U N I V E R Z A V L J U B L J A N I
Fakulteta za elektrotehniko

Simon Dobrišek

SKRIPTA PREDAVANJ PRI PREDMETU

UMETNI INTELIGENTNI SISTEMI

UNIVERZITETNI ŠTUDIJSKI PROGRAM II. STOPNJE
ELEKTROTEHNIKA - AVTOMATIKA IN IFORMATIKA

PRVA IZDAJA

Ljubljana, 2013

Predgovor

Skripta predavanj so pomožno študijsko gradivo, ki je na razpolago študentom predmeta Umetni inteligentni sistemi na podiplomskem drugostopenjskem študijskem programu Elektrotehnike na Fakulteti za elektrotehniko Univerze v Ljubljani na študijski smeri Avtomatika in informatika. Vsebujejo preurejene prosojnice predavanja pri tem predmetu.

Cilj predmeta je seznaniti študente z osnovnimi matematičnimi in računalniškimi pristopi v umetni inteligenci, z zasnovami umetnih inteligentnih sistemov in s primeri izvedb takšnih sistemov. Predavanja so razdeljena na naslednje vsebinske sklope:

Uvodne teme: roboti in avtonomni agenti, umetno zaznavanje, umetna inteligenca, mehko računanje, strojno učenje, ambientalna inteligenca.

Inteligentno reševanje problemov: stanja problema in podproblemi, grafska predstavitev problemov, algoritmi za preiskovanje grafov, izčrpno in hevristično preiskovanje, reševanje problema z razgradnjo na podprobleme.

Ekspertni sistemi: znanje - proceduralno in deklarativno, proces sklepanja, primerjava sistemov za obdelavo podatkov (konvencionalni programi) in sistemov za obdelavo znanja, obrazci za prikaz znanja.

Obrazci za prikaz znanja: predstavitev s pravili (izjave ČE...POTEM), neizrazita logika (neizrazite izjave ČE...POTEM) semantična omrežja, omrežje KRP.

Mehanizmi sklepanja : sklepanje z veriženjem pravil, izjavni račun, predikatni račun, pravila logičnega sklepanja, programski jezik Prolog.

Mehko računanje: neizrazito (angl. "fuzzy") računanje, neizrazito sklepanje, genetski algoritmi, evolucijski programi, učenje regresije in razvrščanja z umetnimi nevronskimi omrežji.

Več-agentni sistemi: inteligentni agenti, več-agentni sistemi, programski model BDI, več-agentna programska ogrodja.

Osnovna literature predmeta

- **N. Pavešić**, *Razpoznavanje vzorcev*, Založba FE in FRI, 2012.
- **I. Kononenko**: *Strojno učenje,* Založba FE in FRI, 2005.
- **S. Russel in P Norvig**: *Artificial Intelligence, A Modern Approach,* Prentice Hall, 2009.

V Ljubljani, oktobra 2013 Simon Dobrišek

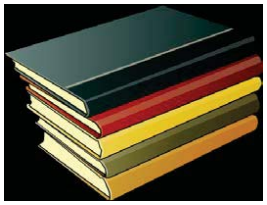
Kazalo

PREDSTAVITEV PREDMETA

VSEBINA

- Opis predmeta
- Izpitni režim
- Vsebina predavanja
- Programski jeziki in orodja
- Laboratorijska učilnica LUKS

Opis Predmeta



Osnovni cilji predmeta

Seznaniti študenta z osnovnimi matematičnimi in računalniškimi pristopi v umetni inteligenci, z zasnovami umetnih inteligentnih sistemov in s primeri izvedb takšnih sistemov.

Predavatelj

doc. dr. Simon Dobrišek – LUKS B204, simon.dobrisek@fe.uni-lj.si

Literatura

N. Pavešić, *Razpoznavanje vzorcev*, Založba FE in FRI, 2012
I. Kononenko: *Strojno učenje*, Založba FE in FRI, 2005.
S. Russel in P. Norvig: *Artificial Intelligence: Modern Approach*, Prentice Hall, 2009.
Gradivo, razdeljeno na predavanjih.

Spletna stran

http://luks.fe.uni-lj.si/sl/studij/UIS

Izpitni Režim

Vaje: Računalniške oziroma programerske naloge v okviru laboratorijskih vaj v učilnici LUKS. Doseganje točk po spodnji tabeli.

Izpit: Doseganje točk z dvema neobveznima domačima nalogama in ustnim izpitom.

Točkovanje izpita		
	največ	najmanj
Domači nalogi	2 × 5	0
Obvezne laboratorijske vaje	3 × 5	7
Izbirni projekt	15	5
Pisni izpit	30	15
Ustni izpit	30	15
Skupaj	100	50

Točke	Ocena
0 – 19	ni frekvence
20 – 49	5
50 – 59	6
60 – 69	7
70 – 79	8
80 – 89	9
90 – 100	10

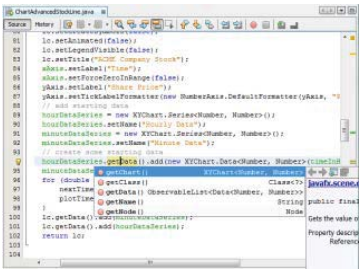
VSEBINA PREDAVANJ (1/2)

- **Uvodne teme:** roboti in avtonomni agenti, umetno zaznavanje, umetna inteligenca, mehko računanje, strojno učenje, ambientalna inteligenca.
- **Inteligentno reševanje problemov:** stanja problema in podproblemi, grafska predstavitev problemov, algoritmi za preiskovanje grafov, izčrpno in hevristično preiskovanje.
- **Ekspertni sistemi:** znanje - proceduralno in deklarativno, proces sklepanja, primerjava sistemov za obdelavo podatkov (konvencionalni programi) in sistemov za obdelavo znanja, obrazci za prikaz znanja.
- **Obrazci za prikaz znanja:** predstavitev s pravili (izjave ČE...POTEM), neizrazita logika (neizrazite izjave ČE...POTEM) semantična omrežja, omrežje KRP.

VSEBINA PREDAVANJ (2/2)

- **Mehanizmi sklepanja :** sklepanje z veriženjem pravil, izjavni račun, predikatni račun, pravila logičnega sklepanja, programski jezik Prolog.
- **Mehko računanje:** genetski algoritmi, evolucijski programi, neizrazito (angl. "fuzzy") računanje.
- **Znanje iz eksperimentalnih podatkov**: učenje regresije (zvezna naključna spremenljivka) in razvrščanja (diskretna naključna spremenljivka) z umetnimi nevronskimi omrežji in stroji podpornih vektorjev.
- **Več-agentni sistemi:** inteligentni agenti, več-agentni sistemi, programski model BDI, več-agentna programska ogrodja.

Programski Jeziki in Orodja



- OpenCV - http://opencv.willowgarage.com/wiki/
- WEKA - http://www.cs.waikato.ac.nz/ml/weka
- SWI Prolog - http://www.swi-prolog.org
- eXpertise2Go - http://www.expertise2go.com
- CLIPS - http://clipsrules.sourceforge.net/
- JESS - http://herzberg.ca.sandia.gov
- JADE - http://jade.tilab.com
- JASON - http://jason.sourceforge.net/wp
- Java, NetBeans, GCC, …

Laboratorij LUKS (1/2)

LUKS
Laboratorij za umetno zaznavanje sisteme in kibernetiko

- Laboratorij za umetno zaznavanje, sisteme in kibernetiko (LUKS) je raziskovalno in pedagoško dejaven na področjih:
  - obdelave signalov ter govornih in slikovnih tehnologij
  - razpoznavanja, analize in razumevanja vzorcev,
  - biometrije in biometričnih varnostnih sistemov
  - umetnih inteligentnih sistemov,
  - teorije informacij in kodiranja.
- Sodelavci laboratorija sodelujejo pri izvajanju študijskega programa druge stopnje predvsem na smeri Avtomatika in informatika.

LABORATORIJ LUKS (2/2)

LUKS
Laboratorij za umetno zaznavanje sisteme in kibernetiko

- V okviru svojih raziskovalnih projektov se posvečajo izbranim sestavinam inteligentnih sistemov, kot so:
  - samodejno razpoznavanje govora in govorcev,
  - tvorjenje umetnega govora,
  - inteligentni govorni vmesniki,
  - biometrično razpoznavanje in preverjanje uporabnikov,
  - podatkovno rudarjenje več-predstavnih vsebin.
- Te sestavine razvijajo kot samostojne izdelke ali pa z njimi dopolnjujejo izdelke interdisciplinarnih sistemov, kot so:
  - inteligentni informacijski sistemi,
  - biometrični varnostni sistemi,
  - ambientalni inteligentni sistemi,
  - pametni nadzorni sistemi.

LABORATORIJSKA UČILNICA

LUKS
Laboratorij za umetno zaznavanje sisteme in kibernetiko

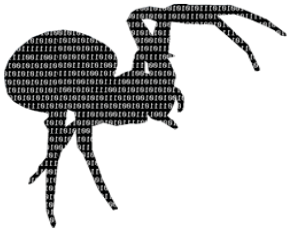
- Več-medijska računalniška učilnica z 18 delovnimi mesti in mrežo 9+2 računalnikov z operacijskim sistemom Linux.
- Učilnica je v prostoru AN204 v drugem nadstropju pred prehodom iz starega v novi del zgradbe fakultete.

UVOD

UVODNE TEME

- Od robotov do avtonomnih agentov
- Umetno zaznavanje
- Umetna inteligenca
- Mehko računanje
- Strojno učenje

Od robotov do avtonomnih agentov



Robot - definicija

- SSKJ: *"Elektronsko vodena **naprava**, ki enakomerno opravlja vnaprej programirana, pogosto človekovemu zdravju škodljiva dela."*
- RIA: *"Programabilni, več-funkcijski **manipulator**, zasnovan za premikanje materiala, delov, orodij ali posebnih naprav s programiranimi gibi za izvedbo zadanih nalog."*
- Wikipedia: *"Navidezni ali elektro-mehanski **umetni avtonomni agent**."*

VRSTE ROBOTOV GLEDE NA UPORABO

- Industrijski in delovni roboti.
- Domači ali hišni roboti.
- Medicinski roboti.
- Servisni in strežni roboti.
- Vojaški in policijski roboti.
- Zabavni roboti.
- Raziskovalni roboti.






VRSTE ROBOTOV GLEDE NA KINEMATIKO

- Stacionarni roboti in robotske roke.
- Mobilni roboti s kolesi.
- Mobilni roboti z nogami.
- Humanoidni roboti.
- Leteči roboti.
- Plavajoči roboti.
- …

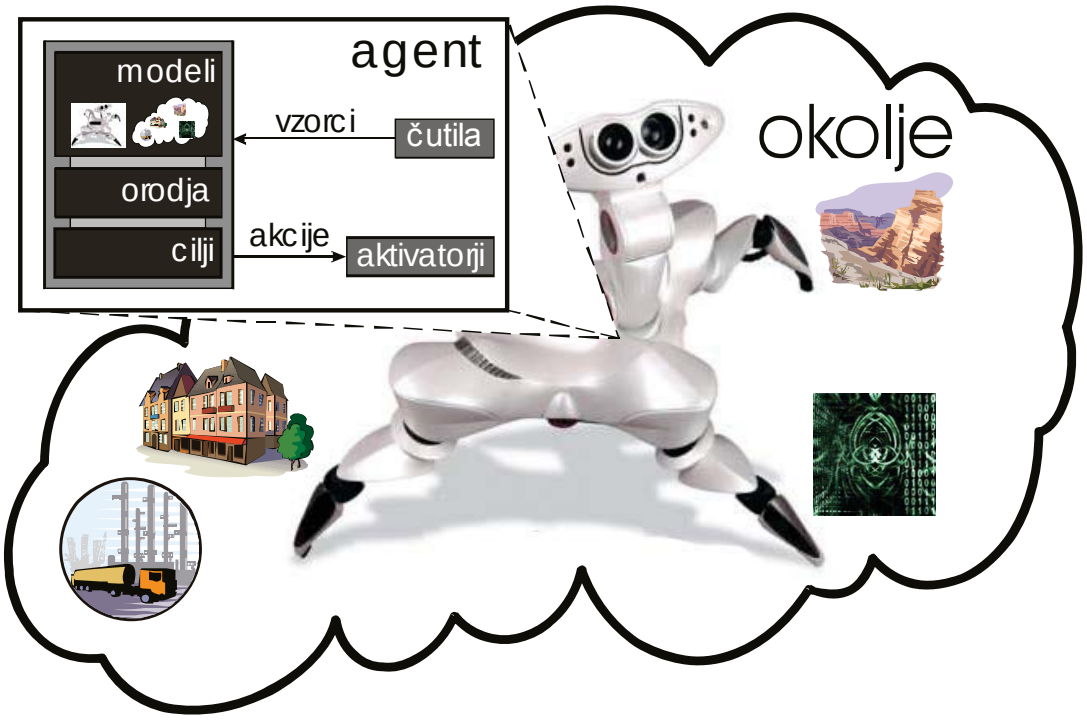




AVTONOMNI AGENT - DEFINICIJA

- Avtonomni agent je sistem:
  - *ki je nameščen v neko okolje ali je njegov del,*
  - *ki to okolje zaznava,*
  - *ki v njem deluje in ga spreminja skozi čas,*
  - *pri čemer sledi neke razvijajoče se zastavljene cilje.*
- Primeri nebioloških avtonomnih agentov so:
  - *avtonomni roboti,*
  - *programski agenti,*
  - *virtualni asistenti,*
  - *agenti umetnega življenja,*
  - *računalniški virusi in črvi.*

OSNOVNI MODEL AVTONOMNEGA AGENTA



Človek kot avtonomni agent

- Človek je najvišji zgled avtonomnega agenta.
- Z umetnimi agenti poskušamo posnemati in preseči naše zmožnosti pri reševanju izbranih nalog.

Vprašanja

- Kako delimo robote glede na uporabo?
- Kako delimo robote glede na kinematiko?
- Kako je definiran avtonomni agent?
- Podajte nekaj primerov avtonomnih agentov.

UMETNO ZAZNAVANJE

UMETNO ZAZNAVANJE

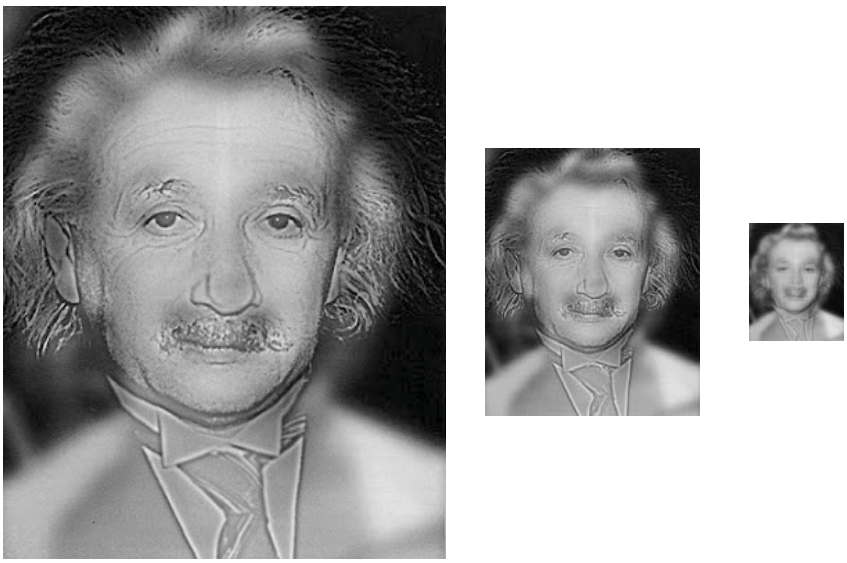
- Zaznavanje je proces, ki agentu mogoči, da se preko čutil ***zaveda prisotnosti*** predmetov, dogodkov ali drugih agentov v okolju, ki ga obkrožajo.
- Zaznavanje je osnova za ***spoznavanje***, ki obsega prevedbo dobljenih spoznanj o okolju v ustrezne pojme.
- Zaznavanju že prej spoznanih predmetov, dogodkov itd, pravimo ***razpoznavanje***
- Umetno zaznavanje je posnemanje teh procesov pri izvedbi umetnih avtonomnih agentov.

