Programiranje i programski jezici

PROgramming in LOGic

•Prolog je logički programski jezik opšte namene povezan sa veštačkom inteligencijom i kompjuterskom lingvistikom.

•Koreni Prologa se nalaze u predikatskom računu prvog reda, formalnoj logici i za razliku od mnogih drugih programskih jezika Prolog je deklarativan, tj. programska logika je izražena relacijskim termima predstavljenim činjenicama i pravilima. Izračunavanja se iniciraju pokretanjem upita nad ovim relacijama.

•Jezik je osmislila grupa predvođena Alain Colmerauer u Marseju početkom 70tih, a prvi Prološki sistem je razvijen 1972 od strane Colmerauer i Philippe Roussel

(Wikipedia)

PROgramming in LOGic

- U početku je bio dizajniran za
  - obradu prirodnih jezika

    a danas se koristi u raznim oblastima poput:
  - Dokazivanje teorema
  - Ekspertni sistemi
  - Igre
  - Sistemi za automatsko odgovaranje
  - Ontologije
  - Sistemi za kontrolu

PROgramming in LOGic

- Baziran je proceduralnoj interpretaciji Hornovskih formula:
  - Iskazna slova *p*, *q*, *r*, ...
  - Formula oblika
- Osnovni mehanizmi zaključivanja su
  - Pravilo rezolucije
  - *Backtracking*
- Edinburška i Lispovska sintaksa –
  - Arity vs. Micro

rel(a1,…,an). ((rel a1 … an))

rel(…):-rel1(…),…,relk(…). ((rel …)(rel1 …)…(relk …))

- Programi mogu biti “proverni” i/ili “generatorni”

PROgramming in LOGic

- Svaki program u Prologu se sastoji od:
  - Činjenica
  - Pravila
  - Pitanja
- Imena relacija i objekata moraju početi MALIM SLOVOM
- Imena promenljivih moraju početi VELIKIM SLOVOM ili “_”
- Svaka činjenica ili pravilo mora se završiti TAČKOM – “.”

SWI-Prolog

- Imenovanje izvornih datoteka – *ime*.pl
- Učitavanje
  - Windows - Duplim klikom na izvornu datoteku se pokreće SWI-Prolog i učitava datoteka
  - Linux – Zadavanjem komande *prolog* u terminalu
  - U okviru SWI-Prologa – navođenjem imena datoteka između znakova [ ] – [*ime*].
  - Nakon izmena izvornih datoteka, ponovno učitavanje se obavlja komandom *make.*
- Listanje
  - svih učitanih predikata(relacija) – *listing.*
  - određenih predikata(relacija) – *listing(ime_predikata).*
- Izlazak – *halt.*

Primer 1

```
a(1).  a(2).
b(X):-a(X).
```

- a(1).
- a(X). - ;
- a(X),write(X),nl,a(3).
- a(X),write(X),nl,fail.
- b(2).
- b(X),write(X),nl. - ;

Primer 2

```
a1(1).   a1(2).
b1(77).  b1(88).
c(X,Y):-a1(X),b1(Y).

c(X,Y),write(X),write(Y),nl,fail.
```

Primer 3

```
s(2).
s(X):-s(X).

d(X):-d(X).
d(1).

s(X),write(X),nl,fail.
d(X),write(X),nl,fail.
```

Porodično stablo

- Date su činjenice:
  - musko(X), zensko(X)
  - roditelj(X,Y) – X je roditelj od Y
- Definisati predikate:
  - majka(X,Y) – X je majka od Y
  - otac(X,Y) – X je otac od Y
  - sestra(X,Y), brat(X,Y)
  - baba(X,Y), deda(X,Y)
  - tetka(X,Y), ujak(X,Y), stric(X,Y)
  - teca(X,Y), ujna(X,Y), strina(X,Y)
  - ...

Jednakost i unifikacija

- X = Y
- *Algoritam unifikacije* – algoritam kojim se ispituje da li su dva terma ujednačva
  - termi su neujednačivi
  - termi su ujednačivi i bukvalno jednaki
  - termi su ujednačivi i spisak zamena je X1 -> t1, X2 -> t2, ...
- vozi(student, bicikl) = vozi(student, X)
- tacka(X, Y, Z) = tacka(X1, Y1,Z1)
- f(X, X) = f(a, b)
- f(X, a(b, c)) = f(Z, a(Z, c))
- a(b, C, d(e, F, g(h, i, J))) = a(B, c, d(E, f, G))

Poredjenje brojeva

- X =:= Y –X i Y su isti brojevi
- X =\= Y – X i Y su različiti brojevi
- X < Y – X je manje od Y
- X > Y – X je veće od Y
- X =< Y – X je manje ili jednako sa Y
- X >= Y – X je veće ili jednako sa Y

