# POVRATAK U PROLOG

Zlatko Sirotić, univ.spec.inf.
ISTRA TECH d.o.o.
Pula

# Neki noviji radovi

- CASE 2015: Višestruko nasljeđivanje - san ili Java 8?
- JavaCro 2015: Java paralelizacija

- HrOUG 2014: Nasljeđivanje je dobro, naročito višestruko - Eiffel, C++, Scala, Java 8
- CASE 2014: Trebaju li nam distribuirane baze u vrijeme oblaka?
- JavaCro 2014: Da li postoji samo jedna "ispravna" arhitektura web poslovnih aplikacija

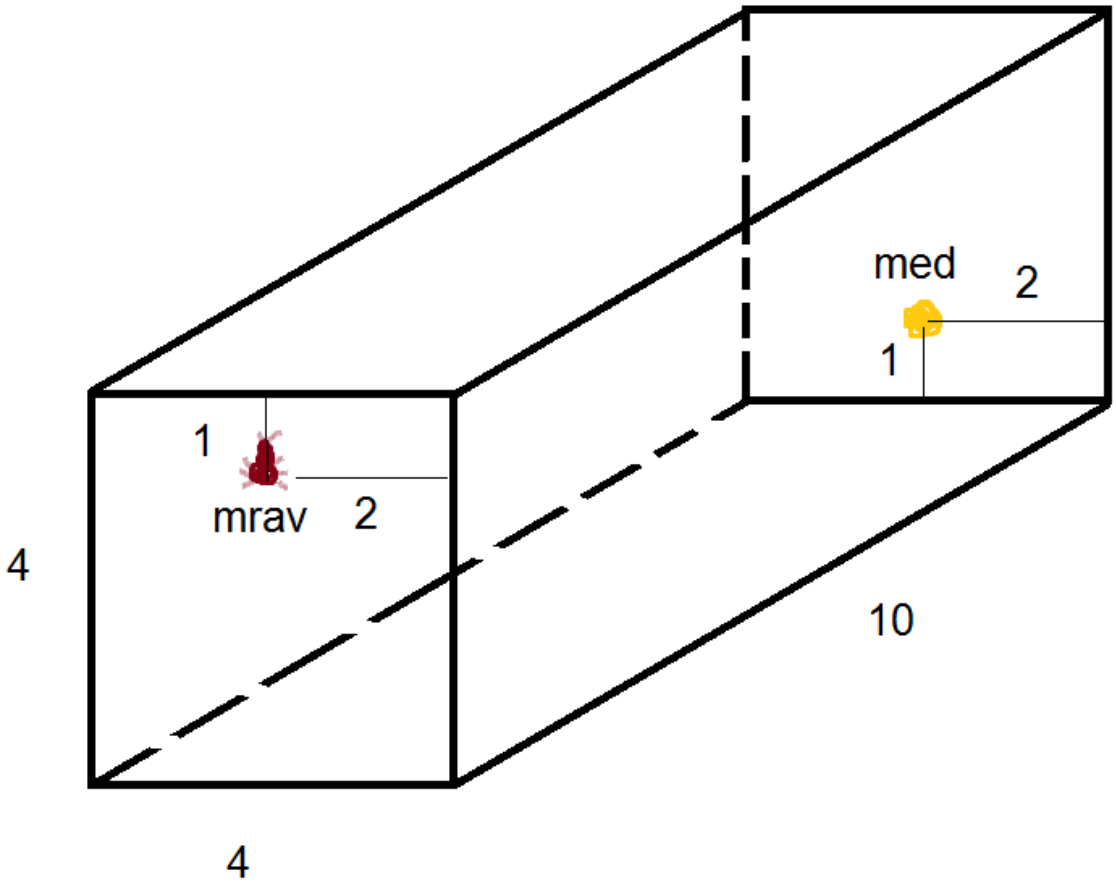
- HrOUG 2013: Transakcije i Oracle - baza, Forms, ADF
- CASE 2013: Što poslije Pascala? Pa … Scala!

# 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.

# Jedan zanimljivi zadatak – mrav i med na prizmi

- Da li postoji kraći put (po površini prizme) od 14 između mrava i kapljice meda?