UMETNO ZAZNAVANJE Z ANALIZO SLIK

- Pri računalniškem vidu je proces zaznavanje analiza oziroma razčlenjevanje slik na podpodročja, ki predstavljajo smiselne celote (predmete, objekte, bitja itd), ter ugotavljanjem njihovih medsebojnih odnosov in lastnosti.

ZAZNAVANJE IZ SLIK

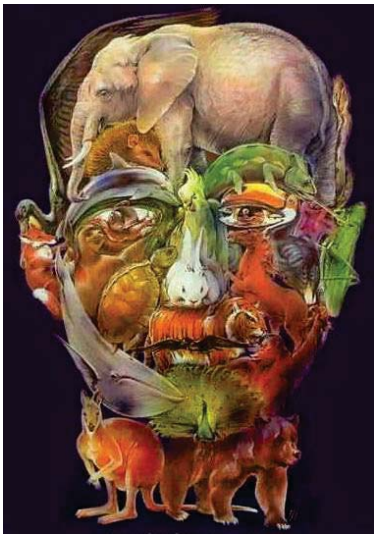
- To, kar ljudje vidimo, ni neka enostavna preslikava dražljajev očesne mrežnice (slike na mrežnici) v simbolno dojemanje slike.



ZAZNAVANJE IZ SLIK

- Optične iluzije razkrivajo kompleksne značilnosti človekovega zaznavanja iz slik.

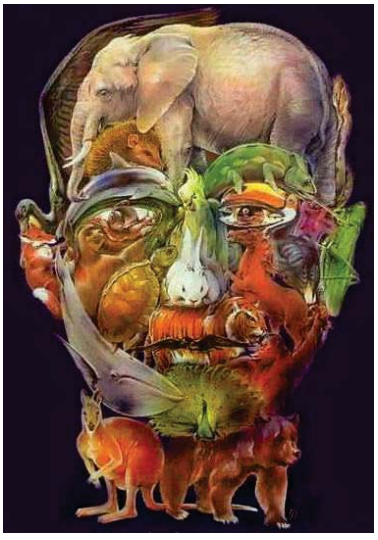
Zaznavanje obrazov



ZAZNAVANJE IZ SLIK

- Optične iluzije razkrivajo kompleksne značilnosti človekovega zaznavanja iz slik.

Zaznavanje obrazov

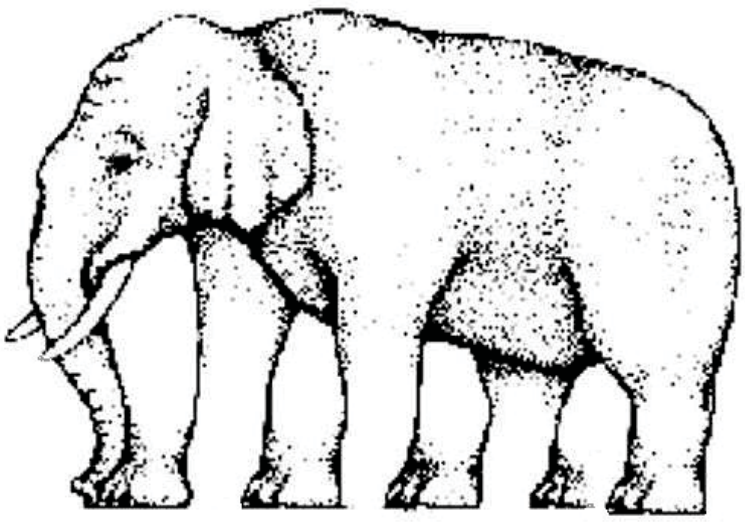


ZAZNAVANJE IZ SLIK

Zaznavanje globine prostora

ZAZNAVANJE IZ SLIK

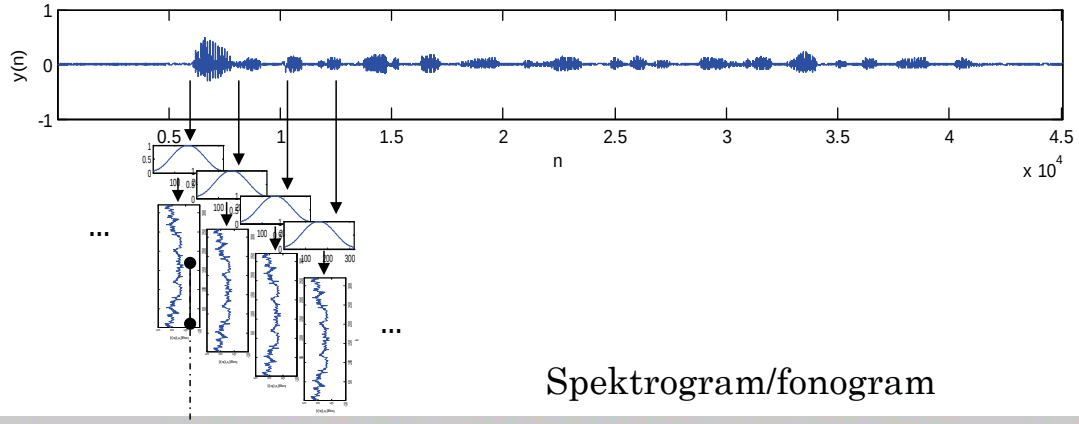
Lokalno in globalno zaznavanje



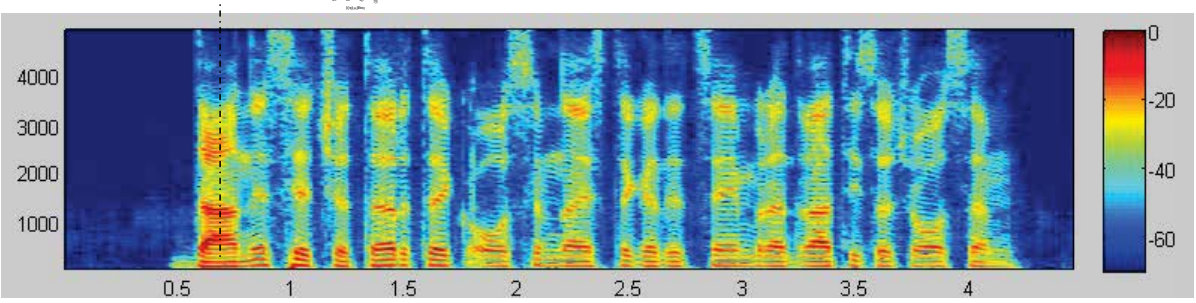
UMETNO ZAZNAVANJE Z ANALIZO ZVOKA

- Razčlenjevanje zvočnega signala na odseke, ki predstavljajo smiselne celote in jih je mogoče povezati z predmeti, bitji, pojavi itd v bližnjem okolju agenta.
- Pri zvočnem zaznavanju je pogosto sočasno prekrivanje/seštevanje zvočnih virov in potrebna je njihova prostorska lokalizacija v okolju sistema.
- Na področju razvoja umetne inteligence ima poseben pomen obravnava zaznavanja in razpoznavanja govora.
- Govorjeni jezik je temeljni izvor človekovega razuma in inteligence.

RAZČLENJEVANJE GOVORNEGA SIGNALA



Spektrogram/fonogram

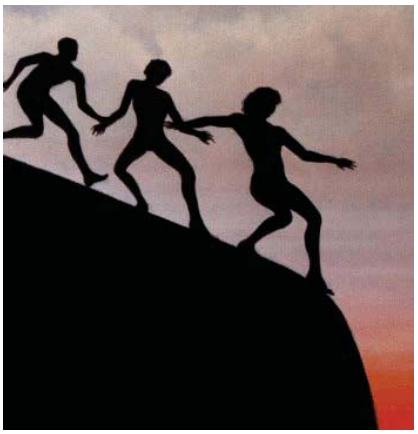


VPRAŠANJA

- Kaj je umetno zaznavanje?
- Kakšna je povezava med umetnim zaznavanjem, spoznavanjem in razpoznavanjem?
- Kako izvedemo umetno zaznavanje z računalniško obdelavo slik?
- Kako izvedemo umetno zaznavanje z računalniško obdelavo zvoka?
- Zakaj je pri razvoju umetne inteligence obravnava govorjenega jezika tako pomembna?

UMETNA INTELIGENCA

KAJ JE INTELIGENCA?



- Inteligenca je sposobnost učinkovitega reševanja problemov na kreativen način, ki ni vnaprej programiran. – Stephen Jay Gould.
- Inteligenca je sposobnost posvajanja in prilagajanja načina reševanja problemov, ki je opažen pri drugih, za svoje lastne potrebe – Jack Copeland.

KAJ DOLOČA INTELIGENCO?

- Sposobnost prilagajanja spremenjenim okoliščinam.
- Sposobnost pomnjenja znanja in njegove uporabe.
- Sposobnost sklepanja in abstraktnega razmišljanja.
- Sposobnost učenja in doumevanja relacij.
- Sposobnost ovrednotenja in presoje/odločanja.
- Sposobnost tvorjenja izvirnih in ustvarjalnih misli.
- …

KAJ JE UMETNA INTELIGENCA?

Vznemirljiv poskus narediti računalnike za misleče stroje. (Haugeland, 1985)	Študij mentalnih funkcij z uporabo računskih modelov. (Charniak in McDermont, 1985)
Avtomatizacija aktivnosti, ki jih povezujemo s človekovim razmišljanjem, odločanjem, reševanjem problemov, učenjem, … (Bellman, 1978)	Študija računskih modelov, ki omogočajo zaznavanje, razumevanje in delovanje (Winston, 1992)
Umetnost ustvarjanja strojev, ki izvajajo opravila, ki zahtevajo inteligenco, če jih izvaja človek. (Kurzweil, 1990)	Področje, ki želi razložiti in posnemati inteligentno obnašanje v smislu računskih procesov. (Schalkoff, 1990)
Razvoj strojev za dela, ki jih človek zaenkrat opravlja še bolje. (Rich in Knight, 1990)	Veja računalniške znanosti, k se ukvarja z avtomatizacijo inteligentnega obnašanja. (Luger in Stubblefield, 1993)

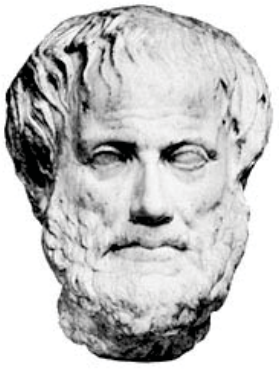
KATEGORIZACIJA DEFINICIJ UI

- Zgornje definicije se ukvarjajo z umovanjem.
- Spodnje definicije se ukvarjajo z obnašanjem.
- Leve definicije se ukvarjajo s primerjavo s človeškimi zmogljivostmi.
- Desne definicije se ukvarjajo s primerjavo z idealom inteligence, ki mu pravimo racionalnost.

INTELIGENTNO OBNAŠANJE

- Inteligentno obnašanje naj bi vključevalo:
  - zaznavanja okolja,
  - delovanje v zahtevnih okoljih,
  - učenje in razumevanje iz primerov,
  - sklepanje za reševanje problemov,
  - pridobivanje skritega znanja s sklepanjem,
  - uporabo znanja v novih okoliščinah,
  - komunikacijo z drugimi in razumevanja jezikov,
  - kreativnost, izvirnost, radovednost, …

RACIONALNO MIŠLJENJE IN DELOVANJE



- Racionalno mišljenje je mišljenje v skladu s principi formalne logike
  - Pravilne predpostavke morajo pripeljati k pravilnim sklepom
- Racionalno delovanje je delovanje z:
  - jasnimi preferencami (prednostmi),
  - upoštevanjem negotovosti pri svojih pričakovanjih in
  - izbiro akcij/operacij z najboljšimi pričakovanimi izidi.

KAJ MORAMO RAZUMETI PRI RAZVOJU UMETNE INTELIGENCE?

- Kako se znanje pridobi, predstavi in shrani;
- Kako se ustvari in nauči inteligentnega obnašanja;
- Kako se razvijejo in uporabijo prepričanja, namere, želje, emocije in preference;
- Kako se senzorski signali pretvorijo v simbole;
- Kako se upravlja s simboli za logično sklepanje o preteklosti in načrtih za prihodnost;
- Kako se komunicira in razume jezik.

MOČNA UMETNA INTELIGENCA

- Močna umetna inteligenca se nanaša na razvoj strojev, ki naj bi v celoti dosegli ali presegli človekove razumske in kognitivne sposobnosti.
- Cilj močne umetne inteligence je razvoj strojev, ki jih po intelektualnih sposobnostih ni mogoče razlikovati od ljudi.
- Močna umetna inteligenca naj bi vsaj do neke mere izkazovala samo-zavedanje.

Šibka umetna inteligenca

- Šibka umetna inteligenca se nanaša na razvoj strojev, ki zmorejo reševati specifične zahtevne probleme in umovanje, ki ne zahteva celotne palete človekovih razumskih in kognitivnih sposobnosti.
- Za šibko umetno inteligenco se ne zahteva izkazovanje samo-zavedanje.

Cilji umetne inteligence

- Razvoj sistema, ki misli in/ali deluje kot človek.
- Razvoj sistema, ki misli in/ali deluje racionalno.

	človeku podobno	racionalno
Misli	Kognitivna znanost	Raziskave principov umovanja
Deluje	Turingov pristop	Razvoj racionalnega agenta

- Kognitivno znanost zanima predvsem razvoj sistema, ki dejansko misli kot človek.
- Pristop s Turingovim preizkusom se posveča razvoju sistema, ki vsaj navzven deluje kot človek.
- Razvoj umetne inteligence pa se v glavnem posveča razvoju sistemov, ki mislijo in/ali delujejo racionalno.

TURINGOV PREIZKUS UMETNE INTELIGENCE

- Izvedba preizkusa
  - Pogovor človeka s prikritim strojem na eni strani ter človekom na drugi strani.
  - Preizkus je uspešen, če ne more ugotoviti, kateri od sogovornikov ni človek.
- Zadržki pri tem preizkusu
  - Stroj, ki simulira človeške pogovorne navade ni nujno inteligenten.
  - Stroj bi lahko bil nadvse inteligenten, pa vendar ni zmožen debatirati s človekom.
  - Veliko manj izobraženim ljudem bi spodletelo na tem preizkusu.

NAPOVEDI ALANA TURINGA ZA LETO 2000

- Stroji z 10^9 biti (119MB) bodo sposobni pretentati tretjino človeških sodnikov med pet-minutnim testom.
- Ljudje bodo sprejeli izraz “misleč stroj”.
- Učenje bo postalo ključni del pri gradnji zmogljivih strojev.

Inteligentni klepetalniki

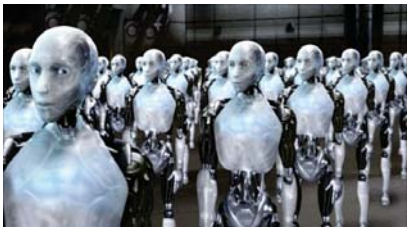
- Obstajajo spletni sistemi, ki se zgledujejo po Turingovem preizkusu umetne inteligence, vendar ga do danes še ne bi prestali.

  ELIZA
  http://www-ai.ijs.si/eliza/eliza.html

  ALICE
  http://alice.pandorabots.com/

Možne posledice razvoja UI

- Artilekt – naslednik človeške vrste.
- Ali lahko UI definiramo kot novo vrsto?
- Svoboščine in pravice teh sistemov.
- Posebnost in edinost teh sistemov.
- Stopnja varnostne zaščite.



Področja Umetne Inteligence

- Umetna inteligenca kot znanost:
  - poskuša razumeti in modelirati človekove sposobnosti obdelave informacije;
  - poskuša razumeti splošne principe za razlago in razumevanje različnih inteligentnih sistemov.
- Umetna inteligenca kot inženirstvo:
  - poskuša razviti nove stroje, ki zmorejo opravljati dela, ki so jih pred tem zmogli opravljati le ljudje.
  - poskuša razviti inteligentne stroje, okolja in učne pripomočke, ki pomagajo ljudem pri bivanju in reševanju problemov.

