St_otrok.

```

```

osnovni(Ime,2) :-
    druzina(Ime,_Od),
    Od = < 7.

osnovni(Ime,1.5) :-
    druzina(Ime,_Od),
    Od = < 9.

osnovni(Ime,0.5) :-
    druzina(Ime,_Od),
    Od = < 11.

osnovni(_,0).

```

```

dodatek(0,_0).
dodatek(_,1,0).
dodatek(_,2,0.5),
dodatek(_,N,1.5) :- N > 2.

```

## 3.14.

```

dodaj(Dejstvo) :-
    clause(Dejstvo,true),!;
    asserta(Dejstvo).

```

3.15.

```
dodaj(Dejstvo) :-  
    not(negacija(Dejstvo)),  
    asserta(Dejstvo).
```

3.16.

Isto kot naloga 3.15.

3.17.  
Negativno znanje v vprašanju dejansko pomeni, da vprašanje ne sme uspeti. Če dodamo dejstvo mora ostati ta lastnost ohranjena:

```
dodaj(Dejstvo) :-  
    asserta(Dejstvo),  
    not((vprasanje(V),  
          call(V)),!).  
dodaj(Dejstvo) :-  
    retract(Dejstvo).
```

```
% če dodamo dejstvo, potem  
% ne sme obstojati vprašanje,  
% ki bi uspelo  
% 'not' v prejšnjem stavku  
% ni uspel, zato odstrani dejstvo
```

#### 4 MATEMATIČNI PROBLEMI

4.1.

```
s(N,N1) :-  
    integer(N),!,  
    N1 is N + 1.  
s(N,N1) :-  
    integer(N1),!,  
    N is N1 - 1.  
s(0,1).  
s(N1,N2) :-  
    s(N,N1),  
    N2 is N1 + 1.
```

4.2

```
vsota(A,0,A) :- !.  
vsota(A,B1,Suma1) :-  
    s(B,B1),  
    vsota(A,B,Suma),  
    s(Suma,Suma1).
```

4.3.

```
razlika(A,0,A) :- !. % še enostavnejše: razlika(A,B,C) :-  
razlika(A,B1,Razlika) :- % %  
    s(B,B1), %  
    razlika(A,B,Razlika1),  
    s(Razlika,Razlika1).
```

4.4.

```
produkt(A,0,0) :- !.
produkt(A,B1,Produkt1) :-
    s(B,B1),
    produkt(A,B,Produkt),
    vsota(Produkt,A,Produkt1).
```

4.5.

```
div(A,B,0) :- B > A, !.
div(A1,B,Kvocient1) :-
    razlika(A1,B,A),
    div(A,B,Kvocient),
    s(Kvocient,Kvocient1).
```

4.6.

mod(A,B,Ostanek) :-	% mod(A,B,A) :- A < B, !.
div(A,B,D),	% mod(A1,B,Ostanek) :-
produkt(B,D,P),	%     razlika(A1,B,A),
razlika(A,P,Ostanek).	%     mod(A,B,Ostanek).

4.7.

```
st_cifer(X,1) :- X < 10,!.
st_cifer(X1,Cifre1) :-
    div(X1,10,X),
    st_cifer(X,Cifre),
    s(Cifre,Cifre1).
```

4.8.

```
st_cifer_N(X,_,Cifre) :- st_cifer(X,Cifre). % glej nal. 4.7.
```

4.9.

```
potenca(X,0,1) :- !.
potenca(X,Pot1,Rezultat1) :-
    s(Pot,Pot1),
    potenca(X,Pot,Rezultat),
    produkt(Rezultat,X,Rezultat1).
```

4.10.

```
log2(1,0) :- !.
log2(X1,LogX1) :-
    div(X1,2,X),
    log2(X,LogX),
    s(LogX,LogX1).
```

4.11.

