Umjetna inteligencija

7. Logičko programiranje u Prologu

prof. dr. sc. Bojana Dalbelo Bašić
izv. prof. dr. sc. Jan Šnajder

Sveučilište u Zagrebu
Fakultet elektrotehnike i računarstva

Ak. god. 2019./2020.

v1.10

Sadržaj

1. Logičko programiranje i Prolog
2. Zaključivanje nad Hornovim klauzulama
3. Programiranje u Prologu
4. Nedeklarativni aspekti Prologa

Sadržaj

1. Logičko programiranje i Prolog
2. Zaključivanje nad Hornovim klauzulama
3. Programiranje u Prologu
4. Nedeklarativni aspekti Prologa

Logičko programiranje

- **Logičko programiranje**: uporaba logičkog zaključivanja kao načina programiranja
- Osnovna ideja: definirati problem kao skup logičkih formula, zatim dati računalu da riješi problem (izvođenje programa = zaključivanje)
- Pristup tipičan za **deklarativno programiranje**: izraziti logiku izračunavanja, ne zamarati se upravljačkim tokom
- Usredotočavamo se na **deklarativan** aspekt programa, a ne na to kako se on izvodi (**proceduralan** aspekt)
- Međutim, ipak nam trebaju neki upravljački mehanizmi, stoga:

Algoritam = Logika + Upravljanje

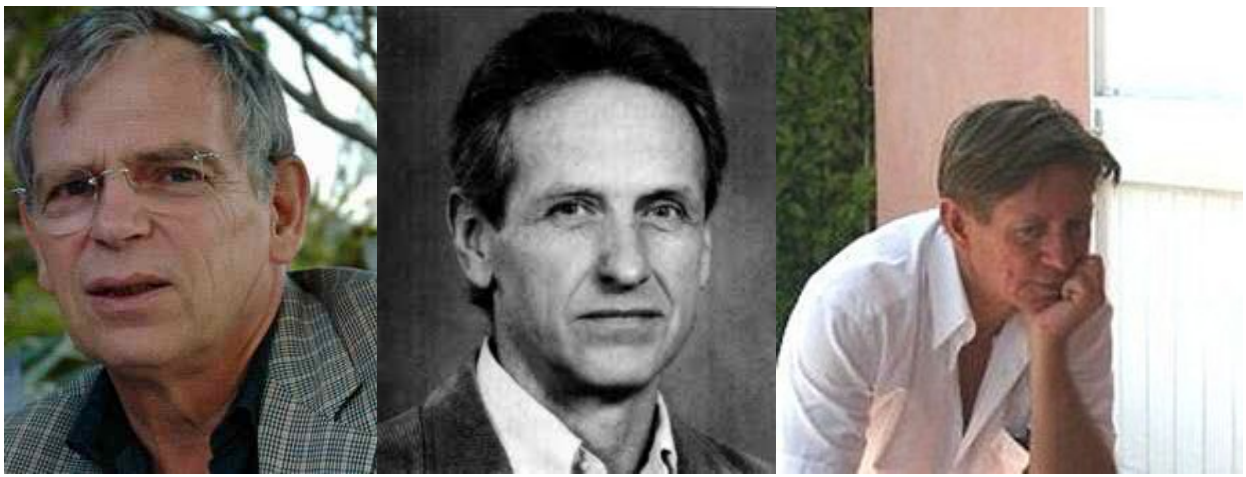
- Različito od automatskog dokazivanja teorema jer:
  1. u program su uključeni eksplicitni upravljački mehanizmi
  2. nije podržana puna ekspresivnost FOL-a

Deklarativno programiranje

- Opisuje **što** se izračunava umjesto **kako** se izračunava (upravljanje)
- Programer definira skup ograničenja koji definiraju prostor rješenja, dok je nalaženje rješenja prepušteno interpreteru
- Glavne značajke deklarativnih programskih jezika:
  - **eksplicitno** stanje umjesto implicitnog
  - **nema popratnih efekata** (engl. *side effects*) ili su oni ograničeni
  - programiranje s **izrazima**
- Dva glavna pristupa:
  - **funkcijsko programiranje**: izraz je funkcija
  - **logičko programiranje**: izraz je relacija (predstavljena predikatom)
- **Prednosti**: formalna konciznost, visok stupanj apstrakcije, pogodno za formalnu analizu, manje pogrešaka
- **Nedostatci**: neučinkovitost, strma krivulja učenja, nije široko prihvaćeno