Podpodročja umetne inteligence

- Delitev glede na vsebino
  - Umetno zaznavanje
  - Obdelava naravnih jezikov
  - Učenje in razvoj
  - Načrtovanje in reševanje problemov
  - Sklepanje in izpeljevanje
  - Robotika
  - Več-agentni sistemi
  - Mehanizmi pomnjenja
  - Predstavitev znanja
  - Razvoj programskih jezikov in orodij
  - …

PODPODROČJA UMETNE INTELIGENCE

- Delitev glede na uporabo
  - Medicina
  - Industrijski in proizvodni sistemi
  - Sistemi za pomoč uporabnikom
  - Robotika
  - Izobraževanje
  - Informacijski sistemi
  - Zabavna industrija
  - Preprečevanje kriminala
  - Vojska
  - Vesoljske raziskave
  - Pravni in socialni sistemi
  - Marketing in prodaja
  - Finančni sistemi

PODPODROČJA UMETNE INTELIGENCE

- Delitev glede na raziskovalna področja
  - Kombinatorično iskanje
  - Strojni, računalniški vid
  - Ekspertni sistemi
  - Mehko računanje
  - Predstavitev in prikaz znanja
  - Strojno učenje in podatkovno rudarjenje
  - Razpoznavanje vzorcev
  - Obdelava naravnega jezika
  - Umetno življenje
  - Robotika
  - Ambientalna inteligenca in pametni prostori
  - Pametni nadzorni sistemi
  - …

MEJNIKI UMETNE INTELIGENCE

- 1997: IBM Deep Blue šahovski program premaga svetovnega prvaka Gary Kasparova.
- 2005: Avtonomni avtomobil Univerze Stanford prevozi 132 miles do cilja po puščavskih cestah (DARPA Grand Challenge).
- 2011: Računalniški program IBM Watson premaga ljudi na kvizu Jeopardy.

VPRAŠANJA

- Kaj določa inteligenco?
- Kako je definirana umetna inteligenca (UI)?
- Kaj moramo razumeti pri razvoju UI?
- Kakšna je razlika med močno in šibko UI
- Katera so podpodročja UI glede na vsebino?
- Katera so podpodročja UI glede na uporabo?
- Katera raziskovalna področja se ubadajo z UI?
- Kakšne so možne posledice razvoja UI?

Mehko Računanje

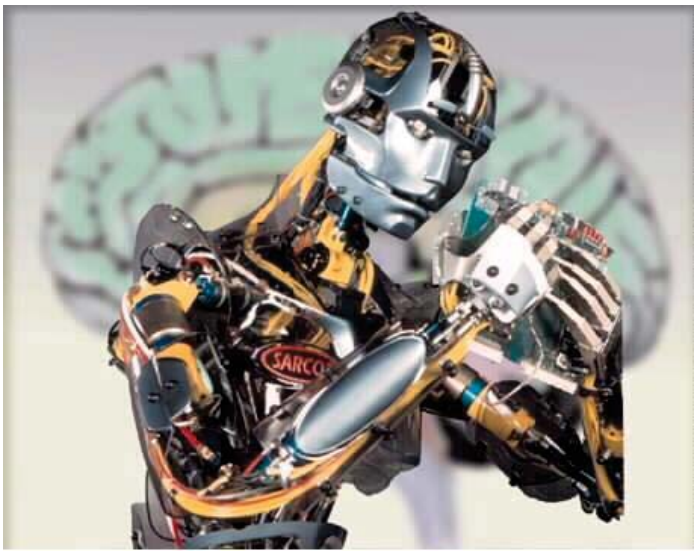
Mehko Računanje

- Mehko računanje zajema širše področje računalniških metod in postopkov za modeliranje in reševanje problemov, ki so prezahtevni za klasično matematično modeliranje.
- Mehko računanje temelji na približnem računanju, vključevanju negotovosti, ohlapnem sklepanju, delnih resnicah, s čimer se zgleduje po človeškem načinu umovanja in odločanja.
- Mehko računanje zajema znanstvena področja, kot so neizrazita („fuzzy“) logika, modeliranje nevronskih omrežij, evolucijsko, genetsko in verjetnostno računanje.

VPRAŠANJA

- Na kakšnem računanju temelji mehko računanje?
- Kako se mehko računanje zgleduje po človeškem umovanju?
- Katera znanstvena področja vključuje mehko računanje?

STROJNO UČENJE



STROJNO UČENJE

- Veja raziskav umetne inteligence, ki temelji na modeliranju okolja in pojavov iz podatkov in je sorodna področju samodejnega razpoznavanja vzorcev.
- Razvoj tehnik, ki dajejo računalniškim sistemom sposobnost „učenja“ in induktivnega verjetnostnega sklepanja iz pridobljenih (senzorskih) podatkov.
- Učenje je proces adaptivnih sprememb sistema, ki mu omogočijo, da naslednjič reši nalogo iste vrste bolj učinkovito.

Vrste Naravnega Učenja

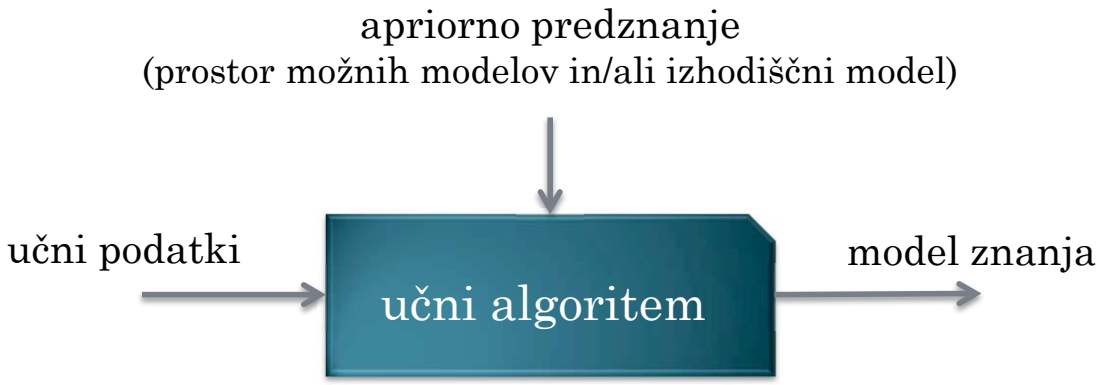
- Učenje z vtisnjenjem (nespremenljivo znanje)
- Učenje s pogojevanjem (Pavlov)
- Verjetnostno (Bayesovo) učenje
- Učenje z zapominjanjem (memoriranjem)
- Učenje s poskusi in napakami
- Učenje s posnemanjem
- Učenje z razumevanjem in vpogledom

Vrste Strojnega Učenja

- Učenje z učiteljem (nadzorovano učenje).
- Učenje brez učitelja (nenadzorovano učenje).
- Delno nadzorovano učenje.
- Spodbujevalno/okrepitveno učenje.
- Učenje transdukcije/pretvorbe
- Učenje učenja.

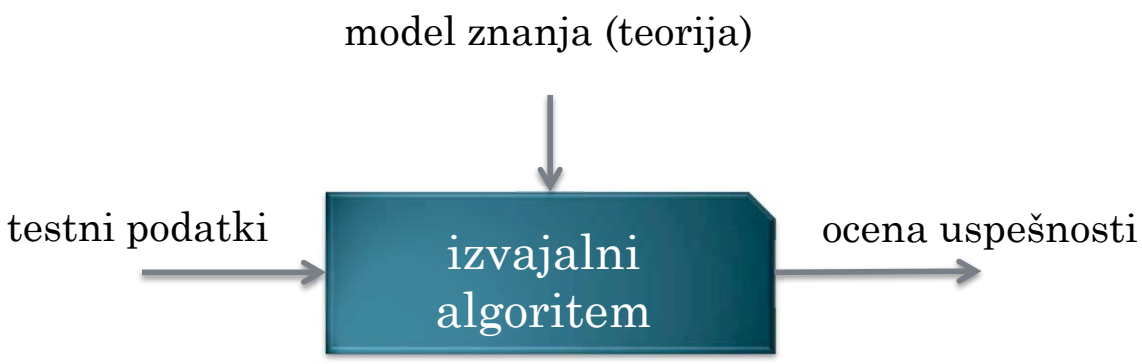
Osnovni Principi strojnega Učenja

- Osnovi pojmi so, podatek, predznanje, učni algoritem, izvajalni algoritem, model znanja (teorija).



Ocenjevanje Uspešnosti

- Ocenjevanje uspešnosti reševanja novih problemov.



Primer učenja z Bayesovim sklepanjem

- Predznanje pri Bayesovem sklepanju modeliramo kot vnaprejšnje prepričanje (zaupanje, negotovost), ki ji pravimo *prior*.
- V skladu z Bayesovim teoremom prepričanje o modelu M prilagodimo glede na verjetje modela o pridobljenem podatku (očitnosti) E.
- Prilagojenemu naknadnemu prepričanju pravimo *posterior*.

$M \in \{M_i\}$

$E \in \{E_j\}$

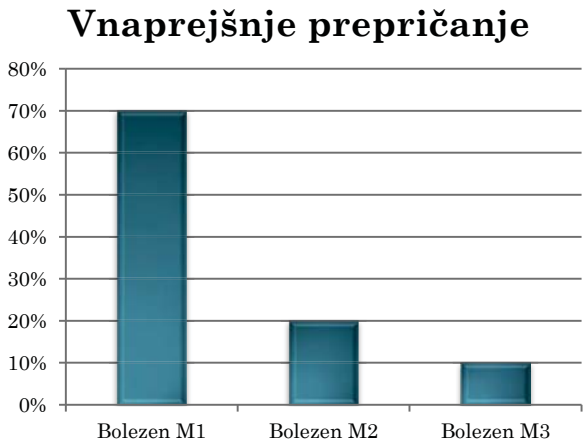
$P(E) = \sum_i P(M_i)P(E|M_i)$

$$P(M|E) = P(M)\ \frac{P(E|M)}{P(E)}$$

$P(M|E) \to P(M)$

Primer učenja z Bayesovim sklepanjem

- Zdravnik obravnava pacienta, ki ima določene neobičajne simptome.
- Zdravnikovo predznanje mu pravi, da ima 70% ljudi s takšnimi simptomi bolezen M_1, 20 % bolezen M_2 in 10 % bolezen M_3.
- Zdravnikovo vnaprejšnje prepričanje o bolezni je torej mogoče ponazoriti z diagramom



Primer učenja z Bayesovim sklepanjem

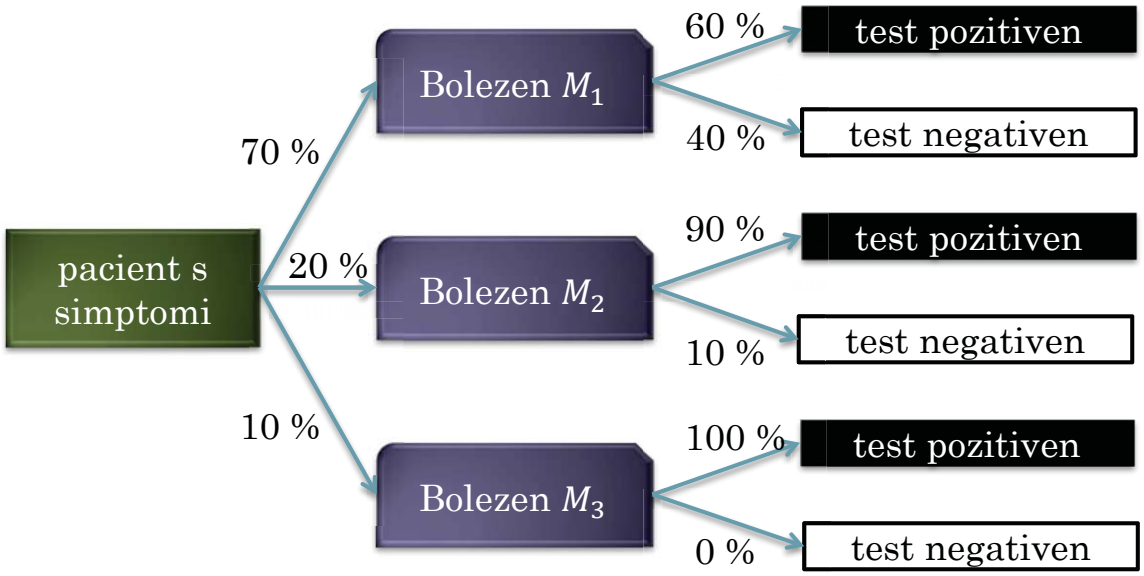
- Zdravnik vzame pacientu vzorec sline in ga v epruveti zmeša s posebno diagnostično kemikalijo.
- Za kemikalijo je znano, da se obarva črno pri 60% pacientov z boleznijo M_1, 90% pacientov z boleznijo M_2 in 100% pacientov z boleznijo M_3, torej

$$P(E|M_1) = 0{,}6 \qquad P(E|M_2) = 0{,}9 \qquad P(E|M_3) = 1{,}0$$

- Kemikalije se obarva črno (očitnost E).
- Kakšno je zdaj zdravnikovo prepričanje o morebitni bolezni?

Primer učenja z Bayesovim sklepanjem

- Drevo možnih dogodkov s podanimi verjetnostmi.



Primer učenja z Bayesovim sklepanjem

- Verjetnost, da imam zdravnik opravka s pacientom, ki ima eno od bolezni in bo test pozitiven je torej

$$P(\mathrm{E}, \mathrm{M}_1) = P(M_1) \cdot P(E|M_1) = 0{,}7 \cdot 0{,}6 = 0{,}42$$
$$P(\mathrm{E}, \mathrm{M}_2) = P(M_2) \cdot P(E|M_2) = 0{,}2 \cdot 0{,}9 = 0{,}18$$
$$P(\mathrm{E}, \mathrm{M}_3) = P(M_3) \cdot P(E|M_3) = 0{,}1 \cdot 1{,}0 = 0{,}1$$

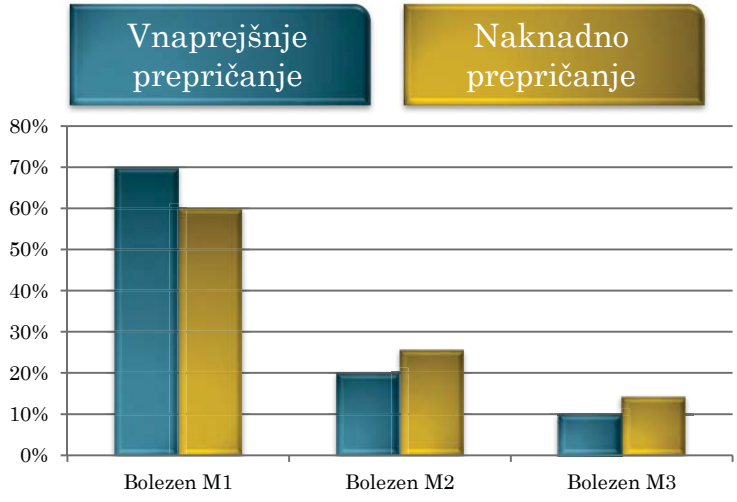
- Zdravnikovo naknadno prepričanje po pozitivnem testu se torej spremeni

$$P(E) = \sum_i P(M_i)P(E|M_i) = 0{,}42 + 0{,}18 + 0{,}1 = 0{,}7$$

$$P(M_1|E) = \frac{0{,}42}{0{,}42+0{,}18+0{,}1} = 0{,}6$$
$$P(M_2|E) = \frac{0{,}18}{0{,}42+0{,}18+0{,}1} = 0{,}257$$
$$P(M_3|E) = \frac{0{,}1}{0{,}42+0{,}18+0{,}1} = 0{,}143$$

Primer učenja z Bayesovim sklepanjem

- Zdravnikovo naknadno prepričanje se tako spremeni, vendar bi se še vedno odločil, da ima pacient bolezen M_1, kljub temu, da je pri vseh bolnikih z boleznijo M_3 test pozitiven, kar se je zgodilo tudi pri danem pacientu.



VPRAŠANJA

- Kaj je strojno učenje?
- Kakšne so vrste naravnega učenja?
- Kakšne vrste strojnega učenja poznamo?
- Kakšni so osnovni principi strojnega učenja?
- Podajte primer učenja z Bayesovim sklepanjem.

