

1. Uvod u umjetnu inteligenciju

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

Sveučilište u Zagrebu
Fakultet elektrotehnike i računarstva

Ak. god. 2011/12.



Creative Commons Imenovanje–Nekomercijalno–Bez prerada 3.0

v1.1

Sadržaj

- 1 Mogu li strojevi misliti?
- 2 Mi i strojevi
- 3 Inteligencija i umjetna inteligencija
- 4 Turingov test
- 5 Povijest umjetne inteligencije



Mogu li strojevi misliti?

- Tisuće godina staro pitanje:
“Kako mi mislimo?”
- Zajedno s nastankom računala nastalo je i
uvjerenje da ćemo **inteligenciju reproducirati
računalom**
- Što je to uopće **inteligencija**? Što
podrazumijevamo pod pojmom **umjetna
inteligencija**?



I prije nastanka računala postojali su **pokušaji da napravimo svoju kopiju** . . .

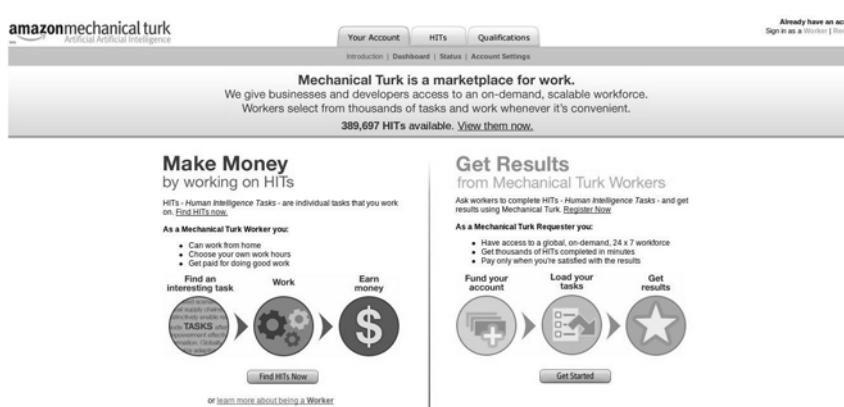
Povijesni pokušaji: Frankenstein

- Izvorna priča Mary Shelley
"Frankenstein, ili moderni Prometej", izdana 1818. godine, opisuje pokušaj znanstvenika Victora Frankensteina da stvori umjetan život



B. Wrightson: Frankenstein creates the fiend

Danas: Amazon Mechanical Turk

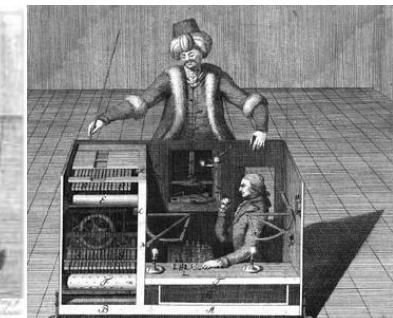
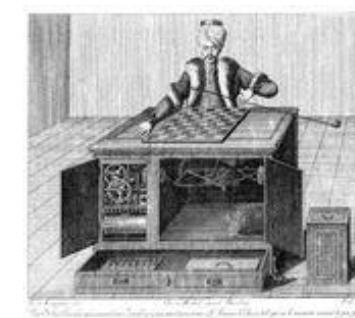


The screenshot shows the homepage of the Amazon Mechanical Turk website. At the top, there are navigation links for 'Your Account', 'HITS', and 'Qualifications'. Below that, there's a link to 'Sign in as a Worker' or 'Requester'. The main heading reads 'Mechanical Turk is a marketplace for work.' It explains that businesses and developers can access a scalable workforce where workers select from thousands of tasks. A statistic shows '389,697 HITs available. View them now.' On the left, there's a section titled 'Make Money by working on HITs' with icons for 'Find an interesting task', 'Work', and 'Earn money'. On the right, there's a section titled 'Get Results from Mechanical Turk Workers' with icons for 'Fund your account', 'Load your tasks', and 'Get results'. Both sections include 'Get Started' buttons.

- Velik broj ljudi plaćen da obavlja HIT-ove (*Human Intelligence Tasks*) – zadatke koji iziskuju ljudsku inteligenciju
- “Artificial Artificial Intelligence”, *crowdsourcing*

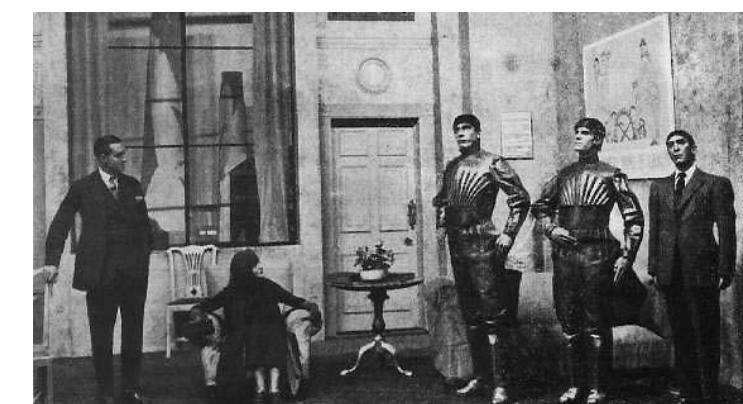
Povijesni pokušaji: Turčin

- 1770. godine Wolfgang von Kempelen konstruirao je automat koji igra šah i izvodi konjićev obilazak
- Automat je izlagan i demonstriran 80 godina po Europi i Americi
- Vješto konstruirana mehanička iluzionistička naprava



Povijesni pokušaji: Robot

- Godine 1921. češki pisac Karel Čapek napisao je dramu *R. U. R. (Rossum's Universal Robots)*
- Robot* (češki *robořeka*) – rad, prisilan rad



Isaac Asimov: "Ja, robot", 1942.

Three Robot Laws:

- ① A robot may not injure a human being or, through inaction, allow a human being to come to harm
- ② A robot must obey any orders given to it by human beings, except where such orders would conflict with the First Law
- ③ A robot must protect its own existence as long as such protection does not conflict with the First or Second Law



I, Robot (20th Century Fox, 2004)

Mogu li strojevi misliti?

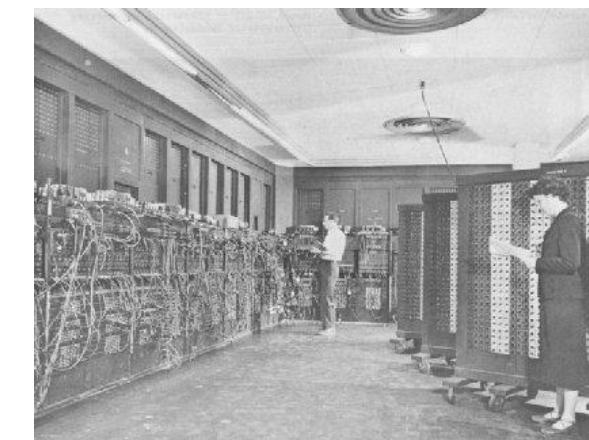
- Računala danas upravljaju vrlo složenim procesima, računalom simuliramo rješavanje složenih problema, donošenje odluka, zaključivanja, prirođan jezik, ...



Rodney Brooks i robot Cog, MIT Media Lab

Računala i električni mozak

- 1945. godine razvijen je ENIAC, prvo električno računalo
- U početku razvoja računala smatralo se da su računala istovjetna **električnim mozgu**



Što ćete naučiti na predmetu *Umjetna inteligencija*?

- Pregled temeljnih postupaka i algoritama UI
- Upoznat ćete sadašnja **ograničenja i mogućnosti** UI
- Upoznat ćete **prednosti i nedostatke** različitih metoda
- Moći ćete prepoznati probleme u kojima je pogodno primijeniti metode umjetne inteligencije
- Pregled različitih filozofskih pogleda na UI

1 Mogu li strojevi misliti?

2 Mi i strojevi

3 Inteligencija i umjetna inteligencija

4 Turingov test

5 Povijest umjetne inteligencije

Deep Blue vs. Garry Kasparov (1)

- 1997. godine računalo Deep Blue (IBM) pobijedilo je svjetskog šahovskog prvaka Garryja Kasparova
- Je li Deep Blue inteligentan?



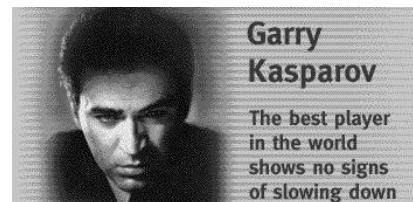
Deep Blue vs. Garry Kasparov (2)



200,000,000 šahovskih pozicija u sekundi

Posjeduje **мало znanja** o šahu, ali **ogromnu sposobnost** izračunavanja

Stroj **nema osjećaje niti intuiciju**, ne zaboravlja, ne može se zbuniti niti osjećati neugodno



3 šahovske pozicije u sekundi

Posjeduje **mnogo znanja** o šahu, ali bitno **manju sposobnost** izračunavanja

Ima osjećaje i **istančanu intuiciju**, ali može osjećati **umor i dosadu** te gubiti koncentraciju

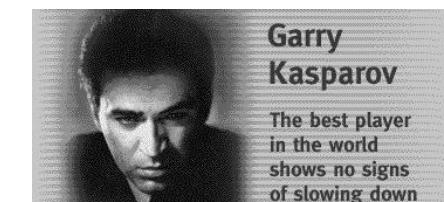
Pretraživanje velikog broja mogućih kombinacija ne zahtijeva inteligenciju!

Deep Blue vs. Garry Kasparov (3)



Deep Blue

This 1.4 ton
8-year-old sure
plays a mean
game of chess



**Garry
Kasparov**

The best player
in the world
shows no signs
of slowing down

Deep Blue **ne uči**, pa ne može iskoristiti umjetnu inteligenciju da bi naučio od svog protivnika

Deep Blue nevjerojatno učinkovito **rješava probleme iz domene šaha**, no manje je "inteligentan" čak i od malog djeteta.

Garry Kasparov može učiti i **brzo se prilagoditi** na temelju svojih uspjeha i pogrešaka.

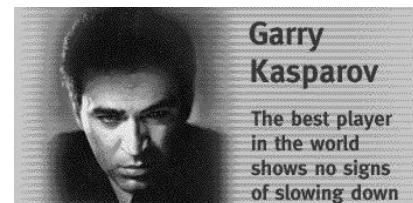
Garry Kasparov je općenito vrlo inteligentan. Autor je nekoliko knjiga i govori mnoge jezike.

Deep Blue vs. Garry Kasparov (4)



Deep Blue

This 1.4 ton
8-year-old super
computer plays a mean
game of chess



**Garry
Kasparov**

The best player
in the world
shows no signs
of slowing down

Izmjene u načinu igre mogu napraviti samo članovi razvojnog tima, i to tek nakon igre

Dok je Deep Blue vrlo vješt u procjeni šahovskih pozicija, no nije u stanju procijeniti slabosti svoga protivnika

Deep Blue mora provesti temeljito pretraživanje svih mogućih pozicija da bi odredio optimalni potez

Garry Kasparov u svakom trenutku **moe promijeniti** svoj način igre

Garry Kasparov je vješt u procjeni svoga protivnika, i u **iskorištavanju protivnikovih slabosti**

Garry Kasparov je sposoban **selektivno pretraživati** da bi odredio sljedeći potez

Pretraživanje prostora stanja

- Šah je razmjerno jednostavan – faktor grana je 35, a pravila igre stanu na jednu stranicu
- Znatno složenije: interpretacija rečenice prirodnog jezika.

Višeznačnost prirodnog jezika

John saw a boy and a girl with a red wagon with one blue and one white wheel dragging on the ground under a tree with huge branches.

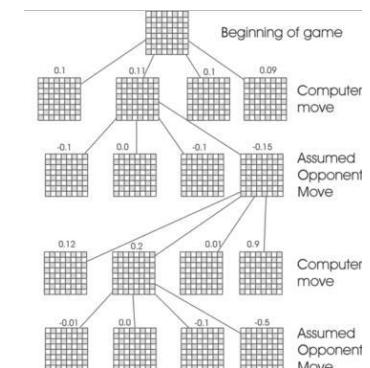
- Ova rečenica može imati 8064 tumačenja – višeznačna je i za ljude!

Pretraživanje prostora stanja

- Rješavanje ovakvih teških problema iziskuje **pretraživanje** velikog prostora stanja
- U prosjeku je moguće napraviti 35 različitih poteza
- Da bi se na majstorskoj razini igrao šah, potrebno je pretraživati **8 koraka unaprijed**, što znači da treba ispitati oko 35^8 ili $2 \cdot 10^{12}$ stanja

Kombinatorna eksplozija

Broj kombinacija **raste eksponencijalno** u svakom koraku ("netraktabilan problem")



Razumijevanje prirodnog jezika

- Flying planes can be dangerous.*
(*Flying planes is dangerous or Flying planes are dangerous*)
- The man tried to take a picture of a man with a turban.*
(Did the man try to take a picture with a turban, or take a picture of a man who is wearing a turban?)
- The man saw the boy with the telescope.*

Prevođenje i komunikacija prirodnim jezikom podrazumijeva **opće znanje** i **razumijevanje konteksta** pomoću kojih se razrješavaju višeznačnosti

Strojno prevodenje

- Strojno prevodenje jedan je od najtežih zadataka umjetne inteligencije, odnosno **obrade prirodnog jezika** kao njezine podgrane

I have a dream, that my four little children will one day live in a nation where they will not be judged by the color of their skin but by the content of their character. I have a dream today

– Martin Luther King

- Prijevod engleski → španjolski → engleski:

I am a sleepy, that my four small children a day of alive in a nation in where they will not be judged by the color of its skin but by the content of its character. I am a sleepy today.

Wolfram Alpha – Computational Knowledge Engine

The screenshot shows the Wolfram Alpha interface. At the top, there's a search bar with the question "How do you feel today?". Below it, a note says "Assuming 'How do you feel today' is a phrase | Use as a question about Alpha instead". Underneath, there's an "Input interpretation" section for the question "How are you?", followed by a "Result" section containing the text "I am doing well, thank you.". At the bottom left, it says "Computed by Wolfram Mathematica", and at the bottom right, there's a link "Download page".

IBM Watson – projekt DeepQA



- 16.2.2011.: Superračunalo **IBM Watson** pobijedilo je u igri Jeopardy najbolje ljudske natjecatelje i osvojilo \$35.734
- Napredni postupci **obrade prirodnog jezika, prikazivanje znanja, zaključivanja i pretraživanja informacija**

Zadatci teški za računalo, a jednostavni za čovjeka

- Razumijevanje prirodnog jezika
- Zdravorazumno zaključivanje (engl. *common sense reasoning*)
- Raspoznavanje uzoraka, razumijevanje slike i dinamičke scene
- Kretanje i navigacija
- Zadatci koju uključuju kreativnost
- ...

Činjenica je: **mi puno toga znamo.**

UI-potpuni problemi (*AI-complete problems*)

Računalni problemi čija je složenost istovjetna rješavanju središnjeg problema umjetne inteligencije: izgradnji stroja inteligentnog koliko i čovjek

Amazon Turk: tipični HIT-ovi

- Označiti objekte koji se nalaze na slici
- Iz skupa slika odabratи sliku koja najbolje predstavlja neki proizvod
- Provjeravati prikladnost sadržaja slike koje su *uploadali* korisnici
- Klasifikacija objekata u satelitskim snimkama
- Prevodenje rečenica s jednog jezika na drugi
- Ocjenjivanje relevantnosti rezultata tražilice za zadane upite
- Ocjenjivanje sličnosti zadanih parova riječi



StarCraft AI Competition (2010)



- Agent (inteligentan program koji samostalno djeluje u okolini) mora biti sposoban rješiti niz teških problema (**planiranje, optimizacija, višeagentsko upravljanje**) u ograničenom vremenu i s ograničenim resursima

Sadržaj

- 1 Mogu li strojevi misliti?
- 2 Mi i strojevi
- 3 Inteligencija i umjetna inteligencija
- 4 Turingov test
- 5 Povijest umjetne inteligencije

Što je inteligencija?

- lat. *intelligere* – razabirati, shvaćati, razumijevati
- Inteligencija je deskriptivan pojam – opisuje neka svojstva jedinke ili grupe jedinki
- Ne postoji suglasnost oko definicije inteligencije
- Većina definicija uključuje koncepte kao što su **apstraktno rasuđivanje, razumijevanje, samosvijest, komunikacija, učenje, planiranje i rješavanje problema**

- Inteligencija – svojstvo uspješnog snalaženja jedinke u novim situacijama (R. Pintner).
- Inteligencija – sposobnost učenja prilagodbe na okolinu (Colvin)
- Inteligencija – opća sposobnost apstraktnog zaključivanja pri rješavanju problema (Terman)
- Inteligencija – urođena opća kognitivna sposobnost (Burt)
- Inteligencija – svrshodno i prilagodljivo ponašanje u danim okolnostima (Psihologija, grupa autora, ŠK, Zagreb, 1992.)
- Inteligencija se manifestira u odnosu na neki posebni društveni i kulturni kontekst (J. Weizenbaum, 1975.)

Što je *umjetna* inteligencija?

- Grana računarske znanosti:
Tehničke znanosti → Računarstvo → Umjetna inteligencija
- Područja umjetne inteligencije (prema *Association of Computing Machinery*, ACM):
 - ① Opće područje (kognitivno modeliranje, filozofske osnove)
 - ② Ekspertni sustavi i primjene
 - ③ Automatsko programiranje
 - ④ Dedukcija i dokazivanje teorema
 - ⑤ Formalizmi i metode prikaza znanja
 - ⑥ Strojno učenje
 - ⑦ Razumijevanje i obrada prirodnih i umjetnih jezika
 - ⑧ Rješavanje problema, metode upravljanja i metode pretraživanja prostora stanja
 - ⑨ Robotika
 - ⑩ Računalni vid, raspoznavanje uzoraka i analiza scena
 - ⑪ Raspodijeljena umjetna inteligencija

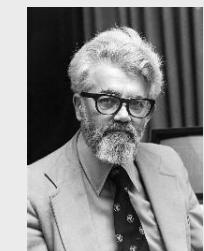
- Umjesto diskusije je li neko ponašanje intelligentno ili ne, možemo prihvatiti jedan pragmatičan pristup:

Ako je neko ponašanje interesantno (ljudsko, mrava, slona, robota...) – kako to ponašanje nastaje?
- Takav pristup omogućava razumijevanje principa na kojima se temelji inteligencija.
- “*understanding by building*”
- **Kognitivna znanost** – interdisciplinarno istraživanje uma

Naziv “umjetna inteligencija” (1)

Dartmouth Conference (Hanover, New Hampshire), 1956.

“... *The study is to proceed on the basis of the conjecture that every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it. An attempt will be made to find how to make machines use language, form abstractions and concepts, solve kinds of problems now reserved for humans, and improve themselves.*” (McCarthy et al. 1955)



“*We think that a significant advance can be made in one or more of these problems if a carefully selected group of scientists work on it together for a summer.*”

Naziv "umjetna inteligencija" (2)

- Znanstvenici s vodećih institucija: CMU, Stanford, MIT, IBM
- Dartmutska konferencija nije donijela spektakularne rezultate, ali je ustanovila novo područje – **umjetnu inteligenciju** – područje različito od operacijskih istraživanja ili teorije upravljanja, koja su prije toga bavila nastojala odgovoriti na slična pitanja

John McCarthy, (1956.)

"Umjetna inteligencija – naziv za znanstvenu disciplinu koja se bavi izgradnjom računalnih sustava čije se **ponašanje** može tumačiti kao intelligentno"

- Inteligenti strojevi vs. intelligentno ponašanje strojeva

Definicije umjetna inteligencije (2)

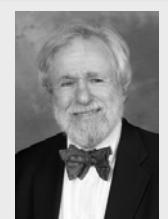
Elain Rich (University of Texas at Austin)

"Umjetna inteligencija bavi se izučavanjem kako računalo učiniti sposobnim da obavlja poslove koje u ovom času ljudi obavljaju bolje."



Eugene Charniak (Brown University)

"Proučavanje mentalnih svojstava kroz uporabu računalnih modela."



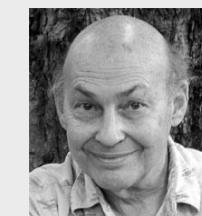
Definicije umjetne inteligencije (1)

Patrick. H. Winston (MIT)



"Proučavanje postupaka koji mogućim čine percipiranje, rasuđivanje i reagiranje."

Marvin Minsky (MIT)



"Znanost o tome kako postići da strojevi izvode zadatke koji bi, kada bi ih radio čovjek, iziskivali inteligenciju."

Klasifikacija definicija umjetne inteligencije (2)

Razmišljati ljudski

- "Uzbudljivi novi pokušaj da se omogući razmišljanje računalima... strojevi s umovima, u punom i doslovnom smislu." (Haugeland, 1985)
- "Automatizacija aktivnosti koje asociramo s ljudskim razmišljanjem, poput donošenja odluka, rješavanja problema, učenja..." (Bellman, 1978)

Ponašati se ljudski

- "Proces stvaranja strojeva koji obavljaju funkcije koje zahtijevaju inteligenciju koju imaju ljudi" (Kurzweil, 1990)
- "Proučavanje kako učiniti da računala rade stvari u kojima su, trenutno, ljudi bolji" (Rich i Knight, 1991)

Razmišljati racionalno

- "Proučavanje mentalnih svojstava kroz uporabu računalnih modela." (Charniak i McDermott, 1985)
- "Proučavanje postupaka koji mogućim čine percipiranje, rasuđivanje i reagiranje." (Winston, 1992)

Ponašati se racionalno

- "Polje rada koje želi objasniti i emulirati inteligentno ponašanje u smislu računalnih procesa" (Schalkoff, 1990)
- "Grana računarnih znanosti koja se bavi automatizacijom intelligentnog ponašanja" (Luger i Stubblefield, 1993)

Definicija umjetna inteligencija

D. W. Patterson (1990.)

"Umjetna inteligencija grana je računarske znanosti koja se bavi proučavanjem i oblikovanjem računarskih sustava koji pokazuju **neki oblik inteligencije**. Takvi sustavi mogu učiti, mogu donositi zaključke o svijetu koji ih okružuje, oni razumiju prirodni jezik te mogu spoznati i tumačiti složene vizualne scene te obavljati druge vrste vještina za koje se zahtijeva **čovjekov tip inteligencije**."

Umjetna inteligencija i druge znanosti

- Humanističke znanosti: **lingvistika, filozofija, psihologija**
- Prirodne znanosti: **matematika, biologija**
- **Kognitivna znanost**: interdisciplinarno znanstveno istraživanje uma (računarstvo, neuroznanost, psihologija, lingvistika, antropologija)



Kognitivna znanost – umjetna inteligencija

Kognitivna znanost

Inteligencija

Znanje

Spoznaja

Učenje

Usvajanje/razumijevanje jezika

Umjetna inteligencija

Umjetna inteligencija

Baza znanja
(knowledge base)

Obrada informacija
(information processing)

Strojno učenje
(machine learning)

Obrada prirodnog jezika

- 1 Mogu li strojevi misliti?
- 2 Mi i strojevi
- 3 Inteligencija i umjetna inteligencija
- 4 Turingov test
- 5 Povijest umjetne inteligencije

Ispitivanje inteligencije

- Alan Turing u u časopisu *Mind*, članku “*Computing Machinery and Intelligence*” (1950.) predložio je operacionalizaciju pitanja “mogu li strojevi razmišljati”
- Pitanje “**Mogu li strojevi razmišljati?**” zamjeniti pokusom “**igra oponašanja**”
- Pokus uspoređuje performanse prepostavljenog intelligentnog stroja i čovjeka na temelju nekog skupa upita



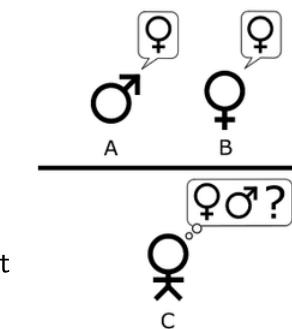
Alan Turing, *Can machines think?* (1950.)

“I believe that in about fifty years’ time it will be possible to program computers to make them play the imitation game so well that an average interrogator will not have more than 70 per cent chance of making the right identification after five minutes of questioning.”



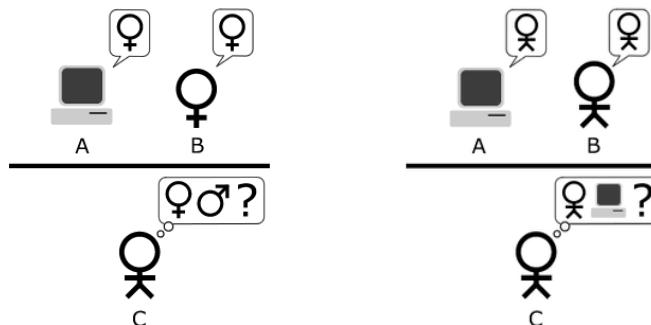
Turingov test (1)

- U igru su uključena tri sudionika s različitim ciljevima: A, B (ispitanici) i C (ispitivač). A i B su suprotnih spolova
- **Cilj igrača C:** postavljanjem pitanja odrediti spol ispitanika
- **Cilj igrača B:** pomoći ispitujućem C
- **Cilj igrača A:** navesti ispitujućem C na pogrešnu identifikaciju
- Pokus se ponavlja i mjeri se uspješnost ispitujućeg C



Turingov test (2)

- Što će se dogoditi ako stroj preuzme ulogu igrača B?
- Hoće li ispitivač C učiniti jednak broj pogrešaka kao kada u igri sudjeluju muškarac i žena?
- **Turing: Ako je broj pogrešaka jednak, onda je stroj intelligentan**
- Standardna varijanta: ispitivač C treba razabrati čovjeka od stroja



Nedostatci Turingovog testa

- Ljudska vs. opća inteligencija (ljudi se ponekad ponašaju neintelligentno, a intelligentno ponašanje ne mora nužno biti ljudsko)
- Prava vs. simulirana inteligencija (filozofski protuargument ponašajno-orientiranog UI)
- Naivnost ispitivača (dokazano u slučaju ELIZA-bota)
- Irrelevantnost testa

Irrelevantnost testa

Udžbenici aeronautike cilj aeronautike ne definiraju kao:

"Izgradnja strojeva koji lete tako slično golubovima da mogu prevariti druge golubove"



- TT je značajniji za filozofiju UI-a, nego što je to za sam razvoj UI-a

Turingov test (2)

- Q: Koje bi sposobnosti (intelligentan) stroj trebao imati, a da prođe TT?
- obrada prirodnog jezika
 - prikaz (predstavljanje) znanja
 - automatsko zaključivanje
 - učenje

Turing je predviđao da će do 2000. računala (s oko 120 MB memorije) imati 30% šanse zavarati ljude

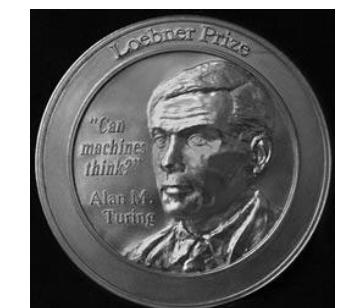
Q: U koju kategoriju spada definicija inteligencije prema TT?

Razmišljati ljudski	Razmišljati racionalno
Ponašati se ljudski	Ponašati se racionalno

A: Ponašati se ljudski

Loebnerova nagrada

- 1990. godine ustanovljena je **Loebnerova nagrada**: \$100,000 i zlatna medalja za prvi razgovorni program (*chatterbot*) čiji se odgovori ne budu razlikovali od čovjekovih
- Kontroverzno natjecanje upitne koristi za UI
- Glavna nagrada još uvijek nije osvojena
- Dobitnici nagrade za "program najsličniji čovjeku":
 - ▶ 2004. R. Wallace: **A.L.I.C.E.**
 - ▶ 2005. Rollo Carpenter: **George**
 - ▶ 2006. Rollo Carpenter: **Joan**
 - ▶ 2007. Robert Medeksza: **Ultra Hal**
 - ▶ 2008. Fred Roberts: **Elbot**
 - ▶ 2009. David Levy: **Do-Much-More**
 - ▶ 2010., 2011. Bruce Wilcox: **Suzette**



Obrnuti Turingov test: CAPTCHA

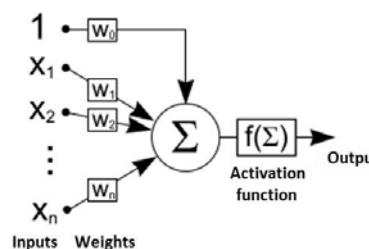
- CAPTCHA – Completely Automated Public Turing Test to Tell Computers and Humans Apart



- Istraživanje (provedeno pomoću Amazon Mechanical Turka) pokazuje da je CAPTCHA "često složenija no što bi trebala biti" – prosječno vrijeme rješavanja je 9,8 sekundi (Bursztein et al., 2010)

Početak: 1943. – 1952.

- 1943. J. McCulloch, W. Pitts: model umjetnog neurona
- 1949. D. Hebb: pravilo za modificiranje veze između dva neurona
- 1951. Minsky i Edmons: prva neuronska mreža od 40 neurona (vakumske cijevi)
- 1950. A. Turing: Turingov test, strojno učenje, genetički algoritmi, podržano učenje



Sadržaj

- 1 Mogu li strojevi misliti?
- 2 Mi i strojevi
- 3 Inteligencija i umjetna inteligencija
- 4 Turingov test
- 5 Povijest umjetne inteligencije

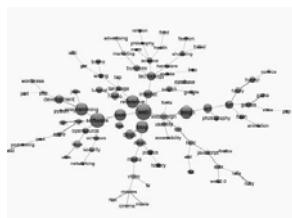
1952. – 1969. Rani entuzijazam, velika očekivanja (1)

- 1952. A. Samuel: igra dame, program koji uči
- 1956. Newell, Shaw i Simon: Logic Theorist (LT) – kraći dokaz teorema iz *Principia Mathematica*
- 1957. Newell & Simon: GPS, prvi program koji je utjelovio ljudski način razmišljanja
- 1958. J. McCarthy: LISP
- 1960.–1962. Widrow i Hoff: Adaline
- 1962. F. Rosenblatt: dokaz konvergencije perceptron-a



1952. – 1969. Rani entuzijazam, velika očekivanja (2)

- 1965. Joseph Weizenbaum – razgovorni program ELIZA
- 1965. Robinson – pravilo rezolucije
- 1966. Quillian – semantičke mreže
- 1969. Minsky, Papert: "Perceptrons" – ograničenje neuronske mreže



1952. – 1969. Otrežnjenje (1)

- Rezultati ranih sustava pokazali su se slabi na širem rasponu problema ili na težim zadacima
- Rani programi sadržavali su **malo ili ništa znanja**, a uspjeh se temeljio na jednostavnim sintaktičkim manipulacijama

Neuspjeh strojnog prevođenja (1957.)