Prolog

- **Prolog** – "Programming in Logic"
- Deklarativni programski jezik
- "The offspring of a successful marriage between natural language processing and automated theorem-proving"
- Alan Colmerauer, Robert Kowalski, i Philippe Rousse 1972. godine



Colmerauer, A., & Roussel, P. (1996). The Birth of Prolog. In History of programming languages (pp. 331–367)

SWI Prolog

www.swi-prolog.org

SWI-Prolog's features

HOME | DOWNLOAD | DOCUMENTATION | TUTORIALS | COMMUNITY | USERS | WIKI

Overview

SWI-Prolog is a versatile implementation of the Prolog language. Although SWI-Prolog gained its popularity primarily in education, its development is mostly driven by the needs for **application development**. This is facilitated by a rich interface to other IT components by supporting many document types and (network) protocols as well as a comprehensive low-level interface to C that is the basis for high-level interfaces to C++, Java (bundled), C#, Python, etc (externally available). Data type extensions such as dicts and strings as well as full support for Unicode and unbounded integers simplify smooth exchange of data with other components.

SWI-Prolog aims at **scalability**. Its robust support for multi-threading exploits multi-core hardware efficiently and simplifies embedding in concurrent applications. Its *Just In Time Indexing* (JITI) provides transparent and efficient support for predicates with millions of clauses.

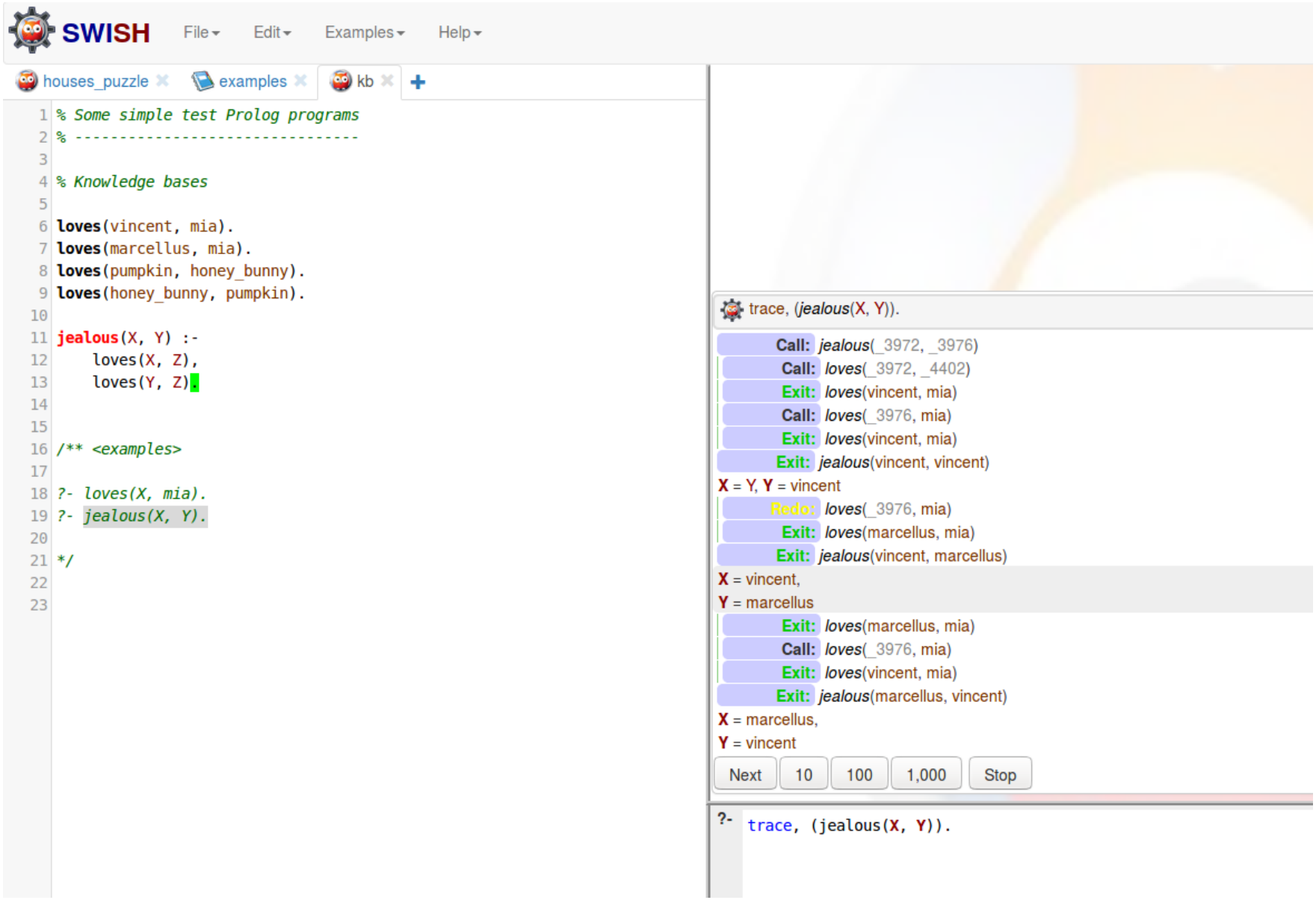
SWI-Prolog **unifies many extensions** of the core language that have been developed in the Prolog community such as *tabling*, *constraints*, *global variables*, *destructive assignment*, *delimited continations* and *interactors*.

