Povratak u Prolog (matematička logika, Prolog) verzija 2

Zlatko Sirotić, univ.spec.inf.

ISTRA TECH d.o.o.

Pula, svibanj 2019.

Uvod

- Primjena "umjetne inteligencije" (engl. skraćenica AI) u informatici ima dugu povijest, još od 50-ih godina 20. stoljeća.
- **Lisp** (LISt Processor) je **funkcijski programski jezik**, koji se intenzivno koristi(o) u AI (uglavnom u SAD), a specificiran je 1958. Od jezika koji se i danas intenzivno koriste, samo Fortran je stariji (1954.-57.).
- Izvan SAD-a se u AI području uglavnom koristi **logički jezik Prolog** (PROgrammming in LOGic), specificiran 1970.
- Temelj za Prolog je (matematička) **logika prvog reda** (first-order logic; **logika sudova** je njen podskup), iako podržava i neke predikate drugog reda (npr. setof i bugof). Visoko je deklarativan (u tom smislu usporediv sa SQL-om).
- IBM-ov AI računalni sustav (hardver i softver) **Watson** pobijedio je 2011. godine ljudske prvake u kvizu "Jeopardy!". Softver je pisan većinom u jezicima C++, Java i Prolog.

Teme

- Matematička logika
- Veza matematičke logike i jezika Prolog
- Prolog primjeri (jednostavni)
- Prolog program za (simboličko) deriviranje

Nezavisan razvoj aksiomatike i logike – do Fregea

AKSIOMATIKA

Euklid (4. st. Pr.Kr.)

V

V

Dedekind (1888.)

(tzv. Peanova aritmetika)

LOGIKA

Aristotel (4. st. Pr.Kr.)

V

V

Boole (1847.)

(matematika logike)

G.Frege (1879.)

(logika matematike)

kreirao logiku 1.reda (osnova za Prolog)

Odnos između (semantičke) valjanosti i (sintaktičke) dokazivosti (izvodljivosti) kod logičkih sustava

- Ako se u logičkom sustavu na temelju valjanosti (svake) tvrdnje može pokazati njena dokazivost (izvodljivost iz aksioma), onda je takav sustav (**semantički) potpun**:

  Ako iz $\models$ **A** (A je valjana) slijedi $\vdash$ **A** (A je dokaziva), logički sustav je **(semantički) potpun**.
  Napomena: operatori $\models$ i $\vdash$ su **metalogički**, a ne logički.

- Ako se u logičkom sustavu mogu dokazati samo valjane tvrdnje, onda je takav sustav **pouzdan** (korektan):

  Ako iz $\vdash$ **A** (A je dokaziva) slijedi $\models$ **A** (A je valjana), logički sustav je **pouzdan**.

Logika sudova (Propositional Logic)

- **Operatori (veznici)** logike sudova (**račun/algebra sudova**):

negacija (negation) ¬

disjunkcija (disjunction) ∨ konjunkcija (conjunction) ∧

implikacija (implication) → ekvivalencija (equivalence) ↔

ekskluzivno ili (exclusive or) ⊕ nili (nor) ↓ ni (nand) ↑

- Primjer formule logike sudova:
  (p → q) ↔ (¬q → ¬p)
- Napomena: mala slova (npr. p, q) označavaju **atomarne (atomic) sudove** ili atome.
- Jedna interpretacija (u prirodnom jeziku):
  ako **pada kiša** (p), tada **trava je mokra** (q)
  ekvivalentno je
  ako **trava nije mokra** (¬q), tada **ne pada kiša** (¬p).