Umetni inteligentni sistemi, UN2-1-IZB-vsi 2013/2014
Avtorske pravice pridržane © 2013 - Univerza v Ljubljani, Fakulteta za elektrotehniko

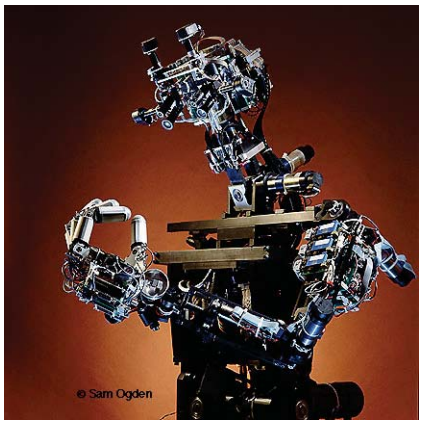
AMBIENTALNI INTELIGENTNI SISTEMI

VSEBINA

- Uvod in terminologija
- Ambientalna inteligenca (AmI)
- Podpora obstoječim razvojnim strategijam
- Razvoj gradnikov AmI sistemov
- Možen scenarij izkušnje z AmI sistemom.
- Raziskovalne in razvojne iniciative
- Zadržki

TERMINOLOGIJA

- Premik raziskovalnega poudarka iz
  - na tehniko osredotočene ***umetne inteligence***,
  - na človeka osredotočeno ***ambientalno inteligenco***.



TERMINOLOGIJA IN SORODNI POJMI

- Pojem ***inteligentnih hiš*** se je sčasoma razpršil na več novih pojmov.

- Poskušajo se uveljaviti novi pojmi kot so:
  - pametne hiše/domovi/okolja,
  - integrirane hiše/domovi/okolja,
  - žive/interaktivne hiše/domovi/okolje.

POJAV VSEPRISOTNEGA RAČUNALNIKA

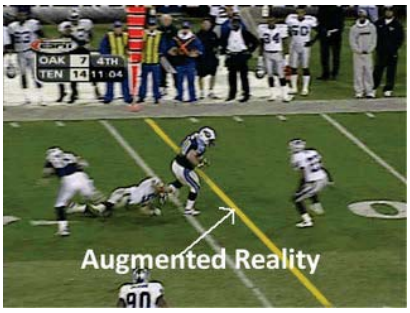
- "Ubiquitous intelligence/computing"
- "Pervasive intelligence/computing"
- "Versatile intelligence/computing"
- "Everyware"
- …

VSEPRISOTNI RAČUNALNIK

- Interakcija med človekom in računalnikom bo sčasoma presegla danes prevladujoče namizno delo

IZBOLJŠANA RESNIČNOST

- Kombinacija resničnih in umetno tvorjenih podatkov.



AMBIENTALNA INTELIGENCA

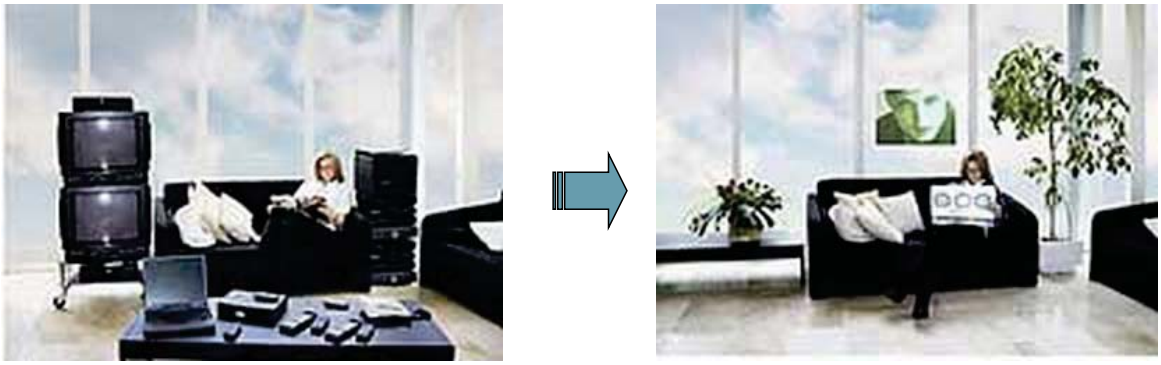
- Ambientalna Inteligenca (AmI) je zgled
  - ***vseprisotnega***,
  - ***nevsiljivega***,
  - ***preglednega*** in
  - ***inteligentnega***

  podpornega sistema za uporabnike.

- Pri ambientalni inteligenci so uporabniki obkroženi z inteligentnimi uporabniškimi vmesniki, ki jih podpira komunikacijsko omrežena tehnologija, vgrajena v običajne predmete in objekte.

AMBIENTALNA INTELIGENCA

- Informacijsko-komunikacijske tehnološke (IKT) komponente so skrite v ozadje, pri čemer so ljudje postavljeni v ospredje pri nadzoru izboljšanega okolja.



KORENINE AMBIENTALNE INTELIGENCE

- Interne delavnice v podjetju Phillips na temo scenarijev uporabniku bolj prijazne interakcije z digitalnimi napravami – 1998.
- D. A. Norman, "The invisible computer", 1999.
- "The AmI challenge", razpis Evropske Komisije, 2001.
- Poročilo evropske izvedenske skupine IS-TAG, "Scenarios for Ambient Intelligence in 2010", 2001

VIZIJA AMI OKOLJA

- AmI okolje je tehnološko izboljšano okolje, ki:
  - se zaveda prisotnosti ljudi,
  - se prilagaja njihovim potrebam,
  - se inteligentno odziva na govor in geste,
  - je sposobno izvajati inteligentni dialog,
  - je nevsiljivo in podpira sproščeno interakcijo,
  - vključuje interoperabilnost med različnimi okolji (domovi, delovni prostori, vozila, javni prostori, …)

UPORABNA VREDNOST ZAMISLI AMI

- Modernizacija socialnega modela
- Izboljšanje varnosti državljanov
- Nove možnosti za delo, učenje in zabavo
- Podpora gradnji socialne skupnosti in skupin
- Nove oblike zdravstvenega in socialnega varstva
- Spoprijemanje z okoljevarstvenimi problemi
- Izboljšanje javnega servisa
- Podpora demokratičnemu političnemu procesu.

GRADNJA SOCIALNE SKUPNOSTI IN SKUPIN

- AmI lahko izboljša soudeležbo posameznikov v socialnih omrežjih z vzpostavitvijo okolij, ki podpirajo razvoj skupnega življenja in dejavnosti.

ZDRAVSTVENO IN SOCIALNO VARSTVO

- AmI okolje omogoča posameznikom bolj odgovorno in aktivno upravljanje s svojim zdravjem.
- AmI omogoča enostavnejše in bolj učinkovito spremljanje zdravstvenega stanja prizadetih.
- AmI omogoča učinkovito podporo varstva ranljivih skupin kot so starejši in otroci.

PODPORA DOMAČEMU OKOLJU

- Domače AmI okolje lahko izpolni in obogati posameznika z bolj fleksibilno soudeležbo pri delu, učenju, zabavi in druženju z drugimi ljudmi.

UPRAVLJANJE IN JAVNI SERVIS

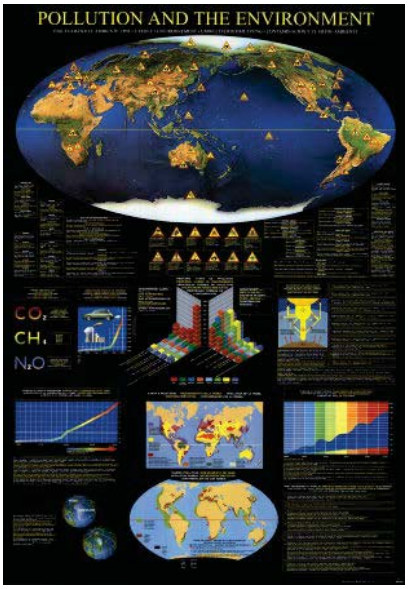
- AmI omogoča vzpostavitev ekonomskega in socialnega javnega servisa, ki je resnično uporabniško prijazen do vseh uporabnikov kjerkoli in kadarkoli.

VARNOST DRŽAVLJANOV

- AmI lahko prispeva k zanesljivi oceni rizikov ter odkrivanju nevarnosti z daljinskimi zaznavanjem in nadzorom.

VAROVANJE OKOLJA

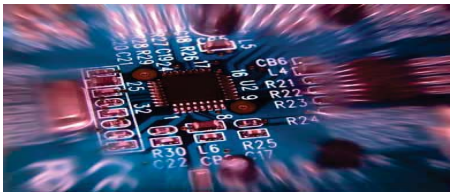
- AmI ponuja nove možnosti pri varovanju okolja in premik iz običajnega spremljanja stanja okolja k upravljanju z znanjem za naprednejšo podporo pri odločanju.

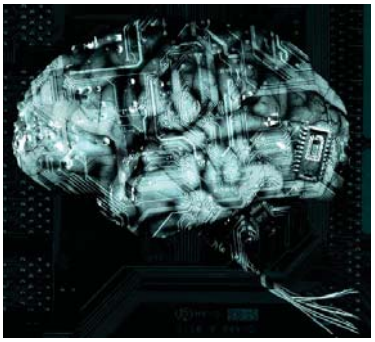


Mobilnost in transport

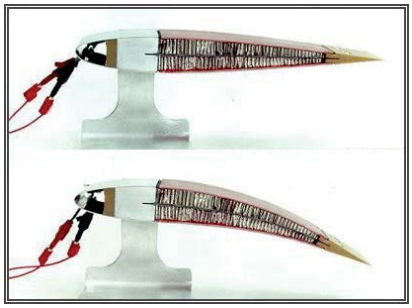
- AmI okolje lahko precej izboljša varnost v prometu in splošno učinkovitost transportnega sistema.

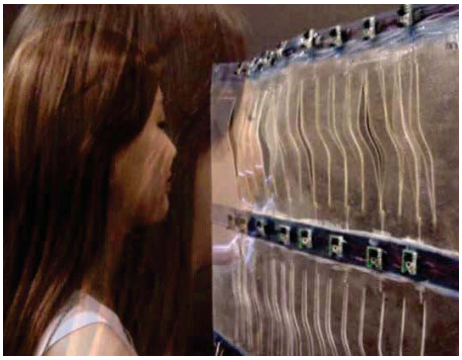
Razvoj glavnih sestavin AmI





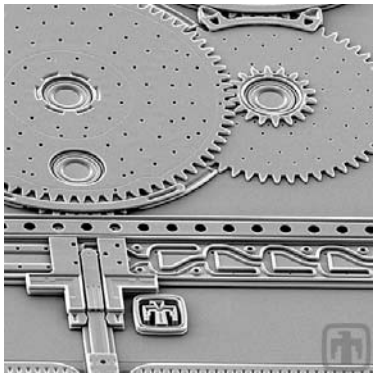
SESTAVINE ZA AMBIENT (1/4)





- Pametni materiali, ki
  - nadzorovano oddajajo svetlobo,
  - so občutljivi na dotik,
  - imajo spomin in obdelujejo podatke,
  - jih lahko vgradimo v blago ipd

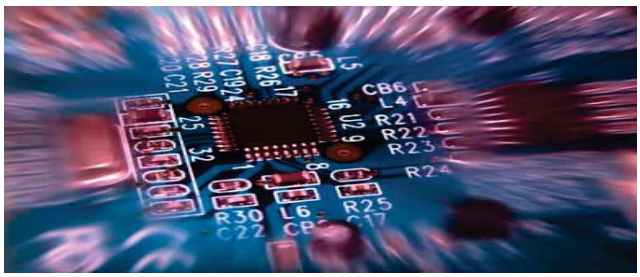
SESTAVINE ZA AMBIENT (2/4)



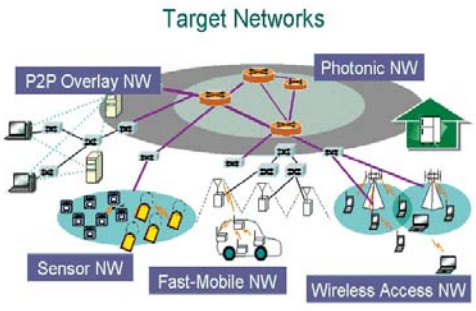
- Mikro elektro-mehanski sistemi in senzorji
  - Nizko energijski aktivatorji
  - Senzorji za dotiki, zvok, vid, vonj, …
  - Integratorji pametnih materialov

SESTAVINE ZA AMBIENT (3/4)

- Vgradni sistemi
  - Prilagodljivi sistemi, ki tečejo v stvarnem času
  - Možnost diagnostike in popravila na daljavo
  - Poudarek na varnosti in zanesljivosti delovanja



SESTAVINE ZA AMBIENT (4/4)



- Vseprisotne komunikacije
  - Piko radijska omrežja za označevanje
  - Internetna dosegljivost za vse objekte
  - Vseprisotni širokopasovni doseg do podatkov
  - Vseprisotni prostoročen govorni nadzor
  - Grafični porabniški vmesniki na poljubni površini
  - Programska oprema, ki se prilagaja okolju

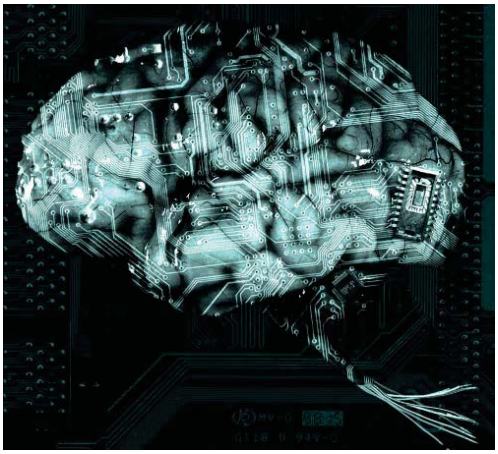
SESTAVINE ZA INTELIGENCO (1/5)

- Upravljanje z vsebinami
  - Jeziki za predstavitev vsebin
  - Orodja za dostop do semantičnega spleta
  - Orodja za analizo vsebin
  - Samodejno bogatenje vsebin z meta podatki

SESTAVINE ZA INTELIGENCO (2/5)

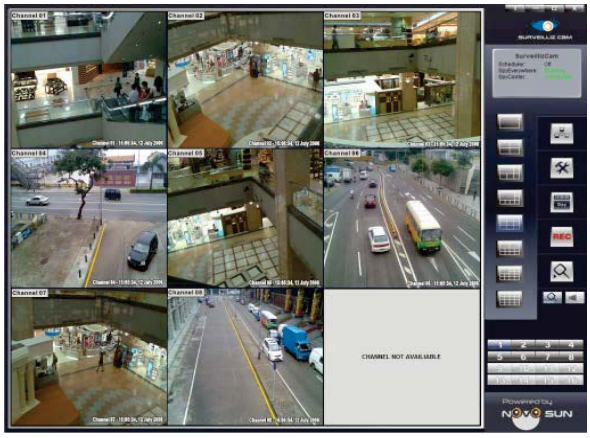
- Več-modalna naravna interakcija, ki vključuje
  - obrazno mimiko,
  - geste,
  - govor,
  - umetni vid.

SESTAVINE ZA INTELIGENCO (3/5)



- Računalniška inteligenca
  - Interaktivno iskanje vsebin
  - Sistemi za dialog
  - Vedenjsko modeliranje
  - Reševanje problemov in načrtovanje

SESTAVINE ZA INTELIGENCO (4/5)



- Zavedanje konteksta
  - Navigacijski sistemi
  - Sistemi za lokalizacijo in sledenje
  - Samodejno iskanje podpore
  - Samodejen nadzor

Sestavine za inteligenco (5/5)

- Računalniške emocije
  - Modeliranje emocionalnih stanj
  - Odzivanje na emocionalna stanja uporabnikov
  - Izražanje emocionalnih stanj

Primeri scenarijev AmI

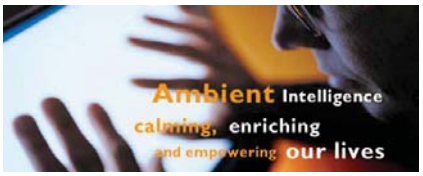
http://www.mojmikro.si/preziveti/in_se/vseprisotni_sluzabnik_v_ozadju

RAZISKOVALNE INICIATIVE

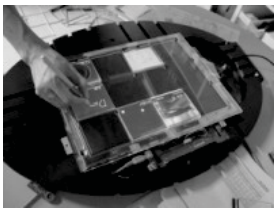
- The Aware Home Research Initiative
  http://awarehome.imtc.gatech.edu/

- MIT Project Oxygen
  http://oxygen.csail.mit.edu/

- Philips HomeLab
  http://www.research.philips.com/



- Sony CSL Interaction Laboratory
  http://www.sonycsl.co.jp/IL/



ZADRŽKI

- Socialni, politični in kulturni zadržki zaradi možnosti izgube zasebnosti in koncentracije prevelike moči v posameznih organizacijah.