Prološki algoritam za određivanje maksimuma i minimuma dva broja

```
max(A,B,A):-A>=B.
max(A,B,B):-A<B.

min(A,B,A):-A<=B.
min(A,B,B):-A>B.
```

- `max(3,6,3).`
- `max(3,6,X).`
- `min(3,6,X).`

Ko je vladao Srbijom?

```
kralj(stefan, 1217,1228).
kralj(radoslav,1228,1233).
kralj(vladisalv,1234,1243).
kralj(uros,1243,1276).
kralj(dragutin,1276,1282).
vladar(X,Y):-kralj(X,A,B), Y>=A,Y=<B.
```

- `vladar(stefan,1220).`
- `vladar(X,1250).`
- `vladar(nemanja,X).`
- `vladar(X,1276).`

Operacije sa brojevima

- X + Y – zbir
- X – Y – razlika
- X * Y – proizvod
- X / Y – količnik
- X // Y – celobrojni količnik
- X mod Y – ostatak pri deljenju

- Dodela vrednosti se vrši operatorom ***is***

Prološki algoritam za izračunavanje zbira prvih N prirodnih brojeva.

```
suma(0,0).
suma(N,S):-N>0, N1 is N-1, suma(N1,S1),S is
S1+N.
```

- suma(2,Y).
- Y=3
- suma(3,6).
- yes

Prološki algoritam za određivanje faktorijela broja.

```
fakt(0,1).
fakt(X,Y):-U is X-1,fakt(U,Z),Y is X*Z.
```

- `fakt(3,6).`
- `fakt(4,X).`
- `fakt(2,X),write(X),nl,fail.`

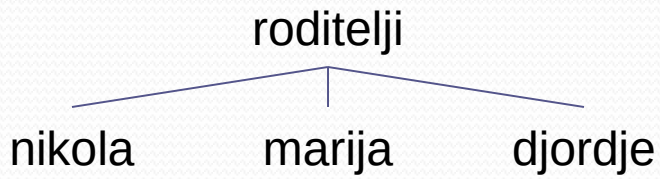
Prološki algoritam za određivanje faktorijela broja.

```
fakt(0,1).
fakt(X,Y):-X>0,U is X-1,fakt(U,Z),Y is X*Z.
```

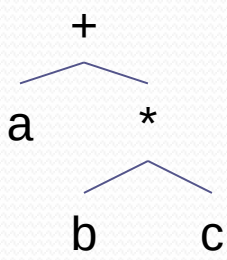
- `fakt(3,6).`
- `fakt(4,X).`
- `fakt(2,X),write(X),nl,fail.`

Strukture, stabla, liste

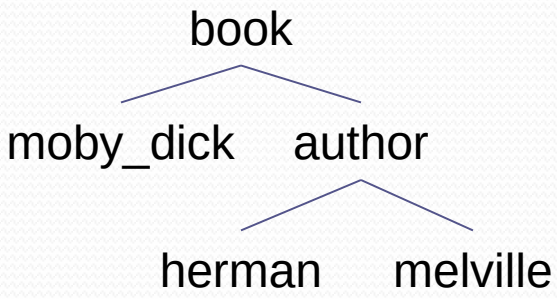
•roditelji(nikola, marija, djordje)



•a + b * c ili +(a, *(b, c))

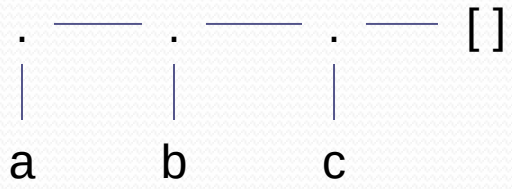


•book(moby_dick, author(herman, melville))

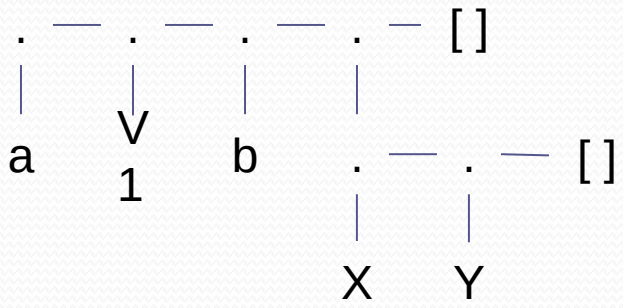


Strukture, stabla, liste

- [a, b, c] ili [a | [b | [c | [ ]]]]



- [a, V1, b, [X, Y]]



