

1 Uvod u umjetnu inteligenciju

1.1. Opisati neke povijesne i fikcijske pokušaje razvoja UI

Povijesni pokušaji: Turčin

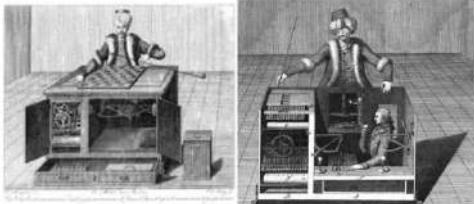
Povijesni pokušaji: Frankenstein

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



B. Wrightson: Frankenstein creates the fiend

- 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
- Vješt konstruirana mehanička iluzionistička naprava



Računala i elektronički mozak

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



1.2. Navesti primjere uspješnih i poznatih UI sustava

- Deep blue (IBM) – šah, pobijedio Kasparova
- IBM Watson – pobijedio u igri Jeopardy (pokazao znanja obrade prirodnog jezika, zaključivanja, pretraživanja informacija)
- WolframAlpha

1.3. Objasniti i ilustrirati koncept UI-potpunog problema

To su računalni problemi čija je složenost ista izgradnji stroja koji je intelligentan koliko i čovjek.

Cilj je napraviti AI koji je intelligentan koliko i čovjek.

Primjeri:

- računalni vid
- natural language processing and understanding
- dealing with unexpected circumstances while solving any real-world problem.

1.4. Uspoređiti sposobnosti čovjeka i UI u igri šah

Deep Blue vs. Garry Kasparov (2)

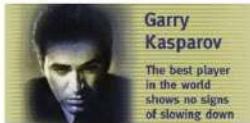
Deep Blue vs. Garry Kasparov (3)



200,000,000 šahovskih pozicija u sekundi

Poseđuje **мало znanja** o šahu, ali **огромну sposobnost** izračunavanja

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



3 šahovske pozicije u sekundi

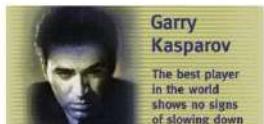
Poseđuje **mnogo znanja** o šahu, ali bitno **manju sposobnost** izračunavanja

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



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 intelligentan. Autor je nekoliko knjiga i govori mnoge jezike

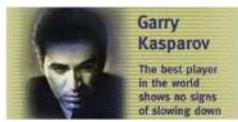
Deep Blue vs. Garry Kasparov (4)



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

Deep Blue je 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 **može promjeniti** svoj način igre

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

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

1.5. Sistematizirati različite pristupe definiciji UI

- Pokušaj sistematizacije definicija:

Razmišljati ljudski	Razmišljati racionalno
Ponašati se ljudski	Ponašati se racionalno

1.6. Navesti osnovna potpodručja UI

- 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 uzorka i analiza scena
- ⑪ Raspodijeljena umjetna inteligencija

1.7. Opisati Turingov test, njegovu svrhu i ograničenja

Motivacija:

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."

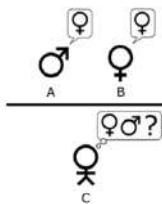


Stvara pokus „igra oponašanja“ - Pokus uspoređuje performanse pretpostavljenog inteligenčnog stroja i čovjeka na temelju nekog skupa upita

Ideja pokusa:

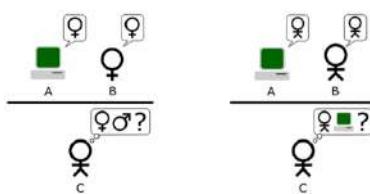
Turingov test

- 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 ispitičuču C
- **Cilj igrača A:** navesti ispitičuča C na pogrešnu identifikaciju
- Pokus se ponavlja i mjeri se uspješnost ispitičuča C



Turingov test

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



Q: Koje bi sposobnosti (inteligenčan) 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

Ograničenja:

Nedostatci Turingovog testa

- Ljudska vs. opća inteligencija (ljudi se ponekad ponašaju neinteligentno, a inteligenčno ponašanje ne mora nužno biti ljudsko)
- Prava vs. simulirana inteligencija (filozofski protuargument ponašajno-orientiranom UI)
- Naivnost ispitičuča (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"

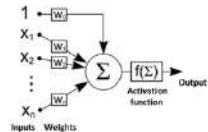


- TT je značajniji za filozofiju UI-a, nego što je to za sam razvoj UI-a

1.8. Opisati osnovna razdoblja i navesti ključne trenutke u povijesti UI

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



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 (2)

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

- 1952. A. Samuel: igra dame, program koji uči
- 1956. Newell, Shaw i Simon: Logic Theorist (LT) – skraćeni dokaz teorema iz knjige *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 perceptronra



1952. – 1969. Otrežnjenje (1)

- Rezultati ranih sustava pokazali su se slabima 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 prevodenja (1957.)

Strojno prevodenje (financirano zbog ubrzavanja prevodenja 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"



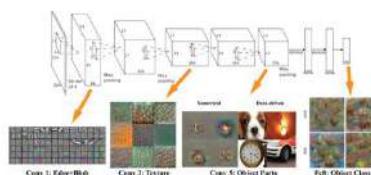
1952. – 1969. Otrežnjenje (2)

1970. – 1979. Sustavi temeljeni na znanju

- Drugi veliki problem – **netraktabilnost** mnogih problema koje je UI pokušavao riješiti
- Početni uspjeh bio je moguć zbog malog broja kombinacija u mikrosvjetovima (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

2010. – danas

- Era **dubokog učenja** (deep learning)
- Duboko učenje – učenje višeslojnih apstrakcija podataka kroz višeslojne neuronske mreže
- Strojno učenje na velikim količinama podataka
- Veliki napretci u računalnom vidu, obećavajući napretci u obradi prirodnog jezika



1980. – 2010.

- 1980 – UI je industrija! (od nekoliko milijuna 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 – percepција okoline kroz senzore i djelovanje na sredinu kroz akcije)
- **Robotika**
- **Strojno učenje**

- 40. godine: razvoj umjetnog neurona, Turingov test, prva neuronska mreža
- 50. i 60. godine: velika očekivanja ali i velika razočaranja jer se vidjelo da nije sve tako lijepo (neuspjeno strojno prevodenje, netraktabilnost, teško skaliranje na veće probleme)
- 70. godine: razvoj sustava temeljenih na znanju (PROLOG)
- 80.–10. godine: skok u razvoju UI, povratak neuronskih mreža, razvoj inteligentnih agenata, robotike, strojnog učenja
- 10.+ godine: razvoj dubokog učenja zbog velike količine podataka

2 Pretraživanje prostora stanja

2.1. Opisati formalnu definiciju problema pretražvanja prostora stanja i dati primjere

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)

Primjeri:

Tipični problemi...



i pronalazak puta od Pule do Buzeta

2.2. Razlikovati između prostora stanja i stabla pretraživanja

Prostor stanja = prostor (skup) stanja s definiranim vezama pomoću kojeg možemo izgraditi graf te taj graf možemo pretraživati (pretražujemo prostor stanja).

Pretraživanje prostora stanja = pretraživanje usmjerenog grafa gdje su vrhovi stanja, lukovi prijelazi i može imate težine (cijene prijelaz)

Stablo pretraživanja = stablo koje gradimo pretraživanjem usmjerenog grafa tako da proširujemo čvorove tj. generiramo sljedbenike funkcijom sljedbenika

- Može biti beskonačno čak i kad je prostor stanja konačan (ciklusi)
- Otvoreni čvor – generiran ali nije obiđen
- Zatvoreni čvor – obiđen
- Čvor pohranjuje stanje i dubinu čvora u stablu

2.3. Definirati i objasniti opći algoritam pretraživanja

Opći algoritam pretraživanja

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

- removeHead(l) – skida prvi element neprazne liste l
- expand(n , succ) – proširuje čvor n uporabom funkcije sljedbenika succ
- insert(n, l) – umeće čvor n u listu l
- Proširivanje čvora treba ažurirati sve komponente čvora:

Proširivanje čvora

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

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

2.4. Definirati osnovna svojstva problema pretraživanja prostora stanja i dati primjere

Karakteristike problema:

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

2.5. Analizirati dani problem pretraživanja prema njegovim osnovnim svojstvima

- Odrediti broj stanja, faktor grananja, dubinu optimalnog rješenja i maksimalnu dubinu stabla

2.6. Definirati osnovna svojstva algoritama pretraživanja prostora stanja

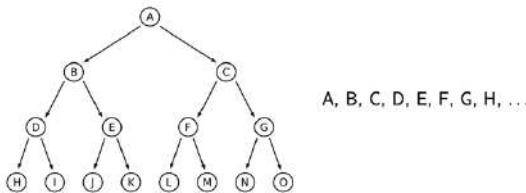
Svojstva algoritama:

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

2.7. Definirati i objasniti algoritam pretraživanja u širinu i njegova svojstva

Pretraživanje u širinu

- Jednostavna slijepa strategija pretraživanja
- Nakon proširenja korijenskog čvora, proširuju se sva njegova djeca, zatim sva djeca djece, 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 – 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. $b = 4, d = 16, 10\text{ B}/\text{čvor} \rightarrow 43\text{ GB}$
- Primjenjivo samo na male probleme

Pretraživanje u širinu – izvedba

- Ovakvu strategiju ostvarit ćemo ako generirane 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*)

2.8. Definirati i objasniti algoritam pretraživanja s jednolikom cijenom i njegova svojstva

Pretraživanje s jednolikom cijenom

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

Pretraživanje s jednolikom cijenom

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

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

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

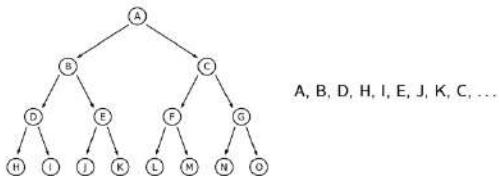
$$\begin{aligned}
 b &= 4 & \text{ukupno učvraje} &= 4,3 \cdot 10^9 \text{ čvor} \cdot \frac{10}{\text{čvor}} \frac{\text{B}}{\text{čvor}} &= 4,3 \cdot 10^9 \text{ B} \\
 d &= 16 & \downarrow & & = 43 \text{ GB} \\
 10 \text{ B}/\text{čvor} & & \hline & & \\
 b^d &= 4^b & 4,3 \cdot 10^9 & &
 \end{aligned}$$

koliko učvraje?

2.9. Definirati i objasniti algoritam pretraživanja u dubinu i njegova svojstva

Pretraživanje u dubinu

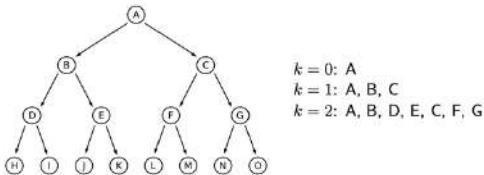
- Pretraživanje u dubinu uvijek prvo proširuje najdublji čvor u stablu pretraživanja
- Postupak se vraća na pliću razinu tek kada dosegne listove (stanja koja nemaju sljedbenika)



- Pretraživanje u dubinu manje je memoriski 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



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

Pretraživanje u dubinu – izvedba

- Strategiju pretraživanja u dubinu ostvarit ćemo ako generirane 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 – rekurzivna izvedba

- Listu *open* možemo izbjegći:

```
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 eksplicitne liste *open* koristi se sistemski stog
- Uz ljenju (nestriktnu) evaluaciju skupa *succ(s)*: prostorna složenost je $\mathcal{O}(m)$ umjesto $\mathcal{O}(bm)$

Ograničeno pretraživanje u dubinu – izvedba

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

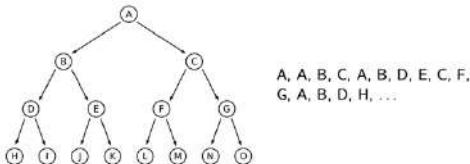
Ograničeno pretraživanje u dubinu

```
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
```

2.10. Definirati i objasniti algoritam iterativnog pretraživanja u dubinu i njegova svojstva

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



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
```

- 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 preostalih čvorova na višim razinama nije problematično
- Vremenska složenost:** $\mathcal{O}(b^d)$
- Prostorna složenost:** $\mathcal{O}(bd)$
- Potpunost:** da, jer koristi dubinsko ograničenje i povećava ga
- Optimalnost:** da, jer pretražuje razinu po razinu
- Iterativno pretraživanje u dubinu preporučena je strategija za probleme s velikim prostorom stanja i nepoznatom dubinom rješenja

2.11. Primjeniti algoritme slijepog pretraživanja na jednostavne probleme

- Pogledati na slike stabala za svaki algoritam gore naveden

2.12. Vrednovati algoritme slijepog pretraživanja po njihovim osnovnim svojstvima

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	$\mathcal{O}(b^k)$	$\mathcal{O}(bk)$	Ne	Ne
Iter. u dubinu	$\mathcal{O}(b^d)$	$\mathcal{O}(bd)$	Da	Da

b – faktor grananja, d – dubina optimalnog rješenja,
 m – maksimalna dubina stabla ($m \geq d$), k – dubinsko ograničenje

- Svi algoritmi pretraživanja nažalost imaju **eksponencijalnu vremensku složenost**

- Kod velikog broja stanja preporuča se koristiti **iterativno pretraživanje u dubinu**

- Iterativno umjesto u širinu zbog manje prostorne složenosti (kad imamo puno stanja vide se velike razlike)

