Programske paradigme

— Logičko programiranje —

Milena Vujošević Janičić

Matematički fakultet, Univerzitet u Beogradu

Sadržaj

1 Logičko programiranje

Proces izračunavanja

- Imperativna paradigma: naredbe, sekvenca, selekcija, iteracija i propratni efekti
- Funkcionalna paradigma: evaluacija funkcija korišćenjem redukcija
- Logička paradigma: logičke metode (npr metod rezolucije)

Programske paradigme

- Pomenute paradigme se temelje na različitim teorijskim modelima
  - Formalizam za imperativne jezike — Tjuringova mašina. Imperativni jezici imaju svu izražajnost Tjuringove mašine (uz ograničenja memorije i resursa).
  - Formalizam za funkcionalne jezike — Lambda račun. Funkcionalni jezici imaju svu izražajnost lambda računa (uz ograničenja memorije i resursa).
  - Formalizam za logičke jezike — Logika prvog reda. Logika prvog reda nije formalizam izračunljivosti, tako da je teško vršiti poređenje ove vrste. Ipak, većina logičkih jezika je Tjuring-kompletno.

1.1 Rešavanje problema

Rešavanje problema

- U logičkom programiranju, logika se koristi kao deklarativni jezik za opisivanje problema, a dokazivač teorema kao mehanizam za rešavanje problema.
- Rešavanje problema je podeljeno između programera koji opisuje problem i dokazivača teorema koji problem rešava.

Rešavanje problema

- Sintaksa opisa problema je dosta slična u jezicima logičke paradigme, ali dokazivač teorema može da se razlikuje.
- Najznačajniji predstavnik logičke paradigme je Prolog, koji se često koristi i kao sinonim za logičku paradigmu.
- U procesu rešavanja problema, Prolog koristi metod rezolucije

1.2 Razvoj logičkog programiranja

Razvoj logičkog programiranja — teorijske osnove

- 1879. G. Frege - Predikatski račun (sredstvo za analizu formalne strukture čistog mišljenja)
- 1915-1936. Gedel, Erban, Skulem, Turing,... Osnovi teorije izračunljivosti.
- 1965. J.A. Robinson, Metod rezolucije.

1.3 Prvi logički jezik

ABSYS

- 1967. ABSYS — Aberdeen SYStem
- M. Foster, T. Elkok, Group for Computing Research, University of Aberdeen (UK)
- Radi sa tvrđenjima (aksiomama) preko kojih se, nakon postavljanja pitanja, deduktivnim putem generiše odgovor.
- ABSYS je anticipira razne koncepte koji se kasnije koriste u logičkom programiranju.

1.4 Razvoj Prologa

Razvoj Prologa

- 1971. Pod rukovodstvom A. Colmerauer-a u Marselju kreiran Q-System (obrada prirodnih jezika).
- 1972. Q-System preimenovan u PROLOG (PROgramming in LOGic). Saradnja sa Robertom Kovalskim iz Edinburga (automatsko dokazivanje). Implementiran prvi interpretator za Prolog. (Saradnici: Ph. Roussel, R. Pasero, J. Trudel)
- 1974. Na kongresu IFIP-a (International Federation for Information Processing) R. Kavalski predstavio Prolog široj javnosti.
- 1977. David Warren naprvio efikasnu implementaciju kompajlera za Prolog za DEC-10 u Edinburgu. (Edinburški Prolog). Osnova sintakse za moderan Prolog.

Razvoj Prologa

- 1981. Seminari u Sirakuzi i Los Andjelesu.
- 1982. Prva međunarodna konferencija o Prologu u Marselju.
- 1983. Japanski projekat o razvoju računara 5. generacije.
- 1986. The Association for Logic Programming http://www.logicprogramming.org/
- 1993. Završen Japanski projekata razvoja računara 5. generacije.
- 1995. ISO Prolog standarad
- 2007, 2012. Korekcije standarada, dodavanje modula

Razvoj Prologa

- Prolog se često koristi kao sinonim za logičko programiranje.
- Nakon početnih godina intenzivnog razvoja jezika, sada je razvoj usmeren ka integracijama jezika sa drugim programskim jezicima.
- I dalje se istražuju efikasniji algoritmi za prevođenje Prolog programa u izvršni kod.
- Razne implementacije Prologa: BProlog, GNU Prolog, JIProlog, Visual Prolog, SWI Prolog...
- Prolog je uticao na razvoj raznih programskih jezika: ALF, Fril, Gödel, Mercury, Oz, Ciao, Visual Prolog, XSB, λProlog...

1.5 Drugi predstavnici logičke paradigme

ASP i Datalog

- Pored Prologa, značajni predstavnici logičke paradime su i ASP (Answer Set Programming) i Datalog
- Sintaksa Dataloga i jezika koji pripadaju ASP podparadigmi su veoma slične Prologu, ali se proces rešavanja problema razlikuje

ASP i Datalog

- ASP za izračunavanje ne koristi metod rezolucije već rešavače za iskaznu logiku. Koristi se najčešće za pretragu kod NP teških problema, npr bojenje grafova, hamiltonovi ciklusi, velike klike...
- Datalog (logic and databases) je u sintaksnom smislu podskup Prologa koji se koristi za integraciju podataka, izvlačenje podataka, umrežavanje, analizu programa, cloud computing. Datalog može da koristi različite efikasne algoritme za određivanje vrednosti upita. Datalog nije Tjuring kompletan.

1.6 Primena

Primena logičkog programiranja

- Logičko programiranje je pogodno za
  - rešavanje problema matematičke logike,
  - obradu prirodnih jezika,
  - podršku relacionim bazama podataka,
  - automatizaciju projektovanja,
  - simboličko rešavanje jednačina,
  - razne oblasti veštačke inteligencije...
- Logičko programiranje nije pogodno za

- I/O algoritme,
- grafiku,
- numeričke algoritme...

2 Logika prvog reda - osnovni pojmovi

2.1 Sintaksa

Logički deo jezika

- Logički deo jezika prvog reda čine
  - skup promenljivih V,
  - skup logičkih veznika $\{\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow\}$,
  - skup kvantifikatora $\{\exists, \forall\}$,
  - skup logičkih konstanti $\{\top, \bot\}$ i
  - skup pomoćnih simbola $\{(,), , \}$.
- Elementi ovih skupova nazivaju se **logički simboli**.

Rečnik ili signatura

- **Rečnik ili signatura** sastoje se od najviše prebrojivih skupova Σ i Π koje redom nazivamo skupom **funkcijskih simbola** i skupom **predikatskih simbola**, kao i od funkcije ar koja preslikava uniju ovih skupova u nenegativne cele brojeve i koju nazivamo **arnost**.
- Presek svaka dva od prethodno nabrojanih skupova je prazan.
- Funkcijske simbole arnosti 0 zovemo simbolima konstanti.
- Skupovi Σ i Π čine nelogički deo jezika prvog reda, a sve njihove elemente zovemo **nelogičkim simbolima**.

Term

Skup termova nad signaturom $\mathcal{L} = (\Sigma, \Pi, ar)$ i skupom promenljivih V je skup za koji važi:

- svaki simbol konstante (tj. svaki funkcijski simbol arnosti 0) je term;
- svaki simbol promenljive je term;
- ako je f funkcijski simbol za koji je $ar(f) = n$ i t_1 , t_2 , ... , t_n su termovi, onda je i $f(t_1, t_2, ..., t_n)$ term.
- Termovi se mogu dobiti samo konačnom primenom prethodnih pravila.

Atomičke formule

Skup atomičkih formula nad signaturom $\mathcal{L} = (\Sigma, \Pi, ar)$ i skupom promenljivih V je skup za koji važi:

- logičke konstante $\top$ i $\bot$ su atomičke formule;
- ako je p predikatski simbol za koji je $ar(p) = n$ i t_1 , t_2 , ... , t_n su termovi, onda je $p(t_1, t_2, ..., t_n)$ atomička formula.

Dobro zasnovane formule (ili samo formule)

- Skup dobro zasnovanih formula nad signaturom $\mathcal{L} = (\Sigma, \Pi, ar)$ i skupom promenljivih V je skup za koji važi:
  - svaka atomička formula je dobro zasnovana formula;
  - ako je A dobro zasnovana formula, onda je i $(\neg A)$ dobro zasnovana formula;
  - ako su A i B dobro zasnovane formule, onda su i $(A \wedge B)$, $(A \vee B)$, $(A \Rightarrow B)$ i $(A \Leftrightarrow B)$ dobro zasnovane formule;
  - ako je A dobro zasnovana formula i x je promenljiva, onda su $((\forall x)A)$ i $((\exists x)A)$ dobro zasnovane formule.
  - Dobro zasnovane formule se mogu dobiti samo konačnom primenom prethodnih pravila.

Osnovni pojmovi

- **Literal** je atomička formula ili negacija atomičke formule.
- **Klauza** je disjunkcija literala.
- Pod terminom **izraz** podrazumevaju se termovi i (dobro zasnovane) formule.

