

Do konca vyzracunajte naslednje izraze v  $\lambda$ -vratunu

$$1.1 \quad (\lambda f x. f(fx))(\lambda R. RR) =$$

$$= (\lambda x. (\lambda R. RR))((\lambda R. RR)x) =$$

$$= (\lambda x. (\lambda R. RR))(xx) =$$

$$= \underline{(\lambda x. xx(xx))}$$

Evaluacija s PLZoo:

lambda > : eager

lambda > : deep

$$2.1 \quad (\lambda f g x. f(gx))(\lambda R. t(tR))t =$$

$$= (\lambda g x. (\lambda R. t(tR))(gx))t =$$

$$= (\lambda g x. t(t(gx)))t =$$

$$= \underline{(\lambda x. t(t(tx)))}$$

$$3.1 \quad ((\lambda x. x)(\lambda f g x. gx))WW =$$

$$= (\lambda f g x. gx)WW =$$

$$= \underline{N}$$

Dovolenoj izračunajte naslednje izraze v  $\lambda$ -računu

1.1  $(\lambda f x. f(f x)) (\lambda R. \text{xx}) =$

$$= (\lambda x. (\lambda z. R z) (\lambda z. R z) x) =$$

$$= (\lambda x. (\lambda z. R z) (x x)) =$$

$$= \underline{(\lambda x. x x (x x))}$$

Evaluacija s PLZoo:

lambda > : larger

lambda > : deep

2.1  $(\lambda f g x. f(g x)) (\lambda R. t(t R)) t =$

$$= (\lambda g x. (\lambda R. t(t R)) (g x)) t =$$

$$= (\lambda g x. t(t g x)) t =$$

$$= \underline{(\lambda x. t(t(t x)))}$$

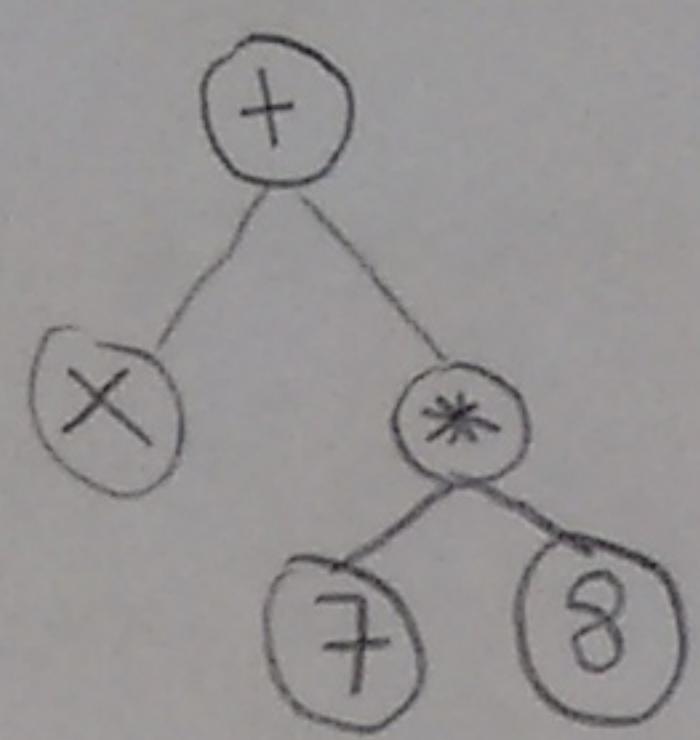
3.1  $((\lambda x. x) (\lambda f g x. \alpha f)) w w w =$

$$= ((\lambda f g x. \alpha f)) w w w =$$

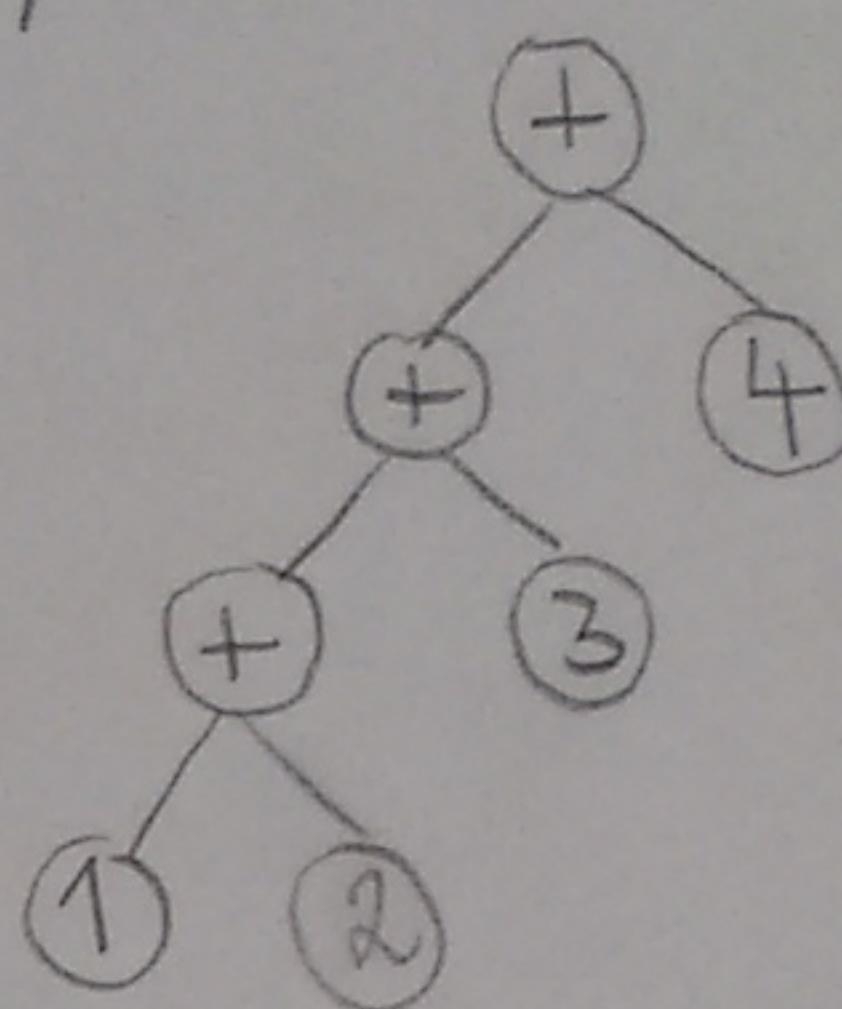
$$= \underline{w}$$

1.] Petvori je konkretne v abstrakte sintakso:

(a)  $x + 7 * 8$

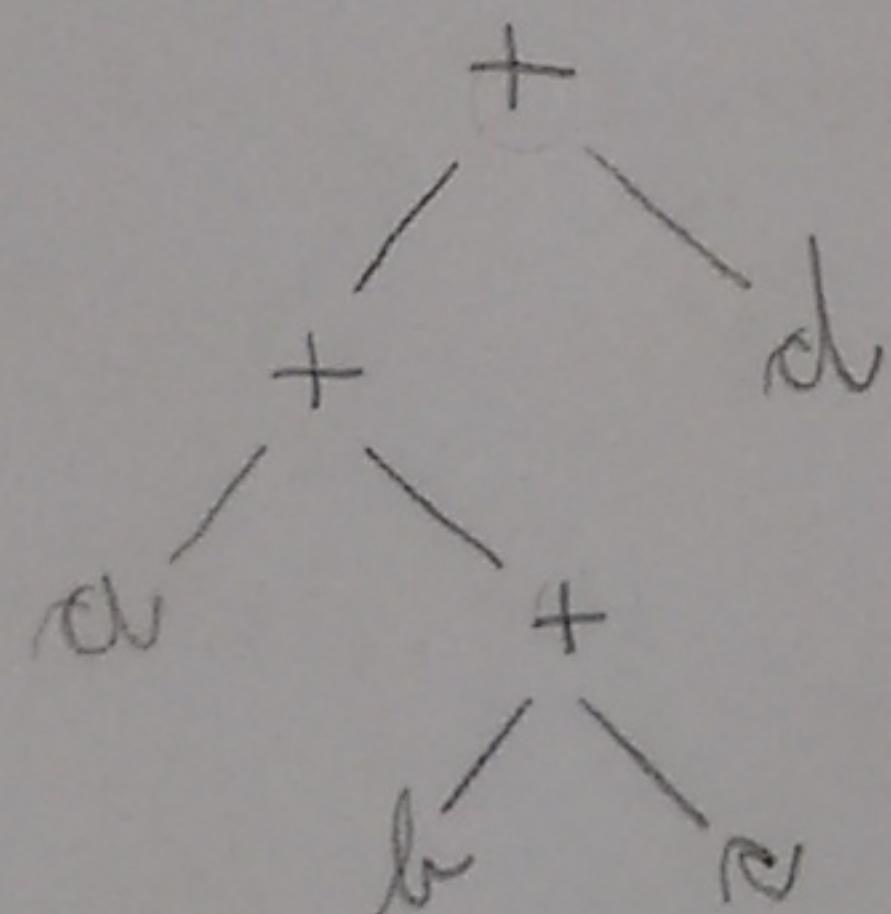


(b)  $1 + 2 + 3 + 4$

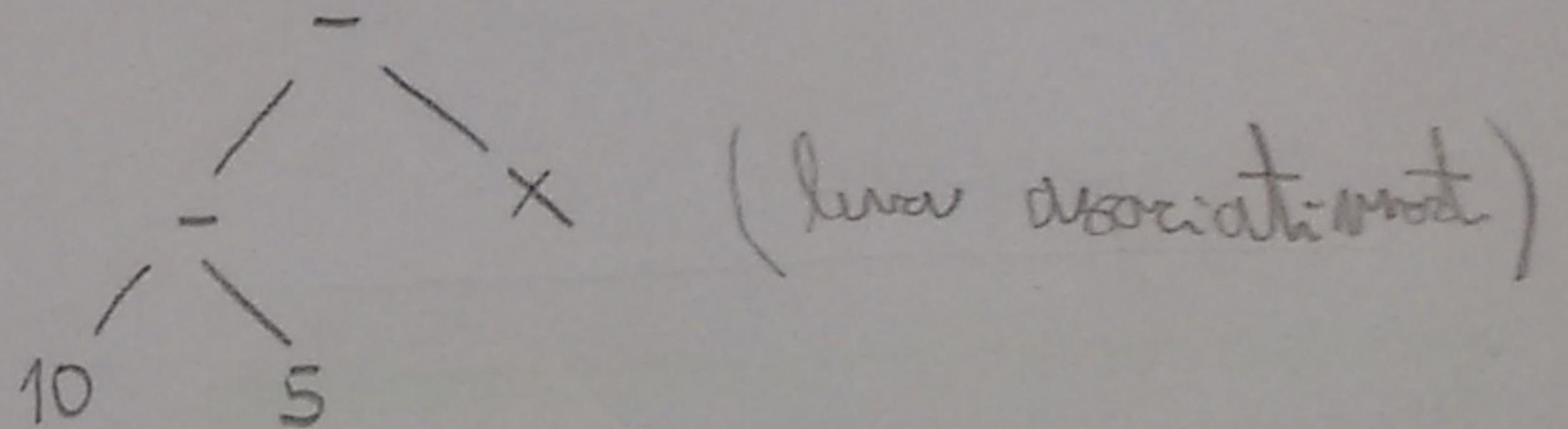


(leva asociativnost)

(c)  $a + (b + c) + d$



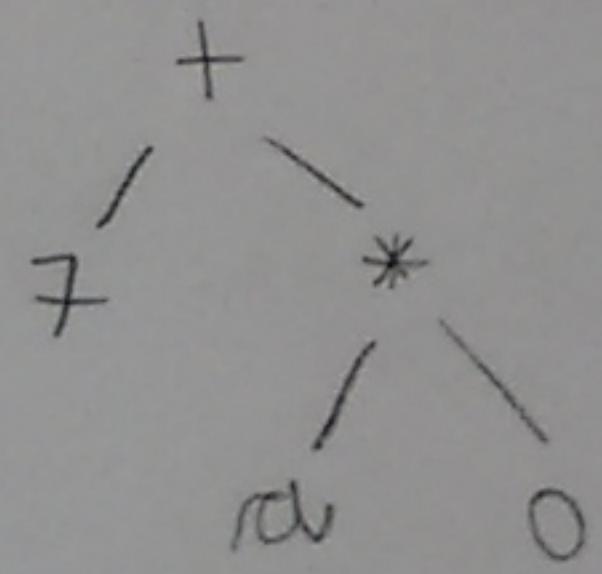
(d)  $10 - 5 - x$



(leva asociativnost)