Strojno prevođenje (financirano zbog ubrzavanja prevođenja ruskih radova o Sputniku) temeljilo se na sintaktičkim transformacijama i zamjenama riječi na temelju engleske i ruske gramatike. Rezultat:

"The spirit is willing but the flesh is weak"
→ "The votka is good but the flesh is rotten"



ELIZA – primjer razgovora

- Men are all alike
- In what ways?
- They are always bugging us about something or other
- Can you think of specific example?
- Well, may boyfriend made me come here
- Your boyfriend made you come here?
- He says I am depresed much of the time
- I am sorry that you are depressed
- Its true, I am unhappy
- Do you think coming here will help you not to be unhappy?
- I need some help that much seems certain
- What would it mean to you if you got some help?
- Perhaps I could learn to get along with my mother
- Tell me more about your family

1952. – 1969. Otrežnjenje (2)

- Drugi veliki problem – **netrakabilnost** mnogih problema koje je UI pokušavao riješiti
- Početni uspjeh bio je moguć zbog malog broja kombinacija u mikrosjetovima (pojednostavljeni problemi)
- Prije razvoja teorija izračunljivosti mislilo se da će se "skaliranje" na veće probleme riješiti povećanjem računalne snage!
- 1969. Minsky i Papert: *Perceptrons* – obeshrabrenje istraživanja neuronskih mreža

1970. – 1979. Sustavi temeljeni na znanju

- DENDRAL, Fiegenbaum, Buchanan (Stanford) – sustav temeljen na znanju zaključuje o molekularnim strukturama organskih spojeva na temelju spektroskopije masa – 450 pravila
- MYCIN, Shortliffe (Stanford), 550 pravila, različit od DENDRALA: nema teorijskog modela kao podlogu, uvođenje faktora izvjesnosti
- Napredak u obradi prirodnog jezika
- PROLOG – logički programski jezik popularan u Evropi
- 1975. Minsky, teorija okvira (frames)

1980. – danas

- 1980 – UI je industrija! (od nekoliko miliona dolara u 1980. do milijardu dolara u 1988.)
- 1982 McDermott – DEC R1 ekspertni sustav
- 1980-ih – Povratak neuronskih mreža (Werbos – algoritam *backpropagation*)
- Inteligentni agenti (agent – percepcija okoline kroz senzore i djelovanje na sredinu kroz akcije)
- Robotika
- Strojno učenje

Sažetak

- Nastojanja da se napravi inteligentan stroj sežu daleko u prošlost
- Mnogi zadatci koji su jednostavniji za čovjeka, teški su za računalo. Jako teške zadatke nazivamo **UI-potpuni zadatci**
- Ne postoji opće prihvaćena definicija UI, no postoje četiri osnovna tipa definicija (**ponašanje/razmišljanje, racionalno/ljudski**)
- **Turingov test** mjeri inteligenciju stroja kroz igru oponašanja. Test je interesantan, ali u praksi nije toliko značajan
- Kroz povijest, UI je imala uspone i padove. Rana ekstravagantna obećanja uglavnom nisu ispunjena
- Računala danas mogu uspješno (i bolje od ljudi) **rješavati mnoge specifične probleme**



Sljedeća tema: Pretraživanje prostora stanja

2. Pretraživanje prostora stanja

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

Sveučilište u Zagrebu
Fakultet elektrotehnike i računarstva

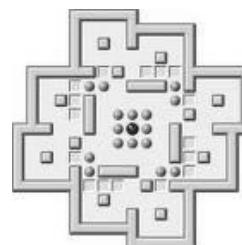
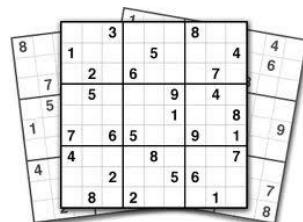
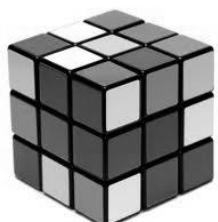
Ak. god. 2011/12.



Creative Commons Imenovanje–Nekomercijalno–Bez prerada 3.0

v3.0

Tipični problemi...



Motivacija

- Mnogo se analitičkih problema može riješiti pretraživanjem prostora stanja
- Krenuvši od **početnog stanja** problema, pokušavamo pronaći **ciljno stanje**
- Slijed akcija koje nas vode do ciljnog stanja predstavljaju rješenje problema
- Problem predstavlja velik broj stanja te velik broj mogućih izbora
- Pretraživanje zato mora biti sustavno



Formalan opis problema

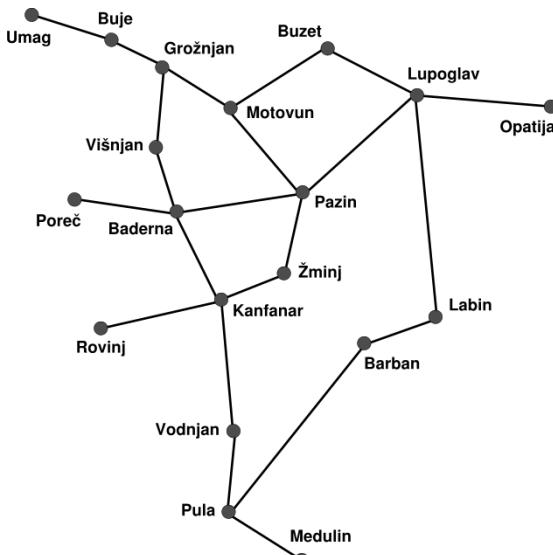
- Neka je S skup stanja (prostor stanja)
- Problem se sastoji od početnog stanja, prijelaza između stanja i ciljnog (ciljnih) stanja

Problem pretraživanja

$problem = (s_0, \text{succ}, \text{goal})$

- ① $s_0 \in S$ je **početno stanje**
 - ② $\text{succ} : S \rightarrow \wp(S)$ je **funkcija sljedbenika** koja definira prijelaze između stanja
 - ③ $\text{goal} : S \rightarrow \{\top, \perp\}$ je **ispitni predikat** istinit samo za ciljna stanja
- Funkcija sljedbenika može se definirati implicitno pomoću skupa **operatora** (različitim operatorima prelazi se u različita stanja)

Primjer: Putovanje kroz Istru



Kako iz Pule do Buzeta?

$$problem = (s_0, \text{succ}, \text{goal})$$

$$s_0 = Pula$$

$$\text{succ}(Pula) = \{Barban, Medulin, Vodnjan\}$$

$$\text{succ}(Vodnjan) = \{Kanfanar, Pula\}$$

:

$$\text{goal}(Buzet) = \top$$

$$\text{goal}(Motovun) = \perp$$

$$\text{goal}(Pula) = \perp$$

:

Zašto Buzet?



Divovska fritada s tartufima

Primjer: Slagalica 3×3

početno stanje:

8		7
6	5	4
3	2	1

ciljno stanje:

1	2	3
4	5	6
7	8	

Koji potezi vode do rješenja?

$$problem = (s_0, \text{succ}, \text{goal})$$

$$s_0 = \begin{bmatrix} 8 & & 7 \\ 6 & 5 & 4 \\ 3 & 2 & 1 \end{bmatrix}$$

$$\text{succ}\left(\begin{bmatrix} 8 & & 7 \\ 6 & 5 & 4 \\ 3 & 2 & 1 \end{bmatrix}\right) = \left\{\begin{bmatrix} 8 & 7 & \\ 6 & 5 & 4 \\ 3 & 2 & 1 \end{bmatrix}, \begin{bmatrix} 8 & 7 & \\ 6 & 5 & 4 \\ 3 & 2 & 1 \end{bmatrix}, \begin{bmatrix} 8 & 7 & \\ 6 & 5 & 4 \\ 3 & 2 & 1 \end{bmatrix}\right\}$$

:

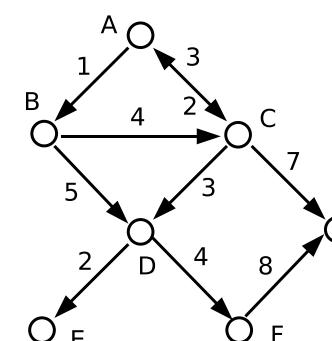
$$\text{goal}\left(\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & \end{bmatrix}\right) = \top$$

$$\text{goal}\left(\begin{bmatrix} 8 & 7 & \\ 6 & 5 & 4 \\ 3 & 2 & 1 \end{bmatrix}\right) = \perp$$

$$\text{goal}\left(\begin{bmatrix} 8 & 7 & \\ 6 & 5 & 4 \\ 3 & 2 & 1 \end{bmatrix}\right) = \perp$$

:

Ideja pretraživanja



- Prostor pretraživanje prostora stanja svodi se na pretraživanje **usmjerenog grafa** (digrafa)
- Vrhovi grafa = stanja
lukovi = prijelazi između stanja
- Graf može biti zadan eksplicitno ili implicitno
- Graf može imati cikluse
- Ako definiramo cijene prijelaza, onda je to **usmjeren težinski graf** (težinski digraf)

Stablo pretraživanja

- Pretraživanjem usmjerenog grafa postepeno gradimo **stablo pretraživanja**
- Stablo gradimo tako da pojedine čvorove **proširujemo**: pomoću funkcije sljedbenika (odnosno operatora) generiramo sve sljedbenike nekog čvora
- Otvoreni čvorovi ili fronta**: čvorovi koji su generirani, ali još nisu prošireni
- Zatvoreni čvorovi**: čvorovi koji su već prošireni
- Redoslijed kojim proširujemo čvorove određuje **strategiju pretraživanja**

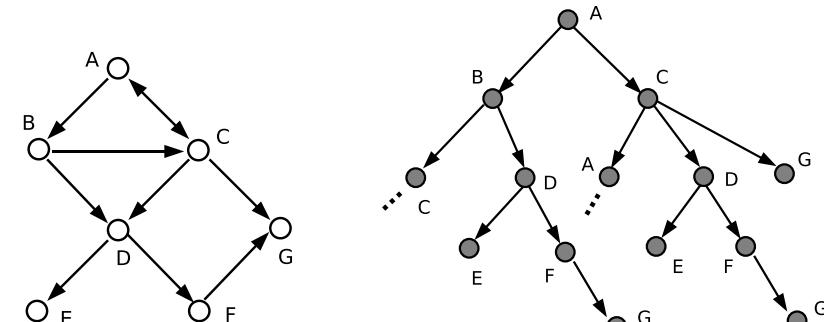
Dalbelo Bašić, Šnajder (UNIZG FER)

UI – Pretraživanje prostora stanja

Ak. god. 2011/12.

9 / 51

Prostor stanja vs. stablo pretraživanja



- Stablo pretraživanja **nastaje** pretraživanjem prostora stanja
 - Stablo pretraživanja može biti beskonačno čak i onda kada je prostor stanja konačan
- NB:** prostor stanja ima cikluse \Rightarrow stablo pretraživanja je beskonačno

Stanje vs. čvor

- Čvor n je podatkovna struktura koja sačinjava stablo pretraživanja
- Čvor pohranjuje stanje**, ali i još neke dodatne podatke:

Podatkovna struktura čvora

$$n = (s, d)$$

s – stanje

d – dubina čvora u stablu

$$\text{state}(n) = s, \text{ depth}(n) = d$$

$$\text{initial}(s) = (s, 0)$$

Opći algoritam pretraživanja

Opći algoritam pretraživanja

```
function search( $s_0$ , succ, goal)
    open  $\leftarrow$  [initial( $s_0$ )]
    while open  $\neq []$  do
         $n \leftarrow \text{removeHead}(open)$ 
        if goal(state( $n$ )) then return  $n$ 
        for  $m \in \text{expand}(n, \text{succ})$  do
            insert( $m$ , open)
    return fail
```

- $\text{removeHead}(l)$ – skida prvi element neprazne liste l
- $\text{expand}(n, \text{succ})$ – proširuje čvor n uporabom funkcije sljedbenika succ
- $\text{insert}(n, l)$ – umeće čvor n u listu l

Q: Je li ovaj algoritam determinističan?

Dalbelo Bašić, Šnajder (UNIZG FER)

UI – Pretraživanje prostora stanja

Ak. god. 2011/12.

11 / 51

Dalbelo Bašić, Šnajder (UNIZG FER)

UI – Pretraživanje prostora stanja

Ak. god. 2011/12.

12 / 51

Proširivanje čvora

- Proširivanje čvora treba ažurirati sve komponente čvora:

Proširivanje čvora

```
function expand( $n$ , succ)
    return { ( $s$ , depth( $n$ ) + 1) |  $s \in \text{succ}(\text{state}(n))$  }
```

- Funkcija će biti složenija kada u čvor budemo pohranjivali dodatne podatke (npr. pokazivač na roditeljski čvor)

Strategije pretraživanja

Dvije osnovne vrste strategija pretraživanja:

- **Slijepo pretraživanje** (engl. *blind, uninformed search*)
- **Usmjereni pretraživanje** (engl. *directed, informed, heuristic search*)

Danas govorimo samo o slijepom pretraživanju.

Usporedba problema i algoritama

Karakteristike problema:

- $|S|$ – **broj stanja**
- b – **faktor grananja** stabla pretraživanja
- d – **dubina optimalnog rješenja** u stablu pretraživanja
- m – **maksimalna dubina** stabla pretraživanja (moguće ∞)

Svojstva algoritama:

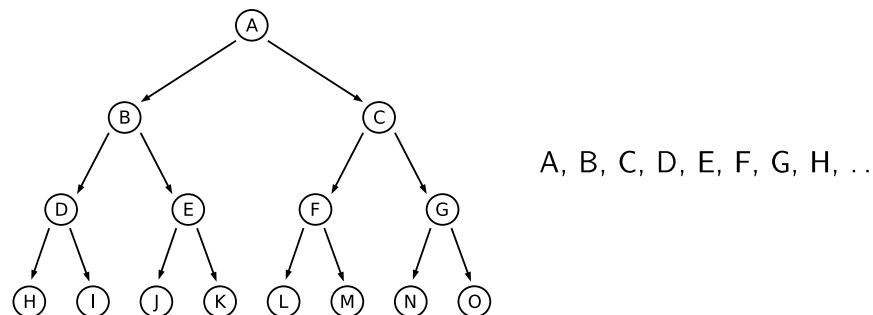
- ① **Potpunost** (engl. *completeness*) – algoritam je potpun akko pronađe rješenje uvijek kada ono postoji
- ② **Optimalnost** (engl. *optimality, admissibility*) – algoritam je optimalan akko pronađe optimalno rješenje (ono s najmanjom cijenom)
- ③ **Vremenska složenost** (broj generiranih čvorova)
- ④ **Prostorna složenost** (broj pohranjenih čvorova)

Slijepo pretraživanje

- ① Pretraživanje u širinu (engl. *breadth-first search, BFS*)
- ② Pretraživanje s jednolikom cijenom (engl. *uniform-cost search*)
- ③ Pretraživanje u dubinu (engl. *depth-first search, DFS*)
- ④ Ograničeno pretraživanje u dubinu
- ⑤ Iterativno pretraživanje u dubinu
- ⑥ Dvosmjerno pretraživanje

Pretraživanje u širinu

- Jednostavna slijepa strategija pretraživanja
- Nakon proširenja korijenskog čvora, proširuju se sva njegova djeca, zatim sva njihova djeca, itd.
- Općenito, čvorovi na dubini d proširuju se tek nakon što se prošire svi čvorovi na razini $d - 1$, tj. pretražujemo **razinu po razinu**



Pretraživanje u širinu – primjer izvođenja

- ① $open = [(Pula, 0)]$
- ② $\text{expand}(Pula, 0) = \{(Vodnjan, 1), (Barban, 1), (Medulin, 1)\}$
 $open = [(Vodnjan, 1), (Barban, 1), (Medulin, 1)]$
- ③ $\text{expand}(Vodnjan, 1) = \{(Kanfanar, 2), (Pula, 2)\}$
 $open = [(Barban, 1), (Medulin, 1), (Kanfanar, 2), (Pula, 2)]$
- ④ $\text{expand}(Barban, 1) = \{(Labin, 2), (Pula, 2)\}$
 $open = [(Medulin, 1), (Kanfanar, 2), (Pula, 2), (Labin, 2), (Pula, 2)]$
- ⑤ $\text{expand}(Medulin, 1) = \{(Pula, 2)\}$
 $open = [(Kanfanar, 2), (Pula, 2), (Labin, 2), (Pula, 2), (Pula, 2)]$
- ⑥ $\text{expand}(Kanfanar, 2) = \{(Baderna, 3), (Rovinj, 3), (Vodnjan, 3), (Zminj, 3)\}$
 $open = [(Pula, 2), (Labin, 2), (Pula, 2), (Pula, 2), (Baderna, 3), \dots]$
- ⋮

Pretraživanje u širinu – izvedba

- Ovakvu strategiju ostvariti ćemo ako proširene čvorove uvijek **dodajemo na kraj** liste otvorenih čvorova

Pretraživanje u širinu

```
function breadthFirstSearch(s0, succ, goal)
    open ← [initial(s0)]
    while open ≠ []
        n ← removeHead(open)
        if goal(state(n)) then return n
        for m ∈ expand(n, succ) do
            insertBack(m, open)
    return fail
```

- Lista otvorenih čvorova zapravo je **red** (engl. *queue*)

Pretraživanje u širinu – svojstva

- Pretraživanje u širinu je **potpuno i optimalno**
- U svakom koraku proširuje se najplići čvor, pa je strategija optimalna (uz pretpostavku da je cijena prijelaza konstantna)
- **Vremenska složenost:**
$$1 + b + b^2 + b^3 + \dots + b^d + (b^{d+1} - b) = \mathcal{O}(b^{d+1})$$
(na zadnjoj razini generiraju se sljedbenici svih čvorova osim ciljnog)
- **Prostorna složenost:** $\mathcal{O}(b^{d+1})$
- Eksponencijalna složenost (pogotovo prostorna) glavni je nedostatak pretraživanja u širinu
- Npr. $b = 4, d = 16, 10 \text{ B/čvor} \rightarrow 43 \text{ GB}$
- Primjenjivo samo na male probleme

Podsjetnik: Asimptotska složenost algoritma

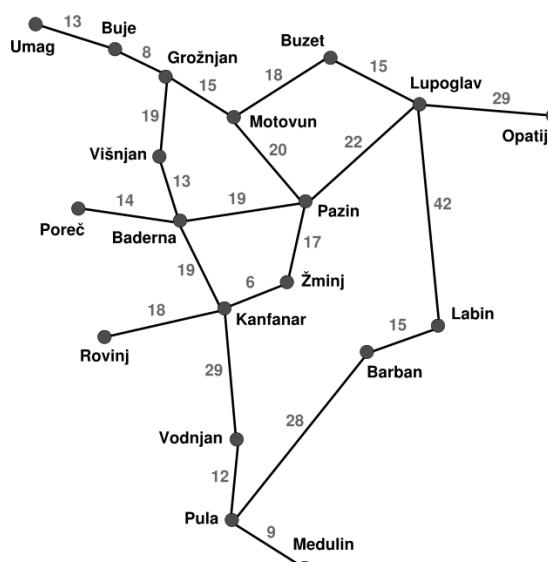
- Asimptotska složenost funkcije: ponašanje funkcije $f(n)$ kada $n \rightarrow \infty$ izražena pomoću jednostavnijih funkcija

Notacija "Veliko-O"

$$\mathcal{O}(g(n)) = \{f(n) \mid \exists c, n_0 \geq 0 \text{ takvi da } \forall n \geq n_0. 0 \leq f(n) \leq c \cdot g(n)\}$$

- Konvencija: umjesto $f(n) \in \mathcal{O}(g(n))$ pišemo $f(n) = \mathcal{O}(g(n))$
- Gornja ograda složenosti (složenost u najgorem slučaju)
- Donja ograda nije definirana, pa npr. $n = \mathcal{O}(n)$, $n = \mathcal{O}(n^2), \dots$ (u principu nas zanima ona najmanja ograda)
- $\Theta(g(n))$ definira i gornju i donju ogradu (engl. *tight bounds*)

Primjer: Putovanje kroz Istru



Kako iz Pule do Buzeta?

problem = (s_0 , succ, goal)

$s_0 = Pula$

succ(Pula) =
{(Barban, 28), (Medulin, 9),
(Vodnjan, 12)}

succ(Vodnjan) =
{(Kanfanar, 29), (Pula, 12)}

:

goal(Buzet) = \top

goal(Motovun) = \perp

goal(Pula) = \perp

:

Cijene prijelaza

- Ako operacije (prijelazi između stanja) nisu jednake cijene, funkciju sljedećeg stanja modificiramo tako da ona za svakog sljedbenika vraća i cijenu prijelaza:

succ : $S \rightarrow \wp(S \times \mathbb{R}^+)$

- U čvoru više ne pohranjujem dubinu nego ukupnu cijenu puta do tog čvora:

$n = (s, c)$, $g(n) = c$

- Funkciju proširenja čvora moramo također modificirati tako da ažurira cijenu puta do čvora:

```
function expand( $n$ , succ)  
    return { $(s, g(n) + c) \mid (s, c) \in \text{succ(state}(n))\}$ 
```

Pretraživanje s jednolikom cijenom

- Kao i pretraživanje u širinu, no u obzir uzimamo cijenu prijelaza

Pretraživanje s jednolikom cijenom

```
function uniformCostSearch( $s_0$ , succ, goal)  
    open  $\leftarrow$  [initial( $s_0$ )]  
    while open  $\neq []$  do  
         $n \leftarrow \text{removeHead}(open)$   
        if goal(state( $n$ )) then return  $n$   
        for  $m \in \text{expand}(n, \text{succ})$  do  
            insertSortedBy( $g, m, open$ )  
    return fail
```

- insertSortedBy(f, n, l) – umeće čvor n u listu l sortiranu uzlazno prema vrijednosti $f(n)$
- Lista $open$ funkcioniра kao **prioritetni red**

Pretraživanje s jednolikom cijenom – primjer izvođenja

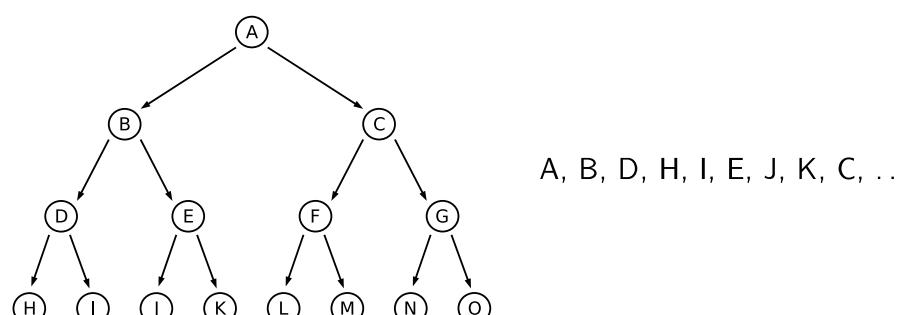
- ① $open = [(Pula, 0)]$
- ② $\text{expand}(Pula, 0) = \{(Vodnjan, 12), (Barban, 28), (Medulin, 9)\}$
 $open = [(Medulin, 9), (Vodnjan, 12), (Barban, 28)]$
- ③ $\text{expand}(Medulin, 9) = \{(Pula, 18)\}$
 $open = [(Vodnjan, 12), (Pula, 18), (Barban, 28)]$
- ④ $\text{expand}(Vodnjan, 12) = \{(Kanfanar, 41), (Pula, 24)\}$
 $open = [(Pula, 18), (Pula, 24), (Barban, 28), (Kanfanar, 41)]$
- ⑤ $\text{expand}(Pula, 18) = \{(Vodnjan, 30), (Barban, 46), (Medulin, 27)\}$
 $open = [(Pula, 24), (Medulin, 27), (Barban, 28), (Vodnjan, 30), \dots]$
- ⋮

Q: Hoće li ovaj algoritam pronaći rješenje (Buzet)?

Q: Hoće li pronaći najkraći put do Buzeta?

Pretraživanje u dubinu

- Pretraživanje u dubinu uvijek prvo proširuje najdublji čvor u stablu pretraživanja
- Postupak se vraća na pliće razine tek kada dosegne listove (stanja koja nemaju sljedbenika)



Pretraživanje s jednolikom cijenom – svojstva

- Algoritam je **potpun i optimalan**
- Ako je C^* optimalna cijena do cilja, a ε minimalna cijena prijelaza, dubina stabla do ciljnog čvora je $d = \lfloor C^*/\varepsilon \rfloor$
- Prostorna i vremenska složenost: $\mathcal{O}(b^{1+\lfloor C^*/\varepsilon \rfloor})$

Pretraživanje u dubinu – izvedba

- Strategiju pretraživanja u dubinu ostvarit ćemo ako proširene čvorove **dodajemo na početak** liste $open$

Pretraživanje u dubinu

```
function depthFirstSearch(s0, succ, goal)  
    open ← [initial(s0)]  
    while open ≠ [] do  
        n ← removeHead(open)  
        if goal(state(n)) then return n  
        for m ∈ expand(n, succ) do  
            insertFront(m, open)  
    return fail
```

- Lista otvorenih čvorova zapravo je **stog**

Pretraživanje u dubinu – primjer izvođenja

- ① $open = [(Pula, 0)]$
- ② $\text{expand}(Pula, 0) = \{(Vodnjan, 1), (Barban, 1), (Medulin, 1)\}$
 $open = [(Vodnjan, 1), (Barban, 1), (Medulin, 1)]$
- ③ $\text{expand}(Vodnjan, 1) = \{(Kanfanar, 2), (Pula, 2)\}$
 $open = [(Kanfanar, 2), (Pula, 2), (Barban, 1), (Medulin, 1)]$
- ④ $\text{expand}(Kanfanar, 2) =$
 $\{(Baderna, 3), (Rovinj, 3), (Vodnjan, 3), (Zminj, 3)\}$
 $open = [(Baderna, 3), (Rovinj, 3), (Vodnjan, 3), (Zminj, 3), (Pula, 2), \dots]$
- ⑤ $\text{expand}(Baderna, 3) = \{(Porec, 4), (Visnjan, 4), (Pazin, 4), (Kanfanar, 4)\}$
 $open =$
 $[(Porec, 4), (Visnjan, 4), (Pazin, 4), (Kanfanar, 4), (Baderna, 3), \dots]$
- ⋮

Q: Je li ovo jedini mogući tijek izvođenja?

Pretraživanje u dubinu – rekurzivna izvedba

- Listu $open$ možemo izbjegići:

Pretraživanje u dubinu (rekurzivna izvedba)

```
function depthFirstSearch(s, succ, goal)
  if goal(s) then return s
  for m ∈ succ(s) do
    r ← depthFirstSearch(m, succ, goal)
    if r ≠ fail then return r
  return fail
```

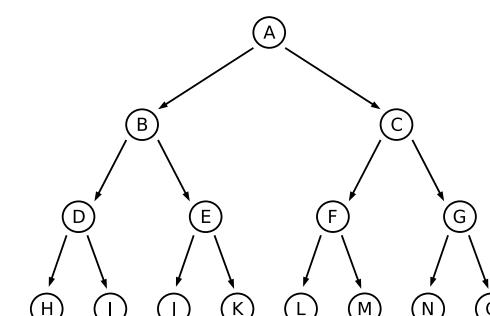
- Umjesto eksplisitne liste $open$ koristi se sistemski stog
- Prostorna složenost je $\mathcal{O}(m)$

Pretraživanje u dubinu – svojstva

- Pretraživanje u dubinu manje je memorijski zahtjevno
- **Prostorna složenost:** $\mathcal{O}(bm)$, gdje je m maksimalna dubina stabla
- **Vremenska složenost:** $\mathcal{O}(b^m)$
(nepovoljno, ako $m \gg d$)
- **Potpunost:** ne, jer može zaglaviti u beskonačnoj petlji
- **Optimalnost:** ne, jer ne pretražuje razinu po razinu
- Pretraživanje u dubinu treba izbjegavati kod stabla pretraživanja čija je maksimalna dubina velika ili beskonačna

Ograničeno pretraživanje u dubinu

- Pretražuje u dubinu, ali ne dublje od zadane granice



$k = 0: A$
 $k = 1: A, B, C$
 $k = 2: A, B, D, E, C, F, G$

Ograničeno pretraživanje u dubinu – izvedba

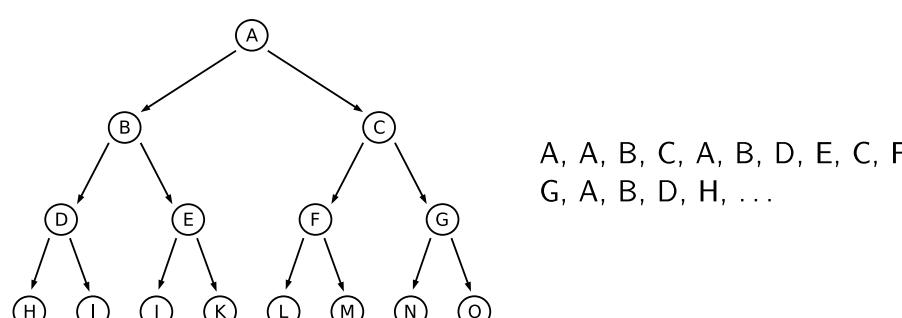
- Čvor proširujemo samo ako se u stablu pretraživanja nalazi iznad dubinskog ograničenja k :

Ograničeno pretraživanje u dubinu

```
function depthLimitedSearch(s0, succ, goal, k)
    open ← [initial(s0)]
    while open ≠ [] do
        n ← removeHead(open)
        if goal(state(n)) then return s
        if depth(n) < k then
            for m ∈ expand(n, succ) do
                insertFront(m, open)
    return fail
```

Iterativno pretraživanje u dubinu

- Izbjegava problem izbora optimalne dubinske granice isprobavajući sve moguće vrijednosti krenuvši od dubine 0
- Kombinira prednosti pretraživanja u dubinu i pretraživanja u širinu



Ograničeno pretraživanje u dubinu – svojstva

- **Prostorna složenost:** $\mathcal{O}(bk)$, gdje je k dubinska granica
- **Vremenska složenost:** $\mathcal{O}(b^k)$
- **Potpunost:** da, ali samo ako $d \leq k$
- **Optimalnost:** ne, jer ne pretražuje razinu po razinu
- Algoritam je uporabiv ako znamo dubinu rješenja d (možemo postaviti $k = |S|$)

Iterativno pretraživanje u dubinu – izvedba

Iterativno pretraživanje u dubinu

```
function iterativeDeepeningSearch(s0, succ, goal)
    for k ← 0 to ∞ do
        result ← depthLimitedSearch(s0, succ, goal, k)
        if result ≠ fail then return result
    return fail
```

Iterativno pretraživanje u dubinu – svojstva

- Strategija se na prvi pogled čini neučinkovitom: više puta proširujemo iste čvorove
- U većini slučajeva to ne predstavlja problem: većina čvorova stabla nalazi se na dubljim razinama, pa ponavljanje proširivanja čvorova na višim razinama nije problematično
- **Vremenska složenost:** $\mathcal{O}(b^d)$
- **Prostorna složenost:** $\mathcal{O}(bd)$
- **Potpunost:** da, jer koristi dubinsko ograničenje
- **Optimalnost:** da, jer pretražuje razinu po razinu
- Iterativno pretraživanje u dubinu preporučena je strategija za probleme s velikim prostorom stanja i nepoznatom dubinom rješenja

Dvosmjerno pretraživanje

- Istovremeno se pretražuje od početnog stanja prema ciljnog stanju i od ciljnog stanja prema početnom stanju
- Pretraživanje se zaustavlja najkasnije onda kada se dvije fronte susretnu na polovici puta
- Npr. ako se u oba smjera koristi pretraživanje u širinu, onda su prostorna i vremenska složenost $\mathcal{O}(2b^{d/2}) = \mathcal{O}(b^{d/2})$
Značajna ušteda!
- Nedostatak: postupak je primjenjiv samo ako problem (1) ima malen broj eksplisitno definiranih ciljnih stanja i (2) svi operatori imaju inverze

Iterativno pretraživanje u dubinu – složenost

- Broj proširenih čvorova kod pretraživanja u širinu je

$$1 + b + b^2 + \cdots + b^{d-2} + b^{d-1} + b^d + (b^{d+1} - b)$$

pa je asimptotska vremenska složenost $\mathcal{O}(b^{d+1})$

- Broj proširenih čvorova kod iterativnog pretraživanja je

$$(d+1)1 + db + (d-1)b^2 + \cdots + 3b^{d-2} + 2b^{d-1} + 1b^d$$

pa je asimptotska vremenska složenost $\mathcal{O}(b^d)$

- Razlika je to manja što je veći faktor grananja

- Npr. za $b = 2$ to je 100% više čvorova

za $b = 3$ to je 50% više čvorova

za $b = 10$ to je 11% više čvorova

Usporedba algoritama slijepog pretraživanja

Algoritam	Vrijeme	Prostor	Potpunost	Opt.
U širinu	$\mathcal{O}(b^{d+1})$	$\mathcal{O}(b^{d+1})$	Da	Da
Jednol. cijena	$\mathcal{O}(b^{1+\lfloor C^*/\epsilon \rfloor})$	$\mathcal{O}(b^{1+\lfloor C^*/\epsilon \rfloor})$	Da	Da
U dubinu	$\mathcal{O}(b^m)$	$\mathcal{O}(bm)$	Ne	Ne
Ogr. u dubinu	$\mathcal{O}(b^k)$	$\mathcal{O}(bk)$	Da, ako $d \leq k$	Ne
Iter. u dubinu	$\mathcal{O}(b^d)$	$\mathcal{O}(bd)$	Da	Da
Dvosmjerno	$\mathcal{O}(b^{d/2})$	$\mathcal{O}(b^{d/2})$	Da	Da

b – faktor grananja, d – dubina optimalnog rješenja,

m – maksimalna dubina stabla ($m \geq d$), k – dubinsko ograničenje

- Svi su algoritmi **eksponencijalne vremenske složenosti!**

- Pretraživanje u dubinu (i njegove varijante) bolje su prostorne složenosti od pretraživanja u širinu

Rekonstrukcija rješenja

- U strukturi čvora moramo pamtit i pokazivač na roditeljski čvor:

$$n = (s, d, p), \text{ parent}(n) = p$$

```
function expand(n, succ)
    return { (s, depth(n) + 1, n) | s ∈ succ(state(n)) }
```

- Krećemo od ciljnog čvora i pratimo pokazivače unazad:

Rekonstrukcija puta do čvora

```
function path(n)
    p ← parent(n)
    if p = null then return [state(n)]
    return insertBack(state(n), path(p))
```

- Vremenska složenost je $\mathcal{O}(d)$

- **NB:** Zatvorene čvorove očito moramo **čuvati u memoriji!**

Problem ponavljanja stanja (2)

- **Rješenje 2:** spriječiti nastanak puteva s ciklusima

Opći algoritam pretraživanja (nadopuna)

```
function search(s0, succ, goal)
    open ← [initial(s0)]
    while open ≠ [] do
        n ← removeHead(open)
        if goal(state(n)) then return n
        for m ∈ expand(n) do
            if state(m) ∉ path(n) then insert(m, open)
    return fail
```

- Sprječava cikluse (osigurava potpunost algoritma)
- Ne sprječava ponavljanje na različitim putevima u stablu pretraživanja
- Povećava vremensku složenost za faktor $\mathcal{O}(d)$

Problem ponavljanja stanja (1)

- **Rješenje 1:** spriječiti povratak u stanje iz kojeg smo došli

Opći algoritam pretraživanja (nadopuna)

```
function search(s0, succ, goal)
    open ← [initial(s0)]
    while open ≠ [] do
        n ← removeHead(open)
        if goal(state(n)) then return n
        for m ∈ expand(n) do
            if state(m) ≠ state(parent(n)) then insert(m, open)
    return fail
```

- **Q:** Rješava li ovo problem ciklusa?

