Prof.dr.sc. Bojana Dalbelo Bašić

Fakultet elektrotehnike i računarstva
Zavod za elektroniku, mikroelektroniku, računalne i inteligentne sustave

www.zemris.fer.hr/~bojana
bojana.dalbelo@fer.hr

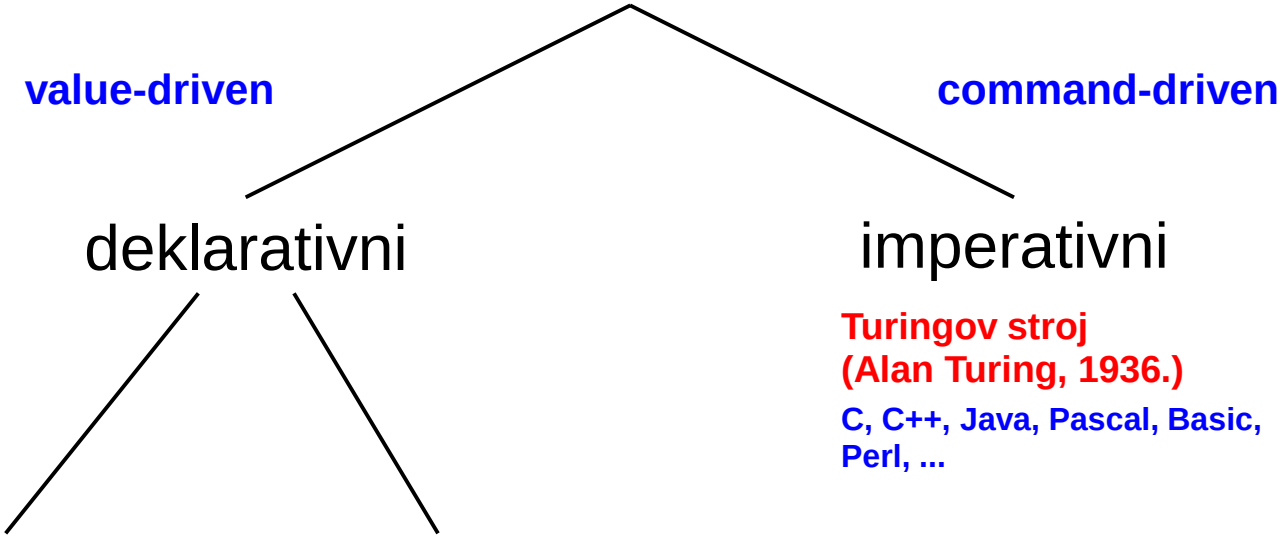
Logičko programiranje

u Prologu

LOGIČKO PROGRAMIRANJE

- **Logičko programiranje** (engl. logic programming)
  - uporaba matematičke logike za programiranje
- **Ideja:** opisati problem logičkim formulama, a rješavanje prepustiti računalu
- **"Algorithm = Logic + Control"**
- Različito od dokazivača teorema (ATP) jer:
  - U program su ugrađeni eksplicitni kontrolni mehanizmi
  - Nije podržana puna ekspresivnost logike

PROGRAMSKI JEZICI



... Python, C#, Mathematica, Matlab, R ?

DEKLARATIVNI PROGRAMSKI JEZICI

- deklarativni jezici opisuju **što** se izračunava umjesto **kako** se to izračunava
  - zadaje se specifikacija **skupa uvjeta** koji definiraju prostor rješenja
  - **pronalaženje rješenja** prepušteno je interpreteru
- osnovne karakteristike:
  - **eksplicitno stanje** umjesto implicitnog stanja
    - **nema popratnih efekata (engl. side-effects)**
  - programiranje s **izrazima**
    - funkcijski jezici: izraz=fukcija
    - logički jezici: izraz=relacija

POPRATNI EFEKTI

```
function foo(x)
begin
y = 0
return 2*x
end
```

```
function main()
begin
y = 5
x = foo(2) + y
return x
end
```

- koja je povratna vrijednost funkcije **main** ?
- povratna vrijednost **ovisi o redoslijedu** izračuna pribrojnika!
- deklarativno ne bi smjelo biti razlike:
  - A+B = B+A

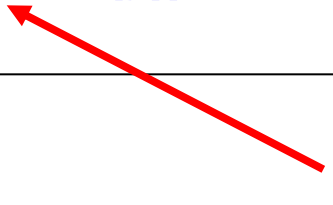
popratni efekt!