SWI-Prolog offers a variety of **development tools**, most of which may be combined at will. The native system provides an editor written in Prolog that is a close clone of Emacs. It provides *semantic* highlighting based on real time analysis of the code by the Prolog system itself. Complementary tools include a graphical debugger, profiler and cross-referencer. Alternatively, there is a mode for GNU-Emacs and, Eclipse plugin called PDT and a VSC plugin, each of which may be combined with the native graphical tools. Finally, a *computational notebook* and web based IDE is provided by SWISH. SWISH is a versatile tool that can be configured and extended to suit many different scenarios.

SWI-Prolog provides an add-on distribution and installation mechanism called **packs**. A *pack* is a directory with minimal organizational conventions and a *control* file that describes the origin, version, dependencies and automatic upgrade support. Packs

SWI Prolog

https://swish.swi-prolog.org/



Sadržaj

1. Logičko programiranje i Prolog
2. Zaključivanje nad Hornovim klauzulama
3. Programiranje u Prologu
4. Nedeklarativni aspekti Prologa

Hornove klauzule (1)

- Prolog koristi podskup FOL-a nazvan **Hornova klauzalna logika**

Hornova klauzula

Hornova klauzula je klauzula (disjunkcija literala) s najviše jednim pozitivnim literalom:

$$\neg P_1 \vee \neg P_2 \vee \cdots \vee \neg P_n \vee Q$$

ili, ekvivalentno:

$$(P_1 \wedge P_2 \wedge \cdots \wedge P_n) \rightarrow Q$$

Negativni literali čine **tijelo** klauzule, dok pozitivan literal čini **glavu**.

Definitna klauzula

Definitna klauzula je Hornova klauzula s točno jednim pozitivnim literalom.

- Hornova klauzula može biti propozicijska ili prvoga reda

Hornove klauzule (2)

- Program u Prologu sačinjen je od niza definitnih klauzula
- Svaka definitna klauzula definira **pravilo zaključivanja** ili **činjenicu**
- Činjenica je definitna klauzula čiji je oblik $True \rightarrow Q \equiv Q$
- Pravila zaključivanja i činjenice intuitivan su način formalizacije ljudskog znanja
- Hornova klazula koji sadržava samo negativne literale naziva se **ciljna klauzula**
- Za zadani logički program, potrebno je dokazati ciljnu klauzulu pomoću **rezolucije opovrgavanjem**

Hornove klauzule (3)

- Hornove klauzule **ograničene su ekspresivnosti**: nije moguće svaku FOL formulu prikazati kao Hornovu klauzulu
  - Npr., $\neg P \rightarrow Q$ or $P \rightarrow (Q \vee R)$
- U praksi se pokazuje da to nije preveliko ograničenje
- Prednost ograničenja na Hornove klauzule: možemo koristiti učinkovite postupke zaključivanja temeljene na **rezoluciji** s **ulančavanjem unaprijed** ili **ulančavanjem unazad**
- Ulančavanje unaprijed/unazad nad Hornovim klauzulama je **potpuno**
- Propozicijske Hornove klauzule: vremenska složenost zaključivanja **linearna** je u broju klauzula
- Hornove klauzule prvoga reda: zaključivanje je neodlučivo, ali općenito učinkovitije nego u neograničenoj FOL

