

Prof.dr.sc. Bojana Dalbello Bašić

Fakultet elektrotehnike i računarstva

Zavod za elektroniku, mikroelektroniku, računalne i inteligentne sustave

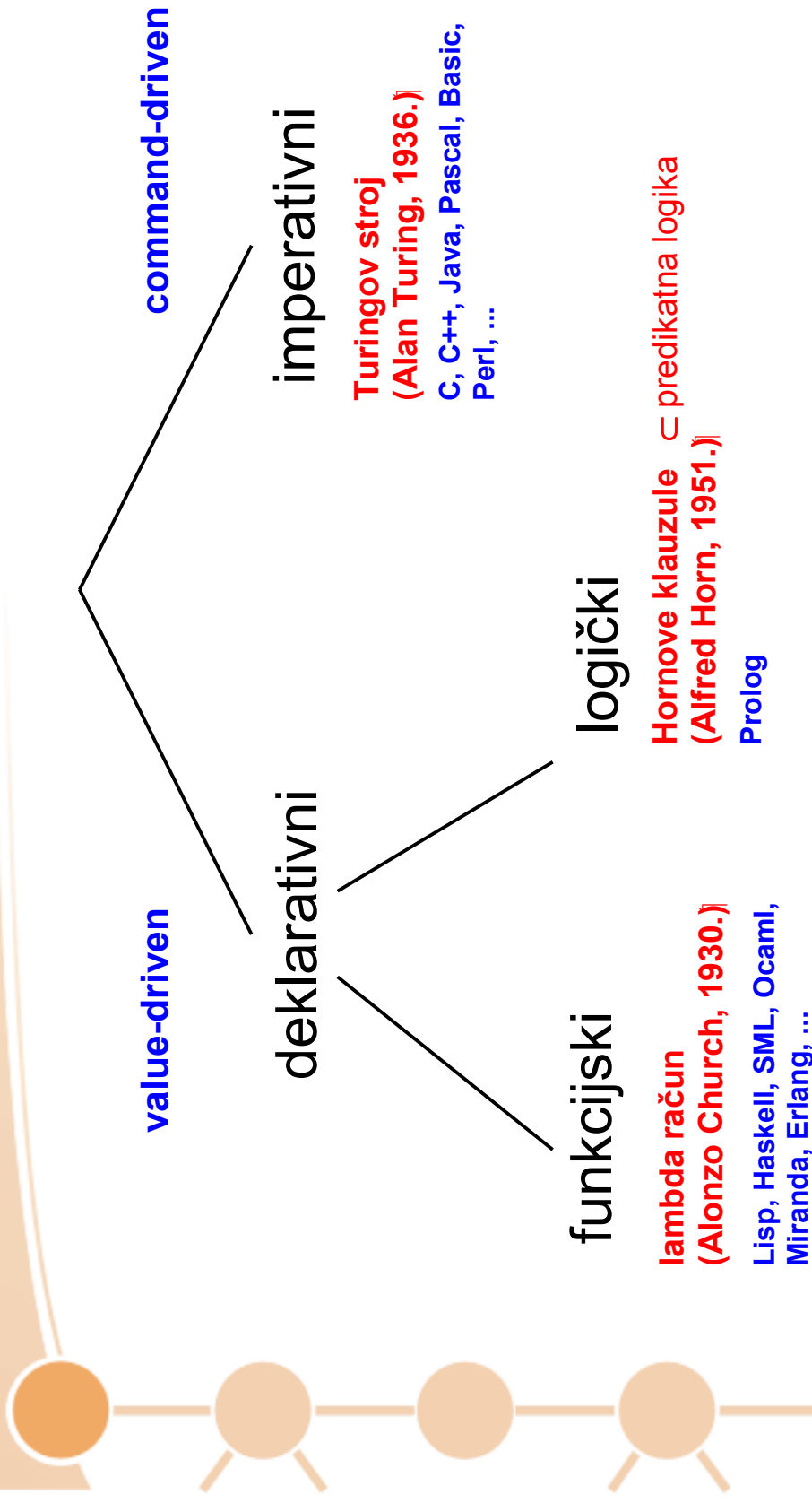
www.zemris.fer.hr/~bojana
bojana.dalbello@fer.hr

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 sva ekspresivnost logike

DEKLARATIVNI 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 implicitno stanje
 - **nema popratnih efekata (engl. side-effects)**
 - programiranje s **izrazima**
 - funkcijski jezici: izraz=funkcija
 - logički jezici: izraz=relacija

POP RATNI 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


POPRAVNI EFEKTI

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

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

false??