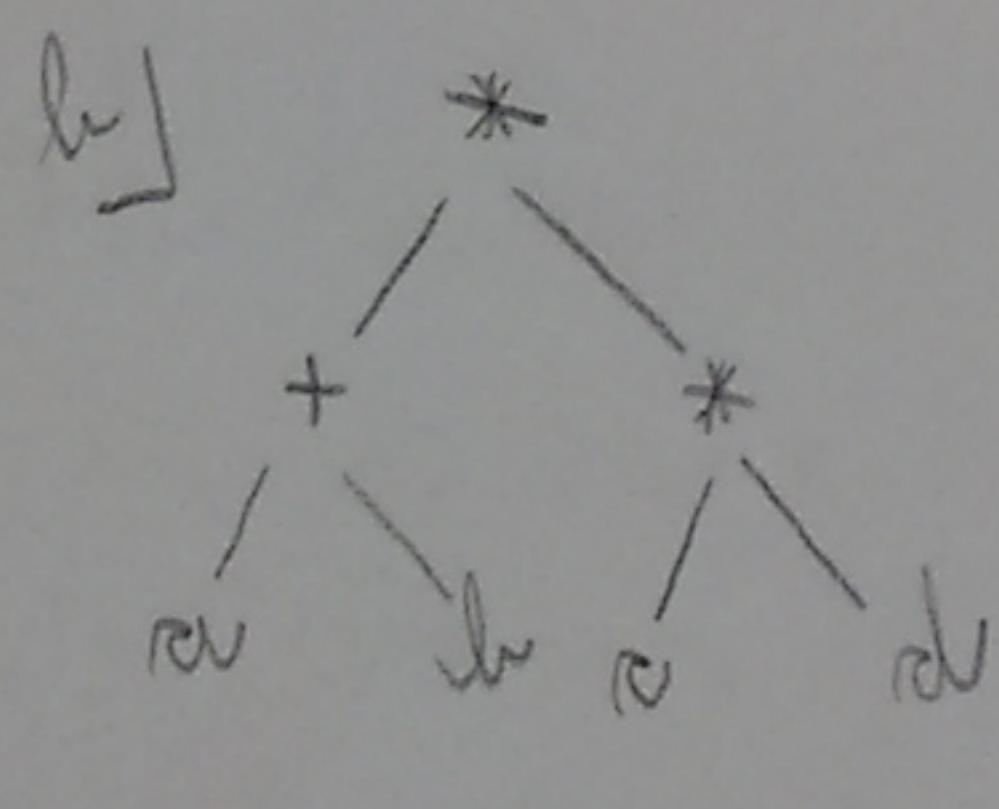
2.] Petvori je abstrakte v konkretne sintakso:

(a)



$7 + a * 0$

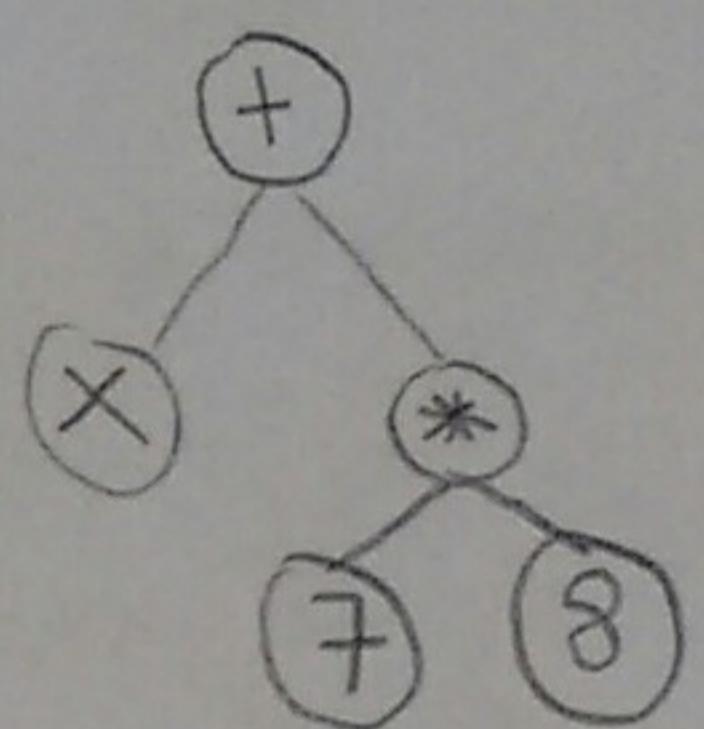
(b)



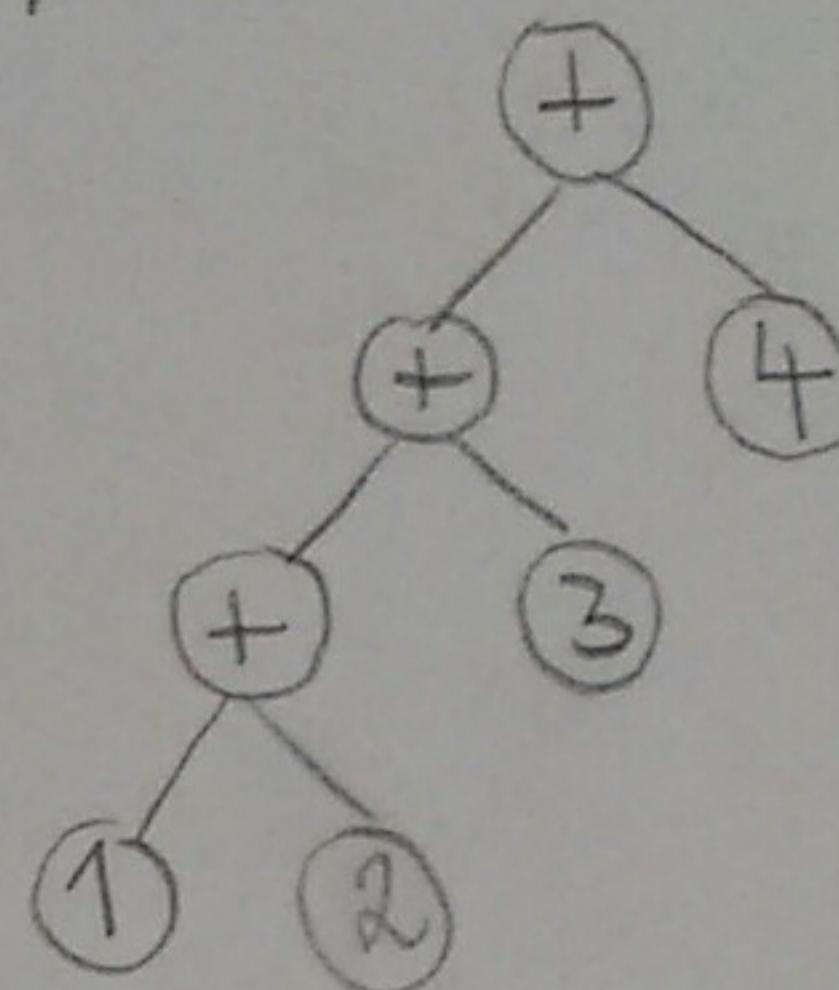
$(a+b)*(a*c+d)$

## 1.] Petrowi iz konkretno v abstraktno sintaksu:

a)  $x + 7 * 8$

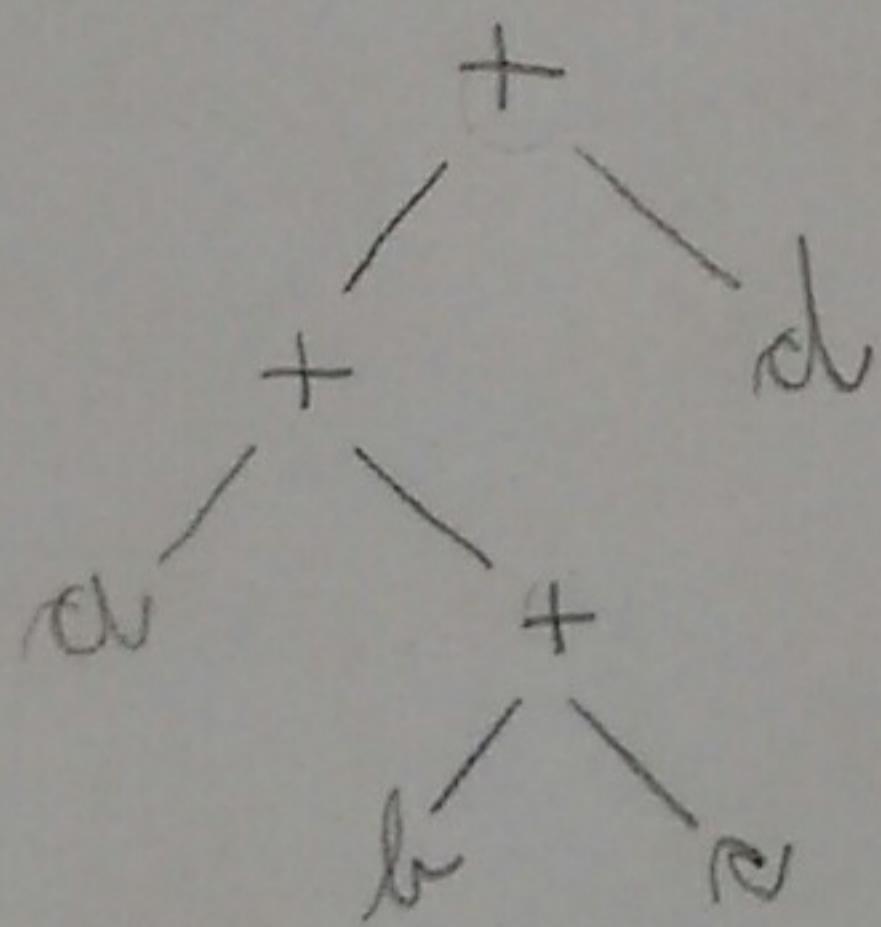


b)  $1 + 2 + 3 + 4$

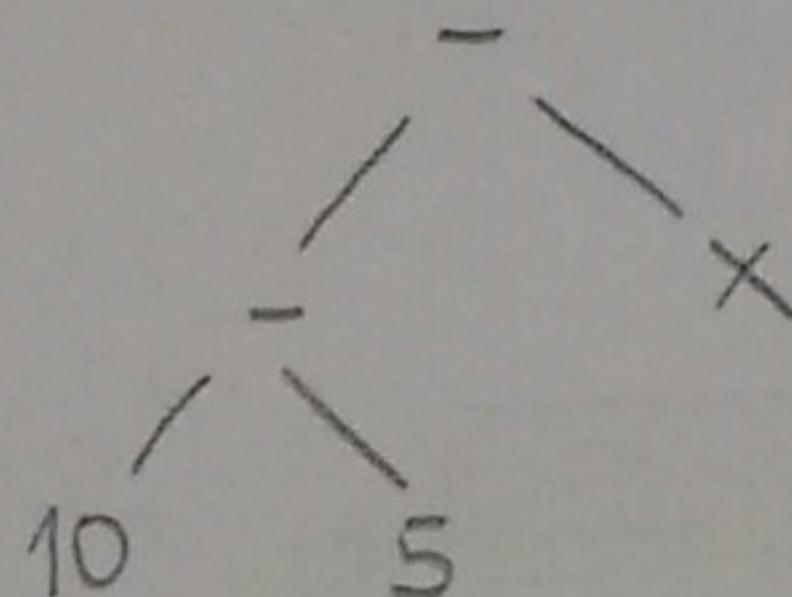


(leva asociativnost)

c)  $a + (b + c) + d$



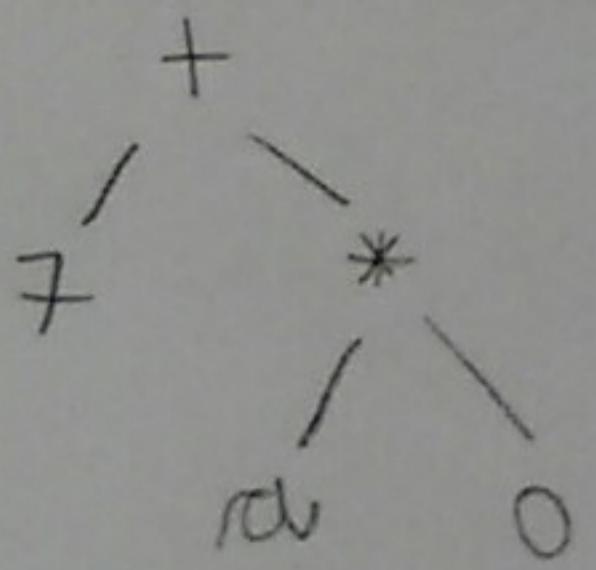
d)  $10 - 5 - x$



(leva asociativnost)

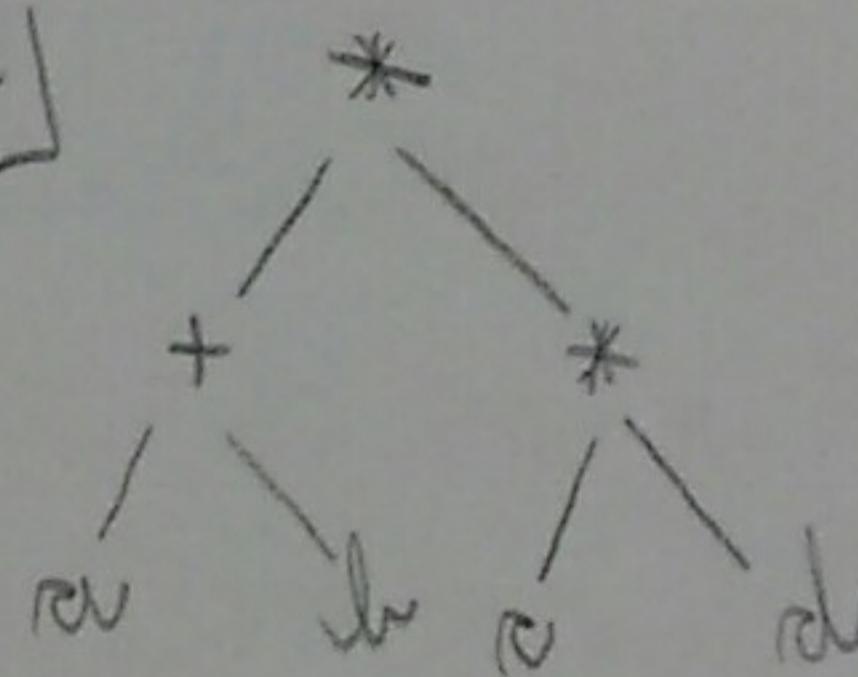
## 2.] Petrowi iz abstraktno v konkretno sintaksu:

a)



$7 + a * 0$

b)



$(a+b)*(c+d)$

uporabo samo znanih pojmov. Temu delu definicije pravimo tudi *rob*.  
definicije je definiranje pojma pri neki vrednosti rekurzijske spremenljivke pojmov in z uporabo pojma samega pri manjših vrednostih rekurzijske spremenljivke.  
način je pojem definiran za vse vrednosti rekurzijske spremenljivke.