Ulančavanje unazad

- Počevši od **baze znanja s činjenicama i pravilima** (logičkim programom) Γ i **negiranim ciljem** $\neg P$, pokušavamo izvesti NIL

$$\neg P \text{ se razrješava sa } C_1 \in \Gamma \text{ i generira nov negirani cilj } \neg P_2$$
$$\neg P_2 \text{ se razrješava sa } C_2 \in \Gamma \text{ i generira nov negirani cilj } \neg P_3$$
$$\vdots$$
$$\neg P_k \text{ se razrješava sa } C_k \in \Gamma \text{ i generira NIL}$$

- Na postupak možemo gledati kao na **pretraživanje prostora stanja**:
  - svako stanje je negirani trenutačan cilj
  - početno stanje: $\neg P$
  - ciljno stanje: klauzula NIL
  - operator: razrješavanje cilja $\neg P_i$ s klauzulom C_j iz Γ
- **NB:** Hornove klauzule zatvorene su pod rezolucijom: rezolventa dviju Hornovih klauzula i sama je Hornova klauzula

Ulančavanje unazad – primjer 1

- Logički program:

$$
\begin{array}{lll}
(1) & A & \equiv A \\
(2) & B & \equiv B \\
(3) & (A \wedge B) \rightarrow C & \equiv \neg A \vee \neg B \vee C \\
(4) & (C \vee D) \rightarrow E & \equiv \neg C \vee E \\
(5) & & \neg D \vee E
\end{array}
$$

- Cilj: klauzula E
- Početno stanje: $\neg E$
- Korak 1: Razrješavanje cilja $\neg E$ i klauzule (4), novi cilj je $\neg C$
- Korak 2: Razrješavanje cilja $\neg C$ i klauzule (3), novi cilj je $\neg A \vee \neg B$
- Korak 3: Razrješavanje cilja $\neg A \vee \neg B$ i klazule (1), novi cilj je $\neg B$
- Korak 4: Razrješavanje cilja $\neg B$ i klauzule (2), izvodi se NIL

Ulančavanje unazad – primjer 2

- Logički program:

$$
\begin{array}{lll}
(1) & A & \equiv A \\
(2) & B & \equiv B \\
(3) & (A \wedge B) \rightarrow D & \equiv \neg A \vee \neg B \vee D \\
(4) & (C \vee D) \rightarrow E & \equiv \neg C \vee E \\
(5) & & \neg D \vee E
\end{array}
$$

- Cilj: klauzula E
- Početno stanje: $\neg E$
- Korak 1: Razrješavanje cilja $\neg E$ i klauzule (4), novi cilj: $\neg C$
- Korak 2: Vraćanje (backtracking) na zadnju točku odabira
- Korak 3: Razrješavanje cilja $\neg E$ i klauzule (5), novi cilj: $\neg D$
- Korak 4: Razrješavanje cilja $\neg D$ i klauzule (3), novi cilj: $\neg A \vee \neg B$
- Korak 5: Razrješavanje cilja $\neg A \vee \neg B$ i klauzule (1), novi cilj: $\neg B$
- Korak 6: Razrješavanje cilja $\neg B$ i klauzule (2), izvodi se NIL

Ulančavanje unazad – algoritam

- Nedeterministički algoritam za rezoluciju nad Hornovim klauzulama

Ulančavanje unazad

function $\mathrm{BackwardChaining}(P, \Gamma)$
  if $P = \mathrm{NIL}$ **then return** $true$
  $L \leftarrow \mathrm{SelectLiteral}(P)$
  $C \leftarrow \mathrm{SelectResolvingClause}(L, \Gamma)$
  if $C = fail$ **then return** $false$
  $P' \leftarrow \mathrm{resolve}(P, C)$
  $\mathrm{BackwardChaining}(P', \Gamma)$

- $\mathrm{SelectLiteral}$ odabire jedan literal iz klauzule P (negirani literal iz negiranog cilja)
- $\mathrm{SelectResolvingClause}$ odabire jednu klauzulu iz Γ čija se glava (koja je jedini pozitivni literal Hornove klauzule) može razriješiti sa L
- Može biti više takvih klauzula, stoga se pretraživanje grana