2.2 Supstitucija i unifikacija

Supstitucija (za term)

- Term dobijen zamenom (supstitucijom) promenljive x termom t_x u termu t označavamo sa $t[x \mapsto t_x]$ i definišemo na sledeći način:
  - ako je t simbol konstante, onda je $t[x \mapsto t_x] = t$;
  - ako je $t = x$, onda je $t[x \mapsto t_x] = t_x$;
  - ako je $t = y$, gde je $y \neq x$, onda je $t[x \mapsto t_x] = t$;
  - ako je $t = f(t_1, t_2, \ldots, t_n)$, onda je $t[x \mapsto t_x] = f(t_1[x \mapsto t_x], t_2[x \mapsto t_x], \ldots, t_n[x \mapsto t_x])$.

Supstitucija (za formule)

- Formulu dobijenu zamenom (supstitucijom) promenljive x termom t_x u formuli $\mathcal{A}$ označavamo sa $\mathcal{A}[x \mapsto t_x]$ i definišemo na sledeći način:
  - $\top[x \mapsto t_x] = \top$;
  - $\bot[x \mapsto t_x] = \bot$;
  - ako je $\mathcal{A} = p(t_1, t_2, \ldots, t_n)$, onda je $\mathcal{A}[x \mapsto t_x] = p(t_1[x \mapsto t_x], t_2[x \mapsto t_x], \ldots, t_n[x \mapsto t_x])$;
  - $(\neg\mathcal{A})[x \mapsto t_x] = \neg(\mathcal{A}[x \mapsto t_x])$;
  - $(\mathcal{A} \wedge \mathcal{B})[x \mapsto t_x] = (\mathcal{A}[x \mapsto t_x] \wedge \mathcal{B}[x \mapsto t_x])$;
  - $(\mathcal{A} \vee \mathcal{B})[x \mapsto t_x] = (\mathcal{A}[x \mapsto t_x] \vee \mathcal{B}[x \mapsto t_x])$;
  - $(\mathcal{A} \Rightarrow \mathcal{B})[x \mapsto t_x] = (\mathcal{A}[x \mapsto t_x] \Rightarrow \mathcal{B}[x \mapsto t_x])$;
  - $(\mathcal{A} \Leftrightarrow \mathcal{B})[x \mapsto t_x] = (\mathcal{A}[x \mapsto t_x] \Leftrightarrow \mathcal{B}[x \mapsto t_x])$;
  - $(\forall x\mathcal{A})[x \mapsto t_x] = (\forall x\mathcal{A})$;
  - $(\exists x\mathcal{A})[x \mapsto t_x] = (\exists x\mathcal{A})$;

Supstitucija (za formule)

- Nastavak:
  - ako je $x \neq y$, neka je z promenljiva koja se ne pojavljuje ni u $(\forall y)\mathcal{A}$ ni u t_x; tada je $(\forall y\mathcal{A})[x \mapsto t_x] = (\forall z)\mathcal{A}[y \mapsto z][x \mapsto t_x]$;
  - ako je $x \neq y$, neka je z promenljiva koja se ne pojavljuje ni u $(\exists y)\mathcal{A}$ ni u t_x; tada je $(\exists y\mathcal{A})[x \mapsto t_x] = (\exists z)\mathcal{A}[y \mapsto z][x \mapsto t_x]$.
- Primetimo da poslednja dva pravila u prethodnoj definiciji obezbeđuju, na primer, da $((\forall y)p(x, y))[x \mapsto y]$ ne bude $(\forall y)p(y, y)$ već $(\forall z)p(y, z)$.

Uopštena zamena (supstitucija)

- **Uopštena zamena** (supstitucija) σ je skup zamena $[x_1 \mapsto t_1]$, $[x_2 \mapsto t_2]$, …, $[x_n \mapsto t_n]$ gde su x_i promenljive i t_i su proizvoljni termovi i gde je $x_i \neq x_j$ za $i \neq j$. Takvu zamenu zapisujemo kraće $[x_1 \mapsto t_1, x_2 \mapsto t_2, \ldots, x_n \mapsto t_n]$.
- Uopštena zamena primenjuje se simultano na sva pojavljivanja promenljivih x_1, x_2, …, x_n u polaznom izrazu i samo na njih (tj. ne primenjuje se na podtermove dobijene zamenama).
- Izraz koji je rezultat primene zamene σ nad izrazom E, označavamo sa $E\sigma$.

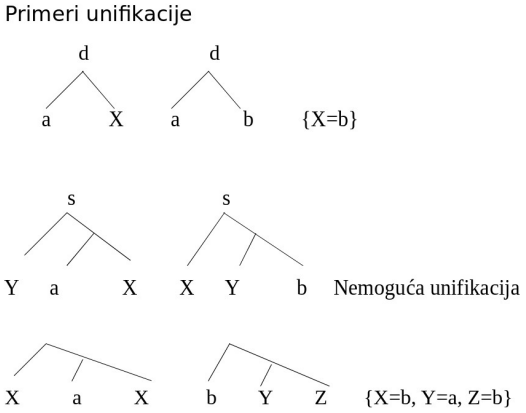
Primeri

- Za $\sigma = [x \mapsto f(y)]$ i $s = g(a, x)$ važi $s\,\sigma = g(a, f(y))$.
- Za $\sigma = [x \mapsto f(x)]$ i $s = g(a, x)$ važi $s\,\sigma = g(a, f(x))$.
- Za $\sigma = [x \mapsto f(y), y \mapsto a]$, $s = g(a, x)$ i $t = g(y, g(x, y))$ važi $s\,\sigma = g(a, f(y))$ i $t\,\sigma = g(a, g(f(y), a))$.

Unifikacija

- Problem unifikacije je problem ispitivanja da li postoji supstitucija koja čini dva izraza (dva terma ili dve formule) jednakim.
- Ako su e_1 i e_2 izrazi i ako postoji supstitucija σ takva da važi $e_1\sigma = e_2\sigma$, onda kažemo da su izrazi e_1 i e_2 **unifikabilni** i da je supstitucija σ **unifikator** za ta dva izraza.
- Dva unifikabilna izraza mogu da imaju više unifikatora.

Unifikacija primeri



Unifikacija

- Neka je term t_1 jednak $g(x, z)$, neka je term t_2 jednak $g(y, f(y))$ i neka je σ supstitucija $[y \mapsto x, z \mapsto f(x)]$. Tada je i $t_1\sigma$ i $t_2\sigma$ jednako $g(x, f(x))$, pa su termovi t_1 i t_2 unifikabilni, a σ je (jedan) njihov unifikator.
- Unifikator termova t_1 i t_2 je npr. i $[x \mapsto a, y \mapsto a, z \mapsto f(a)]$.
- Termovi $g(x, x)$ i $g(y, f(y))$ nisu unifikabilni.

Unifikacija

- Među svim unifikatorima za dva izraza postoji jedan koji je najopštiji, tj svi drugi se mogu dobiti iz njega primenom dodatne supstitucije
- Na primer, ako imamo termove $f(x)$ i $f(y)$ i supstitucije $\sigma_1 = [x \mapsto a, y \mapsto a]$, $\sigma_2 = [x \mapsto b, y \mapsto b]$ i $\sigma = [x \mapsto y]$ u ovom slučaju σ je najopštiji unifikator.
- Postoji algoritam koji pronalazi **najopštiji unifikator** ili vraća neuspeh ukoliko unifikator za dva izraza ne postoji.

2.3 Metod rezolucije

Metod rezolucije

- Metod rezolucije je metod za izvođenje zaključaka (logičkih posledica) u logici prvog reda koji se zasniva na pravilu rezolucije.
- Na primer, posledica formula $A \Rightarrow B$ i $B \Rightarrow C$ je formula $A \Rightarrow C$
- Zapravo, pravilo rezolucije se iskazuje u terminima klauza, tj $\neg A \vee B$ i $\neg B \vee C$ daje $\neg A \vee C$

Primeri

- S = it is sunny,
- C = camping is fun,
- H = the homework is done,
- M = mathematics is easy

Translate the following proposition into the most natural equivalent statement $(M \Rightarrow H) \wedge (S \Rightarrow C)$ If mathematics is easy then the homework is done, and if it is sunny then camping is fun. ili The homework is done when mathematics is easy, and camping is fun when it is sunny.

Primeri

- S = it is sunny,
- C = camping is fun,
- H = the homework is done,
- M = mathematics is easy

Mathematics is easy or camping is fun, as long as it is sunny and the homework is done. If it is sunny and the homework then mathematics is easy or camping is fun. $(S \wedge H) \Rightarrow (M \vee C)$

Primeri

- Ako pada kisa, Joe ponese kisobran. If it rains, Joe brings his umbrella: $(r \Rightarrow u)$ u KNF-u: $\neg r \vee u$
- Ako Joe ima kisobran, on nije mokar. If Joe has an umbrella, he doesn't get wet $(u \Rightarrow \neg w)$ u KNF-u: $\neg u \vee \neg w$
- Ako ne pada kisa, Joe nije mokar. If it doesn't rain, Joe doesn't get wet $(\neg r \Rightarrow \neg w)$ u KNF-u: $r \vee \neg w$

Dokazi da Joe nije mokar. Prove that Joe doesn't get wet $(\neg w)$

Metod rezolucije

1. $\neg r \vee u$ (premisa)
2. $\neg u \vee \neg w$ (premisa)
3. $r \vee \neg w$ (premisa)
4. $\neg r \vee \neg w$ (L1, L2, rezolucija)
5. $\neg w \vee \neg w$ (L3, L4, rezolucija)
6. $\neg w$ (L5, idempotencija)
7. QED

Metod rezolucije

- Obično se dokaz izvodi dobijanjem kontradikcije: tj. dodaje se negacija uslova koji želimo da dokažemo i odatle izvodimo $\perp$, tj $\neg p \Rightarrow \perp$ je isto što i p
- Dokaz kontradikcijom se obično koristi da bi se navodio dokaz tj cilj nam je da dođemo do sve jednostavnijih izraza kako bi stigli do $\perp$

  1. $\neg r \vee u$ (premisa)
  2. $\neg u \vee \neg w$ (premisa)
  3. $r \vee \neg w$ (premisa)
  4. w (negacija zaključka)
  5. $\neg r \vee \neg w$ (L1, L2, rezolucija)
  6. $\neg w \vee \neg w$ (L3, L5, rezolucija)
  7. $\neg w$ (L6, idempotencija)
  8. $\perp$ (L4, L7, rezolucija)

Metod rezolucije

Either Heather attended the meeting or Heather was not invited. If the boss wanted Heather at the meeting, then she was invited. Heather did not attend the meeting. If the boss did not want Heather there, and the boss did not invite her there, then she is going to be fired. Prove Heather is going to be fired.