A: Ne općenito! (samo cikluse duljine 2, tzv. transpozicije)

Problem ponavljanja stanja (3)

- **Rješenje 3:** spriječiti ponavljanje bilo kojeg stanja

Opći algoritam pretraživanja s listom posjećenih stanja

```
function search(s0, succ, goal)
    open ← [initial(s0)]
    visited ← ∅
    while open ≠ [] do
        n ← removeHead(open)
        if goal(state(n)) then return n
        visited ← visited ∪ {state(n)}
        for m ∈ expand(n) do
            if state(m) ∉ visited then insert(m, open)
    return fail
```

- **NB:** Lista (skup) posjećenih stanja pohranjuje stanja, a ne čvorove

- Korištenje liste posjećenih čvorova osigurava potpunost algoritma (sprječava da zaglavi u beskonačnoj petlji)
- Osim toga, može **smanjiti prostornu i vremensku složenost**: Budući da se stanja ne ponavljaju, umjesto složenosti $\mathcal{O}(b^{d+1})$ imamo $\mathcal{O}(\min(b^{d+1}, b|S|))$, gdje je $|S|$ veličina prostora stanja (u praksi je često $b|S| < b^d$)
- Lista posjećenih stanja uobičajeno se implementira tablicom raspršenog adresiranja (engl. *hash table*) (omogućava provjeru “ $\text{state}(m) \notin \text{visited}$ ” u vremenu $\mathcal{O}(1)$)

Primjer: Problem misionara i kanibala (2)

$\text{problem} = (s_0, \text{succ}, \text{goal})$

① $s_0 = (3, 3, L)$

- ▶ broj misionara na lijevoj obali, $\{0, 1, 2, 3\}$
- ▶ broj kanibala na lijevoj obali, $\{0, 1, 2, 3\}$
- ▶ pozicija čamca, $\{L, R\}$

② $\text{succ}(m, c, b) = \{s \mid s \in \text{Succs}, \text{safe}(s)\}$

$$\text{Moves} = \{(1, 1), (2, 0), (0, 2), (1, 0), (0, 1)\}$$

$$\text{Succs} = \{f(s) \mid s \in \text{Moves}\}$$

$$f(m, c, b) = \begin{cases} (m - x, c - y, R) & \text{ako } b = L \\ (m + x, c + y, L) & \text{inače} \end{cases}$$

$$\text{safe}(m, c, b) = (m = 0) \vee (m = 3) \vee (m = c)$$

$$\text{③ } \text{goal}(m, c, b) = \begin{cases} \top & (m = c = 0) \wedge (b = R) \\ \perp & \text{inače} \end{cases}$$

Primjer: Problem misionara i kanibala (1)

Problem misionara i kanibala

Tri misionara i tri kanibala potrebno je jednim čamcem prevesti s jedne strane obale rijeke na drugu, pri čemu se niti u jednom trenutku na jednoj strani obale ne smije naći više kanibala nego misionara. Čamac može prevesti najviše dvije osobe i ne može ploviti prazan. Tražimo rješenje s **najmanjim brojem koraka**.



- Koji algoritam pretraživanja upotrijebiti?
- Kako prikazati problem?

Primjer: Problem misionara i kanibala (3)

- Jesmo li opisali sve što je bitno za problem?
- Jesmo li apstrahirali sve što je nebitno za problem?
- Generiramo li sve moguće poteze?
- Jesu li potezi koje generiramo legalni?
- Generiramo li neželjena stanja?
- Bi li bilo pametnije provjeru ispravnosti stanja ugraditi u ispitni predikat `goal`?
- Trebamo li paziti na ponavljanje stanja?
- Kakve su karakteristike problema?
 - ▶ $|S| = ?$
 - ▶ $b = ?$
 - ▶ $d = ?$
 - ▶ $m = ?$
- Je li ovo težak problem?

Laboratorijski zadatak: Problem ljubomornih muževa

Napišite program koji će **pretraživanjem u širinu** pronaći optimalno rješenje problema ljubomornih muževa.

Problem je opisan na sljedeći način: tri muževa i njihove supruge potrebno je jednim čamcem prevesti s jedne strane obale rijeke na drugu. Niti u jednom trenutku ne smije se dogoditi da neka od supruga bude u prisustvu drugoga muža, a da njezin muž nije također prisutan. Čamac može prevesti najviše dvije osobe i ne može ploviti prazan.

Optimalno rješenje je ono s najmanjim brojem koraka. Program treba ispisati rješenje u obliku niza operatora i stanja koja dovode do ciljnog stanja.

Q: Kako definirati skup S i funkciju succ?

Sažetak

- Mnogi problemi mogu se rješiti pretraživanjem prostora stanja
- Problemi se razlikuju po **broju stanja, faktoru grananja i dubini**
- Poželjna svojstva algoritama pretraživanja su **potpunost, optimalnost i mala prostorna složenost**
- Svi algoritmi pretraživanja nažalost imaju **eksponencijalnu vremensku složenost**
- Kod velikog broja stanja preporuča se koristiti **iterativno pretraživanje u dubinu**
- Treba voditi računa o ciklusima (ponavljanju stanja)



Sljedeća tema: Usmjereno pretraživanje

Laboratorijski zadatak: Igra brojki

Napišite program koji koristi **iterativno pretraživanje u dubinu** kako bi riješio problem igre brojki.

Cilj igre je, krenuvši od šest zadanih cijelih brojeva, pronaći aritmetički postupak kojim se izvodi neki slučajno generirani ciljni broj. Aritmetičke operacije koje se pritom smiju koristiti jesu zbrajanje, oduzimanje, množenje i dijeljenje bez ostatka. Svaki se broj može iskoristiti samo jednom ili niti jednom.

Početnih šest brojeva neka zadaje korisnik, a ciljni broj neka se generira slučajno iz intervala od 100 do 999.

Ako izraz kojim se izvodi točan broj nije pronađen, treba pronaći izraz koji izvodi broj najbliži traženome.

Q: Kako definirati skup S i funkciju succ?

3. Heurističko pretraživanje

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

Sveučilište u Zagrebu
Fakultet elektrotehnike i računarstva

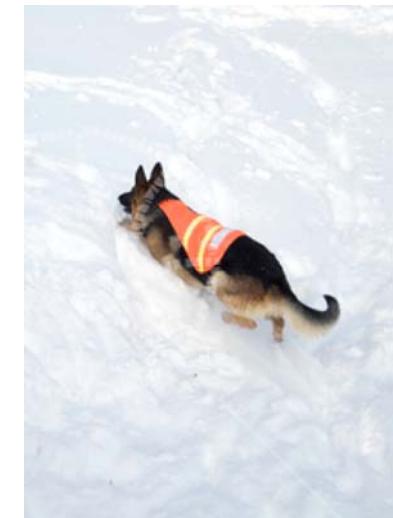
Ak. god. 2011/12.



Creative Commons Imenovanje–Nekomercijalno–Bez prerada 3.0

v3.1

- Slijepi postupci raspolažu isključivo egzaktnim informacijama (početnim stanjem, operatorima i ispitnim predikatom)
- Ne koriste nikakvu dodatnu **informaciju o prirodi problema** koja bi mogla poboljšati učinkovitost pretraživanja
- Ako otprilike znamo u kojem se smjeru nalazi rješenje, zašto ne iskoristiti to znanje kako bismo ubrzali pretragu?



Heuristika

- **Heuristika** – iskustvena pravila o prirodi problema i osobinama cilja čija je svrha pretraživanje brže **usmjereni** k cilju

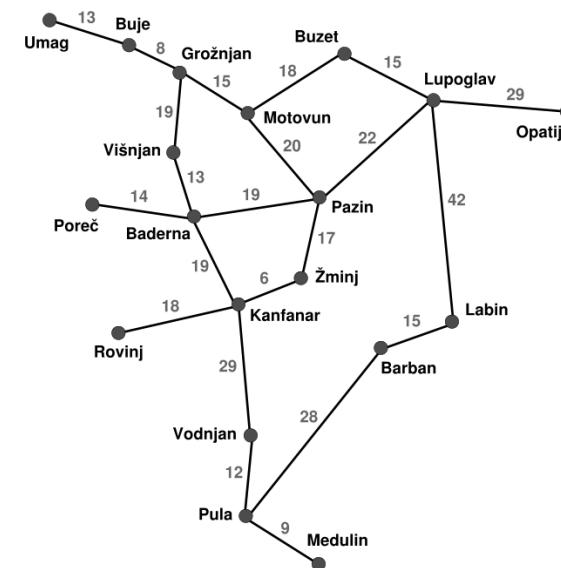
Heuristička funkcija

Heuristička funkcija $h : S \rightarrow \mathbb{R}^+$ pridjeljuje svakom stanju $s \in S$ procjenu udaljenosti od tog stanja do ciljnog stanja

Što je vrijednost $h(s)$ manja, to je čvor s bliži ciljnome stanju. Ako je s ciljno stanje, onda $h(s) = 0$

- Postupke pretraživanja koji koriste heuristiku kako bi suzili prostor pretraživanja nazivamo **heurističkim** ili **usmjerenim**

Primjer: Putovanje kroz Istru



Zračne udaljenosti do Buzeta:

Baderna	25
Barban	35
Buje	21
Grožnjan	17
Kanfanar	30
Labin	35
Lupoglav	13
Medulin	61
Motovun	12
Opatija	26
Pazin	17
Poreč	32
Pula	57
Rovinj	40
Umag	31
Višnjan	20
Vodnjan	47
Žminj	27

Primjer: Slagalica 3×3

početno stanje:

8		7
6	5	4
3	2	1

Heuristička funkcija?

- Broj pločica koje nisu na svome mjestu:

$$h_1\left(\begin{smallmatrix} 8 & 7 \\ 6 & 5 & 4 \\ 3 & 2 & 1 \end{smallmatrix}\right) = 7$$

ciljno stanje:

1	2	3
4	5	6
7	8	

$$Vrijedi h_2(s) \geq h_1(s)$$

Usmjereni pretraživanje

Razmotrit ćemo:

- Pretraživanje "najbolji prvi" (engl. *greedy best-first search*)
- Algoritam A^*
- Pretraživanje usponom na vrh (engl. *hill-climbing search*)

Pretraživanje "najbolji prvi"

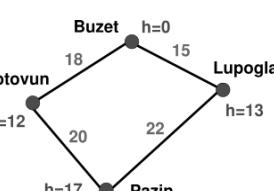
- Proširuje čvor s najboljom heurističkom vrijednošću

Pretraživanje "najbolji prvi"

```
function greedyBestFirstSearch(s0, succ, goal, h)
    open ← [initial(s0)]
    while open ≠ [] do
        n ← removeHead(open)
        if goal(state(n)) then return n
        for m ∈ expand(n, succ) do
            insertSortedBy(f, m, open)
    return fail
where f(n) = h(state(n))
```

Pohlepno pretraživanje

- Ovo je tzv. **pohlepan** (engl. *greedy*) algoritam "najbolji prvi"
- Pohlepno pretraživanje**: odabire se onaj čvor koji se čini najbliži cilju, ne uzimajući u obzir ukupnu cijenu puta
- Odabrani put možda nije optimalan, no algoritam nema mogućnost oporavka od pogreške! Dakle, algoritam **nije optimalan**
- Q:** Primjer?



- Nije potpun** (osim ako koristimo listu posjećenih stanja)
- Vremenska i prostorna složenost:** $\mathcal{O}(b^m)$

Algoritam A^*

- Algoritam "najbolji prvi" koji u obzir uzima i heuristiku i cijenu ostvarenog puta, tj. kombinira "najbolji prvi" i pretraživanje s jednolikom cijenom
- Kao i kod pretraživanja s jednolikom cijenom, pri proširenju čvora ažurira se cijena do tada ostvarenog puta:

```
function expand( $n$ , succ)
    return { $(s, g(n) + c) \mid (s, c) \in \text{succ}(\text{state}(n))$ }
```

Ukupna cijena računa se na temelju:

$g(n)$ – **stvarna cijena** puta od početnog čvora do čvora n
 $h(s)$ – **procjena cijene** puta od stanja s do cilja

$$f(n) = g(n) + h(\text{state}(n))$$

Algoritam A^* – izvedba

Algoritam A^*

```
function aStarSearch( $s_0$ , succ, goal,  $h$ )
    open ← [initial( $s_0$ )]
    closed ←  $\emptyset$ 
    while open ≠ [] do
         $n \leftarrow \text{removeHead}(open)$ 
        if goal(state( $n$ )) then return  $n$ 
        closed ← closed ∪ { $n$ }
        for  $m \in \text{expand}(n)$  do
            if  $\exists m' \in closed \cup open$  such that  $\text{state}(m') = \text{state}(m)$  then
                if  $g(m') < g(m)$  then continue
                else remove( $m'$ , closed ∪ open)
            insertSortedBy( $f$ ,  $m$ , open)
    return fail
where  $f(n) = g(n) + h(\text{state}(n))$ 
```

Podsjetnik: opći algoritam pretraživanja

Opći algoritam pretraživanja

```
function search( $s_0$ , succ, goal)
    open ← [initial( $s_0$ )]
    while open ≠ [] do
         $n \leftarrow \text{removeHead}(open)$ 
        if goal(state( $n$ )) then return  $n$ 
        for  $m \in \text{expand}(n, \text{succ})$  do
            insert( $m$ , open)
    return fail
```

Algoritam A^* – primjer izvođenja

- ➊ $open = [(Pula, 0)]$
 $closed = \emptyset$
- ➋ $\text{expand}(Pula, 0) = \{(Vodnjan, 12), (Barban, 28), (Medulin, 9)\}$
 $open = [(Vodnjan, 12)^{59}, (Barban, 28)^{63}, (Medulin, 9)^{70}]$
 $closed = \{(Pula, 0)\}$
- ➌ $\text{expand}(Vodnjan, 12) = \{(Kanfanar, 41), (Pula, 24)\}$
 $open = [(Barban, 28)^{63}, (Medulin, 9)^{70}, (Kanfanar, 41)^{71}]$
 $closed = \{(Pula, 0), (Vodnjan, 12)\}$
- ➍ $\text{expand}(Barban, 28) = \{(Labin, 43), (Pula, 56)\}$
 $open = [(Medulin, 9)^{70}, (Kanfanar, 41)^{71}, (Labin, 43)^{78}]$
 $closed = \{(Barban, 28), (Pula, 0), (Vodnjan, 12)\}$
- ➎ $\text{expand}(Medulin, 9) = \{(Pula, 18)\}$
 $open = [(Kanfanar, 41)^{71}, (Labin, 43)^{78}]$
 $closed = \{(Barban, 28), (Medulin, 9), (Pula, 0), (Vodnjan, 12)\}$
⋮

Algoritam A^* – svojstva

- **Vremenska i prostorna složenost:** $\mathcal{O}(\min(b^{d+1}, b|S|))$
(u praksi veći problem predstavlja prostorna složenost)
- **Potpunost:** da, jer u obzir uzima cijenu puta
- **Optimalnost:** da, ali pod uvjetom da je heuristika h optimistična:

Optimističnost heuristike

Heuristika h je **optimistična ili dopustiva** (engl. *optimistic, admissible*) akko nikad ne precjenjuje, tj. nikad nije veća od prave cijene do cilja:

$$\forall s \in S. h(s) \leq h^*(s),$$

gdje je $h^*(s)$ prava cijena od stanja s do cilja

- Ako heuristika nije optimistična, može se dogoditi da pretraga zaobiđe optimalni put jer se on čini skupljim nego što zapravo jest
- **Q:** Jesu li heuristike u ranijim primjerima optimistične?

Konzistentna heuristika (1)

- Uz pretpostavku optimistične heuristike, vrijedi $f(n) \leq C^*$ (funkcija cijene je odozgo ograničena)
- Duž staze u stablu pretraživanja, $f(n)$ može općenito rasti i padati, a u cilnjom stanju vrijedi $f(n) = g(n) = C^*$
- Poželjno je da $f(n)$ **monoton raste**:

$$\forall n_2 \in \text{expand}(n_1) \implies f(n_2) \geq f(n_1)$$

(Za savršenu heuristiku h^* vrijedi $f(n_1) = f(n_2) = C^*$)

- Ako $f(n)$ monoton raste, svaki čvor koji **prvi generiramo** za neko stanje bit će čvor s **najmanjom cijenom** za to stanje
- To znači da se niti za jedno stanje ne može dogoditi da na listi zatvorenih čvorova nađemo jeftiniji čvor s istim stanjem, pa ne moramo provjeravati cijenu već zatvorenih čvorova

Primjer: Slagalica 3×3

početno stanje:

8		7
6	5	4
3	2	1

ciljno stanje:

1	2	3
4	5	6
7	8	

Koje su heuristike optimistične?

- $h_1(s) =$ broj pločica koje nisu na svome mjestu
- $h_2(s) =$ zbroj Manhattan-udaljenosti pločica od svoga mesta
- $h_3(s) = 0$
- $h_4(s) = 1$
- $h_5(s) = h^*(s)$
- $h_6(s) = \min(2, h^*(s))$
- $h_7(s) = \max(2, h^*(s))$

Konzistentna heuristika (2)

- Ako $f(n)$ monotono raste, onda $\forall n_2 \in \text{expand}(n_1)$:

$$\begin{aligned} f(n_1) &\leq f(n_2) \\ g(n_1) + \underbrace{h(\text{state}(n_1))}_{s_1} &\leq g(n_2) + \underbrace{h(\text{state}(n_2))}_{s_2} \\ g(n_1) + h(s_1) &\leq \underbrace{g(n_1)}_{g(n_2)} + c + h(s_2) \\ h(s_1) &\leq h(s_2) + c \end{aligned}$$

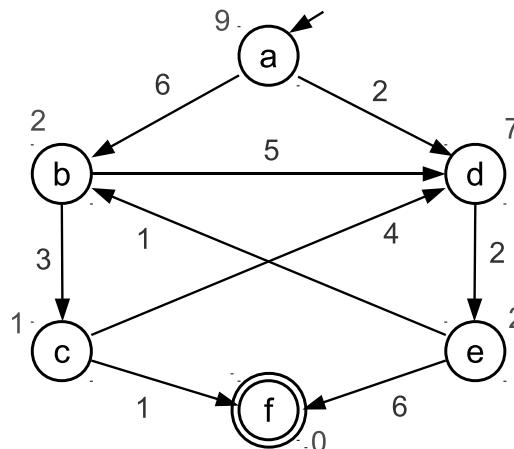
Konzistentnost heuristike

Heuristika h je **konzistentna ili monotona** akko:

$$\forall (s_2, c) \in \text{succ}(s_1). h(s_1) \leq h(s_2) + c$$

- **NB:** Konzistentna heuristika je nužno optimistična, a u praksi su optimistične heuristike ujedno i konzistentne

Konzistentna heuristika – primjer



Q: Je li ova heuristika optimistična? Je li konzistentna?

Algoritam A^* – varijante

Sljedeće varijante algoritma A^* također su **optimalne**:

- ① Varijanta bez liste *closed* (u praksi memoriji zahtjevnija zbog ponavljanja stanja)
- ② Varijanta s listom *closed*, ali bez otvaranja jednom zatvorenih čvorova, uz uvjet da je heuristička funkcija **konzistentna**
- ③ S listom *closed* i bez otvaranja, ali uz **pathmax-korekciju**:

$$f(n) = \max(f(\text{parent}(n)), g(n) + h(\text{state}(n)))$$

Svojstvo dominacije

Dominacija

Neka su A_1^* i A_2^* dva optimalna algoritma s *optimističnim* heurističkim funkcijama h_1 i h_2 . Algoritam A_1^* **dominira** nad algoritmom A_2^* akko:

$$\forall s \in S. h_1(s) \geq h_2(s)$$

Također kažemo da je algoritam A_1^* **obavješteniji** od algoritma A_2^*

- Obavješteniji algoritam općenito pretražuje manji prostor stanja od manje obaviještenog algoritma
- Npr. za slagalicu vrijedi: $h^*(s) \geq h_2(s) \geq h_1(s)$, tj. L1-heuristika daje obavješteniji algoritam od heuristike koja broji razmještene pločice
- Pritom u obzir treba uzeti i složenost izračunavanja heuristike!

Dobra heuristika

- Dobra heuristika je:

- ① optimistična
- ② što obavještenija
- ③ jednostavno izračunljiva

Pesimistične heuristike?

- Ako ne trebamo baš optimalno rješenje, nego neko rješenje koje je dovoljno dobro, možemo koristiti heuristiku koja nije optimistična (heuristiku koja precjenjuje)
- Uporaba takve heuristike dodatno će smanjiti broj generiranih čvorova
- Radimo kompromis između kvalitete rješenja i složenosti pretraživanja
- Kako oblikovati dobру heuristiku za neki zadani problem?

Oblikovanje heuristike

1 Relaksacija problema

- ▶ stvarna cijena **relaksiranog problema** je optimistična heuristika izvornog problema
- ▶ npr. relaksacija slagalice 3×3 : pločice se mogu pomicati bilo kuda \Rightarrow L1-udaljenost
- ▶ dobivamo optimističnu i ujedno konzistentnu heuristiku (zašto?)

2 Kombiniranje optimističnih heuristika

- ▶ Ako su h_1, h_2, \dots, h_n optimistične, možemo ih kombinirati u dominantnu heuristiku koja će također biti optimistična:

$$h(s) = \max(h_1(s), h_2(s), \dots, h_n(s))$$

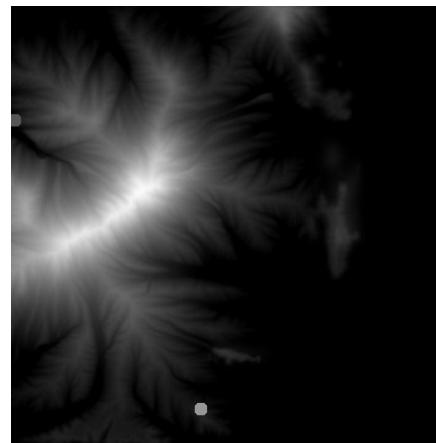
3 Cijena rješavanja podproblema

- ▶ baza uzoraka koja sadržava optimalne cijene pojedinih podproblema

4 Učenje heuristike

- ▶ primjena metoda strojnog učenja. Npr. učimo koeficijente w_1 i w_2 :
- $$h(s) = w_1 x_1(s) + w_2 x_2(s), \text{ gdje su } x_1 \text{ i } x_2 \text{ značajke stanja}$$

Pretraživanje visinske mape (1)



Visinska mapa
(svjetlijiji elementi su na većoj visini)
crveno: početno stanje, zeleno: ciljno stanje

Laboratorijski zadatak: Pretraživanje visinske mape

Napišite program koji će korištenjem algoritama A* izračunati najjeftiniji put između dvije točke na zadanoj visinskoj mapi.

Neka $\Delta v = v(x_2, y_2) - v(x_1, y_1)$. Dozvoljen je pomak s polja (x_1, y_1) na polje (x_2, y_2) ukoliko je $|x_1 - x_2| \leq 1, |y_1 - y_2| \leq 1$ i $\Delta v \leq m$.

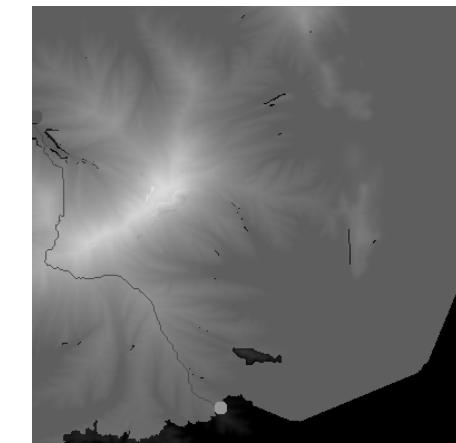
Cijena puta od polja (x_1, y_1) do polja (x_2, y_2) je

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} + \left(\frac{1}{2}\text{sgn}(\Delta v) + 1\right) \cdot |\Delta v|$$

Mapu i parametre program treba učitati iz zadane tekstne datoteke.

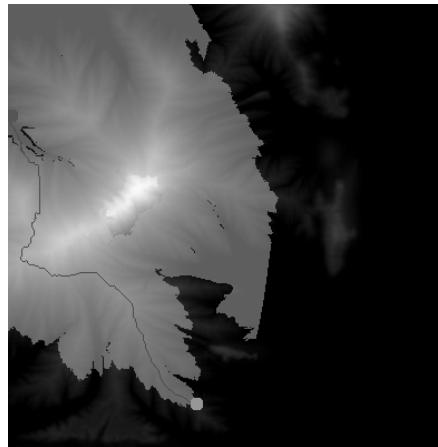
Program na generiranoj slici treba prikazati visinsku mapu, sve zatvorene čvorove, sve otvorene čvorove, pronađeni put, duljinu pronađenog puta i broj koraka algoritma. Potrebno je osmisliti barem tri različite heuristike i isprobati rad algoritama s tim heuristikama.

Pretraživanje visinske mape (2)



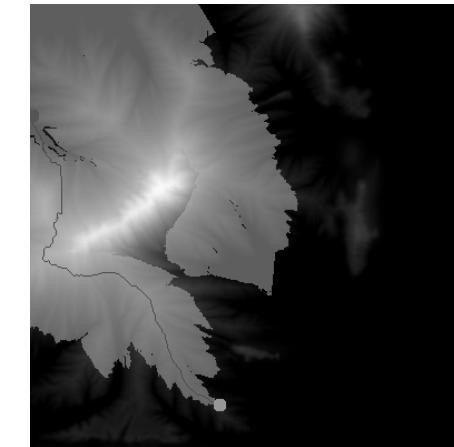
Pretraživanje s jednolikom cijenom
(crveno: pronađen put, žuto: posjećena stanja)
broj zatvorenih čvorova: 140580, dužina puta: 740,58

Pretraživanje visinske mape (3)



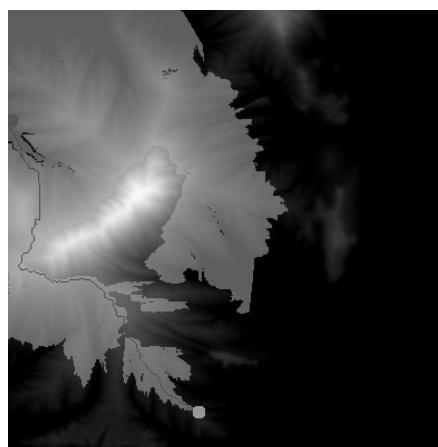
Algoritam A*
(heuristika: zračna udaljenost)
broj zatvorenih čvorova: 64507, dužina puta: 740,58

Pretraživanje visinske mape (4)



Algoritam A*
(heuristika: zračna udaljenost + $\frac{1}{2}\Delta v$)
broj zatvorenih čvorova: 56403, dužina puta: 740,58

Pretraživanje visinske mape (5)



Algoritam A*
(heuristika: zračna udaljenost + Δv)
broj zatvorenih čvorova: 52099, dužina puta: 755,16
Ova heuristika nije optimistična!

Pretraživanje visinske mape (6)



Pohlepno pretraživanje "najbolji prvi"
(uz uporabu liste posjećenih stanja)
broj zatvorenih čvorova: 822, dužina puta: 1428,54

Algoritam uspona na vrh

- Kao pohlepno pretraživanje "najbolji prvi", s tom razlikom da se prošireni čvorovi uopće ne pohranjuju u memoriji

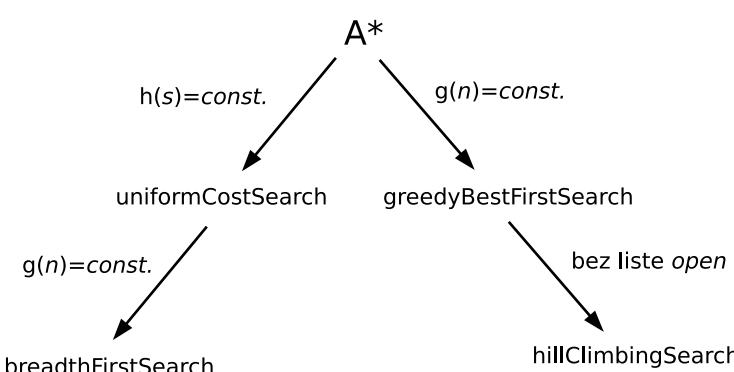
Algoritam uspona na vrh

```
function hillClimbingSearch( $s_0$ , succ,  $h$ )
 $n \leftarrow \text{initial}(s_0)$ 
loop do
     $M \leftarrow \text{expand}(n, \text{succ})$ 
    if  $M = \emptyset$  then return  $n$ 
     $m \leftarrow \text{minimumBy}(f, M)$ 
    if  $f(n) < f(m)$  then return  $n$ 
     $n \leftarrow m$ 
where  $f(n) = h(\text{state}(n))$ 
```

Algoritam uspona na vrh – svojstva

- **Nije potpun i nije optimalan**
- Lako zaglavljuje u tzv. **lokalnim optimumima**
- Učinkovitost uvelike ovisi o izboru heurističke funkcije
- Uobičajeno se koristi tehnika *slučajnog ponovnog starta* (engl. *random-restart*)
- **Vremenska složenost:** $\mathcal{O}(m)$
- **Prostorna složenost:** $\mathcal{O}(1)$

Odnosi između algoritama



- A^* dominira nad algoritmima uniformCostSearch i breadthFirstSearch

Sažetak

- Heuristička funkcija **usmjerava pretraživanje** i tako ga ubrzava
- Heuristička funkcija definira **procjenu udaljenosti** trenutnog stanja od ciljnog stanja. Što je ona manja, to smo bliže cilju
- Heuristika treba biti **optimistična**, može biti **konzistentna**, a poželjno je da bude što **obavještenija** jer to ubrzava pretraživanje
- Algoritami "najbolji prvi" i "uspon na vrh" su **pohlepni** i zato nisu optimalni
- Algoritam A^* je **potpun i optimalan**



Sljedeća tema: Igranje igara

4. Igranje igara

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

Sveučilište u Zagrebu
Fakultet elektrotehnike i računarstva

Ak. god. 2011/12.