Logika sudova (Propositional Logic)

- Prikaz nekih logički ekvivalentnih formula.
  Operator ≡ je **metalogički** operator, tj. predstavlja metalogičku ekvivalenciju, dok je ↔ logički operator:

A ↔ B ≡ (A → B) ∧ (B → A)

A ⊕ B ≡ ¬(A → B) ∨ ¬(B → A)

A → B ≡ ¬A ∨ B

A → B ≡ ¬(A ∧ ¬B)

A ∨ B ≡ ¬(¬A ∧ ¬B)

A ∧ B ≡ ¬(¬A ∨ ¬B)

A ∨ B ≡ ¬A → B

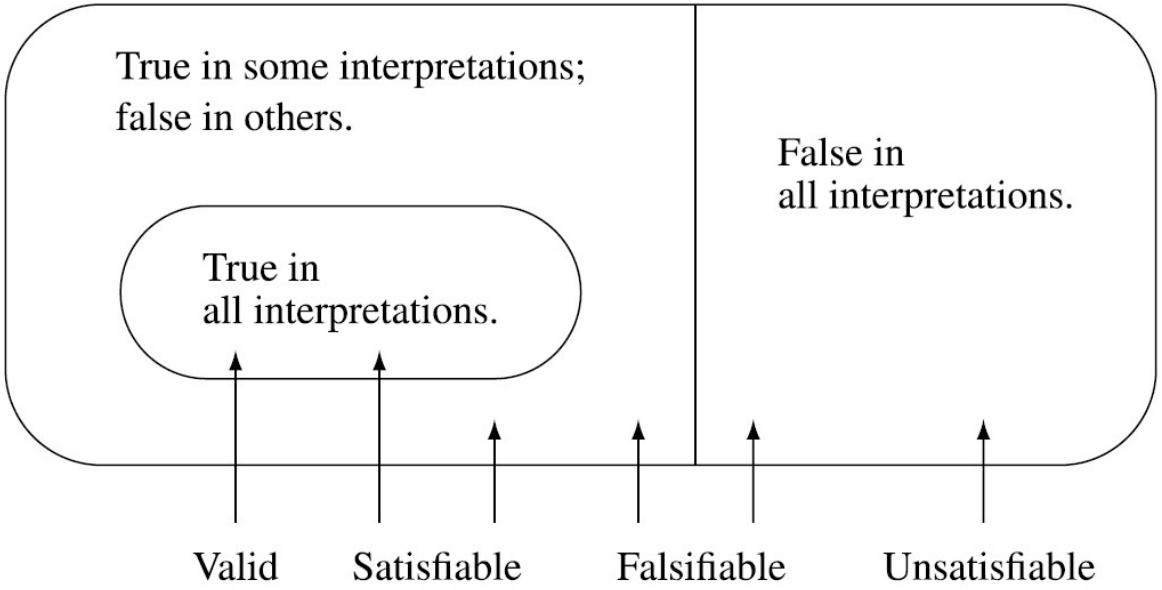
A ∧ B ≡ ¬(A → ¬B)

- Napomena: velika slova (npr. A, B) predstavljaju **logičke varijable**, koje označavaju bilo koju logičku formulu.
- Iz prethodnog se vidi da nisu nužni svi operatori.
  Dovoljna su dva: ¬ i jedan od ∨ ili ∧.
  Pa čak i samo jedan, ↓ ili ↑.

Logika sudova (Propositional Logic)

- Sa **semantičkog** stajališta, formula može biti:

- **zadovoljiva** (satisfiable): istinita u barem jednoj interpretaciji
- **valjana** (valid): istinita u svim interpretacijama (**tautologija**); označava se s ⊨ **A**
- **nezadovoljiva** (unsatisfiable): lažna u svim interpretacijama (što znači da je njena negacija valjana!)
- **oboriva** (falsifiable): lažna u barem jednoj interpretaciji.