1. $A \vee \neg I$
2. $\neg W \vee I$
3. $\neg A$
4. $W \vee I \vee F$
5. F

Metod rezolucije

1. $A \vee \neg I$ (premisa)
2. $\neg W \vee I$ (premisa)
3. $\neg A$ (premisa)
4. $W \vee I \vee F$ (premisa)
5. $\neg F$ (negacija zaključka)
6. $W \vee I$ (L4, L5, rezolucija)
7. I (L2, L6, rezolucija, idempotencija)
8. A (L1, L7, rezolucija)
9. $\perp$ (L3, L8, rezolucija)

Metod rezolucije

- Navedeni zaključak ne bi mogao da se izvede iz formula $A \Rightarrow B'$ i $B'' \Rightarrow C$.
- Međutim, ukoliko su B' i B'' unifikabilni, onda se oni mogu učiniti jednakim za neko σ pa je posledica formula $A\sigma \Rightarrow B'\sigma$ i $B''\sigma \Rightarrow C\sigma$ je formula $A\sigma \Rightarrow C\sigma$

Primer

- Loves(x,y) mean "x loves y,"
- Traveler(x) mean "x is a traveler,"
- City(x) mean "x is a city,"
- Lives(x,y) mean "x lives in y."

Prevesti:

$$\exists x \forall y \forall z (City(x) \wedge Traveler(y) \wedge Lives(z, x)) \Rightarrow (Loves(y, x) \wedge \neg Loves(z, x))$$

There is a city that all travelers love but everyone who lives there doesn't love. Some cities are loved by all travelers but no person who lives there.

Primer

- Loves(x,y) mean "x loves y,"
- Traveler(x) mean "x is a traveler,"
- City(x) mean "x is a city,"
- Lives(x,y) mean "x lives in y."

Prevesti u predikatsku logiku: No traveler loves the city they live in.

$$\forall x \forall y ((Traveler(x) \wedge City(y) \wedge Live(x, y)) \Rightarrow \neg Love(x, y))$$

Primer

Premises:

Mother(Lulu, Fifi)

Alive(Lulu)

$\forall x\ \forall y.\text{Mother}(x,y) \Rightarrow \text{Parent}(x,y)$

$\forall x\ \forall y.(\text{Parent}(x,y) \wedge \text{Alive}(x)) \Rightarrow \text{Older}(x,y)$

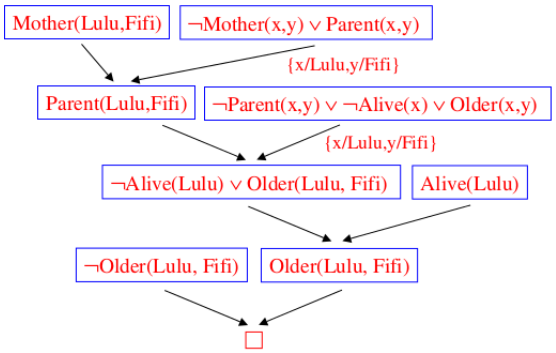
Prove:

Older(Lulu, Fifi)

Denial:

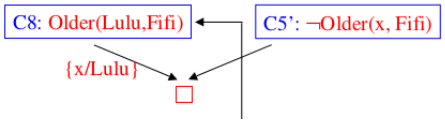
$\neg$Older(Lulu, Fifi)

Primer



Primer

- Suppose the desired conclusion had been
  "Something is older than Fifi"
  $\exists x.\text{Older}(x, \text{Fifi})$
- Denial:
  $\neg\exists x.\text{Older}(x, \text{Fifi})$
  also written as: $\forall x.\neg\text{Older}(x, \text{Fifi})$
  in clause form: $\neg\text{Older}(x, \text{Fifi})$
- Last proof step would have been



Primer

"Who is Lulu older than?"

- Prove that
  "there is an x such that Lulu is older than x"
- In FOL form:
  $\exists x.\text{Older}(\text{Lulu}, x)$
- Denial:
  $\neg\exists x.\text{Older}(\text{Lulu}, x)$
  $\forall x.\neg\text{Older}(\text{Lulu}, x)$
  in clause form: $\neg\text{Older}(\text{Lulu}, x)$
- Successful proof gives
  {x/Fifi} [Verify!!]

Primer

"What is older than what?"

- In FOL form:
  $\exists x\ \exists y.\mathrm{Older}(x, y)$
- Denial:
  $\neg\exists x\exists y.\mathrm{Older}(x, y)$
  in clause form: $\neg\mathrm{Older}(x, y)$
- Successful proof gives
  $\{x/\mathrm{Lulu}, y/\mathrm{Fifi}\}$ [Verify!!]

Primer

A Silly Recitation Problem Symbolize the following argument, and then derive the conclusion from the premises using resolution refutation.

- Nobody who really appreciates Beethoven fails to keep silence while the Moonlight sonata is being played.
- Guinea pigs are hopelessly ignorant of music.
- No one who is hopelessly ignorant of music ever keeps silence while the Moonlight sonata is being played.
- Therefore, guinea pigs never really appreciate Beethoven.

(Taken from a book by Lewis Carroll, logician and author of Alice in Wonderland.)

Primer (by Lewis Carroll)

- The only animals in this house are cats
- Every animal that loves to gaze at the moon is suitable for a pet
- When I detest an animal, I avoid it
- No animals are carnivorous unless they prowl at night
- No cat fails to kill mice
- No animals ever like me, except those that are in this house
- Kangaroos are not suitable for pets
- None but carnivorous animals kill mice
- I detest animals that do not like me
- Animals that prowl at night always love to gaze at the moon
- Therefore, I always avoid a kangaroo

Primer

Simple problem

- Schubert Steamroller:
  - Wolves, foxes, birds, caterpillars, and snails are animals and there are some of each of them. Also there are some grains, and grains are plants. Every animal either likes to eat all plants or all animals much smaller than itself that like to eat some plants. Caterpillars and snails are much smaller than birds, which are much smaller than foxes, which are much smaller than wolves. Wolves do not like to eat foxes or grains, while birds like to eat caterpillars but not snails. Caterpillars and snails like to eat some plants.
- Prove: there is an animal that likes to eat a grain-eating animal
- Some of the necessary logical forms:
  - $\forall$ x (Wolf(x) $\rightarrow$ animal(x))
  - $\forall$ x $\forall$ y ((Caterpillar(x) $\vee$ Bird(y)) $\rightarrow$ Smaller(x,y)
  - $\exists$ x bird(x)
- Requires almost 150 resolution steps (minimal)

3 Prolog

3.1 Teorijske osnove

Teorijske osnove i Prolog

- Kao što se Haskell oslanja na lambda račun, tako se Prolog oslanja na logiku prvog reda i metod rezolucije
- Potrebno je poznavanje pojmova iskazne logike i logike prvog reda.
- Prološki zapis programa je izveden iz zapisa formula logike prvog reda.

Teorijske osnove i Prolog

- Prološki zapis je redukovan zapis formula logike prvog reda — ne mogu se sve formule logike prvog reda izraziti u Prologu.
- U Prologu se mogu izraziti samo Hornove klauze.
- **Hornova klauza**: disjunkcija literala sa najviše jednim nenegiranim literalom

Teorijske osnove i Prolog

- Hornova klauza odgovara implikaciji

$$(A_1 \wedge A_2 ... \wedge A_n) \Rightarrow B$$

jer je to ekvivalentno sa

$$\neg(A_1 \wedge A_2 ... \wedge A_n) \vee B$$

odnosno sa

$$\neg A_1 \vee \neg A_2 ... \vee \neg A_n \vee B$$

- Hornove klauze se u prologu zapisuju u obliku

$$B \Leftarrow (A_1 \wedge A_2 ... \wedge A_n)$$

pri čemu se znak $\Leftarrow$ zapisuje kao :-

$$B :- (A_1, A_2, ..., A_n)$$

Teorijske osnove i Prolog

- Hornove klauze omogućavaju efikasnu primenu metoda rezolucije
- Hornove klauze predstavljaju ograničenje: ne može se izraziti tvrđenje oblika $\neg A \Rightarrow B$
- Prethodnom tvrđenju odgovara formula $A \vee B$, dakle formula koja sadrži dva nenegirana literala

Unifikacija

- Supstitucija je preslikavanje koje promenljive preslikava u termove. Supstitucija se sastoji od konačnog broja pravila preslikavanja, npr $x \to a, y \to f(a, b), u \to v$
- Dva terma t i u se mogu unifikovati ako i samo ako postoji supstitucija σ tako da važi $t\sigma = u\sigma$
- Unifikacija se koristi prilikom traženja rešenja u okviru Prologa.