Ulančavanje unazad u Prologu

- Poredak klauzula u Γ definira programer (tipično: prvo se navode specifičnije, a zatim općenitije klauzule)
- Poredak negativnih literala u svakoj klauzuli (tj. poredak atoma u antecedentu pravila) također definira programer
- Pretraživanje prostora stanja provodi se **pretraživanjem u dubinu**: kada je negirani cilj P razrješen s klauzulom C, negativni literali iz C smještaju se **u izvornome poretku na početak** klauzule P
- $\mathrm{SelectLiteral}$ odabire **prvi** literal u P. Dakle, P implementira **stog** (LIFO)
- Ova strategija dokaza naziva se **SLD** (Selective Linear Definite clause resolution)

Sadržaj

1. Logičko programiranje i Prolog
2. Zaključivanje nad Hornovim klauzulama
3. Programiranje u Prologu
4. Nedeklarativni aspekti Prologa

Činjenice i pravila u Prologu

$$\forall x \big(\text{HUMAN}(x) \rightarrow \text{MORTAL}(x) \big) \wedge \text{HUMAN}(Socrates)$$

```
mortal(X) :- human(X).   % pravilo
human(socrates).         % činjenica
```

- Varijable su pisane velikim slovima, a predikati malim
- Implikacije su oblika `konzekvent :- antecedent`
- Varijable su implicitno univerzalno kvantificirane
- Svaki redak završava točkom

Upiti u Prologu

```
?- human(socrates).   % Je li Sokrat čovjek?
true
?- human(doughnut).   % Je li krafna čovjek?
false
?- mortal(socrates). % Zaključivanje: je li Sokrat smrtan?
true
?- mortal(X).         % Tko je smrtan?
X = socrates
true
```

Dodavanje uvjeta u pravilo

$$\forall x \big((\text{MAMMAL}(x) \wedge \text{SPEAKS}(x)) \rightarrow \text{HUMAN}(x) \big)$$

```
human(X) :- mammal(X), speaks(X).  % zarez označava "i"
```

$$\forall x \big((\text{MAMMAL}(x) \wedge \text{SPEAKS}(x) \wedge \text{PAYS\_TAXES}(x)) \rightarrow \text{HUMAN}(x) \big)$$

```
human(X) :-
  mammal(X),
  speaks(X),
  pays_taxes(X).
```

Dodavanje disjunkcija u pravilo

$$\forall x \big((\text{HUMAN}(x) \vee \text{ALIVE}(x)) \rightarrow \text{MORTAL}(x) \big)$$

- Diskunkcija u uvjetu pravila može se napisati na dva načina
- Ili pravilo razdvojimo u dvije zasebne klauzule:

```
mortal(X) :- human(X).   % prva klauzula
mortal(X) :- alive(X).   % druga klauzula
```

- Ili uvedemo disjunkciju u tijelo pravila:

```
mortal(X) :-
  human(X); alive(X)   % točka-zarez označava "ili"
```

n-arni predikati

- Binarni predikati modeliraju binarne relacije:

```
teacher(socrates, plato).    % Socrat je Platonov učitelj
teacher(cratylus, plato).
teacher(plato, aristotle).
```

- Pravilo za učenika inverzno je pravilu za učitelja:

```
disciple(X, Y) :- teacher(Y, X).
```

- Pravilo koje definira da je netko podučavan od učitelja:

```
taught(X) :- disciple(X, Y).
```

- Budući da nas vrijednost varijable `Y` ne zanima, možemo pisati:

```
taught(X) :- disciple(X, _).
```

Primjeri upita

```
?- teacher(X, plato).   % Tko je Platonov učitelj?
X = socrates
X = cratylus
true
?- teacher(socrates, Y), teacher(cratylus, Y).
   % Koga obojica podučavaju?
Y = plato
true
?- taught(X).  % Tko je podučavan?
X = plato
X = plato
X = aristotle
true
?- disciple(aristotle, socrates).
false
```

Rekurzivno definirani predikati

- $\mathrm{DISCIPLE}(x, y)$ definira samo izravnu relaciju
- Kako možemo definirati tranzitivnu relaciju?
- Npr., $\mathrm{FOLLOWER}(x, y)$, akko je x izravan ili posredan sljedbenik filozofa y:
  - Osnovni slučaj: x je učenik od y
  - Rekurzivni slučaj: x je učenik od z te z je sljedbenik od y

```
follower(X, Y) :-   % osnovna klauzula
  disciple(X, Y).
follower(X, Y) :-   % rekurzivna klauzula
  disciple(X, Z),
  follower(Z, Y).
```

```
?- follower(aristotle, socrates).
true
```