- Možnost razvoja pretirano individualizirane in fragmentirane družbe, v kateri posamezniki v svojem hiperrealnem svetu ne bodo več ločili med stvarnim in navideznim.

VPRAŠANJA

- Kakšen miselni premik naredimo pri prehodu iz umetne inteligence v ambientalno inteligenco?
- Kaj razumemo pod pojmom "vseprisotni izginjajoči računalnik"?
- Kakšna je vizija AmI okolja?
- Kakšna je uporabne vrednosti zamisli AmI okolja?
- Katere so glavne vrste sestavin AmI okolja?
- Kakšni so zadržki pri razvoju AmI okolja?

RAZISKOVALNE SKUPINE

- Alcatel-Lucent Research&Innovation. Ambient Services Group
- Autonomous University of Madrid - AmiLab
- Carnegie Mellon University. CyLab - Ambient Intelligence Lab
- Fraunhofer Institute. Ambient Assisted Living
- Fraunhofer Institute. InHaus
- Kingston University London, Ambient Intelligence Research Group
- MAmI -- Modelling Ambient Intelligence -- UCLM, Spain
- MERL. Ambient Intelligence for Better Buildings
- MIT Media Lab. Ambient Intelligence group
- NTT Research. Ambient Intelligence Research Group
- Philips Research. Ambient Intelligence Research in ExperienceLab
- University of Reading, Ambient & Pervasive Intelligence Research group
- SERENITY Security & Dependability in AmI
- SWAMI: Safeguards in a World of Ambient Intelligence
- GECAD - Knowledge Engineering and Decision Support Research Center

PUBLIKACIJE

- SAME Series - Semantic Ambient Media Series Workshop
- STAMI Series - Space, Time and Ambient Intelligence (STAMI). International Workshop Series.
- UCAmI '10 - Symposium of Ubiquitous Computing and Ambient Intelligence
- HAI'09 - International Workshop on Human Aspects in Ambient Intelligence
- COSIT-Space-AmI-09 - Workshop on "Spatial and Temporal Reasoning for Ambient Intelligence Systems"
- Sensami - a congress on ambient intelligence.
- AITAmI - Workshop on "Artificial Intelligence Techniques for Ambient Intelligence"
- IJACI - The International Journal of Ambient Computing and Intelligence
- JAISE - The International Journal of Ambient Intelligence and Smart Environments.
- AISE - Book Series on Ambient Intelligence and Smart Environments.
- I-o-T.org - Internet of Things : mainly based on Ambient intelligence
- IE'09 - Intelligent Environments Conference 2009

Umetni inteligentni sistemi, UN2-1-IZB-vsi 2013/2014
Avtorske pravice pridržane © 2013 - Univerza v Ljubljani, Fakulteta za elektrotehniko

INTELIGENTNO REŠEVANJE PROBLEMOV

VSEBINA

- Splošno reševanje problemov
- Grafska predstavitev problemov
- Primeri enostanjskih problemov
- Reševanje problemov z iskanjem
- Algoritmi za preiskovanje drevesa problema
- Reševanje problemov z razgradnjo na podprobleme.

SPLOŠNO REŠEVANJE PROBLEMOV

- Reševanje problemov je značilnost mnogih sistemov, ki temeljijo na umetni inteligenci.
- V resničnem življenju je malo problemov, ki so enostavno rešljivi.

- Reševanje problemov je proces tvorjenja rešitev iz pridobljenih podatkov oziroma zaznav.
- Problem označuje množica ciljev, množica objektov in množica operacij na objektih, ki so lahko slabo definirane in se lahko izboljšujejo tekom reševanja problema.

VRSTE PROBLEMOV

- Deterministični in v celoti opazljiv (enostanjski problem).
  - *Agent v celoti pozna (svoje) trenutno stanje pri reševanju problema, rešitev je zaporedje operacij.*
- Ne opazljiv (skladnostni problem).
  - *Agent lahko ne ve kakšno je (njegovo) trenutno stanje pri reševanju problema in samo predvideva posledice operacij (brez senzorjev); rešitev, če obstaja, je zaporedje operacij.*
- Nedeterministični in/ali delno opazljiv (nepredvideni problem).
  - *Zaznave podajajo novo informacijo o trenutnem stanju, izmenjevanje izvajanja iskanja in izvajanja operacij, rešitev je drevo ali pravila vodenja odločanja (politika) .*
- Nedeterministični in ne opazljiv (raziskovalni problem)

Reševanje enostavnih problemov

- Enostavni problemi so navadno deterministični in v celotni opazljivi enostanjski problemi.
- Agentov svet (okolje) je predstavljivo z diskretno množico možnih stanj.
- Možne agentove akcije/operacije, ki spreminjajo njegov svet, so predstavljive z diskretno množico operatorjev.
- Agentov svet je statičen (nespremenljiv) in deterministični-

Tipični primeri reševanja problemov

- Sestavljanje izdelkov v industrijski proizvodnji.
- Optimalna razmestitev gradnikov integriranih sistemov.
- Določitev najkrajše poti pri obiskovanju točk v prostoru.
- Preurejanje prostora v neko končno želeno stanje.
- Določitev niza potez, ki igro pripelje do zmage.
- Dokazovanje izrekov v matematiki, logiki itd.
- …

UVEDBA PROSTORA PROBLEMA

- Prostor problema je abstraktni prostor.
- Prostor problema obsega vsa veljavna stanja problema, ki jih je mogoče tvoriti z vsemi možnimi operatorji na vseh kombinacijah objektov, ki določajo problem.
- Prostor lahko vsebuje eno ali več končnih stanj, ki predstavljajo rešitev problema.
- Iskanje rešitve je iskanje v prostoru problema.
- Iskanje se izvaja z različnimi iskalnimi strategijami.

NAČRTOVANJE REŠEVANJA PROBLEMOV

- Natančna določitev problema – določitev začetnih vhodnih in končnih izhodnih okoliščin, ki bi lahko predstavljale možne rešitve.
- Analiza problema – določitev najbolj pomembnih značilnosti problema, ki vplivajo na izbiro strategije iskanja rešitve
- Določitev in predstavitev celotnega znanja, potrebnega za reševanje problema.
- Izbira najboljše metode reševanja problema glede na predhodne ugotovitve.

Formalna definicija problema

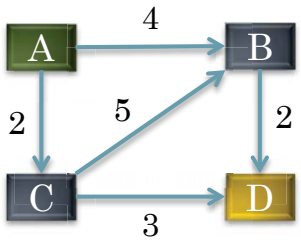
- Problem določajo osnovni elementi in njihove relacije.
- Formalna definicija problema zahteva določitev:
  - prostor stanj z vsemi možnimi stanji problema oziroma postavitvami obravnavanih objektov,
  - določitev ene ali več stanj problema oziroma postavitev objektov, ki predstavljajo možna začetna stanja,
  - določitev ene ali več stanj problema oziroma postavitev objektov, ki predstavljajo možna končna stanja oziroma sprejemljive rešitve problema,
  - določitev množice pravil ki opisujejo možne operacije v prostoru stanj in navadno tudi ceno teh operacij.

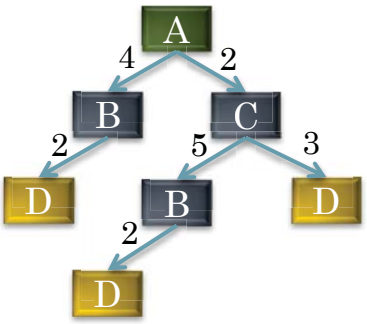
Reševanje problema

- Problem nato rešujemo z uporabo pravil/operacij v skladu z neko primerno strategijo premikanja skozi prostor stanj, dokler ne najdemo poti od začetnega do končnega stanja.
- Temu procesu pravimo iskanje.
- Iskanje je temeljna metoda reševanja problemov, ki niso rešljivi z neko bolj preprosto neposredno metodo.
- V metodo iskanja lahko vedno vgradimo preproste neposredne metode, ki rešujejo bolj preproste podprobleme.
- Večina UI problemov je določeni kot problemi iskanja.

GRAFSKA PREDSTAVITEV PROBLEMA

- Prostor problema navadno predstavimo kot usmerjen graf, kjer vozlišča predstavljajo stanje problema in povezave predstavljajo operatorje, ki stanje problema spremenijo.
- Za poenostavitev iskanja pogosto poskušamo predstaviti prostor problema kot nivojsko drevo.
- Predstavitev z drevesom poenostavi iskanje, vendar za ceno večjega obsega zaradi možnosti ponovitve istih vozlišč.





STRATEGIJA ISKANJA REŠITVE

- Problem rešujemo s sistematskim iskanjem od začetnega do končnega stanja problema v prostoru stanj.
- Razvijamo specifične strategije iskanja za dani problem ali uporabimo splošne strategije iskanja, s katerimi korakoma zmanjšujemo razliko med trenutnim stanjem in končnim ciljnim stanjem problema.
- Strategija iskanja je določena z izbiro vrstnega reda, po katerem se odpira oz. razširja vozlišča grafa/drevesa.
- Strategije iskanja se ovrednoti glede na njihovo popolnost, računsko zahtevnost in optimalnost.

PRIMER FORMULACIJE ENOSTANJSKEGA PROBLEMA

- Problem iskanja optimalne poti formuliramo v štirih točkah:
  - začetno stanje, npr. „*v Ljubljani*“,
  - funkcija naslednik $N(x)$, ki določi naslednja stanja iz prejšnjega stanja, npr. $N(\text{„}v\ Ljubljani\text{“}) = \{\text{„}v\ Domžalah\text{“}, \text{„}v\ Medvodah\text{“}, \text{„}v\ Dobrovi\text{“}, \ldots\}$,
  - preverjanje cilja, ki je lahko eksplicitno, npr. (x == „*v Celju*“) ali implicitno, npr. $NiSmeti(x)$,
  - cene sprememb stanj, npr. ceno spremembe x v y določa $c(x, a, y)$ pri $a \geq 0$, ter kumulativna cena, npr. skupna razdalja v kilometrih ali skupno število izvedenih potrebnih operacij od začetnega do končnega stanja
- Rešitev je zaporedje akcij z najnižjo kumulativno ceno, ki vodi od začetnega do končnega stanja.

PRIMERI ENOSTANJSKIH PROBLEMOV

- Problemi, pri katerih je rešitev opis poti od začetnega do končnega stanja:
  - problem sestavljanja izdelka,
  - problem iskanja poti po zemljevidu,
  - problem sestavljanja sestavljanke (angl. „puzzle“),
  - problem Hanojskega stolpa,
  - problem dveh vrčev z vodo itd.
- Problemi, pri katerih je rešitev samo opis končnega stanja:
  - postavitev elementov (integrirana vezja ipd),
  - ureditev urnika,
  - postavitev osmih kraljic,
  - igra sodoku itd.

Problem Hanojskega stolpa

- Za igro so potrebne tri palice, na katere natikamo diske različnih velikosti. Število diskov je poljubno, vsi pa morajo biti različnega premera.
- Igra se začne tako, da so vsi diski na prvi levi palici in urejeni od vrha do tal v vrstnem redu od najmanjšega do največjega.
- Cilj igre je premakniti stolp diskov s prve na tretjo palico (končni stolp) z najmanjšim možnim številom potez, pri čemer upoštevamo naslednja pravila:
  - naenkrat lahko premaknemo en sam disk na katero koli drugo palico in
  - na vrh manjšega diska ne smemo postaviti na večji disk.

Formulacija problema Hanojskega stolpa

- Problem Hanojskega stolpa rešujemo tako, da uvedemo prostor stanj problema, kjer vsako stanje predstavlja eno možno veljavno razmestitev diskov po palicah.
- Začetno stanje problema potem predstavlja začetna razmestitev (stolp na prvi palici) in končno stanje končna razmestitev diskov (stolp na tretji palici).
- Prostor stanj predstavimo kot usmerjen graf, kjer vozlišča predstavljajo stanja oziroma možno veljavno razmestitev diskov po palicah in povezave predstavljajo operatorje oziroma možne premike diskov v skladu s pravili.

Operatorji problema Hanojskega stolpa

- ***P12***: Premakni disk s palice ena na palico dve!

  Pogoj: palica dve je prazna ali je premer diska na palici dve večji kot je premer diska na palici ena.

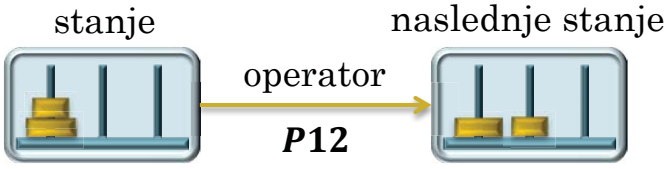
- ***P13***: Premakni disk s palice ena na palico tri!

  Pogoj: palica tri je prazna ali je premer diska na palici tri večji kot je premer diska na palici ena.

- ***P21***: Premakni disk s palice dve na palico ena!

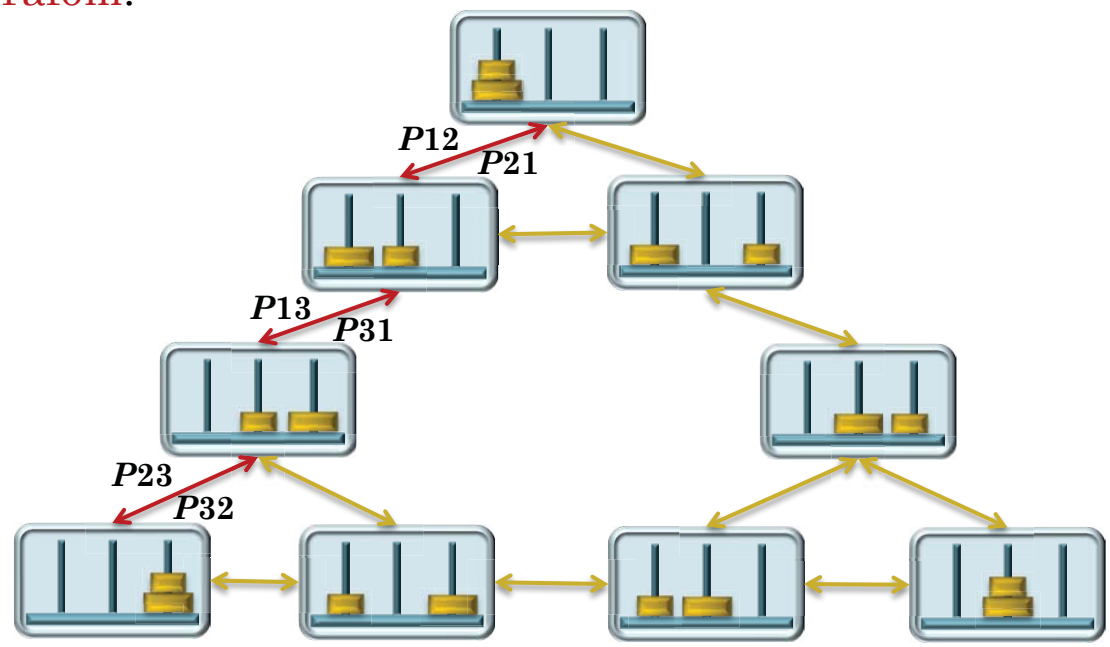
  Pogoj: palica ena je prazna ali je premer diska na palici ena večji kot je premer diska na palici dve.