"prave funkcije" ne prtljaju po memoriji već samo vraćaju vrijednost

POPRATNI EFEKTI

```
function foo(x)
begin
y = 0
return 2*x
end
```

```
…
y = 5
if (foo(y) == foo(y)) then …
…
```



- funkcija **foo** ima popratni efekt stoga **nije referencijalno prozirna**
  - **ne vrijedi Leibnizovo pravilo: “equals for equals”**
- moramo voditi računa o **tijeku izvođenja** (proceduralno) umjesto da se koncentriramo na **značenje programa** (deklarativno)

DEKLARATIVNI JEZICI

- **čisto deklarativni** (engl. *purely declarative*) jezici ne dozvoljavaju popratne efekte
  - Haskell, ...
- radi praktičnosti većina deklarativnih jezika dopušta kontrolirane popratne efekte (“*declarative in style”*)
  - Lisp, SML, Ocaml, **Prolog**, ...
- pogodnosti:
  - formalno koncizni, visoka razina apstrakcije
  - lakša formalna verifikacija
  - manja mogućnost pogreške
- nedostaci:
  - neučinkovitost
  - neke strukture/funkcije iziskuju popratne efekte (npr.?)

- deklarativni jezici **nemaju varijabli** u klasičnom smislu (“*variables do not vary*”)
- **nemaju naredbe pridruživanja** (x = x + 1) jer bi to iziskivalo popratni efekt
- **nemaju programskih petlji**
  - umjesto toga: **rekurzija**

```
function fact(n)
begin
  a = 1
  for i = 1 to n do
  begin
a = a*i
  end
  return a
end
```

Haskell:

```
fact 1 = 1
fact n = n*fact(n-1)
```

PROLOG

- **Prolog** = ***Pro****gramming in* ***Log****ic*
  - deklarativni logički programski jezik
  - 1972.: Alan Colmerauer, Robert Kowalski, Philippe Roussel
  - originalno razvijen za NLP
- fundamentalni koncepti:
  - **rekurzija**
  - **unifikacija**
  - postupak vraćanja (**backtracking**)
- nije čisto deklarativan:
  - **"Algorithm = Logic + Control"**
  - **$(A \wedge B) \rightarrow C \neq (B \wedge A) \rightarrow C$**