Logika sudova (Propositional Logic)

- Logika sudova (za razliku od složenije logike prvog reda) ima **proceduru odlučivanja** (engl. decision procedure), tj. algoritam koji u konačnom vremenu vraća odgovor da li je formula valjana (ili da li je zadovoljiva).
- Jedna procedura odlučivanja je izrada **tablice istinitosti (semantička tablica)**.
- Sljedeća tablica istinitosti pokazuje da je (prije prikazana) formula logike sudova $(p \rightarrow q) \leftrightarrow (\neg q \rightarrow \neg p)$ valjana:

p	q	$(p$	$\rightarrow$	$q)$	$\leftrightarrow$	$(\neg$	q	$\rightarrow$	$\neg$	$p)$
T	T	T	T	T	T	F	T	T	F	T
T	F	T	F	F	T	T	F	F	F	T
F	T	F	T	T	T	F	T	T	T	F
F	F	F	T	F	T	T	F	T	T	F

Logika prvog reda
(First-Order Logic)

- Logika prvog reda (**logika predikata, ili račun predikata**) je nadskup logike sudova, i dodaje joj sljedeće:
  - **predikate**: n-arni predikat se interpretira kao n-arna relacija nad određenom domenom
  - **varijable i konstante** (iz određene domene), koje čine argumente predikata
  - **univerzalni kvantifikator** (universal quantifier) ∀ (čita se "za svaki")
  - **egzistencijalni kvantifikator** (existential quantifier) ∃ (čita se "postoji").
- **LPR s funkcijama** dodaje još **funkcije**, tako da argumenti u predikatima mogu biti i funkcije.
- Kod LPR, **kvantifikatori se primjenjuju samo na varijable**, a ne npr. na predikate (to je logika drugog reda).

Logika prvog reda
(First-Order Logic)

- Primjer jednostavne formule (ovaj primjer nema funkcije):
  ∀x p(a, x) čita se "za svaki x vrijedi p(a, x)"
  (x je varijabla, p je dvomjesni predikat, a je konstanta).
- Jedna interpretacija, kod koje je formula istinita:
  I1 = (N , {≤}, {0}) - domena je N, tj. skup prirodnih brojeva;
  predikat *p* se zamjenjuje relacijom ≤, konstanta *a* s nulom,
  pa za svaki x element od N **vrijedi** da 0 ≤ x.
- Jedna interpretacija, kod koje formula nije istinita:
  I1 = (Z , {≤}, {0}) - domena je Z, tj. skup cijelih brojeva
  (negativni cijeli brojevi, nula i pozitivni cijeli brojevi);
  predikat *p* se zamjenjuje relacijom ≤, konstanta *a* s nulom,
  pa **ne vrijedi** za svaki x element od Z da 0 ≤ x
  (ne vrijedi za negativne brojeve).
- **LPR nema proceduru odlučivanja, ali je poluodlučiva**
  (procedura odlučivanja može ne završiti).

Preneksna konjunktivna normalna forma (PCNF)

- Formula u preneksnoj konjunktivnoj normalnoj formi (prenex conjunctive normal form, PCNF) ima oblik:
  $Q_1\, x_1 \ldots Q_n\, x_n\, M$
  gdje su Q_i kvantifikatori (∀ ili ∃),
  a M je matrica bez kvantifikatora,
  i M se nalazi u CNF formi,
  tj. M ima oblik konjunkcije disjunkcija, npr. (A ∨ B) ∧ (C ∨ D).

- Primjer formule u PCNF:

```
∀y ∀z (   [p(f(y)) ∨ ¬p(g(z)) ∨ q(z)]
        ∧ [¬q(z)   ∨ ¬p(g(z)) ∨ q(y)])
```

- **Svaka formula logike prvog reda može se pretvoriti u logički ekvivalentnu formulu u PCNF obliku.**

Skolemizacija (po logičaru Skolemu)

- Skolemizacija je preoblikovanje formule iz PCNF oblika **u oblik bez egzistencijalnih kvantifikatora** (uvođenjem tzv. Skolemovih funkcija).
- **Skolemizacijom se ne dobiva logički ekvivalentna formula, ali su polazna i Skolemizirana formula isto zadovoljive.**
- Npr. formula koja je u PCNF obliku:
  $\exists x \exists y \forall z((p(x) \vee \neg p(y) \vee q(z)) \wedge (\neg q(x) \vee \neg p(y) \vee q(z)))$
  Skolemizira se u oblik:
  $\forall z((p(a) \vee \neg p(b) \vee q(z)) \wedge (\neg q(a) \vee \neg p(b) \vee q(z)))$
  gdje su a i b Skolemove funkcije (u konkretnom slučaju su to konstante), koje (Skolemove funkcije) odgovaraju egzistencijalno kvantificiranim varijablama x i y.
- Dodatno se može eliminirati univerzalni kvantifikator i matrica napisati u (logički ekvivalentnom) **klauzalnom obliku**:
  $\{ \{p(a), \neg p(b), q(z)\}, \{\neg q(a), \neg p(b), q(z)\} \}$