3 Heurističko pretraživanje

3.1. Razlikovati između algoritama slijepog i heurističkog pretraživanja prostora stanja

- 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?
→ koristiti heurstiku

3.2. Definirati i objasniti heurističku funkciju i dati primjer

- **Heuristika** – iskustvena pravila o prirodi problema i osobinama cilja čija je svrha pretraživanje brže **usmjeriti** 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 ciljnemu stanju. Ako je s ciljno stanje, onda $h(s) = 0$

- Postupke pretraživanja koji koriste heurstiku kako bi suzili prostor pretraživanja nazivamo **heurističkim** ili **usmjerenim**
 - Heurističko pretraživanje = pretraživanje koje koristi heurstiku i tako smanjuje prostor stanja koji pretražujemo

3.3. Definirati i objasniti pohlepno pretraživanje najbolji prvi i njegova svojstva

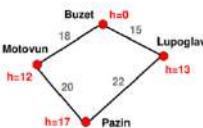
- Proširuje čvor s najboljom heurističkom vrijednošću

Pretraživanje "najbolji prvi"

```
function greedyBestFirstSearch( $s_0$ , succ, goal,  $h$ )
    open ← [initial( $s_0$ )]
    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  $f(n) = h(\text{state}(n))$ 
```

- 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:** $\mathcal{O}(b^m)$

3.4. Definirati i objasniti pretraživanje usponom na vrh i njegova svojstva

- Kao pohlepno pretraživanje "najbolji prvi", s tom razlikom da se generirani 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  $f(n) = h(\text{state}(n))$ 
```

- 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:** $\mathcal{O}(1)$

3.5. Definirati i objasniti algoritam A* i njegova svojstva

- 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

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

- 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 zaobide optimalni put jer se on čini skupljim nego što zapravo jest
- Q:** Jesu li heuristike u ranijim primjerima optimistične?

Algoritam A^*

```
function aStarSearch( $s_0$ , succ, goal,  $h$ )
    open  $\leftarrow [\text{initial}(s_0)]$ 
    closed  $\leftarrow \emptyset$ 
    while open  $\neq []$  do
         $n \leftarrow \text{removeHead}(open)$ 
        if goal(state( $n$ )) then return  $n$ 
        closed  $\leftarrow closed \cup \{n\}$ 
        for  $m \in \text{expand}(n, \text{succ})$  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  $\cup$  open)
            insertSortedBy(f,  $m$ , open)
    return fail
where  $f(n) = g(n) + h(\text{state}(n))$ 
```

3.6. Definirati optimističnost heuristike i dati primjere

- Slika gore

3.7. Definirati i objasniti konzistentnost heuristike i njezin utjecaj na algoritam A*

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 ciljnem stanju vrijedi $f(n) = g(n) = C^*$
- Poželjno je da $f(n)$ monotono 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 $f(n)$ monotono raste, svaki čvor koji prvi ispitamo (i zatvorimo) za neko stanje bit će čvor s najmanjom cijenom za to stanje
- To znači, kod ponovljenih stanja, ne moramo provjeravati cijenu već zatvorenih čvorova (ona će sigurno biti manja ili jednaka)

- Ako $a(9) - - 6 - - > b(2)$ da bi heuristika bila konzistentna mora vrijediti: $9 \leq 6+2$ (ovdje to ne vrijedi)

Konzistentna heuristika (2)

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

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

Konzistentnost heuristike

Heuristika h je konzistentna ili monotona akko:

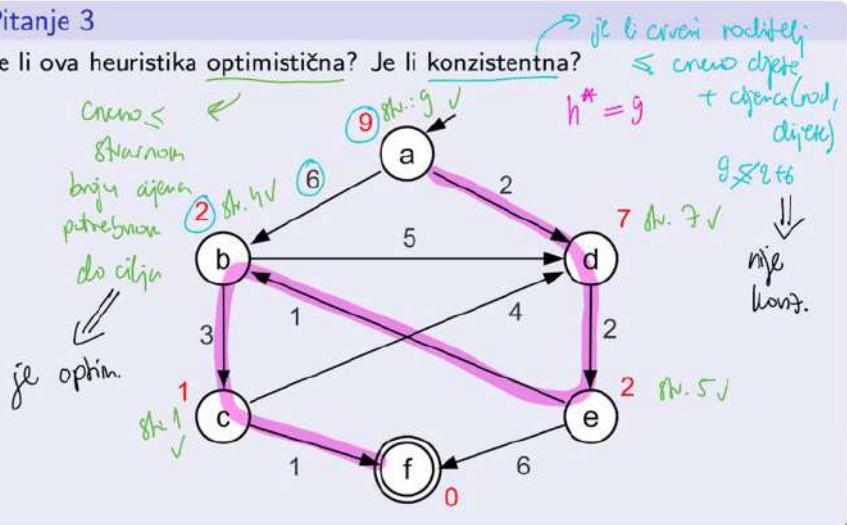
$$\forall (s_2, c) \in \text{succ}(s_1). h(s_1) \leq h(s_2) + c$$

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

3.8. Odrediti je li heuristika za dani problem optimistična ili konzistentna

Pitanje 3

Je li ova heuristika optimistična? Je li konzistentna?



3.9. Objasniti i primijeniti oblikovanje heuristike metodom relaksacije problema

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 obavještenog 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!

Oblikanje heuristike

Relaksacija problema

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

Kombiniranje optimističnih heuristika

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

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

Cijena rješavanja podproblema

- ▶ baza uzorka koja sadržava optimalne cijene pojedinih podproblema

Učenje heuristike

- ▶ primjena metoda strojnog učenja. Npr. učimo koeficijente w_1 i w_2 :
$$h(s) = w_1x_1(s) + w_2x_2(s)$$
, gdje su x_1 i x_2 značajke stanja

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?

3.10. Primjeniti algoritme heurističkog pretraživanja na jednostavne probleme

- Nešto tipa zadatak 3.8. pa treba ispisat za svaki algoritam šta će dat

3.11. Vrednovati algoritme heurističkog pretraživanja po njihovim osnovnim svojstvima

Algoritam	Vremenska složenost	Prostorna složenost	Optimalnost?	Potpunost?
Najbolji prvi	$O(b^m)$		NE	NE
Uspon na vrh	$O(m)$	$O(1)$	NE	NE
A*	$O(\min(b^m, b^* S))$		DA ako je heuristika optim.	DA

4 Igranje igara

4.1. Objasniti i ilustrirati determinističku igru s potpunom informacijom i sumom nula

To su igre u kojima sudjeluje 2 igrača kojima je cilj pobijediti.

Svaki igrač ima konačan broj poteza te potpunu informaciju o svim prethodnim potezima.

Takve igre uvijek imaju pobjedničku strategiju za jednog od igrača.

Temelje se na pretraživanju prostora stanja ali postoji i protivnik pa u svakom stanju treba donijeti optimalnu odluku o sljedećem potezu (optimalna strategija).

Igre sa sumom nula su igre u kojima dobitak jednog igrača predstavlja direktni gubitak drugog igrača (zbroj dobitaka oba igrača je 0).

Primjeri:

- Šah
- Križić-kružić
- Go

4.2. Definirati igranje igara kao problem pretraživanja prostora stanja

Problem pretraživanja sa sljedećim komponentama:

Igra

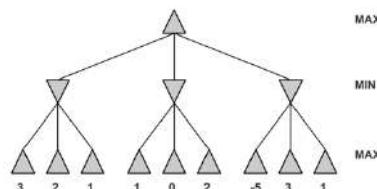
- Početno stanje igre s_0
- Funkcija sljedbenika $\text{succ} : S \rightarrow \wp(S)$, koja definira valjane poteze igre (prijelaze između stanja)
- Test na završno stanje $\text{terminal} : S \rightarrow \{\top, \perp\}$
- Isplatna funkcija utility : $S \rightarrow \mathbb{R}$ koja pridjeljuje numeričku 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

4.3. Objasniti metodu minimax i prepostavke na kojima se temelji

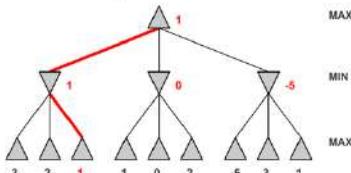
Metoda minimaks

- Igrače ćemo nazvati MAX (računalno) 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



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

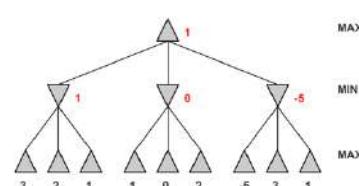


- Da bismo odredili optimalnu strategiju onog igrača koji je upravo na redu, trebamo izračunati minimaks-vrijednost korijenskog čvora

Minimaks-vrijednost

- Minimaks-vrijednost čvora s definirana je rekurzivno:

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



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

- 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 povukli optimalan potez, svaki put kada su na redu igrači moraju novovo izračunati svoju optimalnu strategiju, krenuvši od trenutne pozicije igre u korijenu stabla
- Minimax pretražuje u dubini, pa je njegova prostorna složenost $O(m)$, gdje je m dubina stabla pretraživanja
- Međutim, vremenska složenost je $O(b^m)$, gdje je b faktor grananja igre. To je vrlo nezgodno!

4.4. Definirati algoritam minimax s dubinskim ograničenjem i objasniti njegova svojstva

Algoritam minimaks s presjecanjem

```

function maxValue(s, d)
    if terminal(s) then return utility(s)
    if d = 0 then return h(s)
    m  $\leftarrow -\infty$ 
    for t  $\in \text{succ}(s)$  do
        m  $\leftarrow \max(m, \text{minValue}(t, d - 1))$ 
    return m