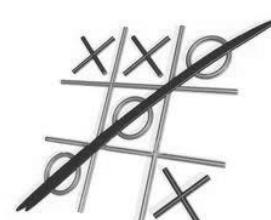


Creative Commons Imenovanje–Nekomercijalno–Bez prerada 3.0

v2.1

Igre

- Također problem pretraživanja prostora stanja, ali postoji **protivnik**
- U svakom stanju potrebno je donijeti **optimalnu odluku** o sljedećem potezu, tj. treba pronaći optimalnu **strategiju**
- Fokusiramo se na **determinističke** igre s **dva igrača, potpunom informacijom i sumom nula**



Formalizacija problema

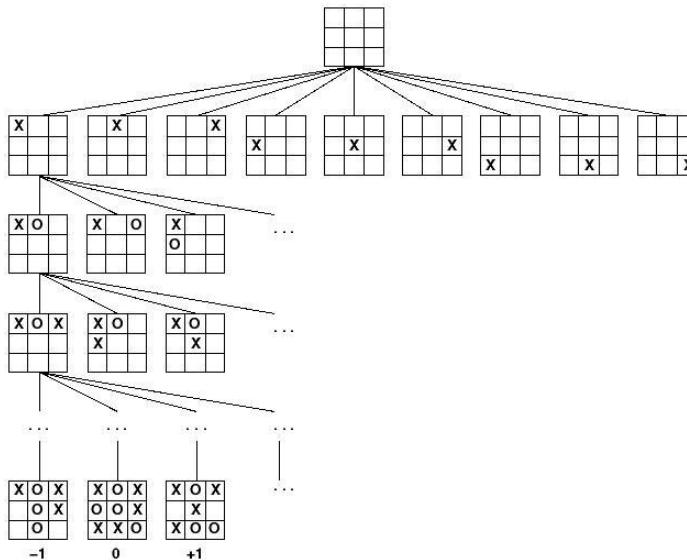
Problem pretraživanja sa sljedećim komponentama:

Igra

- **Početno stanje** igre s_0
- **Funkcija sljedbenika** $\text{succ} : S \rightarrow \wp(S)$, koja definira valjane poteze igre (prijelaze između stanja)
- **Test na završno stanje** terminal : $S \rightarrow \{\top, \perp\}$
- **Isplatna funkcija** utility : $S \rightarrow \mathbb{R}$ koja pridjeljuje numeričku vrijednost koju kao nagradu dobiva igrač u završnom stanju igre
Npr. u šahu: $\text{utility}(s) \in \{+1, 0, -1\}$

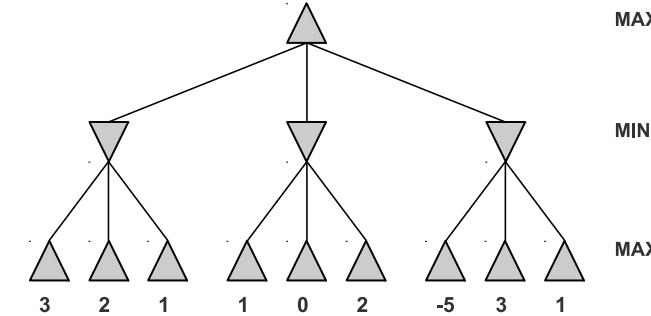
Početno stanje i funkcija sljedbenika definiraju **stablo igre**

Stablo igre



Metoda minimaks

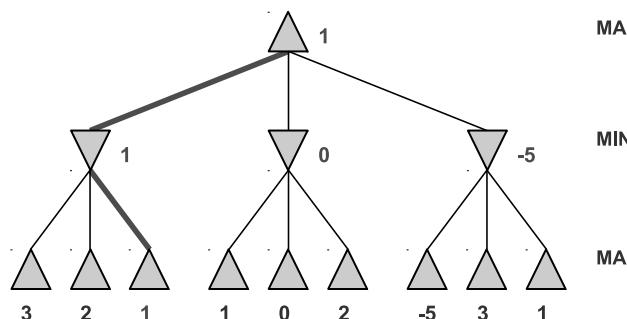
- Igrače ćemo nazvati MAX (računalo) i MIN (protivnik)
- Igrač MAX nastoji **maksimizirati** svoj dobitak, dok igrač MIN nastoji **minimizirati** dobitak igrača MAX
- Igrači igraju naizmjenično: čvorovi na parnoj udaljenosti neka su MAX, a čvorovi na neparnoj udaljenosti neka su MIN



- Q:** Što je u ovom slučaju optimalna strategija igrača MAX?

Optimalna strategija

- Optimalna strategija** igrača MAX je strategija koja mu donosi najveći dobitak, uz pretpostavku da igrač MIN koristi istu strategiju
- Svaki igrač koristi strategiju koja **minimizira maksimalan gubitak**

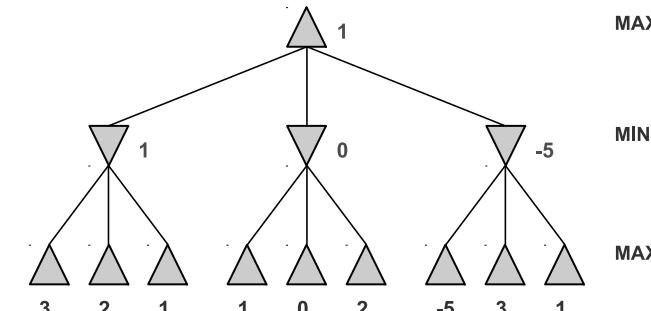


- Da bismo odredili **optimalnu strategiju** onog igrača koji je upravo na redu, trebamo izračunati **minimaks-vrijednost** korijenskog čvora

Minimaks-vrijednost

- Minimaks-vrijednost čvora s definirana je rekurzivno:

$$m(s) = \begin{cases} \text{utility}(s) & \text{ako } \text{terminal}(s) \\ \max_{t \in \text{succ}(s)} m(t) & \text{ako } s \text{ je MAX čvor} \\ \min_{t \in \text{succ}(s)} m(t) & \text{ako } s \text{ je MIN čvor} \end{cases}$$



$$m(s_0) = \max (\min(3, 2, 1), \min(1, 0, 2), \min(-5, 3, 1)) = 1$$

Algoritam minimaks

```
function maxValue(s)
    if terminal(s) then return utility(s)
    m ← −∞
    for t ∈ succ(s) do
        m ← max(m, minValue(t))
    return m

function minValue(s)
    if terminal(s) then return utility(s)
    m ← +∞
    for t ∈ succ(s) do
        m ← min(m, maxValue(t))
    return m
```

NB: Pretraživanje **u dubinu** ostvareno dvjema međusobno rekursivnim funkcijama (alternacija između stanja MAX i MIN)

Nesavršene odluke

- U stvarnosti nemamo vremena potpuno pretražiti stablo sve do završnih čvorova
- Moramo donositi **vremenski ograničene i nesavršene odluke**
- Pretraživanje treba **presjeći** na određenoj dubini d i napraviti **procjenu** vrijednosti isplatne funkcije uporabom **heurističke funkcije**
- Vrijednost $h(s)$ je procjena isplativosti stanja s za igrača MAX
- Npr. za šah: zbroj vrijednosti igračevih figura
- Heuristička funkcija često je definirana kao **težinska linearna kombinacija** više značajki:

$$h(s) = w_1x_1(s) + w_2x_2(s) + \dots + w_nx_n(s)$$

- **NB:** Igrači tipično imaju različite heurističke funkcije (zato se i čine nepredvidivi)

Minimax algorithm – napomene

- U praksi je protivnikova strategija nepoznata (i vjerojatno različita od strategije igrača MAX), pa zbog toga nije moguće savršeno predviđjeti protivnikove poteze (inače bi igra ionako bila dosadna)
- Zbog toga, kako bi napravili optimalan potez, svaki puta kada su na redu igrači moraju nanovo izračunati svoju optimalnu strategiju, krenuvši od trenutne pozicije igre u korijenu stabla
- Minimax pretražuje u dubinu, pa je njegova prostorna složenost $\mathcal{O}(m)$, gdje je m dubina stabla pretraživanja
- Međutim, vremenska složenost je $\mathcal{O}(b^m)$, gdje je b faktor grananja igre. To je vrlo nezgodno!

Algoritam minimaks (2)

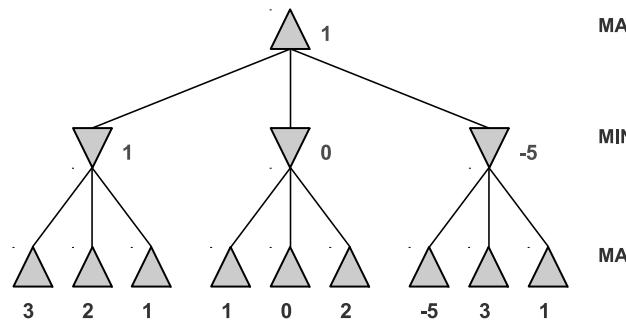
Algoritam minimaks s presjecanjem

```
function maxValue(s, d)
    if terminal(s) then return utility(s)
    if d = 0 then return h(s)
    m ← −∞
    for t ∈ succ(s) do
        m ← max(m, minValue(t, d − 1))
    return m

function minValue(s, d)
    if terminal(s) then return utility(s)
    if d = 0 then return h(s)
    m ← +∞
    for t ∈ succ(s) do
        m ← min(m, maxValue(t, d − 1))
    return m
```

Podrezivanje alfa-beta

- Broj stanja igre raste eksponencijalno s brojem poteza
- Primjenom **podrezivanja** taj broj možemo međutim preploviti
- Q: Možemo li odrediti minimaks-vrijednost, a da ne obiđemo cijelo stablo igre? A: Da!



$$m(s_0) = \max (\min(3, 2, 1), \min(1, X, X), \min(-5, X, X)) = 1$$

Algoritam minimaks (3)

Algoritam minimaks s podrezivanjem alfa-beta

```

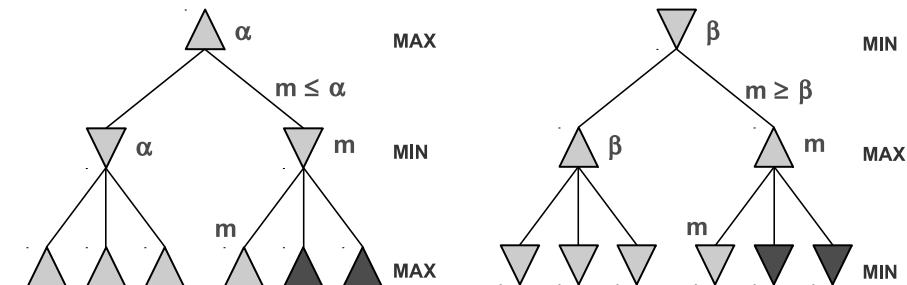
function maxValue( $s, \alpha, \beta$ )      -- početno:  $\text{maxValue}(s_0, -\infty, +\infty)$ 
   $m \leftarrow \alpha$ 
  for  $t \in \text{succ}(s)$  do
     $m \leftarrow \max(m, \text{minValue}(t, m, \beta))$ 
    if  $m \geq \beta$  then return  $\beta$       -- beta-podrezivanje
  return  $m$ 

function minValue( $s, \alpha, \beta$ )
  if terminal( $s$ ) then return utility( $s$ )
   $m \leftarrow \beta$ 
  for  $t \in \text{succ}(s)$  do
     $m \leftarrow \min(m, \text{maxValue}(t, \alpha, m))$ 
    if  $m \leq \alpha$  then return  $\alpha$       -- alfa-podrezivanje
  return  $m$ 

```

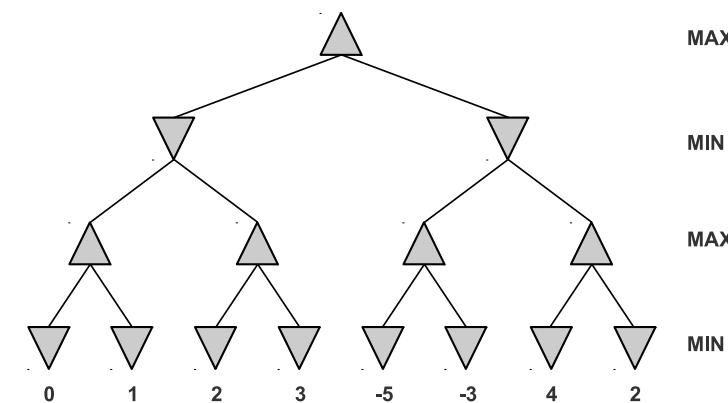
Podrezivanje alfa-beta

- Podrezujemo kad god se za neki čvor ustanovi da potezi **ni u kojem slučaju ne mogu biti povoljniji** od nekog već istraženog poteza
- Ako su to potezi računala: **alfa-podrezivanje**
- Ako su to potezi protivnika: **beta-podrezivanje**

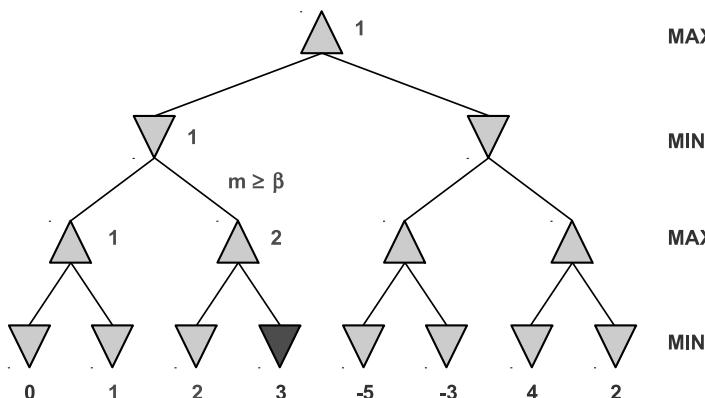


α – najveća nađena MAX-vrijednost β – najmanja nađena MIN-vrijednost

Podrezivanje alfa-beta – primjer (1)



Podrezivanje alfa-beta – primjer (2)



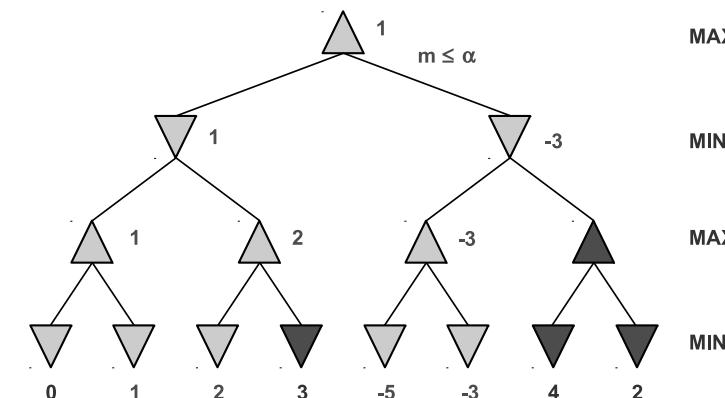
Laboratorijski zadatak: Igra šibica

Napišite program za igru šibica na temelju **algoritma minimaks**.

U igri šibica sudjeluju dva igrača. Pred njima se nalazi n odvojenih hrpa šibica (u svakoj hrpi može se nalaziti različit broj šibica). Igrač koji je na potezu mora sa samo jedne hrpe maknuti najmanje jednu a najviše k šibica. Igra završava kada su sve šibice uklonjene, a gubi onaj igrač koji je morao odigrati zadnji potez.

Napravite minimalističko korisničko sučelje koje će prikazivati trenutno stanje igre te omogućiti čovjeku da igra protiv računala. Za svaki potez igrača program treba dojaviti je li taj potez odigran optimalno u smislu minimaks-odluke. Korisnik pri pokretanju programa zadaje parametre n (broj hrpa), k (najveći broj šibica koje je u jednom potezu moguće ukloniti) te početni broj šibica u svakoj od n hrpa.

Podrezivanje alfa-beta – primjer (3)



Laboratorijski zadatak: Igra dame

Napišite program za igru Dame temeljen na **algoritmu minimaks** uz ograničeno pretraživanje i **podrezivanje alfa-beta**.

Definirajte barem dvije različite heurističke funkcije za vrednovanje isplativosti svake pozicije igre. Pri oblikovanju heurističke funkcije posebnu pažnju pokušajte posvetiti situacijama kada se u igri dođe do kraljeva. Pretraživanje ograničite vremenski, i to brojem razmatranih pozicija, dubinom pretraživanja ili stvarnim vremenom.

Napravite minimalističko korisničko sučelje koje će omogućiti čovjeku da igra protiv računala te prikazivati trenutnu poziciju igre. Za svaki potez igrača program treba dojaviti je li taj potez odigran optimalno u smislu minimaks-odluke. Omogućite i da program igra protiv samoga sebe, pri čemu igrači mogu koristiti različite heurističke funkcije.

Sažetak

- Igranje igara je problem pretraživanje stanja kod kojega se izmjenjuju potezi dvaju protivnika
- **Algoritam minimaks** pronalazi optimalnu strategiju koja **minimizira maksimalan očekivani gubitak** koji bi mogao zadati protivnik
- U stvarnosti nije moguće potpuno pretražiti stablo igre je preveliko, pa umjesto toga pretražujemo do određene dubine i zatim koristimo **heurističku funkciju** da bismo procijenili vrijednosti stanja igre
- Različiti igrači koriste različite heurističke funkcije. Igrač ne poznaje protivnikovu heurstiku
- **Podrezivanje alfa-beta** smanjuje broj čvorova koje treba ispitati
- Nismo razmotrili: igre s više igrača, igre s elementom slučajnosti



Sljedeća tema: Logički agenti

5. Logika i zaključivanje

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

Sveučilište u Zagrebu
Fakultet elektrotehnike i računarstva

Ak. god. 2011/12.



Creative Commons Imenovanje–Nekomercijalno–Bez prerada 3.0

v1.7



Motivacija

- **Prikazivanje znanja** (engl. *knowledge representation*) središnji je problem umjetne inteligencije
- Rješavanje mnogih problema iz stvarnog svijeta iziskuje **goleme količine znanja**, čak i kada se ograničimo na neku usku domenu
- Drugi važan problem je **zaključivanje** (engl. *inference*): kako iz prikazanog znanja izvoditi novo znanje
- Na temelju zaključivanja sustav može donositi odluke, planirati, razumijevati prirodan jezik, dokazivati teoreme, itd.



Logic: another thing that penguins aren't very good at.

Simbolizam vs. konektivizam

- Simbolička logika temeljni je alat **simboličkog pristupa** umjetnoj inteligenciji:

Simbolički pristup

Sve znanje iz eksternog svijeta može se prikazati **simbolima**. Zaključivanje se provodi **manipulacijom** nad tim simbolima. Inteligentno rasudivanje ili ponašanje svodi se na zaključivanje.

- Tomu suprotan je **konektivistički pristup**:

Konektivistički pristup

Mentalna stanja i ponašanje proizlazi iz **interakcije** velikog broja međusobno **povezanih jednostavnih** obradbenih jedinica. Tipična paradigma ovog pristupa jesu umjetne neuronske mreže.

- Držat ćemo se (još neko vrijeme) simboličkog pristupa

Simbolička logika

Simbolička logika (matematička logika) grana je matematike koja se bavi matematičkim konceptima izraženima u **formalnim logičkim sustavima**. Takvi sustavi omogućavaju apstraktno rasuđivanje



Ekspresivnost logike

- Postoje razne vrste logike koje se razlikuju u navedene tri komponente
- Neki formalizmi su više, a neki manje **ekspresivni** (izražajni)
- Što je logika ekspresivnija, to je složenija njezina sintaksa, semantika i teorija dokaza. U logici koja je vrlo ekspresivna možemo detaljno opisati stanje svijeta
- S druge strane, što je logika ekspresivnija, to manje možemo dokazati
- Logiku u kojoj možemo provjeriti valjanost bilo koje formule zovemo **odlučivom** (engl. *decidable*)

Kompromis između ekspresivnosti i odlučivosti

U načelu, što je logika ekspresivnija, to je u njoj manje toga moguće dokazati. Vrlo ekspresivni logički sustavi nisu odlučivi. Sustavi koji nisu vrlo ekspresivni su odlučivi, ali u njima se ne može puno toga prikazati.

- Ekspresivnost ovisi o **ontološkim** i **epistemološkim** prepostavkama

Formalan logički sustav

Svaki sustav formalne logike sastoji se od tri komponente:

Komponente formalnog logičkog sustava

- Sintaksa** (engl. *syntax*) – opisuje jezične strukture koje sačinjavaju rečenice logike, odnosno definira formalna pravila za izgradnju logičkih formula
- Semantika** (engl. *semantics*) – opisuje značenje jezičnih struktura, npr. koje su jezične strukture istinite a koje lažne, ili koji je odnos jezičnih elemenata i objekata u stvarnosti
- Teorija dokaza** (engl. *proof theory*) – definira mehanizme koji omogućavaju zaključivanje, tj. izvođenje zaključaka iz danih premsa
 - (1) i (2) omogućavaju prikazivanje znanja iz stvarnog svijeta
 - (3) omogućava izvođenje novog znanja

Ontološke određenosti

Ontologija

Ontologija je filozofska disciplina (grana metafizike) koja se bavi pitanjima **postojanja i stvarnosti**, odnosno pitanjem koji entiteti postoje i kakvi su njihovi međusobni odnosi



Ontološke određenosti (engl. *ontological commitments*)

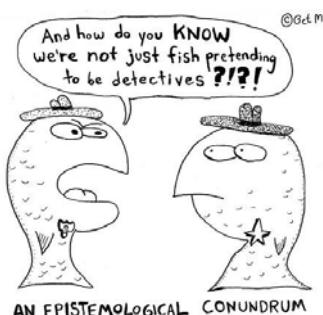
Ontološke određenosti definiraju što prepostavljamo da u svijetu postoji. Npr., kod **propozicijske logike** prepostavljamo da se svjet sastoji od činjenica koje su istinite ili lažne, **vremenska logika** dodatno prepostavlja da u svijetu postoji uređaj vremenskih trenutaka, itd.

- NB:** Riječ **ontologija** u umjetnoj inteligenciji ima dodatno značenje: formalan prikaz znanja unutar neke domene (no o tome kasnije)

Epistemološke određenosti

Epistemologija

Epistemologija je je grana filozofije koja se bavi **znanjem** – njegovom prirodom i njegovim dosegom, izvorom i ograničenjem.



Epistemološke određenosti (engl. *epistemological commitments*)

Epistemološke određenosti definiraju moguća stanja znanja. Npr., kod **propozicijske logike** svaka je činjenica ili istinita ili lažna. Nije moguće da je neka činjenica djelomično istinita, da je nesigurna, ili da u nju samo vjerujemo. Postoje vrste logika koje imaju takve epistemološke određenosti koje omogućavaju iskazivanje vjerovanja i različitih stupnjeva pouzdanosti.

Sintaksa propozicijske logike (1)

Simboli propozicijske logike

- ① Skup **propozicijskih varijabli** ili **atomičkih formula**,
 $V = \{A, B, C, \dots\}$
- ② **Logički operatori** ili logički veznici:
 - ▶ \neg (negacija)
 - ▶ \vee (operator ili)
 - ▶ \wedge (operator i)
 - ▶ \rightarrow (implikacija)
 - ▶ \leftrightarrow (ekvivalencija)
- ③ Logičke konstante *True* i *False* koje označavaju uvijek istinitu odnosno uvijek lažnu propoziciju

Vrste logike

- Propozicijska logika (logika sudova)
- Predikatna logika (logika objekata)
- Vremenska logika (engl. *temporal logic*)
- Opisna logika (engl. *description logic*)
- Neizrazita logika (engl. *fuzzy logic*)
- Modalna logika (engl. *modal logic*)
- Epistemička logika
- ...

Mi ćemo se usredotočiti na propozicijsku logiku. Ta logika ima minimalne pretpostavke: postoje činjenice koje su istinite ili lažne.

Drugi put ćemo razmatrati predikatnu logiku, koja je ekspresivnija.

Sintaksa propozicijske logike (2)

Dobro oblikovana formula (wff)

Dobro oblikovana formula (engl. *well-formed formula*, wff) ili, jednostavnije, **formula** propozicijske logike definirana je rekurzivno

- ① Atom je formula
 - ② Ako je F formula tada je i $(\neg F)$ formula
 - ③ Ako su F i G formule tada su formule:
 - ▶ $(F \wedge G)$
 - ▶ $(F \vee G)$
 - ▶ $(F \rightarrow G)$
 - ▶ $(F \leftrightarrow G)$
 - ④ Ništa drugo nije wff
- Dopuštamo izostavljanje zagrada u pravilu (2). Dopuštamo izostavljanje vanjskih zagrada u pravilu (3).

Primjeri atoma:

- $A = \text{"Zemlja je okrugla"}$
- $B = \text{"Harry Potter se školuje u Hogwartsu"}$
- $C = \text{"Propozicijska logika je najmoćnija shema za prikaz znanja"}$
- $D = \text{"Minotaur je mitsko biće"}$

Primjeri formula:

- C
- $\neg C$
- $((A \vee B) \vee \neg C)$
- $((B \vee D) \wedge (\neg B \vee C)) \rightarrow (A \vee C)$
- $((C \vee D) \rightarrow (\neg A \leftrightarrow B))$

Interpretacija

Neka je F dobro oblikovana formula propozicijske logike te neka su E_1, E_2, \dots, E_n propozicijske varijable koji se u njoj pojavljuju.

Interpretacija $I : V \rightarrow \{\top, \perp\}$ formule F jest pridjeljivanje vrijednosti istinitosti iz skupa $\{\top, \perp\}$ varijablama iz skupa V .

Funkcija I svakoj propozicijskoj varijabli E_i pridjeljuje vrijednost $I(E_i) = \top$ (istinito) ili vrijednost $I(E_i) = \perp$ (lažno), ali ne i oboje.

- Formula koja ima n atoma ima 2^n različitih interpretacija
- Svaka interpretacija I opisuje jednu moguću **situaciju u svijetu**

Semantika logičkih operatora

Tablice istinitosti

F	G	$\neg F$	$F \wedge G$	$F \vee G$	$F \rightarrow G$	$F \leftrightarrow G$
\perp	\perp	\top	\perp	\perp	\top	\top
\perp	\top	\top	\perp	\top	\perp	
\top	\perp	\perp	\perp	\top	\perp	
\top	\top	\perp	\top	\top	\top	\top

- Svi operatori imaju "prirodnu" semantiku (kao u jeziku), osim operatora \rightarrow , koji je manje intuitivan
- U formuli $F \rightarrow G$, formulu F nazivamo **antecedens**, a formulu G nazivamo **konzekvens**
- **NB:** Implikacija ne znači nužno uzročnost (kauzalnost). $F \rightarrow G$ ne znači da F uzorkuje G !

Provjera istinitosti formule

Istinitost formule

Za zadalu interpretaciju I , istinitost formule F definirana je rekurzivno:

$$\begin{aligned} I(\text{True}) &\equiv \top \\ I(\text{False}) &\equiv \perp \\ I(\neg F) &\equiv \neg I(F) \\ I(F \vee G) &\equiv I(F) \vee I(G) \\ I(F \wedge G) &\equiv I(F) \wedge I(G) \end{aligned}$$

- Npr., istinitost formule $((A \vee B) \wedge C) \wedge (\neg B \vee C)$ za interpretaciju $I(A) = \perp, I(B) = \top, I(C) = \top$:

$$\begin{aligned} I(((A \vee B) \wedge C) \wedge (\neg B \vee C)) &\equiv \\ I((A \vee B) \wedge C) \wedge I(\neg B \vee C) &\equiv \\ (I(A \vee B) \wedge I(C)) \wedge (I(\neg B) \vee I(C)) &\equiv \\ ((I(A) \vee I(B)) \wedge I(C)) \wedge (I(\neg B) \vee I(C)) &\equiv \dots \end{aligned}$$

Model

Model

Ako je formula F istinita u interpretaciji I , onda kažemo da je I **model** formule F .

Također kažemo da interpretacija I **zadovoljava** formulu F .

- Model opisuje jednu **situaciju**
- Npr., model formule $\neg A \wedge D$ je situacija u kojoj $A = \perp$ (*Zemlja nije okrugla*) i $D = \top$ (*Minotaur je mitsko biće*)
- Jedna formula može imati više modela, pa onda opisuje više situacija odjednom
- Npr., formula $A \vee D$ ima više modela. (Koje?)

Valjanost, proturječnost i zadovoljivost (1)

Valjana formula

Formula je **valjana (tautologija)** ako i samo ako je istinita za svaku svoju interpretaciju.

Proturječna formula

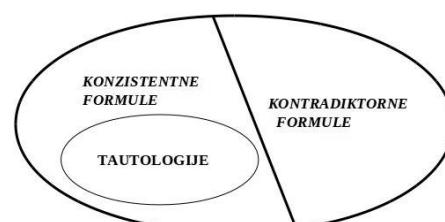
Formula je **proturječna (kontradikcija, nezadovoljiva, nekonzistentna antitautologija)** ako i samo ako je lažna za svaku svoju interpretaciju.

Zadovoljiva formula

Formula je **zadovoljiva (konzistentna, ispunjiva)** ako i samo ako je istinita barem za jednu interpretaciju.

Valjanost, proturječnost i zadovoljivost (2)

- Formula je zadovoljiva akko nije proturječna
- Formula je valjana akko je njezina negacija kontradikcija
- Ako je formula valjana, onda je i zadovoljiva, ali obrat ne vrijedi
- Ako formula nije valjana, onda ne znači da je proturječna
- Ako formula nije proturječna, onda je po definiciji zadovoljiva, ali ne mora biti valjana



Valjanost, proturječnost i zadovoljivost – primjeri

- P – zadovoljiva
- $P \vee Q$ – zadovoljiva
- $\neg(P \vee Q)$ – zadovoljiva
- $\neg P \wedge P$ – proturječna
- $P \wedge P$ – zadovoljiva
- $P \rightarrow Q$ – zadovoljiva
- $P \vee \neg P$ – valjana
- $(P \rightarrow Q) \wedge P$ – zadovoljiva
- $((P \rightarrow Q) \wedge P) \rightarrow Q$ – valjana
- $((P \rightarrow Q) \wedge P) \rightarrow \neg Q$ – zadovoljiva
- $P \leftrightarrow Q$ – zadovoljiva

Ekvivalentne formule

Formula F je **ekvivalentna** formulji G , što označavamo kao $F \equiv G$, ako i samo ako su vrijednosti istinitosti formula F i G jednake za svaku moguću interpretaciju formula F i G

- **NB:** Ne treba brkati $F \equiv G$ i $F \leftrightarrow G$
- $F \equiv G$ nije formula propozicijske logike (je \equiv nije dio sintakse propozicijske logike)
- $F \leftrightarrow G$ je valjana formula akko vrijedi $F \equiv G$

Ekvivalencije propozicijske logike (2)

(15) $G \vee False$	\equiv	G	
(16) $G \vee \neg G$	\equiv	$True$	– isklj. trećega
(17) $(F \wedge G) \wedge H$	\equiv	$F \wedge (G \wedge H)$	
(18) $(F \vee G) \vee H$	\equiv	$F \vee (G \vee H)$	{ asocijativnost }
(19) $F \wedge G$	\equiv	$G \wedge F$	
(20) $F \vee G$	\equiv	$G \vee F$	{ komutativnost }
(21) $F \vee (G \wedge H)$	\equiv	$(F \vee G) \wedge (F \vee H)$	
(22) $F \wedge (G \vee H)$	\equiv	$(F \wedge G) \vee (F \wedge H)$	{ distributivnost }
(23) $\neg(F \vee G)$	\equiv	$\neg F \wedge \neg G$	
(24) $\neg(F \wedge G)$	\equiv	$\neg F \vee \neg G$	{ de Morganovi zakoni }
(25) $F \vee (F \wedge G)$	\equiv	F	
(26) $F \wedge (F \vee G)$	\equiv	F	{ apsorpcija }
(27) $F \vee (\neg F \wedge G)$	\equiv	$F \vee G$	
(28) $F \wedge (\neg F \vee G)$	\equiv	$F \wedge G$	

Ekvivalencije propozicijske logike (1)

(1) $\neg\neg F$	\equiv	F	– involucija
(2) $F \rightarrow G$	\equiv	$\neg F \vee G$	– ukl. implikacije
(3) $F \rightarrow G$	\equiv	$\neg G \rightarrow \neg F$	– kontrapozicija
(4) $F \rightarrow (G \rightarrow H)$	\equiv	$G \rightarrow (F \rightarrow H)$	
(5) $F \rightarrow (G \rightarrow H)$	\equiv	$(F \wedge G) \rightarrow H$	
(6) $F \leftrightarrow G$	\equiv	$(F \wedge G) \vee (\neg F \wedge \neg G)$	
(7) $F \leftrightarrow G$	\equiv	$(F \rightarrow G) \wedge (G \rightarrow F)$	
(8) $F \leftrightarrow G$	\equiv	$(\neg F \vee G) \wedge (\neg G \vee F)$	
(9) $G \wedge G$	\equiv	G	– idempotencija
(10) $G \wedge True$	\equiv	G	
(11) $G \wedge False$	\equiv	$False$	
(12) $G \wedge \neg G$	\equiv	$False$	– zakon kontradikcije
(13) $G \vee G$	\equiv	G	– faktorizacija
(14) $G \vee True$	\equiv	$True$	

Logička posljedica

- Koji zaključci slijede iz danih premissa?