# Složenija varijanta – mrav i med na valjku

- Prethodni zadatak mi je bio dao prijatelj Korado Korlević, prije 35 godina.
- Tada sam ga riješio (za par sati) - **kraći put postoji!**
- No, odmah sam sebi bio postavio pitanje:
  **"Što ako bi bio valjak, a ne prizma?"**
  (isto visine 10, promjera 4, sa istim pozicijama mrava i meda).
- Tada sam bio intuitivno uvjeren da kraći put ne postoji.
  A i mislio sam da za rješenje treba složena matematika, npr. račun varijacija.
- Vratio sam se tome prije dvije godine,
  u trenucima kada mi je dosadila informatika ☺.

# Teme

- Zanimljivi zadatak – postavka (već smo vidjeli)
- Matematička logika, aksiomatski sustavi
- Veza matematičke logike i jezika Prolog
- Prolog primjeri (jednostavni)
- Prolog program za (algebarsko) deriviranje
- Primjena Prolog programa za deriviranje na početni zadatak

# Nezavisan razvoj aksiomatike i logike – do Fregea

| AKSIOMATIKA | LOGIKA |
|---|---|
| Euklid (4. st. Pr.Kr.) | Aristotel (4. st. Pr.Kr.) |
| V | V |
| V | V |
| Dedekind (1888.) | Boole (1847.) |
| (tzv. Peanova aritmetika) | (matematika logike) |

G.Frege (1879.)
(logika matematike)
**kreirao logiku 1.reda (osnova za Prolog)**

# Euklidska geometrija

- Najvjerojatnije ju je Euklid sabrao i dopunio, ali je većina znanja stvorena prije.

- Neki kažu da je to vrhunsko djelo ljudskog genija i primjer kako treba izgledati aksiomatski sustav.

- Neki kažu da Euklidova geometrija i nije savršen primjer aksiomatskog sustava, te navode dva glavna razloga:
  - **postoje definicije koje se "vrte u krug"**
  - **postoje tvrdnje koje se implicitno pretpostavljaju**, nisu navedene kao aksiomi, niti se mogu izvesti iz njih (Euklidov aksiomatski sustav je upotpunjen krajem 19. stoljeća).

# Kako se gradi aksiomatski sustav (ne nužno formalni)

- Treba imati određeni broj **nedefiniranih termina**.
- Na temelju njih, prema utvrđenim **pravilima definiranja**, grade se **definirani termini**.
- Treba imati određeni broj **aksioma**.
- Na temelju njih, prema utvrđenim **pravilima zaključivanja**, izvode se **teoremi**.

# Obavezne ili (vrlo) poželjne osobine aksiomatskog sustava

- Mora biti **konzistentan**. Ako nije konzistentan, onda se iz njega može dokazati bilo koja tvrdnja (što znači – neka tvrdnja i njena negacija), **pa je takav sustav bezvrijedan**.

- Vrlo je poželjno da je **potpun**, tj. da se unutar njega može ili dokazati, ili opovrći bilo koja tvrdnja izrečena u terminima tog sustava. **Nepotpuni sustav nije bezvrijedan.**

- Vrlo poželjno je da je **odlučiv**, tj. da ima opću metodu kojom se može utvrditi (ne)valjanost neke tvrdnje izrečene u terminima tog sustava.

- Većinom je poželjno da je **neredundantan**, što znači da su aksiomi međusobno nezavisni.

# 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 potpun:

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

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

# Neeuklidske geometrije - 5. Euklidov aksiom

- Najvažniji pokretač formalizacije aksiomatskih sustava bilo je uvođenje tzv. **neeuklidskih geometrija**, koje su uglavnom nastale pobijanjem **5. Euklidovog aksioma o paralelama**.
- Aksiom kaže (ne u izvornoj Euklidovoj interpretaciji) da se kroz točku koja se nalazi izvan pravca (u istoj ravnini) može povući točno jedan paralelan pravac. **Dugo se smatralo da 5. aksiom zavisi od drugih**.
- U 19. stoljeću stvoreni su aksiomatski sustavi (**Gauss, Lobačevski, Bolay, Riemann**) u kojima je 5. aksiom zamijenjen aksiomom da se kroz točku ne može povući niti jedan paralelan pravac (**eliptičke geometrije**), ili da se može povući beskonačno puno pravaca (**hiperboličke geometrije**).
- U tim se geometrijama više ne može "prirodno dokazivati".