- ***P23***: …
- …



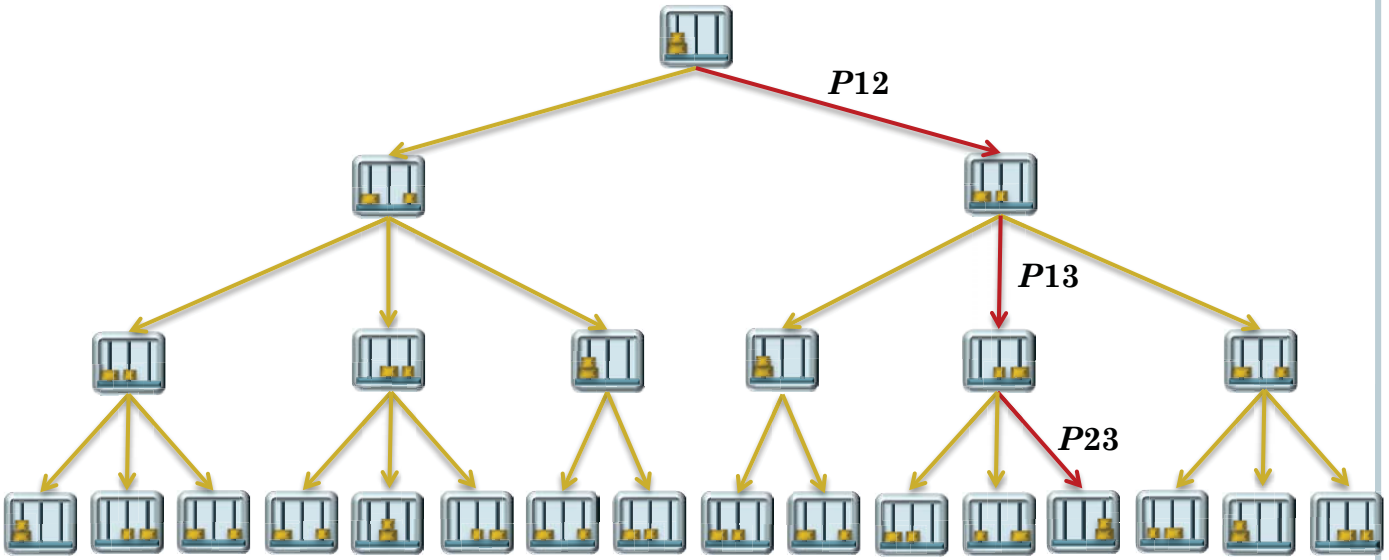
Ponazoritev problema Hanojskega stolpa

- Problema Hanojskega stolpa z dvema diskoma ima devet možnih stanj.
- Prostor stanj lahko ponazorimo z (dvosmernim) usmerjenim grafom.



Drevesna ponazoritev problema

- Prostor stanj lahko ponazorimo tudi z drevesom z določeno globino



Problem Hanojskega stolpa (4/4)

- Pot označena z rdečo označuje rešitev problema.
- Predstavitev z drevesom vsebuje veliko več vozlišč kot z grafom.
- V drevesu se isto stanje lahko pojavi v več vozliščih.
- Drevo je obsežnejše, vendar omogoča njegov zapis v implicitni obliki.
- Implicitna oblika omogoča sprotno ustvarjanje vozlišč med iskanjem poti od začetnega do končnega stanja.

Problem dveh vrčev z vodo

- Na razpolago imamo dva vrča, eden z volumnom treh in drugi z volumnom štirih litrov.
- Vodo lahko v vrča natakamo s pipe, pretakamo z enega v drugi vrč ali jo z vrča zavržemo.
- Cilj je doseči točno dva litra vode v štirilitrskem vrču.

Stanja problema dveh vrčev z vodo

- Stanje problema tokrat določa količina vode v vsakem vrču.
- Stanje predstavimo z dvema realnim spremenljivkama V_3 in V_4, ki označujeta količino vode v enem in drugem vrču.

$$0 \leq V_3 \leq 3\,, \quad 0 \leq V_4 \leq 4$$

- Začetno stanje predstavljata vrednosti

$$V_3 = 0\,, \quad V_4 = 0$$

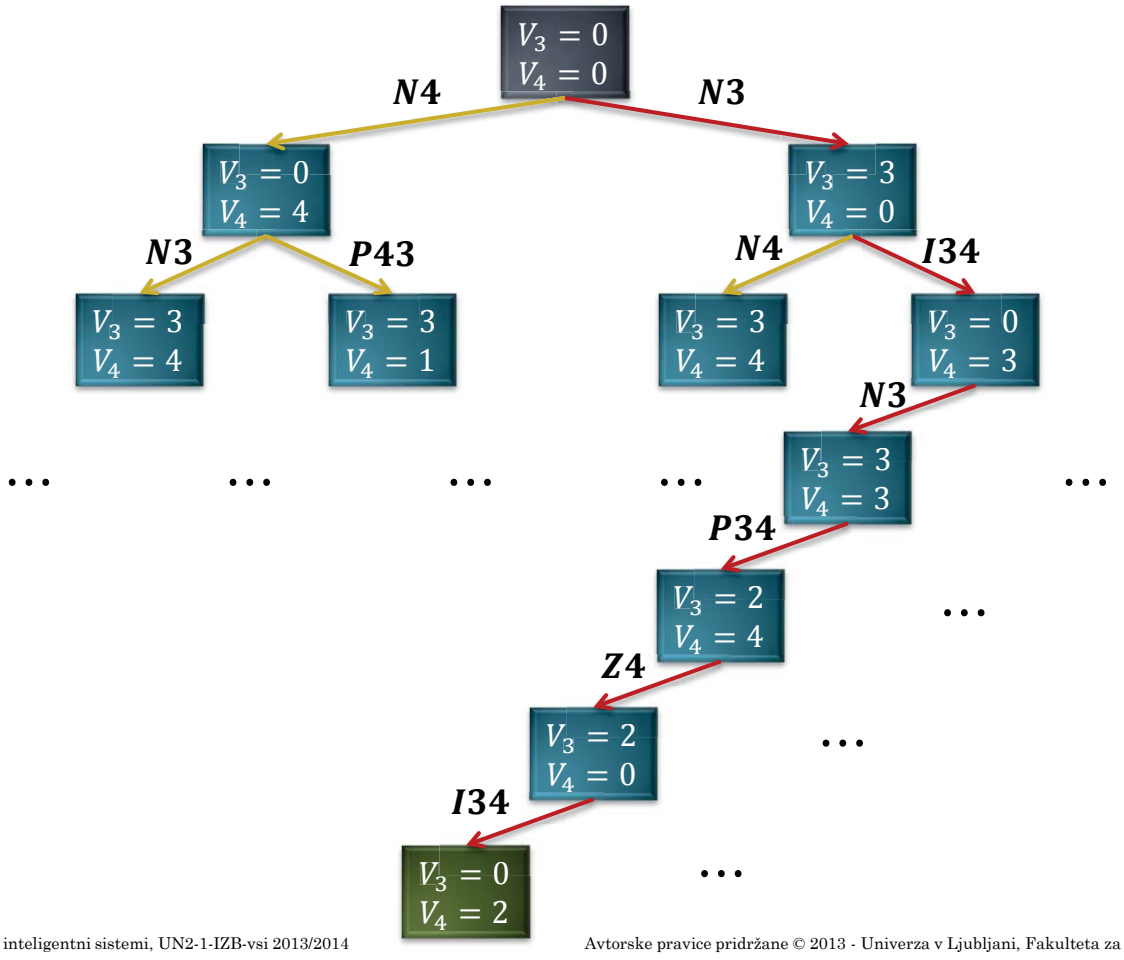
- Končno stanje pa vrednost (vrednost V_3 ni pomembna)

$$V_4 = 2$$

Operatorji problema dveh vrčev z vodo

- ***N4***: Napolni vrč V_4!
  Pogoj: $V_4 < 4$ Posledica: $V_4 = 4$
- ***Z4***: Zlij vso vodo z vrča V_4 po tleh!
  Pogoj: $V_4 > 0$ Posledica: $V_4 = 0$
- ***P43***: Prelij vodo z vrča V_4 v vrč V_3, dokler ta ni poln!
  Pogoj: $V_3 < 3 \wedge V_4 \geq 3 - V_3$ Posledica: $V_3 = 3,\ V_4 = 4 - (3 - V_3)$
- ***P43***: Prelij vodo z vrča V_3 v vrč V_4, dokler ta ni poln!
  Pogoj: $V_4 < 4 \wedge V_3 \geq 4 - V_4$ Posledica: $V_3 = V_3 - (4 - V_4),\ V_4 = 4$
- ***I34***: Izprazni vrč V_3 tako, da ga preliješ v vrč V_4!
  Pogoj: $V_3 + V_4 < 4 \wedge V_3 > 0$ Posledica: $V_3 = 0,\ V_4 = V_3 + V_4$

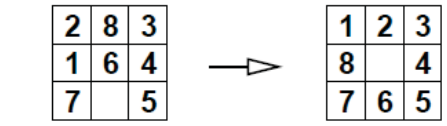
Graf problema dveh vrčev z vodo



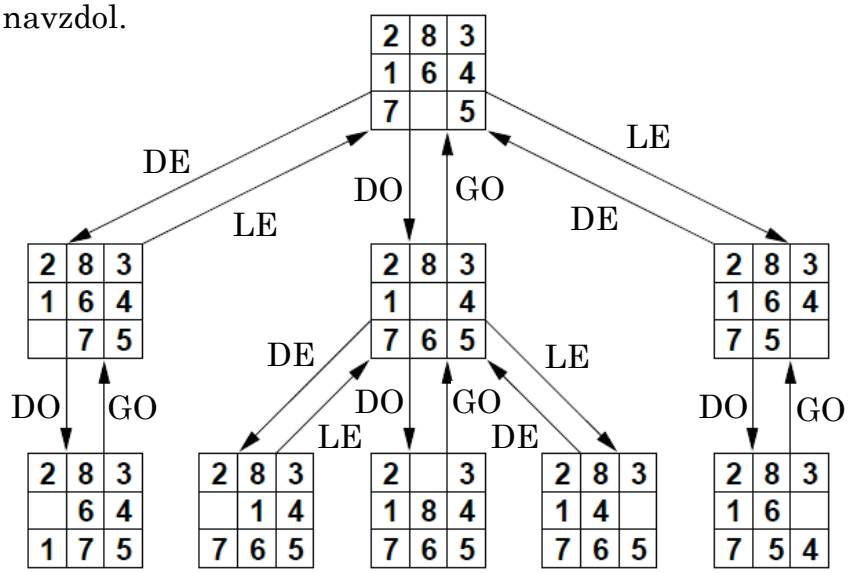
Problem 8-mestne sestavljanke

Operatorji:

- LE: Premik ploščice v levo.
- DE: Premik ploščice v desno
- GO: Premik ploščice navzgor.
- DO: Premik ploščice navzdol.

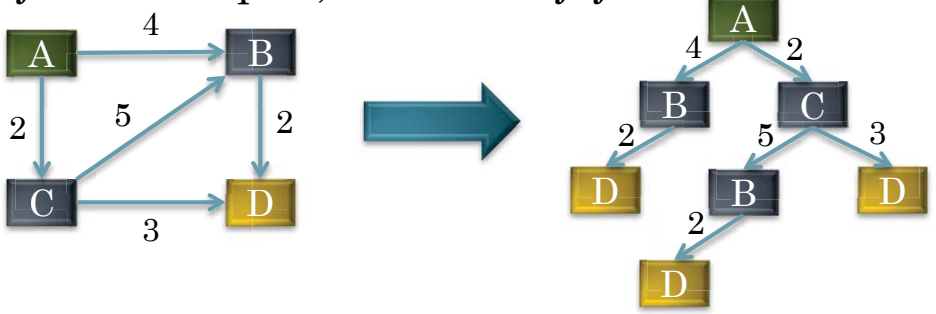


začetno stanje končno stanje



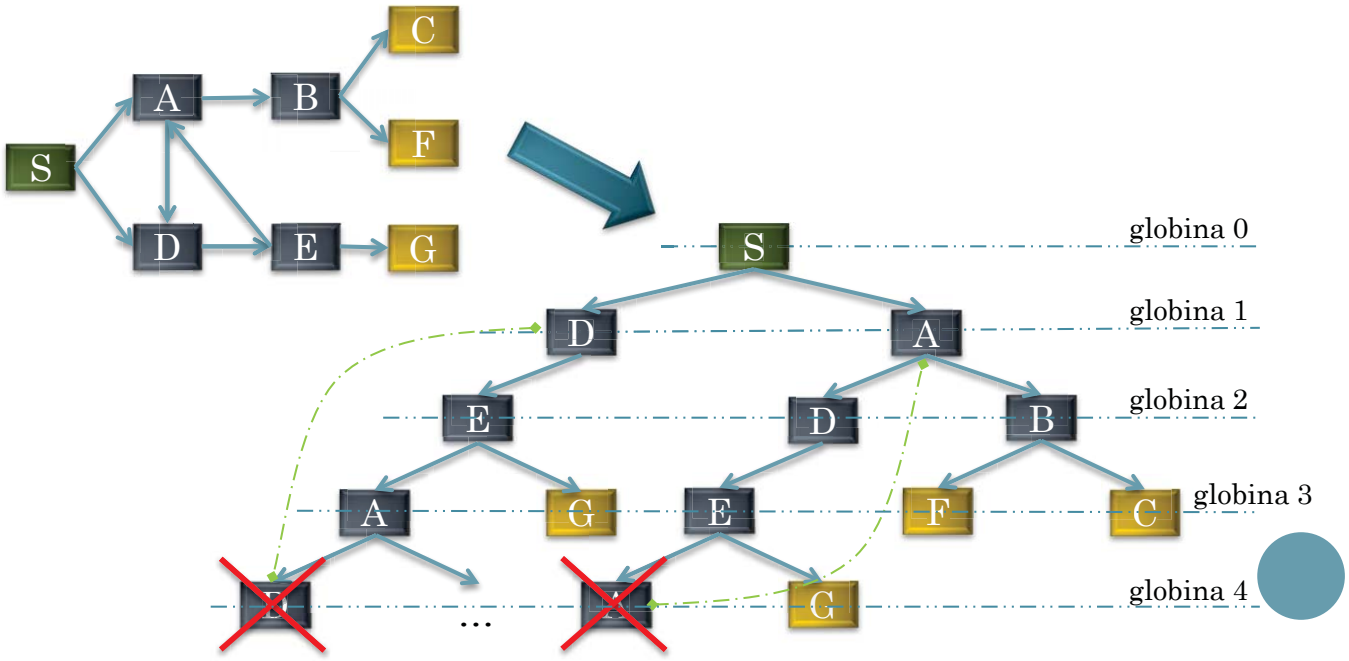
Pretvorba usmerjenih grafov v drevesa

- Množico vseh možnih poti v usmerjenem grafu lahko predstavimo kot drevo.
  - Drevo je usmerjen graf, pri katerem ima vsako vozlišče največ eno predhodno vozlišče.
  - Koren drevesa je vozlišče, ki nima predhodnega vozlišča.
  - List drevesa je vozlišče, ki nima naslednjega vozlišča.
  - Vejitveni faktor drevesa je (povprečno) število naslednikov vozlišča.
- Usmerjeni graf pretvorimo v drevo s podvajanjem vozlišč in prekinitvijo krožnih poti, če te obstajajo.



Pretvorba usmerjenih grafov v drevesa

- Pri pretvorbi izberemo začetno (korensko) vozlišče in sledimo vse poti do končnih vozlišč (listov) oziroma vozlišč, ki so že obiskana (v poti).



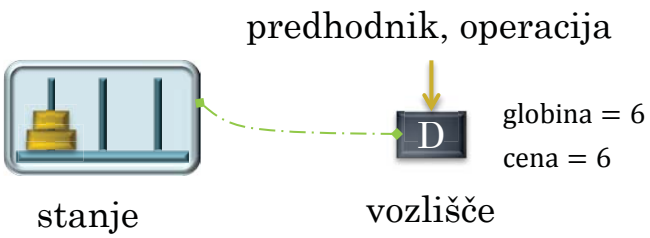
Reševanje problema z iskanjem

- Problem rešujemo s preiskovanjem drevesa in s tvorjenjem naslednikov že raziskanih vozlišč (t.j. razširjanje vozlišč).

```
function iskanje(problem, strategija) returns rešitev ali neuspeh
   inicializiraj iskalno drevo z uporabo začetnega stanja problema
   loop do
      if iskalno drevo ne vsebuje vozlišča za razširjanje
         then return neuspeh
      izberi list v iskalnem drevesu za razširitev v skladu s strategijo
      if izbrano vozlišče predstavlja ciljno stanje
         then return rešitev
      else razširi izbrano vozlišče in dodaj naslednike v iskalno drevo
   end
```

Relacija med stanji in vozlišči

- Stanje je predstavitev (fizikalne) konfiguracije problema
- Vozlišče je podatkovna struktura ki tvori iskalno drevo in vključuje identiteto predhodnega vozlišča vseh naslednjih vozliščih ter podatek o globini in ceni poti.
- Stanja sama po sebi nimajo predhodnikov, naslednikov, globine in cene poti.
- Razširitev vozlišča tvori nova vozlišča z ustreznimi podatki in uporablja funkcijo naslednik $N(x)$ danega problema, ki tvori naslednje stanje iz prejšnjega stanja.