Hello world!

```
?- write('Hello world!'), nl.
Hello world!
true.
```

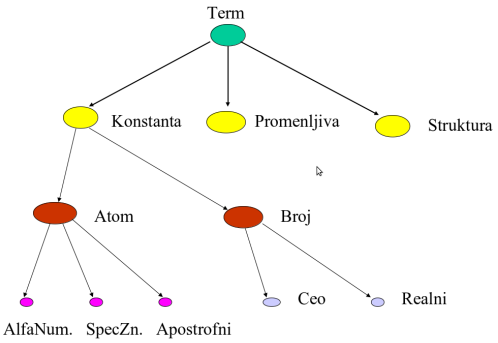
Prolog program

- Programiranje u Prologu se sastoji u
  - obezbeđivanju činjenica o objektima i odnosima među njima
  - defnisanju pravila o objektima i odnosima među njima,
  - formiranju upita o objektima i odnosima među njima.

3.2 Sintaksa

Termovi u Prologu

Termovi su osnovni gradivni elementi Prologa i širi su od pojma terma logike prvog reda, tj $Term\_Prologa \neq Term\_Logike\_Prvog\_Reda$



Termovi

- Promenljive se zapisuju početnim velikim slovom ili simbolom _ (simbolom _ počinju imena anonimnih promenljivih čije vrednosti nisu bitne)
- Strukture ili predikati se grade od atoma i termova. Ako je p atom, a t_1, t_2 do t_n termovi, onda je $p(t_1, t_2, ...t_n)$ term.

3.3 Programi

Činjenice

- Činjenice: Hornove klauze bez negiranih literala (dakle, samo B).
- Pomoću činjenica opisuju se svojstva objekta, odnosno relacije između objekata.
- Na primer

```
zivotinja(panda).
biljka(bambus).
jede(panda, bambus).
```

- Činjenice su konstrukcije Prologa koje se sastoje iz funktora iza kojeg slede argumenati između malih zagrada.
- Ukoliko ima više argumenata, razdvajaju se zapetama.
- Funktor predstavlja naziv relacije, a argumenti nazive objekata.

Činjenice

- Imena u činjenicama treba zadavati u skladu za njihovim značenjem (semantikom).
- Prolog prihvata činjenice kao apsolutne istine (aksiome) i ne proverava njihovu tačnost.
- U tom smislu činjenice u Prologu su:

```
biljka(panda).
zivotinja(bambus).
jede(bambus, panda).
```

- Skup činjenica formira bazu činjenica (bazu podataka).

Pravila

- Pravila: Hornove klauze u punom obliku.
- Pravila predstavljaju opšta tvrđenja o objektima i relacijama među njima.
- To su konstrukcije koje se sastoje iz glave i tela povezanih vrat-simbolom (koji odgovara implikaciji).

  ```
  GLAVA  :-  TELO.
  ```

  Pravila su uslovne rečenice oblika: Važi GLAVA ako važi TELO.

Pravila

- Na primer, ako želimo da saopštimo da sve što poseduje Ana poseduje i Pavle, ne moramo navoditi sve činjenice o posedovanju predmeta za Pavla, već je dovoljno navesti pravilo:

  ```
  poseduje(pavle,X) :- poseduje(ana, X).
  ```

- Pravila mogu uključivati više različitih predikata razdvojenih zarezom (zarez odgovara logičkoj konjunkciji). Na primer:

  ```
  sunce(beograd).
  kiša(beograd).
  duga(X) :- kiša(X), sunce(X).
  ```

- Za razliku od činjenica, u pravila su uključene promenljive.

Pravila

- U definisanju pravila, može se koristiti rekurzija.
- Ali, ne mogu se koristiti beskonačni ciklusi. Na primer:

  ```
  roditelj(A,B) :- dete(B, A).
  dete(A,B) :- roditelj(B,A).
  ```

- Takođe, levostrane rekurzije nisu dozvoljene. Na primer, naredno pravilo je u redu:

  ```
  macka(X) :- majka(Y,X), macka(Y).
  ```

  Ali, naredno pravilo (iako deklarativno ima isti smisao) će dovesti do greške prilikom generisanja rešenja

  ```
  macka(X) :- macka(Y), majka(Y,X).
  ```

- Ovo je posledica načina na koji Prolog dolazi do rešenja (o čemu će biti više reči kasnije).

Baza znanja

- Pomoću činjenica i pravila saopštavaju se Prologu informacije o relacijama i objektima.
- Činjenice i pravila se zajedničkim imenom nazivaju tvrđenja.
- Skup činjenica i pravila određuje bazu znanja (bazu podataka u širem smislu).

Closed-World Assumption

- Jedino 'znanje' kojim raspolaže Prolog, u vezi sa problemom koji rešava, nalazi se u bazi znanja.
- Pretpostavka zatvorenosti, eng. Closed-World Assumption — netačno je sve što nije eksplicitno navedeno kao tačno.
- Prolog nalazi rešenja (daje odgovore na postavljene upite) korišćenjem činjenica i pravila iz baze znanja.

Upiti

- Upiti: Hornove klauze bez nenegiranog literala $(A_1, A_2, ..., A_n)$
- Upiti su konstrukcije Prologa preko kojih korisik komunicira sa bazom znanja.
- Upiti se najčešće realizuju u interaktivnom radu korisnika sa računarom.
- To je moguće ako korisniku stoji na raspolaganju Prolog-mašina (interpretator za Prolog sa pratećim softverskim komponentama).

Interpretator

- Odzivni znak (promt) interpretatora je: `?-`, pomoć se može dobiti sa `help` a izlazak iz interpretatora sa `halt`
- Ubacivanje činjenica i pravila se ostvaruje sa `assert`, npr `assert(zivotinja(panda)).`
- Sa `listing` se može proveriti spisak postojećih činjenica i pravila

Upiti

- Najednostavniji upiti služe za ispitivanje da li se činjenice nalaze u bazi podataka.
- Upiti su konstrukcije koje se završavaju tačkom.
- Na primer:

```
?-zivotinja(panda).
yes
?-zivotinja(bambus).
no
```

Upiti

- Za postavljanje složenijih upita, tj. za dobijanje i drugačijih odgovora od yes i no, potrebno je koristiti promenljive.
- Na primer:

```
?-zivotinja(X).
X = panda
?-biljka(Y).
Y = bambus
```

- Svaki upit Prolog "shvata" kao cilj koji treba ispuniti (dostići).
- Prolog pokušava da instancira promenljive i prikaže njihovu vrednost.
- Prolog generiše odgovore upoređujući upit sa bazom podataka od početka ka kraju.

Upiti

- Jedan cilj (upit) može se sastojati iz nekoliko potciljeva.
- Na taj način postavljaju se složeniji upiti.
- Ukoliko cilj sadrži više potciljeva, potciljevi se razdvajaju zapetom (,). Ovde zapeta ima ulogu operatora konjunkcije.
- Na primer:

```
?- jede(panda, Nesto), jede(medved, Nesto).
```

  Ovo se shvata kao pitanje: da li postoji Nesto što jede i panda i medved i šta je primer toga?

- Prolog ispunjava cilj tako što ispunjava svaki potcilj i to s levo u desno.

Primer

```
planeta(merkur).
planeta(venera).
planeta(zemlja).
planeta(mars).
planeta(venera).
planeta(saturn).
planeta(jupiter).
planeta(uran).
planeta(neptun).
veca(venera, merkur).
veca(zemlja, venera).
veca(uran, zemlja).
veca(saturn, uran).
manja(saturn, jupiter).
veca(X,Y) :- zvezda(X), planeta(Y).
zvezda(sunce).
veca(X,Y) :- planeta(Z), veca(X,Z), veca(Z,Y).
veca(X,Y) :- manja(Y,X).
manja(venera, saturn).
manja(zemlja, jupiter).
```

3.4 Sistemski predikati i operatori

Unifikacija =

- Unifikacija (ujednačavanje) je jedna od najvažnijih operacija nad termima. Simbol za ovu operaciju u Prologu je =.
- Neformalno, unifikacija nad termima T i S vrši se na sledeći način:
  (a) Ako su T i S konstante, unifikuju se ako predstavljaju istu konstantu.
  - Ako je S promenljiva, a T proizvoljan objekat, unifikacija se vrši tako što se termu S dodeli T. Obrnuto, ako je S proizvoljan objekat, a T promenljiva, onda T primi vrednost terma S.
  - Ako su S i T strukture, unifikacija se može izvršiti ako:
    * imaju istu arnost i jednake dominantne simbole (funktore) i
    * sve odgovarajuće komponente se mogu unifikovati.

Sistemski predikati i operatori

- Logika prvog reda je veoma apstraktna notacija koja nema algoritamsku prirodu.
- Zato je prirodno da jezici, kao što je Prolog, moraju da sadrže proširenja koja jeziku daju algoritmičnost, kao i proširenja potrebna za komunikaciju sa spoljašnjim svetom.