function minValue(s, d)
    if terminal(s) then return utility(s)
    if d = 0 then return h(s)
    m  $\leftarrow +\infty$ 
    for t  $\in \text{succ}(s)$  do
        m  $\leftarrow \min(m, \text{maxValue}(t, d - 1))$ 
    return m
```

- Ograničeno pretraživanje u dubinu svojstva:
 - o Prostorna složenost.: $O(m)$
 - o Vremenska složenost: $O(b^m)$
 - o Nije optimalan niti potpun

4.5. Objasniti heurističku funkciju igre i navesti primjere

Pretraživanje stabla presjećemo na određenoj dubini *d* i radimo procjenu vrijednosti isplatne funkcije u obliku heurističke funkcije *h*(*s*).

To radimo jer u stvarnosti nije moguće potpuno pretražiti stablo igre.

Vrijednost *h*(*s*) je procjena isplativosti stanja *s* za igrača MAX.

Često definirana kao težinska linearna kombinacija više značajki:

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

U većini slučajeva svaki igrač ima svoju heurističku funkciju.

4.6. Primjeniti algoritam minimax s danom heurstikom na jednostavno stablo igre

4.7. Objasniti podrezivanje alfa-beta i dati primjer

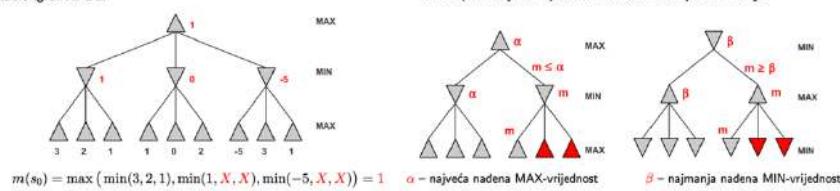
- Cilj: smanjiti broj čvorova koje treba ispitati

Podrezivanje alfa-beta

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

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 podrezujemo ispod MIN-čvora: **alfa-podrezivanje**
- Ako podrezujemo ispod MAX-čvora: **beta-podrezivanje**



4.8. Primjeniti algoritam minimax s podrezivanjem alfa-beta na jednostavno stablo igre

5 Prikazivanje znanja formalnom logikom

5.1. Objasniti ulogu prikazivanja znanja u UI i dati primjere

- **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.

Primjeri:

- Virtualni asistenti
- Inteligentni agenti
- Svijet Wumpusa (ona igra na ploči gdje istraživač mora doći do zlata bez da upadne u rupe, čudovišta ...)

5.2. Objasniti kompromis izmedu ekspresivnosti i odlučivosti logike

- Postoje razne vrste logike koje se razlikuju u navedene tri komponente
- Neke su više a neke manje **ekspresivne** (izražajne)
- Prednosti visoke ekspresivnosti:
 - ▶ detaljan opis stvarnog svijeta
- Nedostatci visoke ekspresivnosti:
 - ▶ složenija sintaksa, semantika i teorija dokaza
 - ▶ ne možemo sve dokazati ⇒ neodlučivost
- 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.

5.3. Objasniti ontološke i epistemološke prepostavke i dati primjere

Ontološke prepostavke (engl. *ontological commitments*)

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

Epistemološke prepostavke (engl. *epistemological commitments*)

Epistemološke prepostavke 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 prepostavke koje omogućavaju iskazivanje vjerovanja i različitih stupnjeva pouzdanosti.

5.4. Odrediti vrijednost istinitosti PL formule za danu interpretaciju

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**

- Istinitost PL određuje se pomoću tablice istinitosti

Postupak ovakvih zadataka:

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

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

5.5. Definirati i odrediti valjanost, nekonzistentnost i zadovoljivost PL formule

- 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

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.

5.6. Definirati logičku posljedicu i odrediti vrijedi li za dani primjer

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

P	Q	$P \vee Q$	$\neg P$	$\frac{F}{(P \vee Q) \wedge \neg P}$	$F \rightarrow Q$
\perp	\perp	\perp	\top	\perp	\top
\perp	\top	\top	\top	\top	\top
\top	\perp	\top	\perp	\perp	\top
\top	\top	\top	\perp	\perp	\top

- Dokaz opovrgavanjem:

P	Q	$P \vee Q$	$\neg P$	$\frac{F}{(P \vee Q) \wedge \neg P}$	$\neg Q$	$F \wedge \neg Q$
\perp	\perp	\perp	\top	\perp	\top	\perp
\perp	\top	\top	\top	\top	\perp	\perp
\top	\perp	\top	\perp	\perp	\top	\perp
\top	\top	\top	\perp	\perp	\perp	\perp

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 ".

- Određuje se pomoću tablice istinitosti

5.7. Razlikovati između PL i FOL the objasniti motivaciju za FOL

Razlike:

- U PL postoje samo činjenice koje su T/F a u FOL postoje objekti i odnosi između njih
- FOL je ekspresivnija od PL ali nije odlučiva tj. poluodlučiva je

Motivacija:

- (1) *Svaki student pohađa predavanja.*
- (2) *Ivan je student.*
- (3) *Ivan pohađa predavanja.*

- Ovo jednostavno zaključivanje ne možemo formalizirati u PL
- Zašto? Zato što propozicije nemaju internu strukturu. Imali bismo P, Q, R, bez ikakvih odnosa između njih
- Trebamo moći izraziti **odnose između objekata**: Ivan je pripadnik studenata

5.8. Definirati wff FOL formulu i odrediti je li dana FOL formula wff

Dobro oblikovana formula (wff)

Dobro oblikovana formula (wff) FOL-a definirana je rekurzivno:

- (1) Atom je formula
- (2) Ako je F formula, onda je i $(\neg F)$
- (3) Ako su F i G formule onda su formule i $(F \wedge G)$, $(F \vee G)$, $(F \rightarrow G)$ i $(F \leftrightarrow G)$
- (4) Ako je F formula koja sadrži varijablu x koja nije vezana, onda su $(\forall x)F$ i $(\exists x)F$ također formule
- (5) Ništa drugo nije formula

- Dopuštamo izostavljanje zagrada u pravilu (2), zagrada u pravilu (3) ako su to skroz vanjske zagrade, te zagrada u pravilu (4).

Wff: da ili ne?

- ① $\forall x(L(x, z))$
- ② $(\forall y)(\exists x)P(x, y, z)$
- ③ $\exists xP(x) \rightarrow Q(a)$
- ④ $\exists x(P(x) \rightarrow \forall x \forall y Q(x, y))$
- ⑤ $\forall x \forall y (\forall z P(z, x) \rightarrow Q(y, z))$

1:ne, zbog zagrada oko L (te zagrade niti jedno pravilo ne dopušta)

2:da

3:da

4:ne, jer je x od Q u dosegu 'za svaki' i 'postoji' i to je nemoguće

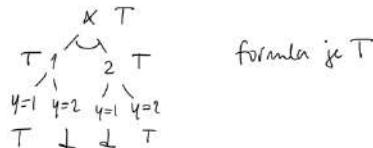
5:da

5.9. Odrediti vrijednost istinitosti FOL formule za danu interpretaciju

Pitanje 1

Odredite vrijednost istinitosti formule $\forall x \exists y P(x, y)$ u interpretaciji s domenom $D = \{1, 2\}$ i sljedećim preslikavanjem za predikatni simbol P :

$P(1, 1)$	$P(1, 2)$	$P(2, 1)$	$P(2, 2)$
\top	\perp	\perp	\top



formula je \top

5.10. Dati primjer modela PL ili FOL formule

5.11. Objasniti što to znači da je FOL neodlučiva

Neodlučivost FOL-a

- **Valjanost, proturječnost i zadovoljivost** definirani su u FOL-u na isti način kao i u PL-u
- U PL-u, formula sa n varijabli ima 2^n interpretacija (zašto?)
- **Q:** Koliko interpretacija ima neka formula u FOL-u?
- **A:** Beskonačno mnogo!
 - ⇒ Postoji beskonačno mnogo domena D (neke od njih su i same beskonačne!)
 - ⇒ Zbog toga u FOL-u ne radimo s tablicama istinitosti!
- **Q:** Kako to utječe na semantiku (konkretno: na logičku posljedicu)?
- **A:** Ne možemo dokazati logičku posljedicu iscrpnim pobrojavanjem interpretacija ⇒ **neodlučivost**
- Budući da ne možemo pobrojavati, trebamo naći neki učinovitiji postupak ⇒ **teorija dokaza**
- Koju god metodu koristili, ona će biti ograničena zbog **neodlučivosti**

5.12. Odrediti i provjeriti FOL formulu za zadalu jednostavnu rečenicu prirodnog jezika

- Ivan je marljiv (M) student (S).

$$M(Ivan) \wedge S(Ivan)$$

- Svi studenti su pametni (P).

$$\forall x(S(x) \rightarrow P(x))$$

- Niti jedan student nije pametan.

$$\forall x(S(x) \rightarrow \neg P(x)) \equiv \forall x \neg(S(x) \wedge P(x)) \equiv \neg \exists x(S(x) \wedge P(x))$$

- Neki studenti su pametni.

$$\exists x(S(x) \wedge P(x))$$

6 Automatsko zaključivanje

6.1. Razlikovati između dokazivanja logičkih i deduktivnih posljedica

- 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)

Dakle, teorija dokaza nudi dvije prednosti naspram dokazivanja logičke posljedice:

- ① **Učinkovitost:** umjesto da iscrpno pretražujemo sve moguće interpretacije, deduktivnu posljedicu možemo brže dokazati (pogotovo ako koristimo pametnu strategiju dokazivanja). Primijetite, međutim, da ne možemo izbjegći neodlučivost FOL-a
- ② **Interpretabilnost:** možemo objasniti zašto nešto slijedi iz premsa (pozivajući se na pravila zaključivanja) \Rightarrow dobivamo **dokaz**

6.2. Definirati deduktivnu posljedicu i dati primjere

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 izvodjenju dedukcije govorimo o **dokazivanju teorema**

6.3. Definirati svojstva ispravnosti i potpunosti te dati primjere za oba svojstva

Ispravnost

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

Formalno, pravilo zaključivanja r je ispravno ako i samo ako

ako $F_1, \dots, F_n \vdash_r G$ onda $F_1, \dots, F_n \vdash G$

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

Potpunost

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

ako $F_1, \dots, F_n \models G$ onda $F_1, \dots, F_n \vdash_R G$

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

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

6.4. Dokazati ispravnost danog pravila zaključivanja

Ispravnost = svi T u tablici istinitosti u najdesnjem stupcu

6.5. Definirati rezolucijsko pravilo za PL i FOL te njegova svojstva

Rezolucijsko pravilo može se primijeniti samo na disjunkcije (klauzule). Skup premisa je konjunkcija klauzula (CNF).

- Metoda rezolucije (metoda razrješavanja) koristi se u propozicijskoj logici i logici prvoga reda
- Metodu je predložio J. A. 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

Rezolucijsko pravilo nad PL klauzulama

$$\frac{F_1 \vee \dots \vee \textcolor{red}{F_i} \vee \dots \vee F_n \quad G_1 \vee \dots \vee \textcolor{red}{G_j} \vee \dots \vee G_m}{F_1 \vee \dots \vee F_{i-1} \vee F_{i+1} \vee \dots \vee F_n \vee G_1 \vee \dots \vee G_{j-1} \vee G_{j+1} \vee \dots \vee G_m}$$

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

Premise nazivamo roditeljske klauzule, a dedukciju nazivamo rezolventa.

- Primjeri:

$$\begin{aligned} & A \vee B \vee \neg C, D \vee \neg B \vee E \vdash A \vee \neg C \vee D \vee E \\ & \neg A \vee B, \textcolor{red}{A} \vdash B \\ & A \vee \textcolor{red}{B}, A \vee \neg B \vdash A \vee A \\ & A \vee B, \neg A \vee \neg B \vdash B \vee \neg B \\ & A \vee \textcolor{red}{B}, \neg A \vee \neg B \vdash A \vee \neg A \end{aligned}$$

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:

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

- 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

- Vrlo slično rezoluciji u PL, uz dodatak mehanizma unifikacije
- Rezolucija opovrgavanjem funkcioniра на исти начин као у PL
- Roditeljske klauzule требају бити **standardizирани**

- Dokazali smo да је rezolucijsko правило исправно. Но је ли потпуно?
- Лако је показати да rezolucijsko правило није потпуно
- Npr., размотримо dedukciju $F \vdash F \vee G$
- Nju не можемо извести rezolucijskim pravilom (зашто?)
- Међутим, vrijedi $F \vdash F \vee G$ (provjerite!)
- Budući da vrijedi $F \vdash F \vee G$, a da rezolucijskim pravilom ne можемо deduktivno извести $F \vdash F \vee G$, закључујемо да rezolucijskim pravilom не можемо доказати све логичке посљедице, па закључујемо да **rezolucijsko правило није потпуно**

Rezolucijsko pravilo nad FOL klauzulama