```
logN(1,_,0) :- !.
logN(X1,N,LogX1) :-
    div(X1,N,X),
```

$\log N(X, N, \log X),$   
 $s(\log X, \log X_1).$

4.12.

$\text{abs}(X, X) :- X >= 0.$   
 $\text{abs}(X, AbsX) :-$   
 $X < 0,$   
 $AbsX \text{ is } -X.$

$\text{sign}(X, 1) :- X > 0.$   
 $\text{sign}(X, 0) :- X = 0.$   
 $\text{sign}(X, -1) :- X < 0.$

4.13.

$fak(0, 1) :- !.$   
 $fak(N1, Nfak1) :-$   
 $N \text{ is } N1 - 1,$   
 $fak(N, Nfak),$   
 $Nfak1 \text{ is } Nfak * N1.$

4.14.

$\text{ack}(0, Y, 1) :- !.$   
 $\text{ack}(1, 0, 2) :- !.$   
 $\text{ack}(X, 0, Ack) :- X >= 2, Ack \text{ is } X + 2, !.$   
 $\text{ack}(X1, Y1, Ack1) :-$   
 $X \text{ is } X1 - 1,$   
 $\text{ack}(X, Y1, Ack),$   
 $Y \text{ is } Y1 - 1,$   
 $\text{ack}(Ack, Y, Ack1).$

4.15.

$\text{: op}(800, xfx, iss).$

$X \text{ iss } A :-$   
 $\text{aritm\_izraz}(A),$   
 $X \text{ is } A.$

$\text{aritm\_izraz}(A) :- \text{number}(A), !.$   
 $\text{aritm\_izraz}(A) :-$   
 $\text{nonvar}(A),$   
 $\text{not}(\text{atom}(A)),$   
 $A = ..[\text{Op} \mid \text{Args}],$   
 $\text{izrazi}(\text{Args}).$

$\text{izrazi}([A]) :- \text{aritm\_izraz}(A), !.$   
 $\text{izrazi}([A \mid \text{Args}]) :-$   
 $\text{aritm\_izraz}(A),$   
 $\text{izrazi}(\text{Args}).$

4.16.

```
med(X,Min,Max) :-  
    X is Min + 1,  
    X < Max.  
med(X,Min,Max) :-  
    Min < Max,  
    Min1 is Min + 1,  
    med(X,Min1,Max).
```

4.17.

Preverja, če je X delitelj od Y.

```
d(X,Y) :- 0 is Y mod X.
```

4.18.

```
ima_delitelja(X) :-  
    med(N,1,X), % glej nal. 4.16  
    med(M,1,X),  
    X is M * N.
```

4.19.

```
prastevilo(X) :- not(ima_delitelja(X)). % glej nal. 4.18.
```

4.20.

```
:- op(200, yfx, ^ ).
```

```
odvod(- I, - Di) :- odvod(I, Di).  
odvod(U + V, Du + Dv) :-  
    odvod(U, Du),  
    odvod(V, Dv).  
odvod(U - V, Du - Dv) :-  
    odvod(U, Du),  
    odvod(V, Dv).  
odvod(U * V, U * Dv + V * Du) :-  
    odvod(U, Du),  
    odvod(V, Dv).  
odvod(U / V, (V * Du - U * Dv) / V ^ 2) :-  
    odvod(U, Du),  
    odvod(V, Dv).  
odvod(U ^ N, N * U ^ N1 * Du) :-  
    integer(N),  
    N \= 0,  
    N1 is N - 1,  
    odvod(U, Du).  
odvod(N ^ U, N ^ U * ln(N) * Du) :-  
    integer(N),  
    odvod(U, Du).  
odvod(sin(U), Du * cos(U)) :-
```

```

odvod(U,Du).
odvod(cos(U), - Du * sin(U)) :- 
    odvod(U,Du).
odvod(tg(U), D) :- 
    odvod(sin(U)/cos(U),D).
odvod(ctg(U), D) :- 
    odvod(cos(U)/sin(U),D).
odvod(ln(U), 1/U * Du) :- 
    odvod(U,Du).
odvod(x,1).
odvod(N,0) :- 
    integer(N).

```

4.21.

```

spremeni(Milje,Km) :- 
    Km iss Milje * 1.607. % glej nal. 4.15.
spremeni(Milje,Km) :- 
    Milje iss Km / 1.607.

```

4.22.

```

spremeni(Rim,Arab) :- 
    name(Rim,Sez),
    spr(Sez,Vr),
    sestej(Vr,Arab). % glej nal. 2.10

```

```

spr([],[]).
spr([X|Rep],[N|Rep1]) :- 
    ([X] = "M", N = 1000;
     [X] = "D", N = 500;
     [X] = "C", N = 100;
     [X] = "L", N = 50;
     [X] = "X", N = 10;
     [X] = "V", N = 5),
    spr(Rep,Rep1).
spr([X,Y|Rep],[N|Rep1]) :- 
    ([X,Y] = "IX", N = 9;
     [X,Y] = "IV", N = 4),
    spr(Rep,Rep1).
spr([X|Rep],[1|Rep1]) :- 
    [X] = "I",
    spr(Rep,Rep1).

```

4.23.