Sistemski predikati i operatori

Sistemski predikati — predikati koji su ugrađeni u sam jezik

- Operator unifikacije =
- Operatori jednakosti i nejednakosti: (unifikacija) =, \=, (dodela) is, (aritmetika) =:=, =/=, (identitet) ==, \==
- Relacijski operatori <, >, =<, >=.
- Aritmetički operatori $+, -, *, /, mod, is$
- Rad sa datotekama (u smislu učitavanja baze znanja): user, consult, reconsult
- true, fail, not, !

Sistemski predikati i operatori

- Klasifikacija terma var, nonvar, atom, integer, atomic
- Rad sa klauzama listing, asserta, assertz, retract, clause
- Rad sa strukturama functor, arg, ..=, name
- Predikati repeat i call
- Rad sa ulazom i izlazom: read, get, write, nl, tab, put, display
- Rad sa datotekama: see, seen, seeing, tell, told, telling
- Definisanje korisničkih operatora op

Sistemski predikati i operatori

- Bez sistemskih predikata Prolog je čist deklarativni jezik.
- Svaka verzija Prologa ima neke specifične sistemske predikate.
- Dodavanjem predikata dodaju se propratni efekti.
- Najveći broj operatora zahteva konkretizovane (instancirane) promenljive
- Na primer, $X < Y$, zahteva konkretizovane promenljive X i Y.

Primeri

```
dana_u_mesecu(31, januar, _).
dana_u_mesecu(29, februar, G):- prestupna(G).
dana_u_mesecu(28, februar, G):- not(prestupna(G)).
dana_u_mesecu(31, mart, _).
dana_u_mesecu(30, april, _).
dana_u_mesecu(31, maj, _).
dana_u_mesecu(30, juni, _).
dana_u_mesecu(31, juli, _).
dana_u_mesecu(31, avgust, _).
dana_u_mesecu(30, septembar, _).
dana_u_mesecu(31, oktobar, _).
dana_u_mesecu(30, novembar, _).
dana_u_mesecu(31, decembar, _).
prestupna(G) :- je_deljivo(G,400).
prestupna(G) :- not(je_deljivo(G,100)), je_deljivo(G,4).
je_deljivo(X,Y) :- 0 is X mod Y.
```

Primeri

```
vojnik('Aca Peric', 183, 78).
vojnik('Milan Ilic', 192, 93).
vojnik('Stanoje Sosic', 173, 81).
vojnik('Sasa Minic', 162, 58).
vojnik('Dragan Sadzakov', 180, 103).
vojnik('Pera Peric', 200, 80).
vojnik('Rade Dokic', 160, 56).
zadovoljava(Ime) :- vojnik(Ime, Visina, Tezina),
                    Visina>170, Visina<190, Tezina =< 95,
                    Tezina>60.
otpada(Ime) :- vojnik(Ime, _, Tezina),Tezina =< 60.
otpada(Ime) :- vojnik(Ime, Visina, _), Visina>200.
```

Metaprogramiranje u Prologu

- Prolog programi su jednoobrazni — podaci i programi izgledaju isto.
- To daje mogućnost jednostavnog metaprogramiranja: kreiranje programa koji u fazi izvršavanja mogu da stvaraju ili menjaju druge programe, ili da sami sebe proširuju.
- Najjednostavnije metaprogramiranje u Prologu je dodavanje činjenica u bazu znanja prilikom izvršavanja programa (asserta, assertz), ili njihovo izbacivanje iz baze podataka (retract, retractall)
- Predikati functor, arg, =.. omogućavaju kreiranje i zaključivanje o novim predikatima za vreme izvršavanja programa.

Liste

- Liste su važne i u Prologu
- Lista predstavlja niz uređenih elemenata proizvoljne dužine. Element liste može biti bilo koji term pa čak i druga lista.
- Lista je: prazna lista u oznaci []; struktura .(G,R) gde je G ma koji term, a R lista.
- Funktor u strukturi liste je tačka (.), prvi argument naziva se glava, a drugi rep. Drugi argument je uvek lista.
- Zbog rekurzivne definicije liste, rekurzija je najpogodniji način za obradu listi.

Liste

- Ako pođemo od definicije liste, imamao:
  - `[]` prazna lista (lista koja ne sadrži ni jedan element)
  - `.(a, [])` je jednočlana lista gde je a nekava term.
  - `.(b, .(a,[]))` je dvočlana lista (a i b su termi).
  - `.(c, .(b, .(a,[])))` je tročlana lista (a, b i c su termi).
  - ...

Liste

- Zapis liste .(a,.(b, .(c, []))) je glomazan i nepregledan.
- Zato se koristi zapis [], [a], [a,b], [a, b,c], [a,b,c,d], ...
- Na primer

```
Lista                          Glava              Rep
[iva, mara, dara]              iva                [mara, dara]
[[voz, tramvaj],trolejbus]     [voz, tramvaj]     [trolejbus]
[a]                            a                  []
[]                             -                  -
```

Primer

```
?- duzina(Lista, Duzina).
duzina([],0).
duzina([G|R],D) :- duzina(R,D1), D  is D1+1.

?- suma(Lista, ZbirElemenataListe).
suma([], 0).
suma([Glava|Rep],S) :-suma(Rep,S1), S is Glava+S1.
```

3.5 Deklarativna i proceduralna interpretacija

Činjenica

Prolog-konstrukcije se mogu interpretirati deklarativno i proceduralno.

- Činjenica: p(a).
- Deklarativno: p(a) je istinito.
- Proceduralno: zadatak p(a) je izvršen.

Pravilo

- Pravilo: p(X) :- q(X), r(X).
- Deklarivno: Za svako X, p(X) je istinito ako je istinito q(X) i r(X).
- Proceduralno: Da bi se izvršio zadatak p(X), prvo izvrši zadatak q(X), a zatim izvrši zadatak r(X).

U proceduralnom tumačenju se određuje i redosled ispunjavanja cilja.

Upit

- Upit: ?- p(X).
- Deklarativno: Da li postoji vrednost promenljive X za koju važi p(X).
- Proceduralno: Izračunavanjem ostvari cilj p(X) i nađi vrednost promenljive X za koju važi svojstvo p.

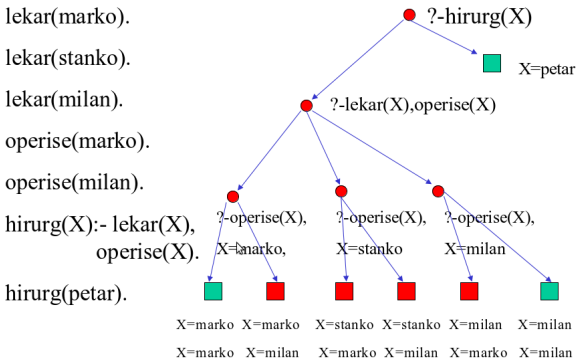
3.6 Stablo izvođenja

Kako Prolog rešava postavljeni problem?

- Svaki upit Prolog tretira kao cillj koji treba dostići (ostvariti, ispuniti) — proceduralna interpretacija.
- Ostvarivanje cilja Prolog-mašina čini pokušavajući da dokaže saglasnost cilja sa bazom znanja. U tom procesu baza znanja se pregleda od vrha ka dnu i moguće su dve situacije:
  - pronađeno je tvrđenje koje se uparuje sa postvljenim ciljem (uspešno je ostavaren cilj - uspeh)
  - nije pronađeno tvrđenje koje se uparuje sa postavljenim ciljem (cilj nije ispunjen - neuspeh).
- U slučaju uspeha, korisnik može zahtevati da se ponovo dokaže saglasnost cilja sa bazom podataka (pronalaženje novog rešenja).

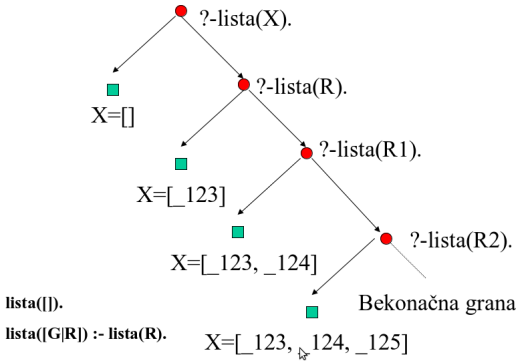
Stablo izvođenja

Stablo izvođenja omogućava slikovit prikaz načina rešavanja problema u Prologu.



Stablo izvođenja

Stablo izvođenja omogućava slikovit prikaz načina rešavanja problema u Prologu.



Terminologija

- Stablo izvođenja — stablo pretrage
- Stablo izvođenja se sastoji iz grana i čvorova.
- Grana u stablu izvođenja može biti konačna ili beskonačna.
- Stablo izvođenja je konačno ako su sve grane u njemu konačne, u suprotnom je beskonačno.

Terminologija

- Čvorovi koji su označeni upitom (ciljem) nazivaju se nezavršnim. (Iz njih se izvode sledeći čvorovi.)
- Listovi u stablu izvođenja nazivaju se završnim čvorovima. Grane koje vode do završnih čvorova su konačne. Grane koje nemaju završne čvorove su beskonačne.