Rezolucija

- Algoritam (proceduru, metodu) rezolucije (resolution) kreirao je John Alan **Robinson, 1965.**
  (na temelju onoga što su napravili Davis i Putnam, 1960.).
- Algoritam rezolucije je **u logici sudova** pouzdan (sound) i kompletan (complete), te je on i procedura odlučivanja za nezadovoljivost formule (u klauzalnom obliku).
- **U logici prvog reda, algoritam rezolucije je i dalje pouzdan i potpun**, **ali nije procedura odlučivanja (nego poluodlučivanja), jer algoritam može ne završiti.**
- **Algoritam opće rezolucije** (general resolution) se temelji na dva "podalgoritma":
  - **temeljnoj rezoluciji** (ground resolution), koja se primjenjuje nad **temeljnim klauzulama** logike prvog reda, kao da su to sudovi logike sudova
  - **unifikaciji** (unification).

Rezolucija

- Kod **temeljne rezolucije** se iz dvije temeljne klauzule **eliminiraju (međusobno) suprotne klauzule** (clashing clauses) i dobiva se jedna temeljna klauzula – **rezolventa**.
  Primjer:
  {q(f(b)), r(a, f(b))} i
  {p(a), ¬q(f(b)), r(f(a), b)} daje
  {r(a, f(b)), p(a), r(f(a), b)}
- **Unifikacijom** se "skoro suprotne klauzule" pokušavaju učiniti ("pravim") suprotnim klauzulama, primjenom **supstitucije.**
  Primjer: **p(f(x), g(y))** i **¬p(f(f(a)), g(z))**
  nakon primjene **supstitucije {x ← f(a), y ← z}**
  postaju ("prave") suprotne klauzule
  p(f(f(a)), g(z))
  ¬p(f(f(a)), g(z))
  koje se mogu međusobno poništiti.

Rezolucija

- Još jedan primjer opće rezolucije.
- Dvije temeljne klauzule:
  {p(f(x), g(y)), q(x, y)} i
  {¬p(f(f(a)), g(z)), q(f(a), z)}
  imaju "skoro suprotne klauzule (označene crvenom bojom).

- One se supstitucijom **{x ← f(a), y ← z}**
  pretvaraju u ("prave") suprotne klauzule, pa se nad temeljnim klauzulama može primijeniti temeljna rezolucija.

- Time se dobije temeljna klauzula-rezolventa:
  {q(f(a), z), q(f(a), z)}
  ili jednostavnije
  {q(f(a), z)}

Logičko programiranje

- Rezolucija je izvorno kreirana kao metoda (algoritam) za automatsko dokazivanje teorema (automatic theorem proving).
- Kasnije se otkrilo da reducirana varijanta rezolucije (npr. SLD rezolucija) može poslužiti za programiranje. Taj pristup zove se **logičko programiranje**.
- Program se izražava kao skup logičkih klauzula. Upit je klauzula koja se dodaje tom skupu.
- **Tehnički gledano, upit predstavlja negaciju tvrdnje koju želimo dokazati. Ako upit ne uspije, ta tvrdnja je dokazana (dokazivanje opovrgavanjem).**
- **"Usput" se dobiju (na temelju unifikacije) i rezultati programa - to čini (praktičnu) razliku između dokazivanja teorema i logičkog programiranja.**

Hornove klauzule (Horn clauses)