```

spremeni(Arab,Rim) :- 
    spr(Arab,1000,"M",Ost,Tren,Praz),
    spr(Ost,500,"D",Ost1,Praz,Praz1),
    spr(Ost1,100,"C",Ost2,Praz1,Praz2),
    spr(Ost2,50,"L",Ost3,Praz2,Praz3),

```

## III. DEL: REŠITVE NALOG

```
spr(Ost3,10,"X",Ost4,Praz3,Praz4),
spr1(Ost4,Praz4),
name(Rim,Tren).
```

```
spr(N,M,[R],Ost,[R | Rep],Praz) :-
    N >= M,
    N1 is N - M,
    spr(N1,M,[R],Ost,Rep,Praz).
spr(N,M,_N,Praz,Praz) :-
    N < M.
```

```
spr1(9,"IX") :- !.
spr1(4,"IV") :- !.
spr1(N,Sez) :-
    spr(N,5,"V",Ost,Sez,Praz),
    spr(Ost,1,"I",0,Praz,[]).
```

4.24.

```
kv_koren(N,K) :-
    Max is N div 2,
    kv_koren(N,K,0,Max).
```

```
kv_koren(N,K,K,_) :- N is K * K.
kv_koren(N,Min,Min,Max) :-
    Max - Min < 2.
kv_koren(N,K,Min,Max) :-
    Nov is (Max - Min) div 2,
    Nov2 is Nov * Nov,
    if Nov2 > N then kv_koren(N,K,Min,Nov2) % glej nal.1.24.
        else kv_koren(N,K,Nov2,Max).
```

4.25.

/\* če na vhodu ni več simbolov, potem vhodno besedo sprejme,  
če se nahaja v končnem stanju \*/

```
spr([],_,S,F) :- element(S,F), % S je trenutno stanje
               write(sprejel). % F je množica konč. stanj
```

/\* en prehod v seznamu prehodov ima lahko obliko npr. S-A-S1, kar pomeni prehod iz stanja S v S1 nad simbolom A \*/

```
spr([A | Rep],Prehodi,S,F) :-
    element(S-A-S1,Prehodi), % izberi prehod iz S nad simbolom A
    spr(Rep,Prehodi,S1,F).
spr(_,_,_,_) :- write(ni_sprejel).
```

4.26.

```

/* za neuspešen zaključek potrebujemo posebno proceduro */
sprejel(V,P,S,F) :- spr1(V,P,S,F).
sprejel(_,_,_) :- write(ni_sprejel).

/* en epsilon-prehod v seznamu prehodov ima lahko obliko npr. S + S1, kar
pomeni epsilon prehod iz stanja S v S1. V vsakem koraku naredimo poljubno
število epsilon prehodov(brez ciklanja!) in en navaden prehod */
spr1(Vhod,Prehodi,S,F) :-
    spr(Prehodi,S,S1,[S]), % zadnji argument je sled
    eps(Prehodi,S1,[S]),
    spr(Vhod,Prehodi,S1,F).

eps(_,S,S,_). % ne naredi nobenega epsilon-prehoda
eps(Prehodi,S,S1,Sled) :-
    element(S + S2,Prehodi),
    not(element(S2,Sled)), % nismo v ciklu epsilon prehodov
    eps(Prehodi,S2,S1,[S2|Sled]).

/* če na vhodu ni več simbolov, potem vhodno besedo sprejme, če se nahaja v
končnem stanju */
spr([],_,S,F) :- element(S,F), % S je trenutno stanje
                write(sprejel). % F je množica konč. stanj

/* en prehod v seznamu prehodov ima lahko obliko npr. S-A-S1, kar pomeni
prehod iz stanja S v S1 nad simbolom A */
spr([A|Rep],Prehodi,S,F) :-
    element(S-A-S1,Prehodi), % izberi prehod iz S nad simbolom A
    spr1(Rep,Prehodi,S1,F).

```

## 4.27.

izrac(0,0) :- !.	% ničelna funkcija
izrac(X + 1,X1) :- !, X1 is X + 1.	% naslednik
izrac(X,X) :- number(X),!.	% rezultat projekcije
izrac(min(Y,G),Y) :- !,	% minimizacija
dobi_min(Y,G).	
izrac(Fun,Vrednost) :- !,	
Fun =.. [F Argumenti],	
izracunaj(Argumenti,Arg1),	
Kfun =.. [F Arg1],!,	
def(Fun1 = Fun2),	
ustreza(Kfun,Fun1),!,	
izrac(Fun2,Vrednost).	

% izračunaj vrednosti argumentov

```

izracunaj([],[]).
izracunaj([X|Arg],[K|Konst]) :-
    izrac(X,K),!,
    izracunaj(Arg,Konst).

```

% funkciji si ustreza, če ju lahko prilagodimo, ali če bi ju  
 % lahko prilagodili, le da je prvi argument druge funkcije  
 % oblike Y + 1 (primitivna rekurzija)