- Ako završni čvor daje rešenje, naziva se se čvor uspeha (označen zelenom bojom), a odgovarajuća grana je grana uspeha.
- Završni čvor koji ne predstavlja rešenje je čvor neuspeha (označen crvenom bojom), a odgovarajuća grana je grana neuspeha.

Veza stabla izvođenja i metoda rezolucije

- U smislu deklarativne semantike, stablo izvođenja odgovara poretku primene pravila rezolucije na postojeći skup činjenica i pravila.
- U smislu proceduralne semantike, stablo izvođenja odgovara procesu ispunjavanje ciljeva i podciljeva.
- Redosled činjenica i pravila u bazi znanja određuje redosled ispunjavanja ciljeva i izgled stabla izvođenja.
- S obzirom da su sve klauze Hornove, proces izvođenja je relativno jednostavan, ali efikasnost može i dalje da bude problem.

Veza stabla izvođenja i metoda rezolucije

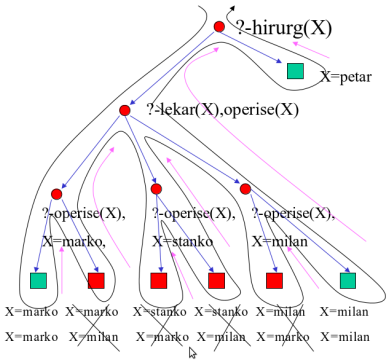
- Implementacija rezolucije može da se ostvari izvođenjem u širinu i izvođenjem u dubinu.
- Izvođenjem u širinu, radi se paralelno na svim potciljevima datog cilja, dok se izvođenjem u dubinu najpre izvede jedan potcilj, pa se dalje nastavlja sa ostalim potciljevima.
- Koja implementacija bi bila bolja?

Veza stabla izvođenja i metoda rezolucije

- U Prologu je implementacija ostvarena izvođenjem u dubinu, jer su na taj način manji memorijski zahtevi.
- To znači da redosled tvrđenja u bazi znanja utiče i na efikasnost pronalaženja rešenja.
- Takođe, redosled može da utiče i na konačnost najlevlje grane, pa time i na to da li će rešenje uopšte biti pronađeno.

Način obilaska stabla izvođenja i nalaženje rešenja

Pomoću stabla izvođenja potpuno je određen prostor izvođenja za jedan cilj. Njega čine svi putevi koji vode od korene stabla do njegovih čvorova.



Veza stabla izvođenja i metoda rezolucije

- Kako je redosled izvođenja poznat i deterministički određen, to se može iskoristiti za dobijanje na efikasnosti Prolog programa time što se činjenice i pravila u bazi znanja ređaju u odgovarajućem redosledu.
- Takođe, na efikasnost utiče i davanje pogodnog redosleda potciljeva u pravilima.
- Ovo narušava pravila deklarativnosti po kojima redosled ne bi trebao da utiče na izvršavanje programa.

3.7 Operator sečenja

Operator sečenja !

- Cut opertor, operator sečenja, rez operator, opertor odsecanja
- Sistemski operator koji omogućava brže izvršavanje programa i uštedu memorijskog prostora kroz eksplicitnu kontrolu backtracking-s
- Ovaj operator je zapravo cilj koji uvek uspeva
- Operator sečenja odseca pojedine grane na stablu pretraživanja, pa samim tim smanjuje prostor pretraživanja. To omogućava brže nalaženje rešenja. U isto vreme ne moraju se pamtiti mnogobrojne tačke prilikom traženja sa vraćanjem, što dovodi do uštede memorijskog prostora.

Operator sečenja

- Neka imamo tvrđenje:

```
A :- B1, B2, ..., Bk, !, ..., Bm
```

- Kada se neki cilj G unifikuje sa A, aktivira se prethodno tvrđenje.
- Tada se cilj G naziva roditeljski cilj za A.
- Kada se dođe do reza, uspeli su potciljevi: B1, B2, ...,Bk.

- Sečenje uspeva i rešenje B1, B2, ..., Bk se ”zamrzava”, tj. predikat onemogućava traženje alternativnih rešenja.
- Takođe se onemogućava ujedinjavanje cilja A sa glavom nekog drugog predikata koji je u bazi podataka iza navedenog tvrđenja.

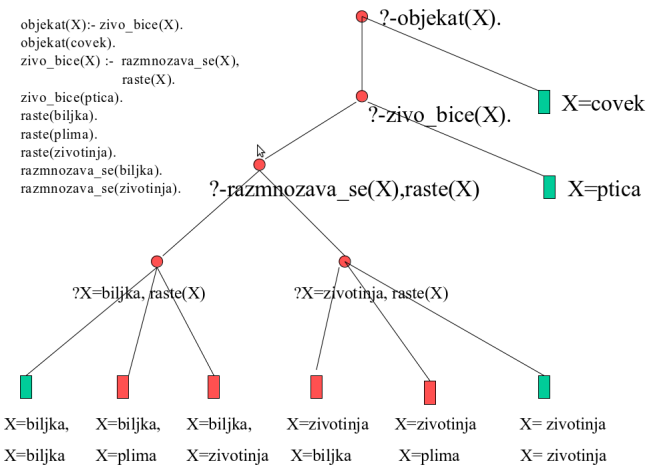
Operator sečenja

- Na primer:

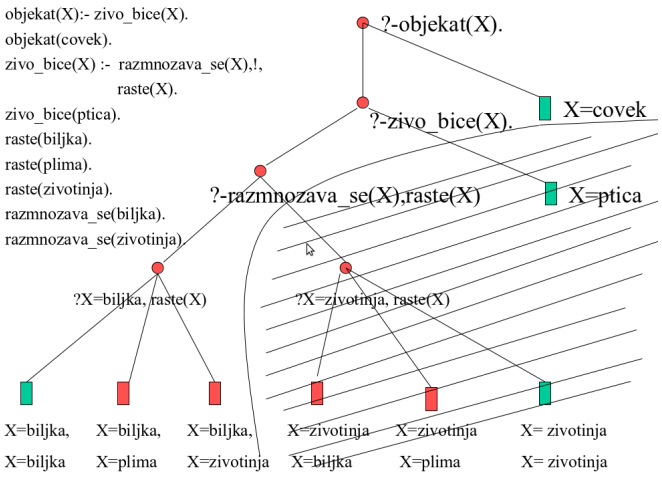
```
A :- P, Q, R, !, S, T.
A :- U.
G :- L, A, D.
 ?- G.
```

- U startu, traženje sa vraćanjem moguće je samo za: P,Q i R.
- Kad uspe R uspeva i sečenje i alternativna rešenja se više ne traže.
- Alternativno rešenje A:-U, takođe se ne razmatra.
- Alternativna rešenja su moguća između S i T, tj desno od operatora sečenja je moguć backtracking
- Ukoliko se ne nađe rešenje za fiksirane vrednosti P, Q, R, onda je dozvoljen backtracking.

Primer



Primer



Problemi sa operatrom sečenja

- Predikat sečenja može se uporediti sa GOTO-naredbom u proceduralnim jezicima.
- Narušava deklarativni stil programiranja i može da proizvede neželjene efekte (greške).
- Pogrešna uptoreba operatora sečenaja je najčešći uzrok grešaka programiranja u Prologu.
- Ipak, ovaj predikat je često neophodan za dobijanje efikasnih rešenja i treba ga koristiti (ali oprezno!).

Problemi sa operatrom sečenja

- Ako se predikat sečenja upotrebljava tako da ne narušava deklarativno svojstvo predikata tj. ne menja njegovu semantiku niti skup rešenja (već samo pojačava njegova deterministička svojstva i ubrzava proces izračunavanja), onda se naziva zeleni predikat sečenja.
- U suprotnom, reč je o crvenom predikatu sečenja.
- Treba biti veoma oprezan sa upotrebom predikata sečenja.

Primene operatora sečenja

- Kada želimo da saopštimo Prologu: "Nađeno je potrebno rešenje, ne treba dalje tražiti!"

```
auto(mercedes, 200000,  10000).
auto(skoda, 100000, 5000).
auto(fiat, 50000, 4000).
auto(bmv, 12000, 12000).
zadovoljava(M) :- auto(M, K, C), C<5000, !.
```

Primene operatora sečenja

- Kada želimo da saopštimo Prologu: "Nađeno je jedinstveno rešenje i ne treba dalje tražiti!"

```
kolicnik(N1,N2,Rez):- ceo_broj(Rez),
           P1 is Rez*N2,
           P2 is (Rez+1)*N2,
           P1=<N1, P2>N1, !.
ceo_broj(0).
ceo_broj(X) :- ceo_broj(Y), X is Y+1.
```

Primene operatora sečenja

- Kada želimo da saopštmo Prologu: "Na pogrešnom si putu, završiti pokušaj zadovoljenja cilja!".

```
inostrani(dzon).
inostrani(martin).
dohodak(dzon, 5000).
dohodak(martin, 10000).
dohodak(dragan, 10000).
dohodak(pera, 20000).
placa_porez(X) :- inostrani(X), !, fail.
placa_porez(X) :- dohodak(X,D), D>10000.
```

- Koristi se u kombinaciji sa fail-predikatom. Može samo da se koristi za proverne svrhe, ne i za generisanje rešenja.