- 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))

- 
- čisto deklarativni (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:
 - učinkovitost
 - 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 to iziskuje 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
  end
  return a
end
```

Haskell:

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


- **Prolog** = *Programming in Logic*
 - deklarativni logički programski jezik
 - 1972.: Alan Colmerauer, Robert Kowalski, Philippe Rousset
 - 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:**
 - $\sim P1 \vee \sim P2 \vee \sim P3 \vee \dots \vee \sim Pn \vee Q$
 - klauzula u kojoj je najviše jedan literal pozitivan
- **Ekvivalentno: $(P1 \wedge P2 \wedge P3 \wedge \dots \wedge Pn) \rightarrow Q$**
- Specijalan slučaj za $Pi = true$: **$true \rightarrow Q == Q$**
- Zaključivanje nad Hornovim formulama:
 - primjeni modus ponens dokle god je moguće
- Nažalost, ne može se svaka formula pretvoriti u Hornov oblik **$\sim P \rightarrow Q$**
- ... niti možemo s MP dokazati svaku logičku posljedicu

NEPOTPUNOST!



Primjer programa u Prologu

$$\forall x (\text{COVJEK}(x) \rightarrow \text{SMRTAN}(x)) \wedge \text{COVJEK}(\text{Sokrat})$$

- baza znanja (logički program):

smrtan (x) :- covjek (x) .
covjek (sokrat) .

— pravilo
— činjenica

- 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) .
```

operator “*i*”

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

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

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

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

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

- provjerite! operator “;”