- Hornove klauzule su posebna vrsta logičkih klauzula, oblika:
  $A \leftarrow B_1, \ldots, B_n \equiv A \leftarrow B_1 \wedge \ldots \wedge B_n \equiv A \vee \neg B_1 \vee \ldots \vee \neg B_n$
  tj. imaju **najviše jedan pozitivan literal** (ovdje je to A).
- **Definitne klauzule** imaju **točno jedan** pozitivan literal.
- Pozitivni literal A je **glava** (head),
  a negativni literali B_i su **tijelo** (body) klauzule.
- Koristi se i sljedeća terminologija:
  - **činjenica** je Hornova klauzula bez tijela: $A \equiv A \leftarrow True$
  - **cilj ili upit** je klauzula bez glave: $\leftarrow B_1, \ldots, B_n$
  - **pravilo zaključivanja** ima oboje: $A \leftarrow B_1, \ldots, B_n$
- Kako se vidi, kod logičkog programiranja preferira se
  korištenje **operatora reverzne implikacije** $\leftarrow$,
  umjesto operatora (uobičajene) implikacije $\rightarrow$.
- Operator $\leftarrow$ u Hornovoj klauzuli $A \leftarrow B_1, \ldots, B_n$
  prirodno se čita: **da bi dokazao A, dokaži B_1 i ... i B_n**.

SLD rezolucija

- **Selective Linear Definite clause resolution** (linearna rezolucija za jezik definitnih klauzula s funkcijom izbora) je osnovna metoda zaključivanja u logičkom programiranju.
- Kao i opća metoda rezolucije, i ona je pouzdana (sound) i kompletna za pobijanje (refutation complete), **ali samo ako se primjenjuje na Hornove klauzule**.
- SLD rezoluciju čini sekvenca koraka rezolucije između klauzule-cilja (upita) i programske klauzule.
- SLD rezoluciju određuju pravilo za biranje literala u upitu - **pravilo računanja** (computation rule), i pravilo za biranje određene programske klazule - **pravilo pretraživanja** (search rule).
- **SLD rezolucija je jako ovisna o odabranim pravilima računanja i pretraživanja**. Čak i kada postoje jedan ili više korektnih odgovora, SLD rezolucija može ne završiti, ili završiti bez nalaženja odgovora (ovisno o tim pravilima).

Primjer logičkog programa u obliku Hornovih klauzula i SLD rezolucije

```
 1. ancestor(x, y) ← parent(x, y)
 2. ancestor(x, y) ← parent(x, z), ancestor(z, y)

 3. parent(bob, allen)
 4. parent(catherine, allen)
 5. parent(dave, bob)
 6. parent(ellen, bob)
 7. parent(fred, bob)
 8. parent(harry, george)
 9. parent(ida, harry)
10. parent(joe, harry)
```

Primjer logičkog programa u obliku Hornovih klauzula i SLD rezolucije

- Pogledajmo primjer zaključivanja nad prethodnim programom (pod rednim brojem 11. je polazni cilj):

```
11. ← ancestor(y, bob), ancestor(bob, z)
12. ← parent(y, bob), ancestor(bob, z)      1,11 {y←dave}
13. ← ancestor(bob, z)                      5,12
14. ← parent(bob, z)                        1,13 {z←allen}
15. <prazna klauzula>                       3,14
```

- **Nađen je jedan točan odgovor (od više): y=dave i z=allen.**
- U ovom slučaju su primijenjena ova **pravila**:
  - **računanja**: uvjeti u upitu biraju se **s lijeva na desno**
  - **pretraživanja**: klauzule se pretražuju **od vrha prema dnu**.
- Da su primijenjena druga pravila, moglo se desiti da **program nikad ne završi, ili ne nađe točan odgovor.**

Prolog program (ekvivalentan prethodnom)