$$\frac{F_1 \vee \dots \vee \textcolor{red}{F_i} \vee \dots \vee F_n \quad G_1 \vee \dots \vee \textcolor{red}{G_j} \vee \dots \vee G_m}{F_1 \delta \vee \dots \vee F_{i-1} \delta \vee F_{i+1} \delta \vee \dots \vee F_n \delta \vee G_1 \delta \vee \dots \vee G_{j-1} \delta \vee G_{j+1} \delta \vee \dots \vee G_m \delta}$$

gdje су F_i i G_j literali који се могу **komplementarno unificirati** а δ је њихов најопćenitiji unifikатор (MGU).

Rezolventa је disjunkcija свих preostalih literala roditeljskiх klauzula уз **примјену supstitucije** δ на сваки literal.

Razrješавanjem dviju jediničних klauzula izvodi се **prazna klauzula NIL**.

(1) Svi studenti pođu na predavanje.

(2) Ivan je student.

↪ Ivan pođu na predavanje.

Pretvorba u klauzalni oblik i rezolucija opovrgavanjem:

- (1) $\neg S(x) \vee P(x)$
- (2) $S(Ivan)$
- (3) $\neg P(Ivan)$ (negacija cilja)
- (4) $\neg S(Ivan)$ (из 1 и 3 уз $\delta = \{Ivan/x\}$)
- (5) NIL (из 2 и 4 уз $\delta = \emptyset$)

- Rezolucijsko правило је исправно али није потпуно (само из F се нemože izvedi FvG)

6.6. Definirati i objasniti faktorizaciju i standardizaciju te dati primjere

Faktorizacija: G v G можемо сmanjiti на G (osigurava potpunost)

Standardizacija: u FOL, ne smiju postojati 2 klauzule које sadrže исте varijable па ako se то desi moramo raditi preimenovanje (F(x) и F(x, y) moramo pretvoriti npr. у F(x'), F(x, y))

6.7. Objasniti i ilustrirati nepotpunost izravne rezolucije

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**

- Npr. ne može se izvesti $F \vee G$ samo pomoću F

6.8. Definirati rezoluciju opovrgavanjem za PL

- Umjesto da dokazujemo $F_1, \dots, F_n \vdash G$, 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 pravu klauzulu čija je semantička vrijednost \perp
- Ako rezolucijskim zaključivanjem izvedemo klauzulu NIL, onda znači da su premise proturječne (jer je rezolucijsko pravilo ispravno)

- 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 $F \vdash G$ možemo dokazati metodom opovrgavanja tako da dokazemo 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!**

- Rezolucija opovrgavanjem je potpuna i optimalna

6.9. Navesti i objasniti dvije osnovne rezolucijske strategije

- Dvije vrste **rezolucijskih strategija**:
 - strategije pojednostavljenja (engl. *simplification strategies*)
 - upravljačke strategije (engl. *control strategies*)
- **Strategije pojednostavljanja** 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 pojednostavljenja

Strategija brisanja

Uklanjanje redundantnih klauzula:

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

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 komplementarni par literalâ F_i i $\neg F_i$

Upravljačke rezolucijske strategije

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

- Temelji se na pretpostavci da je skup ulaznih premisa **konzistentan**
- Naime, kada premise ne bi bile konzistentne, iz njih bi logički slijedila bilo koja formula!
- Stoga, kako bismo kod rezolucije opovrgavanjem dokazali nekonzistentnost, moramo kombinirati klauzule ulaznih premisa s klauzulama negiranog cilja, ili s novoizvedenim klauzulama
- **Skup potpore (SoS)**: klauzule dobivene negacijom cilja i sve novoizvedene klauzule
- **Strategija skupa potpore**: **barem jedna** roditeljska klauzula uvijek dolazi iz SoS
- SoS se povećava kako izvodimo nove klauzule
- Ova je strategija potpuna i u načelu učinkovitija od strategije zasićenja (pogotovo ako je SoS malen)

6.10. Definirati najopćenitiji zajednički unifikator i dati primjere

Najopćenitiji unifikator (MGU)

- Zanimaju nas unifikatori koji daju **što je moguće općenitiju zajedničku instancu**, jer tako osiguravamo općenitost zaključka, a time i potpunost postupka zaključivanja

Najopćenitiji unifikator (engl. *most general unifier, MGU*)

Supstitucija δ je **najopćenitiji unifikator** (MGU) akko za svaki unifikator γ od K_1 i K_2 postoji supstitucija θ za koju vrijedi $\gamma = \delta \circ \theta$.

- Intuitivno, γ je manje općenit unifikator koji se iz najopćenitijeg unifikatora δ može dobiti dodatnom supstitucijom θ

Primjer

Unifikatori za $P(x)$ i $P(y)$:

- $\delta = \{y/x\}$ (MGU)
- $\gamma = \{b/x, b/y\} = \delta \circ \theta$, gdje $\theta = \{b/y\}$

6.11. Odrediti najopćenitiji zajednički unifikator dvaju izraza

- Nađi MGU izraza

$$K_1 = P(g(u), z, f(z)) \quad K_2 = P(x, y, f(b))$$

- Korak 1:

- $\{g(u)/x\}$ unificira prve podizraze od K_1 i K_2 koji se ne slažu
- $K_1\{g(u)/x\} = P(g(u), z, f(z))$
- $K_2\{g(u)/x\} = P(g(u), y, f(b))$

- Korak 2:

- $\{y/z\}$ unificira sljedeće podizraze koji se ne slažu
- kompozicija $\{g(u)/x\} \circ \{y/z\} = \{g(u)/x, y/z\}$
- $K_1\{g(u)/x, y/z\} = P(g(u), y, f(y))$
- $K_2\{g(u)/x, y/z\} = P(g(u), y, f(b))$

- Korak 3:

- $\{b/y\}$ unificira posljednje podizraze koji se ne slažu
- kompozicija $\{g(u)/x, y/z\} \circ \{b/y\} = \{g(u)/x, b/z, b/y\} = \delta$
- $K_1\delta = P(g(u), b, f(b))$
- $K_2\delta = P(g(u), b, f(b))$

Vraća grešku ako se nemože unificirati

6.12. Primjeniti pretvorbu u klauzalni oblik zadane PL ili FOL formule

6.13. Primjeniti rezoluciju opovrgavanjem na zadani skup PL ili FOL formula

6.14. Objasniti koje su posljedice neodlučivosti FOL na rezoluciju opovrgavanjem

Potpunost rezolucije opovrgavanjem i neodlučivost FOL

- Kao i kod PL, rezolucija opovrgavanjem je ispravna i potpuna
 - Ispravna:** ako iz $F \wedge \neg G$ izvede NIL, onda je G logička posljedica od F
 - Potpuna:** ako je G logička posljedica od F , onda iz $F \wedge \neg G$ izvedi NIL
- Dakle, logička posljedica i deduktivna posljedica su jedno te isto
- Potpunost je dokazao je J.A. Robinson (1965.), ali je K. Gödel već ranije dokazao postojanje takvog postupka (1929.)
- Međutim, za razliku od PL, FOL nije odlučiva!

Neodlučivost (engl. *undecidability*) problema valjanosti u FOL

Ne postoji algoritam koji za svaku formulu F daje "da" ako je F valjana ili "ne" ako F nije valjana.

- Neodlučivost u FOL dokazali su A. Church i A. Turing (1935.)
- ne postoji algoritam dokazivanja svake logičke posljedice
 - naime, prema teoremu semantičke dedukcije, $F \models G$ akko $\vdash F \rightarrow G$

Potpunost FOL

- Preciznije, FOL je **poluodlučiva**:
 - postoje algoritmi koji daju "da" ako je F valjana, ali ako F nije valjana, algoritam može nikada ne završiti
- Rezolucija opovrgavanjem jedan je takav algoritam
- Zbog poluodlučivosti, moći rezolucije opovrgavanjem je ograničena:
 - Ako G jest logička posljedica od F , postupak će uvijek izvesti NIL
 - Ako G nije logička posljedica F , postupak može nikada ne završiti

Primjer

Vrijedi

$$\forall x(\forall y P(y) \rightarrow P(x)) \not\models \forall x P(x)$$

no rezolucija opovrgavanjem to ne može dokazati (postupak ne završava).

7 Logičko programiranje u Prologu

7.1. Objasniti deklarativno programiranje i logičko programiranje

Jedan od pristupa (načina) deklarativnog programiranja je logičko programiranje
Logičko programiranje

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

Algoritam = Logika + Upravljanje

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

Deklarativno programiranje

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

7.2. Definirati Hornovu klauzulu i definitnu klauzulu te navesti primjere za svaku

Hornove klauzule (1)

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

Hornova klauzula

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

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

ili, ekvivalentno:

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

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

Definitna klauzula

Definitna klauzula je Hornova klauzula s točno jednim pozitivnim literalom.

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

- Cilj Hornove logike tj. logičkog programa koji koristi tu logiku: dokazati ciljnu klauzulu pomoću rezolucije opovrgavanjem

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

7.3. Odrediti je li zadana PL i FOL formula Hornova ili definitna klauzula

- Hornova: ima najviše 1 pozitivan literal

$$\neg P \rightarrow Q \text{ or } P \rightarrow (Q \vee R)$$

- Defintina: ima točno 1 pozitivan literal

7.4. Objasniti i ilustrirati rezoluciju s ulančavanjem unazad nad Hornovim klauzulama

- Logički program:

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

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

7.5. Razlikovati izmedu činjenica i pravila u Prologu

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

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

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

7.6. Napisati FOL formulu u Prologu i obrnuto

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

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

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

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

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

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

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

- Ili uredemo disjunkciju u tijelo pravila:

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

n-arni predikati

- Binarni predikati modeliraju binarne relacije:

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

- Pravilo za učenika inverzno je pravilu za učitelja:

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

- Pravilo koje definira da je netko podučavan od učitelja:

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

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

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

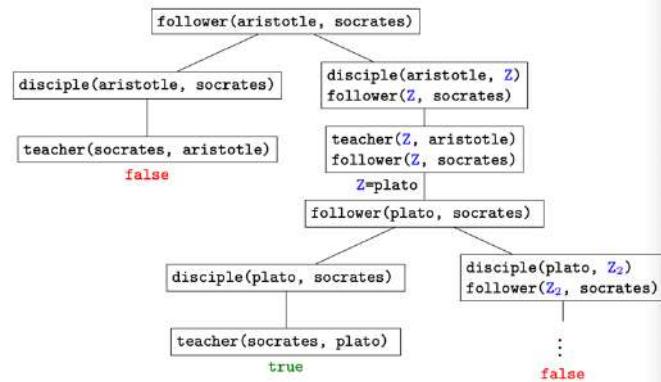
- , je i \rightarrow a i b \rightarrow c == c :- a, b.
- ; je ili \rightarrow a ili b \rightarrow c == c :- a; b.

7.7. Skicirati Prologovo stablo pretraživanja za zadani logički program i cilj

- Primjenom SLD rezolucije na logički program i negirani cilj generira se stablo pretraživanja koje je zapravo **stablo dokaza**
- Svaki čvor je **stog negativnih literala** koje treba razriješiti
- Cilj je izvesti NIL, tj. isprazniti stog
- Prolog pokušava razriješiti vršni literal stoga s **literalom glave** svake klauzule u programu (prisjetimo se: stog sadržava negativne literale, dok su glave klauzula pozitivni literali)
- Takvi literali potencijalno se mogu **komentarno unificirati**
- Klauzule se pretražuju od vrha prema dnu programa (zbog toga je poređak klauzula važan)
- Ako se vršni literal L može komplementarno unificirati pomoću MGU θ s glavom klauzule C , onda se L skida sa stoga, tijelo od C dolazi na vrh stoga, na literale stoga primjenjuje se θ , i pretraga se nastavlja
- Ako se niti jedna klauzula programa ne može komplementarno unificirati sa L , pretraga se vraća (**backtracking**) na posljednju točku odabira \Rightarrow **pretraživanje u dubinu**
- Ako ste stog isprazni, to znači da je izvedena klauzula NIL, pa procedura vraća **true**
- Ako je pretraga završena a stog nije prazan, procedura vraća **false**