# Hilbertov program dokazivanja konzistentnosti aritmetike

- Hilbert je kao jedan od najvažnijih zadataka postavio: **dokazati konzistentnost i potpunost aksiomatskog sustava aritmetike pomoću formalnih metoda isključivo finitnim sredstvima**.
- *Finitno* znači da se upotrebljava **konačan broj strukturalnih svojstava izraza u formaliziranim računima i konačan broj koraka dokazivanja**.
- **Formalne metode** sprečavaju da se u postupku dokazivanja implicitno pretpostave neke tvrdnje koje nisu sadržane u aksiomima ili neke metode zaključivanja koje nisu u popisu definiranih metoda zaključivanja (bolje rečeno – baratanja formulama).
- Napomena: dokazati nezavisnost aksioma je (uglavnom) lakši problem - pomoću **modela**.

# Kurt Gödel i nepotpunost aritmetike

- 1929. je (sa 23 godine!) **dokazao potpunost predikatnog računa 1. reda**. To nije lako, za razliku od daleko lakšeg problema dokazivanja potpunosti (ili konzistentnosti) računa sudova.
- 1931. godine objavljen je njegov rad kojim je srušio nade Hilbertovog programa.
- Prvo, **dokazao je nepotpunost aksiomatskog sustava aritmetike** (ili bilo kog sustava sličnog njemu). Pritom se koristio **"prevođenjem" metamatematičkih tvrdnji u aritmetičke tvrdnje**.
- To znači da se ne može napraviti konačan broj (konzistentnih) aksioma iz kojih će se moći dokazati sve aritmetičke istine – **formalizacija ima granice!**

# Kurt Gödel i konzistentnost aritmetike

- Drugo – dokazao je **nemogućnost dokazivanja konzistentnosti sustava aritmetike unutar nje same** (finitnom metodom).
- Drugi dokaz **nije isključivao** mogućnost dokazivanja konzistentnosti aritmetike nefinitnim metodama.
- Gentzen (Hilbertov učenik) je 1936. **dokazao konzistentnost aritmetike nefinitnim metodama**.
- I sam Gödel je 1940./41. našao **konstruktivnu interpretaciju aritmetike** kojom je dokazao njenu konzistentnost, ali to opet nije po Hilbertovim finitnim principima.

# Usporedba osobina logike sudova, logike 1. reda i aritmetike

| AKSIOMATSKI SUSTAV | Konzistentan | Potpun | Odlučiv |
|---|---|---|---|
| Logika sudova | DA | DA<br>(Post, 1921.) | DA |
| Logika prvog reda | DA | DA<br>(Gödel, 1929.) | **NE**<br>(Church, Turing 1936.) |
| Aritmetika | DA<br>(Gentzen, 1936., nefinitnim metodama);<br><br>**NE može se dokazati unutar nje same**<br>(Gödel, 1931.) | **NE**<br>(Gödel, 1931.) | **NE**<br>(Church, Turing, 1936.)<br><br>= Churchov teorem<br><br>= Hilbertov Entscheidungs-problem (1928.) |

# Digresija – četiri znanstvena šoka u 20. stoljeću

- **Specijalna i opća teorija relativnosti.**

- **Kvantna mehanika.** Čak niti sam Einstein nije prihvatio kvantnu mehaniku (njegova izreka: **"Bog se ne kocka!**"). Opća teorija relativnosti i kvantna mehanika su međusobno kontradiktorne - to znači da najviše jedna od njih može biti istinita.

- **Gödelovi teoremi** o nepotpunosti i nemogućnosti finitnog dokazivanja konzistentnosti aritmetike.

- **Teorija determinističkog kaosa.** Nemjerljivo mala različita početna stanja dovode do značajno različitih ishoda **nakon nekog vremena** (za razliku od potpuno stohastičkih sustava).