- **Program** je skup klauzula oblika **glava :- tijelo.**
  Koristi se **:-** umjesto ← i obavezna je točka na kraju.
- **Klauzula-pravilo** (rule clause) ima oba dijela,
  klauzula-činjenica (fact) ima samo glavu,
  klauzula-cilj ili upit (goal) ima samo tijelo.
- **Varijable** se u Prologu pišu velikim početnim slovima.
- **Proceduru** čine klauzule s istom glavom (ovdje su dvije):

```
ancestor(X, Y) :- parent(X, Y). % X je predak od Y
ancestor(X, Y) :- parent(X, Z), ancestor(Z, Y).
parent(bob, allen). % bob je allenov roditelj
parent(catherine, allen).
parent(dave, bob).
parent(ellen, bob).
parent(fred, bob).
parent(harry, george).
parent(ida, harry).
parent(joe, harry).
```

Prolog program (ekvivalentan prethodnom logičkom programu)

- Primjeri postavljanja upita (klauzula-ciljeva):

```
?- parent(bob, allen). % da li je bob alenov roditelj?
true.
?- ancestor(allen, bob). % da li je allen bobov predak?
false.

?- parent(X, allen). % tko je allenov roditelj?
X = bob ; % znak ; mi kucamo - tražimo sljedeći odgovor
X = catherine. % točka - sustav zna da više nema odgovora

?- ancestor(ellen, X). % Kome je ellen predak?
X = bob ;
X = allen ; % sustav još ne zna da nema više odgovora
false.
```

Način rada Prolog programa

- Prolog sustav koristi SLD rezoluciju (SLDNF varijantu).
- **Pravilo računanja** (computation rule):
  uvjeti u upitu biraju se s lijeva na desno.
- **Pravilo pretraživanja** (search rule):
  klauzule se pretražuju od vrha prema dnu procedure.
- Primjer: upit ?- ancestor(Y, bob), ancestor(bob, Z).
  će uspjeti i vratiti supstituciju Y = dave, Z = allen.
- Prolog sustav će to raditi ovako:

```
:- ancestor(Y, bob), ancestor(bob, Z).
:- parent(Y, bob), ancestor(bob, Z).{Y <- dave}
:- ancestor(bob, Z).
:- parent(bob, Z). {Z <-allen}
:-
```

Prolog program može ne završiti (zapravo, završi sa: out of local stack)

- Pravilo računanja s lijeva na desno (i to naročito kad imamo lijevu rekurziju) i pravilo pretraživanja od vrha prema dnu, mogu uzrokovati nezavršavanje Prolog programa - kada ga prikažemo kao stablo, program tada ima beskonačne grane. To je svojstveno bilo kojoj SLD rezoluciji s takvim pravilima.
- Prolog sustav radi tako da pretražuje prvo u dubinu (depth-first), a ne prvo u širinu (breadth first), pa neće naći rješenje koje se nalazi desno od beskonačne grane (na stablu).
- Dodatno, Prolog program može neuspješno završiti zato što sustav (zbog performansi) izostavlja jednu od kontrola algoritma za SLD rezoluciju, kontrolu javljanja (occurs-check).
- Zato su Prolog programi vrlo osjetljivi na redoslijed klauzula i redoslijed uvjeta (unutar klauzule-pravila ili klauzule-cilja; klauzule-činjenice nisu problematične, jer nemaju uvjeta).

Prolog primjenjuje SLDNF varijantu SLD rezolucije

- Prolog primjenjuje **SLDNF** rezoluciju (Negation as Failure), kod koje se **negacija shvaća kao konačan neuspjeh zadovoljavanja cilja**.
- Negacija se označava pomoću operatora **not**.
- NaF **ponekad može dati nelogične odgovore**:

```
lijepo_pjeva(X) :- not(kresti(X)), ptica(X).
ptica(svraka).
ptica(slavuj).
kresti(svraka).
```

- Logičan odgovor na pitanje da li slavuj lijepo pjeva:

```
?- lijepo_pjeva(slavuj).
true.
```

- Nelogičan odgovor na pitanje da li neka ptica lijepo pjeva:

```
?- lijepo_pjeva(X).
false.
```