Prologovo stablo pretraživanja

- Primjenom SLD rezolucije na logički program i negirani cilj generira se stablo pretraživanja koje je zapravo **stablo dokaza**
- Svaki čvor je **stog negativnih literala** koje treba razriješiti
- Cilj je izvesti NIL, tj. isprazniti stog
- Prolog pokušava razriješiti vršni literal stoga s **literalom glave** svake klauzule u programu (prisjetimo se: stog sadržava negativne literale, dok su glave klauzula pozitivni literali)
- Takvi literali potencijalno se mogu **komplementarno unificirati**
- Klauzule se pretražuju **od vrha prema dnu** programa (zbog toga je poredak klauzula važan)

Prologovo stablo pretraživanja

- Ako se vršni literal L može komplementarno unificirati pomoću MGU θ s glavom klauzule C, onda se L skida sa stoga, tijelo od C dolazi na vrh stoga, na literale stoga primjenjuje se θ, i pretraga se nastavlja
- Ako se niti jedna klauzula programa ne može komplementarno unificirati sa L, pretraga se vraća **(backtracking)** na posljednju točku odabira $\Rightarrow$ **pretraživanje u dubinu**
- Ako ste stog isprazni, to znači da je izvedena klauzula NIL, pa procedura vraća `true`
- Ako je pretraga završena a stog nije prazan, procedura vraća `false`

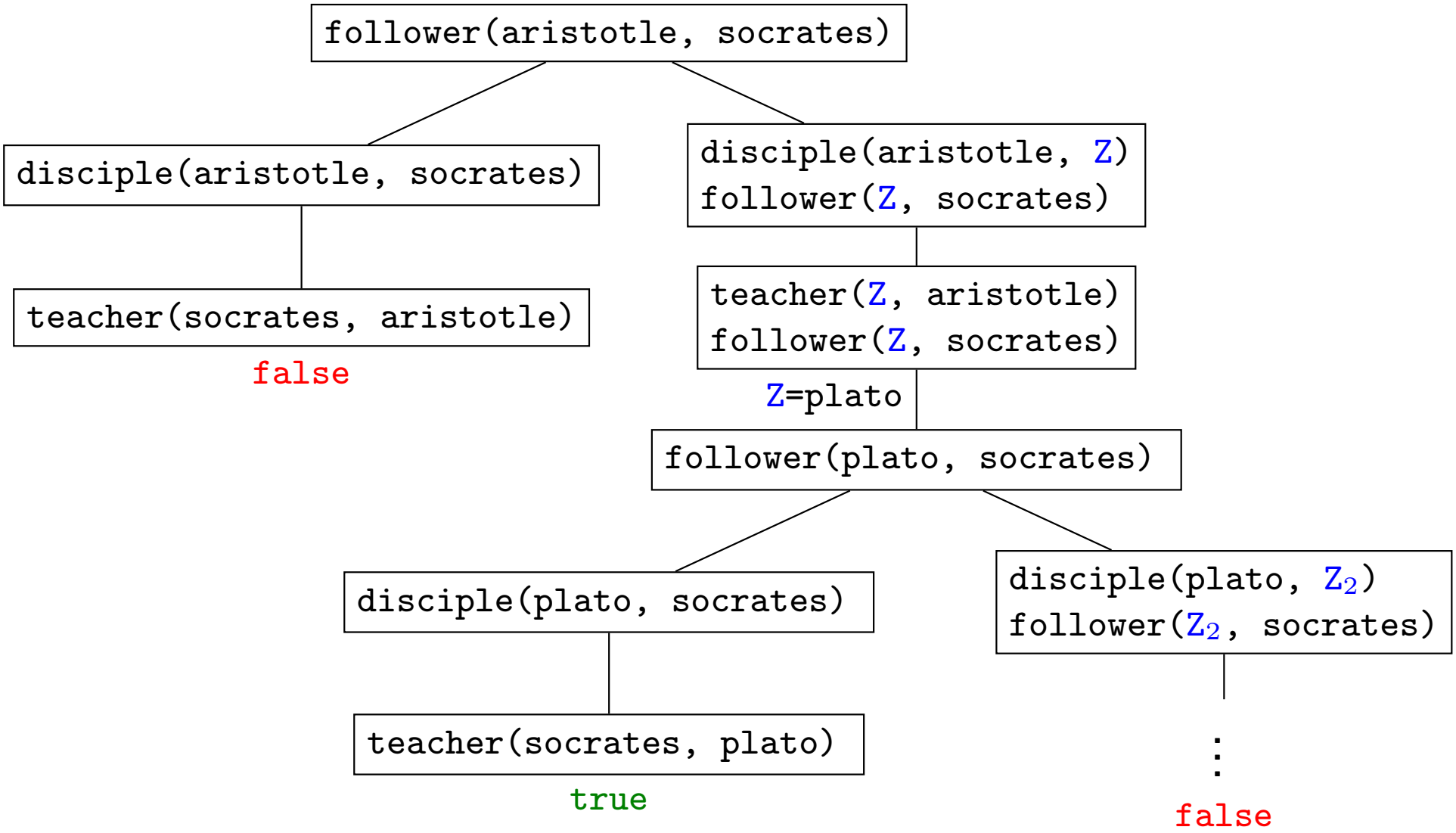
Primjer programa u Prologu

```
% Pravila
follower(X, Y) :-
  disciple(X, Y).
follower(X, Y) :-
  disciple(X, Z),
  follower(Z, Y).

disciple(X, Y) :-
  teacher(Y, X).

% Činjenice
teacher(socrates, plato).
teacher(cratylus, plato).
teacher(plato, aristotle).
```