# 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 je trava 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:

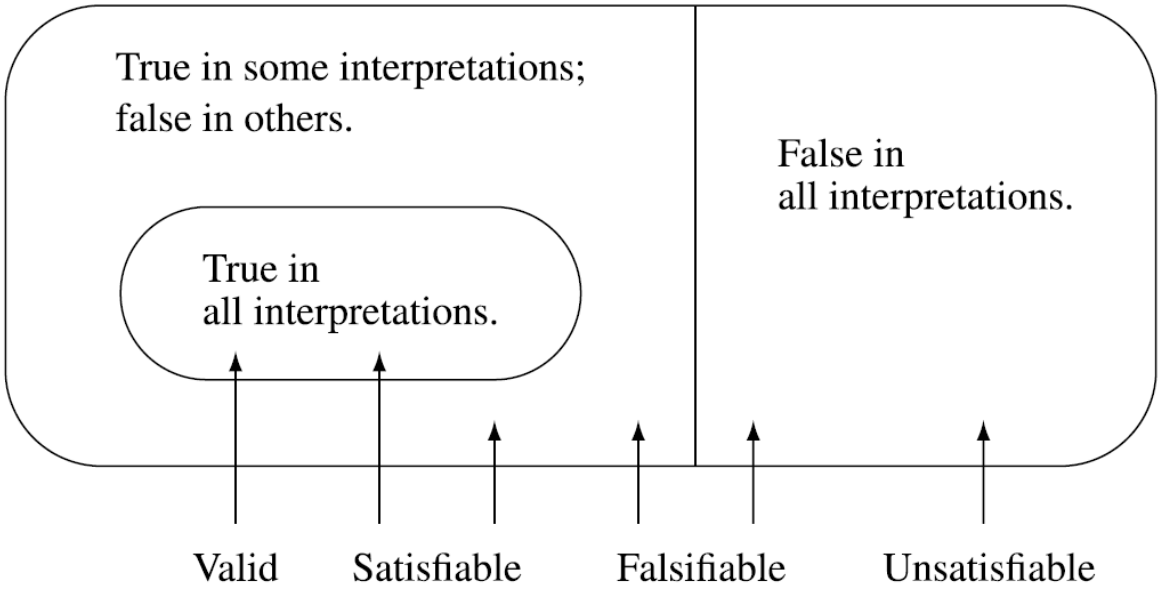
| | |
|---|---|
| **A↔B ≡ (A→B) ∧ (B→A)** | **A⊕B ≡ ¬(A→B)∨¬(B→A)** |
| **A→B ≡ ¬A∨B** | **A→B ≡ ¬(A∧¬B)** |
| **A ∨ B ≡ ¬(¬A∧¬B)** | **A∧ B ≡ ¬(¬A∨¬B)** |
| **A ∨ B ≡ ¬A→B** | **A∧ B ≡ ¬(A→¬B)** |

- 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 poluodlučljiva**
  (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), ¬p(b), q(z)}, {¬q(a), ¬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 **nažalost 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 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(y))** i
  **¬p(f(f(a)), g(y))**
  **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 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:
  $A \leftarrow B_1, ..., B_n \equiv A \vee \neg B_1, \ldots, \neg B_n$
  tj. imaju **najviše jedan pozitivan literal** (ovdje je to A).
- **Definitivne klauzule** imaju **točno jedan** pozitivan literal.
- Pozitivni literal A je **glava** (head), a negativni literali $B_i$ su **tijelo** (body) klauzule. Koristi se sljedeća terminologija:
  - **činjenica (fact)** je Horn klauzula bez tijela: $A \leftarrow$
  - **cilj ili upit (goal)** je klauzula bez glave: $\leftarrow B_1, \ldots, B_n$
  - **klauzula programa (program clause)** je Hornova klauzula s jednim pozitivnim i najmanje jednim negativnim literalom.
- Kako se vidi, kod logičkog programiranja preferira se korištenje ←, **operatora reverzne implikacije**, umjesto uobičajene implikacije →.
- Operator ← u $A \leftarrow B_1, \ldots, B_n$
  prirodno se čita: **da bi dokazao A, dokaži $B_1, \ldots, B_n$.**

# 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 određuje pravilo za biranje literala u klauzuli-cilju - **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 sada jedan 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
13. ← ancestor(bob, z)                        6,12,{y←ellen}
14. ← parent(bob, z)                          1,13
15. <prazna klauzula>                         3,14,{z←allen}
```

- **Nađen je jedan točan odgovor (od više): y=elen i z=allen.**
- U ovom slučaju je primijenjeno pravilo računanja da se uvjeti u upitu biraju se **s lijeva na desno**, i pravilo pretraživanja da se klauzule 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 logičkom programu)

- 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, a klauzula-cilj ili upit (goal) 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). %rekurzija
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), nad prethodnim programom:

```
?- 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 (određene) 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**.
- NF **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

- Za razliku od prethodnog primjera, 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 (algebarsko) deriviranje

```
:- op(100, xfy, [^]).
deriviraj(F, X, R) :- d(F, X, R1), sredi(R1, R).
%! označava rez – sprečava backtracking u proceduri.
%Reže klauzule (za proceduru) ispod, i uvjete desno od !.
d(Const, X, 0) :- atomic(Const), !. %derivacija konstante
d(F, X, 1) :-  F == X, !. %derivacija varijable (X)
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 (algebarsko) 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, !.
d(ln(F), X, R) :-
  d(F, X, R1),
  R = R1 / F, !.
d(-F, X, R) :-
  d(F, X, R1),
  R = -R1, !.
```

# Prolog program za (algebarsko) deriviranje

```
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, !.

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

# Prolog program za (algebarsko) deriviranje

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

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

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

# Prolog program za (algebarsko) deriviranje

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

# Prolog program za (algebarsko) deriviranje

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

# Prolog program za (algebarsko) deriviranje

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

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

sredi(A / B, 1) :- A == B, !.
```

# Prolog program za (algebarsko) deriviranje

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

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

# Mrav i med na valjku - rješenje

- Polazna ideja je da se **valjak prikaže u dvije dimenzije** (dva kruga i plašt), pa se po plaštu kotrlja gornji krug, a povezano s gornjim krugom kotrlja se i donji krug.

- Četiri točke:
  - med i mrav (zamišljeni kao točke)
  - i dva dirališta gornjeg i donjeg kruga s plaštom

  **moraju uvijek biti na istom pravcu.**

- Uz malo primjene srednjoškolske matematike, može se dobiti da kut donjeg kruga x2 ovisi o kutu gornjeg kruga x na ovaj način:

  **x2 = x - 5 * sin(x) / (2 + cos(x))**

# Mrav i med na valjku - rješenje

- Funkcija koja prikazuje udaljenost između mrava i meda (u ovisnosti o kutu gornjeg kruga x) je ovakva:

**udaljenost = sqr**

**((-sin(5 * sin(x) / (cos(x) + 2) - x) + 10 * sin(x) / (cos(x) + 2) + sin(x))^2 + (-cos(5 * sin(x) / (cos(x) + 2) - x) + cos(x) + 14)^2)**

ili (drugačije pisano)

**((-sin(5 * sin(x) / (cos(x) + 2) - x) + 10 * sin(x) / (cos(x) + 2) + sin(x))^2 + (-cos(5 * sin(x) / (cos(x) + 2) - x) + cos(x) + 14)^2)** ^(1/2)

- Minimalna udaljenost je približno **13,32,** za x od približno 2,64 radijana (približno 151°, x2 = 29°) i za x od približno 3,64 radijana (približno 209°).

# Mrav i med na valjku - rješenje

- Ovako izgleda ta funkcija – vide se dva simetrična minimuma (graph-plotter.cours-de-math.eu/):

**Functions:**

Watch out: $x^{-2}$ has to be written as **x^(-2)**.

**First Graph:** f(x) Derivative
sqr ((-sin(5 * sin(x) / (cos(x) + 2) - x) + 10 * sin(x) / ( Blue 1
From to Connect Show term