Logička posljedica

Formula G je **logička (semantička) posljedica** formula F_1, \dots, F_n ako i samo ako svaka interpretacija koja zadovoljava formulu $F_1 \wedge \dots \wedge F_n$ također zadovoljava formulu G .

Drugim riječima: formula G je logička posljedica formula F_1, \dots, F_n akko je svaki model od $F_1 \wedge \dots \wedge F_n$ ujedno i model od G .

Pišemo $F_1, F_2, \dots, F_n \models G$ i čitamo “ F_1, \dots, F_n logički (semantički) povlači (engl. logically entails, semantically entails) G ”.

- Logička posljedica je **znanje** koje slijedi iz premissa
- Epistemološki gledano, to znanje nije novo jer implicitno već postoji u premissama. Međutim, logička posljedica to znanje čini eksplicitnim

Logička posljedica – primjer

- Dokažimo da je Q logička posljedica formula $P \vee Q$ i $\neg P$, tj.:

$$P \vee Q, \neg P \models Q$$

- Konstruirajmo tablicu istinitosti:

P	Q	$P \vee Q$	$\neg P$	Q
⊥	⊥	⊥	⊤	⊥
⊥	⊤	⊤	⊤	⊤
⊤	⊥	⊤	⊥	⊥
⊤	⊤	⊤	⊥	⊤

- Premise imaju samo jedan model. To je ujedno i model formule Q , pa je Q po definiciji logička posljedica danih premsa

Dokaz logičke posljedice (2)

Dokaz opovrgavanjem (engl. *refutation method*)

Formula G je logička posljedica premsa F_1, F_2, \dots, F_n ako i samo ako je

$$F_1 \wedge \dots \wedge F_n \wedge \neg G$$

proturječna formula (kontradikcija). Drugim riječima:

$$F_1 \wedge \dots \wedge F_n \models G$$

ako i samo ako

$$\models \neg(F_1 \wedge \dots \wedge F_n \wedge \neg G)$$

Intuitivno, da bi formula G bila logička posljedica premsa, ne može biti da su premsi istinite, a da G nije istinita

Dokaz logičke posljedice (1)

Izravan dokaz (engl. *direct method*)

Formula G je logička posljedica premsa F_1, F_2, \dots, F_n ako i samo ako je

$$(F_1 \wedge \dots \wedge F_n) \rightarrow G$$

valjana formula (tautologija). Drugim riječima:

$$F_1 \wedge \dots \wedge F_n \models G$$

ako i samo ako

$$\models (F_1 \wedge \dots \wedge F_n) \rightarrow G$$

Gornja tvrdnja zapravo je **teorem semantičke dedukcije** (engl. *semantic deduction theorem*)

Dokaz logičke posljedice – primjer

- Dokažimo: $P \vee Q, \neg P \models Q$

- Izravan dokaz:

P	Q	$P \vee Q$	$\neg P$	$\overbrace{(P \vee Q) \wedge \neg P}^F$	$F \rightarrow Q$
⊥	⊥	⊥	⊤	⊥	⊤
⊥	⊤	⊤	⊤	⊤	⊤
⊤	⊥	⊤	⊥	⊥	⊤
⊤	⊤	⊤	⊥	⊥	⊤

- Dokaz opovrgavanjem:

P	Q	$P \vee Q$	$\neg P$	$\overbrace{(P \vee Q) \wedge \neg P}^F$	$\neg Q$	$F \wedge \neg Q$
⊥	⊥	⊥	⊤	⊥	⊤	⊥
⊥	⊤	⊤	⊤	⊤	⊥	⊥
⊤	⊥	⊤	⊥	⊥	⊤	⊥
⊤	⊤	⊤	⊥	⊥	⊥	⊥

Tri zadatka zaključivanja

- (1) **Provjera modela** (engl. *model checking*): je li zadana interpretacija I model za formulu F , tj. je li formula F istinita uz interpretaciju I ?
- (2) **Provjera zadovoljivosti ili SAT-problem** (engl. *satisfiability checking*): ima li zadana formula F model, tj. postoji li i jedna interpretacija u kojoj je F istinita?

Problem se također naziva **izgradnja modela** (engl. *model building*).

- (3) **Provjera valjanosti** (engl. *validity checking*): je li zadana formula F valjana, tj. istinita za svaku moguću interpretaciju?

- Zašto nismo spomenuli problem dokazivanja logičke posljedice?
Zato što se on svodi na problem provjere valjanosti (prema teoremu semantičke dedukcije)

Q: Jesu li navedeni problemi računalno teški?

Složenost zaključivanja (2)

- U propozicijskoj logici, SAT-problem je **odlučiv**: postupkom od konačno mnogo koraka možemo utvrditi ima li zadana formula model
- Prema odlučiv, problem je **netrakabilan**: složenost je $\mathcal{O}(2^n)$, gdje je n broj propozicijskih varijabli (trebamo ispitati 2^n redaka tablice)
- Zapravo, SAT-problem jedan je od prvih poznatih **NP-potpunih** problema (Cook, 1971)
- Nažalost, u predikatnoj logici (i mnogim drugim logikama) SAT-problem **nije odlučiv**: ne postoji algoritam koji bi mogao provjeriti zadovoljivost svih formula
- Algoritmi koji rješavaju SAT-problem nazivaju se **SAT-rješavači** (engl. *SAT-solvers*). U propozicijskoj logici mogu riješiti sve probleme, ali u predikatnoj samo neke probleme (zbog neodlučivosti)

Složenost zaključivanja (1)

- Problem provjere modela vrlo je jednostavan. Za propozicijsku logiku, taj je problem **odlučiv**: za svaku formulu F i interpretaciju I dobivamo u konačno mnogo koraka odgovor je li I model od F
- Problem provjere modela odlučiv je također i za predikatnu logiku (uz neka manja ograničenja)
- Alat koji radi provjeru modela zove se **provjernik modela** (engl. *model checker*)
- Problemi provjere valjanosti i zadovoljivosti (SAT) su teži! Zapravo, ta dva problema jednako su teška, jer su to dualni problemi:

Formula F je valjana akko $\neg F$ nije zadovoljiva

- Dakle, ako imamo algoritam koji rješava SAT-problem, možemo ga koristiti i za provjeru valjanosti, a to onda znači da možemo dokazivati logičke posljedice

Primjer: Diplomatski problem (1)

Diplomatski problem

Kao predstavnik protokola, zaduženi ste poslati pozivnice za diplomatski bal koji se održava u ambasadi. Međutim, postoje ograničenja:

- (1) Veleposlanik želi da, ako pozovete Tursku, svakako pozovete i UK
- (2) Pomoćnik veleposlanika želi da pozovete Tursku ili Argentinu, ili obje.
- (3) Zbog nedavnog diplomatskog incidenta, ne možete pozvati i UK i Argentinu.

Koga ćete pozvati?

- Logički prikaz problema:
$$(T \rightarrow U) \wedge (T \vee A) \wedge \neg(U \wedge A)$$
- **Q:** O kojem se zadatku ovdje radi (provjera modela, provjera zadovoljivosti, provjera valjanosti)?



Primjer: Diplomatski problem (2)

- Radi se o zadatku provjere zadovoljivosti. Zanima nas postoji li model za zadanu formulu (i koji je to model)
- Trebamo ispitati 2^3 interpretacija
- Rješenje (ukupno 2 modela):
 $I(U) = \top, I(T) = \top, I(A) = \perp$
 $I(U) = \perp, I(T) = \perp, I(A) = \top$
- O kojem bi se zadatku radilo da je pitanje bilo:
 - ▶ "Mogu li pozvati Veliku Britaniju i Tursku, a ne pozvati Argentinu?"
 - ▶ "Da li u svakom slučaju pozivam Veliku Britaniju?"
 - ▶ "Ako pozovem Tursku, onda zovem i Argentinu?"

Teorija dokaza (engl. proof theory)

- Ljudi ne zaključuju na način da dokazuju semantičku posljedicu (ne pokušavaju pronaći interpretaciju koja je istinita za F , a nije za G)!
- Umjesto toga, pokušavamo pokazati kako se G može **izvesti** iz premsa pomoću konačnog broja **pravila zaključivanja** (engl. *inference rules*)
- Svako pravilo zaključivanja treba biti **opravdano** i što jednostavnije
- Pravila zaključivanja omogućuju dobivanje novih formula na temelju zadanih premsa, bez eksplicitnog referenciranja na semantiku logike (istinosne vrijednosti propozicija)



Problem s dokazivanjem semantičke posljedice

- Imamo dva problema
- Prvi problem je **netraktabilnost**: moramo provjeriti 2^n interpretacija
- Zamislimo da želimo dokazati $F_1, \dots, F_{100} \models F_1$. Trebali bismo provjeriti 1.27×10^{30} interpretacija. To je nemoguće!
- Osim toga, u ovom slučaju to je nepotrebno jer je *očigledno* da relacija logičke posljedice vrijedi
- Drugi problem je **neodlučivost**: u propozicijskoj logici broj interpretacija je konačan, ali u predikatnoj neće biti, pa nemamo šanse ispitati sve interpretacije
- Postoji li rješenje za ove probleme? Da, donekle
- Umjesto da se bavimo interpretacijama i modelima (semantikom), trebamo se baviti pravilima zaključivanja (teorijom dokaza)

Deduktivna posljedica

Deduktivna posljedica

Formula G je **dedukcija** (engl. *deduction*) ili **deduktivna posljedica** (engl. *deductive consequence*) formula F_1, F_2, \dots, F_n akko je G moguće **izvesti** (engl. *derive*) iz premsa F_1, F_2, \dots, F_n pravilima zaključivanja.

Pišemo $F_1, F_2, \dots, F_n \vdash G$ i čitamo " F_1, \dots, F_n **izvodi** (engl. *derives*) ili **deduktivno povlači** (engl. *deductively entails*) G ".

Teorem

Formula G je **teorem** akko vrijedi $\vdash G$, tj. ako je formula G deduktivno izvediva iz praznog skupa premsa.

- Dokazati $F \vdash G$ je ekvivalentno kao dokazati $\vdash F \rightarrow G$
- Zato umjesto o izvođenju dedukcije govorimo o **dokazivanju teorema**

Dokazivanje teorema

- U umjetnoj inteligenciji zanima kako automatizirati dokazivanje
- Time se bavi područje **automatskog zaključivanja** (engl. *automated reasoning*) ili **automatskog dokazivanja teorema** (engl. *automated theorem proving*, ATP)
- U teoriji dokaza razvijeni su različiti **postupci dokazivanja** (engl. *proof methods*)
- Sustav koji implementira neki postupak dokazivanja naziva se **dokazivač teorema** (engl. *theorem prover*)
- Dokazivači teorema razlikuju se od provjernika modela po tome što se oslanjaju samo na "sintaktičke manipulacije simbola" (ne zamaraju se modelima i semantikom)
- Naravno, to što dokazivači teorema izvode mora biti **semantički ispravno**, dakle opravdivo u smislu modela

Ispravnost i potpunost pravila

Ispravnost

Pravilo zaključivanja je **ispravno** (engl. *sound*) ako, primjenjeno na skup premisa, izvodi formulu koja je **logička posljedica** tih premsisa.

Formalno, pravilo zaključivanja r je ispravno ako i samo ako

$$\text{ako } F_1, \dots, F_n \vdash_r G \text{ onda } F_1, \dots, F_n \models G$$

Potpunost

Skup pravila R je **potpun** (engl. *complete*) ako i samo ako je njime moguće izvesti sve logičke posljedice:

$$\text{ako } F_1, \dots, F_n \models G \text{ onda } F_1, \dots, F_n \vdash_R G$$

Ispravnost i potpunost povezuju semantiku i teoriju dokaza
(povezuju u oba smjera relacije \vdash i \models)

Pravila zaključivanja

- Primjer pravila zaključivanja:

"Ako su dvije tvrdnje istinite, onda je istinita i njihova konjunkcija"

Pravilo konjunkcije

$$\frac{A \quad B}{A \wedge B} \quad \text{ili} \quad A, B \vdash A \wedge B$$

- Što je sa sljedećim pravilom?

$$A \vee B \vdash A$$

- Intuitivno, prvo pravilo je **semantički ispravno**, a drugo nije
- Da bi pravilo bilo ispravno, deduktivan zaključak koji njime izvodimo mora biti **logička posljedica premisa**

Ispravnost i potpunost – primjer

- Dokažimo da je pravilo $F \rightarrow G, F \vdash G$ (**modus ponens**) ispravno. Trebamo dokazati $F \rightarrow G, F \models G$. Izravan dokaz:

F	G	$F \rightarrow G$	$(F \rightarrow G) \wedge F$	$((F \rightarrow G) \wedge F) \rightarrow G$
⊥	⊥	⊤	⊥	⊤
⊥	⊤	⊤	⊥	⊤
⊤	⊥	⊥	⊥	⊤
⊤	⊤	⊤	⊤	⊤

- Dokažimo da pravilo $F \rightarrow G, G \vdash F$ (**abdukcija**) nije ispravno. Trebamo dokazati $F \rightarrow G, G \not\models F$. Izravan dokaz:

F	G	$F \rightarrow G$	$(F \rightarrow G) \wedge G$	$((F \rightarrow G) \wedge G) \rightarrow F$
⊥	⊥	⊤	⊥	⊤
⊥	⊤	⊤	⊤	⊤
⊤	⊥	⊥	⊤	⊥
⊤	⊤	⊤	⊤	⊤

Postupci dokazivanja

- Automatsko zaključivanje koristi neki od postupaka dokazivanja
- Taj postupak mora biti semantički ispravan, a poželjno je da je potpun
- Razvijeni su mnogi postupci:
 - ▶ sustavi prirodnog zaključivanja (engl. *natural deduction systems*)
 - ▶ aksiomatski (hilbertovski) sustavi
 - ▶ sekventni računi (engl. *sequent calculi*)
 - ▶ tableau-metoda
 - ▶ metoda rezolucije (engl. *resolution method*)

Mi ćemo se usredotočiti na metodu rezolucije. Ta je metoda ispravna i potpuna te se lako implementira na računalu.

Metoda rezolucije

- **Metoda rezolucije** (metoda razrješavanja) koristi se u propozicijskoj logici i predikatnoj logici prvog reda
- Metodu je predložio J. Robinson 1965. godine
- Metoda se sastoji od samo jednog pravila zaključivanja:

Rezolucijsko pravilo

$$\frac{A \vee F \quad \neg A \vee G}{F \vee G} \quad \text{ili} \quad A \vee F, \neg A \vee G \vdash F \vee G$$

Što je ekvivalentno s:

$$\neg F \rightarrow A, A \rightarrow G \vdash \neg F \rightarrow G$$

- Prednost je što radimo sa samo jednim pravilom, a to bitno pojednostavljuje automatsko zaključivanje

Pravila prirodnog zaključivanja

- Pravila prirodnog zaključivanja (engl. *natural deduction*) (Gentzen, 1935) najbliža su su "prirodnom" načinu zaključivanja
- Pravila ima desetak (ovisno o verziji). Neka od pravila:

(1)	$F, G \vdash F \wedge G$	– uvođenje konjukcije
(2)	$F \wedge G \vdash F$	– eliminacija konjukcije
(3)	$F \vdash F \vee G$	– uvođenje disjunkcije
(4)	$F, F \rightarrow G \vdash G$	– modus ponens
(5)	$\neg G, F \rightarrow G \vdash \neg F$	– modus tollens
(6)	$F \rightarrow G, G \rightarrow H \vdash F \rightarrow H$	– ulančavanje (silogizam)
(7)	$F, F \leftrightarrow G \vdash G$	
	⋮	

- Pravila nisu prikladna za automatsko zaključivanje jer ih je previše (iziskuju složenu upravljačku strukturu)

Pravilo rezolucije – ispravnost

- Uvjerimo se da je pravilo rezolucije **ispravno**
- Trebamo dokazati da je deduktivna posljedica rezolucijskog pravila također i logička posljedica, tj. trebamo dokazati:

$$A \vee F, \neg A \vee G \models F \vee G$$

- Npr. izravnom metodom:

A	F	G	$\overbrace{A \vee F}^P$	$\neg A$	$\overbrace{\neg A \vee G}^Q$	$P \wedge Q$	$\overbrace{F \vee G}^R$	$(P \wedge Q) \rightarrow R$
T	T	T	T	⊥	T	T	T	T
T	T	⊥	T	⊥	⊥	⊥	T	T
T	⊥	T	T	⊥	T	T	T	T
T	⊥	⊥	T	⊥	⊥	⊥	⊥	T
⊥	T	T	T	T	T	T	T	T
⊥	T	⊥	T	T	T	T	T	T
⊥	⊥	T	⊥	T	T	⊥	T	T
⊥	⊥	⊥	⊥	T	T	⊥	⊥	T

Klauzula

- Rezolucijsko pravilo može se primijeniti samo na disjunkcije
- Ako želimo primjenjivati isključivo rezolucijsko pravilo, premise trebaju biti u obliku disjunkcije. Takav oblik nazivamo **klauzula**

Klauzula (engl. *clause*)

Literal je atom ili njegova negacija. **Klauzula** je disjunkcija konačnog broja literala G_i :

$$G_1 \vee G_2 \vee \cdots \vee G_n, \quad n \geq 0$$

Klauzula koja sadrži samo jedan literal naziva se **jedinična klauzula** (engl. *unit clause*).

- Primjeri literala: $A, F, \neg A, \neg F, G, \neg G$
- Primjeri klauzula: $A \vee F, \neg A \vee G, A \vee \neg B \vee C \vee \neg D, F$

Konjunktivna normalna forma

- Ako su premise klauzule, skup premsa je **konjunkcija klauzula** (premise su implicitno povezane operatorom \wedge)
- **Q:** Ograničava li to primjenu rezolucije? **A:** Ne!
- Bilo koju formulu propozicijske logike moguće je prikazati kao konjunkciju klauzula pretvorbom u **konjunktivnu normalnu formu**

Konjunktivna normalna forma (engl. *conjunctive normal form*, CNF)

Formula F je u **konjunktivnoj normalnoj formi** akko je F u obliku

$$F_1 \wedge F_2 \wedge \cdots \wedge F_n$$

pri čemu je F_i oblika

$$G_{i1} \vee G_{i2} \vee \cdots \vee G_{im}$$

gdje su G_{ij} literalni (atomi ili njihove negacije).

Rezolucija nad klauzulama

Rezolucijsko pravilo nad klauzulama

$$\frac{F_1 \vee \cdots \vee F_i \vee \cdots \vee F_n}{F_1 \vee \cdots \vee F_{i-1} \vee F_{i+1} \vee \cdots \vee F_n \vee G_1 \vee \cdots \vee G_{i-1} \vee G_{i+1} \vee \cdots \vee G_n} \quad \frac{G_1 \vee \cdots \vee G_i \vee \cdots \vee G_m}{G_1 \vee \cdots \vee G_{i-1} \vee G_{i+1} \vee \cdots \vee G_n}$$

gdje su F_i i G_i **komplementarni literali** (jedan je negacija drugoga).

Premise nazivamo **roditeljske klauzule**, a dedukciju nazivamo **rezolventa**.

- Primjeri:

$$A \vee B \vee \neg C, D \vee \neg B \vee E \vdash A \vee \neg C \vee D \vee E$$
$$\neg A \vee B, A \vdash B$$

$$A \vee B, A \vee \neg B \vdash A \vee A$$

$$A \vee B, \neg A \vee \neg B \vdash B \vee \neg B$$

$$A \vee B, \neg A \vee \neg B \vdash A \vee \neg A$$

Pretvorba u CNF

- Svaka se formula može pretvoriti u CNF u četiri slijedna koraka

Pretvorba formule u CNF

$$(1) \text{ Uklanjanje ekvivalencije: } F \leftrightarrow G \equiv (\neg F \vee G) \wedge (\neg G \vee F)$$

$$(2) \text{ Uklanjanje implikacije: } F \rightarrow G \equiv \neg F \vee G$$

$$(3) \text{ Potiskivanje negacije do atoma: } \neg(F \vee G) \equiv \neg F \wedge \neg G$$
$$\neg(F \wedge G) \equiv \neg F \vee \neg G$$

$$(4) \text{ Primjena distributivnosti: } F \vee (G \wedge H) \equiv (F \vee G) \wedge (F \vee H)$$
$$F \wedge (G \vee H) \equiv (F \wedge G) \vee (F \wedge H)$$

Svaki se korak ponavlja sve dok je primjenjiv.

U svim koracima, kad god je to moguće, primjenjuje se ekvivalencija za involuciju $\neg\neg F \equiv F$.

Pretvorba u CNF – primjer

$$(C \vee D) \rightarrow (\neg A \leftrightarrow B)$$

(1) Uklanjanje ekvivalencije:

$$(C \vee D) \rightarrow (\neg A \leftrightarrow B)$$

$$(C \vee D) \rightarrow ((\neg \neg A \vee B) \wedge (\neg B \vee \neg A))$$

(2) Uklanjanje implikacije:

$$(C \vee D) \rightarrow ((A \vee B) \wedge (\neg B \vee \neg A))$$

(3) Potiskivanje negacije:

$$\neg(C \vee D) \vee ((A \vee B) \wedge (\neg B \vee \neg A))$$

(4) Primjena distributivnosti:

$$(\neg C \wedge \neg D) \vee ((A \vee B) \wedge (\neg B \vee \neg A))$$

$$((\neg C \wedge \neg D) \vee (A \vee B)) \wedge ((\neg C \wedge \neg D) \vee (\neg B \vee \neg A))$$

$$((\neg C \vee A \vee B) \wedge (\neg D \vee A \vee B)) \wedge ((\neg C \wedge \neg D) \vee (\neg B \vee \neg A))$$

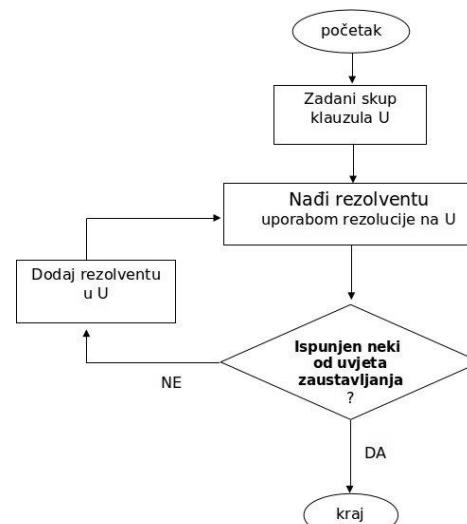
$$((\neg C \vee A \vee B) \wedge (\neg D \vee A \vee B)) \wedge ((\neg C \vee \neg B \vee \neg A) \wedge (\neg D \vee \neg B \vee \neg A))$$

$$(\neg C \vee A \vee B) \wedge (\neg D \vee A \vee B) \wedge (\neg C \vee \neg B \vee \neg A) \wedge (\neg D \vee \neg B \vee \neg A)$$

Rezolucija

Postupak se ponavlja sve dok:

- (1) izvedena je ciljna formula
- (2) ne može se izvesti nova formula
- (3) iscrpljeni su računalni resursi



Klauzalni oblik

- Formula u konjunktivnoj normalnoj formi može se prikazati kao skup klauzula između kojih se implicitno podrazumijeva konjunkcija
- Klauzule se mogu prikazati kao skup literalova između kojih se implicitno podrazumijeva disjunkcija
- Dakle, formula se može prikazati kao **skup skupova literalova**
- To nazivamo **klauzalni oblik**
- Npr.:
$$(\neg C \vee A \vee B) \wedge (\neg D \vee A \vee B) \wedge (\neg C \vee \neg B) \Rightarrow \{\{\neg C, A, B\}, \{\neg D, A, B\}, \{\neg C, \neg B\}\}$$
- Klauzule se također mogu pisati jedna ispod druge:
$$\begin{aligned} &\neg C \vee A \vee B \\ &\neg D \vee A \vee B \\ &\neg C \vee \neg B \end{aligned}$$

Rezolucija – primjer

- Dokažimo rezolucijom: $A \rightarrow B, B \rightarrow C, A \vdash C$
- Premise u klauzalnom obliku:
 - (1) $\neg A \vee B$
 - (2) $\neg B \vee C$
 - (3) A
- Rezolucijskim postupkom izvodimo:
 - (4) $\neg A \vee C$ (iz 1 i 2)
 - (5) C (iz 3 i 4)
- Ili:
 - (4') B (iz 1 i 3)
 - (5') C (iz 2 i 4')

Nepotpunost rezolucije

- Dokazali smo da je rezolucijsko pravilo ispravno. No je li potpuno?
- Lako je pokazati da rezolucijsko pravilo nije potpuno
- Npr., razmotrimo dedukciju $F \vdash F \vee G$
- Nju ne možemo izvesti rezolucijskim pravilom (zašto?)
- Međutim, vrijedi $F \models F \vee G$ (provjerite!)
- Budući da vrijedi $F \models F \vee G$, a da rezolucijskim pravilom ne možemo deduktivno izvesti $F \vdash F \vee G$, zaključujemo da rezolucijskim pravilom ne možemo dokazati sve logičke posljedice, pa zaključujemo da **rezolucijsko pravilo nije potpuno**
- Rezolucija koju smo do sada primjenjivali je **izravna rezolucija** (engl. *direct resolution*). Izravna rezolucija nije potpuna
- Postoji i **rezolucija opovrgavanjem** (engl. *refutation resolution*). Rezolucija opovrgavanjem jest **potpuna**

Rezolucija opovrgavanjem (2)

- Dokazano je (**ground resolution theorem**) da uvijek kada je skup klauzula proturječan, rezolucijom možemo izvesti klauzulu NIL
- To znači da uvijek možemo dokazati nekonzistentnost skupa klauzula
- A to znači da možemo dokazati svaku logičku posljedicu. **Q:** Zašto?
- **A:** Zato što $F \models G$ možemo dokazati metodom opovrgavanja tako da dokažemo da je $F \wedge \neg G$ proturječna formula
- A to onda znači da je rezolucija opovrgavanjem potpuna, zato što njome možemo dokazati bilo koju logičku posljedicu
- Dakle, rezolucija opovrgavanjem je **ispravna i potpuna!**
- Umjesto da koristimo izravnu rezoluciju, trebamo negirati ciljnu formulu G , dodati je premisama F_1, \dots, F_n i pokušati izvesti NIL
- Ako uspijemo, formula G je logička posljedica premlisa, inače nije

Rezolucija opovrgavanjem (1)

- Umjesto da dokazujemo $F_1, \dots, F_n \models G$, nastojimo dokazati da je $F_1 \wedge \dots \wedge F_n \wedge \neg G$ proturječna formula
- Kao poseban slučaj rezolucijskog pravila imamo:

$$\frac{A \quad \neg A}{\text{NIL}}$$

- NIL označava praznu klauzulu čija je semantička vrijednost \perp
- Ako rezolucijskim zaključivanjem izvedemo klauzulu NIL, onda znači da su premise proturječne. **Q:** Zašto?
- **A:** Zato što je rezolucijsko pravilo ispravno. Ako $F \vdash \text{NIL}$, onda $F \models \text{NIL}$. Ako $F \models \text{NIL}$, onda F mora biti proturječno. **Q:** Zašto?
- **A:** Zbog definicije logičke posljedice. Ako je logička posljedica proturječna formula, onda premise također moraju biti proturječne

Rezolucija opovrgavanjem – primjer 1

- Pokažimo da rezolucijom opovrgavanjem možemo dokazati $F \vdash F \vee G$:
- Negacija ciljne formule: $\neg(F \vee G) \equiv \neg F \wedge \neg G$
- Skup klauzula:
 - (1) F
 - (2) $\neg F$
 - (3) $\neg G$
- Iz (1) i (2) izvodimo klauzulu NIL
- Dokazali smo da je skup klauzula proturječan, odnosno da je $F \vee G$ deduktivna/logička posljedica premise F



Rezolucija opovrgavanjem – primjer 2

- Vratimo se na diplomatski problem. Premise su:

$$(T \rightarrow U) \wedge (T \vee A) \wedge \neg(U \wedge A)$$

- Pretvorba u klauzalni oblik:

- (1) $U \vee \neg T$
- (2) $T \vee A$
- (3) $\neg U \vee \neg A$

- Dokažimo: "Ako pozovem Argentinu, neću pozvati Tursku": $A \rightarrow \neg T$

- Negacija cilja: $\neg(T \rightarrow \neg A) \equiv \neg(\neg T \vee \neg A) \equiv T \wedge A$

- Nove klauzule:

- (4) T
- (5) A

- Rezolucijski postupak:

- (6) U (iz 1 i 4)
- (7) $\neg A$ (iz 3 i 6)
- (8) NIL (iz 5 i 7)

Algoritam rezolucije opovrgavanjem

Algoritam rezolucije opovrgavanjem (za propozicijsku logiku)

```
function plResolution( $F, G$ )
  clauses  $\leftarrow$  cnfConvert( $F \wedge \neg G$ )
  new  $\leftarrow \emptyset$ 
  loop do
    for each  $(c_1, c_2)$  in selectClauses(clauses) do
      resolvents  $\leftarrow$  plResolve( $c_1, c_2$ )
      if NIL  $\in$  resolvents then return true
      new  $\leftarrow$  new  $\cup$  resolvents
    if new  $\subseteq$  clauses then return false
    clauses  $\leftarrow$  clauses  $\cup$  new
```

- cnfConvert – pretvara formulu u konjunktivan normalan oblik
- selectClauses – odabire skup parova klauzula za razrješavanje
- plResolve – razrješava roditeljske klauzule i vraća skup rezolventi

Faktorizacija

- Rezolucija opovrgavanjem je potpuna uz uvjet da su klauzule **faktorizirane**
- Faktorizacija je primjena ekvivalencije $G \vee G \equiv G$ kojom se višekratno pojavljivanje istog literala zamjenjuje jednim literalom
- Primjer:
 $\neg A \vee \neg A, A \vee A \vdash A \vee \neg A$
- Skup klauzula je proturječan, a izveli smo valjanu formulu
- Q: Je li to ispravno? A: Naravno da jest. Valjana formula je logička posljedica bilo koje formule. Ali od toga nemamo koristi.
- Međutim, da smo napravili faktorizaciju, dobili bismo:
 $\neg A, A \vdash \text{NIL}$
- Kako bismo zadržali potpunost, treba primjenjivati faktorizaciju kad god je to moguće

Algoritam rezolucije opovrgavanjem – napomene

- Primjena rezolucijskog pravila na par klauzula može dati više rezolventi, pa zato radimo sa skupom rezolventi
- Kako bi se zadržala potpunost, potrebno je uvijek **faktorizirati** sve dobivene rezolvente
- Broj mogućih različitih klauzula je konačan (ako se provodi faktorizacija), pa algoritam sigurno završava u konačnom broju koraka
- Treba voditi računa o tome koji su parovi već bili razrješeni i ne razrješavati ih ponovo
- Izvođenje klauzule NIL iz skupa klauzula zapravo je **problem pretraživanja**: u svakom koraku trebamo odabrati par klauzula koje ćemo razrješiti
- Treba nam **strategija pretraživanja**, koja se u kontekstu rezolucije naziva **rezolucijska strategija**

- Dvije vrste **rezolucijskih strategija**:
 - ▶ strategije pojednostavljivanja (engl. *simplification strategies*)
 - ▶ upravljačke strategije (engl. *control strategies*)
- Strategije pojednostavljivanja uklanjuju redundantne i nevažne klauzule generirane tijekom postupka dokazivanja čime se sprječava njihovo daljnje nepotrebno razrješavanje
- Upravljačke strategije određuju način odabira roditeljskih klauzula
- Rezolucijska strategija treba biti **potpuna**: mora izvesti NIL, ako je skup klauzula nekonzistentan
- To ne treba brkati s potpunošću pravila zaključivanja (općenito, da bi postupak dokazivanja bio potpun, trebamo kombinirati potpuna pravila s potpunom strategijom pretraživanja)

Upravljačke rezolucijske strategije

Strategija zasićenja po razinama (engl. *level saturation strategy*)

- Rezolvente izvodimo razinu po razinu (kao kod pretraživanja u širinu): razrješavamo sve moguće parove klauzula na prvoj razini (početni skup klauzula), zatim na drugoj razini, itd.
- Na i -toj razini, roditeljske klauzule uzimaju se s razina 1 do $(i - 1)$
- Ovo je potpuna strategija, ali je vrlo neučinkovita (problem kombinatorne eksplozije)

Strategija skupa potpore (engl. *set-of-support strategy*)

- **Skup potpore** je skup klauzula nastao negacijom ciljne formule i naknadnim dodavanjem izvedenih klauzula
- Barem jedna roditeljska klauzula uvijek mora biti iz skupa potpore
- Strategija je potpuna i u pravilu učinkovitija od strategije zasićenja (to više što je skup popore manji)

Strategija brisanja

Uklanjanje redundantnih klauzula:

- Klauzula koja je **pokrivena** (engl. *subsumed*) drugom klauzulom može se obrisati
- Prema ekvivalenciji apsorpcije: $F \wedge (F \vee G) \equiv F$
- Ako se u skupu klauzula nađe par klauzula C_1 i C_2 takvih da $C_1 \subseteq C_2$, klauzula C_2 može se obrisati (klauzule su prikazane kao skupovi literala)

Uklanjanje nevažnih klauzula:

- Klauzula koja je **valjana** (tautologija) je nevažna (zašto?)
- Ako je rezolventa valjana klauzula, može ju se odmah pobrisati
- Provjera valjanosti klauzule je jednostavna: klauzula je valjana akko sadrži komplementaran par literalja F_i i $\neg F_i$

Sažetak

- Postoje razne vrste logike koje se međusobno razlikuju po **ekspresivnosti**. Propozicijska logika je najjednostavnija
- Svaki logički sustav sastoji se **(1) sintakse, (2) semantike i (3) teorije dokaza**
- **Logička posljedica** je zaključak koji semantički slijedi iz premlisa, a **deduktivna posljedica** je zaključak koji izvodimo pravilima zakl.
- Pravila zaključivanja moraju biti **ispravna** i poželjno je da su **potpuna**
- Tri zadatka zaključivanja su **(1) provjera modela, (2) provjera zadovoljivosti i (3) provjera valjanosti**. Prvi je jednostavan, druga dva su netraktabilna, ali barem **odlučiva** u propozicijskoj logici
- **Rezolucija opovrgavanjem** (uz faktorizaciju) je potpuna metoda dokazivanja



Sljedeća tema: Dokazivanje teorema u logici prvoga reda

Prof.dr.sc. Bojana Dalbelo Bašić

Fakultet elektrotehnike i računarstva
Zavod za elektroniku, mikroelektroniku, računalne i inteligentne sustave

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

UMJETNA INTELIGENCIJA

Zaključivanje uporabom predikatne logike (1)



© Bojana Dalbelo Bašić

FER

1



AUTOMATSKO ZAKLJUČIVANJE UPORABOM PREDIKATNE LOGIKE

- PROPOZICIJSKA LOGIKA << LJUDSKO ZAKLJUČIVANJE
- *Premise:*
Svaki student pohađa predavanja.
Ivan je student.
- *Intuitivno zaključujemo:*
Ivan pohađa predavanja.
- Ovakvo jednostavno zaključivanje nije moguće u propozicijskoj logici.