Za zgled si oglejmo definicijo procedure fak(N,Fak), tako da je Fak = N!

fak(0,1).  
fak(N,Fak) :-  
    N1 is N-1,  
    fak(N1,Fak1),  
    Fak is N\*Fak1.

$$N! = \underbrace{1 \cdot 2 \cdot 3 \cdots N-1}_{\text{FAK1}} * N$$

Pri tem je "is" vgrajeni operator, ki povzroči izračun aritmetičnega izraza prilagoditev rezultata enostavnemu izrazu levo od sebe, če je to možno.

V našem primeru je rekurzijska spremenljivka kar prvi argument (N).

naslednik(Naslednik,Prednik) :-

    otrok(Naslednik,Prednik).

naslednik(Naslednik,Prednik) :-

    otrok(Naslednik,Vmesni),

    naslednik(Vmesni,Prednik).

V tem primeru je rekurzija spremenljivka dejansko število vmesnih prednikov med osebama, ki ni v naprej podano in tudi ne nastopa kot argument v proceduri. Mejni pogoj nastopi takrat, ko ni nobenega vmesnega naslednika, ampak sta osebi kar v relaciji otrok. Drugi stavek pa definira relacijo naslednik z uporabo relacije same vendar med osebama, med katerima je manjše število vmesnih prednikov, kar zopet zagotavlja, da se bo program zaustavil.

ko ni nobenega vmesnega naslednika, ampak sta osebi kar v relaciji otrok. Drugi stavek pa definira relacijo naslednik z uporabo relacije same vendar med osebama, med katerima je manjše število vmesnih prednikov, kar zopet zagotavlja, da se bo program zaustavil.

Pogosto je potrebno definirati več robnih pogojev. Primer je procedura fib(N,F) za izračun N-tega člena Fibonaccijevega zaporedja, ki ima po definiciji dva robna pogoja: prva dva elementa sta enaka 1, vsak naslednji člen pa je vsota prejšnjih dveh.

fib(0,1).  
fib(1,1).  
fib(N,F) :-  
    N1 is N - 1,  
    fib(N1,F1),  
    N2 is N - 2,  
    fib(N2,F2),  
    F is F1 + F2.

$$f(0) = 1$$

$$f(1) = 1$$

Pogosta je uporaba rekurzije pri interaktivnem delu (v dialogu z uporabnikom). Robni pogoj je v tem primeru ukaz uporabnika, ki pomeni zahtevo za končanje (npr. "koniec"):

```
fib(N1,F1),  
N2 is N - 2,  
fib(N2,F2),  
F is F1 + F2.
```

Pogosta je uporaba rekurzije pri interaktivnem delu (v dialogu z uporabnikom). Robni je v tem primeru ukaz uporabnika, ki pomeni zahtevo za končanje (npr."konec"):

delaj :-

```
write(' vtipkaj ukaz: '),
read(Ukaz),
(Ukaz = konec ;
izvrsti(Ukaz),
delaj).
```

Procedura zahteva od uporabnika, naj vtipka ukaz. če je vtipkani ukaz "konec", po izvajanje konča, sicer se izvrši ukaz in ponovno celotni postopek.

## 3.2 Seznamni

*D&L*

brisi( \_,[],[] ).

brisi( Element,[Element | Rep],Rep ).

brisi( Element,[Glava | Rep],[Glava | Rep1] ):-

brisi( Element,Rep,Rep1 ).

Rekurzija teče po dolžini prvega seznama (drugega argumenta). V tej proceduri nastopata dva robna pogoja. Prvi nastopi, ko je seznam prazen (prvi stavek). Ta pogoj je potreben za primer, ko elementa ni v prvotnem seznamu. Drugi pogoj nastopi, če element je v seznamu (drugi stavek). Ker ne vemo v naprej, kakšna bo dolžina seznama pri tem robnem pogoju, je robni pogoj podan z lastnostjo seznama (to je, da se element nahaja v glavi seznama).

dva redna pogoja. Prvi nastopi, ko je seznam prazen (prvi stavek). Ta pogoj je potreben za primer, ko elementa ni v prvotnem seznamu. Drugi pogoj nastopi, če element je v seznamu (drugi stavek). Ker ne vemo v naprej, kakšna bo dolžina seznama pri tem robnem pogoju, je robni pogoj podan z lastnostjo seznama (to je, da se element nahaja v glavi seznama).

Najbolj pogosti operaciji nad seznami sta iskanje elementa v seznamu s proceduro element(X,Sez), ki preveri, če je X v seznamu Sez:

*member*

```
element(X,[X|_]).      not element(X,[X|_]) .  
element(X,[_|Rep]):-    } not element(X,[_|Rep1]) :- not element(X,Rep1) .  
          element(X,Rep).
```

in stik dveh seznamov s proceduro stik(Seznam1,Seznam2,Seznam3), ki preveri, če je Seznam3 stik seznamov Seznam1 in Seznam2, ozioroma stakne Seznam1 in Seznam2 v Seznam3.

*Conv*

```
stik([],Seznam,Seznam).  
stik([Glava|Rep],Seznam2,[Glava|Rep3]) :-  
      stik(Rep,Seznam2,Rep3).
```

Rekurzija teče po dolžini prvega seznama. Robni pogoj nastopi takrat, ko je prvi seznam prazen (dolžina seznama je enaka 0). če pa seznam ni prazen, je relacija stik definirana sama s seboj nad krajšim prvim seznamom, kar zagotavlja, da se bo izvajanje ustavilo pri robnem

Rekurzija teče po dolžini prvega seznama. Robni pogoj nastopi takrat, ko je prvi seznam prazen (dolžina seznama je enaka 0). če pa seznam ni prazen, je relacija stik definirana sama s seboj nad krajšim prvim seznamom, kar zagotavlja, da se bo izvajanje ustavilo pri robnem pogoju.

Pogosta operacija nad seznamom je pretvorba v seznam elementov, ki jih dobimo tako, da izvedemo neko transformacijo nad vsakim elementom prvotnega seznama. če želimo npr. iz seznama števil dobiti seznam njihovih kvadratov, lahko definiramo proceduro kvadrati(Seznam,Seznam\_kvadratov):

```
kvadrati([],[]).  
kvadrati([I|Rep],[Kv|Kvrep]):-  
    Kv is I*I,  
    kvadrati(Rep,Kvrep).
```

V splošnem imamo dvomestno proceduro P, ki nam naredi željeno transformacijo nad elementom seznama. Tako lahko definiramo splošno proceduro spremeni(Seznam,P,Seznam1), ki spremeni seznam Seznam v Seznam1 z uporabo dvomestnega procedure P:

(leva asociativnost)

28

spremeni([],\_,[]).  
spremeni([Glava | Rep], P, [Glava1 | Rep1]) :-  
    Tr = .. [P, Glava, Glava1],  
    call(Tr),  
    spremeni(Rep, P, Rep1).

VI  
Vgrajena procedura call(Cilj) zahteva izpolnitev cilja Cilj. Predikata "call/1" in " = ..'2" sta *metalogična*, saj omogočata operacije nad logičnimi konstrukti (v našem primeru konstrukcijo literala).

Tipičen pristop k transformaciji seznama je izvedba transformacije najprej nad repom in zatem ustrezna obravnava glave prvotnega seznama. Za zgled si poglejmo proceduro

call(1),  
spremeni(Rep,P,Rep1).

VI

Vgrajena procedura call(Cilj) zahteva izpolnitev cilja Cilj. Predikata "call/1" in " $=..^2$ " sta metalogična, saj omogočata operacije nad logičnimi konstrukti (v našem primeru konstrukcijo literala).

Tipičen pristop k transformaciji seznama je izvedba transformacije najprej nad repom in zatem ustreznega obravnavanja glave prvotnega seznama. Za zgled si poglejmo proceduro obrni(Seznam,Obrnjen), ki obrne seznam Seznam v Obrnjen:

```
obrni([],[]).  
obrni([Glava|Rep],Obrnjen):-  
    obrni(Rep,Obr_rep),  
    stik(Obr_rep,[Glava],Obrnjen).
```

V tem primeru najprej obrnemo rep seznama in nato postavimo glavo prvotnega seznama na konec obrnjenega repa. Ta operacija je relativno zamudna zaradi klica procedure stik.

Drugi pristop k transformaciji seznama je sprotna graditev seznama, tako da naredimo ustrezeno operacijo nad glavo seznama in nato nadaljujemo s transformacijo nad repom. V tem primeru je na začetku grajeni seznam prazen, med izvajanjem raste in na koncu je potrebno rezultat prenesti kot poseben argument. Poglejmo, kako bi na ta način rešili obračanje

Ta pro  
zgodi,

Pri te  
Tretji  
pono

3.3

Avto  
v sle  
defin  
posk

stik(Obr\_rep,[Glava],Obrnjen).

V tem primeru najprej obrnemo rep seznama in nato postavimo glavo prvotnega seznama na konec obrnjenega repa. Ta operacija je relativno zamudna zaradi klica procedure stik.

Drugi pristop k transformaciji seznama je sprotna graditev seznama, tako da naredimo ustrezeno operacijo nad glavo seznama in nato nadaljujemo s transformacijo nad repom. V tem primeru je na začetku grajeni seznam prazen, med izvajanjem raste in na koncu je potrebno rezultat prenesti kot poseben argument. Poglejmo, kako bi na ta način rešili obračanje seznamov:

~~obrni~~  
obrni(Seznam,Obrnjen) :- obrni(Seznam,[],Obrnjen).

obrni([Glava | Rep],Trenutni,Rezultat):-  
    obrni(Rep,[Glava | Trenutni],Rezultat).  
obrni([],Rezultat,Rezultat).

Pri tej rešitvi smo morali dodati poseben argument (drugi argument), ki predstavlja del seznama, ki smo ga do sedaj že obrnili. Na začetku je ta seznam prazen. Vsakič ko obdelamo en element prvotnega seznama, dodamo v trenutni rezultat nov element. Ko obdelamo vse elemente prvotnega seznama (ko je prvi argument prazen seznam), je trenutni seznam tudi končni rezultat (zadnji stavek).

Pri tem je  
Tretji arg  
ponovno

### 3.3 Kon

Avtomats  
v slepo v  
definiran  
poskušal  
takega el  
element :

To nam d  
pri prver  
to je cilja  
ki poišče

če pri kl

## UVOD V PROLOG

```
povezana(Vozel1,Vozel2):-  
    povezava(Vozel1,Vozel2).  
povezana(Vozel1,Vozel2):-  
    povezava(Vozel1,Vmesni),  
    povezana(Vmesni,Vozel2).
```

Ta program bi deloval, če v našem grafu ne bi bilo ciklov. če pa obstaja v grafu cikel, se lahko zgodi, da se program nikoli ne ustavi. Temu se izognemo z uvedbo seznama pregledanih vozelic.

```
povezana(Vozel1,Vozel2 ):- isci_pot(Vozel1,Vozel2,[]).
```

```
isci_pot(Vozel1,Vozel2,[]):-
```

povezana(Vozel1,Vozel2):-

povezava(Vozel1,Vozel2).

povezana(Vozel1,Vozel2):-

povezava(Vozel1,Vmesni),

povezana(Vmesni,Vozel2).