**Second Graph:** g(x) Derivative
Red 1
From to Connect Show term

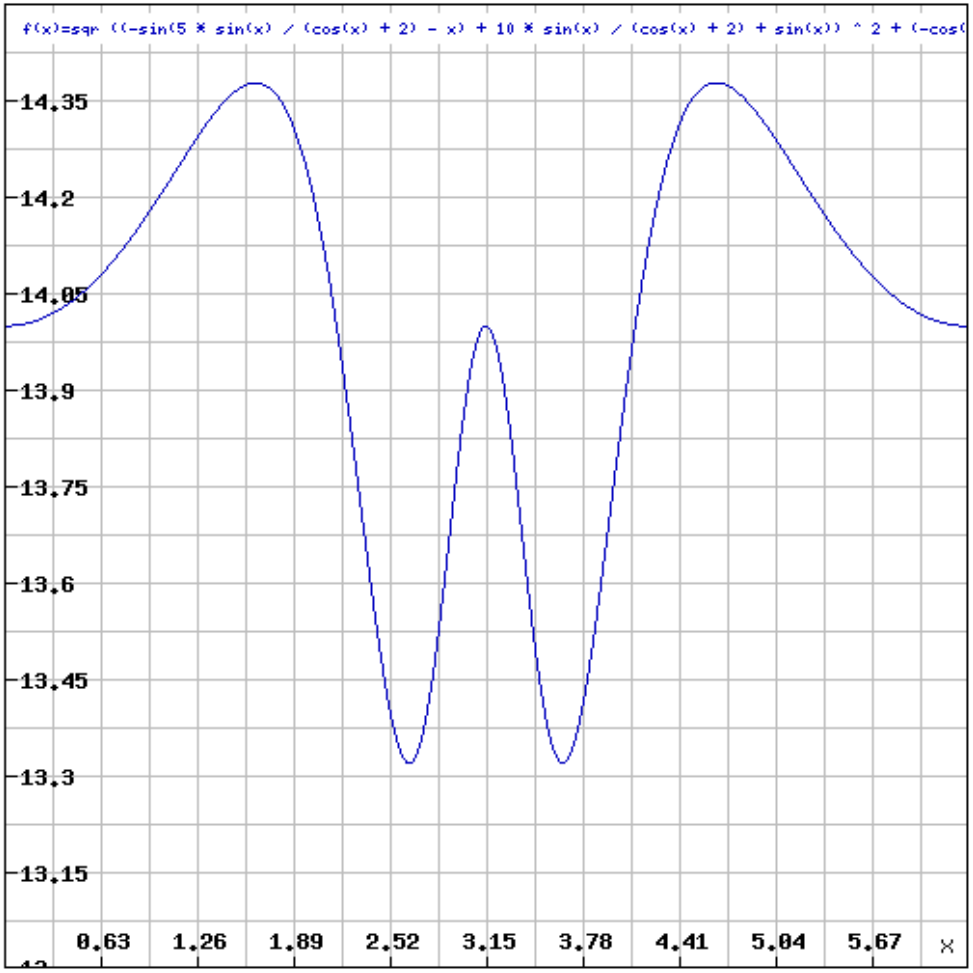
**Third Graph:** h(x) Derivative
Green 1
From to Connect Show term