2



SINTAKSA PREDIKATNE LOGIKE

- **Ontologija?**
- **Ontološka prepostavka** predikatne logike prvog reda:
 - postoje:
 - **Objekti**
 - **Relacije između objekata**
 - Svojstva objekata: 1-mjesne relacije
 - Propozicije: 0-mjesne relacije
- Jače ontološke prepostavke od onih propozicijske logike
 - Propozicijska logika: postoje sudovi koji su istiniti/lažni
 - Zbog toga: **veća ekspresivnost od propozicijske logike!**
- Naprednije logike imaju još jače ontološke prepostavke!

3



4



- Skup elemenata nad kojim se izvodi zaključivanje uporabom predikatne logike naziva se **domena**
- Elementi domene označeni su posebnim imenima i nazivaju se **konstante**
 - Mala slova s početka abecede: a, b, c, \dots ili nizovi znakova s velikim početnim slovom: $Ivan, Ana, \dots$
 - *Primjer:* Domena može biti skup cijelih brojeva Z , tada su $1, 2, 3, \dots$ konstante
- **Varijable** se označavaju simbolima u, v, w, x, y, \dots i mogu poprimiti bilo koju vrijednost iz domene
- **Funkcije** preslikavaju jedan ili više elemenata domene u taj isti skup
 - *Primjer:* Funkcija add preslikava dva elementa domene u njihov zbroj. Dakle, $add(2, 3) = 5$

5

- *Definicija*
- **Predikati** su "funkcije" koje preslikavaju jedan ili više elemenata domene u jednu od vrijednosti istinitosti: *istinu* ili *laž*
- Predikati nam govore o svojstvima elemenata domene ili o njihovim međusobnim odnosima.
- *Primjeri* predikata:
 - Neka je domena skup cijelih brojeva Z .
 - Predikati su: $ODD(x)$, $EVEN(x)$, $GT(u, v)$.
 - $ODD(6) \equiv \text{laž}$
 - $EVEN(6) \equiv \text{istina}$
 - $GT(add(1, 2), 4) \equiv \text{laž}$

6

Konstante, varijable, funkcije i predikati čine četiri disjunktna skupa u predikatnoj logici.

Definicija

- **Izraz** (engl. term) je definiran rekurzivno:
 - konstanta je izraz
 - varijabla je izraz
 - $f(t_1, t_2, \dots, t_n)$ je izraz akko je f funkcija od n argumenata, gdje su t_1, t_2, \dots, t_n izrazi
- *Primjer:* Izrazi su $2, 3, add(3, 4), add(v, add(1, 4))$

7

Definicija

- $P(t_1, t_2, \dots, t_n)$ je **atom** akko P označava predikat od n argumenata, a t_1, t_2, \dots, t_n su izrazi

Primjer

- $GT(add(1, 2), 4)$ je atom

FORMULE

- Za izgradnju formula u predikatnoj logici koriste se isti logički veznici kao i u propozicijskog logici $\sim, \wedge, \vee, \rightarrow, \leftrightarrow$

Primjer

- $(ODD(3) \wedge GT(5, 2))$ čini formulu

8

SINTAKSA PREDIKATNE LOGIKE

- U predikatnoj logici još se koriste dva posebna simbola:
 - \forall univerzalni kvantifikator (čita se "za svaki"),
 - \exists egzistencijalni kvantifikator (čita se "postoji").

Primjer

- Formula $\forall x(GT(add(x,1),x))$ se interpretira: "za svaki x domene vrijedi da je $x+1$ veći od x "
- Za formulu u gornjem primjeru se kaže da je pojavljivanje varijable x **ograničeno** (kvantificirano) sa $\forall x$
- Kažemo da je varijabla **vezana** ako je negdje ograničena kvantifikatorom, a **slobodna** ako to negdje nije



9

SINTAKSA PREDIKATNE LOGIKE

- Podformula $(GT(add(x,1),x))$ zove se **doseg** (područje djelovanja, djelokrug, dohvati, domena) od $\forall x$ jer se na tu formulu odnosi $\forall x$

Primjer

- $\exists y(GT(y, 4))$
- postoji (*postoji barem jedan ili za neki*) element domene y tako da je y veći od 4;
- y je vezana varijabla, tj. pojavljivanje y ograničeno je sa $\exists y$,
- doseg od $\exists y$ je $GT(y, 4)$



10

SINTAKSA PREDIKATNE LOGIKE

Primjer

- $\forall u(ODD(u) \rightarrow EVEN(add(u,1)))$
- za svaki element domene u vrijedi: ako je u neparan tada je $u+1$ paran.
- u je vezana varijabla, tj. pojavljivanje u ograničeno je sa $\forall u$,
- doseg $\forall u$ je $(ODD(u) \rightarrow EVEN(ADD(u,1)))$
- $\forall x(\exists y(GT(x, y)))$
- za svaki x iz domene postoji y iz domene tako da je x veće od y .
- pojavljivanje x ograničeno je sa $\forall x$, pojavljivanje y ograničeno je sa $\exists y$.
- doseg od $\forall x$ je $\exists y(GT(x, y))$, doseg od $\exists y$ je $GT(x, y)$ – doseg jednog kvantifikatora je unutar dosega drugog kvantifikatora!



11

SINTAKSA PREDIKATNE LOGIKE

Definicija

- Kaže se da je varijabla **vezana** u formuli akko je barem jedno pojavljivanje varijable ograničeno. Varijabla je **slobodna** u formuli akko barem jedno pojavljivanje nije ograničeno

Primjer

- U formuli $\forall x(GT(x,y))$, x je vezana, ali y je slobodna
- U formuli $(\forall x(GT(x,y)) \wedge \exists y(ODD(y)))$, y je ujedno i slobodna i vezana varijabla
 - Slobodna varijabla y nezavisna je od vezane



12

Formula u predikatnoj logici je definirana rekurzivno:

Definicija

- **Dobro oblikovana formula (wff)** gradi se na sljedeći način:
 1. atom je formula;
 2. ako je F formula tada je i $(\neg F)$ formula;
 3. ako su F i G formule tada su formule:
 - $(F \wedge G)$
 - $(F \vee G)$
 - $(F \rightarrow G)$
 - $(F \leftrightarrow G)$

13

Primjer :

- Formula $((\forall x) GT(x,y) \wedge (\exists y) ODD(y))$ može se napisati kraće kao $\forall x GT(x,y) \wedge \exists y ODD(y)$
- Zgrade uvijek moraju zatvarati argumente funkcije ili predikata

Zadatak

- Kako bi formulom predikatne logike izrazili da za svaki cijeli broj x vrijedi da je x paran ili je $x+1$ paran ?
- Je li izraz $\forall x(ODD(x \vee add(x+1)))$ dobro oblikovana formula?

15

4. Ako je F formula takva da **sadržava varijablu x koja u njoj nije vezana**, tada su formule:
 - $(\forall x) F$,
 - $(\exists x) F$.
 5. Nema drugih formula osim upravo definiranih
- **Dogovorno dopuštamo uklanjanje zagrade:**
 - u pravilu 2 (zgrade oko negirane formule)
 - u pravilu 3 ako su to **vanske zgrade**
 - u pravilu 4 (zgrade oko kvantifikatora)
 - Uočite da se definicije atoma i formule razlikuju u propozicijskoj i predikatnoj logici.

14

- Ovdje razmatrana predikatna logika je tzv. **PREDIKATNA LOGIKA PRVOG REDA** (engl. *First Order Predicate Logic - FOP*) u kojoj samo varijable mogu biti kvantificirane
- Ako je P predikat i f funkcija tada formule poput $\forall P(P(x))$ i $\exists f(f(x))$ nisu razmatrane u predikatnoj logici prvog reda

16



SEMANTIKA PREDIKATNE LOGIKE

- Za danu interpretaciju formuli se pridružuje odgovarajuća vrijednost istinitosti
- **Ne mogu se interpretirati formule koje sadrže slobodne varijable** pa ćemo od sada podrazumijevati da formule ne sadrže slobodne varijable
- Formule su **ekvivalentne** ako poprimaju istu vrijednost istinitosti za svaku moguću interpretaciju



- **Interpretacija formule** u predikatnoj logici sastoji se od sljedećeg:
 1. Određivanje elemenata domene. Svaki element domene označava se simbolom za konstantu.
 2. Definiranje preslikavanja f za svaku funkciju f od n argumenata $f(c_1, c_2, c_3, \dots, c_n)$, gdje c_i predstavlja konstante iz domene.
 3. Pridruživanje vrijednosti istinitosti svakom predikatu od n argumenata $P(t_1, t_2, t_3, \dots, t_n)$
n-torce za koje $P(t_1, t_2, t_3, \dots, t_n)$ zovemo **ekstenzija predikata**



TABLICA EKVIVALENCIJA PREDIKATNE LOGIKE

- $F(x)$ i $G(x)$ označavaju formule koje sadrže slobodnu varijablu x dok $H\{x\}$ označava formulu koja ne sadrži varijablu x
- | | | |
|--|---|--|
| [1] $\forall x F(x)$ | = | $\forall y F(y)$ |
| [2] $\exists x F(x)$ | = | $\exists y F(y)$ |
| [3] $\sim \forall x F(x)$ | = | $\exists x (\sim F(x))$ |
| [4] $\sim \exists x F(x)$ | = | $\forall x (\sim F(x))$ |
| [5] $(\forall x F(x) \vee \forall x G(x))$ | = | $(\forall x F(x) \vee \forall y G(y))$ |
| [6] $(\forall x F(x) \vee \exists x G(x))$ | = | $(\forall x F(x) \vee \exists y G(y))$ |



TABLICA EKVIVALENCIJA PREDIKATNE LOGIKE

[7]	$(\exists x F(x) \vee \forall x G(x))$	\equiv	$(\exists x F(x) \vee \forall y G(y))$
[8]	$(\exists x F(x) \vee \exists x G(x))$	\equiv	$(\exists x F(x) \vee \exists y G(y))$
[9]	$(\forall x F(x) \wedge \forall x G(x))$	\equiv	$(\forall x F(x) \wedge \forall y G(y))$
[10]	$(\forall x F(x) \wedge \exists x G(x))$	\equiv	$(\forall x F(x) \wedge \exists y G(y))$
[11]	$(\exists x F(x) \wedge \forall x G(x))$	\equiv	$(\exists x F(x) \wedge \forall y G(y))$
[12]	$(\exists x F(x) \wedge \exists x G(x))$	\equiv	$(\exists x F(x) \wedge \exists y G(y))$
[13]	$(\forall x F(x) \vee \forall y G(y))$	\equiv	$\forall x \forall y (F(x) \vee G(y))$
[14]	$(\forall x F(x) \wedge \forall y G(y))$	\equiv	$\forall x \forall y (F(x) \wedge G(y))$
[15]	$(\forall x F(x) \vee H\{x\})$	\equiv	$\forall x (F(x) \vee H\{x\})$
[16]	$(\forall x F(x) \wedge H\{x\})$	\equiv	$\forall x (F(x) \wedge H\{x\})$
[17]	$(\exists x F(x) \vee H\{x\})$	\equiv	$\exists x (F(x) \vee H\{x\})$
[18]	$(\exists x F(x) \wedge H\{x\})$	\equiv	$\exists x (F(x) \wedge H\{x\})$
[19]	$\forall x (F(x) \wedge G(x))$	\equiv	$(\forall x F(x) \wedge \forall x G(x))$
[20]	$\forall x (F(x) \wedge G(x))$	\equiv	$(\forall x F(x) \wedge \forall y G(y))$

21

TABLICA EKVIVALENCIJA PREDIKATNE LOGIKE

[21]	$\forall x (F(x) \wedge G(x))$	\equiv	$\forall x \forall y (F(x) \wedge G(y))$
[22]	$\exists x (F(x) \vee G(x))$	\equiv	$(\exists x F(x) \vee \exists x G(x))$
[23]	$\exists x (F(x) \wedge G(x))$	\equiv	$(\exists x F(x) \wedge \exists y G(y))$
[24]	$\exists x (F(x) \vee G(x))$	\equiv	$\exists x \exists y (F(x) \vee G(y))$

- Jesu li ekvivalencije formule [22]-[23] ako se umjesto kvantifikatora \exists koristi kvantifikator \forall ?

22

SEMANTIKA PREDIKATNE LOGIKE

Primjer 1

- Odredite vrijednosti istinitosti formule

$$\forall x \exists y (P(x) \wedge Q(x, f(y)))$$

za interpretaciju:

- Domena {a, b}
- $f(a) = b$ i $f(b) = a$ i sljedeće vrijednosti istinitosti atoma

P(a)	P(b)	Q(a,a)	Q(a,b)	Q(b,a)	Q(b,b)
false	true	true	true	false	true

- Za $x = a$, $P(x)$ je false.
 - Formula nije istinita za sve vrijednosti x iz domene.
- Dakle $\forall x \exists y (P(x) \wedge Q(x, f(y)))$ je laž

23

SEMANTIKA PREDIKATNE LOGIKE

Primjer 2

- Odredite vrijednosti istinitosti formule

$$\forall x \exists y (P(x) \wedge Q(x, f(y)))$$

za interpretaciju:

- Domena {1, 2}
- $f(1) = 2$ i $f(2) = 1$ i sljedeće vrijednosti istinitosti atoma

P(1)	P(2)	Q(1,1)	Q(1,2)	Q(2,1)	Q(2,2)
true	true	true	true	false	true

24

SEMANTIKA PREDIKATNE LOGIKE

- Evaluacija istinitosti:
 - Za $x = 1$, $P(1)$ je true.
 - $Q(x, f(y))$ je istinito za $y=1$ i $y=2$
- Za $x = 2$, $P(2)$ je true.
- $Q(x, f(y))$ je istinito za $y=1$
- Pokazali smo da za sve vrijednosti x iz domene postoji vrijednost y iz domene za koju je formula istinita
- Dakle, $\forall x \exists y (P(x) \wedge Q(x, f(y)))$ je istinita

25

POLUODLUČLJIVOST PREDIKATNE LOGIKE

- Te dvije metode temelje se na provjeri svih mogućih interinterpretacija formula. Kako u predikatnoj logici domena može biti beskonačna, (npr. skup cijelih brojeva), može biti i beskonačno mnogo interpretacija formule
- Dakle, u takvim slučajevima nije moguće provjeravati tautologiju ili kontradikciju formule pod svim mogućim interpretacijama pa se izravna metoda i metoda opovrgavanja **ne mogu primjeniti na predikatnu logiku**

27

POLUODLUČLJIVOST PREDIKATNE LOGIKE

- Podsjetimo se pojmove:
 - **tautologije, kontradikcije i konzistencije** formula propozicijske logike. Iste definicije vrijede i u predikatnoj logici.
- Kako bismo dokazali da je formula G je logička posljedica formula F_1, F_2, \dots, F_n , uveli smo dvije temeljne metode:
- Izravnu metodu
 - pokazujemo da je $((F_1 \wedge F_2 \wedge \dots \wedge F_n) \rightarrow G)$ tautologija
- Metoda opovrgavanja
 - pokazujemo da je $(F_1 \wedge F_2 \wedge \dots \wedge F_n \wedge \neg G)$ proturječe

26

POLUODLUČLJIVOST PREDIKATNE LOGIKE

- Važan matematički rezultat:

U predikatnoj logici **ne postoji općenit postupak** kojim se dokazuje da je G logička posljedica formule F (ili, ekvivalentno, da je formula $F \rightarrow G$ tautologija), ako ona to jest, odnosno kojim se dokazuje da G nije logička posljedica formule F , ako ona to nije.
Ipak, postoje postupci kojima se, ako G jest logička posljedica, to može dokazati, no ti postupci mogu **nikad ne završiti u slučajevima kada G nije logička posljedica**. U tom smislu **predikatna je logika poluodlučljiva**.

- **Rezolucija opovrgavanjem** koristi se u predikatnoj logici, ali je njezina moć ograničena poluodlučljivošću predikatne logike

28



- Sva pravila zaključivanja (uveđena u okviru "prirodnog zaključivanja") koja vrijede za propozicijsku logiku vrijede i za predikatnu logiku
- Pored toga predikatna logika ima dodatna pravila, npr.:

Pravilo univerzalne specijalizacije
- Ako je formula istinita za svaki element domene onda je istinita i za jedan određeni element domene
- Ako je x varijabla i b **bilo koja konstanta iz domene** tada:

$$\forall x F(x) \vdash F(b)$$



Pravilo univerzalne specijalizacije

Svaki element domene može biti zamjena za univerzalno kvantificiranu varijablu

- Primjer uporabe pravila univerzalne specijalizacije u postupku zaključivanja:
- *Premise*
 - Svaki muž voli svoju ženu.
 - Marko je muž.
- *Cilj koji treba dokazati*
 - Marko voli svoju ženu



- *Predikati*:
 - MUŽ(x); definira da je x muž.
 - VOLI(x,y)
- *Funkcija*:
 - žena(x): funkcija koja preslikava x u žena_od_ x . Ako je $z = \text{žena}(x)$, to znači da je z žena od x
- *Cilj koji treba dokazati*:
 - VOLI(Marko, žena(Marko))



TEORIJA DOKAZA PREDIKATNE LOGIKE

- Premise se mogu napisati
 - [1] $\forall x(MUŽ(x) \rightarrow VOLI(x, žena(x)))$
 - [2] **MUŽ(Marko)**
- Zaključivanje:
- Marko je konstanta u domeni osoba, stoga primjenjujući pravilo univerzalne specijalizacije na [1] zaključujemo:
 - [3] **(MUŽ(Marko) \rightarrow VOLI(Marko, žena(Marko)))**
- Primjenjujući Modus ponens na [2] i [3] dobivamo:
 - [4] **VOLI(Marko, žena(Marko))**
- Tako smo dokazali da je cilj logička posljedica premlisa

Prof.dr.sc. Bojana Dalbelo Bašić

Fakultet elektrotehnike i računarstva
Zavod za elektroniku, mikroelektroniku, računalne i inteligentne sustave

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

UMJETNA INTELIGENCIJA

Zaključivanje uporabom predikatne logike (2)



© Bojana Dalbelo Bašić

FER

1



2

PREDUVJETI ZA REZOLUCIJSKO ZAKLJUČIVANJE

Pretvaranje formule u klauzalni oblik

- Rezolucijsko pravilo u predikatnoj logici zahtijeva pretvaranje formule u **klauzalni oblik**
- Implicitno se podrazumijeva:
 - sve varijable u klauzuli su **univerzalno kvantificirane**
 - između klauzula je konjunkcija
- Sve klauzule u predikatnoj logici su **standardizirane** – ne postoje dvije klauzule koje sadrže iste **varijable**
- Pretvorba u klauzalni oblik kroz 10 slijednih koraka



3

REZOLUCIJSKO ZAKLJUČIVANJE

- Nastavljamo s teorijom dokaza



PRETVARANJE FORMULE U KLAUZALNI OBLIK

1. **Uklanjanje \leftrightarrow**
 - $(F \leftrightarrow G) \equiv (\sim F \vee G) \wedge (\sim G \vee F)$
2. **Uklanjanje \rightarrow**
 - $(F \rightarrow G) \equiv (\sim F \vee G)$
3. **Smanjivanje dosega operadora negacije** tako da se odnosi samo na jedan atom
 - $\sim(F \vee G) \equiv (\sim F \wedge \sim G)$
 - $\sim(F \wedge G) \equiv (\sim F \vee \sim G)$
 - $\sim\forall x F(x) \equiv \exists x (\sim F(x))$
 - $\sim\exists x F(x) \equiv \forall x (\sim F(x))$

Kada se u nekom od prethodna tri koraka pojavi dvostruka negacija, eliminiraj je primjenom **involutivnosti** $(\sim(\sim F)) \equiv F$



4

PRETVARANJE FORMULE U KLAUZALNI OBLIK

4. Preimenuj varijable tako da svaki kvantifikator vezuje jedinstvenu varijablu.

- $(\forall x F(x) \vee \forall x G(x)) \equiv (\forall x F(x) \vee \forall y G(y))$
- $(\forall x F(x) \vee \exists x G(x)) \equiv (\forall x F(x) \vee \exists y G(y))$
- $(\exists x F(x) \vee \forall x G(x)) \equiv (\exists x F(x) \vee \forall y G(y))$
- $(\exists x F(x) \vee \exists x G(x)) \equiv (\exists x F(x) \vee \exists y G(y))$
- $(\forall x F(x) \wedge \forall x G(x)) \equiv (\forall x F(x) \wedge \forall y G(y))$
- $(\forall x F(x) \wedge \exists x G(x)) \equiv (\forall x F(x) \wedge \exists y G(y))$
- $(\exists x F(x) \wedge \forall x G(x)) \equiv (\exists x F(x) \wedge \forall y G(y))$
- $(\exists x F(x) \wedge \exists x G(x)) \equiv (\exists x F(x) \wedge \exists y G(y))$

5

PRETVARANJE FORMULE U KLAUZALNI OBLIK

- U složenijim izrazima u kojima vrijednost zamjene zavisi od ostalih varijabli u formuli, egzistencijalno kvantificirane varijable zamjenjuju se tzv. SKOLEM-FUNKCIJOM

Primjer:

- U formuli $\forall x \exists y \text{MAJKA}(y, x)$;
 - vrijednost od y zavisi o x .
- Skolemizacija daje
 - $\forall x \text{MAJKA}(f(x), x)$,
 - gdje je $f(x)$ Skolem funkcija

7

PRETVARANJE FORMULE U KLAUZALNI OBLIK

5. Skolemizacija

- Zamijeni sve egzistencijalno kvantificirane varijable Skolem-izrazima

Primjer:

- $\exists x \text{SESTRA}(x, \text{Ivan})$;

Skolemizacija daje:

- $\text{SESTRA}(\text{Ana}, \text{Ivan})$



Skolem-konstanta

6

PRETVARANJE FORMULE U KLAUZALNI OBLIK

- Argumenti Skolem-funkcije su one univerzalno kvantificirane varijable čiji doseg uključuje doseg egzistencijalno kvantificirane varijable koja se zamjenjuje

Primjer:

- $\exists u \forall v \forall w \exists x \forall y \exists z F(u, v, w, x, y, z)$;

8

PRETVARANJE FORMULE U KLAUZALNI OBLIK

- Argumenti Skolem-funkcije su one univerzalno kvantificirane varijable čiji doseg uključuje doseg egzistencijalno kvantificirane varijable koja se zamjenjuje

Primjer:

- $\exists u \forall v \forall w \exists x \forall y \exists z F(u, v, w, x, y, z);$
Eliminiraju se $\exists u$, $\exists x$, i $\exists z$:

9

PRETVARANJE FORMULE U KLAUZALNI OBLIK

- Argumenti Skolem-funkcije su one univerzalno kvantificirane varijable čiji doseg uključuje doseg egzistencijalno kvantificirane varijable koja se zamjenjuje

Primjer:

- $\exists u \forall v \forall w \exists x \forall y \exists z F(u, v, w, x, y, z);$
Eliminiraju se $\exists u$, $\exists x$, i $\exists z$:
- $$\forall v \forall w \forall y F(a, v, w, f(v, w), y, g(v, w, y));$$

10

PRETVARANJE FORMULE U KLAUZALNI OBLIK

- Argumenti Skolem-funkcije su one univerzalno kvantificirane varijable čiji doseg uključuje doseg egzistencijalno kvantificirane varijable koja se zamjenjuje

Primjer:

- $\exists u \forall v \forall w \exists x \forall y \exists z F(u, v, w, x, y, z);$
- Eliminiraju se $\exists u$, $\exists x$, i $\exists z$:
 $\forall v \forall w \forall y F(a, v, w, f(v, w), y, g(v, w, y));$
i zamjenjuju redom SKOLEM-IZRAZIMA:
 $a, f(v, w), g(v, w, y)$, gdje je a Skolem-konstanta
dok su f, g Skolem-funkcije. .

- Niti jedan od simbola a, f, g ne smije se pojavljivati u izvornoj formuli

11

PRETVARANJE FORMULE U KLAUZALNI OBLIK

- Skolemizacija kao postupak ne daje nužno definiciju Skolem-funkcije nego se radi o metodi pridjeljivanja imena funkcijama koje moraju postojati

Primjer:

- $\forall x \exists y GT(y, x)$
- Skolemizacija daje $\forall x GT(f(x), x)$,
- f je Skolem-funkcija koja može biti:
 - $f(x) = x + 1$ ili
 - $f(x) = x + 5$ itd.

12

PRETVARANJE FORMULE U KLAUZALNI OBLIK

OPRAVDANJE ZA SKOLEMIZACIJU?

- Zašto egzistencijalno kvantificiranu varijablu smijemo zamijeniti konstantom?
- $\exists x \text{ SESTRA}(x, \text{Ivan}) \equiv \text{SESTRA}(\text{Ana}, \text{Ivan})$???
- Odgovor:
 - gornja ekvivalencija općenito ne vrijedi, ali...
 - ... skolemizacija ne utječe na svojstvo **nezadovoljivosti** formule!
- Ako $\exists x \text{ SESTRA}(x, \text{Ivan}) \equiv \perp$
onda $\text{SESTRA}(\text{Ana}, \text{Ivan}) \equiv \perp$
- U kontekstu rezolucije opovrgavanjem to je dovoljno
 - Ako su premise + negirani cilj proturječni, bit će takvi i nakon skolemizacije

13

PRETVARANJE FORMULE U KLAUZALNI OBLIK

- Premjesti sve kvantifikatore (preostali su samo univerzalni) na lijevu stranu formule tako da se na lijevoj strani nalazi niz kvantifikatora koji se nazivaju **prefiks**. Desna strana formule koja se naziva **matrica**, oslobođena je svih kvantifikatora.
Međusobni uređaj kvantifikatora ostaje nepromijenjen!
Formula u takvom obliku se naziva **PRENEKS NORMALNI OBLIK**

- $(\forall x F(x) \vee \forall y G(y)) \equiv \forall x \forall y (F(x) \vee G(y))$
- $(\forall x F(x) \wedge \forall y G(y)) \equiv \forall x \forall y (F(x) \wedge G(y))$
- $(\forall x F(x) \vee H\{x\}) \equiv \forall x (F(x) \vee H\{x\})$
- $(\forall x F(x) \wedge H\{x\}) \equiv \forall x (F(x) \wedge H\{x\})$

14

PRETVARANJE FORMULE U KLAUZALNI OBLIK

- Eliminiraj **prefiks**, ostavi samo matricu. Podrazumijeva se da su sve varijable u formuli univerzalno kvantificirane. (Nema slobodnih varijabli u formuli)
- Pretvorи matricu u **konjunkciju disjunkcija** (konjunktivnu normalnu formu) koristeći distributivnost
 - $(F \vee (G \wedge H)) \equiv ((F \vee G) \wedge (F \vee H))$
 - $((G \wedge H) \vee F) \equiv ((G \vee F) \wedge (H \vee F))$
- Napiši konjunktivnu normalnu formu kao **skup klauzula** brišući operatore konjunkcija. Implicitno se podrazumijava konjunkcija između klauzula

15

PRETVARANJE FORMULE U KLAUZALNI OBLIK

- Standardiziraj klauzule preimenovanjem varijabli tako da nema dvije klauzule koje sadrže iste varijable
 - Sve varijable u klauzulama implicitno su kvantificirane (korak 7) i postoji implicitna konjunkcija između klauzula (korak 9) pa je preimenovanje varijabli valjano i dopušteno jer se temelji na sljedećoj ekvivalenciji:
$$\forall x (F(x) \wedge G(x)) \equiv \forall x \forall y (F(x) \wedge G(y))$$
- **Pazi!** Svrha preimenovanja nije ukloniti višestruko pojavljivanje iste varijable **unutar iste klauzule**!
 - Npr. općenito ne vrijedi:
$$\forall x P(x,x) \equiv \forall x \forall y P(x,y)$$
$$\forall x (P(x) \vee Q(x)) \equiv \forall x \forall y (P(x) \vee Q(y))$$

16

PRETVARANJE FORMULE U KLAUZALNI OBLIK

Primjer

- Pretvorba formule u klauzalni oblik

$$\forall y \forall z (\exists u (P(y, u) \vee P(z, u)) \rightarrow \exists u Q(y, z, u))$$

1. Uklanjanje \leftrightarrow

OK

2. Uklanjanje \rightarrow

$$\forall y \forall z (\neg(\exists u (P(y, u) \vee P(z, u))) \vee \exists u Q(y, z, u))$$

3. Smanjivanje dosega operatora \sim

$$\forall y \forall z (\forall u (\neg P(y, u) \wedge \neg P(z, u)) \vee \exists u Q(y, z, u))$$

4. Preimenovanje varijabli

$$\forall y \forall z (\forall u (\neg P(y, u) \wedge \neg P(z, u)) \vee \exists v Q(y, z, v))$$

17

PRETVARANJE FORMULE U KLAUZALNI OBLIK

Primjer

- Pretvorba formule u klauzalni oblik

$$\forall y \forall z (\exists u (P(y, u) \vee P(z, u)) \rightarrow \exists u Q(y, z, u))$$

1. Uklanjanje \leftrightarrow

OK

2. Uklanjanje \rightarrow

$$\forall y \forall z (\neg(\exists u (P(y, u) \vee P(z, u))) \vee \exists u Q(y, z, u))$$

3. Smanjivanje dosega operatora \sim

$$\forall y \forall z (\forall u (\neg P(y, u) \wedge \neg P(z, u)) \vee \exists u Q(y, z, u))$$

4. Preimenovanje varijabli

$$\forall y \forall z (\forall u (\neg P(y, u) \wedge \neg P(z, u)) \vee \exists v Q(y, z, v))$$

18

PRETVARANJE FORMULE U KLAUZALNI OBLIK

5. Skolemizacija

$$\forall y \forall z (\forall u (\neg P(y, u) \wedge \neg P(z, u)) \vee Q(y, z, f(y, z)))$$

6. Preneks normalni oblik

$$\forall y \forall z \forall u ((\neg P(y, u) \wedge \neg P(z, u)) \vee Q(y, z, f(y, z)))$$

7. Eliminiraj prefiks

$$(\neg P(y, u) \wedge \neg P(z, u)) \vee Q(y, z, f(y, z))$$

8. Konjunktivni normalni oblik

$$((\neg P(y, u) \vee Q(y, z, f(y, z))) \wedge (\neg P(z, u) \vee Q(y, z, f(y, z))))$$

19

PRETVARANJE FORMULE U KLAUZALNI OBLIK

9. Skup klauzula

$$\neg P(y, u) \vee Q(y, z, f(y, z))$$

$$\neg P(z, u) \vee Q(y, z, f(y, z))$$

10. Standardizacija (nakon $u \rightarrow v, y \rightarrow w, z \rightarrow x$),

$$\neg P(y, u) \vee Q(y, z, f(y, z))$$

$$\neg P(x, v) \vee Q(w, x, f(w, x))$$

20

ZAMJENA (SUPSTITUCIJA)

- Univerzalno kvantificirane varijable zamjenjivali smo konstantom.