Ta program bi deloval, če v našem grafu ne bi bilo ciklov. Če pa obstaja v grafu cikel, se lahko zgodi, da se program nikoli ne ustavi. Temu se izognemo z uvedbo seznama pregledanih vozlov:

povezana(Vozel1,Vozel2 ):- isci\_pot(Vozel1,Vozel2,[]).

isci\_pot(Vozel1,Vozel2,\_):-

povezava(Vozel1,Vozel2).

isci\_pot(Vozel1,Vozel2,Sled):-

povezava(Vozel1,Vmesni),

not(element(Vmesni,Sled)),

isci\_pot(Vmesni,Vozel2,[Vozel1 | Sled]).

Pri tem je `not(Cilj)` vgrajena procedura, katere klic uspe, če Cilj ne uspe (glej poglavje 3.3). Tretji argument predstavlja seznam vozlov, ki smo jih že prehodili in v katere ne smemo ponovno priti.

element poišče samo prvo pojavitev danega elementa v seznamu.

To nam omogoča vgrajena procedura *rez* (*cut*), ki jo v prologu označujemo s klicajem (!). Rez pri prvem klicu vedno uspe, pri avtomatskem vračanju pa povzroči neuspeh *nadrejenega cilja*, to je cilja, ki se je prilagodil glavi stavki, v katerega telesu se nahaja rez. Proceduro element1, ki poišče samo prvo pojavitev danega elementa v seznamu, lahko zapišemo sedaj:

```
element1(X,[X|_]):- !.  
element1(X,[_|Sez]):-  
    element1(X,Sez).
```

če pri klicu prvotne procedure element zahtevamo dodatne rešitve (v dialogu s prologovim interpretterjem to dosežemo tako, da vtipkamo podpičje), jih bo prolog poiskal:

```
?-element(f(X),[f(1),g(2),h(3),f(4),f(5),h(6)]).  
X = 1;  
X = 4;  
X = 5;  
no
```

?-element1(f(X),[f(1),g(2),h(3),l(4),-c]).

X = 1;

no

V zgornjem primeru z uporabo reza programa opisno nismo spremenili (prva rešitev je vedno enaka). Postopkovno pa se je izvajanje spremenilo. Včasih je koristno spremeniti tudi opisni pomen programa, zaradi večje učinkovitosti. Za zgled si poglejmo proceduro max(Seznam,Max), ki poišče največji element seznama števil:

max([X],X).

max([X|Rep],X):-

    max(Rep,Max),

    Max < X.

% ' = <' je vgrajena procedura, glej dodatek 2

max([X|Rep],Max):-

    max(Rep,Max),

    Max > X.

Procedura poišče najprej največji element v repu seznama. Če je glava seznama večja od največjega števila v repu, potem je to rezultat (drugi stavek), sicer pa je rezultat kar največji element v repu (tretji stavek). V drugem primeru smo morali dvakrat računati največji element v repu, kar je prav gotovo nesmiselno. Bralec je mogoče opazil, da bo program v splošnem hitrejši, če zamenjamo drugi in tretji stavek, saj je večja verjetnost, da se nahaja največji element v repu seznama kar v drugem.

max( $R_C P$ ,  $X_{MAX}$ ),

Max > X.

Procedura poišče najprej največji element v repu seznama. če je glava seznama večja od največjega števila v repu, potem je to rezultat (drugi stavek), sicer pa je rezultat kar največji element v repu (tretji stavek). V drugem primeru smo morali dvakrat računati največji element v repu, kar je prav gotovo nesmiselno. Bralec je mogoče opazil, da bo program v splošnem hitrejši, če zamenjamo drugi in tretji stavek, saj je večja verjetnost, da se nahaja največji element v repu seznama kot v glavi, a to vseeno ne reši našega problema. Dvojnemu delu se lahko izognemo z uporabo reza:

max([X], X).

max([X | Rep], Max):-

    max(Rep, Max),

    Max >= X, !.

max([X | \_], X).

Zavedati se moramo, da smo sedaj spremenili opisni pomen tretjega stavka in da zamenjavo drugega in tretjega stavka sedaj ni dovoljena, ker tretji stavek opisno ni pravilen.

Z uporabo reza lahko direktno implementiramo *if-then-else* konstrukt iz pascala:

if then else( $P_C Q$ ,  $R$ ,  $S$ ) :-

če klic  
če pa l  
v zadn  
in zara  
prolog  
oseba,  
že opr

Sedaj  
spremi  
Prolog  
in vse  
vprašan

odgovo

```
max(Rep,Max),  
Max >= X, !.  
max([X | _],X).
```

Zavedati se moramo, da smo sedaj spremenili opisni pomen tretjega stavka in da zamenjava drugega in tretjega stavka sedaj ni dovoljena, ker tretji stavek opisno ni pravilen.

Z uporabo reza lahko direktno implementiramo *if-then-else* konstrukt iz pascala:

```
if_then_else(P,Q,_) :- P,! ,Q.  
if_then_else(_,_,R) :- R.
```

Izpolnitev cilja *if\_then\_else(P,Q,R)* pomeni izvršitev stavka 'if P then Q else R', kjer so P,Q in R cilji.

če uporabimo rez znotraj procedure "call(Cilji)", ki povzroči izpolnitev Ciljev, lahko preprečimo ponovno izpolnjevanje samo ciljev znotraj stavka call. Tako bo rez v stavku

```
x :- a, b, call((c,d,! ,e)), f, g.
```

Sedaj bo pro  
spreminjati v  
Prolog pri izv  
in vse ostalo  
vprašanje

odgovoril z "

### 3.4 Večsn

Lepa lastnos  
potem ta fun  
Y = f(X), kar

Lahko je na vhodu Y in določimo samo odgovor "da" ali "ne", lahko pa uporabljamo proceduro `stik` kot generator vseh parov X in Y, ki so v relaciji f (glej sliko 3.2). Tipičen primer take procedure je procedura roditelj, ki smo jo definirali v poglavju 1. Vsi parametri procedure so hkrati vhodni in izhodni, kar je odvisno od načina uporabe procedure.

če si pogledamo proceduro `stik` iz pogl. 3.2, hitro ugotovimo, da jo lahko uporabljamo na različne načine. Npr. iskanje elementa v seznamu lahko realiziramo z:

`element(Element,Seznam) :- stik(, [Element | _], Seznam).`

Razstavljanje seznama na dva poljubna podseznama lahko realiziramo z:

`razstavi(Seznam, Sez1, Sez2) :- stik(Sez1, Sez2, Seznam).`

Brisanje elementa iz seznama lahko realiziramo z:

`brisi(Element, Seznam, Seznam1) :-  
stik(Sez1, [Element | Sez2], Seznam),  
stik(Sez1, Sez2, Seznam1).`

Vendar pa večsmernost ne velja vedno in tudi ni vedno potrebna. Tako npr. procedure, ki uporabljajo vgrajeno proceduro "is" za aritmetične operacije, niso večsmerne ampak strogo enosmerne. Operator "is" povzroči izračun aritmetičnega izraza desno od operatorja in obratni smeri

Večsm  
proce  
prakti  
proce

pote  
elen

odg

Razstavljanje seznama za praktične procedур

razstavi(Seznam, Sez1, Sez2) :- stik(Sez1, Sez2, Seznam).

Brisanje elementa iz seznama lahko realiziramo z:

brisi(Element, Seznam, Seznam1) :-  
stik(Sez1, [Element | Sez2], Seznam),  
stik(Sez1, Sez2, Seznam1).

Vendar pa večsmernost ne velja vedno in tudi ni vedno potrebna. Tako npr. procedure, ki uporabljajo vgrajeno proceduro "is" za aritmetične operacije, niso večsmerne ampak strogo enosmerne. Operator "is" povzroči izračun aritmetičnega izraza desno od operatorja in prilagoditev rezultata enostavnemu izrazu levo od operatorja, če je to možno. V obratni smeri to ne gre. če npr. zahtevamo

?- X is 2 + 18/5.

dobimo rezultat  $X = 5.6$ . Vendar če bi hoteli :

?- 5.6 is A + B/C.

bi prolog moral najti vse trojice A, B in C, ki v izrazu  $A + B/C$  dajo rezultat 5.6, kar je že tako preprostih izrazov. To je večsmernost pri aritmetiki.

potem je element

odgovor

kar je s

Namre

elemen

prilaga

argum

zapiše

Večsmernost prologa se pokvari tudi z uporabo *metalogičnih predikatov*. To so vgrajene procedure, ki opisujejo lastnosti konstruktov matematične logike. Na voljo so programerju iz praktičnih razlogov. Tu si poglejmo samo zgled uporabe "metalogike". Če definiramo proceduro element(Element,Seznam) takole:

```
element(Element,[Element | _]).  
element(Element,[_|Rep]):-  
    element(Element,Rep).
```

potem je dejanski pomen procedure iskanje takega elementa seznama, ki se lahko prilago elementu Element. Na ta način bi prolog na vprašanje

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

odgovoril z

X = a

## II. DEL: ZBIRKA NALOG IZ PROGRAMIRANJA V PROLOGU

### 1 SINTAKTIČNI IN SEMANTIČNI PROBLEMI

1.1. Spremeni v prologove stavke naslednje stavke v naravnem jeziku!

- a) Sonja igra klavir.
- b) Irena igra vse instrumente, ki imajo klaviaturo.
- c) Ali igra Igor kakšen instrument?
- d) Katere instrumente igra Igor?
- e) Boris igra instrumente, ki jih igrata in Mojca in Sonja.
- f) Boris igra vse instrumente, ki jih igrata Mojca in Sonja.
- g) Kdo zna igrati kitaro in harmoniko?
- h) (\*) Ali zna kdo igrati kitaro in harmoniko?

1.2. Kako bi se znebil dvoumnosti pri naslednjih dejstvih, ki tvorijo isto proceduro a predstavljajo različne relacije?

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

1.3. (\*) Prevedi naslednje stavke v prologu v naravni jezik:

- a) `ima_rad('Ana','Niko').`
- b) `znak( = => ,implikacija).`
- c) `znak( & ,konjunkcija).`
- d) `znak( V ,disjunkcija).`
- e) `znak( * ,Zvezdica).`
- f) `lahko_porabi(Ana,10000) :- os_doh(Ana,OD),OD > 10000.`
- g) `lahko_porabi(niko,_100) :- os_doh(niko,_200),_200 > _100.`
- h) `ima_rad(Ana,Niko).`
- i) `?- ima_rad(Ana,Niko).`
- j) `?- ima_rad(Ana,_).`
- k) `?- not(ima_rad(Ana,'Niko')).`

1.4. Stavki, v katerih konjunkcija nastopa v kombinaciji z disjunkcijo so težko berljivi. Kako bi izboljšal berljivost naslednjih stavkov?

- a) `stara_mama(X,Y) :- mama(X,Z),(mama(Z,Y);oce(Z,Y)).`
- b) `premagal(X,Y) :- predal(Y,X);tocke(X,N),tocke(Y,M),N > M.`
- c) `med(C,A,B) :- levo_od(A,C),desno_od(B,C);  
levo_od(B,C),desno_od(A,C).`
- d) `ima_rad(Fant,Dekle) :- lepa(Dekle),bogata(Dekle);  
dobra(Dekle),inteligentna(Dekle).`
- e) `pot_domov(Pisarna,Dom) :- pes(Pisarna,Postaja),  
(vlak(Postaja,Postaja1);avtobus(Postaja,Postaja1)),  
pes(Postaja1,Dom).`

1.5. Kaj je glava in kaj rep naslednjih seznamov?

- a) [a,b,c,d] a
- b) [a,b|X]
- c) [X|Y]
- d) [X|[Y]]
- e) [a|b]
- f) [X]
- g) X
- h) [[a,b,c|d],e|[f,g|X]]
- i) []
- j) [a|[b|[c|[d]]]]

1.6. Kako se prilagodijo (če se prilagodijo) naslednji pari struktur?

- a) datum(D,M,1983) in datum(D1,maj,Y1)
- b) trikotnik(tocka(1,1),A,tocka(2,3)) in trikotnik(X,tocka(4,Y),tocka(2,Y))
- c) X in [a,b,c]
- d) [a,b|X] in [a,b,c]
- e) [a,X,Y] in [a,f(b),c]
- f) [X|[Y]] in [1,2,3]
- g) [X,Y] in [1,2,3]
- h) [X,Y] in [a,[b,c]]
- i) [a,b,c|X] in [a,b,c]
- j) [X|Y] in [a]
- k) [X|Y] in []
- l) [X,Y,Z] in [a|[b|[c]]]
- m) [X,Y,Z] in [a,b|c]
- n) [2|X] in X
- o) [X|Y] in X

1.7. Imejmo dejstvo  $c([X|Y], X, Y)$ . Kaj bo prolog odgovoril na naslednja vprašanja?

- a)?-c([a,b,c,d],G,R).
- b)?-c(L,a,[b,c,d]).
- c)?-c(L,[a],[a]).
- d)?-c([[]|R],G,G).
- e)?-c([a,b|R],G,[b,c,d]).
- f)(\*)?-c(L,a,b).
- g)(\*)?-c(L,a,T),c(T,b,T1),c(T1,c,[]).