3.8 Svojstva Prologa

Svojstva Prologa

- Prolog nije čist deklarativni jezik (kontrola rezolucije i backtracking-a)
- Prolog je restrikcija logike prvog reda na formule koje se mogu svesti na Hornove kauze
- Hornovim klauzama ne mogu se opisati sva tvrđenja logike prvog reda

Svojstva Prologa

- Prolog može da dokaže da je neki cilj ispunjen, ali ne i da neki cilj nije ispunjen.
- To je posledica oblika Hornovih klauza $A :- B_1, B_2 ... B_n$ ako su sve B_i tačne, možemo da zaključimo da je A tačno, ali ne postoji način da zaključimo da A nije tačno
- Prema tome, ukoliko Prolog ne može da dokaže da je neki cilj ispunjen, on prijavljuje da cilj nije ispunjen iako je to zapravo nemogućnost dokazivanja tačnosti, a ne dokazivanje netačnosti.
- Preciznije, Prolog je true/fail sistem, a ne true/false sistem i to treba imati na umu.

Svojstva Prologa — operator NOT

- Operator NOT nije operator negacije u smislu logičke negacije.
- Operator NOT se naziva „negacija kao nuspeh" — NOT(cilj) uspeva ukoliko cilj ne uspeva
- Zapravo, logičko NOT ne može da bude sastavni deo Prologa što je posledica oblika Hornovih klauza.
- To znači da, ukoliko imamo npr naredni oblik `NOT(NOT(cilj))` to ne mora da bude ekvivalentno sa cilj, što može da dovede do raznih problema i neočekivanih rezultata.
- Zbog toga i sa upotrebom operatora NOT treba biti veoma pažljiv i koristiti ga samo za instancirane promenljive.

Svojstva Prologa — operator NOT

```
vozac(janko).                    ?-dobar_vozac(X).
vozac(marko).                    no
vozac(petar).                    ?-dobar_vozac(petar).
pije(janko).                     yes
pije(marko).                     ?-pije(X).
dobar_vozac(X) :- not(pije(X)),  X=janko;
                  vozac(X).      X=marko;
                                 no
                                 ?-not(not(pije(X))).
                                 yes
```

Svojstva Prologa — operator NOT

```
vozac(janko).                ?-dobar_vozac(X).
vozac(marko).                X=petar;
vozac(petar).                no
pije(janko).                 ?-dobar_vozac(petar).
pije(marko).                 yes

dobar_vozac(X) :- vozac(X), not(pije(X)).
```

Svojstva Prologa — operator NOT

Objasniti naredne rezultate:

```
?- not(X==1), X=1.
X = 1
yes
?- X=1, not(X==1).
no
```

Operator NOT

X nije instancirano, pa stoga nije identično jedinici, i not operator uspeva. Da bi uspeo i drugi cilj, X se unifikuje sa 1.

```
?- not(X==1), X=1.
X = 1
yes
```

U ovom slučaju, X se najpre unifikuje sa 1, i kako je 1 == 1 not ne uspeva.

```
?- X=1, not(X==1).
no
```

Svojstva Prologa — generisanje efikasnih algoritama

- Osnovni cilj logičkog programiranja je da se obezbedi programiranje takvo da programer da specifikaciju šta program treba da uradi bez specifikacije na koji način to treba da se ostvari.
- Visok nivo apstrakcije i deklarativnosti nosi sa sobom cenu neefikasnosti.

Svojstva Prologa — generisanje efikasnih algoritama

- Sortiranje može jednostavno da se defniše kao generisanje permutacije i provera da li je permutacija sortirana.
- Na primer, kriterijum da li je niz sortiran može u Prologu da se napiše:

```
sorted([]).
sorted([X]).
sorted([X, Y | List]) :- X =< Y, sorted([Y | List]).
```

- Međutim, sam algoritam sortiranja je potpuno neefikasan: problem je što nije dat način na koji da se izvrši sortiranje i jedini način da se to ostvari je enumeracijom svih permutacija liste dok se ne kreira ona permutacija koja zadovoljava svojstvo sortiranosti liste.

Svojstva Prologa — generisanje efikasnih algoritama

- Zapravo, ne postoji način da se opis sortiranosti liste prevede u efikasan način sortiranja elemenata liste — rezolucija ne može da generiše efikasan algoritam.
- Prema tome, ako želimo efikasno izvršavanje, moramo da damo specifikaciju algoritma i u Prologu.
- Možda ćemo jednoga dana doći do tačke da su sami programski sistemi u mogćnosti da otkriju dobre algoritme iz deklarativnih specifikacija, ali to još uvek nije slučaj.

Svojstva Prologa — moduli

- Kreiranje modula u Prologu je propisano ISO standardom, ali ne podržavaju svi kompajleri ovo svojstvo što značajno otežava razvoj kompleksnog softvera.
- Postoje nekompatibilnosti između sistema modula različitih Prolog kompajlera.
- Takođe, većina kompajlera implementira i dodatne funkcionalnosti koje nisu podržane standardom, što onemogućava kompatibilnost između kompajlera.

Svojstva Prologa — razno

- Prolog je Tjuring-kompletan jezik, to se može pokazati korišćenjem Prologa da simulira Tjuringovu mašinu.
- Prolog je netipiziran jezik, postoje razni pokušaji uvođenja tipova u Prolog
- Prolog ima mehanizam prepoznavanja i optimizacije repne rekurzije, tako da se ona izvršava jednako efikasno kao i petlje u proceduralnim jezicima
- Predikati višeg reda su predikati koji uzimaju jedan ili više predikata kao svoje argumenate. To izlazi iz okvira logike prvog reda, ali ISO standard propisuje neke predikate višeg reda (npr call predikat)

4 Programiranje ograničenja u logičkom programiranju

Programiranje ograničenja

- Programiranje ograničenja je deklaratinvo programiranje
- Programiranje ograničenja je programiranje u kojem se relacije između promenljivih zadaju u vidu ograničenja.
- Od sistema se zatim očekuje da izračuna rešenje,tj da izračuna vrednosti promenljivih koje zadovoljavaju data ograničenja.
- Ograničenja se razlikuju od ograničenja u imperativnoj paradigmi. Na primer, $x < y$ u imperativnoj paradigmi se evaluira u tačno ili netačno, dok u paradigmi ograničenja zadaje relaciju između objekata x i y koja mora da važi.

Programiranje ograničenja

- Ograničenja mogu da budu različitih vrsta, na primer, ograničenja iskazne logike (A ili B je tačno), linearna ograničenja ($x \leq 15$), ograničenja nad konačnim domenima
- Rešavanje ograničenja vrši se različitim rešavačima, npr SAT i SMT
- Programiranje ograničenja ima primene pre svega u operacionim istraživanjima (tj u rešavanju kombinatornih i optimizacionih problema)

Programiranje ograničenja — primeri

- Pridružiti cifre slovima

```
 SEND
+MORE
 ----
MONEY
```

- Rasporediti kraljice na šahovskoj tabli
- Rešiti sistem nejednakosti, npr $x \in \{1, .., 100\}, y \in \{1, .., 100\}, x + y < 100, x > 10, y < 50$
- Odrediti položaje za najmanji broj predajnika tako da pokriva određeni prostor
- Definisati ponašanje semafora tako da protok saobraćaja bude najbolji

Programiranje ograničenja

- Podrška za ograničenja su ili ugrađena u programski jezik (npr Oz, Kaleidoscope) ili su data preko neke biblioteke.
- Postoje različite biblioteke za programiranje ograničenja za jezike C, C++, JAVA, Phyton, za .NET platformu, Ruby
- Neke od biblioteka su IBM ILOG CPLEX, Microsoft Z3
- Programiranje ograničenja je često raspoloživo u okviru sistema za logičko programiranje
- Programiranje ograničenja u logici — Constraint logic programing (CLP)

Programiranje ograničenja

- Programiranje ograničenja je nastalo u okviru logičkog programiranja (Prolog II, Jaffar i Lassez, 1987)
- Logičko programiranje i programiranje ograničenja imaju puno zajedničkih osobina
- Većina Prolog implementacija uključuje jednu ili više biblioteka za programiranje ograničenja
- B-Prolog, CHIP V5, Ciao, ECLiPSe, SICStus, GNU Prolog, Picat, SWI Prolog

4.1 B-Prolog

B-Prolog

- B-Prolog http://www.picat-lang.org/bprolog/
- B-Prolog je nastao 1994.
- B-Prolog je komercijalni proizvod, ali se može koristiti za učenje i istraživanja besplatno
- B-Prolog implementira ISO standard Prologa ali i razna proširenja
- B-Prolog ima bogat sistem za programiranje ograničenja

Programiranje ograničenja u B-Prologu

- B-Prolog podržava programiranje ograničenja na različitim domenima, npr na konačnom i iskaznom domenu.
- Programiranje ograničenja nad konačnim domenom sastoji se od tri dela
  1. Generisanje promenljivih i njihovih domena
  2. Generisanje ograničenja nad promenljivama
  3. Obeležavanje (labeling) — instanciranje promenljivih
- Za programiranje ograničenja uvode se novi predikati, koje ćemo razmotriti kroz primere.