Primjer

- [1] $\forall x(MUŽ(x) \rightarrow VOLI(x, ŽENA(x)))$
 - [2] MUŽ(Marko)
 - [3] $(MUŽ(Marko) \rightarrow VOLI(Marko, ŽENA(Marko)))$.
- Općenito varijable mogu biti **zamijenjene (supstituirane)** čitavim izrazima
 - Supstitucija je neophodna za primjenu rezolucijskog pravila



21

ZAMJENA (SUPSTITUCIJA)

Skup uređenih parova čini **zamjenu** (supstituciju)

$$\alpha = \{t_1/y_1, t_2/y_2, \dots, t_n/y_n\},$$

gdje su t_i , $i = 1, 2, \dots, n$ izrazi,

y_i , $i = 1, 2, \dots, n$ su različite varijable, i vrijedi $t_i \neq y_i$ za bilo koji $i = 1, 2, \dots, n$.

Primjer:

- izraz $GT(x, y)$,
- supstitucija $\alpha = \{\text{prosinac}/x, \text{travanj}/y\}$,
- rezultat $K\alpha = GT(\text{prosinac}, \text{travanj})$.



22

ZAMJENA (SUPSTITUCIJA)

- Kad se zamjena α primjeni na neki izraz K , tada se svako pojavljivanje y_i u K istodobno zamjeni sa t_i .
- Dobiveni izraz označavamo sa $K\alpha$ i kažemo da je $K\alpha$ slučaj ili primjer izraza K (*engl. instance*).
- Prazna zamjena ε je takva da je $K\varepsilon = K$

Primjer:

$$\text{izraz } K = P(w, f(x), z),$$

- supstitucija $\alpha = \{a/x, g(u)/z\}$,
- rezultat $K\alpha = P(w, f(a), g(u))$



23

KOMPOZICIJA ZAMJENA

- Neka su α i β dvije zamjene. Kompozicija zamjena je definirana kao nova zamjena $\alpha \circ \beta$ tako da je $K(\alpha \circ \beta) = (K\alpha)\beta$, za neki izraz K
- Uporaba zamjene $\alpha \circ \beta$ na izrazu K isto kao prvo α na K , a onda se na tako dobiveni izraz primjeni β .
- Kompozicija zamjena $\alpha \circ \beta$ može biti izvedena na temelju sljedećeg postupka u dva koraka



24

KOMPOZICIJA ZAMJENA

- Neka su dane α i β zamjene
 $\alpha = \{t_1/y_1, t_2/y_2, \dots, t_n/y_n\}$,
 $\beta = \{s_1/x_1, s_2/x_2, \dots, s_m/x_m\}$

1. korak

- Iz α i β konstruiraj skupove $\lambda_1, \lambda_2, \lambda_3$:

$$\begin{aligned}\lambda_1 &= \{t_1\beta/y_1, t_2\beta/y_2, \dots, t_n\beta/y_n, s_1/x_1, s_2/x_2, \dots, s_m/x_m\} \\ \lambda_2 &= \{t_i\beta/y_i \mid t_i\beta/y_i \in \lambda_1 \text{ i } t_i\beta = y_i \text{ za } 1 \leq i \leq n\}, \\ \lambda_3 &= \{s_i/x_i \mid s_i/x_i \in \lambda_1 \text{ i } x_i \in \{y_1, y_2, \dots, y_n\}\};\end{aligned}$$

2. korak

- Izvedi kompoziciju zamjena $\alpha \circ \beta$ uporabom formule

$$\alpha \circ \beta = \lambda_1 - \lambda_2 - \lambda_3$$

25

Primjer

- Neka su zadane zamjene
 $\alpha = \{z/u, h(u)/w\}$
 $\beta = \{a/u, z/w, u/z\}$
- Izvedi kompoziciju $(\alpha \circ \beta)$ na temelju postupka u dva koraka, a zatim pokaži da vrijedi $K(\alpha \circ \beta) = (K\alpha)\beta$ za izraz
 $K = P(u, w, f(z))$

1. korak – konstrukcija skupova λ_1, λ_2 i λ_3 .

- $\lambda_1 = \{z\beta/u, h(u)\beta/w, a/u, z/w, u/z\}$
 $= \{u/u, h(a)/w, a/u, z/w, u/z\}$
- $\lambda_2 = \{u/u\}$
- $\lambda_3 = \{a/u, z/w\}$

2. korak – izvođenje kompozicije zamjena $\alpha \circ \beta$ uporabom formule

$$\alpha \circ \beta = \lambda_1 - \lambda_2 - \lambda_3 = \{h(a)/w, u/z\}$$

26

KOMPOZICIJA ZAMJENA - PRIMJER

- Još treba pokazati da se isti rezultat dobije primjenom definicije zamjena tj. da je $K(\alpha \circ \beta) = (K\alpha)\beta$ za slučaj
 $K = P(u, w, f(z))$
 $\alpha = \{z/u, h(u)/w\}$
 $\beta = \{a/u, z/w, u/z\}$
- (i) prema definiciji:
 - $K\alpha = P(z, h(u), f(z))$
 - $(K\alpha)\beta = P(u, h(a), f(u))$
- (ii) primjenom izvedene formule za $\alpha \circ \beta$:
 $\alpha \circ \beta = \{h(a)/w, u/z\}$
 $K(\alpha \circ \beta) = P(u, h(a), f(u))$
- Dakle vrijedi $K(\alpha \circ \beta) = (K\alpha)\beta$

27

KOMPOZICIJA ZAMJENA - PRIMJER

Asocijativnost

$$(\alpha \circ \beta) \circ \gamma = \alpha \circ (\beta \circ \gamma)$$

ε je neutralni element s lijeva

$$\varepsilon \circ \alpha = \alpha$$

ε je neutralni element s desna

$$\alpha \circ \varepsilon = \alpha$$

Općenito ne vrijedi komutativnost

$$\alpha \circ \beta \neq \beta \circ \alpha$$

28

- Općeniti izrazi K_1 i K_2 mogu se svesti na isti oblik akko postoji supstitucija γ takva da vrijedi

$$K_1 \gamma = K_2 \gamma$$

- Supsticija γ se naziva **unifikator** (*engl. unifier*)
- Izraz $K_i \gamma$, $i = 1, 2$ naziva se **zajednički slučaj** ili primjer (*engl. common instance*)
- Kaže se da su opći izrazi K_1 i K_2 unificirani pomoću γ
- Izrazi mogu imati više unifikatora

29



- Atomi $P(x)$ i $P(y)$, gdje su x i y varijable, imaju unifikatore:
 - $\gamma_1 = \{b/x, b/y\}$ koji daje zajednički slučaj $P(b)$ i
 - $\gamma_2 = \{z/x, z/y\}$ koji daje zajednički slučaj $P(z)$.
- Ako je b konstanta, a z varijabla možemo reći da je $P(z)$ općenitiji zajednički slučaj nego li je to $P(b)$
- Naime, $P(b)$ je slučaj od $P(z)$ uz supstituciju $\{b/z\}$, ali NE VRIJEDI OBRATNO. (**Zašto?**)

30



NAJOPĆENITIJI UNIFIKATOR

- Za gornji primjer kažemo da je unifikator γ_2 **općenitiji** od unifikatora γ_1 za atome $P(x)$ i $P(y)$. Naime, tada postoji zamjena γ_3 takva da je

$$(P \gamma_2) \gamma_3 = P \gamma_1$$
- $P(x)$ i $P(y)$ mogu imati više unifikatora – zanima nas najopćenitiji unifikator
- Unifikator δ se naziva **najopćenitiji unifikator** akko za svaki unifikator γ od K_1 i K_2 , zajednički primjer $K_i \delta$ je općenitiji (ili jednak) od zajedničkog primjera $K_i \gamma$
- Izraz $K_i \delta$ se naziva **najopćenitiji zajednički slučaj**
 - $K_i \gamma$ je primjer od $K_i \delta$

31



NAJOPĆENITIJI UNIFIKATOR

δ je **najopćenitiji unifikator** akko za svaki unifikator γ od K_i , $i=1,2$ postoji zamjena θ takva da je

$$K_i \gamma = (K_i \delta) \theta = K_i (\delta \theta)$$

tj. $\gamma = \delta \theta$.

- MGU se može tumačiti kao da ima za posljedicu najmanje moguće promjene da bi se unificirali izrazi
- Pretpostavimo da tijekom postupka nalaženja zajedničkog primjera dva izraza postoji izbor zamijeniti varijablu konstantom ili zamijeniti varijablu drugom varijablom. Procedura nalaženja MGU će uvijek izabrati ovo drugo
- Dva izraza mogu imati više najopćenitijih unifikatora – u tom su slučaju svi najopćenitiji zajednički izrazi jednako općeniti

32



ALGORITAM ZA UNIFIKACIJU IZRAZA

- U literaturi postoji više poznatih algoritama za nalaženje najopćenitijeg unifikatora (dojavljuju pogrešku ako se izrazi ne mogu unificirati)
- Razmotrit ćemo rekurzivni algoritam MGUNIFIER (Luger, Stubblefield, 1993; Shinghal, 1992)
- Algoritam koristi pojmove:
 - glava općenitog izraza
 - rep općenitog izraza
- Općeniti izraz možemo napisati kao **listu** gdje su elementi liste odvojeni prazninama, a svaki element liste je opet moguća lista nazvana podizraz

33

REKURZIVNI ALGORITAM ZA NALAŽENJE NAJOPĆENITIJEZ ZAJEDNIČKOG UNIFIKATORA

procedura MGUNIFIER (K_1, K_2)

ako ili K_1 ili K_2 simbolizira konstantu, varijablu, funkciju, predikat ili praznu listu **tada**:

ako K_1 i K_2 identični, **tada** vrati praznu supstituciju {};

ako K_1 ili K_2 prazne liste, **tada** vrati pogrešku;

ako K_1 predstavlja varijablu **tada**:

ako se K_1 pojavljuje u K_2 **tada** vrati pogreška;

inače vrati $\{K_2 / K_1\}$;

ako K_2 predstavlja varijablu **tada**:

ako se K_2 pojavljuje u K_1 **tada** vrati pogreška;

inače vrati $\{K_1 / K_2\}$;

ako niti K_1 niti K_2 ne predstavlja varijablu **tada** vrati pogreška;

inače

$\alpha := \text{MGUNIFIER}(\text{glava od } K_1, \text{ glava od } K_2)$;

ako $\alpha = \text{pogreška}$ **tada** vrati pogreška;

35

ALGORITAM ZA UNIFIKACIJU IZRAZA

Primjer

- $P(x, y, f(b))$ može biti napisano kao lista $(P \ x \ y \ (f \ b))$ gdje su element liste P, x, y, (f b).
Zadnji element liste je i sam lista, sastavljen od elemenata f i b
- **Glava liste** $(P \ x \ y \ (f \ b))$ je P, dok je **rep liste** $(x \ y \ (f \ b))$
- Sintaksa lista je sintaksa programskog jezika LISP

Sintaksa FOPL	Sintaksa liste
$P(a, b)$	$(P \ a \ b)$
$P(f(a), g(x,y))$	$(P \ (f \ a) \ (g \ x \ y))$
$\text{EQUAL}(Eva, MAJKA(Iva))$	$(\text{EQUAL} \ Eva \ (\text{MAJKA} \ Iva))$
$P(x) \wedge Q(y)$	$((P \ x) \wedge (Q \ y))$

34

REKURZIVNI ALGORITAM ZA NALAŽENJE NAJOPĆENITIJEZ ZAJEDNIČKOG UNIFIKATORA

... (nastavak)

$K_3 :=$ rezultat primjene zamjene α na rep od K_1 ;

$K_4 :=$ rezultat primjene zamjene α na rep od K_2 ;

$\beta := \text{MGUNIFIER}(K_3, K_4)$;

ako $\beta = \text{pogreška}$ **tada** vrati pogreška;

vrati kompoziciju zamjena $\alpha \circ \beta$;

36

- Procedura MGUNIFIER vraća pogrešku u slučajevima kada unifikacija nije moguća:
 - varijabla treba biti zamjenjena izrazom koji ju sadrži. Provjerava se pojavljuje li se varijabla u izrazu kojim se zamjenjuje. Takva zamjena vodila bi beskonačnoj petlji.
 - kada bi trebalo zamijeniti konstantu, funkciju ili predikat – što je u suprotnosti s definicijom supstitucije

37



- Dani su literali $P(x, x)$ i $P(f(z), z)$.
- MGUNIFIER će prvo naći supstituciju $\{f(z)/x\}$ koja daje $P(f(z), f(z))$ i $P(f(z), z)$.
- Kada ne bi bilo provjere uvjeta, sljedeća supstitucija $\{f(z)/z\}$ bi dala $P(f(f(z)), f(f(z)))$ i $P(f(f(z)), f(z))$ i tako u beskonačnost

38



- Primjeri izraza koji se ne mogu unificirati

K1	K2	Uzrok pogreški kod unifikacije
$P(x)$	$P(f(x))$	Vidi prethodni primjer (beskonačna petlja)
$P(f(x))$	$P(x)$	
$P(x)$	$Q(x)$	Predikat se ne može zamijeniti drugim predikatom.
$P(f(x))$	$P(g(x))$	Funkcija se ne može zamijeniti drugom funkcijom.

PRIMJER

39



- Nađi MGU izraza:
 $K1 = P(g(u), z, f(z))$ i $K2 = P(x, y, f(b))$
 - $\{g(u)/x\}$ unificira prve podizraze od K1 i K2 koji se ne slažu
 $K1\{g(u)/x\} = P(g(u), z, f(z))$
 $K2\{g(u)/x\} = P(g(u), y, f(b))$
 - $\{y/z\}$ unificira sljedeće izraze koji se ne slažu
 $\text{Kompozicija } \{g(u)/x\} \cdot \{y/z\} = \{g(u)/x, y/z\}$
 $K1\{g(u)/x, y/z\} = P(g(u), y, f(y))$
 $K2\{g(u)/x, y/z\} = P(g(u), y, f(b))$
 - $\{b/y\}$ unificira sljedeće izraze koji se ne slažu
 $\text{Kompozicija } \{g(u)/x, y/z\} \cdot \{b/y\} = \{g(u)/x, b/z, b/y\}$
je **MGU**, koji označavamo s δ .
- Primjenom mgu $\delta = \{g(u)/x, b/z, b/y\}$ na K1 i K2 dobivamo najopćenitiji zajednički primjer
 $P(g(u), b, f(b))$

40



PRIMJER

- Ako za neka 2 izraza MGU nije jedinstven, tj. postoji više najopćenitijih unifikatora, odgovarajući najopćenitiji zajednički izrazi su **alfabetske varijante** – razlikuju se samo po *imenima* varijabli
- Primjer*

Skup izraza	Najopćenitiji zajednički primjer
$\{P(x), P(a)\}$	$P(a)$
$\{P(f(x), y, g(y)), P(f(x), z, g(x))\}$	$P(f(x), x, g(x))$
$\{P(f(x, g(a,y)), g(a, y)), P(f(x,z), z)\}$	$P(f(x, g(a, y)), g(a, y))$

41

REZOLUCIJA

- Rezolucija u FOPL slična je rezoluciji u propozicijskoj logici
- Neka su L_k , $k = 1, \dots, i$ i M_l , $l = 1, \dots, j$ literali (neki negirani, a neki afirmativni)
- Neka su dane dvije klauzule
 - (I1) $L_1 \vee L_2 \vee \dots \vee L_i$
 - (I2) $M_1 \vee M_2 \vee \dots \vee M_j$
- Prije primjene rezolucije klauzule trebaju biti standardizirane (preimenovati varijable)
- Prepostavimo da u (I1) i (I2) postoje literali koji se mogu komplementarno unificirati. Prepostavimo (bez gubitka općenitosti – vrijedi komutativnost) da su to literali L_1 i M_1
- Prepostavimo da se unifikacija L_1 i M_1 može obaviti sa **MGU** δ .

43

UNIFIKACIJA LITERALA

Definicija

- Dva literala mogu se **unificirati** akko
 - oba označavaju negirane atome ili ova označavaju afirmativne atome
 - ti atomi se mogu unificirati.

Primjer

$P(x) \text{ i } P(y) \text{ ili } \sim P(x) \text{ i } \sim P(y)$

Definicija

- Dva literala mogu se **komplementarno unificirati** akko
 - jedan od njih je negirani atom, a drugi je afirmativni atom
 - ti se atomi mogu unificirati.

Primjer

$P(x) \text{ i } \sim P(y) \text{ ili } \sim P(x) \text{ i } P(y)$

42

REZOLUCIJA

- Sada možemo razriješiti *roditeljske klauzule* (I1) i (I2) po L_1 i M_1 i izvesti *resolventnu klauzulu*

$$(I3) \quad L_2\delta \vee L_3\delta \vee \dots \vee L_i\delta \vee \underbrace{M_2\delta \vee M_3\delta \vee \dots \vee M_j\delta}_{\text{Ostatak (I1)}} \quad \underbrace{\dots}_{\text{Ostatak (I2)}}$$

- Literali u resolventi dobiju se primjenom \vee na uniju literalova u roditeljskim klauzulama izuzevši literale s kojima se razrješavanje obavlja

Napomena

- Razrješavanjem dviju jediničnih klauzula izvodi se prazna klauzula **NIL**

44

PRIMJER REZOLUCIJE

Nadi rezolventu za sljedeće klauzule

- $P(g(y), x, f(z)) \vee Q(z, b) \vee R(x)$
- $S(x, y) \vee \neg P(x, y, f(a))$
- **Klauzule nisu standadizirane!** (simboli za varijable x i y pojavljuju se u obje klauzule)

Potrebito je preimenovati varijable u prvoj klauzuli:

- x postaje varijabla w,
- y postaje varijabla u.

Standardizirane klauzule:

- (I1) $P(g(u), w, f(z)) \vee Q(z, b) \vee R(w)$
- (I2) $S(x, y) \vee \neg P(x, y, f(a))$

Nadi mgu!

45

REZOLUCIJA

Rezolucijsko pravilo je ispravno (zdravo)

- Prema pravilu univerzalne specijalizacije $I_1\delta$ je logička posljedica od I_1 . Slično, $I_2\delta$ je logička posljedica od I_2 .
- Zaključivanjem istim kao i u propozicijskoj logici zaključujemo da je I_3 je logička posljedica $I_1\delta \vee I_2\delta$.
- Dakle I_3 je logička posljedica $I_1 \vee I_2$ te je **rezolucijsko pravilo u predikatnoj logici ispravno**.
- Literali L_1 i M_1 su unificirani pomoću mgu pa je resolventa I_3 u najopćenitijem mogućem obliku

47

PRIMJER REZOLUCIJE

- (I1) $P(g(u), y, f(a)) \vee Q(a, b) \vee R(y)$

- (I2) $S(g(u), y) \vee \neg P(g(u), y, f(a))$

▪ Razrješavanjem po $P(g(u), y, f(a))$ i $\neg P(g(u), y, f(a))$ dobiva se resolventna klauzula:

- (I3) $S(g(u), y) \vee Q(a, b) \vee R(y)$

46

REZOLUCIJA

Rezolucija opovrgavanjem je potpuna

- Kao i u propozicijskoj logici tako i u predikatnoj logici vrijedi da je rezolucija opovrgavanjem potpuna. (Robinson, 1965)
- Formula G je logička posljedica premisa F_1, F_2, \dots, F_n akko možemo pokazati da je ulazni skup $F_1, F_2, \dots, F_n, \neg G$ nekonzistentan izvođenjem resolventne klauzule NIL.
- Svojstvo da je rezolucija zdrava i da je rezolucija opovrgavanjem potpuna osigurava da možemo uvijek dokazati teorem G, ako je G teorem.
- Međutim ako G nije teorem postupak dokazivanja teorema rezolucijom opovrgavanjem može nikad ne završiti – u tom je smislu predikatna logika poluodlučljiva

48

TEMELJNE KLAZULE I ODLUČLJIVOST

- Literali koji sadrže samo konstante (bez varijabli) nazivaju se **temeljni literali** (engl. *ground literals*), a njihova disjunkcija naziva se **temeljna klauzula** (engl. *ground clause*)
- Podrazred predikatne logike koji dopušta samo pojavljivanje konstanti, funkcija i predikata (a ne varijabli) je **ODLUČLJIV**
- Ta potklasa ima istu moć zaključivanja kao i propozicijska logika

49

PRIMJER

Primjer rezolucije opovrgavanjem u podrazredu predikatne logike koja ne dozvoljava pojavljivanje varijabli.

- Za isti ovaj primjer, cilj "Majka je zadovoljna" dokazali smo u propozicijskoj logici:

- prirodnim zaključivanjem i
- rezolucijom opovrgavanjem.

- [i] Ivan se probudio
- [ii] Ivan nosi pribor za čišćenje
- [iii] Majka je zadovoljna ako se Ivan probudi i čisti svoju sobu
- [iv] Ako Ivan nosi pribor za čišćenje, tada on čisti svoju sobu

- Dokažite rezolucijom opovrgavanjem cilj: *Majka je zadovoljna*

50

PRIMJER

- Ulazni skup tvore premise i negacija cilja pretvorene u klauzalnu formu

[I1] BUDAN(Ivan)
[I2] NOSI(Ivan, pribor)
[I3] ~BUDAN(Ivan) \vee ~ČISTI(Ivan, soba) \vee ZADOVOLJNA(majka)
[I4] ~NOSI(Ivan, pribor) \vee ČISTI (Ivan, soba)
[I5] ~ZADOVOLJNA(majka) CILJ

- Iz ulaznog skupa izvodimo slijedeće resolvente:

[I3] ~BUDAN(Ivan) \vee ~ČISTI(Ivan, soba) \vee ZADOVOLJNA(majka)
[I5] ~ZADOVOLJNA(majka)

[I6] ~BUDAN(Ivan) \vee ~ČISTI(Ivan, soba)

51

PRIMJER

[I6] ~BUDAN(Ivan) \vee ~ČISTI(Ivan, soba)

[I1] BUDAN(Ivan)

[I7] ~ČISTI(Ivan, soba)

[I4] ~NOSI(Ivan, pribor) \vee ČISTI (Ivan, soba)

[I8] ~NOSI(Ivan, pribor)

[I2] NOSI(Ivan, pribor)

[I9] NIL

52

- Slično kao i u propozicijskoj logici, klauzule u predikatnoj logici moraju biti faktorizirane kako bi se sačuvala potpunost postupka opovrgavanjem

Primjer kako rezolucija gubi svoju potpunost ako nema faktorizacije:

$$[I1] P(u) \vee P(w)$$

$$[I2] \neg P(x) \vee \neg P(y)$$

- Uz uporabu mgu = {u/x} i razrješavanjem po $P(u)$ i $\neg P(u)$ dobivamo resolventu

$$[I3] P(w) \vee \neg P(y)$$

(Razrješavanjem po drugim literalima dobivamo alfabetske varijante od I3)



PRIMJER

Primjer:

$$[I1] P(u) \vee P(w)$$

$$[I2] \neg P(x) \vee \neg P(y)$$

- Uz faktorizaciju:
- Faktoriziramo I1 uporabom mgu={w/u} dobivamo

$$[I1'] P(w)$$

- Faktoriziramo I2 uporabom mgu={y/x} dobivamo

$$[I2'] \neg P(y)$$

- Razrješavanjem jediničnih klauzula $P(w)$ i $\neg P(y)$ uz uporabu mgu = {w/y} izvodimo praznu klauzulu

$$[I3'] NIL$$



- Klauzula u predikatnoj logici može biti faktorizirana akko sadrži literale koji se mogu unificirati

$$[I] L_1 \vee L_2 \vee L_3 \vee \dots \vee L_n$$

- Prepostavimo (bez gubitka općenitosti) da se literali L_1 i L_2 mogu unificirati sa MGU δ . Gornja klauzula tada može biti faktorizirana dajući klauzulu I'

$$[I'] L_2\delta \vee L_3\delta \vee \dots \vee L_n\delta$$

- Klauzula I' se naziva **faktor-klauzula** od I

- Ako skup literalova ima MGU δ tada je $I\delta$ faktor klauzula od I , gdje su višestruka pojavljivanja literalova u $I\delta$ zamjenjena jednim pojavljivanjem literalova



PRIMJER

Primjer

$$P(x, y, f(b)) \vee S(x, y) \vee P(g(u), w, f(z))$$

gdje su: P i S - predikatni simboli,
 f i g - funkcijski simboli,
 x, y, u, w, z – varijable,
 b - konstanta.

- Prvi i treći literal mogu se unificirati pomoću mgu $\delta = \{g(u)/x, y/w, b/z\}$. Faktor klauzula je:

$$P(g(u), y, f(b)) \vee S(g(u), y)$$



FAKTORIZACIJA U PREDIKATNOJ LOGICI

- Faktor koji se sastoji samo od jednog literala naziva se **jedinični faktor**.
- Od sada pa nadalje će se smatrati da je resolventa I₃ izvedena iz klauzula I₁ i I₂ akko dobivamo I₃ na bilo koji od sljedećih načina:
 - 1) razrješavanjem I₁ i I₂
 - 2) razrješavanjem I₁ i faktora od I₂
 - 3) razrješavanjem faktora od I₁ i I₂
 - 4) razrješavanjem faktora od I₁ i faktora od I₂

57

PRIMJER

Primjer

- [1] $\forall x(\text{MUŽ}(x) \rightarrow \text{VOLI}(x, \text{žena}(x)))$
- [2] MUŽ(Marko)

Dokaži uporabom rezolucije opovrgavanjem:

$\text{VOLI}(\text{Marko}, \text{žena}(\text{Marko}))$.

Premise i negacija ciljne formule pretvore se u klauzalni oblik

- [I1] $\neg\text{MUŽ}(x) \vee \text{VOLI}(x, \text{žena}(x))$
- [I2] MUŽ(Marko)
- [I3] $\neg\text{VOLI}(\text{Marko}, \text{žena}(\text{Marko}))$

Iz ulaznog skupa rezolucijom izvodimo:

- [I4] $\neg\text{MUŽ}(\text{Marko})$ - uporabom **mgu**={Marko/x} iz [I1] i [I3].
- [I5] NIL - uporabom {} iz [I2] i [I4].

Time je dokazano da je $\text{VOLI}(\text{Marko}, \text{žena}(\text{Marko}))$ dedukcija

59

REZOLUCIJSKI POSTUPAK OPOVRGAVANJEM (ALGORITAM)

Da bi se dokazalo da je ciljna formula G deduktivna posljedica od F₁, F₂, ..., F_n, potrebno je primijeniti sljedeće korake:

1. Pretvori F₁, F₂, F₃, ..., F_n, $\neg G$ u **klauzalni oblik**.
2. Izaberite iz skupa klauzula dvije klauzule koje su razrješive. Ako je potrebno, standardiziraj te klauzule.
Izvedi resolventu i dodaj je skupu klauzula. Ponavljam ovaj korak sve dok se ne ispunji jedan od uvjeta:
 - Izvedena je prazna klauzula NIL . (Dokazano je da je cilj G teorem, ulazni skup je nekonzistentan.)
 - Niti jedan par klauzula ne može se razrješiti ili ne može se izvesti niti jedna nova klauzula (Dokazano je da cilj G nije teorem)
 - Neki unaprijed zadani resursi računala su iscrpljeni. (Nema odluke o G – predikatna logika je poluodlučljiva)

58

PRIMJER

Primjer

- Robot dostavlja pakete.
- Robot zna da su svi paketi u sobi 27 manji od svakog paketa u sobi 28.
- A i B su paketi.
- Paket A je u sobi 27 ili u sobi 28, ali robot ne zna u kojoj.
- Paket B je u sobi 27 i nije manji od paketa A.

Uporabom rezolucije opovrgavanjem pokaži kako robot može zaključiti da je paket A u sobi 27.