Primjer stabla dokaza



Primjer stabla dokaza – napomene

- Svaki čvor = prema dolje rastući stog negativnih literala
- Stablo se grana kod razrješavanja literala `follower`, jer za predikat `follower` postoje dva pravila
- Na zahtjev korisnika, nakon što je cilj dokazan, Prolog može nastaviti pretragu za alternativnim rješenjima (koja u ovom slučaju neće dati rezultata)
- Nemojte brkati stog negativnih literala (koji odgovara stanju u prostoru pretraživanja) sa stogom algoritma pretraživanja u dubinu (koji nije prikazan na prethodnome slajdu)

Prologov trag izvođenja

```
[trace] ?- follower(aristotle, socrates).
   Call:  (7) follower(aristotle, socrates) ?  creep
   Call:  (8) disciple(aristotle, socrates) ?  creep
   Call:  (9) teacher(socrates, aristotle) ?  creep
   Fail:  (9) teacher(socrates, aristotle) ?  creep
   Fail:  (8) disciple(aristotle, socrates) ?  creep
   Redo:  (7) follower(aristotle, socrates) ?  creep
   Call:  (8) disciple(aristotle, _G5025) ?  creep
   Call:  (9) teacher(_G5024, aristotle) ?  creep
   Exit:  (9) teacher(plato, aristotle) ?  creep
   Exit:  (8) disciple(aristotle, plato) ?  creep
   Call:  (8) disciple(plato, socrates) ?  creep
   Call:  (9) teacher(socrates, plato) ?  creep
   Exit:  (9) teacher(socrates, plato) ?  creep
   Exit:  (8) disciple(plato, socrates) ?  creep
   Exit:  (7) follower(aristotle, socrates) ?  creep
true.
```

Sadržaj

1. Logičko programiranje i Prolog
2. Zaključivanje nad Hornovim klauzulama
3. Programiranje u Prologu
4. Nedeklarativni aspekti Prologa

Redoslijed atoma/klauzula kod rekurzije

```
follower(X, Y) :-   % osnovna klauzula
  disciple(X, Y).
follower(X, Y) :-   % rekurzivna klauzula
  disciple(X, Z),
  follower(Z, Y).
```