Algoritmi za preiskovanje drevesa

- Enotne strategije iskanja temeljijo na uporabi samo tiste informacije, ki je del definicije problema.
- Iskanje je lahko izčrpno (popolno in optimalno) ali hevristično (ni nujno popolno in optimalno).
- Obstaja vrsta predlaganih strategij iskanja, kot so:
  - Iskanje v širino;
  - Iskanje v globino;
  - Iskanje z enakomerno ceno;
  - Iskanje z omejeno globino;
  - Iskanje z iterativnim poglabljanjem;
  - Iskanje z uporabo hevristične funkcije;
  - …

Ovrednotenje strategije iskanja

- Strategija iskanja je popolna, če z njo zagotovo najdemo rešitev v vseh primerih, ko ta formalno obstaja.
- Strategija iskanja je optimalna, če z njo zagotovo najdemo rešitev z najnižjo možno ceno, če ta formalno obstaja.
- Časovna zahtevnost je sorazmerna številu vozlišč, ki se med iskanjem tvori oziroma odpre.
- Prostorska zahtevnost je sorazmerna številu vozlišč, ki se naenkrat hranijo v pomnilniku med iskanjem

Računska zahtevnost algoritmov

- Računsko zahtevnost strategije iskanja določa časovna in prostorska zahtevnost iskalnega algoritma.
- Teoretično mero časovne ali prostorske zahtevnosti danega algoritma, ki obdela n podatkih označimo z $O(f(n))$.
- Zahtevnost $O(n)$ pomeni, da je časovna ali prostorska zahtevnost sorazmerna količini vhodni podatkov.
- Čas t, potreben za izvedbo algoritma, je v tem primeru $t \propto n$ oziroma $t = kn$, kjer je k neka nenegativna konstanta.
- Podobno je lahko zahtevnost algoritma $O(n^2)$, $O(2^n)$, $O(\log_2(n))$, $O(n \log_2(n))$ itd.

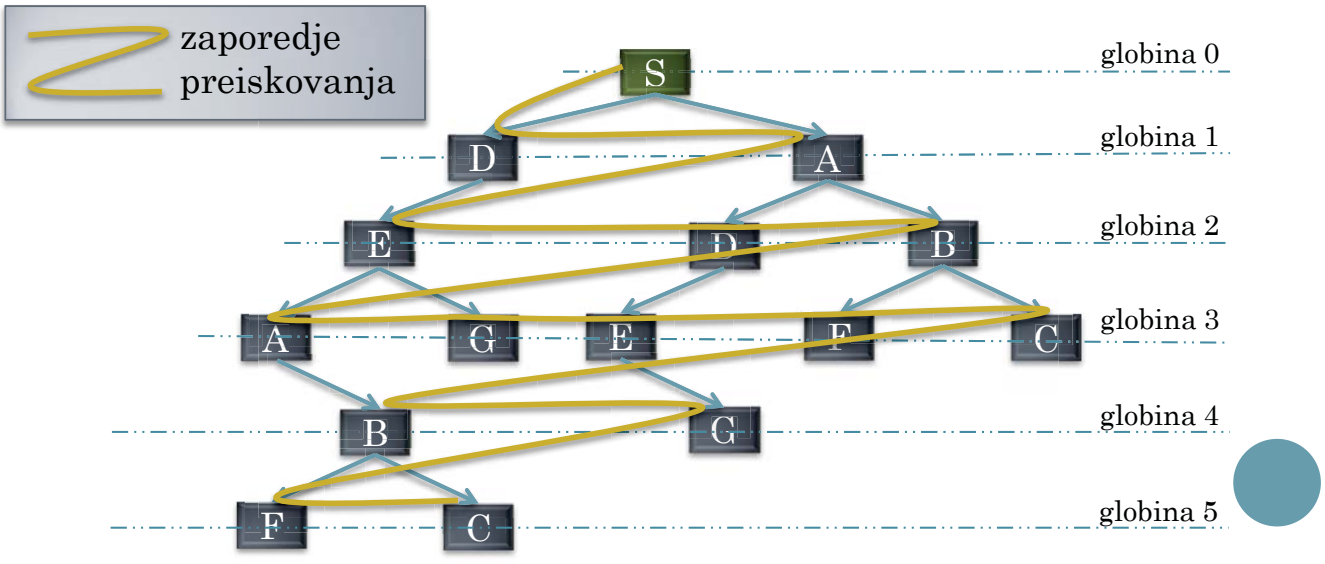
Primer merjenja računske zahtevnosti

Ukazi	Št. operacij
`min = a[0][0]`	1
`for(i=0 ; i<n ; i++)`	$1 + 2n$
`  for(j=0 ; j<n ; j++)`	$n * (1 + 2n)$
`  if(min > a[i][j])`	$n * n$
`  min = a[i][j]`	$< n * n$
`print min`	1

- Skupno število operacij je tako nekje med $3n^2 + 3n + 3$ in $4n^2 + 3n + 3$.
- Konstant navadno ne upoštevamo in ugotovimo, da je zahtevnost algoritma $O(n^2)$, torej sorazmerna kvadratu dimenzije matrike.

Strategija iskanja v širino

- Temelji na načelu – *„Preišči in razširi najbolj plitvo vozlišče!“*
- Izvedba z uporabo FIFO vrste pri vodenju seznama odprtih vozlišč, t.j. na novo tvorjeni nasledniki gredo na konec vrste.

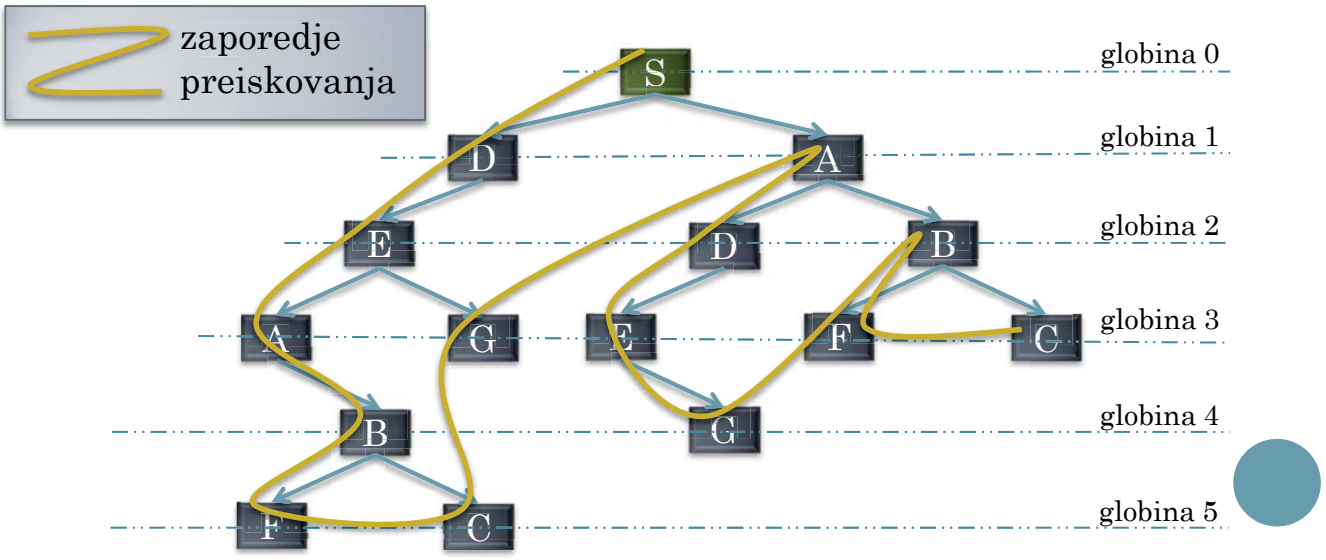


Lastnosti strategije iskanja v širino

- Naj b označuje največji vejitveni faktor drevesa, d globino končnega vozlišča z najnižjo ceno poti in m največjo možno globino prostora stanj (ta je lahko tudi ∞).
- Strategija je popolna, če je b končen.
- Časovna zahtevnost algoritma je $O(b^{d+1})$.
- Prostorska zahtevnost algoritma je prav tako $O(b^{d+1})$.
- Strategija je optimalna, če so cene vseh operacij enak 1.
- Največja pomanjkljivost je prostorska zahtevnost algoritma.

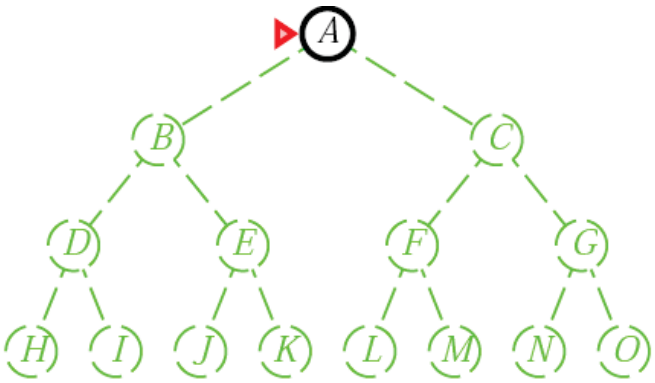
Strategija iskanja v globino

- Temelji na načelu – *„Preišči in razširi najgloblje vozlišče!“*
- Izvedba z uporabo LIFO vrste pri vodenju seznama odprtih vozlišč, t.j. na novo tvorjeni nasledniki gredo na začetek vrste.



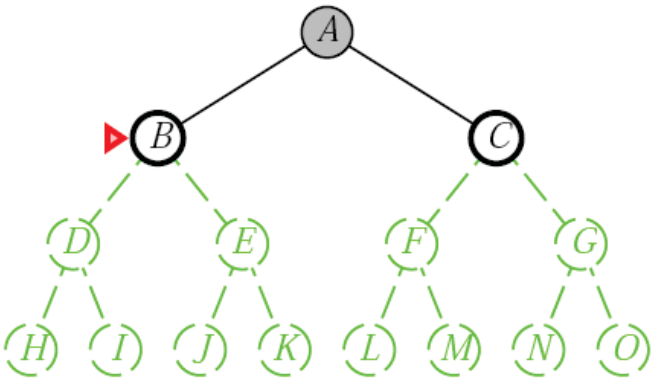
SIMULACIJA ISKANJA V GLOBINO

- Simulacija iskanja s ponazoritvijo preiskovanih, razširjenih in zaprtih vozlišč



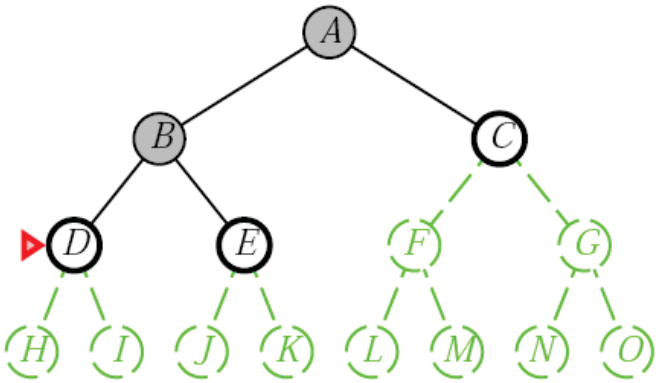
SIMULACIJA ISKANJA V GLOBINO

- Simulacija iskanja s ponazoritvijo preiskovanih, razširjenih in zaprtih vozlišč



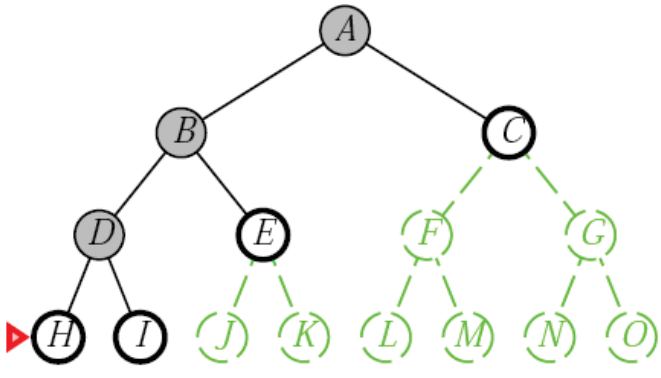
SIMULACIJA ISKANJA V GLOBINO

- Simulacija iskanja s ponazoritvijo preiskovanih, razširjenih in zaprtih vozlišč



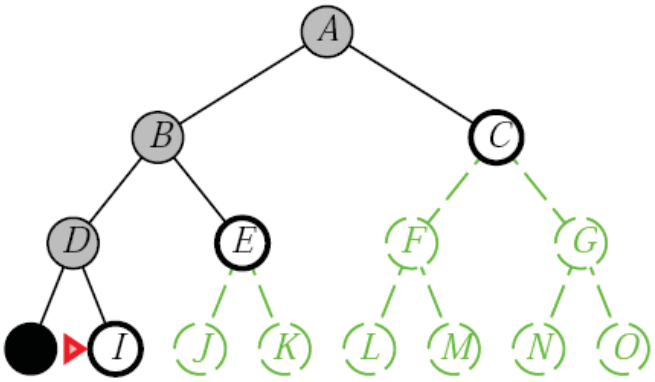
SIMULACIJA ISKANJA V GLOBINO

- Simulacija iskanja s ponazoritvijo preiskovanih, razširjenih in zaprtih vozlišč



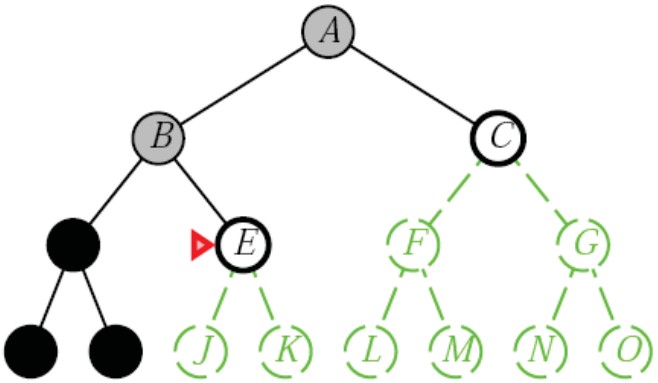
SIMULACIJA ISKANJA V GLOBINO

- Simulacija iskanja s ponazoritvijo preiskovanih, razširjenih in zaprtih vozlišč



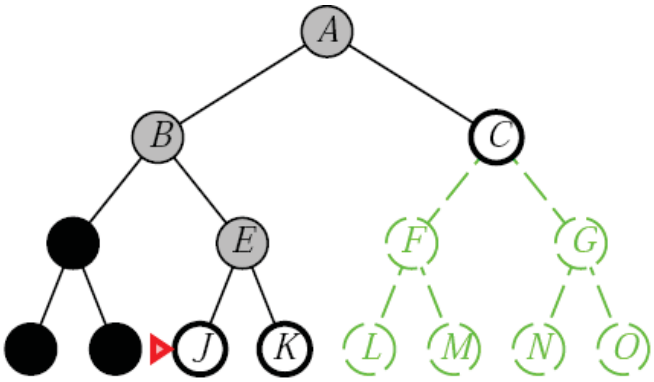
SIMULACIJA ISKANJA V GLOBINO

- Simulacija iskanja s ponazoritvijo preiskovanih, razširjenih in zaprtih vozlišč