```
% Pravila
follower(X, Y) :-  
    disciple(X, Y).  
  
follower(X, Y) :-  
    disciple(X, Z),  
    follower(Z, Y).  
  
disciple(X, Y) :-  
    teacher(Y, X).  
  
% Činjenice
teacher(socrates, plato).  
teacher(cratylus, plato).  
teacher(plato, aristotle).
```

Primjer stabla dokaza



- Uzima se $Z=plato$ s desne strane jer je to jedino T tj. jedino postoji činjenica $teacher(plato, aristotle)$.
- Radi se pretraživanje u dubinu i backtracking kao vraćanje na posljednju točku odabira

7.8. Napisati program u Prologu za jednostavan problem s rekurzijom i negacijom

Rekurzivno definirani predikati

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

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

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

Negacija

- Hornov oblik ne dopušta negaciju atoma u antecedentu, npr.:
$$(Q(x) \wedge \neg P(x)) \rightarrow R(x) \equiv \neg Q(x) \vee P(x) \vee R(x)$$
dva pozitivna literala!
- Logičko programiranje bez negacije bilo bi suviše ograničeno
- Prolog uvodi operator `not`, koji se može koristiti u tijelu pravila:
`R(X) :- Q(X), not(P(X)).`
- Semantika operatorka `not` je drugačija od one u logici:

Negacija kao neuspjeh (engl. negation as failure) – NAF
Literal `not(P(x))` je istinit ako se ne može dokazati `P(x)`, inače je lažan.

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

redoslijed klauzula u rekurziji je bitan

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

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

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

- `not(human(polynesia))` je true jer se nemože dokazati tj. nije definiran `human(polynesia)`.

7.9. Definirati i objasniti negaciju kao neuspjeh i prepostavku zatvorenog svijeta

Prepostavka zatvorenog svijeta

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

Prepostavka zatvorenog svijeta (closed world assumption) – CWA

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

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

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

ali u Prologu:

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

7.10. Objasniti i ilustrirati dva nedeklarativna aspekta Prologa

1. Redoslijed klauzula kod rekurzije je bitan
2. Operator `not` se ponaša drugačije nego u logici i označava da je `not(P(x))` istinit onda kada se `P(x)` ne može dokazati (npr. nije definiran) a inače je lažan.

8 Ekspertni sustavi

8.1. Objasniti razliku izmedu ekspertnih sustava i drugih simboličkih pristupa UI

- **Ljudsko rasudivanje vs. dokazivanje teorema**

- ▶ ES ne razmišljaju značajno brže niti drugačije od običnih ljudi
- ▶ Njihova ekspertiza posljedica je toga što imaju mnogo više znanja o konkretnom zadatku od običnog čovjeka

- **Znanje vs. pretraživanje**

- ▶ Rani sustavi pokušali su zaobići potrebu za znanjem primjenom učinkovitih tehnika pretraživanja (npr., igranje šaha)
- ▶ ekspertni sustavi ubrzo su pokazali da znanje može eliminirati potrebu za pretraživanjem
- ▶ Kombinacija pretraživanja i znanja: pretraživanje usmjereni znanjem (e.g., DENDRAL)



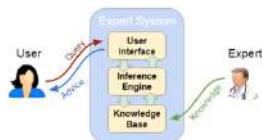
Ekspertni sustavi (sustavi temeljeni na znanju) oponašaju rasuđivanja stručnjaka u nekoj uskoj domeni.

8.2. Skicirati i objasniti arhitekturu ekspertnog sustava

Stroj za zaključivanje vs. baza znanja

Ekspertni sustavi koriste različite tehnologije za prikazivanje znanja, no svima su zajedničke dvije arhitekturne karakteristike:

- ➊ Razlikovanje između stroja za zaključivanje (engl. inference engine) i baze znanja (engl. knowledge base)
 - ▶ stroj za zaključivanje dohvaća pravila i činjenice iz baze znanja
- ➋ Uporaba deklarativnog stila prikazivanja znanja
 - ▶ pravila su podatkovne strukture s vlastitom semantikom, a ne dio programskog koda u kojem je implementirano stroj za zaključivanje

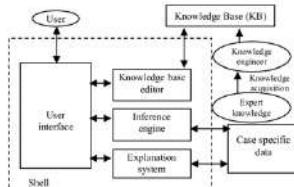


Ljuska ekspertnog sustava sačinjena je od stroja za zaključivanje i korisničkog sučelja te se može kombinirati s različitim bazama znanja.

8.3. Definirati ljusku ekspertnog sustava i navesti primjere

Ljuska ekspertnog sustava

- Stroj za zaključivanje odvojen je od baze znanja \Rightarrow različite baze mogu se koristiti kao "plug-inovi"
- **Ljuska ekspertnog sustava** (engl. expert system shell): alat za izgradnju ekspertnog sustava
 - ▶ stroj za zaključivanje
 - ▶ uređivač baze znanja
 - ▶ korisničko sučelje i modul za objašnjavanje zaključka



Primjeri ljuski ekspertnih sustava:

- CLIPS
- JESS
- PyCLIPS
- PYKE
- ES-builder

8.4. Objasniti i ilustrirati ulančavanje unaprijed i unazad te tipične primjene

- unaprijed: počinje se od zadanih podataka i pokušava se doći do zaključka poznati podatci -----> cilj
 - o malo podataka puno rješenja
- unazad: počinje se od zaključka i pokušava se dokazati valjanost hipoteza poznati podatci <----- cilj
 - o malo zaključaka puno podatak
- obostrano

poznati podatci <-----> cilj

8.5. Primijeniti ulančavanje unaprijed na zadani skup pravila i činjenica

8.6. Primijeniti ulančavanje unazad na zadani skup pravila i ciljnu hipotezu

9 Modeliranje neizvjesnosti

9.1. Izvesti Bayesovo pravilo za jednu hipotezu i jedan dokaz

$$P(X|Y) = \frac{P(Y \wedge X)}{P(Y)} \rightarrow P(Y \wedge X) = P(X|Y)P(Y)$$

$$P(Y|X) = \frac{P(X \wedge Y)}{P(X)} = \frac{P(Y \wedge X)}{P(X)} = \frac{P(X|Y)P(Y)}{P(X)}$$

$H=X$ — hipoteza

$E=Y$ — dokaz

$$P(X|E) = \frac{P(E|X) P(X)}{P(E)} \rightarrow \text{najjednostavniji oblik Bay. pravila}$$

$$P(E) = P(E|H)P(H) + P(E|\neg H)P(\neg H)$$

9.2. Definirati i objasniti Bayesovo pravilo za više hipoteza i dokaza

$$\begin{aligned} P(H_i | E_1 E_2 \dots E_n) &= \frac{P(E_1 E_2 \dots E_n | H_i) P(H_i)}{P(E_1 E_2 \dots E_n)} = \\ &= \frac{P(E_1 | H_i) P(E_2 | H_i) \dots P(E_n | H_i) P(H_i)}{\sum_{k=1}^m P(E_k | H_k) P(E_2 | H_k) \dots P(E_n | H_k) P(H_k)}, \quad i=1 \dots m \end{aligned}$$

H_i — i -ta hipoteza $H_1 \dots H_m$

E_n — n -i dokaz $E_1 \dots E_n$

9.3. Primijeniti Bayesovu shemu na probleme s više hipoteza i dokaza

9.4. Navesti prednosti i nedostatke Bayesove sheme

Prednosti

- Vrlo je dobro teoretski utemeljena,
- Najrazvijenija je od svih metoda za upravljanje i rješavanje neizvjesnosti

Nedostaci

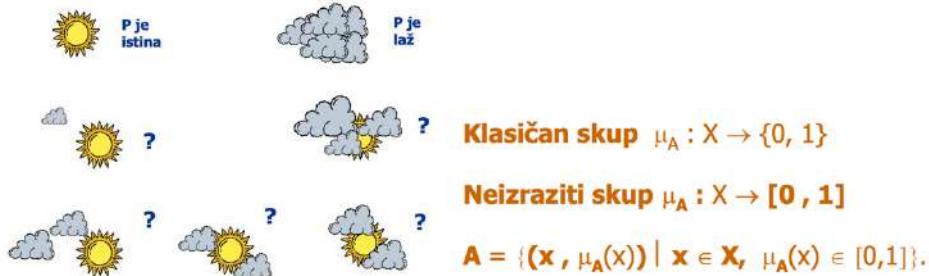
- Potrebna je velika količina podataka o vjerojatnosti da bi se izgradila baza znanja.
- Sve vjerojatnosti trebaju biti zadane !

9.5. Objasniti i ilustrirati neizraziti skup te razliku između neizrazitog i običnog skupa

- Neizrazita logika = računanje s riječima.
- Vrijedi sve kao i za klasične skupove ali ne vrijedi zakon trećega i zakon kontradikcije.
- Govorimo o mjeri koliko neki element pripada nekom skupu a ne da li tom skupu pripada ili ne pripada (ne $\{0,1\}$ nego $[0,1]$)
- Klasična logika (obični skupovi) ne može oblikovati znanje koje je nejasno, neprecizno izraženo riječima ali neizrazita logika može

Zašto nam klasična logika nije dovoljna sa stajališta primjene u intelligentnim sustavima?

p = "Danas je sunčan dan"



- Neizrazito = nema jasnih/izrazitih granica između T i F

9.6. Razlikovati izmedu pripadnosti neizrazitom skupu i vjerojatnosti

9.7. Definirati lingvističku (jezičnu) varijablu i dati primjer

- Lingvistička varijabla poprima vrijednosti u obliku riječi tj. rečenica a ne numeričkih vrijednosti

- Definicija (jezična varijabla) [Zadeh, 1975]

Jezična varijabla je petorka (x, Tx, U, G, Mx) , gdje je:
 x - naziv jezične varijable;

Tx - skup jezičnih vrijednosti (termina, izraza) koje može poprimiti jezična varijabla $Tx \rightarrow$ skup termina (*term-set*);

U univerzalni skup (*engl. universe of discourse*) je stvarna fizička domena u kojoj elementi iz T poprimaju numeričke vrijednosti. ($U \rightarrow$ kontinuiran ili diskretan);

G je gramatika tj. skup sintaktičkih pravila koji generiraju skup T iz skupa osnovnih termina;

Mx je semantička funkcija koja daje (kvantitativno) značenje (interpretaciju) jezičnim izrazima. Mx je funkcija koja $\forall x \in T$ (tj. svakoj jezičnoj vrijednosti) pridružuje neki neizraziti podskup od U

Primjer

- Jezična varijabla: STAROSNA DOB
- $x = \text{STAROSNA DOB}$
- $T(\text{STAROSNA DOB}) = \text{mlad} + \text{nije mlad} + \text{vrlo mlad} + \text{ne vrlo mlad} + \text{vrlo vrlo mlad} + \dots + \text{srednjih godina} + \text{star} + \text{nije star} + \text{vrlo star} + \text{vrlo vrlo star} + \dots + \text{ne star i ne srednjih godina} + \dots$
- $U = \{0, 1, 2, 3, 4, \dots, 99, 100\}$

9.8. Definirati, ilustrirati i primijeniti lingvističke modifikatore i operatore

Modifikatori:

- Vrlo (koncentracija) = kvadriranje
 - o $A = \text{malen}$
 - o $A^2 = \text{vrlo malen}$
- Manje ili više (dilatacija) = korjenovanje
 - o $A = \text{malen}$
 - o $\sqrt{A} = \text{korijen iz } A = \text{manje ili više malen}$

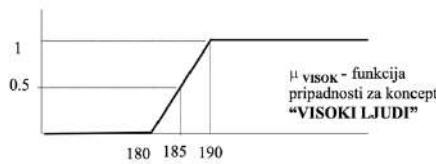
Operacija:

- Unija = max
 - o $\mu_{A \cup B}(x) = \max(\mu_A(x), \mu_B(x))$
- Presjek = min
 - o $\mu_{A \cap B}(x) = \min(\mu_A(x), \mu_B(x))$
- komplement = $1 - \mu_A$
 - o $\mu_{\sim A}(x) = 1 - \mu_A(x)$

9.9. Objasniti i ilustrirati poveznicu između neizrazitih skupova i neizrazite logike

Primjer

- Funkcija pripadnosti neizrazitog skupa i vrijednost istinitosti neke propozicije povezani su na sljedeći način:
- Istinitost propozicije "Element x pripada skupu A " ekvivalentna je stupanj pripadnosti elementa x neizrazitom skupu A tj. $\mu_A(x)$; i obrnuto:
- Stupanj pripadnosti elementa x neizrazitom skupu A ekvivalentan je istinitosti propozicije "Element x pripada skupu A "



$\rightarrow \mu_{\text{visokiLjudi}}(185) = 0.5 \rightarrow$ istinitost propozicije *Ivica je visok* je ekvivalentna stupnju pripadnosti visine od 185cm skupu visokih ljudi

9.10. Definirati neizrazitu relaciju i dati primjer

Primjer

- Neizraziti skup **A** – definiran je funkcijom pripadnosti $\mu_A(x) : X \rightarrow [0, 1]$, gdje je X univerzalni skup, a $\mu_A(x)$ broj između 0 i 1 koji određuje u kojoj mjeri element x pripada neizrazitom skupu **A**
- Neizrazita relacija **R** – definirana je funkcijom $\mu_R : X \times Y \rightarrow [0, 1]$, gdje $\mu_R(x, y)$ određuje u kojoj su mjeri u relaciji elementi x i y iz univerzalnih skupova X i Y