Prolog primjenjuje SLDNF varijantu SLD rezolucije

- Prolog radi na temelju tzv. **pretpostavke zatvorenog svijeta** (closed world assumption – CWA) :
  ako se nešto ne može izvesti iz baze znanja, to je lažno.
- Sljedeći primjer, u kojem je zamijenjen redoslijed uvjeta u klauzuli lijepo_pjeva, daje logične odgovore:

```
lijepo_pjeva(X) :- ptica(X), not(kresti(X)).
ptica(svraka).
ptica(slavuj).
kresti(svraka).
```

- Logičan odgovor na pitanje da li slavuj lijepo pjeva:

```
?- lijepo_pjeva(slavuj).
true.
```

- Logičan odgovor na pitanje da li neka ptica lijepo pjeva:

```
?- lijepo_pjeva(X).
X = slavuj.
```

Prolog program
za (simboličko) deriviranje

```
:- op(100, xfy, [^]).
% glavna procedura
deriviraj(F, X, R) :-
    d(F, X, R1),
    sredi(R1, R).

% ! označava rez: reže klauzule (iste procedure)
% ispod one u kojoj se nalazi, i utječe da se
% ciljevi lijevo od njega zadovolje samo jednom

% derivacija konstante
d(Const, X, 0) :- atomic(Const), !.

% derivacija varijable (X)
d(F, X, 1) :- F == X, !.
```

Prolog program
za (simboličko) deriviranje

```
d(ln(F), X, R) :-
   d(F, X, R1),
   R = R1 / F, !.

d(sin(F), X, R) :-
   d(F, X, R1),
   R = cos(F) * R1, !.

d(cos(F), X, R) :-
   d(F, X, R1),
   R = -sin(F) * R1, !.

d(tan(F), X, R) :-
   R1 = sin(F) / cos(F),
   d(R1, X, R), !.
```

Prolog program
za (simboličko) deriviranje

```
d(asin(F), X, R) :-
   d(F, X, R1),
   R = 1 / (1 - F ^ 2) ^ 0.5 * R1, !.

d(acos(F), X, R) :-
   d(F, X, R1),
   R = -1 / (1 - F ^ 2) ^ 0.5 * R1, !.

d(atan(F), X, R) :-
   d(F, X, R1),
   R = 1 / (1 + F ^ 2) * R1, !.
```

Prolog program
za (simboličko) deriviranje

```
d(-F, X, R) :-
   d(F, X, R1),
   R = -R1, !.

d(F1 + F2, X, R) :-
   d(F1, X, R1),
   d(F2, X, R2),
   R = R1 + R2, !.

d(F1 - F2, X, R) :-
   d(F1, X, R1),
   d(F2, X, R2),
   R = R1 - R2, !.
```

Prolog program
za (simboličko) deriviranje

```
d(F1 * F2, X, R) :-
   d(F1, X, R1),
   d(F2, X, R2),
   R = R1 * F2 + F1 * R2, !.

d(F1 / F2, X, R) :-
   d(F1, X, R1),
   d(F2, X, R2),
   R = (R1 * F2 - F1 * R2) / F2 ^ 2, !.
```

Prolog program
za (simboličko) deriviranje

```
% derivacija potencije
d(F ^ S, X, R) :-
   atomic(S),
   d(F, X, R1),
   R = S * F ^ (S - 1) * R1, !.

% logaritamsko deriviranje
d(F1 ^ F2, X, R) :-
   d(F1, X, R1),
   d(F2, X, R2),
   R = F1 ^ F2 * (F2 / F1 * R1 + R2 * ln(F1)), !.
```

Prolog program
za (simboličko) deriviranje

- Pokazuje se da je sređivanje deriviranog izraza skoro kompleksnije nego samo deriviranje.
- Sljedeći programski kod uspijeva srediti neke izraze, ali moguće je da dođe ne samo do "ružnog", već i do netočnog "sređivanja".

```
sredi(A, R) :- var(A), R = A, !.

sredi(A, R) :- atomic(A), R = A, !.

sredi(ln(A), R) :-
    sredi(A, R1),
    R = ln(R1), !.
```