- 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).
```

$\text{ucen}(X) :- \text{ucitelj}(\mathbf{y}, X) .$

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

- dokaži:

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

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=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) .

— trivijalni slučaj

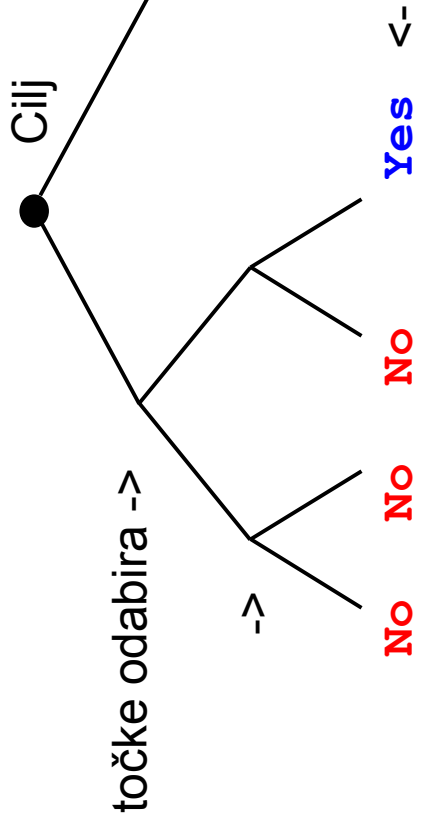
— rekurzivni slučaj

- upit:

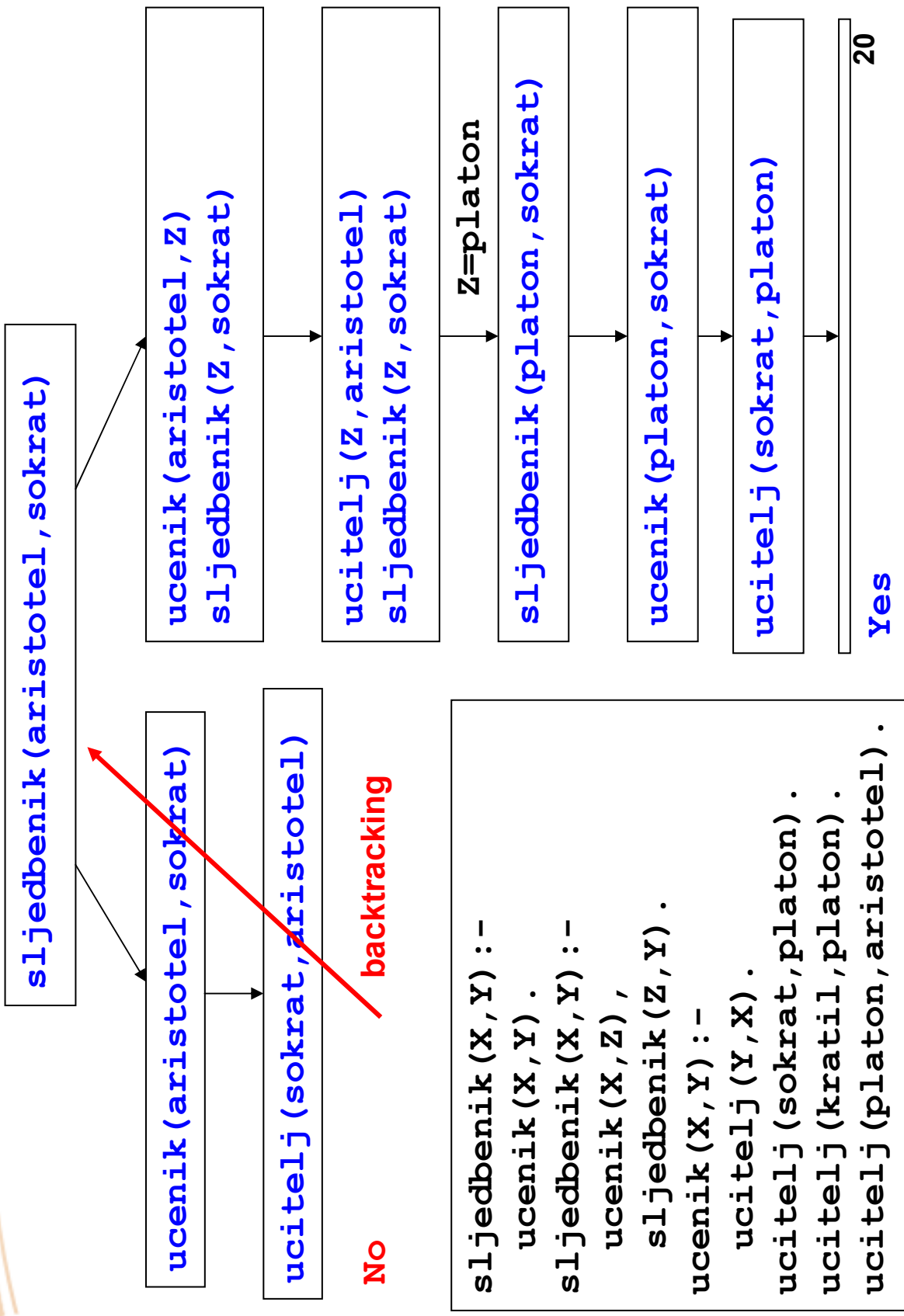
?- sljedbenik(aristotel,sokrat) .
 Yes

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



Deklarativno vs. proceduralno

- Nažalost, Prolog nije čisto deklarativan pa je redosljed 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) .
```

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

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

- “out of stack”



- 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
```

operator
unifikacije

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

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

nema provjere pojavljivanja
varijable! (“occur-check”)

- 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 $\text{not}(P(x))$ istinit, inače je lažan
- pretpostavljamo **zatvorenost svijeta (CWA)**
 - sve što nije u bazi znanja je lažno

- baza znanja:

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

- upiti:

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


■ **Zadatak 2.5: Primjena Prologa u porodičnoj domeni** **(5 bodova)**

U programskom jeziku Prolog **definirajte bazu znanja koja opisuje vaše porodične relacije** i omogućuje izvođenje novih zaključaka. Baza znanja neka za sve članove porodice (njih barem deset) definira ekstenziju predikata roditeljstva (2-mjesna relacija), te predikata muškog i ženskog spola (1-mjesna relacija odnosno svojstvo). Nad takvom bazom definirajte predikate **otac**, **majka**, **brat**, **sestra**, **djed**, **baka**, **ujak**, **predak**, i **rod**.

Uvjerite se u **deklarativnu i proceduralnu ispravnost** svakog od ovih predikata.

Definirajte upite kojima se iz baze znanja dohvaćaju (1) sve osobe koje su nekome ujak, (2) svi potomci zadane osobe i (3) svi parovi osoba koji nisu u rodu.

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