		W		
		1	2	3
V	1	1	0.8	0.3
	2	0.8	1	0.8
	3	0.3	0.8	1

- 1 je približno jednako 3 \rightarrow iz tablice s vrijednošću 0.3
- 2 je približno jednako 2 \rightarrow iz tablice s vrijednošću 1

- Ako je **A** neizraziti skup na X , a **B** neizraziti skup na Y , tada je **A x B neizrazita relacija** na univerzalnom skupu $X \times Y$ definirana sa $\mu_{A \times B} : X \times Y \rightarrow [0, 1]$,

$$\mu_{A \times B}(x, y) = \min(\mu_A(x), \mu_B(y))$$

Svaka implikacija predstavlja neku neizrazitu relaciju.
Implikacija "Ako x je **A** onda y je **B**" određuje
neizrazitu relaciju **A x B** na $X \times Y$.

9.11. Objasniti generalizirani modus ponens i dati primjer

Neizrazita logika = generalizirani modus ponens

- Definiramo relaciju "približno jednaki" brojevi iz implikacije

	1	2	3
1	1	0.8	0.3
2	0.8	1	0.8
3	0.3	0.8	1

A "malen broj"

1	2	3
1	0.5	0.1
2	0.8	1
3	0.3	0.8

R = "približno jednaki brojevi"

1	2	3
1	0.8	0.3
2	0.3	0.8
3	1	1

- Pravilo zaključivanja za generalizirani modus ponens (tzv. **Zadehovo pravilo min-max kompozicije**) kaže da je tada kompozicija A o R jednaka neizrazitom skupu B iz zaključka modus ponensa, gdje je neizraziti skup B definiran sa funkcijom pripadnosti
- $\mu_B(w) = \max(\min(\mu_A(v), \mu_R(v,w)))$

$$\mu_B(1) = \max(\min(\mu_A(v), \mu_R(v,1))) = \\ \max\{\min(1,1), \min(0.5,0.8), \min(0.1,0.3)\} = \\ \max\{1, 0.5, 0.1\} = 1$$

$$\mu_B(2) = \max(\min(\mu_A(v), \mu_R(v,2))) = \\ \max\{\min(1,0.8), \min(0.5, 1), \min(0.1,0.8)\} = \\ \max\{0.8, 0.5, 0.1\} = 0.8$$

9.12. Primjeniti generalizirani modus ponens na zadani problem

10 Strojno učenje

10.1. Objasniti osnovne koncepte SU (model, treniranje, predviđanje, generalizacija)

Na temelju viđenih podataka (vidi ih kroz treniranje) model može predvidjeti svojstva (tj. generalizirati) novih, još neviđenih, podataka.

10.2. Razlikovati tri osnovna pristupa strojnog učenju i navesti primjere za svaki

1. Nadzirano

- Pomoću ulaza i očekivanog izlaza radi se preslikavanje
- Klasifikacija (skupina) i regresija (točna vrijednost)
- Npr. Predviđanje temperature sutra (regresija), predviđanje u kakvo će biti vrijeme sutra (klasifikacija: toplo (15-20 stupnjeva), hladno (5-10 stupnjeva), ...)

2. Nenadzirano

- Pomoću ulaza bez izlaza pokušavaju se naći pravilnosti
- Grupiranje, otkrivanje novih vrijednosti, smanjenje dimenzionalnosti
- Npr. Grupiranje korisnika, clustering DNA, ...

3. Podržano

- Pokušava se naći optimalna strategija uz odgođenu nagradu
- Npr. igre šah, Go

10.3. Navesti barem pet tipičnih primjena strojnog učenja

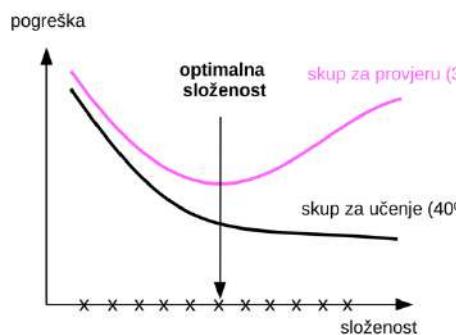
1. Raspoznavanje rukom pisanih znamenki
2. Prepoznavanje uzoraka
3. Analiza ponašanja korisnika na temelju prethodnih akcija (koje reklame mu ponudit)
4. Autonomno vozilo (prepoznavanje pješaka, automatsko parkiranje)
5. Prepoznavanje prometnih znakova

10.4. Objasniti prenaučenost i unakrsnu provjeru

Prenapučenost: model je presložen pa se previše prilagodio viđenim podatcima a daje loše predikcije za neviđene podatke (loše generalizira)

Unakrsna provjera: provjera koju provodimo kako bi vidjeli koliko će dobro model raditi na neviđenim podatcima

- Podjela seta na train/val/test ili samo train/test
- Računamo točnost na skupu za val ako je podjela na 3 a na test ako je podjela na 2 skupa



10.5. Definirati i objasniti naivan Bayesov klasifikator i prepostavke na kojima se temelji

Naivan Bayesov klasifikator: algoritam nadziranog SU temeljen na Bayesovom pravilu:

$$P(H_i|E_1, E_2, \dots, E_n) = \frac{P(E_1|H_i)P(E_2|H_i) \cdots P(E_n|H_i)P(H_i)}{\sum_j P(E_1|H_j)P(E_2|H_j) \cdots P(E_n|H_j)P(H_j)}$$

- Bayesovo pravilo izravno se može upotrijebiti kao klasifikacijski model
 - ▶ Hipoteza $H \rightarrow$ **klasa y**
 - ▶ Dokazi $E_1, \dots, E_n \rightarrow$ **primjer $x = (x_1, \dots, x_n)$**
- Model:

$$P(y|x) = \frac{P(x|y)P(y)}{P(x)} = \frac{P(x|y)P(y)}{\sum_{y'} P(x|y')P(y')}$$

gdje

 - ▶ $P(y)$ je **apriorna vjerojatnost klase** (engl. *prior*)
 - ▶ $P(x|y)$ je **izglednost klase** (engl. *class likelihood*)
 - ▶ $P(y|x)$ je **aposteriorna vjerojatnost klase** (engl. *posterior*)
- Nas zanima aposteriorna vjerojatnost $P(y|x)$: **koja je vjerojatnost da primjer x pripada klasi y** (odnosno da ima dotičnu oznaku)
- Primijetite da je to suprotno od izglednosti klase $P(x|y)$: koja je vjerojatnost da klasa y ima primjer x
- Isto kao i u slučaju više dokaza, uvodimo **prepostavku uvjetne nezavisnosti**: značajke x_j su uvjetno nezavisne za danu klasu y , tj. vrijedi $P(x_j, x_k|y) = P(x_j|y)P(x_k|y)$
- Uz tu prepostavku, model Bayesovog klasifikatora je:

$$\begin{aligned} h_{\text{MAP}} &= \operatorname{argmax}_y P(y|x) = \\ &= \operatorname{argmax}_y P(x|y)P(y) \stackrel{u.n.}{=} \operatorname{argmax}_y P(y) \prod_{j=1}^n P(x_j|y) \end{aligned}$$
 - Ovaj model nazivamo **naivan Bayesov klasifikator** (engl. *naïve Bayes classifier*)
 - Model je "naivan" jer prepostavka o uvjetnoj nezavisnosti značajki općenito ne vrijedi (između značajki općenito postoje interakcije)
 - Međutim, u praksi se pokazuje da model unatoč tome radi solidno
 - Naivan jer uvjetna nezavisnost u stvarnom svijetu ne vrijedi a tu ju prepostavljamo

10.6. Definirati i objasniti procjenu najveće izglednosti i Laplaceovo zaglađivanje

Procjena najveće izglednosti (MLE)

Procjene najveće izglednosti za apriorne vjerojatnosti klasa i izglednosti klasa Bayesovog klasifikatora jesu:

$$P(y = v) = \frac{|D_{y=v}|}{|D|}$$
$$P(x_j = w|y = v) = \frac{|D_{y=v} \wedge x_j=w|}{|D_{y=v}|}$$

gdje je D skup primjera, a D_P je podskup primjera koji zadovoljavaju P

- Podložno prenapučenosti (neviđeni primjer odmah klasificira s 0) → treba napraviti zaglađivanje:

Procjene izglednosti klase s Laplaceovim zaglađivanjem

$$P(x_j = w|y = v) = \frac{|D_{y=v} \wedge x_j=w| + \alpha}{|D_{y=v}| + \alpha|V(x_j)|}$$

gdje je $V(x_j)$ skup mogućih vrijednosti značajke (atributa) x_j , a $\alpha \geq 0$ je parametar zaglađivanja

- Odabir $\alpha = 0$ daje nezaglađenu procjenu, a $\alpha = 1$ daje zaglađivanje "dodaj jedan" (engl. *add-one smoothing*)

10.7. Primjeniti Bayesov klasifikator na zadani skup podataka (treniranje i predikcija)

10.8. Navesti prednosti i nedostatke naivnog Bayesovog klasifikatora

• Prednosti:

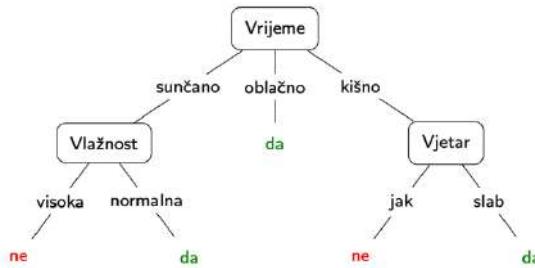
- ▶ jednostavan i brz algoritam (lako je trenirati model)
- ▶ može raditi s viševrijednosnim značajkama i sa više klasa odjednom
- ▶ radi dobro s velikim brojem značajki (linearan je u broju značajki)
- ▶ daje dobre predikcije ako vrijede uvjetne nezavisnosti značajki

• Nedostatci:

- ▶ u stvarnosti, značajke su rijetko uvjetno nezavisne i točnost klasifikacije je to lošija što su značajke više zavisne
- Tipične primjene: klasifikacija dokumenata, filtriranje neželjene pošte
- Ako prepostavke o uvjetnoj nezavisnosti ne vrijede, može se koristiti **polunaivan Bayesov klasifikator** (složeniji model)
- Ako su značajke kontinuirane, za izglednosti klasa koristimo Gaussove distribucije (**Gaussov Bayesov klasifikator**)
- Bayesov klasifikator pripada širokoj porodici modela koje nazivamo **probabilistički grafički modeli**

10.9. Objasniti i dati primjer klasifikacije pomoću stabla odluke

- Stablo odluke za skup primjera "Dan za odbojku na pijesku":



- Klasifikacija novog primjera:

$$x = (Vrijeme=sunčano, Temp=niska, Vlažnost=visoka, Vjetar=jak)$$

\Rightarrow ne

Čvor: atribut

Grana: značajka atributa

List: klasifikacija (odluka, oznaka klase)

Jedna staza od korijena do lista = jedno pravilo

Praksa:

- Graditi što jednostavnije stablo jer bolje generalizira
- Kriterij za odabir najpovoljnije značajke = IG, E

10.10. Definirati i objasniti kriterij informacijske dobiti

- Kriterij **informacijske dobiti** (engl. *information gain*) mjeri očekivano smanjenje entropije skupa primjera uslijed podjele primjera po vrijednostima neke značajke

Informacijska dobit

Informacijska dobit (IG) značajke x na skupu primjera D je:

$$IG(D, x) = \underbrace{E(D)}_{\substack{\text{entropija} \\ \text{početnog skupa}}} - \underbrace{\sum_{v \in V(x)} \frac{|D_{x=v}|}{|D|} E(D_{x=v})}_{\substack{\text{očekivana vrijednost entropije} \\ \text{nakon podjele skupa na temelju značajke}}}$$

gdje je $E(D)$ entropija skupa primjera D , D_P je podskup primjera koji zadovoljavaju uvjet P , a $V(x)$ je skup mogućih vrijednosti značajke x .

- Želimo manji E tj. veći IG
- Biramo značajku s najvećim IG kao sljedeći čvor

10.11. Definirati algoritam ID3

1. Za svaki atribut računaj E
2. Za svaki atribut računaj E svih značajki
3. Računaj IG za svaki atribut
4. Uzmi atribut s najvećim IG kao čvor i radi to rekursivno dok god ima neobiđenih atributa

10.12. Primijeniti algoritam ID3 na zadani skup podataka (treniranje i predikcija)

10.13. Objasniti prenaučenost stabla odluke i pristupe da se ona sprječi

Prenaučenost: dobro klasificira viđene a loše neviđene primjere.

Mogući uzrok su pogrešne vrijednosti značajki ili pogrešnu oznaku klase.

- Da pristupa za sprječavanje prenaučenosti:
 - ① **Ograničavanje rasta stabla** prije dosezanja savršene klasifikacije na skupu primjera za učenje
 - ② **Naknadno podrezivanje prenaučenog stabla**
 - * zamjena postabala listovima
 - * pretvorba stabla u ako-onda pravila te uklanjanje uvjeta u antecedentima pravila
- **Q:** Kako odrediti optimalnu veličinu stabla?
- **Intrinzični kriteriji** za zaustavljanje rasta stabla:
 - ▶ dosezanje unaprijed definirane maksimalne dubinu stabla
 - ▶ broj primjera u nekom čvoru je manji od unaprijed zadanoг broja
 - ▶ pad entropije je manji od manji od unaprijed definiranog praga
- **Ekstrinzični kriterij** za zaustavljanje rasta ili podrezivanje stabla:
 - ▶ pad točnosti (porast pogreške) na **skupu primjera za provjeru**

10.14. Navesti prednosti i nedostatke stabla odluke

Prednosti:

- Mogu se koristit i za klasifikaciju i za regresiju
- Lako ih je interpretirati

Nedostatci

- Ne daju vjerojatnosti klasifikacijske odluke
- Lako se prenauče
- Potpuno različita stabla za male razlike u skupu za učenje

11 Umjetne neuronske mreže

11.1. Razlikovati između simboličke i konekcionističke UI

- Od prvih dana razvoja umjetne inteligencije (rane '50) postoje dva pristupa razvoju inteligentnih sustava:
 - Simbolički pristup:** znanje iz neke domene nastoji se obuhvatiti skupom atomičkih semantičkih objekata (simbola) i zatim manipulirati tim simbolima pomoću algoritamskih pravila
 - Konektivistički pristup:** temelji se na izgradnji sustava arhitekture slične arhitekturi mozga koji, umjesto da ga se programira, **uči samostalno na temelju iskustva**
- Simbolički pristup je dobar u mnogim područjima (osobito isplativ postao je razvojem ekspertnih sustava), ali nije ispunio rana ekstravagantna obećanja.
- Neuspjeh leži u pogrešnoj prepostavci da je svako znanje moguće formalizirati i da je možak stroj koji podatke obrađuje formalnim pravilima.



11.2. Objasniti osnovne koncepte UNM (UNM, unaprijedna i unaprijedna slojevita UNM)

U širem smislu: umjetna replika ljudskog mozga kojom se nastoji simulirati postupak učenja i obrade podataka.

Umjetna neuronska mreža

Umjetna neuronska mreža je skup međusobno povezanih jednostavnih procesnih elemenata (**neurona**) čija se funkcionalnost temelji na bioškom neuronu i koji služe distribuiranoj paralelnoj obradi podataka.

- Omogućavaju robusnu obradu podataka.
- Može ih se koristiti za probleme **klasifikacije** te za probleme **regresije**.
- Sposobne su **učiti** iz podataka.

Dvije faze rada s ANN:

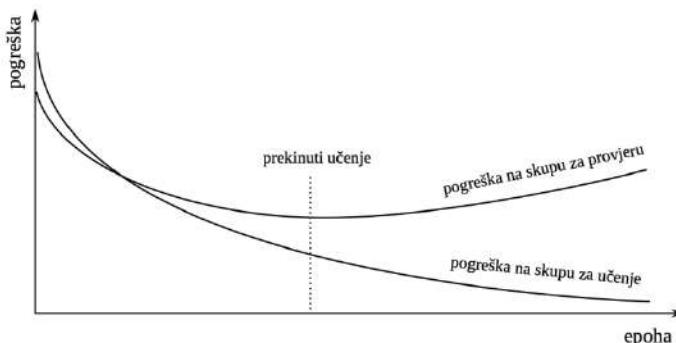
- Faza učenja (treniranja) i
 - Faza obrade podataka (iskorištavanja, eksploracije).
- Učenje je iterativan postupak predočavanja ulaznih primjera (uzoraka, iskustva) i eventualno očekivanog izlaza pri čemu dolazi do postupnog prilagođavanja težina veza između neurona.
- Jedno predočavanje svih uzoraka naziva se **epochom**.
- Razlikujemo:
- Pojedinačno učenje** (engl. *on-line*): učenje se događa nakon svakog predočenog uzorka
 - Učenje s minigrupama** (engl. *mini-batches*): učenje se događa nakon više predočenih uzoraka
 - Grupno učenje** (engl. *batch*): učenje se događa tek nakon svih predočenih uzoraka
- Znanje o izlazu kao funkciji ulaza pohranjeno je implicitno u težinama veza neurona



Vrste učenja: nadzirano, nenađzirano, podržano.

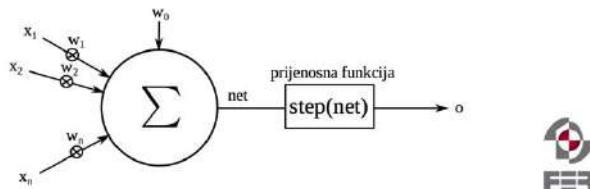
Uči se na skupu za učenje a točnost se prati na skupu za provjeru.

Podložno pretreniranosti.



11.3. Skicirati, objasniti i formalno definirati McCulloch-Pittsov model umjetnog neurona

- McCulloch-Pitts definiraju jednostavan model biološkog neurona (1943.): **TLU-perceptron** (engl. *Threshold Logic Unit*)
 - Vrijednost sa svakog ulaza x_i množi se osjetljivošću tog ulaza w_i i akumulira u tijelu.
 - Ukupnoj sumi dodaje se pomak w_0 (još se označava i b od engl. *bias*). Time je definirana akumulirana vrijednost $net = \left(\sum_{i=1}^n x_i \cdot w_i \right) + w_0$.
 - Ta se vrijednost propušta kroz prijenosnu funkciju čime nastaje izlazna vrijednost: $o = step(net)$.



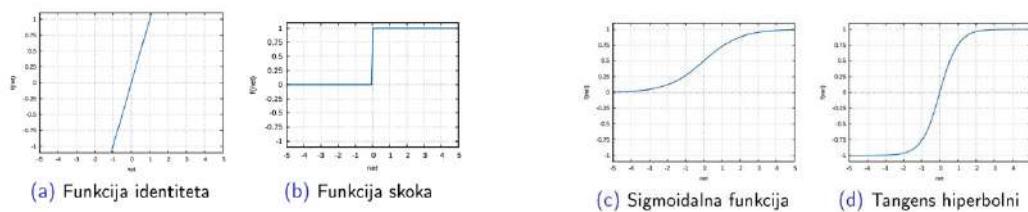
X – vrijednost ulaza

W – težine

Net – suma $x_i \cdot w_i + w_0$

O – izlaz koji dobivamo propuštanjem net-a kroz prijenosnu funkciju step

11.4. Navesti i definirati različite prijenosne funkcije



- Identitet

$$f(net) = net$$

- Funkcija skoka

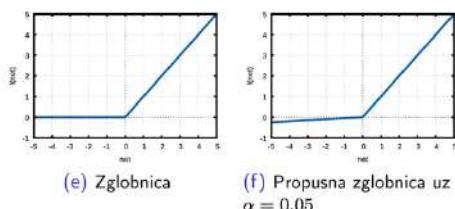
$$f(net) = step(net) = \begin{cases} 0 & net < 0 \\ 1 & \text{inače} \end{cases}$$

- Sigmoidalna funkcija

$$f(net) = sigm(net) = \frac{1}{1+e^{-net}}$$

- Tangens hiperbolni

$$f(net) = \tanh(net) = \frac{2}{1+e^{-2 \cdot net}} - 1 = 2 \cdot sigm(2x) - 1$$



- Zglobnica

$$f(net) = \max(0, net)$$

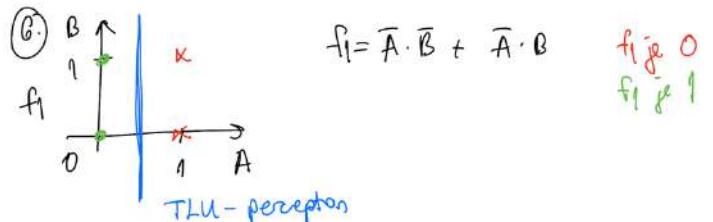
- Propusna zglobnica

$$f(net) = \begin{cases} net & net > 0 \\ \alpha \cdot net & \text{inače} \end{cases}$$

e) je ReLU

11.5. Objasniti TLU-perceptron

- Slika ista kao za McCulloch-Pittsov model neurona
- Ima linearu decizijsku granicu pa možemo riješiti problem samo linearno razdvojivih primjera



11.6. Definirati i objasniti Rosenblattov algoritam učenja perceptrona

- 1949. godine Hebb dolazi do spoznaje na temelju proučavanja bioloških neurona: *učiti zanči mijenjati jakosti veza*
- 1958. godine Rosenblatt spaja Hebbovu ideju i McCulloch-Pitts model te definira *pravilo učenja perceptrona*

Pravilo učenja perceptrona

- ➊ Ciklički prolazi kroz svih N uzoraka za učenje, jedan po jedan.
- ➋ Klasificiraj trenutni uzorak.
 - ➌ Ako se klasificira korektno, ne mijenjaj težine i
 - ➍ ako je to N -ti uzastopni uzorak klasificiran korektno, prekini učenje,
 - ➎ inače priđi na sljedeći uzorak.
 - ➏ Ako se ne klasificira korektno, korigiraj težine perceptrona prema sljedećem izrazu:
 $w_i(k+1) \leftarrow w_i(k) + \eta \cdot (t - o) \cdot x_i$

- Kako radi: postupak zadatka s tim (računanje net-a i outputa (o), računanje faktora korekcije, korekcija težina ako treba)

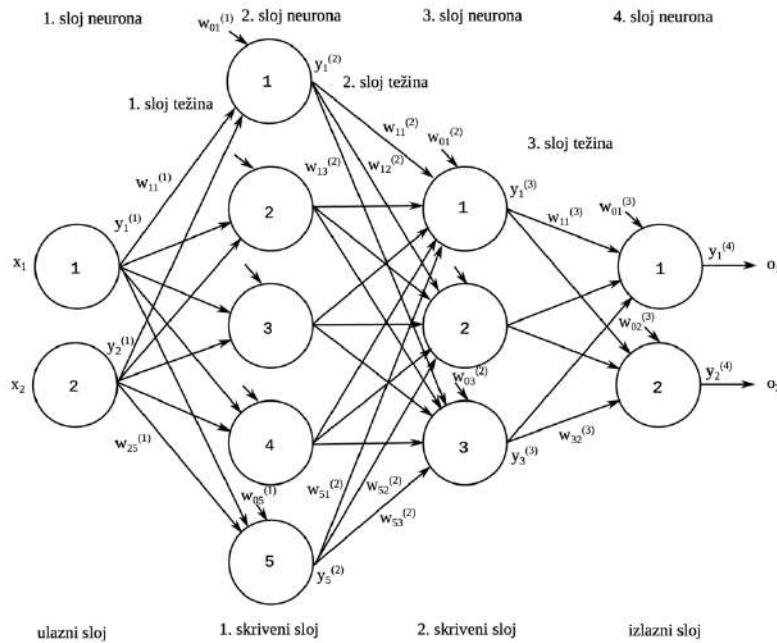
11.7. Primjeniti Rosenblattov algoritam učenja perceptrona na dani skup podataka

- Tablica: epoha (x1, x2, y) net faktor o korekcija-da/ne?

11.8. Objasniti potrebu za nelinearnim prijenosnim funkcijama

Da bi mogli rješavati nelinearne probleme (povećanje ekspresivnosti)

11.9. Objasniti notaciju za strukturu unaprijedne slojevite neuronske mreže



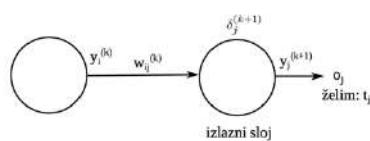
Slika: Unaprijedna slojevita potpuno-povezana neuronska mreža

11.10. Objasniti problem dodjele odgovornosti i kako algoritam BP rješava taj problem

?

11.11. Definirati izraze za pogreške i ažuriranje težina algoritma BP

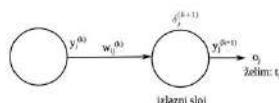
Izračun pogreške u neuronu izlaznog sloja.



Pogreška: umnožak derivacije prijenosne funkcije neurona ($y_j^{(k+1)} \cdot (1 - y_j^{(k+1)})$) i stvarne pogreške ($t_j - o_j$).

$$\delta_j^{k+1} = o_{s,j} \cdot (1 - o_{s,j}) \cdot (t_{s,j} - o_{s,j}).$$

Korekcija: proporcionalno umnošku stope učenja η , izlaza neurona lijevo od težine i pogreške neurona desno od težine:



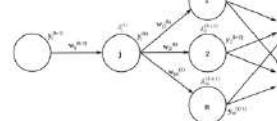
$$\Delta w_{ij}^k = \eta \cdot y_i^{(k)} \cdot \delta_j^{(k+1)}$$

$$w_{ij}^k \leftarrow w_{ij}^k + \Delta w_{ij}^k$$

Za pravove "lijevo" je konstanta 1, pa imamo:

$$\Delta w_{0j}^k = \eta \cdot \delta_j^{(k+1)}$$

Izračun pogreške u neuronu skrivenog sloja: $\delta_j^{(k)}$.



Pogreška: umnožak derivacije prijenosne funkcije promatranoj neurona i težinske sume pogrešaka neurona kojima on šalje svoj izlaz:

$$\delta_j^{(k)} = y_j^{(k)} \cdot (1 - y_j^{(k)}) \cdot (w_{j1}^{(k)} \cdot \delta_1^{(k+1)} + \dots + w_{jm}^{(k)} \cdot \delta_m^{(k+1)}).$$

11.12. Primjeniti algoritam BP na danu UMN i jedan primjer za učenje

12 Prirodom inspirirani optimizacijski algoritmi

12.1. Objasniti koncept izranjajuće inteligencije

- Izranjajuća inteligencija: neki sustavi sastavljeni od jednostavnih elemenata iskazuju inteligentno ponašanje
- Npr. jata ptica, riba, kolonije mrava, rojevi pčela...

12.2. Definirati optimizacijski problem i problem zadovoljavanja ograničenja

- Optimizacijski problem: problem pronalaženja najboljeg rješenja problema (najjeftinijeg, najkraćeg, ...)
- Problem zadovoljavanja ograničenja: put od početnog do konačnog stanja nije bitan nego je rješenje isključivo konačno stanje

12.3. Objasniti koncepte heuristike i metaheuristike te dati primjere

Heuristika: algoritam koji pronađe dovoljno dobra rješenja, nemaju garantiju optimalnosti ali imaju nisku računsku složenost.

Npr. ?

Metaheuristika: skup algoritamskih koncepta koji koristimo za definiranje heurističkih metoda primjenjivih na širok skup problema

Npr. Evolucijski algoritam, mravlji algoritam, ...

12.4. Objasniti teorem No-Free-Lunch i njegove posljedice na optimizacijske algoritme

• Teorem "No free lunch", Wolpert & Macready, 1995, 1997:

– Svi algoritmi koji traže ekstrem funkcije cilja ponašaju se upravo jednako s obzirom na bilo koju mjeru performansi, kada se pogleda njihovo prosječno ponašanje nad svim mogućim funkcijama cilja.

• Teorem "No free lunch", Wolpert & Macready, 1995, 1997:

– Konkretno, ako algoritam A nadmašuje algoritam B na nekim funkcijama cilja, tada, grubo govoreći, mora postojati upravo toliko drugih funkcija cilja nad kojima B nadmašuje A.

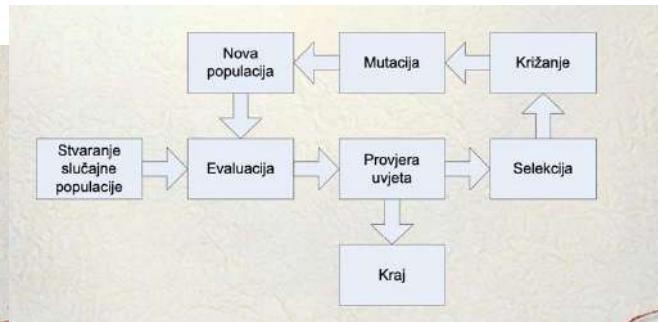
The theorem states that all optimization algorithms perform equally well when their performance is averaged across all possible problems.

There is provably no single best optimization algorithm or machine learning algorithm.

12.5. Objasniti genetički algoritam

• Kako radi GA?

- Radimo s populacijom kromosoma
- Svaki kromosom je jedno rješenje problema
- Svako rješenje ima svoju dobrotu (engl. fitness) ili kaznu
- U našem primjeru dobrota i $f(x)$ su suprotni
→ veći $f(x)$, manja dobrota
→ $f(x)$ odgovara kazni

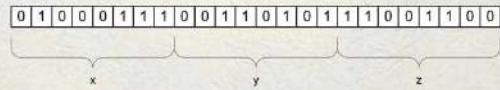


12.6. Objasniti binarno kodiranje kromosoma i dati primjere

- Binarni kromosom 
 - Niz nula i jedinica koji se interpretira kao rješenje problema (vrijednost varijable)
 - Npr. Trobitni kromosom: 000, 001, ..., 111
 - Ako predstavlja realnu varijablu iz područja [-2, 2], tada:
000= -2, 001= -1.43, ..., 111= 2
 - Broj bitova za određenu preciznost?

- Binarni kromosom

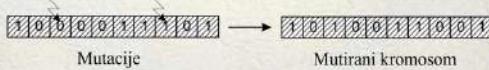
- Složeniji primjer
 - rješenje za funkciju od tri varijable x, y, z



12.7. Objasniti i ilustrirati genetičke operatore (selekcija, križanje, mutacija) i elitizam

- Uloga
 - Selekcija** → selekcijski pritisak → brzina konvergencije
 - Križanje** → pretraživanje okoline roditelja
 - Mutacija** → bijeg iz lokalnih ekstremi, veliki skokovi u prostoru pretraživanja

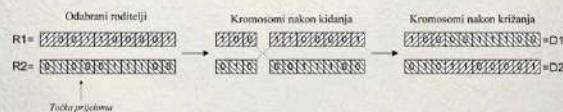
- Operator mutacije
 - Zadana vjerojatnost mutacije bita
 - Operator okreće vrijednost bita



- Može generirati veliku promjenu!

- Križanje s jednom točkom prijeloma

- Odabiru se dva roditelja
- Slučajno se odabire točka prijeloma
- Obavlja se križanje



- Izbor roditelja – više tehnika

- Proporcionalna selekcija (engl. Roulette-wheel selection)
 - Što je jedinka bolja, to ima veću šansu biti izabrana

$$probSel(i) = \frac{fit(i)}{\sum_{j=1}^n fit(j)}$$

Elitizam: Elitizam je svojstvo algoritma da ne može izgubiti najbolje pronađeno rješenje.

- Ubrzava konvergenciju algoritma

12.8. Primjeniti genetičke operatore na danu reprezentaciju rješenja

12.9. Objasniti način rada algoritma kolonije mrava (ACO)

Pseudokod 3.2.1 — Algoritam Ant System.