Programiranje ograničenja u B-Prologu

Generisanje promenljivih i njihovih domena

- Definisanje domena: Vars in D, ili Vars :: D
- D se definiše kao Početak..Korak..Kraj
- Na primer 1..5..20 definiše domen 1,6,11,16
- Korak nije obavezan, podrazumeva se da je jedan. Na primer, 1..10 — svi brojevi od 1 do 10

Programiranje ograničenja u B-Prologu

- Postoje predikati za opšta ograničenja, a mogu se zadavati numerička ograničenja
- Opšta, npr alldifferent, ili alldistinct
- Numerička počinju sa #, npr #<, #<=...
- Labeling se odnosi na instanciranje promenljivih i na njihovo prikazivanje

4.2 Primeri

Primeri

Promenljive: X i Y, Domeni: X je bilo koji broj od 1 do 10, Y neparni brojevi od 5 do 30. Ograničenje: $X \geq Y$, Instanciranje: po X

```
 ?-  X in 1..10, Y in 5..2..30, X#>=Y, labeling(X).
X = 5
Y = 5 ?;
X = 6
Y = 5 ?;
X = 7
Y = _0490::[5,7] ?;
X = 8
Y = _0490::[5,7] ?;
X = 9
Y = _0490::[5,7,9] ?;
X = 10
Y = _0490::[5,7,9] ?;
no
```

Primeri

Promenljive: X i Y, Domeni: X je bilo koji broj od 1 do 10, Y neparni brojevi od 5 do 30. Ograničenje: $X \geq Y$, Instanciranje: po X i Y

```
 ?-  X in 1..10, Y in 5..2..30, X#>=Y, labeling(X), labeling(Y).
X = 5
Y = 5 ?;
X = 6
Y = 5 ?;
X = 7
Y = 5 ?;
X = 7
Y = 7 ?;
X = 8
Y = 5 ?;
...
```

Primeri

```
  SEND
 +MORE
  ----
 MONEY
```

Primeri

```
sendmoremoney(Vars) :- Vars = [S,E,N,D,M,O,R,Y], %generisanje promenljivih
    Vars :: 0..9,                                %definisanje domena
    S #\= 0,                                     %ogranicenja
    M #\= 0,
    all_different(Vars),
                 1000*S + 100*E + 10*N + D
    +            1000*M + 100*O + 10*R + E
    #= 10000*M + 1000*O + 100*N + 10*E + Y,
    labeling(Vars).                              %instanciranje
```

Primeri

```
| ?- consult(primer1).
| ?- sendmoremoney(Vars).
Vars = [9,5,6,7,1,0,8,2] ?;
no
```

Primeri

Ukrštenica:

```
|X1 X2 | |
|X3 X4 X5|
| | X6 X7|
```

Primeri

```
crossword(Vars):-
     Vars=[X1,X2,X3,X4,X5,X6,X7],
     Words2=[(0'I,0'N),
          (0'I,0'F),          %IF
          (0'A,0'S),          %AS
          (0'G,0'O),          %GO
          (0'T,0'O)],         %TO
     Words3=[(0'F,0'U,0'N),   %FUN
          (0'T,0'A,0'D),      %TAD
          (0'N,0'A,0'G),      %NAG
          (0'S,0'A,0'G)],     %SAG
     [(X1,X2),(X1,X3),(X5,X7),(X6,X7)] in Words2,
     [(X3,X4,X5),(X2,X4,X6)] in Words3,
     labeling(Vars),
     format("~s~n",[Vars]).
```

Primeri

```
| ?- consult(primer2).
| ?- crossword(Vars).
ASSAGGO
Vars = [65,83,83,65,71,71,79] ?;
INNAGGO
Vars = [73,78,78,65,71,71,79] ?;
no
```

Primeri

Raspoređivanje kraljica

```
queens(N):-
    length(Qs,N),
    Qs :: 1..N,
    foreach(I in 1..N-1, J in I+1..N,
            (Qs[I] #\= Qs[J],
             abs(Qs[I]-Qs[J]) #\= J-I)),
    labeling([ff],Qs),
    writeln(Qs).
```

Primeri

```
| ?- consult(primer3).
| ?- queens(5).
[1,3,5,2,4]

yes
| ?- queens(8).
[1,5,8,6,3,7,2,4]

yes
| ?- queens(10).
[1,3,6,9,7,10,4,2,5,8]

yes
| ?- queens(15).
[1,3,5,14,11,4,10,7,13,15,2,8,6,9,12]

yes
```

5 Literatura i pitanja

5.1 Literatura

Literatura — osnovna

- Concepts of programming languages, Robert W. Sebesta
- Programming Language Pragmatics, Michael L. Scott
- Skripta Veštačka inteligencija, Predrag Janičić, Mladen Nikolić
- Slajdovi prof Dušana Tošića sa istoimenog kursa
- B-Prolog http://www.picat-lang.org/bprolog/download/manual.pdf
- http://www.hakank.org/bprolog/

Literatura — dodatna

- Absys: the first logic programming language — A retrospective and a commentary http://www.sciencedirect.com/science/article/pii/0743106690900309
- The early years of logic programming — Robert Kowalski http://www.doc.ic.ac.uk/~rak/papers/the%20early%20years.pdf http://www.doc.ic.ac.uk/~rak/papers/History.pdf
- D. Tošić, R. Protić: PROLOG kroz primere, Tehnička knjiga, Beograd, 1991.
- B. Bajković, M.Đurišić, S.Matković: PROLOG i logika, Krug, Beograd, 1998.

5.2 Pitanja

Pitanja

- Šta čini teorijske osnove logičkog programiranja?
- Na koji način se rešavaju problemi u okviru logičke paradigme?
- Koji su osnovni predstavnici logičke paradigme?
- Za koju vrstu problema je pogodno koristiti logičko programiranje?
- Za koju vrstu problema nije pogodno koristiti logičko programiranje?

Pitanja

- Definisati logičke i nelogičke simbole logike prvog reda.
- Definisati term logike prvog reda.
- Definisati atomičku formulu logike prvog reda.
- Šta je literal? Šta je klauza?
- Definisati supstituciju za termove.
- Definisati supstituciju za atomičke formule.

Pitanja

- Ukoliko je zadata supstitucija $\sigma = ...$ i term $t = ...$ izračunati $t\sigma$
- Šta je problem unifikacije?
- Kada kažemo da su izrazi unifikabilni?
- Da li za dva izraza uvek postoji unifikator?
- Ukoliko za dva izraza postoji unifikator, da li on mora da bude jedinstven?
- Ukoliko su dati termovi $t_1 = ...$ i $t_2 = ...$ izračunati jedan unifikator ovih termova.
- Šta je metod rezolucije?

Pitanja

- Šta čini teorijske osnove logičkog programiranja?
- Na koji način se rešavaju problemi u okviru logičke paradigme?
- Koji su osnovni predstavnici logičke paradigme?
- Za koju vrstu problema je pogodno koristiti logičko programiranje?
- Za koju vrstu problema nije pogodno koristiti logičko programiranje?

Pitanja

- Šta je Hornova klauza i čemu ona odgovara?
- Šta je supstitucija?
- Kada se dva terma mogu unifikovati?
- Od čega se sastoji programiranje u Prologu?

Pitanja

- Šta su činjenice, šta se pomoću njih opisuje?
- Šta su pravila i šta se pomoću njih zadaje?
- Šta određuju činjenice i pravila?
- Šta govori pretpostavka o zatvorenosti?
- Šta su upiti i čemu oni služe?
- Šta su termovi?

Pitanja

- Kako se vrši unifikacija nad termima u Prologu?
- Šta je lista?
- Napisati pun i skraćen zapis liste od tri elementa a, b i c.
- Šta omogućava metaprogramiranje?
- Napisati deklarativno tumačenje naredne Prolog konstrukcije ...
- Napisati proceduralno tumačenje naredne Prolog konstrukcije ...

Pitanja

- Šta je stablo izvođenja i čemu ono odgovara u smislu deklarativne/proceduralne semantike?
- Koji su osnovni elementi stabla izvođenja?
- Nacrtati stablo izvođenja za naredni Prolog program ...
- Na koji način redosled tvrđenja u bazi znanja utiče na pronalaženje rešenja u Prologu?

Pitanja

- Koja je uloga operatora sečenja?
- Nacrtati stablo izvođenja za naredni program bez operatora sečenja, i sličan program sa operatorom sečenja ...
- Šta je crveni a šta zeleni operator sečenja?
- Koja je uloga opreatora sečenja u narednom primeru ...
- Da li se Hornovim klauzama mogu opisati sva tvrđenja logike prvog reda?
- Šta Prolog ne može da dokaže?
- Koje su osobine NOT operatora?

Pitanja

- Da li Prolog može da obezbedi generisanje efikasnih algoritama?
- Kakva je kompatibilnost između različitih Prolog kompajlera?
- Da li je Prolog Tjuring kompletan jezik?
- Kakav je sistem tipova u prologu?

Pitanja

- Šta je programiranje ograničenja?
- Koji su predstavnici paradigme programiranja ograničenja?
- Po čemu se razlikuje izraz $x < y$ u imperativnoj paradigmi i paradigmi ograničenja?
- Od čega se sastoji programiranje ograničenja nad konačnim domenom?
- Napisati program u B-Prologu koji pronalazi sve vrednosti promenljivih X, Y i Z za koje važi da je $X <= Y$ i $X + Y <= Z$ pri čemu promenljive pripadaju narednim domenima$X \in \{1, 2, ..., 50\}$, $Y \in \{5, 10, ..100\}$ i $Z \in \{1, 3, 5, ...99\}$