HORNOVE FORMULE

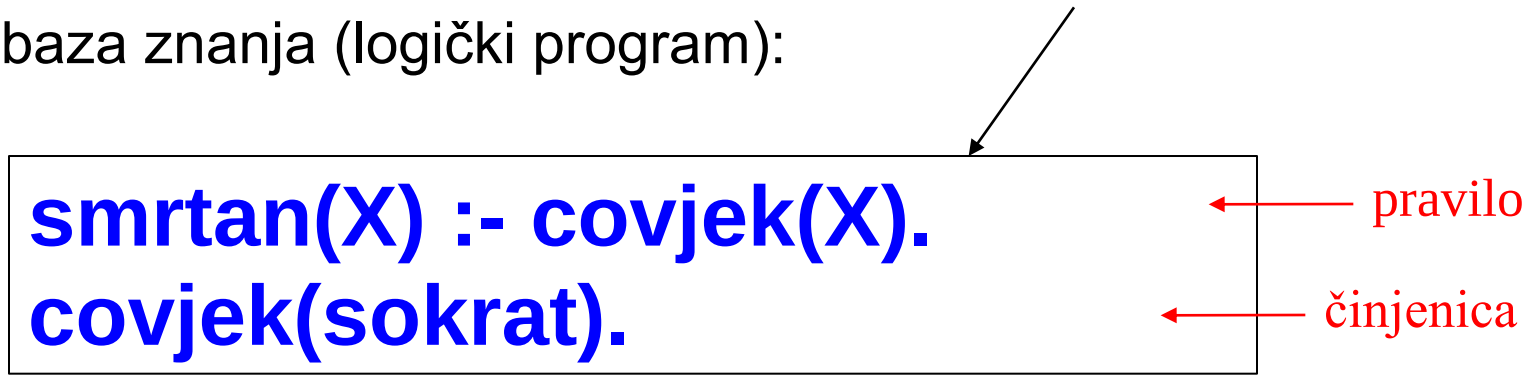
- Programi u Prologu sačinjeni su od slijeda **pravila**
- **Svako pravilo je FOPL-formula u Hornovom obliku:**
  - **~P1 ∨ ~P2 ∨ ~P3 ∨ ... ∨ ~Pn ∨ Q**
  - tj. klauzula u kojoj je najviše jedan literal pozitivan
- **Ekvivalentno: (P1 ∧ P2 ∧ P3 ∧ ... ∧ Pn) → Q**
- Specijalan slučaj za *Pi = true*: ***true* → Q == Q**
- **Definitna Hornova klauzula:** točno jedan literal je pozitivan
- Nažalost, ne može se svaka formula pretvoriti u (definitni) Hornov oblik .... **~P → Q, P → (Q ∨ R)**

- Zaključivanje nad Hornovim formulama: **rezolucija opovrgvanjem** (krenuvši od **~Q**)

Primjer programa u Prologu

$\forall x\,(COVJEK(x) \rightarrow SMRTAN(x)) \wedge COVJEK(Sokrat)$

- baza znanja (logički program):



- varijable velikim, predikatni simboli malim slovima
- implikacija u obliku **konzekvens :- antecedens**
- svaki redak završava točkom
- varijable su implicitno univerzalno kvantificirane

Primjer upita (1)

```
smrtan(X) :- covjek(X).
covjek(sokrat).
```

- sada možemo **postavljati upite**:

```
?- covjek(sokrat).
Yes
?- smrtan(sokrat).
Yes
?- smrtan(X).
X=sokrat
Yes
```

Konjunkcija atoma

- antecedens može sadržavati veći broj atoma:

```
covjek(X) :-
  sisavac(X), govori(X).
```


```
covjek(X) :-
  sisavac(X),
  govori(X),
  placa_porez(X).
```

Disjunkcija atoma

- jedan predikat može biti definiran s više **stavaka**:

```
smrtan(X) :- covjek(X).
smrtan(X) :- ziv(X).
```

- između stavaka se **podrazumijeva konjunkcija**
- tomu je ekvivalentno:

```
smrtan(X) :-
  covjek(X); ziv(X).
```

operator *"ili"*

- provjerite!

Višemjesni predikati

- predikati mogu biti *n*-mjesni:

```
ucitelj(sokrat,platon).
ucitelj(kratil,platon).
ucitelj(platon,aristotel).
```

- kako definirati predikat **ucenik** pomoću **ucitelj** ?

```
ucenik(X,Y) :- ucitelj(Y,X).
```

- kako definirati jednomjesni predikat **ucen** ?

```
ucen(X) :- ucitelj(Y,X).
```

ucen(X) :- ucitelj(Y,X).

- kako je kvantificirana varijabla **Y**?
- **sve** varijable su implicitno **univerzalno** kvantificirane u cijelom pravilu, ali **Y** se javlja samo u antecedensu pa možemo reći da je **egzistencijalno** kvantificirana

- dokaži:

$\forall x \, \forall y \, (\text{UCITELJ}(y,x) \rightarrow \text{UCEN}(x)) \equiv \forall x \, (\exists y \, \text{UCITELJ}(y,x) \rightarrow \text{UCEN}(x))$

Primjer upita (2)

- primjeri upita:

```
?- ucitelj(X,platon).
X=sokrat
X=kratil
Yes
?- ucitelj(sokrat,Y),ucitelj(kratil,Y).
Y=platon
Yes
?- ucenik(X,_).
X=platon
X=platon
X=aristotel
Yes
?- ucenik(aristotel,sokrat).
No
```