```
ponavljaj dok nije kraj
    ponovi za svakog mrava
        stvori rješenje
        vrednuj rješenje
    kraj ponovi
    ispari feromonske tragove
    ponovi za sve mrave
        ažuriraj feromonske tragove
    kraj ponovi
kraj ponavljanja
```

- Stvaranje rješenja:
 - o Pronalazak hamiltonovog ciklusa kroz graf (to radi svaki mrav zasebno, a razlikuju im se količine feromonskih tragova (tau))
 - **Algoritam Ant System**
 - Uporaba heurističke informacije dodatno poboljšava ponašanje
- $$\tau_0 = \frac{m}{C^{mn}}$$

$$p_j^k = \begin{cases} \frac{\tau_j^\alpha \cdot \eta_{ij}^\beta}{\sum_{i \in N_i^k} (\tau_i^\alpha \cdot \eta_{il}^\beta)}, & \text{ako } j \in N_i^k \\ 0, & \text{ako } j \notin N_i^k \end{cases}$$
- o Mrav kreće iz početnog čvora i na temelju vrijednosti p_{ij} bira sljedeći čvor sve dok ne dođe do zadnjeg čvora (dok svi čvorovi nisu obiđeni točno 1)
 - o C na nn je najkraća duljina puta pronađena nekim jednostavnim algoritmom
 - o m je broj mrava
 - o tau – jakost feromonskog traga
 - o ni – vrijednost heuristike
 - o Uz premali τ_0 pretraga će brzo biti usmjerena prema području koje su mravi slučajno odabrali u prvoj iteraciji, a uz preveliki τ_0 količine feromona koje mravi ostavljaju u svakoj iteraciji bit će premale da bi mogle usmjeravati pretragu, pa će se morati potrošiti puno iteracija kako bi mehanizam isparavanja uklonio višak feromona, i time omogućio da zapčne stvarna pretraga.
 - Vrednovanje rješenja:
 - o Računanje ukupne duljine puta koju je svaki mrav prešao
 - Isparavanje feromonskih tragova:
 - **Procedura: Ispari tragove**
 - Funkcija feromonske tragove na svim bridovima umanji za određeni iznos
- $$\tau_{ij} \leftarrow \tau_{ij} \cdot (1 - \rho)$$

- Geometrijska progresija!
 - Izuzetno skupo (graf ima puno bridova)
- Ažuriraj tragove:
 - **Procedura: Ažuriraj tragove**
 - Funkcija za odabranog mrava dodaje nove feromonske tragove iznosa:
- $$\Delta \tau_{ij}^k = \begin{cases} 1/C^k, & \text{ako je brid } i-j \text{ na stazi } k \text{ - tog mrava} \\ 0, & \text{inačn} \end{cases}$$

- Novo stanje je tada:
- $$\tau_{ij} \leftarrow \tau_{ij} + \sum_{k=1}^m \Delta \tau_{ij}^k$$

12.10. Objasniti kako različiti parametri algoritma ACO utječu na ponašanje algoritma

- Uz premali τ_0 pretraga će brzo biti usmjerena prema području koje su mravi slučajno odabrali u prvoj iteraciji, a uz preveliki τ_0 količine feromona koje mravi ostavljaju u svakoj iteraciji bit će premale da bi mogle usmjeravati pretragu, pa će se morati potrošiti puno iteracija kako bi mehanizam isparavanja uklonio višak feromona, i time omogućio da zapčne stvarna pretraga.
- Ako je $\alpha = 0$, utjecaj feromonskog traga se poništava, i pretraživanje se vodi samo heurističkom informacijom. Ako je pak $\beta = 0$, utjecaj heurističke informacije se poništava, i ostaje utjecaj isključivo feromonskog traga, što često dovodi do prebrze konvergencije suboptimalnom rješenju (gdje mravi slijede jedan drugoga po relativno lošoj stazi). Dobri rezultati postižu se uz $\alpha \approx 1$ i β između 2 i 5.

12.11. Razlikovati izmedu algoritama ACO i Mravlji sustav?

12.12. Primijeniti jedan korak algoritma ACO/Mravlji sustav na dani problem

13 Podržano učenje

13.1. Razlikovati između nadziranog, nenadziranog i podržanog učenja

- Već prije napisano

13.2. Opisati model problema nad kojim se može primijeniti podržano učenje

- Ljudi ne uče tako! Primjerice, malo dijete koje uči hodati brzo počinje trčati - nagrada je ushićenje. No brzo nakon toga, slijedi pad i bol, pa dijete nauči biti opreznije. Učenje se odvija na temelju dobivene nagrade tijekom akcije ili s nekon odgodom.
- *Podržano učenje* oponaša ovakav način učenja.



13.3. Nавести vrste okolina, vrste politika te vrste problema

Okolina je mjesto u kojem agent poduzima akciju te ta akcija mijenja stanje okoline a agent prima nagradu.

- Vrste: deterministička i stohastička

Okolina u kojoj agent djeluje može biti deterministička ili pak stohastička.

- Kada agent poduzme neku akciju, okolina će dojaviti kako je ta akcija promjenila stanje okoline te koliku nagradu agent prima
- Oboje mogu biti stohastički procesi - moguće je da iz istog stanja s istom akcijom agenta okolina ne prelazi uvijek u isto sljedeće stanje, kao i da svaki puta daje neku drugačiju nagradu.

Razmatramo samo okoline koje zadovoljavaju svojstvo Markovljevog procesa odlučivanja, tj. kod kojih je uvjetna vjerojatnost prelaska okoline u sljedeće stanje određiva samo na temelju podataka dostupnih u trenutnom stanju:

$$p(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, s_{t-2}, a_{t-2}, \dots, s_0, a_0) = p(s_{t+1}|s_t, a_t)$$



Politika

Način na koji agent odabire akciju naziva se **politika** (engl. *policy*) i označava slovom π .

- Politika agenta je funkcija koja preslikava stanje s u odabranu akciju a . Jedan od mogućih zadataka podržanog učenja jest naučiti optimalnu politiku za problem s kojim je agent suočen.

Zadaća agenta je birati akcije na način koji maksimizira sumu primljenih nagrada, odnosno:

$$R_t = r_{t+1} + \gamma \cdot r_{t+2} + \gamma^2 \cdot r_{t+3} + \gamma^3 \cdot r_{t+4} + \dots + \gamma^{T-t-1} \cdot r_T = \sum_{k=0}^{T-t-1} \gamma^k \cdot r_{t+k+1} \quad (1)$$

pri čemu je γ faktor kojim se balansira između želje da se maksimizira isključivo trenutna nagrada poteza (za $\gamma = 0$) odnosno suma svih dugoročno primljenih nagrada (uz $\gamma = 1$).



Problem

Vrijedi:

$$\begin{aligned} R_t &= r_{t+1} + \gamma \cdot r_{t+2} + \gamma^2 \cdot r_{t+3} + \gamma^3 \cdot r_{t+4} + \dots + \gamma^{T-t-1} \cdot r_T \\ &= r_{t+1} + \gamma \cdot (r_{t+2} + \gamma^1 \cdot r_{t+3} + \gamma^2 \cdot r_{t+4} + \dots + \gamma^{T-t-2} \cdot r_T) \\ &= r_{t+1} + \gamma \cdot R_{t+1} \end{aligned}$$

Problem koji agent rješava može biti:

- *epizodički* - ako je gotov u konačnom broju koraka, što podrazumijeva da u okolini mora postojati terminalno stanje
- *kontinuirani* - ako agent akcije može obavljati do u beskonačnost

13.4. Definirati funkciju vrijednosti i funkciju vrijednosti akcije pod zadanim politikom

Funkcije vrijednosti i vrijednosti akcije

Funkcija vrijednosti pod politikom π :

$$v_\pi(s) = \mathbb{E}_\pi[R_t | s_t = s] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{k+t+1} | s_t = s \right] \quad (2)$$

Funkcija vrijednosti akcije pod politikom π :

$$q_\pi(s, a) = \mathbb{E}_\pi[R_t | s_t = s, a_t = a] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{k+t+1} | s_t = s, a_t = a \right] \quad (3)$$

Vrijedi:

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s'} p(s'|s, a) [r(s, a, s') + \gamma \cdot v_\pi(s')] \quad (4)$$

odnosno ako uočimo da unutarna suma zapravo predstavlja q-vrijednosti:

$$v_\pi(s) = \sum_a \pi(a|s) \cdot q_\pi(s, a) \quad (5)$$



13.5. Objasniti algoritam vrednovanja politike i provesti ga nad zadanim problemom

Ako na raspolaganju imamo model politike i model okoline funkciju vrijednosti pod politikom možemo odrediti puno učinkovitije postupkom **vrednovanja politike** (engl. *policy evaluation*).

Postupak je iterativni.

- ① Inicijaliziraj vrijednosti na početne (primjerice 0) u polju v . Terminalna polja moraju biti 0.
- ② Ponavljaj dok je ukupna promjena veća od nekog praga
 - ① Koristeći polje v i izraz (4) nanovo izračunaj vrijednost za svako stanje i rezultat pohrani u v'
 - ② $v \leftarrow v'$

Postupak dokazano konvergira i učinkovit je.

13.6. Definirati Bellmanove jednadžbe te optimalne vrijednosti i optimalnu politiku

Ako se agent nalazi u stanju s , kako bi igrao optimalno, treba odabratи onu akciju koja maksimizira sumu nagrada koje će dobiti uz odabir te akcije i dalje ponovno igrajući optimalno. No tada za funkciju vrijednosti vrijede **Bellmanove jednadžbe**:

$$v^*(s) = \max_a \left(\sum_{s'} p(s'|s, a) [r(s, a, s') + \gamma \cdot v^*(s')] \right) \quad (6)$$

$$v^*(s) = \max_a q^*(s, a) \quad (7)$$

pri čemu smo oznakom $v^*(s)$ označili optimalne vrijednosti stanja, a oznakom $q^*(s, a)$ optimalne q-vrijednosti.

Optimalne vrijednosti stanja odnosno q-vrijednosti su formalno definirane ovako:

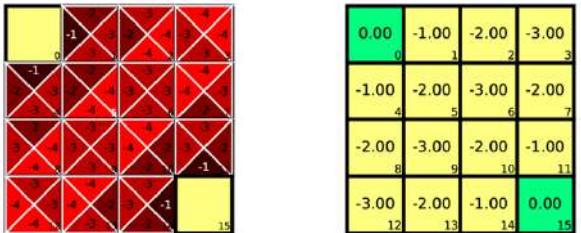
$$v^*(s) = \max_\pi v_\pi(s)$$

$$q^*(s, a) = \max_\pi q_\pi(s, a)$$

gdje maksimum pretražujemo po svim mogućim politikama. Politiku π za koju se taj maksimum postiže označavat ćemo s π^* i zvati **optimalnom politikom**.

13.7. Objasniti algoritam iteracije vrijednosti i provesti ga nad zadanim problemom

Iteracija vrijednosti (engl. *Value iteration*) je postupak sličan vrednovanju politike, samo što se vrijednosti ažuriraju prema Bellmanovoj jednadžbi (6). Lijevo: q-vrijednosti; desno: funkcija vrijednosti.



13.8. Objasniti utjecaj parametra γ te nagrade življena na ponašanje agenta

Zadaca agenta je birati akcije na način koji maksimizira sumu primljenih nagrada, odnosno:

$$R_t = r_{t+1} + \gamma \cdot r_{t+2} + \gamma^2 \cdot r_{t+3} + \gamma^3 \cdot r_{t+4} + \dots + \gamma^{T-t-1} \cdot r_T = \sum_{k=0}^{T-t-1} \gamma^k \cdot r_{t+k+1} \quad (1)$$

pri čemu je γ faktor kojim se balansira između želje da se maksimizira isključivo trenutna nagrada poteza (za $\gamma = 0$) odnosno suma svih dugoročno primljenih nagrada (uz $\gamma = 1$).



13.9. Objasniti ϵ -pohlepnu politiku i svrhu parametra ϵ

Sada ćemo razmotriti slučaj kada agent prolazi epizodu po epizodu, i učenje obavlja izravno tijekom interakcije s okolinom čiji nema model.

Obradit ćemo algoritam **Q-učenja** koji uči q-vrijednosti na temelju kojih je potom moguće izravno odrediti optimalnu politiku.

Algoritam učenja je iterativan, i kreće od početne inicijalizacije q-vrijednosti (primjerice, sve vrijednosti mogu biti 0). Agent prolazi kroz svaku epizodu i u svakom koraku koristi ϵ -pohlepnu politiku koju istovremeno i uči.

ϵ -pohlepna znači da u ϵ posto slučajeva agent odabire slučajno akciju, a u $(1 - \epsilon)$ posto slučajeva bira najbolju akciju prema trenutno naučenoj politici. U ranim fazama ϵ bi trebao biti velik kako bi agentu omogućio da hrabro istražuje; kasnije ϵ treba smanjivati kako bi se agent više oslanjao na naučenu politiku.



13.10. Objasniti algoritam Q-učenja i svrhu parametra α

Algoritam Q-učenja izravno uči q-vrijednosti koje su definirane kao:

$$q(s, a) = r(s, a, s') + \gamma \cdot v_\pi(s')$$

Koristi činjenicu da je:

$$v_\pi(s') = \max_{a'} q(s', a')$$

pa ažuriranje q-vrijednosti radi prema izrazu:

$$q(s, a) \leftarrow (1 - \alpha) \cdot q(s, a) + \alpha \cdot \left(r(s, a, s') + \gamma \cdot \max_{a'} q(s', a') \right) \quad (10)$$

Kod ovog pristupa, optimalna vrijednost stanja s' procjenjuje se na temelju svih procijenjenih q-vrijednosti akcija u stanju s' .

α je stopa učenja i kreće se između 0 i 1. Ako je 0, nema ažuriranja vrijednosti; ako je 1, bitna je samo novoizračunata vrijednost. Vrijednosti između toga rade linearnu interpolaciju, što možemo pokazati ovako:

$$(1 - \alpha) \cdot q(s, a) + \alpha \cdot \left(r(s, a, s') + \gamma \cdot \max_{a'} q(s', a') \right) = \\ q(s, a) + \alpha \cdot \left\{ \left(r(s, a, s') + \gamma \cdot \max_{a'} q(s', a') \right) - q(s, a) \right\}$$