```
ustreza(F,F) :- !.  

ustreza(F1,F) :-  

  F = .. [G,Y+1|Arg],  

  F1 = .. [G,N|Arg],!,  

  Y is N - 1.
```

% minimizacija: poišči prvi Y za katerega je G enako 0

```
dobi_min(Y,G) :-  

  daj(Y), % generiraj naravna števila od 0 naprej  

  izrac(G,0).
```

```
daj(0).  

daj(Y1) :-  

  daj(Y),  

  Y1 is Y + 1.
```

4.28.

% prehodi v obliki: pr((Stanje,Znak) - (Novo\_st,N\_znak,Smer))  
 % b = blank, d = desno, l = levo

```
turing(Prehodi,Stanje,Koncno,Vhod):-  

  turing(Prehodi,Stanje,Koncno,[],Vhod).
```

% predzadnji argument je obrnjen del besede levo od glave  
 % zadnji argument je del besede nad glavo in desno od glave

```
turing(Prehodi,Koncno,Koncno,_,[X|_]) :-  

  not(element(pr((Koncno,X) -_),Prehodi)),  

  write('Vhodno besedo je sprejel').  

turing(Prehodi,Koncno,Koncno,_,[]) :-  

  not(element(pr((Koncno,b) -_),Prehodi)),  

  write('Vhodno besedo je sprejel').  

turing(Prehodi,Stanje,Koncno,_,[X|_]) :-  

  Stanje \= Koncno,  

  not(element(pr((Stanje,X) -_),Prehodi)),  

  write('Vhodno besedo ni sprejel').  

turing(Prehodi,Stanje,Koncno,_,[]) :-  

  Stanje \= Koncno,  

  not(element(pr((Stanje,b) -_),Prehodi)),  

  write('Vhodno besedo ni sprejel').  

turing(Prehodi,Stanje,K,Levo,[X|Desno]) :-  

  element(pr((Stanje,X) - (Stanje1,d,Y)),Prehodi),  

  turing(Prehodi,Stanje1,K,[Y|Levo],Desno).
```

1 + 2 + 3 + 4



24

```
turing(Prehodi, Stanje, K, Levo, []):-  
    element(pr((Stanje, b) - (Stanje1, d, Y)), Prehodi),  
    turing(Prehodi, Stanje1, K, [Y | Levo], []).  
turing(Prehodi, Stanje, K, [X1 | Levo], [X2 | Desno]):-  
    element(pr((Stanje, X2) - (Stanje1, l, Y)), Prehodi),  
    turing(Prehodi, Stanje1, K, Levo, [X1, Y | Desno]).  
turing(Prehodi, Stanje, K, [X | Levo], []):-  
    element(pr((Stanje, b) - (Stanje1, l, Y)), Prehodi),  
    turing(Prehodi, Stanje1, K, Levo, [X, Y]).  
turing(Prehodi, Stanje, K, [], [X2 | Desno]):-  
    element(pr((Stanje, X2) - (Stanje1, l, Y)), Prehodi),  
    write('Glava usla cez levi rob traku').  
turing(Prehodi, Stanje, K, [], []):-  
    element(pr((Stanje, b) - (Stanje1, l, Y)), Prehodi),  
    write('Glava usla cez levi rob traku').
```

## LOGIČNI PROBLEMI IN IGRE

1.

```
:- op(200, fy, [not, vsak_x, eksistira_x]).  
:- op(300, xfy, & ).  
:- op(400, xfy, \vee ).  
:- op(500, xfx, => ).  
:- op(600, xfx, <=> ).
```

```
wff(F) :- atomic(F).  
wff(not F) :- wff(F).  
wff(F & G) :- wff(F), wff(G).  
wff(F \vee G) :- wff(F), wff(G).  
wff(F => G) :- wff(F), wff(G).  
wff(F <=> G) :- wff(F), wff(G).  
wff(vsak_x F) :- wff(F).  
wff(eksistira_x F) :- wff(F).
```

2.

% za def. operatorjev glej nal. 5.1 (brez kvantifikatorjev)

```
zadovolji(true).  
zadovolji(X & Y) :- zadovolji(X), zadovolji(Y).  
zadovolji(X v Y) :- zadovolji(X); zadovolji(Y).  
zadovolji(X => Y) :- zadovolji(Y v not X).  
zadovolji(X <=> Y) :- zadovolji((Y => X) & (X => Y)).  
zadovolji(not X) :- ne_zadovolji(X).
```