Prolog program
za (simboličko) deriviranje

```
sredi(sin(A), R) :-
    sredi(A, R1),
    R = sin(R1), !.

sredi(cos(A), R) :-
    sredi(A, R1),
    R = cos(R1), !.

sredi(tan(A), R) :-
    sredi(A, R1),
    R = tan(R1), !.

sredi(-A, R) :- sredi(A, R1),
    (R1 == 0, R = 0, !;
     R = (-R1), !).
```

Prolog program
za (simboličko) deriviranje

```
sredi(A + B, R) :- sredi(A, R1), sredi(B, R2),
    (number(R1), number(R2), R is R1 + R2, !;
     R1 ==  0, R = R2, !;
     R2 ==  0, R = R1, !;
     R1 == R2, R = 2 * R1, !;
     R = R1 + R2, !).

sredi(A - B, R) :- sredi(A, R1), sredi(B, R2),
    (number(R1), number(R2), R is R1 - R2, !;
     R1 ==  0, R = (-R2), !;
     R2 ==  0, R =  R1, !;
     R1 == R2, R is 0, !;
     R = R1 - R2, !).
```

Prolog program
za (simboličko) deriviranje

```
sredi(A * B, R) :-
   A == B,
   sredi(A, R1),
   R = R1 ^ 2, !.

sredi(A * B, R) :- sredi(A, R1), sredi(B, R2),
   (number(R1), number(R2), R is R1 * R2, !;
    (R1 == 0; R2 == 0), R is 0, !;
    R1 ==  1, R = R2, !;
    R2 ==  1, R = R1, !;
    R1 == R2, R = R1 ^ 2, !;
    R = R1 * R2, !).
```

Prolog program
za (simboličko) deriviranje

```
sredi(A / B, 1) :- A == B, !.
sredi(A / B, R) :- sredi(A, R1), sredi(B, R2),
    (number(R1), number(R2), R is R1 / R2, !;
     R1 ==  0, R is 0, !;
     R2 ==  1, R = R1, !;
     R1 == R2, R is 1, !;
     R = R1 / R2, !).

sredi(A ^ B, R) :- sredi(A, R1), sredi(B, R2),
    (R1 == 0, R is 0, !;
     R1 == 1, R is 1, !;
     R2 == 0, R is 1, !;
     R2 == 1, R = R1, !;
     number(R2), R2 < 0, R3 is -R2, R =  1 / R1 ^ R3, !;
     R = R1 ^ R2, !).
```

Zaključak

- Danas je "in" **multiparadigmatsko programiranje**, tj. istovremeno korištenje više programskih jezika, koji pripadaju različitim paradigmama (objektnoj, funkcijskoj, logičkoj ...), ili korištenje programskih jezika koji pokrivaju više paradigmi (npr. Scala). Najčešće se danas koristi mješavina objektnih i funkcijskih paradigmi.
- **No, i logička paradigma je (ponovno) sve zanimljivija.** Logičko programiranje je visoko deklarativno i nezamjenjivo za određene klase programskih problema.
- Podsjetili smo se na programski jezik Prolog.
- Prikazan je jedan malo složeniji primjer, koji je autor napravio prije više od 25 godina - program koji u manje od 100 redaka programskog koda "zna" derivirati funkcije jedne varijable.

Literatura (dio)

- Ben-Ari, M. (2012): Mathematical Logic for Computer Science (3. izdanje), Springer
- Čubrilo, M. (1989): Matematička logika za ekspertne sisteme, Informator, Zagreb
- Gopalakrishnan, G. (2006): Computation Engineering - Applied Automata Theory and Logic, Springer
- Nilsson, U., Maluszynski J. (2000): Logic, programming and Prolog (2. izdanje), John Wiley & Sons (the book may be downloaded … for personal use …)
- Radovan, M. (1987): Programiranje u PROLOGu, Informator, Zagreb
- SWI Prolog Reference Manual, version 7.2.0, May 2015, www.swi-prolog.org