1.8. Kaj bo prolog odgovoril na naslednja vprašanja?

- a) ?- X is 17 \* 3 - 5.
- b) ?- 5 is X - 7.
- c) ?- 5 \* 3 is 15.
- d) ?- 15 is 5 \* 3.
- e) ?- 2 >= 2.
- f) ?- X =:= 5 \* 3.
- g) ?- 15 - 3 =:= 3 \* 4.
- h) ?- X = 1 + 2.

- i)  $?- X == 1 + 2.$   
j)  $(*) ?- X = 1 + 2, 1 + 2 * 3 = X * 3.$   
k)  $(*) ?- X = 5 + 3, Y \text{ is } 2 * X.$   
l)  $(**) ?-(X + Y) + Z = A + (B + C).$   
m)  $(**) ?- X = a + Y, Y = b + Z, Z = c + d.$

1.9. Dan imamo program:

```
ima_rad(metka, tone).
ima_rad(ana, marko).
ima_rad(tone, metka).
ima_rad(tone, ana).
```

Kaj bo prolog odgovoril na naslednja vprašanja?

- a)  $?- \text{ima\_rad}(X, \text{tone}).$   $X = \text{metka};$   
b)  $?- \text{ima\_rad}(X, Y), \text{ima\_rad}(Y, ana).$   $X = \text{metka}; Y = \text{tone};$   
c)  $?- \text{ima\_rad}(X, Y), \text{ima\_rad}(Y, X).$   $X = \text{metka}; Y = \text{tone}$   
d)  $(*) ?- \text{ima\_rad}(X, Y), \text{ima\_rad}(Y, Z), \text{ima\_rad}(Z, W), \text{ima\_rad}(W, X).$   $X = \text{metka}; Y = \text{tone}; Z = \text{metka}; W = \text{tone}$

1.10. (\*) Imejmo dejstvo  $\text{stik}(X-Y, Y-Z, X-Z).$  Kaj bo prolog odgovoril na naslednja vprašanja?

- a)  $?- \text{stik}([a,b,c|X]-X, [d,e|Y]-Y, Z).$   
b)  $?- \text{stik}([a|X]-X, Y, [a,b,c,d,e|Z]-Z).$   
c)  $?- \text{stik}([1,2|X]-X, [3,4|Y]-Y, Z), \text{stik}(Z, [5|W]-W, R).$

1.11. (\*\*) Izrazi v prologu naslednjo izjavo, ki vodi v neskončno regresijo: "če imaš nek cilj, potem moraš poiskati pot do tega cilja, kar je novi cilj".

1.12. Ali so naslednji pari vprašanj ekvivalentni?

- a)  $?- \text{not}(\text{not}(\text{lepo})). \text{in} ?- \text{lepo}.$   
b)  $?- \text{not}(\text{not}(\text{mama}(\text{ana}, \text{tone}))). \text{in} ?- \text{mama}(\text{ana}, \text{tone}).$   
c)  $?- \text{not}(\text{not}(\text{pameten}(X))). \text{in} ?- \text{pameten}(X).$   
d)  $?- \text{not}(\text{not}(X = 2 + 3)). \text{in} ?- X = 2 + 3.$   
e)  $?- \text{not}(X + Y == 2 + 3). \text{in} ?- X + Y \backslash == 2 + 3.$   
f)  $?- \text{not}(X + Y = 2 + 3). \text{in} ?- X + Y \backslash = 2 + 3.$

1.13. (\*\*) Prevedi naslednja vprašanja v prolog. Predpostavi, da imaš program v obliki dejstev 'oseba(Oseba)' in 'instrument(I)'.

- a) Ali je res, da nihče ne mara vina?  
b) Ali imajo vsi radi vino?  
c) Ali je kdo, ki igra vse instrumente?  
d) Ali je kak instrument, ki ga vsi igrajo?  
e) Ali vsi igrajo vse instrumente?

1.14. Dan imamo sledeč program v prologu:

```
a(0 + 1, 1 + 0).
a(1 + X, 1 + Y) :- a(X + 1, 1 + Y).
a(X + 1, 1 + Y) :- a(X, Y).
```

Kaj bo prolog odgovoril na naslednja vprašanja?

- a)?-a(0 + 1 + 1, X).
- b)?-a(1 + 1 + 0 + 1, X).
- c)?-a(1 + 0 + 1, X).
- d)?-a(X, 1 + 1 + 0).

1.15. Kaj dela program iz naloge 1.14. Kateri so regularni izrazi, ki jih sprejme kot prvi argument?

1.16. Dan imamo program :

```
pq(X, Y) :- p(X), q(X), !, p(Y), q(Y).
p(1). p(2). p(3). p(4) :- !. p(5).
q(2). q(3). q(4). q(5). q(6).
```

Kaj odgovori ta program na vprašanja?

- a)?-pq(A, B). % zanima nas prvi odgovor
- b)?-setof(X, p(X), L).
- c)?-setof(A \* B, pq(A, B), L).

1.17. Dan je program:

```
d([], nil).
d([X | L], s(D)) :- d(L, D).
```

Kaj izpiše prolog ob vprašanjih:

- a)?-d([1, 2, 3, 4], X).
- b)?-d(L, s(s(s(nil)))).

1.18. Imamo proceduro:

```
inv(X + Y, Y1 + X1) :- inv(X, X1), inv(Y, Y1).
inv(X, X).
```

Napišite vse odgovore, ki nam jih da prolog na vprašanje:

?-inv(1 + (2 + 3), Z).

1.19. Dan imamo program za računanje večjega od dveh števil

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

Kaj bo prolog odgovoril na naslednji vprašanji?

- a) ?-max(15,13,X).  
 b) ?-max(15,13,X), 2\*X < 30.  
 c) Napiši pravilen program max!

1.20. Prevedi naslednje stavke v prolog:

- a) Jože je človek.
- b) Jože je smrten.
- c) Vsak človek je smrten.
- d) Nina sedi med Ireno in Igorjem.
- e) Janez ima rad vsakogar, ki ima rad vino.
- f) Vsi moški imajo radi vse ženske.
- g) (\*) Vsi moški imajo rad eno žensko.
- h) (\*) Vsi moški imajo rad po eno žensko.
- i) (\*\*) Nekateri ljudje so pametni.
- j) (\*\*) Nobena stvar ni hkrati slaba in dobra.

1.21. (\*\*) Prevedi naslednje stavke v prolog:

- a) Pravilo 1: Konstanta je term.

Pravilo 2: Spremenljivka je term.

Pravilo 3: če je f n-mestni funkcijski simbol in če so  $t_1, \dots, t_n$  termi za  $n \geq 1$ , potem je  $f(t_1, \dots, t_n)$  tudi term.

Terme lahko dobimo samo z uporabo pravil 1,2 in 3.

- b) če je P n-mestni predikatni simbol in so  $t_1, \dots, t_n$  termi za  $n \geq 1$ , potem je  $P(t_1, \dots, t_n)$  atomična formula.  
 c) če je A atomična formula, potem sta A in  $\text{not}(A)$  literala.

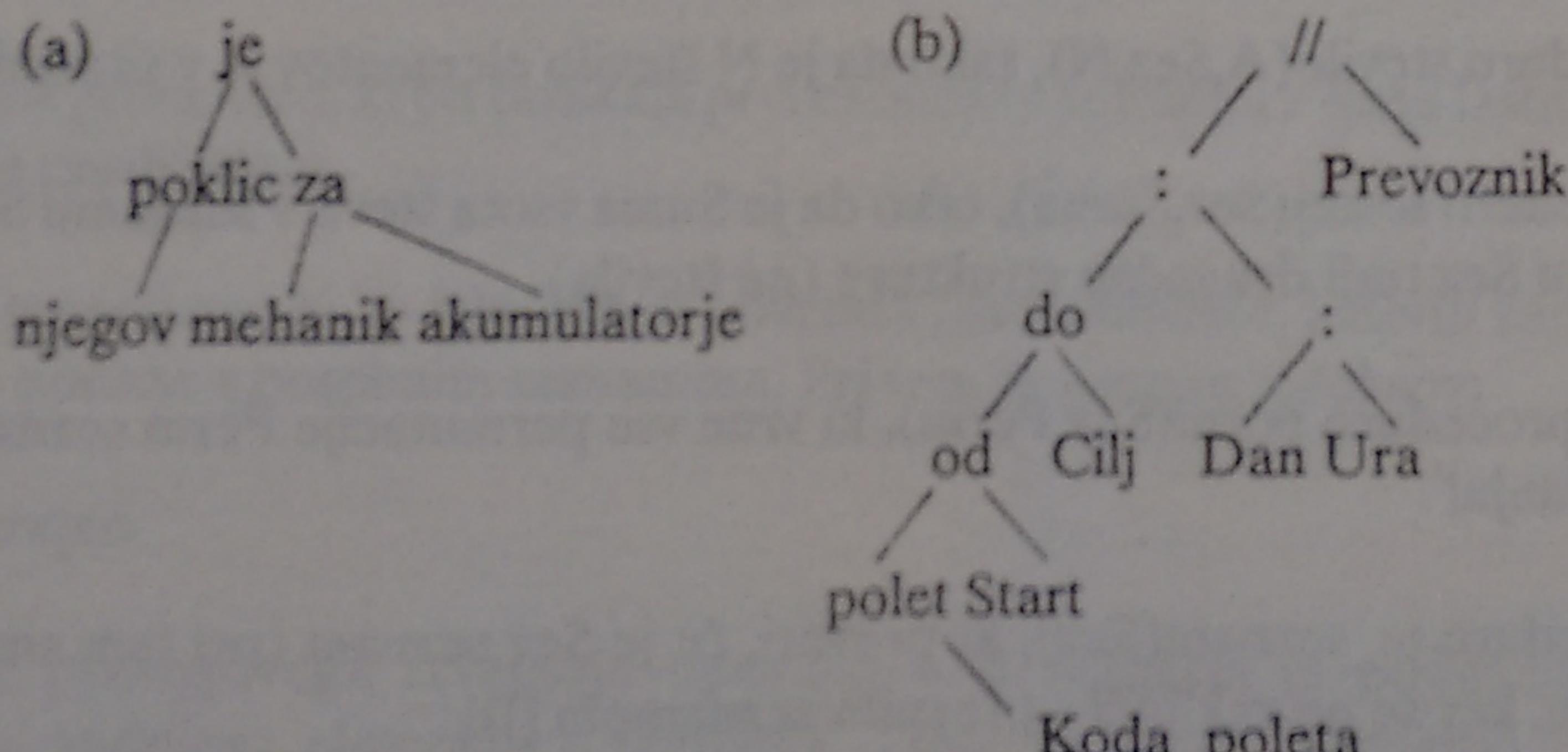
1.22. Dane imamo definicije operatorjev:

- $\text{:}-\text{op}(800,\text{fx},\text{if}).$
- $\text{:}-\text{op}(750,\text{xfx},\text{then}).$
- $\text{:}-\text{op}(749,\text{xfx},\text{else}).$

Nariši drevesno strukturo izrazov:

- a) if X then Y.
- b) if  $X > 15 * Y$  then R is 0 else  $\text{fak}(X,R)$ .

1.23 Ustrezno definiraj operatorje in zapiši izraze, ki jim ustrezajo strukture na sliki 6.1.



Slika 6.1 Dva primera struktur

+3 +4  
1.24. Sestavi interpreter za izraze iz naloge 1.22.

1.25. (\*\*\*) Sestavi proceduri:

- a) se\_lahko\_prilagodita(X,Y), ki je resnična, če se X in Y lahko prilagodita, vendar ju ne prilagodi
- b) ista\_spremenljivka(X,Y), ki je resnična, če sta X in Y ista neopredeljena spremenljivka

## 2 OPERACIJE NAD SEZNAMI IN MNOŽICAMI

2.1 Sestavi proceduro

- a) prvi(Sez,El), ki vrne prvi element seznama.
- b) tretji(Sez,El), ki vrne tretji element seznama.
- c) zadnji(Sez,El), ki vrne zadnji element seznama.
- d) n\_ti(Sez,N,El), ki vrne N-ti element seznama.

2.2. Sestavi proceduro, ki bo poiskala desni sosednji element danega elementa v seznamu.

2.3. Sestavi proceduro razbij(Sez,Glava,Rep), tako da je Glava glava seznama Sez, Rep pa rep seznama Sez!

2.4. Sestavi proceduro podseznam(Podsez,Sez), ki preveri, če je Podsez podseznam seznama Sez!

2.5. Sestavi proceduri dodaj\_z(A,Sez,Sez1), ki doda element A na začetek seznama Sez in dodaj\_k(A,Sez,Sez1), ki doda element A na konec seznama Sez!

2.6. Sestavi proceduri brisi1(A,Sez1,Sez), ki zбриše en A iz seznama Sez1 in brisi\_vse(A,Sez1,Sez), ki zbrisne vse A-je iz seznama Sez1! Izvajanje naj uspeta, tudi če ni nobenega elementa v seznamu.