KONSTANTE

27, 28, A, B

PREDIKATI

- PAKET(x) - x je paket – skraćeno P(x)
- U_SOBI(x, y) - x je u sobi y – skraćeno U(x, y)
- MANJI(x, y) - x je manji od y – M(x, y)

60

- **BAZA ZNANJA**

$$\forall x \forall y ((P(x) \wedge P(y) \wedge U(x, 27) \wedge U(y, 28)) \rightarrow M(x, y))$$

P(A)

P(B)

U(A, 27) \vee U(A, 28)

U(B, 27) \wedge \neg M(B, A)



61

- **BAZA ZNANJA**

$$\forall x \forall y (P(x) \wedge P(y) \wedge U(x, 27) \wedge U(y, 28)) \rightarrow M(x, y)$$

P(A)

P(B)

U(A, 27) \vee U(A, 28)

U(B, 27) \wedge \neg M(B, A)

- **BAZA ZNANJA u klauzalnoj formi**

$$\neg P(x) \vee \neg P(y) \vee \neg U(x, 27) \vee \neg U(y, 28) \vee M(x, y)$$

P(A)

P(B)

U(A, 27) \vee U(A, 28)

U(B, 27)

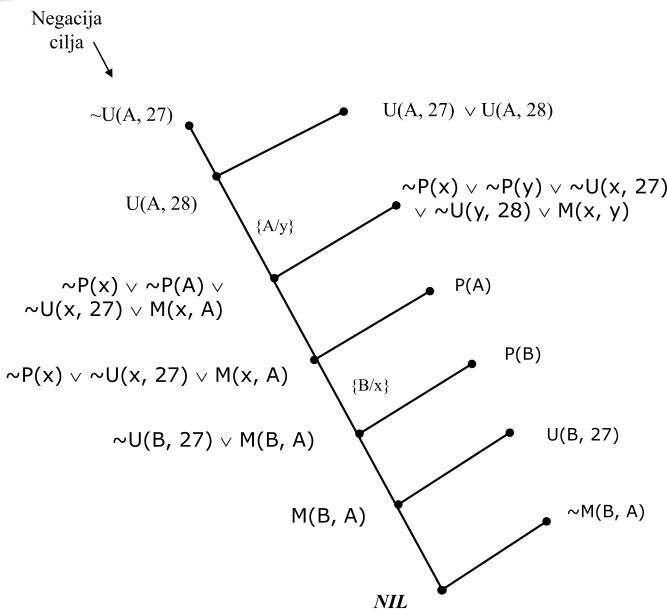
\neg M(B, A)

\neg U(A, 27)

(negacija cilja)

62

STABLO DOKAZA



63

VRSTE REZOLUCIJA

- **Binarna rezolucija** je kombiniranje dvije klauzule koje sadrže komplementarne literale

Primjer

$$Q(x) \vee \neg P(x, a)$$

$$\neg Q(b) \vee R(x)$$

Uz unifikaciju {b/x} resolventna klauzula je
 $\neg P(b, a) \vee R(b)$

- **Rezolucija s rezultirajućom jediničnom klauzulom** (engl. *unit resulting resolution*)

Istodobno razrješavanje više klauzula kako bi se izvela jedinična klauzula. Sve roditeljske klauzule osim jedne su jedinične, a ta ima točno jedan literal više od ukupnog broja jediničnih klauzula

64



VRSTE REZOLUCIJA

Primjer

VJENČANI(Ana, Marko)

~OTAC(Marko, Ivan)

~VJENČANI(x, y) \vee ~MAJKA(x, z) \vee OTAC(y, z)

(uz supstituciju {Ana/x, Marko/y, Ivan/z} resolventa je jedinična klauzula)

~MAJKA(Ana, Ivan)

- **Linearna rezolucija**

Ako je uvijek jedna od roditeljskih klauzula izvedena u prethodnom koraku

- **Linearna rezolucija na ulaznom skupu**

Ako je jedna od roditeljskih klauzula uvijek iz izvornog ulaznog skupa klauzula

65

STRATEGIJE REZOLUCIJE

Unifikacija + rezolucija \rightarrow **automatsko zaključivanje**

ALI

- Rezolucija je nedjelotvorna bez dalnjih razrada!
- Razrješavanje slučajno odabralih klauzula \rightarrow kombinatorna eksplozija
- Važna uporaba metoda koje ograničavaju pretraživanje

Odabir redoslijeda razrješavanja klauzula kako bi **postupak rezolucije bio djelotvorniji** zove se
STRATEGIJA

66

STRATEGIJE REZOLUCIJE

1. **Uređajne strategije** (engl. *ordering strategies*) – određuju slijed (poredak) kojim će se klauzule razrješavati
2. **Strategije skraćivanja** (engl. *pruning strategies*) – uklanjanje iz skupa klauzula onih klauzula i literala koji nisu neophodni za dokaz
3. **Strategije ograničavanja** (engl. *restriction strategies*) – propisuju primjenu rezolucije samo na one klauzule koje se smatraju vitalne za dokaz

67

STRATEGIJA SKUPA POTPORA

- Jedna od najvažnijih strategija (iz skupa strategija ograničavanja)
- Neka je S kontradiktoran skup klauzula i neka je T podskup od S. Tada je T **skup potpore** od S ako je S \ T konzistentan.
- Rezolucija skupa potpore je rezolucija u kojoj nikad nisu obje klauzule iz S \ T. Za svaku rezolventu barem je jedna roditeljska klauzula iz T.
 - pretpostavka: baza znanja je konzistentna, jer inače nam ni dokazivanje cilja ne znači puno
 - dodatna prednost – stabla dokaza su razumljiva jer su usmjerenja cilju

68

EKSTRAKCIJA ODGOVORA POMOĆU REZOLUCIJE OPOVRGAVANJEM

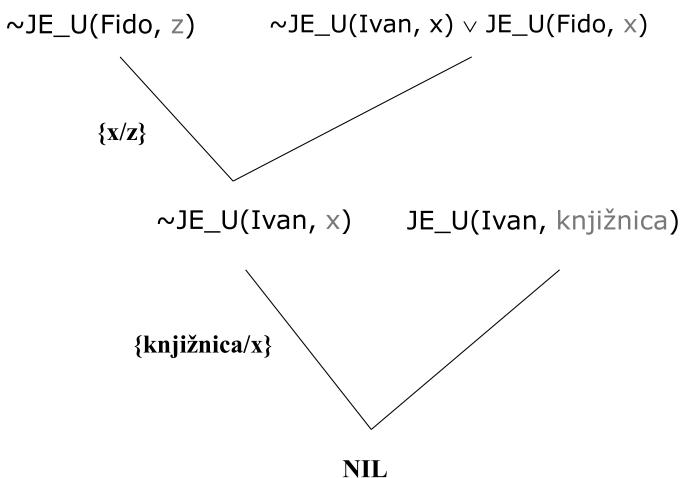
- Ako je neka formula oblika $\exists x G(x)$ logička posljedica nekih premsa tada to možemo dokazati rezolucijom opovrgavanjem.
- Rezolucijom opovrgavanjem možemo i više od toga tj. odgovoriti na pitanje:

Za koje vrijednosti x
je $G(x)$ logička posljedica premsa?

69



EKSTRAKCIJA ODGOVORA POMOĆU REZOLUCIJE OPOVRGAVANJEM



71



EKSTRAKCIJA ODGOVORA POMOĆU REZOLUCIJE OPOVRGAVANJEM

Primjer:

- Pas Fido je svadje gdje je njegov gazda Ivan.
- Ivan je u knjižnici.

Gdje je Fido?

Aksiomi:

$$\forall x JE_U(Ivan, x) \rightarrow JE_U(Fido, x)$$

$$JE_U(Ivan, knjižnica)$$

Klauzalna forma:

$$\sim JE_U(Ivan, x) \vee JE_U(Fido, x)$$

$$JE_U(Ivan, knjižnica)$$

Hipoteza:

$$\exists z JE_U(Fido, z) \quad (z?)$$

Negacija hipoteze:

$$\forall z \sim JE_U(Fido, z) \quad \dots \quad (\text{Fido je nigdje})$$

$$\sim JE_U(Fido, z) \quad (\text{klauzalna forma negacije hipoteze})$$

70

EKSTRAKCIJA ODGOVORA

- Supstitucije pod kojima je nađena kontradikcija jesu supstitucije pod kojima je prepostavka (hipoteza) istinita.
- Čuvanje informacije o unifikacijama tijekom dokaza omogućava odgovor na upit

Odgovor:

hipoteza	supstitucije	odgovor na pitanje
$\exists z JE_U(Fido, z)$	$\{x/z, knjižnica/x\}$	$= JE_U(Fido, knjižnica)$

Napomena: Nađemo li kompoziciju ovih dviju suspsticija:

$\alpha = \{x/z\}$ i $\beta = \{knjižnica/x\}$ (postupkom u dva koraka) dobit ćemo $\alpha \circ \beta = \{knjižnica/z, knjižnica/x\}$

72

Primjer

1. Ankica je mama od Branke	MAJKA(Ankica, Branka)
2. Za sve x i y vrijedi: Ako je x kćerka od y onda je y majka od x	$\forall x \forall y (\text{KĆERKA}(x, y) \rightarrow \text{MAJKA}(y, x))$
3. Zorica je kćerka od Branke.	KĆERKA (Zorica, Branka)
4. Za sve x, y, z vrijedi: Ako je x majka od y , y majka od z , tada je x baka od z .	$\forall x \forall y \forall z (\text{MAJKA}(x, y) \wedge \text{MAJKA}(y, z) \rightarrow \text{BAKA}(x, z))$

Dokaži rezolucijom opovrgavanjem: Zorica ima baku tj.
 $\exists v(\text{BAKA}(v, \text{Zorica}))$

73

Premise u klauzalnom obliku + negacija cilja:

- [I1] MAJKA(Ankica, Branka)
- [I2] $\neg \text{KĆERKA}(u, w) \vee \text{MAJKA}(w, u)$
- [I3] KĆERKA (Zorica, Branka)
- [I4] $\neg \text{MAJKA}(x, y) \vee \neg \text{MAJKA}(y, z) \vee \text{BAKA}(x, z)$
- [I5] $\neg \text{BAKA}(v, \text{Zorica})$

Ivodimo

[I6] $\neg \text{MAJKA}(x, y) \vee \neg \text{MAJKA}(y, \text{Zorica})$	mgu={x/v, Zorica/z}, [I4] i [I5]
[I7] $\neg \text{KĆERKA}(\text{Zorica}, y) \vee \neg \text{MAJKA}(x, y)$	mgu={y/w, Zorica/u}, [I2] i [I6]
[I8] $\neg \text{MAJKA}(x, \text{Branka})$	mgu={Branka/y} [I3] i [I7]
[I9] NIL	mgu={Ankica/x} [I1] i [I8]

74

- Uporaba rezolucije opovrgavanjem za odgovor na pitanje iz skupa premsa [I1]-[I4]

Tko je Zoričina baka?

- Ne zanima nas samo odgovor postoji li v tako da je v baka od Zorice tj. $\exists v (\text{BAKA}(v, \text{Zorica}))$, nego nas zanima vrijednost varijable v

Postupak:

- Svaka klauzula koja se dobije kao negacija cilja pretvara se u tautologiju. To se radi tako da klauzuli dodajemo negaciju svakog literalu koji ona sadrži. (Ako je $G(x)$ negacija cilja, tada je $G(x) \vee \neg G(x)$ tautologija)

75

- [J1] MAJKA(Ankica, Branka)

- [J2] $\neg \text{KĆERKA}(u, w) \vee \text{MAJKA}(w, u)$

- [J3] KĆERKA (Zorica, Branka)

- [J4] $\neg \text{MAJKA}(x, y) \vee \neg \text{MAJKA}(y, z) \vee \text{BAKA}(x, z)$

- [J5] $\neg \text{BAKA}(v, \text{Zorica}) \vee \text{BAKA}(v, \text{Zorica})$

- Sada ponavljamo potpuno isti postupak kao u dokazivanju formule $\exists v(\text{BAKA}(v, \text{Zorica}))$.

- Rezultat: umjesto **NIL** – odgovor na pitanje *tko je Zoričina baka*

76

PRIMJER

1. $\sim \text{MAJKA}(x, y) \vee \sim \text{MAJKA}(y, \text{Zorica}) \vee \text{BAKA}(x, \text{Zorica})$	$\text{mgu}=\{x/v, \text{Zorica}/z\}$, [J4] i [J5]
2. $\sim \text{KĆERKA}(\text{Zorica}, y) \vee \sim \text{MAJKA}(x, y) \vee \text{BAKA}(x, \text{Zorica})$	$\text{mgu}=\{y/w, \text{Zorica}/u\}$, [J2] i [J6]
3. $\sim \text{MAJKA}(x, \text{Branka}) \vee \text{BAKA}(x, \text{Zorica})$	$\text{mgu}=\{\text{Branka}/y\}$ [J3] i [J7]
4. $\text{BAKA}(\text{Ankica}, \text{Zorica})$	$\text{mgu}=\{\text{Ankica}/x\}$ [J1] i [J8]

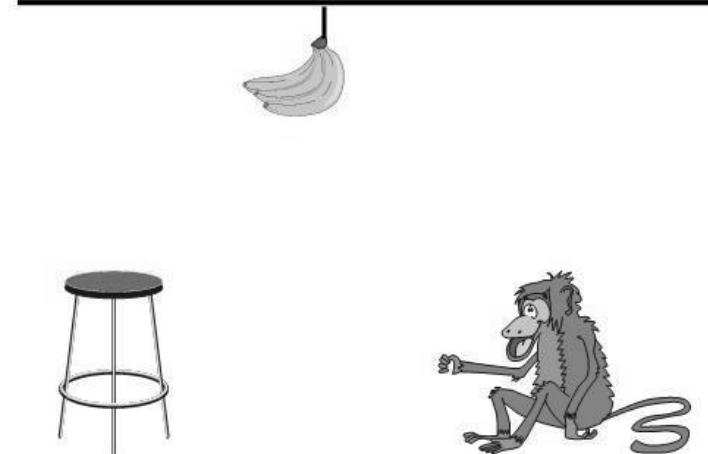
Ankica je Zoričina baka

77



PRIMJER REZOLUCIJE

- Majmun i banane



79

Primjer: Majmun i banane



78

PRIMJER REZOLUCIJE

U sobi se nalaze:

- majmun,
- stolica,
- banane koje vise sa sredine stropa, ali na visini koja nije na dohvatu ruke majmuna.

Ako je majmun dovoljno bistar, on može:

- postaviti stolicu ispod snopa banana,
- popesti se na stolicu i
- dohvatiti banane.

Zadatak:

- Uporabom FOPL prikaži činjenice iz svijeta "majmun-banane"
- Uporabom rezolucije opovrgavanjem dokaži da majmun može dohvatiti banane

80



PRIMJER REZOLUCIJE

- U kreiranju baze znanja važno je odrediti:
 - sve relevantne objekte (majmun, banane, stolica, pod)
 - odnose među njima (npr. banane nisu blizu poda, stolica se može pomaknuti ispod banana).
- Sve nevažno treba izostaviti (npr. prozori, vrata...)

KONSTANTE

pod, stolica, banana, majmun

VARIJABLE

x, y, z

PREDIKATI

MOŽE_DOHVATITI(x, y)	<i>x može dohvatiti y</i>
SPRETAN(x)	<i>x je spretan</i>
BLIZU(x, y)	<i>x je blizu y</i>
JE_NA(x, y)	<i>x je na y</i>

81

ISPOD(x, y)	<i>x je ispod y</i>
VISOKA(x)	<i>x je visok(a)</i>
U_SOBI(x)	<i>x je u sobi</i>
MOŽE_POMAKNUTI_BLIZU(x, y, z)	<i>x može pomaknuti y blizu z</i>
MOŽE_POPETI_NA(x, y)	<i>x se može popeti na y</i>

AKSIOMI BAZE ZNANJA

U_SOBI(banane)
U_SOBI(stolica)
U_SOBI(majmun)
VISOKA(stolica)
SPRETAN(majmun)
MOŽE_POMAKNUTI_BLIZU(majmun, stolica, banana)
MOŽE_POPETI_NA(majmun, stolica)
~BLIZU(banane, pod)

82

PRIMJER REZOLUCIJE

$$(U_SOBI(x) \wedge U_SOBI(y) \wedge U_SOBI(z) \wedge MOŽE_POMAKNUTI_BLIZU(x, y, z)) \rightarrow BLIZU(z, pod) \vee ISPOD(y, z)$$

$$MOŽE_POPETI_NA(x, y) \rightarrow JE_NA(x, y)$$

$$(JE_NA(x, y) \wedge ISPOD(y, banane) \wedge VISOK(y)) \rightarrow BLIZU(x, banane)$$

$$(SPRETAN(x) \wedge BLIZU(x, y)) \rightarrow MOŽE_DOHVATITI(x, y)$$

- Aksiomi se mogu pretvoriti u klauzalnu formu (uporabom De Morganovih zakona i ekvivalencije $P \rightarrow Q \equiv \neg P \vee Q$)

83

Klauzalni oblik baze znanja

1. U_SOBI(banane)
2. U_SOBI(stolica)
3. U_SOBI(majmun)
4. VISOKA(stolica)
5. SPRETAN(majmun)
6. MOŽE_POMAKNUTI_BLIZU(majmun, stolica, banana)
7. MOŽE_POPETI_NA(majmun, stolica)
8. ~BLIZU(banane, pod)
9. ~MOŽE_POPETI_NA(x, y) \vee JE_NA(x, y)
10. ~SPRETAN(x) \vee ~BLIZU(x, y) \vee MOŽE_DOHVATITI(x, y)
11. ~JE_NA(x, y) \vee ~ISPOD(y, banane) \vee ~VISOK(y) \vee BLIZU(x, banane)
12. ~U_SOBI(x) \vee ~U_SOBI(y) \vee ~U_SOBI(z) \vee ~MOŽE_POMAKNUTI_BLIZU(x, y, z) \vee BLIZU(z, pod) \vee ISPOD(y, z)
13. ~MOŽE_DOHVATITI(majmun, banana) (negacija cilja)

84

PRIMJER REZOLUCIJE

Dokaz rezolucijom

14. $\sim \text{MOŽE_POMAKNUTI_BLIZU}(\text{majmun}, \text{stolica}, \text{banane}) \vee \text{BLIZU}(\text{banane}, \text{pod}) \vee \text{ISPOD}(\text{stolica}, \text{banane})$
(resolventa od [1], [2], [3] i [12] uz supstituciju {majmun/x, stolica/y, banane/z})
15. $\text{BLIZU}(\text{banane}, \text{pod}) \vee \text{ISPOD}(\text{stolica}, \text{banane})$
(resolventa od [6], [14])
16. $\text{ISPOD}(\text{stolica}, \text{banane})$
([8] i [15])
17. $\sim \text{JE_NA}(x, \text{stolica}) \vee \sim \text{VISOK}(\text{stolica}) \vee \text{BLIZU}(x, \text{banane})$
(supstitucija {stolica/y} i [16] i [11])
18. $\sim \text{JE_NA}(x, \text{stolica}) \vee \text{BLIZU}(x, \text{banane})$ ([4] i [17])



85

PRIMJER REZOLUCIJE

19. $\sim \text{JE_NA}(\text{majmun}, \text{stolica})$ ([7] i [9])
20. $\text{BLIZU}(\text{majmun}, \text{banane})$
([18] i [19] supstitucija {majmun/x})
21. $\sim \text{BLIZU}(\text{majmun}, y) \vee \text{MOŽE_DOHVATITI}(\text{majmun}, y)$
([10] i [5] supstitucija {majmun/x})
22. $\text{MOŽE_DOHVATITI}(\text{majmun}, y)$
([20] i [21] supstitucija {banane/y})
23. {}
([13] i [22])



86

PRIMJER REZOLUCIJE

- Ovaj dokaz nije vođen niti jednom posebnom strategijom iako se vodilo računa o odabiru roditeljskih klauzula. U suprotnom, puno nepotrebnih koraka može biti učinjeno.

Zadatak

- Možete li komentirati koji su svi oblici rezolucije učinjeni kroz korake od [14] do [23]?



87

Primjer: Financijski savjetnik



88

CASE STUDY - Financijski savjetnik

- Uporaba predikatne logike u predstavljanju znanja i zaključivanju iz problemske domene – financijsko savjetovanje
- Pomoć u odluci da li sredstva:
 - pohraniti na bankovni račun (štednja)
 - ulagati u dionice
 - kombinirati gornje
- Preporuke su individualne **i zavise od svakog pojedinog ulagača tj. od:**
 1. **uštede** ulagača – koja zavisi o stanju na bankovnom računu pojedincu i broj uzdržavanih članova obitelji.
 2. **prihoda** – iznos prihoda, broj uzdržavanih članova obitelji, stalnost prihoda

89

Preporuke:

1. Pojedinci sa **neodgovarajućom uštedom** trebaju uvijek prvo povećati uštedu, bez obzira na prihod.
2. Pojedinci sa **odgovarajućom uštedom i odgovarajućim prihodima** trebaju razmotriti profitabilnija i rizičnija ulaganja u dionice.
3. Pojedinci s **neodgovarajućim prihodima i odgovarajućom uštedom** trebali bi razdijeliti svoja ulaganja između:
 - štednje (povećaju svoju sigurnost) i
 - ulaganja u dionice (pokušaju povećati prihode)

90

CASE STUDY - Financijski savjetnik

- **Odgovarajuća ušteda** – barem 5 000 \$ po uzdržavanom članu obitelji.
- **Odgovarajući prihod** – barem 15 000 \$ godišnje plus 4000\$ po uzdržavanom članu obitelji.

Kako modelirati ova znanja u predikatnoj logici?

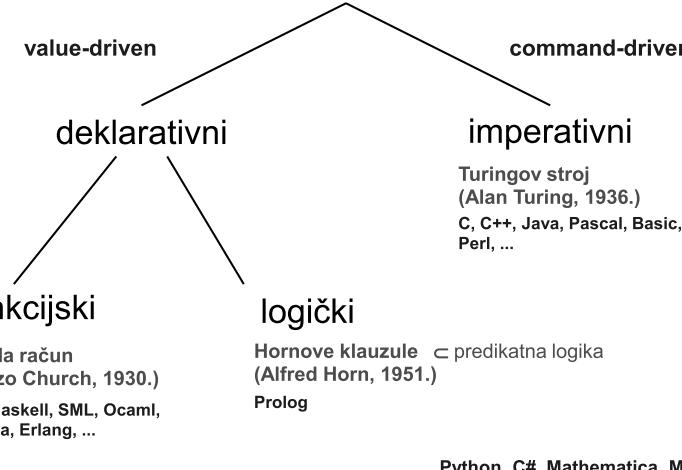
91

Fakultet elektrotehnike i računarstva
Zavod za elektroniku, mikroelektroniku, računalne i inteligentne sustave

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

Logičko programiranje u Prologu

PROGRAMSKI JEZICI



- **Logičko programiranje** (engl. logic programming)
 - uporaba matematičke logike za programiranje
- **Ideja:** opisati problem logičkim formulama, a rješavanje prepustiti računalu
- **"Algorithm = Logic + Control"**
- Različito od dokazivača teorema (ATP) jer:
 - U program su ugrađeni eksplicitni kontrolni mehanizmi
 - Nije podržana puna ekspresivnost logike

DEKLARATIVNI PROGRAMSKI JEZICI

- deklarativni jezici opisuju **što** se izračunava umjesto **kako** se to izračunava
 - zadaje se specifikacija **skupa uvjeta** koji definiraju prostor rješenja
 - **pronalaženje rješenja** prepušteno je interpretatoru
- osnovne karakteristike:
 - **eksplicitno stanje** umjesto implicitnog stanja
 - nema popratnih efekata (engl. side-effects)
 - programiranje s **izrazima**
 - funkcijski jezici: izraz=funkcija
 - logički jezici: izraz=relacija

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

```
function main()
begin
y = 5
x = foo(2) + y
return x
end
```

- koja je povratna vrijednost funkcije **main** ?
- povratna vrijednost **ovisi o redoslijedu** izračuna pribrojnika!
- deklarativno ne bi smjelo biti razlike:
 - $A+B = B+A$

popratni efekt!

“prave funkcije” ne prtljaju po memoriji već samo vraćaju vrijednost

5

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

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

false???

- funkcija **foo** ima popratni efekt stoga **nije referencijalno prozirna**
 - ne vrijedi Leibnizovo pravilo: “**equals for equals**”
- moramo voditi računa o **tijeku izvođenja** (proceduralno) umjesto da se koncentriramo na **značenje programa** (deklarativno)

6

- čisto deklarativni** (engl. *purely declarative*) jezici ne dozvoljavaju popratne efekte
 - Haskell, ...
- radi praktičnosti većina deklarativnih jezika dopušta kontrolirane popratne efekte (“*declarative in style*”)
 - Lisp, SML, Ocaml, **Prolog**, ...
- pogodnosti:
 - formalno koncizni, visoka razina apstrakcije
 - lakša formalna verifikacija
 - manja mogućnost pogreške
- nedostaci:
 - neučinkovitost
 - neke strukture/funkcije iziskuju popratne efekte (npr.?)

7

- deklarativni jezici **nemaju varijabli** u klasičnom smislu (“*variables do not vary*”)
- nemaju naredbe pridruživanja** ($x = x + 1$) jer bi to iziskivalo popratni efekt
- nemaju programskih petlji**
 - umjesto toga: rekurzija

```
function fact(n)
begin
a = 1
for i = 1 to n do
begin
a = a*i
end
return a
end
```

Haskell:

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

8

- **Prolog = Programming in Logic**
 - deklarativni logički programske jezike
 - 1972.: Alan Colmerauer, Robert Kowalski, Philippe Roussel
 - originalno razvijen za NLP
- fundamentalni koncepti:
 - **rekurzija**
 - **unifikacija**
 - postupak vraćanja (**backtracking**)
- nije čisto deklarativen:
 - "Algorithm = Logic + Control"
 - $(A \wedge B) \rightarrow C \neq (B \wedge A) \rightarrow C$

9

- Programi u Prologu sačinjeni su od slijeda **pravila**
- **Svako pravilo je FOPL-formula u Hornovom obliku:**
 - $\neg P_1 \vee \neg P_2 \vee \neg P_3 \vee \dots \vee \neg P_n \vee Q$
 - tj. klauzula u kojoj je najviše jedan literal pozitivan
- **Ekvivalentno:** $(P_1 \wedge P_2 \wedge P_3 \wedge \dots \wedge P_n) \rightarrow Q$
- Specijalan slučaj za $P_i = \text{true}$: $\text{true} \rightarrow Q == Q$
- **Definitna Hornova klauzula:** točno jedan literal je pozitivan
- Nažalost, ne može se svaka formula pretvoriti u (definitni) Hornov oblik $\neg P \rightarrow Q, P \rightarrow (Q \vee R)$
- Zaključivanje nad Hornovim formulama: **rezolucija opovrgavanjem** (krenuvši od $\neg Q$)

10

Primjer programa u Prologu

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

- baza znanja (logički program):

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

- varijable velikim, predikatni simboli malim slovima
- implikacija u obliku **konzekvens :- antecedens**
- svaki redak završava točkom
- varijable su implicitno univerzalno kvantificirane

11

Primjer upita (1)

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

- sada možemo **postavljati upite**:

```
?- covjek(sokrat).
Yes
?- smrtan(sokrat).
Yes
?- smrtan(X).
X=sokrat
Yes
```

12

Konjunkcija atoma

- antecedens može sadržavati veći broj atoma:

```
covjek(X) :-  
    sisavac(X), govorl(X).
```

operator "i"

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

13

Disjunkcija atoma

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

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

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

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

- provjerite!

operator "ili"

14

Višemjesni predikati

- predikati mogu biti *n*-mjesni:

```
ucitelj(sokrat,platon).  
ucitelj(kratil,platon).  
ucitelj(platon,aristotel).
```

- kako definirati predikat **ucenik** pomoću **ucitelj** ?

```
ucenik(X,Y) :- ucitelj(Y,X).
```

- kako definirati jednomjesni predikat **ucen** ?

```
ucen(X) :- ucitelj(Y,X).
```

15

Kvantifikacija varijabli

```
ucen(X) :- ucitelj(Y,X).
```

- kako je kvantificirana varijabla Y?
- sve varijable su implicitno **univerzalno** kvantificirane u cijelom pravilu, ali Y se javlja samo u antecedensu pa možemo reći da je **egzistencijalno** kvantificirana
- dokaži:

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

16

- primjeri upita:

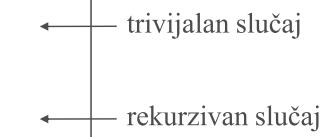
```
?- ucitelj(X,platon).
X=sokrat
X=kratil
Yes
?- ucitelj(sokrat,Y),ucitelj(kratil,Y).
Y=platon
Yes
?- ucenik(X,_).
X=platon
X=platon
X=aristotel
Yes
?- ucenik(aristotel,sokrat).
No
```

17

- definirajmo **tranzitivnu** relaciju **SLJEDBENIK(x,y)**

- “x je ucenik od y”
- “x ucenik od nekog z, a z je sljedbenik od y”

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



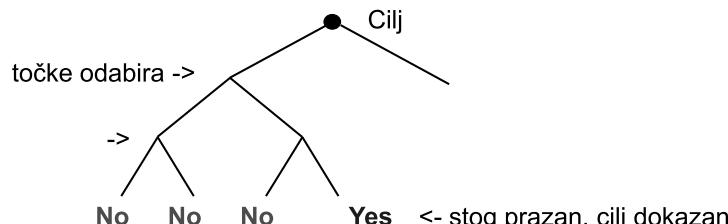
- upit:

```
?- sljedbenik(aristotel,sokrat).
Yes
```

18

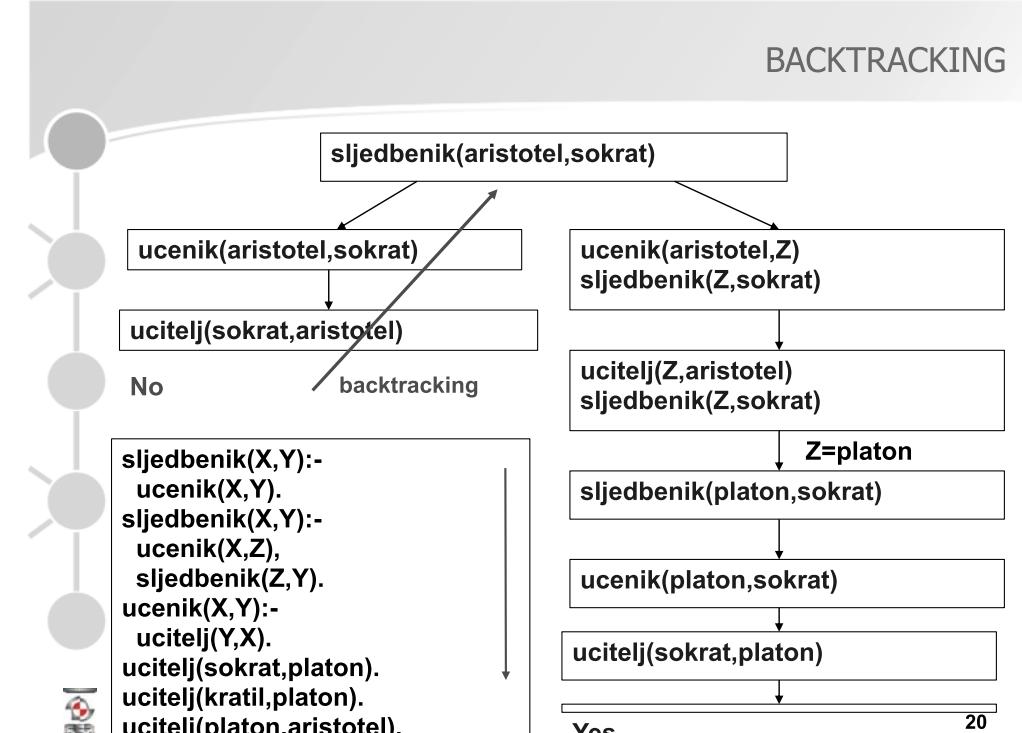
BACKTRACKING

- Prolog dokazuje **unatrag** od cilja k premisama, metodom **pretraživanja u dubinu**
 - antecedens postaje novi medjucilj koji treba dokazati i koji se dodaje na stog**
- Kada jedna grana dokaza ne uspije, Prolog se vraća (BACKTRACKS) na zadnju točku odabira
- Ako iscrpi sve mogućnosti, vraća **No**, inače **Yes**



19

```
sljedbenik(X,Y):-  
  ucenik(X,Y).  
sljedbenik(X,Y):-  
  ucenik(X,Z),  
  sljedbenik(Z,Y).  
ucenik(X,Y):-  
  ucitelj(Y,X).  
ucitelj(sokrat,platon).  
ucitelj(kratil,platon).  
ucitelj(platon,aristotel).
```



20

- Nažalost, Prolog nije čisto deklarativen pa je redoslijed stavaka i atoma itekako bitan:

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

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

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

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

- “out of stack”

21

- U svakom koraku zaključivanja Prolog nastoji **unificirati** trenutni cilj sa stoga s konzeksima pravila ili činjenicama u bazi znanja
- unificirati možemo i eksplicitno:

?- **ucitelj(Z,aristotel) = ucitelj(platon,aristotel)**

Z = platon

Yes

operator
unifikacije

?- **ucenik(Z,aristotel) = ucenik(platon,Z)**

No

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

X = f(f(f(...)))

nema provjere pojavljivanja
variabile! (occurs check)



22

NEGACIJA

- Hornov oblik ne dopušta negaciju u antecedensu, ali Prolog dopušta operator **not**

```
covjek(X) :-  
  govori(X),  
  not(imam(X,perje)).
```

- negacija pomoću neuspjeha (NAF)**
 - ako **P(x)** ne možeš dokazati,
onda je istinito **not(P(x))**, inače je lažno
- prepostavljamo **zatvorenost svijeta (CWA)**
 - sve što nije u bazi znanja je lažno

23

- baza znanja:

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

- upiti:

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



24

NEGACIJA

- **Zadatak 2.5: Primjena Prologa u porodičnoj domeni**

(3 boda)

U programskom jeziku Prolog definirajte bazu znanja koja opisuje porodične relacije i omogućuje izvođenje novih zaključaka. U bazi znanja definirajte činjenice **roditelj**, **musko**, **zensko** i **dob**, te zatim pravila **otac**, **majka**, **brat**, **sestra**, **djed**, **baka**, **ujak**, **predak** i **srodan**.

Definirajte upite kojima se iz baze znanja dohvaćaju **(1)** svi roditeljski parovi, **(2)** sve osobe koje su nekome ujak, **(3)** sva djeca koja imaju stariju braću, **(4)** sve prabake koje su to već deset godina, **(5)** svi potomci neke zadane osobe, **(6)** sve punoljetne osobe bez djece, **(7)** svi parovi osoba koje nisu u srodstvu, **(8)** sve majke s troje ili više djece, **(9)** sve majke s točno troje djece te **(10)** sve majke s troje ili manje djece.

```
roditelj(zeus,atena).
```

```
roditelj(zeus,apolo).
```

```
roditelj(hera,ares).
```

...

```
musko(zeus).
```

```
zensko(hera).
```

...

```
otac(X,Y):- ...
```