Draw Reset Standard

**Display properties:** Image type: png

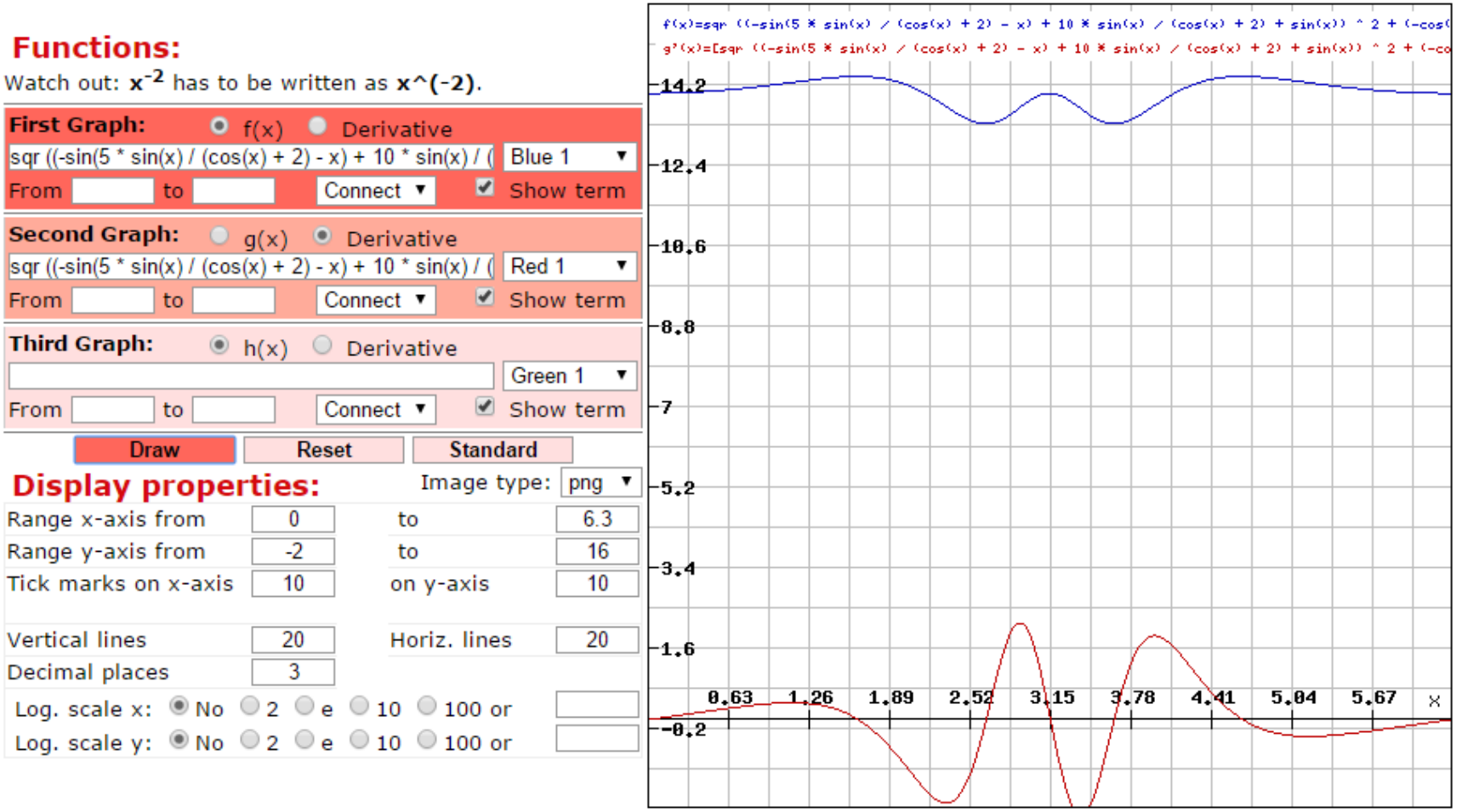
| | | | |
|---|---|---|---|
| Range x-axis from | 0 | to | 6.3 |
| Range y-axis from | 13 | to | 14.5 |
| Tick marks on x-axis | 10 | on y-axis | 10 |
| Vertical lines | 20 | Horiz. lines | 20 |
| Decimal places | 3 | | |

Log. scale x: No 2 e 10 100 or
Log. scale y: No 2 e 10 100 or



# Mrav i med na valjku - rješenje

- Ovako izgledaju funkcija i njena "numerička" derivacija (koja ima nule tamo gdje funkcija ima minimum ili maksimum):



# Mrav i med na valjku - rješenje

❖ Derivacija dobivena uz pomoć našeg Prolog programa:

deriviraj((-sin(5 * sin(X) / (cos(X) + 2) - X) + 10 * sin(X) / (cos(X) + 2) + sin(X))^2 + (-cos(5 * sin(X) / (cos(X) + 2) - X) + cos(X) + 14)^2)^(1/2), X, R).

((-sin(5*sin(x)/ (cos(x)+2)-x)+10*sin(x)/ (cos(x)+2)+sin(x))^2+ (-cos(5*sin(x)/ (cos(x)+2)-x)+cos(x)+14)^2)^0.5* (0.5/ ((-sin(5*sin(x)/ (cos(x)+2)-x)+10*sin(x)/ (cos(x)+2)+sin(x))^2+ (-cos(5*sin(x)/ (cos(x)+2)-x)+cos(x)+14)^2)* (2* (-sin(5*sin(x)/ (cos(x)+2)-x)+10*sin(x)/ (cos(x)+2)+sin(x))* (- (cos(5*sin(x)/ (cos(x)+2)-x)* ((5*cos(x)* (cos(x)+2)-5*sin(x)* -sin(x))/ (cos(x)+2)^2-1))+ (10*cos(x)* (cos(x)+2)-10*sin(x)* -sin(x))/ (cos(x)+2)^2+cos(x))+2* (-cos(5*sin(x)/ (cos(x)+2)-x)+cos(x)+14)* (- (-sin(5*sin(x)/ (cos(x)+2)-x)* ((5*cos(x)* (cos(x)+2)-5*sin(x)* -sin(x))/ (cos(x)+2)^2-1))+ -sin(x))))

# Mrav i med na valjku - rješenje

- Derivacija dobivena pomoću Prolog programa i "numerička" derivacija preklapaju se (nije matematički dokaz, ali …):

**Functions:**

Watch out: $x^{-2}$ has to be written as **x^(-2)**.