- definirajmo **tranzitivnu** relaciju **SLJEDBENIK(x,y)**
  - "*x* je ucenik od *y*"
  - "*x* ucenik od nekog *z*, a *z* je sljedbenik od *y*"

```
sljedbenik(X,Y):-
  ucenik(X,Y).
sljedbenik(X,Y):-
  ucenik(X,Z),
  sljedbenik(Z,Y).
```

← trivijalan slučaj

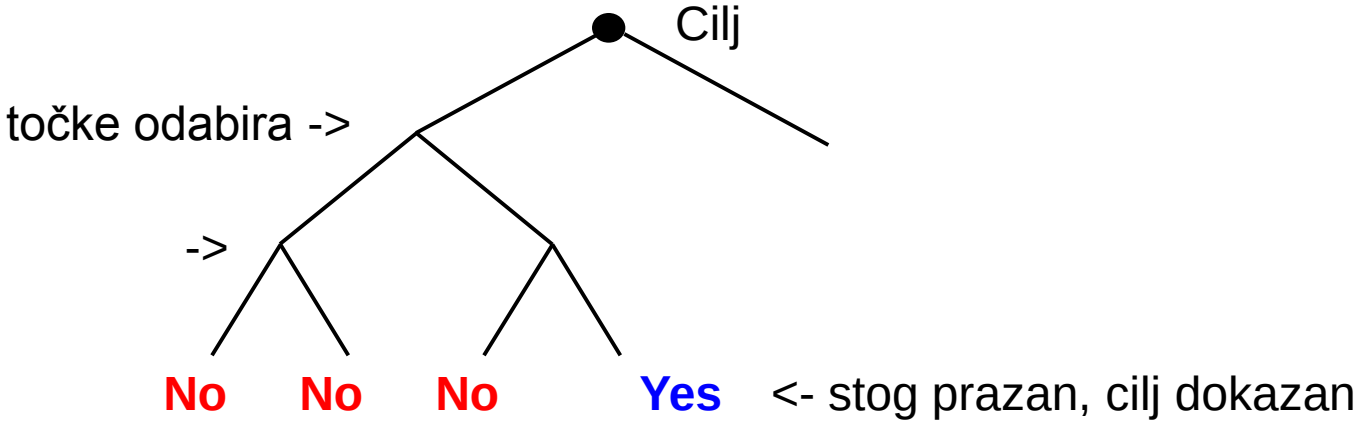
← rekurzivan slučaj

- upit:

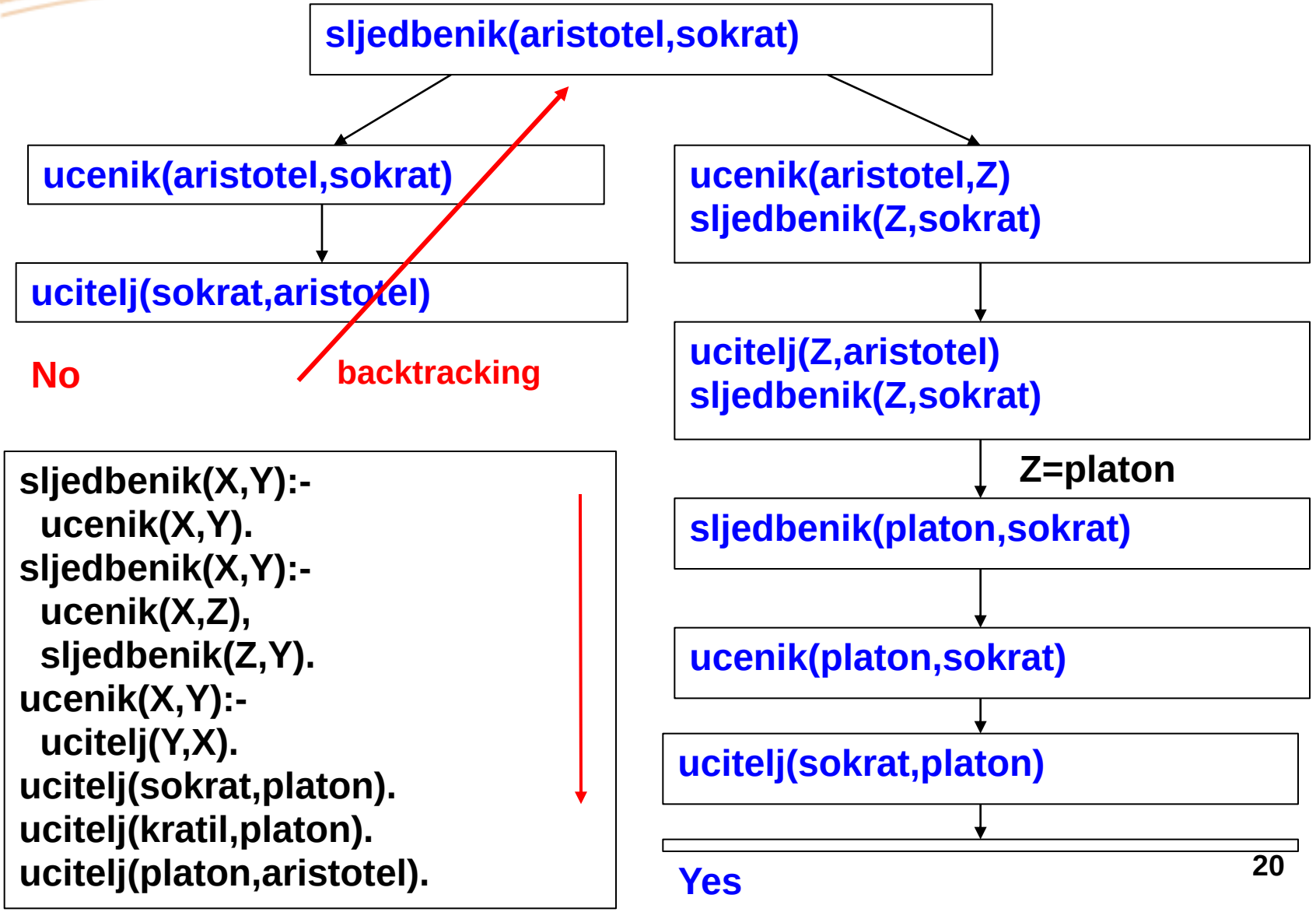
```
?- sljedbenik(aristotel,sokrat).
Yes
```

BACKTRACKING

- Prolog dokazuje **unatrag** od cilja k premisama, metodom **pretraživanja u dubinu**
  - **antecedens postaje novi međucilj koji treba dokazati i koji se dodaje na stog**
- Kada jedna grana dokaza ne uspije, Prolog se vraća (BACKTRACKS) na zadnju točku odabira
- Ako iscrpi sve mogućnosti, vraća **No**, inače **Yes**



BACKTRACKING



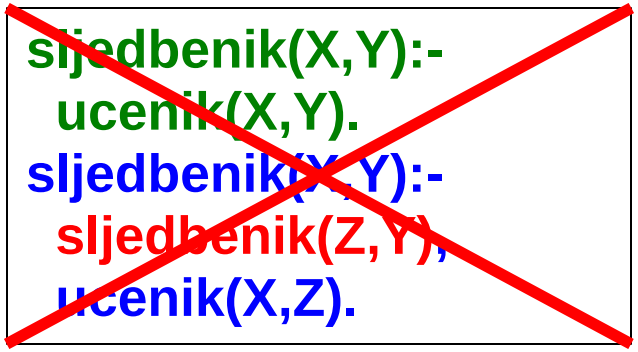

```
sljedbenik(X,Y):-
  ucenik(X,Y).
sljedbenik(X,Y):-
  ucenik(X,Z),
  sljedbenik(Z,Y).
ucenik(X,Y):-
  ucitelj(Y,X).
ucitelj(sokrat,platon).
ucitelj(kratil,platon).
ucitelj(platon,aristotel).
```