2.7. Sestavi proceduro daljsi(Sez1,Sez2), ki preveri, če je seznam Sez1 daljši od seznama Sez2!

2.8. Sestavi proceduro dolzina(Seznam,N), tako da je N dolžina seznama Seznam!

2.9. Sestavi proceduro stevilo(A,Sez,N), tako da je N število elementov A v seznamu Sez!

2.10. Sestavi proceduro sestej(Sez,Suma), tako da je Suma vsota števil v seznamu Sez! Pri tem so lahko v seznamu Sez tudi drugačne strukture (ne števila).

2.11. (\*\*\*) Sestavi proceduro perm(Sez,Perm), ki vrne vse permutacije Perm seznama Sez pri avtomatskem vračanju!

2.12. Sestavi proceduro je\_seznam(Sez), ki preveri, če je Sez seznam (pri tem smatramo, da [a,b,c|d] ni seznam, ker se ne zaključi s praznim seznamom [])!

2.13. Sestavi proceduro, ki bo preverila, če je dani seznam palindrom (se enako bere z leve in z desne strani).

2.14. Sestavi proceduro množica(Sez,Mno), da je Mno množica dobljena iz seznama Sez, tako da so izvrženi vsi elementi, ki se pojavijo več kot enkrat v seznamu Sez!

2.15. Sestavi proceduro element(E,Mno), ki preveri, če je E element množice Mno!

2.16. Sestavi proceduro dodaj\_el(E,Mno,Mno1), ki doda množici element E!

2.17. Sestavi proceduro podmnožica(Podmn,Mno), ki preveri, če je množica Podmn podmnožica množice Mno!

2.18. Sestavi proceduro je\_množica(Mno), ki preveri, če je Mno množica!

2.19. Sestavi proceduro unija(Mn1,Mn2,Unija), tako da je Unija unija množic Mn1 in Mn2!

2.20. Sestavi proceduro presek(Mn1,Mn2,Presek), tako da je Presek presek množic Mn1 in Mn2!

2.21. Sestavi proceduro brisi\_el(E,Mno1,Mno), ki zbriše element E iz množice! Klic naj uspe tudi, če elementa E ni v množici Mno1!

2.22. Sestavi proceduro brez(Mn1,Mn2,Razlika), tako da je Razlika razlika množic Mn1-Mn2!

2.23. Sestavi proceduro enaki(Mn1,Mn2), ki preveri, če sta množici Mn1 in Mn2 enaki (Pri tem ni nujno da je vrstni red elementov množice v obeh množicah enak)!

2.24. Sestavi proceduro moc(Mno,Moc), ki vrne moč Moc množice Mno!

2.25. (\*\*) Sestavi proceduro potenca(Mno,Pot), ki vrne potenčno množico Pot množice Mno!

2.26. Definiraj operator "in" tako, da boš lahko uporabljal naslednjo sintaksu seznamov: "a in b in c in konec" bo npr. predstavljal seznam [a,b,c], "a in X" pa seznam [a | X]. Sestavi proceduri stik in element iz pogl. 3.2 v I.delu, ki bosta delovali za sezname v tej sintaksi.

2.27 (\*) Sestavi proceduro, ki bo poiskala prvi element v seznamu z dano lastnostjo definirano z enomestnim predikatom.

2.28. Sestavi proceduro, ki bo izluščila iz seznama tiste elemente, katerih zaporedne številke v seznamu so podane s posebnim seznamom. Pri tem je seznam indeksov:

- a) urejen
- b) (\*) neurejen

2.29. Slabost prologovega izvajanja operacij nad seznamami je počasnost, ki izvira iz nedosegljivosti zadnjega elementa v seznamu. Temu se lahko izognemo z uvedbo zapisa seznamov kot razlike dveh seznamov (glej pogl. 4.4 v I. delu). Sestavi proceduri razlika(Sez,Raz),

ki spremeni seznam Sez v nabolj splošno obliko razlike dveh seznamov, in vrni\_sez(Raz,Sez), ki spremeni seznam, zapisan v najbolj splošni obliki razlike dveh seznamov v standardno sintakso!

2.30. Sestavi proceduro pretvori(Seznam\_cifer,Stevilo), ki bo pretvorila seznam cifer v celo število s stikom cifer.

2.31. (\*) Sestavi proceduro, ki bo pobrala vse podizraze danega izraza v seznam. Uporabi razliko dveh seznamov!

2.32. Definirajmo zaporedje števil tako, da je nov element vsota vseh prejšnjih elementov zaporedja. Pri tem je prvi element poljubno fiksno naravno število. Sestavi proceduro, ki bo preverila, če dani seznam predstavlja tako zaporedje. Predpostavi, da so elementi zaporedja zapisani od desne proti levi.

2.33. Kaj je seznam z dvema repoma?

BINARNO DREVO!

### 3 PODATKOVNE BAZE

3.1. Dani sta proceduri predavanja(Dan,Od,Do,Predmet) in vaje(Dan,Od,Do,Predmet). Kako bi prologu zastavil naslednja vprašanja?

- a) Katera predavanja imam v ponedeljek?
- b) Kakšen urnik imam v torek?
- c) Kdaj imam predavanja iz predmeta Programiranje 2?
- d) Ali imam v petek kakšne vaje?

3.2. Poleg procedur iz naloge 3.1 imamo še proceduri profesor(Profesor,Predmet) in asistent(Asistent,Predmet). Kako bi prologu zastavil naslednja vprašanja? (dovoljeno je, da prolog vrne tudi dodatne (ne zahtevane) podatke)

- a) Kdo predava predmet Programski jeziki 2?
- b) Katere predmete predava profesor Polak?
- c) Kdaj ima profesor Delak predavanja?
- d) Kateri so učitelji pri predmetu Programski jeziki 2?
- e) Ali ima asistent Kolar v petek vaje?

3.3. Dane so procedure profesor(Profesor,Razred), uci(Profesor,Predmet) in ucenec(Ucenec,Razred). Kako bi v prologu zastavil naslednja vprašanja? (pazi, da prolog ne bo odgovoril preveč)

- a) Kateri profesor uči v 4.C razredu matematiko?
- b) Kateri profesorji učijo slovenščino?
- c) Katere učence uči profesor Mavec?
- d) Ali ima 2.A razred na urniku kemijo?
- e) Koliko učencev na šoli ima na urniku glasbeno vzgojo?
- f) Kateri profesorji učijo več kot 5 razredov?

3.4. Sestavi proceduro pouk(Dan,Ura) na osnovi procedur iz naloge 3.1, ki vrne "da", če imamo v dnevnu Dan ob uri Ura pouk!

3.5. Avtobusni vozni red je določen z procedurami vsak\_dan(Mesto1,Odhod,Mesto2,Prihod), delavnik(Mesto1,Odhod,Mesto2,Prihod) in nedelja(Mesto1,Odhod,Mesto2,Prihod). Sestavi proceduro zveza(Vrsta\_dneva,Mesto1,Odhod,Mesto2,Prihod), ki bo vrnila zvezo med mestoma Mesto1 in Mesto2, pri čemer je Vrsta\_dneva lahko nedelja ali delavnik! Predpostavi, da v voznem redu ni ciklov!

3.6. (\*) Sestavi proceduro zveza1 iz naloge 3.5, ki bo delovala kljub ciklom v voznem redu.

3.7. Dano imamo proceduro zveza1 iz naloge 3.6. Kako bi prologu zastavili naslednja vprašanja? (prolog lahko vrne tudi odvečne podatke)

- a) Kakšne zveze imam ob delavnikih iz Celja do Murske Sobote?
- b) Kdaj moram kreniti iz Ljubljane, če želim biti v nedeljo do 13. ure v Mariboru?
- c) Kateri je prvi avtobus iz Maribora do Celja ob delavnikih od 14. ure dalje?

3.8. V bazi podatkov imamo proceduri

populacija(Drzava,Preb\_v\_mio) in povrsina(Drzava,Kv\_km\_v\_mio). Sestavi proceduro gostota(Drzava,Preb\_na\_kv\_km)!

3.9. V podatkovni bazi imamo shranjene podatke o izdelkih neke tovarne. Vsak izdelek je sestavljen iz več delov. Del izdelka je lahko elementaren ali pa sestavljen iz več poddelov. V bazi imamo dve vrsti dejstev: 'del(Ime\_dela, Seznam\_delov)' in 'elementaren(Ime\_dela)'. Sestavi:

- a) proceduro 'preveri1(Sez)', ki vrne seznam delov, ki nastopajo v bazi kot sestavljeni in elementarni hkrati
- b)(\*)proceduro 'sestavni\_deli(Ime\_dela,Sez\_elementarnih\_delov)'
- c) (\*) proceduro 'preveri2(Sez)', ki vrne seznam elementarnih delov, ki ne nastopajo v nobenem dejstvu 'elementaren'
- d) proceduro izdelki(Seznam), ki vrne seznam izdelkov
- e) Optimiziraj podatkovno bazo tako, da izvržeš nepotrebne podatke.

3.10. Imejmo bazo dejstev v obliki 'casopis(Ime,Kraj\_izdaje)', 'naklada(Ime,Stevilo\_v\_1000)' in 'prodaja(Ime,Seznam\_parov\_Kraj\_Stevilo)'. Sestavi proceduro 'ok(Ime)', ki bo resnična, če se časopis Ime izdaja in prodaja v enaki količini.

3.11. (\*) Dan imamo urnik za celo fakulteto v obliki množice dejstev 'urnik(Letnik, Predmet, Profesor, Predavalnica, Dan,Ura)'. Sestavi proceduro 'preveri', ki bo preverila konsistentnost urnika, t.j., če se nekemu profesorju ali letniku predavanja pokrivajo, ali pa če je v isti predavalnici istočasno več predavanj.

3.12. Imejmo relacijsko shemo:

TOZD(TOZD#,NAZIV,MESTO)  
 DELAVEC(DELAVEC#,IME,KVALIFIKACIJA,DOHODEK,TOZD#)  
 PROJEKT(PROJEKT#,NAZIV,SREDSTVA)  
 VLOGA(DELAVEC#,PROJEKT#,FUNKCIJA)

ZBI  
Prevedi v prolog naslednja vprašanja zapisana v jeziku za poizvedovanje SQL:

- a) SELECT TOZD#, KVALIFIKACIJA  
FROM DELAVEC
- b) SELECT IME,TOZD#  
FROM DELAVEC
- c) SELECT \*  
FROM DELAVEC  
WHERE KVALIFIKACIJA = 'VK'
- d) SELECT IME,TOZD#  
FROM DELAVEC  
WHERE KVALIFIKACIJA = 'VK'
- e) SELECT IME  
FROM DELAVEC  
WHERE KVALIFIKACIJA = 'VK'  
AND DOHODEK > 10000
- f) SELECT IME  
FROM DELAVEC  
WHERE TOZD# = 4  
OR TOZD# = 7
- g) SELECT IME  
FROM DELAVEC  
WHERE TOZD# IN SELECT TOZD#  
FROM TOZD  
WHERE MESTO = 'LJUBLJANA'
- h) SELECT DELAVEC.IME, TOZD.MESTO  
FROM DELAVEC,TOZD  
WHERE DELAVEC.TOZD# = TOZD.TOZD#
- i) SELECT DELAVEC.IME, TOZD.MESTO, VLOGA.PROJEKT#,  
VLOGA.FUNKCIJA  
FROM DELAVEC,TOZD,VLOGA  
WHERE DELAVEC.KVALIFIKACIJA = 'VK'  
AND TOZD.TOZD# = DELAVEC.TOZD#  
AND VLOGA.DELAVEC# = DELAVEC.DELAVEC#  
AND VLOGA.PROJEKT# IN SELECT PROJEKT#  
FROM PROJEKT  
WHERE SREDSTVA > 950000
- j) SELECT NAZIV,MESTO  
FROM TOZD  
WHERE TOZD# IN  
SELECT TOZD#  
FROM DELAVEC  
WHERE DOHODEK > 10000  
AND DELAVEC# IN  
SELECT DELAVEC#  
FROM VLOGA  
WHERE FUNKCIJA = 'IZVAJALEC'  
AND PROJEKT# IN

```
SELECT PROJEKT#
FROM PROJEKT
WHERE SREDSTVA < 5000000
```

3.13 Zasnuj ustrezeno bazo podatkov in s pravili predstavi naslednji sklep občinske skupščine: "Za otroke, ki se udeležujejo šole v naravi, iz družin, kjer je mesečni dohodek na člana do 7 M din, bomo iz sredstev za socialno pomoč ogroženim prispevali 2 M din pomoči, pri dohodku na člana do 9 M din bomo prispevali 1.5 M, pri dohodku do 11 M pa 0.5 M. Če se udeležujeta šole v naravi dva učenca iz iste družine in če sta upravičena do prispevka, bomo prispevali vsakemu otroku še po 0.5 M, če pa so trije taki otroci, bomo prispevali vsakemu še 1.5 M."

3.14. Sestavi proceduro dodaj(Dejstvo), ki doda v podatkovno bazo Dejstvo, če ga še ni v bazi podatkov. Pri tem obstojajo tudi pravila z imenom predikata v glavi enakim kot ime predikata dejstva (ni dovoljeno preverjanje obstoja dejstva z 'call(Dejstvo)').