**First Graph:** f(x) Derivative
((-sin(5*sin(x)/ (cos(x)+2)-x)+10*sin(x)/ (cos(x)+2)+s | Blue 1
From to Connect Show term

**Second Graph:** g(x) Derivative
sqr ((-sin(5 * sin(x) / (cos(x) + 2) - x) + 10 * sin(x) / ( | Red 1
From to Connect Show term

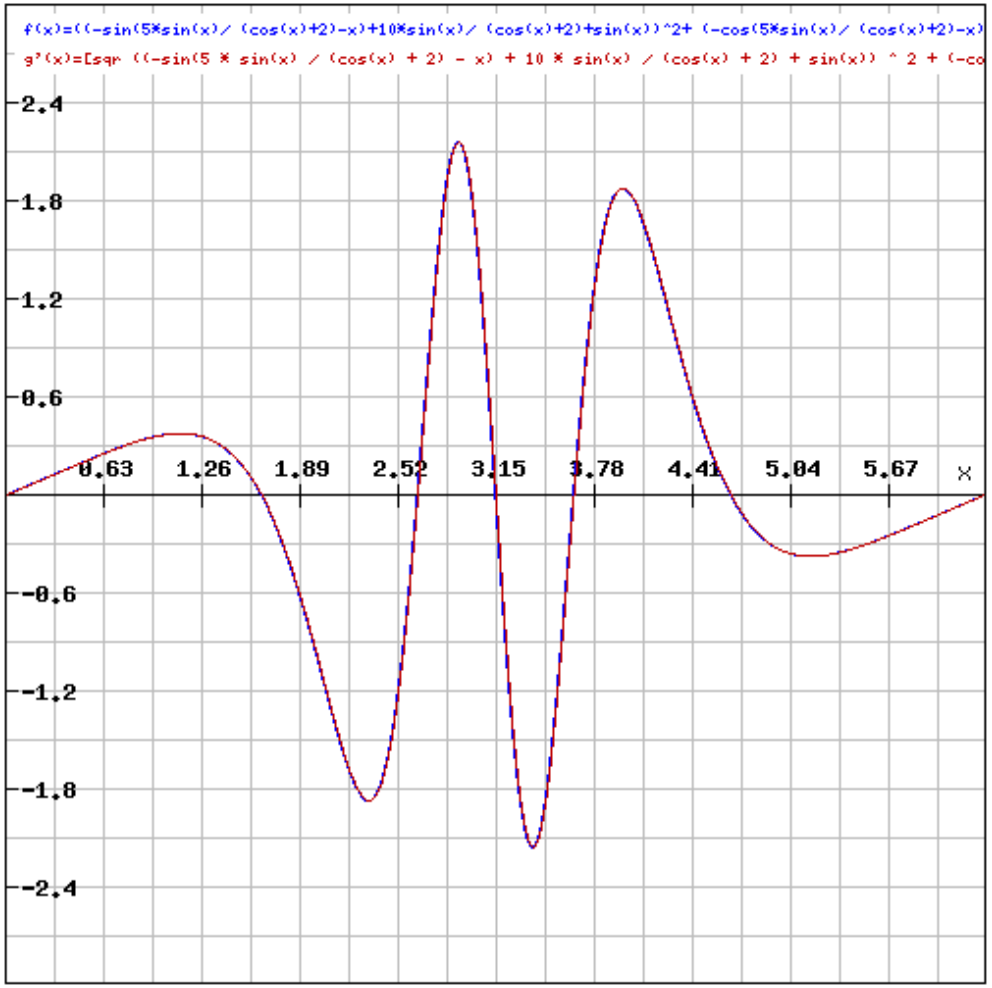
**Third Graph:** h(x) Derivative
Green 1
From to Connect Show term

Draw Reset Standard

**Display properties:** Image type: png

| | | | |
|---|---|---|---|
| Range x-axis from | 0 | to | 6.3 |
| Range y-axis from | -3 | to | 3 |
| Tick marks on x-axis | 10 | on y-axis | 10 |
| Vertical lines | 20 | Horiz. lines | 20 |
| Decimal places | 3 | | |

Log. scale x: No 2 e 10 100 or
Log. scale y: No 2 e 10 100 or



# 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 20 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