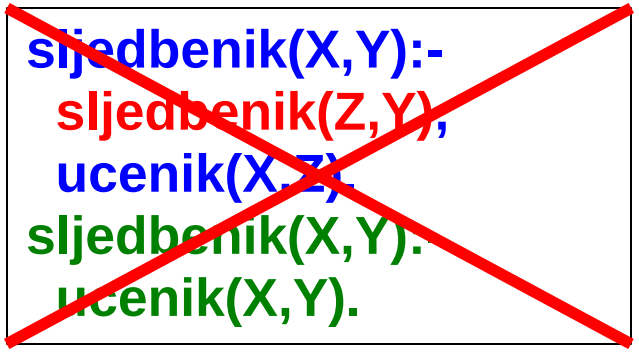
Deklarativno vs. proceduralno

- Nažalost, Prolog nije čisto deklarativan pa je redoslijed stavaka i atoma itekako bitan:

```
sljedbenik(X,Y):-
  ucenik(X,Y).
sljedbenik(X,Y):-
  ucenik(X,Z),
  sljedbenik(Z,Y).
```

```
sljedbenik(X,Y):-
  ucenik(X,Z),
  sljedbenik(Z,Y).
sljedbenik(X,Y):-
  ucenik(X,Y).
```





- *"out of stack"*

UNIFIKACIJA

- U svakom koraku zaključivanja Prolog nastoji **unificirati** trenutni cilj sa stoga s konzekvensima pravila ili činjenicama u bazi znanja
- unificirati možemo i eksplicitno:

```
?- ucitelj(Z,aristotel) = ucitelj(platon,aristotel)
Z = platon
Yes

?- ucenik(Z,aristotel) = ucenik(platon,Z)
No

?- X = f(X).
X = f(f(f(...)))
```

operator unifikacije

nema provjere pojavljivanja varijable! (*occurs check*)

NEGACIJA

- Hornov oblik ne dopušta negaciju u antecedensu, ali Prolog dopušta operator **not**

```
covjek(X) :-
  govori(X),
  not(ima(X,perje)).
```

- **negacija pomoću neuspjeha** (NAF)
  - ako **P(x)** ne možeš dokazati,
    onda je istinito **not(P(x))**, inače je lažno
- pretpostavljamo **zatvorenost svijeta** (CWA)
  - sve što nije u bazi znanja je lažno

NEGACIJA

- baza znanja:

```
ima(polinezija,perje).
govori(polinezija).
govori(sokrat).
covjek(X) :-
  govori(X),
  not(ima(X,perje)).
```

- upiti:

```
?- covjek(sokrat).
Yes
?- covjek(polinezija).
No
?- not(covjek(polinezija)).
Yes
```

- **Zadatak 2.5: Primjena Prologa u porodičnoj domeni** (3 boda)

  U programskom jeziku Prolog definirajte bazu znanja koja opisuje porodične relacije i omogućuje izvođenje novih zaključaka. U baza znanja definirajte činjenice **roditelj, musko, zensko** i **dob,** te zatim pravila **otac**, **majka**, **brat**, **sestra**, **djed**, **baka**, **ujak**, **predak** i **srodan**.

  Definirajte upite kojima se iz baze znanja dohvaćaju **(1)** svi roditeljski parovi, **(2)** sve osobe koje su nekome ujak, **(3)** sva djeca koja imaju stariju braću, **(4)** sve prabake koje su to već deset godina, **(5)** svi potomci neke zadane osobe, **(6)** sve punoljetne osobe bez djece, **(7)** svi parovi osoba koje nisu u srodstvu, **(8)** sve majke s troje ili više djece, **(9)** sve majke s točno troje djece te **(10)** sve majke s troje ili manje djece.

Laboratorijski zadatak

```
roditelj(zeus,atena).
roditelj(zeus,apolo).
roditelj(hera,ares).
...
musko(zeus).
zensko(hera).
...
otac(X,Y):- ...
```