3.15. Imejmo podatkovno bazo s pozitivnimi dejstvi v navadni prologovi sintaksi in negirana dejstva v obliki 'negacija(Dejstvo)', ki predstavljajo negativno znanje, ki ga sam prolog ne zna upoštevati .Sestavi proceduro dodaj(Dejstvo), ki bo dodala dejstvo v podatkovno bazo, če ne vodi v protislovje z negativnimi dejstvi trenutne podatkovne baze, t.j. se ne da izpeljati iz negativnega znanja.

3.16. Dodajmo v podatkovno bazo iz naloge 3.15 tudi pozitivna pravila v navadni prologovi sintaksi in negativna pravila v obliki 'negacija(Glava) :- Telo.'. Sestavi proceduro dodaj(Dejstvo), ki bo dodala dejstvo v podatkovno bazo, če ne vodi v protislovje, t.j. se ne da izpeljati iz negativnega znanja.

3.17. (\*\*) Imejmo podatkovno bazo, kateri smo dodali negativno znanje v obliki vprašanj, ki jih zapišemo kot dejstva: vprasanje(Konjunkcija\_disjunkcija\_trditev), kar pomeni, da vprašanje pri danem programu ne sme uspeti. Sestavi proceduro dodaj\_stavek(Stavek), ki bo dodala stavek v podatkovno bazo, če ni v protislovju s pozitivnim in negativnim znanjem.

## 4 MATEMATIČNI PROBLEMI

4.1. (\*) Sestavi proceduro s(N,N1), katere klic uspe, če je  $N1 = N + 1$ . Procedura naj deluje na vse 4 možne načine : kot preverjanje, kot generator naslednika, kot generator prednika in kot generator parov pozitivnih celih števil, ki so v relaciji "s".

4.2. Sestavi proceduro vsota(A,B,Suma), ki preveri, če velja, da je  $A + B = \text{Suma}$ . Od aritmetičnih operacij lahko uporabiš samo proceduro naslednika ("s") iz naloge 4.1. A in B sta lahko pozitivna ali enaka 0.

4.3. Sestavi proceduro razlika(A,B,Razlika), ki preveri, če velja, da je  $A - B = \text{Razlika}$ . Od aritmetičnih operacij lahko uporabiš samo proceduro naslednika ("s") iz naloge 4.1. A je večji ali enak 0, B je manjši ali enak A.

4.4. Sestavi proceduro produkt(A,B,Produkt), ki preveri, če velja, da je  $A * B = \text{Produkt}$ . Od aritmetičnih operacij lahko uporabiš samo proceduro naslednika ("s") iz naloge 4.1 in procedure iz nalog 4.2 in 4.3. A in B sta večja ali enaka 0.

4.5. Sestavi proceduro div(A,B,Kvocient), ki preveri, če velja, da je  $A \div B = \text{Kvocient}$  (celoštevilčno deljenje). Od aritmetičnih operacij lahko uporabiš samo proceduro naslednika ("s") iz naloge 4.1 in procedure iz nalog 4.2..4.4. A in B sta celi pozitivni števili, A je lahko tudi 0.

4.6. Sestavi proceduro mod(A,B,Ostanek), tako da je Ostanek ostanek pri celoštevilčnem deljenju A z B. A in B sta pozitivni celi števili. Uporabljaš lahko procedure iz nalog 4.1..4.5.

4.7. Sestavi proceduro st\_cifer(X,Cifre), tako da je Cifre število cifer (desetiških) števila X. Uporabljaš lahko procedure iz nalog 4.1..4.6.

4.8. Posploši proceduro iz naloge 4.7 tako, da bo izračunavala število cifer za poljuben številski sistem z osnovo manjšo ali enako 10.

4.9. Sestavi proceduro potenca(A,Pot,Rezultat) tako, da je Rezultat enak Pot-ti potenci števila A. Pot je celo število, večje ali enako 0. Uporabljaš lahko procedure iz nalog 4.1..4.8.

4.10. (\*) Sestavi proceduro log2(X,LogX) tako, da je LogX enak celiemu delu dvojiškega logaritma števila X. X je pozitivno celo število. Uporabljaš lahko procedure iz nalog 4.1..4.9.

4.11. Posploši proceduro iz naloge 4.10 na logaritme s poljubno pozitivno celoštevilčno osnovo.

4.12. Sestavi proceduri abs(X,AbsX), kjer je  $\text{Abs}X = |X|$ , in sign(X,SignX), kjer je

$$\text{Sign}X = \begin{cases} 1, & \text{če } X > 0 \\ 0, & \text{če } X = 0 \\ -1, & \text{če } X < 0. \end{cases}$$

4.13. Sestavi proceduro fak(N,Nfak) tako, da je  $Nfak = N!$ .

4.14. Ackermannova funkcija je definirana rekurzivno:

1.  $A(0,Y) = 1$ ,
2.  $A(1,0) = 2$ ,
3.  $A(X,0) = X + 2$ , za  $X \geq 2$ ,
4.  $A(X+1,Y+1) = A(A(X,Y+1),Y)$ .

Sestavi proceduro ack(X,Y,Ack), ki bo računala funkcijo A.

4.15. (\*) Sestavi proceduro 'X iss Y', ki bo delovala kot vgrajena procedura 'is', le da v primeru, ko Y ni izračunljiv, ne bo povzročila prekinitve izvajanja, ampak samo ne bo uspela.

4.16. (\*) Sestavi proceduro med(X,Spodnji,Zgornji), ki bo vrnila pri avtomatskim vračanjem vsa cela števila X, ki so večja od števila Spodnji in manjša od števila Zgornji!

4.17. Kaj dela naslednji program:

$d(X, Y) :- Z \text{ is } Y \text{ div } X, Y \text{ is } X * Z.$

Poenostavi zgornji program!

4.18. Sestavi proceduro  $\text{ima\_delitelja}(X)$ , ki preveri, če ima celo število  $X$  delitelja (ni praštevilo).

4.19. Sestavi proceduro  $\text{prastevilo}(X)$ , ki preveri, če je  $X$  praštevilo.

4.20. Sestavi proceduro  $\text{odvod}(\text{Izraz}, \text{Odvod})$ , kjer je Izraz matematični izraz, Odvod pa odvod tega izraza. V izrazu lahko nastopa spremenljivka  $x$ , cela števila, operatorji  $+, -, *, /, ^$  ( $3^x$  pomeni 3 na  $x$ ) in funkcije sin, cos, tg, ctg in ln.

4.21. Sestavi program za spremjanje milj v kilometre in obratno.

4.22. Sestavi program za spremjanje rimskeih števil v arabska.

4.23. Sestavi program za spremjanje arabskih števil v rimska.

4.24. (\*) Sestavi program za računanje celoštevilčnega dela kvadratnega korena, ki bo temeljil na binarnem iskanju.

4.25. Deterministični končni avtomat je podan z množico stanj in množico prehodov med stanji nad vhodnimi simboli. Na začetku je avtomat v začetnem stanju in na vhodu dobi vhodno besedo, t.j. niz simbolov. Stroj prehaja iz stanja v stanje glede na vhodne simbole. če se stroj, ko je "porabil" vse vhodne simbole ustavi v končnem stanju, pravimo, da je vhodno besedo sprejel. Sestavi interpreter determinističnega končnega avtomata, ki na vhodu dobi množico prehodov, začetno stanje, množico končnih stanj in vhodno besedo in ki vrne sporočilo, če je vhodno besedo sprejel ali ne.

4.26. (\*) Nedeterministični končni avtomat z epsilon prehodi je splošnejša oblika končnega avtomata iz naloge 4.25, kjer lahko stroj iz nekega stanja nad nekim simbolom preide v enega od več možnih stanj in da obstojajo t.i. epsilon prehodi, ki niso odvisni od vhodnih simbolov in pri prehodu ne porabijo nobenega vhodnega simbola. Sestavi interpreter takega avtomata z analognim vhodom kot v nalogi 4.25.

4.27. (\*\*) Splošno rekurzivne funkcije predstavljajo razred izračunljivih problemov. Definirane so z naslednjimi pravili:

1) Začetne funkcije so splošno rekurzivne:

1.1 Ničelna funkcija:  $Z(X) = 0$

1.2 Funkcija naslednika  $N(X) = X + 1$

1.3 Projekcija  $U_{in}(X_1, \dots, X_n) = X_i, n \geq 1, i \geq n$

2) Pravilo substitucije: če so  $g(X_1, \dots, X_n), h_1(Y_1, \dots, Y_m), \dots, h_n(Y_1, \dots, Y_m)$  splošno rekurzivne funkcije, potem je tudi funkcija  $f(Y_1, \dots, Y_m) = g(h_1(Y_1, \dots, Y_m), \dots, h_n(Y_1, \dots, Y_m))$  splošno rekurzivna.

- 3) Pravilo primitivne rekurzije: če sta  $h(X_1, \dots, X_n)$  in  $g(X_1, \dots, X_n, Y, Z)$  splošno rekurzivni funkciji, potem je funkcija  $f(X_1, \dots, X_n, Y)$  podana z
- $$f(0, X_1, \dots, X_n) = h(X_1, \dots, X_n)$$
- $$f(Y + 1, X_1, \dots, X_n) = g(X_1, \dots, X_n, Y, f(Y, X_1, \dots, X_n))$$

tudi splošno rekurzivna funkcija

- 4) Operator minimizacije: če je  $g(X_1, \dots, X_n, Y)$  splošno rekurzivna funkcija, potem je
- $$\min(Y, g(X_1, \dots, X_n, Y)), \text{ ki je definirana z}$$
- $$\min(Y, g(X_1, \dots, X_n, Y)) = \text{najmanjši } Y, \text{ kjer je } G \text{ enak } 0, \text{ četak } Y \text{ obstaja, sicer nedefinirana}$$

tudi splošno rekurzivna funkcija.

Dokaži, da se da s prologom izračunati vse, kar se izračunati da, tako da sestaviš interpreter splošno rekurzivnih funkcij izrac(Funkcija, Vrednost), ki bo izračunal Vrednost Funkcije z danimi argumenti, če so ustrezena definicije funkcij podane kot seznam dejstev oblike 'def(Leva\_stran = Desna\_stran).' Na primer:

```
% primer splošno rekurzivne funkcije
def(suma(0,Y) = Y).
def(suma(N + 1,Y) = suma(N,Y + 1)).
def(produkt(0,Y) = 0).
def(produkt(N + 1,Y) = suma(Y,produkt(N,Y))).
def(razlika(N,0) = N).
def(razlika(0,Y) = 0). % nenegativna razlika
def(razlika(N + 1,Y) = razlika(N,dec(Y))).
def(dec(0) = 0). % nenegativni dekrement
def(dec(Y + 1) = Y).
```

% primer dialoga:

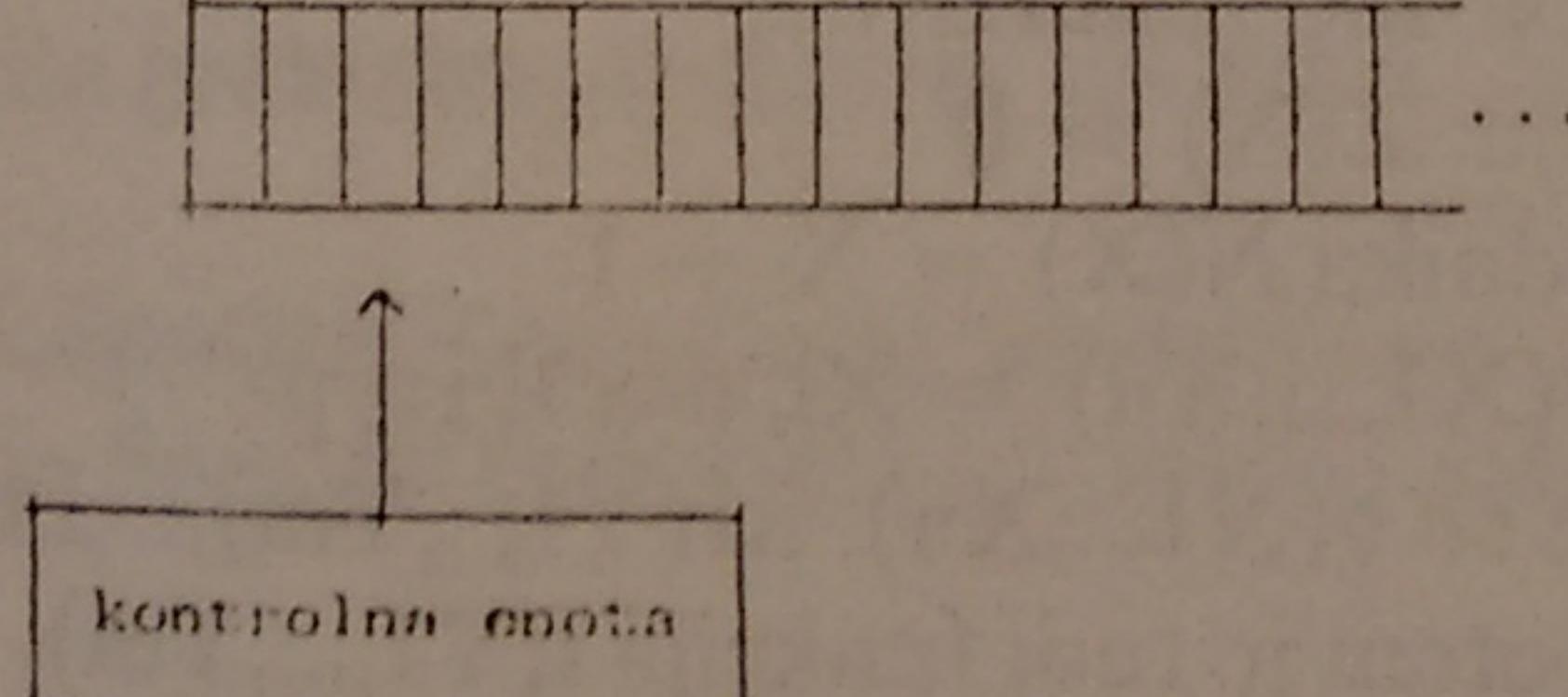
% prvo celo število, katerega kvadrat je večji od 20

?-izrac(min(Y,razlika(20,produkt(Y,Y))),X).