- Formalno, redoslijed klauzula i atoma unutar njih trebao bi biti proizvoljan, jer vrijedi komutativnost operatora '$\wedge$' i '$\vee$':

$$
\begin{aligned}
& \big(\neg D(x,y) \vee F(x,y)\big) \wedge \big(\neg D(x,z) \vee \neg F(z,y) \vee F(x,y)\big) \\
\equiv\ & \big(\neg D(x,y) \vee F(x,y)\big) \wedge \big(\neg F(z,y) \vee \neg D(x,z) \vee F(x,y)\big) \\
\equiv\ & \big(\neg D(x,z) \vee \neg F(z,y) \vee F(x,y)\big) \wedge \big(\neg D(x,y) \vee F(x,y)\big) \\
\equiv\ & \big(\neg F(z,y) \vee \neg D(x,z) \vee F(x,y)\big) \wedge \big(\neg D(x,y) \vee F(x,y)\big)
\end{aligned}
$$

Redoslijed atoma/klauzula kod rekurzije

- Međutim, zbog toga što Prolog koristi SLD, ove komutativnosti ne vrijede i redoslijed klauzula i atoma postaje bitan

1:
```
follower(X, Y) :-
  disciple(X, Y).
follower(X, Y) :-
  disciple(X, Z),
  follower(Z, Y).
```

2:
```
follower(X, Y) :-
  disciple(X, Z),
  follower(Z, Y).
follower(X, Y) :-
  disciple(X, Y).
```

3:
```
follower(X, Y) :-
  disciple(X, Y).
follower(X, Y) :-
  follower(Z, Y),
  disciple(X, Z)..
```

4:
```
follower(X, Y) :-
  follower(Z, Y),
  disciple(X, Z).
follower(X, Y) :-
  disciple(X, Y).
```

$\Rightarrow$ **deklarativno značenje** odstupa od **proceduralnog značenja**!

Negacija

- Hornov oblik ne dopušta negaciju atoma u antecedentu, npr.:

$$\big(Q(x) \wedge \neg P(x)\big) \rightarrow R(x) \;\equiv\; \neg Q(x) \underbrace{\vee P(x) \vee R(x)}_{\text{dva pozitivna literala!}}$$

- Logičko programiranje bez negacije bilo bi suviše ograničeno
- Prolog uvodi operator `not`, koji se može koristiti u tijelu pravila:

```
R(X) :- Q(X), not(P(X)).
```

- Semantika operatora `not` je drugačija od one u logici:

Negacija kao neuspjeh (engl. *negation as failure*) – NAF

Literal `not(P(x))` je istinit ako se ne može dokazati `P(x)`, inače je lažan.

- U logici: `not(P(x))` je istinit akko `P(x)` je lažan

Negacija – primjer

```
human(X) :-
  speaks(X),
  not(has_feathers(X)).

speaks(socrates).
speaks(polynesia).
has_feathers(polynesia).
```

```
?- human(polynesia).
false
?- human(socrates).
true
?- not(human(polynesia)).
true
```

Pretpostavka zatvorenog svijeta

- NAF: ako se `P(x)` ne može dokazati, onda je `not(P(x))` istinit
- Standardna semantika: ako je `not(P(x))` istinit, onda je `P(x)` lažan
- U kombinaciji: ako se `P(x)` ne može dokazati, onda je `P(x)` lažan
- Drugim riječima, sve što se ne može dokazati je lažno

Pretpostavka zatvorenog svijeta *(closed world assumption)* – CWA

Sve što se ne može dokazati (činjenice koje nisu u bazi znanja i koje se iz ne mogu izvesti) je lažno.

- Ne dopuštamo da nešto bude nepoznato (niti istinito niti lažno)
- To dovodi do odstupanja od standardne semantike. Npr., u logici:

$$P, (P \wedge \neg Q) \rightarrow R \not\vdash R$$

ali u Prologu:

$$P, (P \wedge \neg Q) \rightarrow R \vdash R$$

Sažetak

- **Logičko programiranje** vrsta je deklarativnog programiranja, a **Prolog** je logički programski jezik
- **Hornova klauzalna logika** je podskup FOL-a koji omogućava učinkovito zaključivanje rezolucijom
- **Hornove klauzule** su klauzule s najviše jednim pozitivnim literalom, a **definitne klauzule** su klauzule s točno jednim pozitivnim literalom
- Program u Prologu je niz definitnih klauzula prvog reda, koje odgovaraju **činjenicama** i **pravilima**
- Izvođenje programa provodi primjenom **rezolucijom opovrgavanjem s ulančavanjem unazad** (SLD rezolucija)
- Nekomutativnost disjunkcije/konjunkcije i **negacija kao neuspjeh** nedeklarativni su aspekti Prologa

Sljedeća tema: Ekspertni sustavi