Primer

```
p([1,2,3]).
p([a,b,c,[d,22,pera]]).
```

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

Prološki algoritam za ispitivanje da li je *x* element liste *y*

```
elem(X,[X|_]).
elem(X,[_|V]):-elem(X,V).
```

- `elem(5,[1,2,5,4]).`
- `elem(7,[]).`
- `elem(X,[1,2,3,4]),write(X),nl,fail.`

Prološki algoritam za ispitivanje da li je *x* lista

```
islist([]).
islist([_|B]):-islist(B).
```

- `islist([1,2,3,4]).`
- `islist([1,2,[3,4],5]).`
- `islist([]).`
- `islist(a).`
- `islist(X).`

Nenegativan ceo broj

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

- `is_integer(123).`
- `is_integer(X).`
- `is_integer(X),write(X),nl,fail.`

Rez

```
r1(1). r1(2).  r1(3).
```

- `r1(X),write(X),nl,fail.`
- `r1(3).`

Rez

```
r1(1). r1(2):-!.  r1(3).
```

- `r1(X),write(X),nl,fail.`
- `r1(3).`

Rez

```
pisi(0):-!.
pisi(X):-write(X),nl,X1 is X-1,pisi(X1).
```

- `pisi(3).`
- `pisi(3),fail.`

Rez

```
aa(1).  aa(2).  aa(3).  bb(11). bb(22).
bb(X):-aa(X),!.   bb(33).
cc(111).  cc(222).
dd(X,Y):-aa(X),cc(Y),!.   dd(7,77).
ee(X,Y):-aa(X),!,cc(Y).   ee(8,88).
ff(X,Y):-!,aa(X),cc(Y).   ff(9,99).   ff(100).
```

- `bb(X),write(X),nl,fail.`
- `dd(X,Y),write(X),write(‘ ‘),write(Y),nl,fail.`
- `ee(X,Y),write(X),write(‘ ‘),write(Y),nl,fail.`
- `ff(X,Y),write(X),write(‘ ‘),write(Y),nl,fail.`

Predikat append

- append([1,2,3],[a,b,c,d],X).
- append(X,[1,2,3],[[a,b],c,1,2,3]).
- append(X,[1,2,3],[[a,b],c,1,2]).
- append([a,b,c],X,Y).
- append(X,Y,[a,b,c]).

```
app([],X,X):-!.
app([A|B],C,[A|D]):-app(B,C,D).
```

Presek dve liste

```
pres([],_,[]).
pres([A|B],Y,[A|Z]):-member(A,Y),pres(B,Y,Z).
pres([_|B],Y,Z):-pres(B,Y,Z).
presek(X,Y,Z):-pres(X,Y,Z),!.
```

- `pres([3,1,2,4],[1,3,5],X).`
- `pres([3,1,2,4],[1,3,5],X),write(X),nl,fail.`
- `presek([3,1,2,4],[1,3,5],X).`
- `presek([3,1,2,4],[1,3,5],X),write(X),nl,fail.`

Unija dve liste

```
unija1([],X,X).
unija1([X|L1],L2,[X|L]):-not(member(X,L2)),
unija1(L1,L2,L).

unija1([X|L1],L2,L):-member(X,L2),
unija1(L1,L2,L).

unija(X,Y,Z):-unija1(X,Y,Z),!.
```

Razlika dve liste

```
razlika1([],X,[]).
razlika1([X|L1],L2,L):-member(X,L2),
razlika1(L1,L2,L).

razlika1([X|L1],L2,[X|L]):-not(member(X,L2)),
razlika1(L1,L2,L).

razlika(X,Y,Z):-razlika1(X,Y,Z),!.
```

Simetrična razlika dve liste

```
simraz(X,Y,Z):-pegla(X,X1), pegla(Y,Y1), razlika(X1,Y1,Z1),
razlika(Y1,X1,Z2), unija(Z1,Z2,Z).

pegla1([],[]).
pegla1([X|L],L1):-member(X,L),pegla1(L,L1).
pegla1([X|L],[X|L1]):-not(member(X,L)),pegla1(L,L1).
pegla(X,Y):-pegla1(X,Y),!.

razlika1([],X,[]).
razlika1([X|L1],L2,L):-member(X,L2),razlika1(L1,L2,L).
razlika1([X|L1],L2,[X|L]):-not(member(X,L2)),razlika1(L1,L2,L).
razlika(X,Y,Z):-razlika1(X,Y,Z),!.

unija1([],X,X).
unija1([X|L1],L2,[X|L]):-not(member(X,L2)),unija1(L1,L2,L).
unija1([X|L1],L2,L):-member(X,L2),unija1(L1,L2,L).
unija(X,Y,Z):-unija1(X,Y,Z),!.
```