Y = 5

X = 5

4.28. (\*\*\*) Turingov stroj je definiran kot avtomat z neskončnim trakom v eno smer, razdeljenim na prostorčke za en tračni simbol, s čitalno/pisalno glavo, ki lahko bere in izpisuje simbole na trak in ki se lahko premika levo ali desno, vendar ne levo čez rob traku, in kontrolno enoto, ki se lahko nahaja v enem od stanj iz končne množice stanj. V enem prehodu Turingov stroj glede na trenutno stanje in trenutni simbol, ki se nahaja na traku nad čitalno glavo, zapiše novi simbol čez prejšnjega, preide v neko novo stanje in premakne čitalno glavo za eno mesto v levo ali desno.



Slika 6.2 Turingov stroj

Na začetku se na traku nahaja končno število simbolov iz vhodne abecede, preostali del traku pa je označen s posebnimi simboli (blanki). Stroj se nahaja v začetnem stanju, čitalna glava pa nad najbolj levim simbolom na traku. Pravila, po katerih stroj deluje, so podana s prehodno funkcijo. Če se po končnem številu korakov Turingov stroj ustavi v posebnem končnem stanju, pravimo, da je vhodno besedo sprejel.

Da se pokazati, da se s Turingovim strojem da izračunati vse, kar se izračunati da. Sestavi interpreter Turingovega stroja, ki na vhodu dobi seznam znakov, ki se trenutno nahajajo na traku, prehodno funkcijo, začetno stanje in ime končnega stanja, in ki simulira delovanje Turingovega stroja, dokler se ne ustavi (če se ustavi) ter sporoči, ali je vhodno besedo sprejel ali ne.

## 5 LOGIČNI PROBLEMI IN IGRE

5.1. Sestavi proceduro `wff(Izraz)`, ki preveri, če je izraz Izraz dobro sestavljen (well formed formula-WFF). Pravila, ki povedo, če je nek izraz WFF so sledeča:

Pravilo 1: Atomična formula je WFF.

Pravilo 2: če sta F in G WFF, potem so "not F", "F & G",  
 "F ∨ G",  $F \Rightarrow G$  in  $F \Leftrightarrow G$  tudi WFF. (pri tem  
 operator not veže najmočneje, zatem &, ∨,  $\Rightarrow$  in  
 najšibkeje veže  $\Leftrightarrow$ )

Pravilo 3: če je F WFF, potem sta "vsak\_X F" in  
 "eksistira\_X F" tudi WFF in operatorja vežeta enako  
 močno kot operator not

Pravilo 4: WFF lahko dobimo natanko s pravili 1,2 in 3.

5.2. Sestavi proceduro 'izpolni(Logični\_izraz)', ki bo poiskala take vrednosti logičnih spremenljivk v izrazu (true ali false), da bo izraz resničen. Ustrezno definiraj operatorje za konjunkcijo, disjunkcijo, negacijo, implikacijo in ekvivalenco.

5.3. Zakaj pri rešitvi naloge 5.2. za neresnične izraze ne moremo uporabiti stavka:  
`ne_izpolni(Izraz) :- not(izpolni(Izraz)).`

5.4. Sestavi interpreter logičnih izrazov z operatorji negacije (not), konjunkcije (&) in disjunkcije (∨).

5.5. Sestavi:

- a) program, ki sam sebe izpiše
- b) proceduro, ki izpiše trenutno podatkovno bazo
- c) (\*\*\*) program, ki sam sebe izpiše brez uporabe metalogičnih predikatov retract, clause,...

5.6. Ali obstaja program, ki sam sebe interpretira?

- a) nad istim vhodom
- b) nad različnim vhodom

2 + 3 + 4

5.7. Ali proceduri mama(Mama,Otrok), oce(Oce,Otrok) zadostujeta za definicijo vseh krvnih sorodstvenih vezi? Sestavi proceduro neznan\_spol(Oseba), ki vrne osebo, katere spol ni mogo določiti, in proceduro nima\_otrok(Oseba), ki bo vrnila osebo, ki nima svojih otrok.

5.8. Dan imamo program:

|                         |                 |
|-------------------------|-----------------|
| roditelj(lucija,igor).  | zenska(sonja).  |
| roditelj(andrej,igor).  | zenska(lucija). |
| roditelj(andrej,boris). | moski(andrej).  |
| roditelj(andrej,sonja). | moski(boris).   |

|                       |              |
|-----------------------|--------------|
| moski(igor).          | moski(igor). |
| brat(Oseba1,Oseba2):- |              |
| moski(Oseba1),        |              |
| roditelj(Rod,Oseba1), |              |
| roditelj(Rod,Oseba2). |              |

Kaj bo prolog odgovoril na naslednja vprašanja:

- a)?-roditelj(andrej,X),zenska(X).
- b)?-moski( ),zenska( ).
- c)?-roditelj( ,X),zenska( ).
- d)?-setof(B1-B2,brat(B1,B2),Sez).

Napiši pravilno definicijo procedure brat!

5.9. Dani imamo proceduri roditelj(Roditelj,Otrok) in spol(Oseba,Spol). Sestavi procedure za naslednje krvne sorodstvene vezi:

- a)mama(Mama,Otrok),
- b)sin(Sin,Roditelj),
- c)stric(Stric,Necak\_inja),
- d)prednik(Prednik,Potomec),
- e)v\_krv\_sorodstvu(Oseba1,Oseba2),
- f)stara\_mama(S\_mama,Vnuk\_inja),
- g)bratranec(Bratranec,Bratranec\_sestricna).

5.10. K proceduram iz naloge 5.9 dodajmo še pravne sorodstvene vezi s proceduro porocena(Moz,Zena). Sestavi naslednje procedure:

- a)zena(Zena,Moz),
- b)tasca(Tasca,Zet),
- c)ocim(Ocim,Sinovec).

5.11. (\*) Sestavi proceduro sorod(Oseba1,Oseba2,Sorodstvo), ki vrne sorodstveno vez med dvema osebama.

5.12. Hierarhična struktura v podjetju je podana z dejstvi oblike direktno\_nadrejeni(Nadrejeni,Podrejeni). Sestavi procedure:

- a) nadrejeni(Nadrejeni,Podrejeni), kjer ni nujno, da je Nadrejeni direktno Nadrejen Podrejenemu.

- b) kolega1( $X, Y$ ), ki preveri, če imata  $X$  in  $Y$  istega direktno - nadrejenega  
 c) kolega( $X, Y$ ), ki preveri, če sta  $X$  in  $Y$  na istem hierarhičnem nivoju.  
 d) direktor( $X$ ), ki vrne direktorja podjetja  
 e) izvajalec( $X$ ), ki preveri, če  $X$  ni nikomur nadrejen

5.13. (\*\*)) Dani imamo dve pravili, ki definirata relacijo daje:

- (1) če je  $X$  naravno število potem  $2X$  daje  $X$ . ( $2X$  je število, ki ga dobimo, če številu  $X$  dodamo na levi strani cifro 2)
- (2) če  $X$  daje  $Y$ , potem  $3X$  daje  $Y2Y$

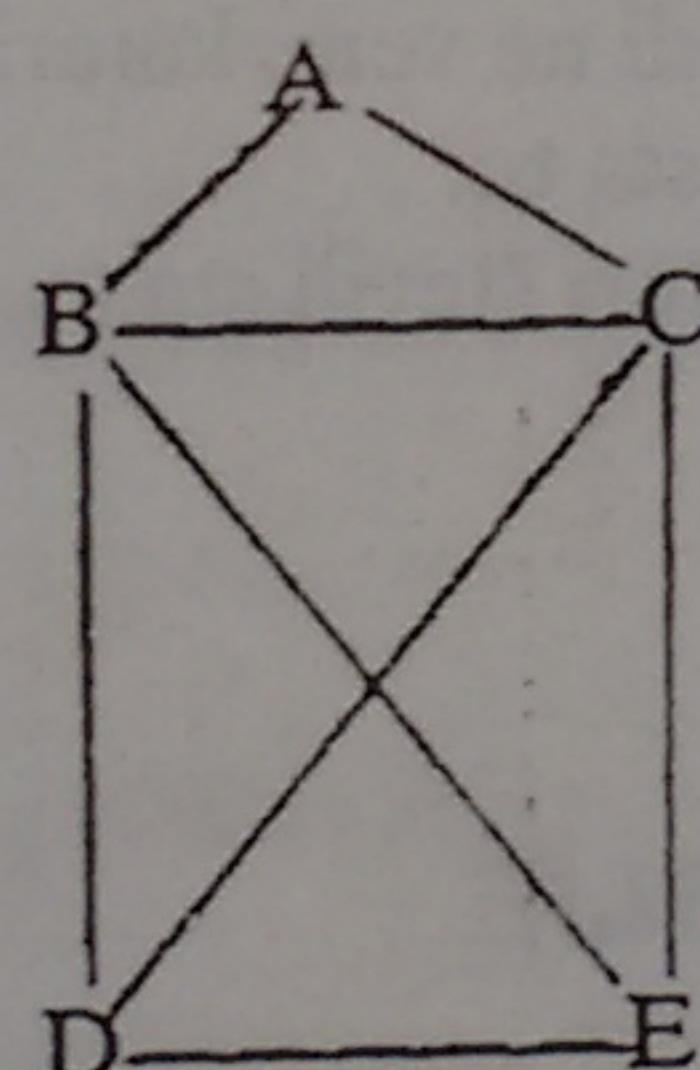
- a) Sestavi proceduro daje\_sebe( $X$ ), ki bo vrnila tak  $X$ , da  $X$  daje  $X$ .  
 b) Sestavi proceduro daje\_X2X( $X$ ), ki bo vrnila tak  $X$ , da  $X$  daje  $X2X$ .  
 c) Sestavi splošno relacijo  $X$  daje  $Y$ , ki vrne tak  $X$  in  $Y$ , da  $X$  daje  $Y$ .

5.14. (\*) Trije kanibali in trije belci želijo priti čez reko. Na voljo imajo čoln, v katerem se lahko vozita naenkrat samo dva človeka. Pri tem je potrebno zadostiti pogoju, da med prevažanjem na nobeni obali ne ostane več kanibalov kot belcev. Sestavi proceduro, ki bo vrnila seznam vmesnih stanj v obliki  $p(\text{Levo}, \text{Desno}, \text{Polozaj\_colna})$ , pri čemer sta Levo in Desno seznama oseb na levi oziroma desni obali. 'b' naj označuje belca in 'k' kanibala.

5.15. Sestavi proceduro 'uredi(Domine, Vrsta)', ki bo uredila seznam domin v vrsto v skladu s pravili igre. Vsaka domina je predstavljena s parom naravnih števil med 1 in 6, npr. 3/4. Upoštevaj da smemo domine tudi obračati v smer, ki nam ustreza.

5.16. (\*) Igra z vžigalicami je definirana takole: Imamo poljubno število kupčkov s poljubnim številom vžigalic. Vsak igralec v eni potezi vzame neničelno število vžigalic iz enega kupčka. Zgubi tisti, ki prvi nima poteze. Sestavi program, ki bo optimalno igral igro z vžigalicami.

5.17. (\*) Sestavi program, ki bo izpisal zaporedje povezav, ki jih moramo napraviti, da narišemo sliko 6.3 v enem mahu, t.j., da med risanjem ne dvignemo svinčnika od papirja.



Slika 6.3

5.18. (\*) Logični izrazi so sestavljeni iz operatorjev konjunkcije ( $\&$ ), disjunkcije ( $\vee$ ), negacije (not), implikacije ( $\Rightarrow$ ) in ekvivalence ( $\Leftrightarrow$ ). Izraz je v konjunktivni normalni obliki, če je zapisan kot konjunkcija disjunktov, kjer je vsak disjunkt, disjunkcija literalov. Literal je lahko tudi negativen (negiran). če imamo npr. logični izraz