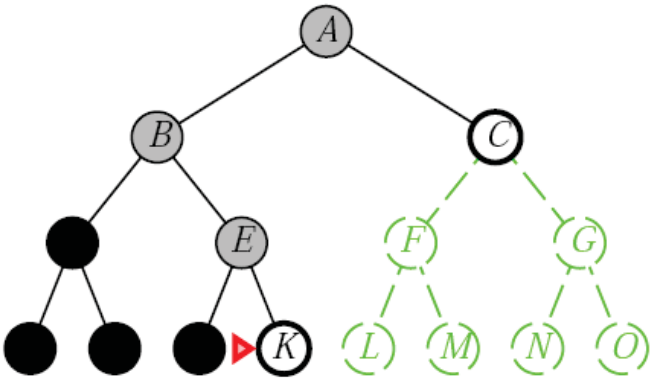
SIMULACIJA ISKANJA V GLOBINO

- Simulacija iskanja s ponazoritvijo preiskovanih, razširjenih in zaprtih vozlišč



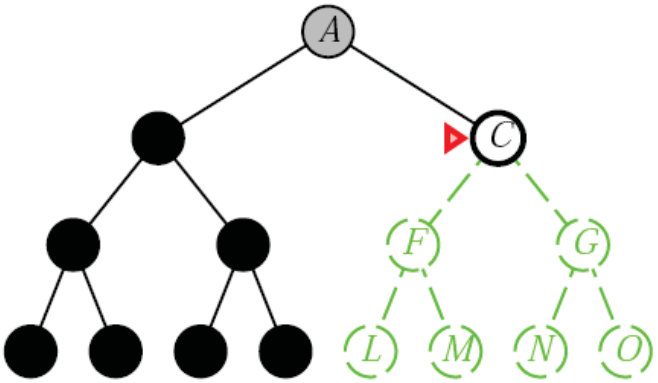
SIMULACIJA ISKANJA V GLOBINO

- Simulacija iskanja s ponazoritvijo preiskovanih, razširjenih in zaprtih vozlišč



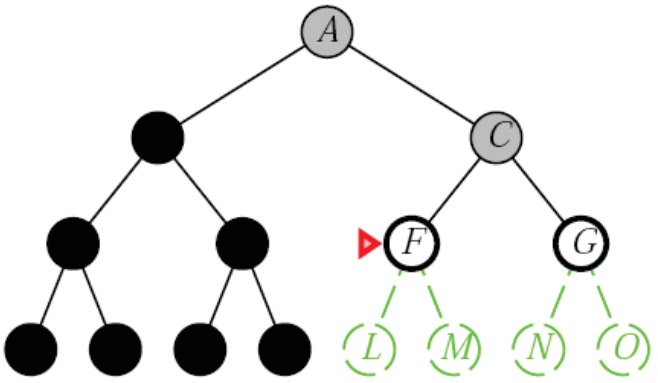
SIMULACIJA ISKANJA V GLOBINO

- Simulacija iskanja s ponazoritvijo preiskovanih, razširjenih in zaprtih vozlišč



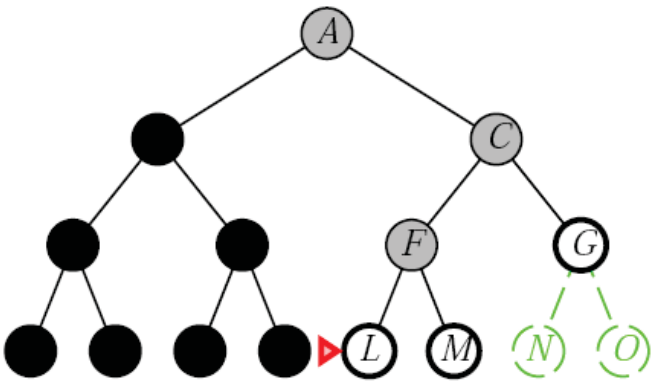
SIMULACIJA ISKANJA V GLOBINO

- Simulacija iskanja s ponazoritvijo preiskovanih, razširjenih in zaprtih vozlišč



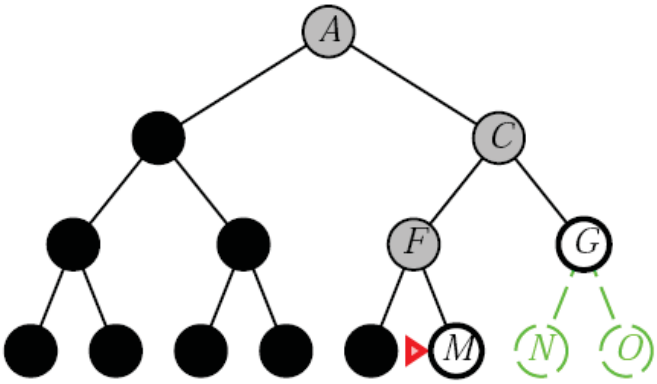
SIMULACIJA ISKANJA V GLOBINO

- Simulacija iskanja s ponazoritvijo preiskovanih, razširjenih in zaprtih vozlišč



SIMULACIJA ISKANJA V GLOBINO

- Simulacija iskanja s ponazoritvijo preiskovanih, razširjenih in zaprtih vozlišč



Lastnosti strategije iskanja v globino

- Naj b označuje največji vejitveni faktor drevesa, d globino ciljnega vozlišča z najnižjo ceno poti in m največjo možno globino prostora stanj (ta je lahko tudi ∞).
- Strategija je popolna, če je m končen.
- Časovna zahtevnost algoritma je $O(b^m)$.
- Prostorska zahtevnost algoritma je O(bm).
- Strategija ni optimalna niti pri cenah operacij 1 .
- Največja pomanjkljivost je časovna zahtevnost pri $b \gg d$.

Strategija iskanja z enakomerno ceno

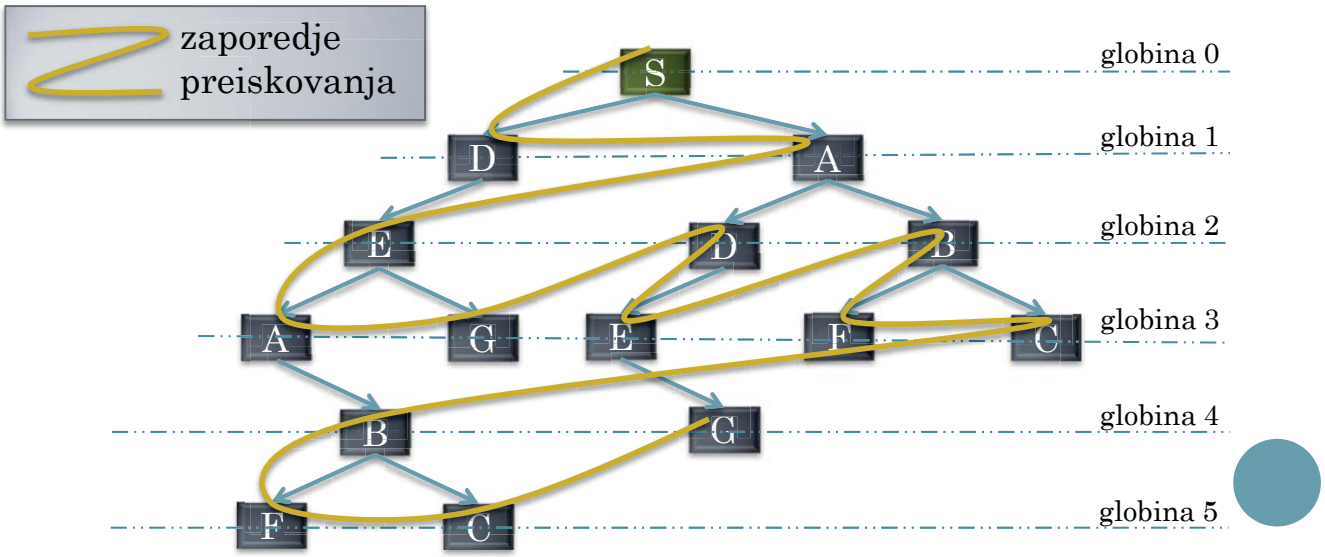
- Temelji na načelu – *„Preišči in razširi vozlišče z najnižjo ceno poti!“*
- Izvedba z urejanjem seznama odprtih vozlišč glede na ceno poti.
- Pri konstantni ceni operacij je ta strategija ekvivalentna strategiji iskanja v širino.
- Strategija je popolna in optimalna.
- Prostorska in časovna zahtevnost je sorazmerna številu vmesnih vozlišč s ceno poti, ki je nižja od cene najcenejše poti do nekega končnega vozlišča.

STRATEGIJA ISKANJA Z OMEJENO GLOBINO

- Temelji na strategiji iskanja v globino z omejevanjem največje globine iskanja.
- Če je največja preiskana globina večja od globine končnega vozlišča z najnižjo ceno poti, $l \geq d$, je ta strategija popolna.
- Časovna zahtevnost algoritma je $O(b^l)$.
- Prostorska zahtevnost algoritma je $O(bl)$.
- Strategija ni optimalna niti pri cenah operacij 1.
- Slabost strategije je problem izbire primerne globine l, saj d navadno ni vnaprej znan.

STRATEGIJA ISKANJA S POGLABLJANJEM

- Temelji na iterativnem spreminjanju največje globine l pri strategiji iskanja z omejeno globino.
- Spodaj je ponazorjeno iskanje s poglabljanjem pri $l = 1$.



STRATEGIJA ISKANJA S POGLABLJANJEM

- Strategija je popolna.
- Časovna zahtevnost algoritma je $O(b^d)$.
- Prostorska zahtevnost algoritma je $O(bd)$.
- Strategija je optimalna pri cenah operacij 1.
- Strategija se lahko prilagodi tudi za iskanje z enakomerno ceno.

STRATEGIJA ISKANJA S HEVRISTIČNO FUNKCIJO

- Temelji na načelu – *„Preišči in razširi vozlišče z najboljšim ocenjenim obetom za dosego končnega stanja!“*
- Tej strategiji iskanja pravimo tudi požrešno iskanje.
- Izvedba z uvedbo hevristične funkcije $h(n)$, ki za dano stanje n podaja hevristično oceno cene poti do najbližjega končnega stanja, ki predstavlja rešitev problema.
- Strategija ni popolna.
- Časovna zahtevnost algoritma je $O(b^m)$, vendar se lahko precej izboljša z dobro izbrano hevristično funkcijo.
- Prostorska zahtevnost algoritma je $O(b^m)$, vendar se lahko precej izboljša z dobro izbrano hevristično funkcijo
- Strategija ni optimalna niti pri cenah operacij 1.

ALGORITEM ISKANJA A*

- Temelji na načelu – *„Preišči in razširi vozlišče z najnižjo ceno že opravljene poti in tudi najboljšim obetom za dosego končnega stanja!“*
- Kombinacija cene $g(n)$ že opravljene poti do danega stanja n ter hevristične funkcije $h(n)$, ki za dano stanje n podaja oceno cene poti do najbližjega končnega stanja.
- Izvedba z urejanjem seznama odprtih vozlišč glede na skupno oceno $f(n) = g(n) + h(n)$ cene poti od začetnega do končnega stanja.
- Za optimalnost algoritma mora hevristična funkcija $h(n)$ izpolnjevati nekaj pogojev.

LASTNOSTI ALGORITMA A*

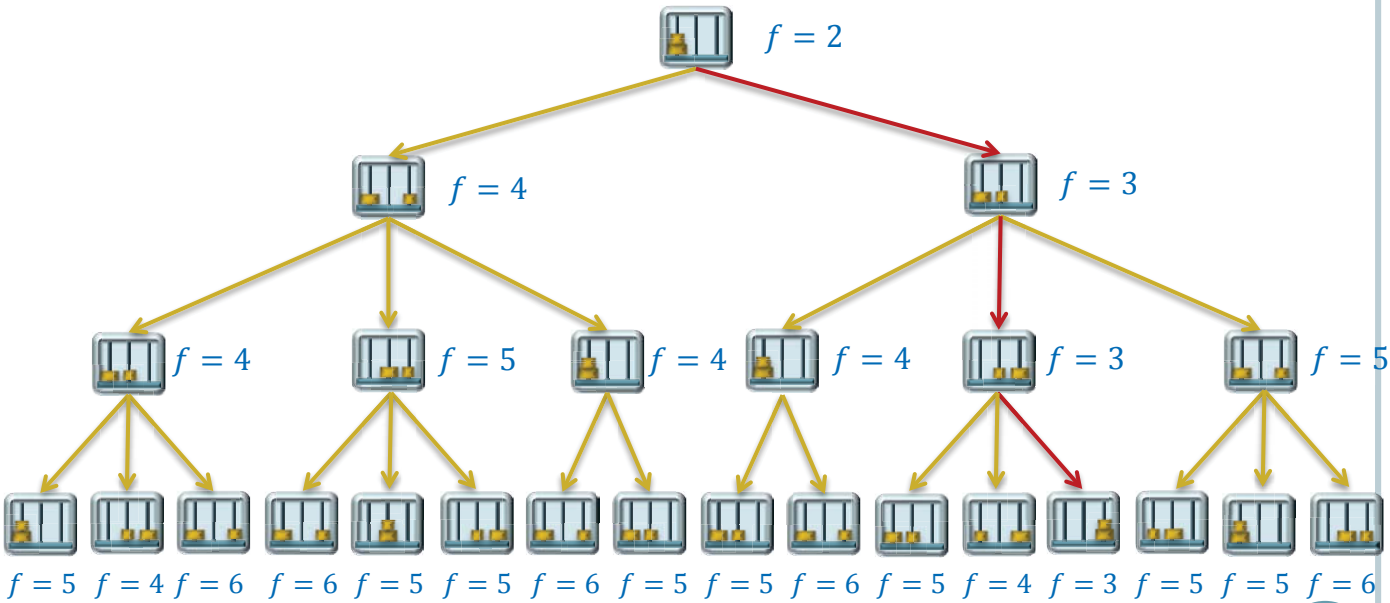
- Hevristična funkcija $h(n)$ mora podajati vrednost ocene cene poti od danega do končnega stanja, ki je nižja ali enaka dejanski ceni te poti, torej $h(n) \leq h^*(n)$, kjer je $h^*(n)$ dejanska cena.
- Hevristična funkcija $h(n)$ mora biti nenegativna, torej $h(n) \geq 0$, in njena vrednost za končna stanja mora biti nič, torej $h(G) = 0$, pri čemer G označuje končno stanje.
- Strategija je popolna.
- Časovna in prostorska zahtevnost algoritma sta $O(b^m)$, vendar se lahko precej izboljšata z dobro izbrano hevristično funkcijo.
- Strategija je optimalna, če hevristična funkcija izpolnjuje navedene pogoje.

Primeri hevrističnih funkcij

- Pri iskanju cestne poti na zemljevidu je hevristična funkcija navadno zračna razdalja do ciljne točke.
- Pri problemu Hanojskega stolpa je hevristična funkcija obratno sorazmerna številu in premeru diskov na tretji palici.
- Pri več mestni sestavljanki je hevristična funkcija sorazmerna številu ploščic na napačni lokaciji.
- Pri industrijskem sestavljanju izdelkov je hevristična funkcija sorazmerna številu manjkajočih sestavin izdelka.
- …

Zgled s problemom Hanojskega stolpa

- Vrednosti hevrističnih ocen cen celotnih poti od začetnega do končnega stanja v drevesu, ki ponazarja dani problem



VPRAŠANJA

- Kaj navadno označuje problem?
- Katere vrste problemov obravnavamo na področju UIS?
- Kateri so tipični primeri reševanja problemov?
- Kako formalno definiramo problem?
- Na kakšne način z iskanjem rešujemo probleme?
- Kateri so primeri enostanjskih problemov?
- Kako pretvorimo usmerjeni graf v drevo?
- Katere strategije preiskovanja drevesa problema poznamo?
- Kako vrednotimo strategije iskanja?

REŠEVANJE PROBLEMOV Z NJEGOVO RAZGRADNJO NA PODPROBLEME

- Poleg uporabe hevrističnih funkcij probleme rešujemo z uporabo znanja o sami strukturi problema.
- Težje probleme tako inteligentno rešujemo z njegovo razgradnjo na lažje podprobleme.
- Podproblemi so vedno lažje rešljivi kot celotni problem.
- Rešitev celotnega problema tvorimo s sestavljanjem rešitev lažjih podproblemov v načrt reševanja.
- Razgradnjo problema predstavimo z IN/ALI grafi oziroma drevesi.

Razgradnja problema na podprobleme 1/1

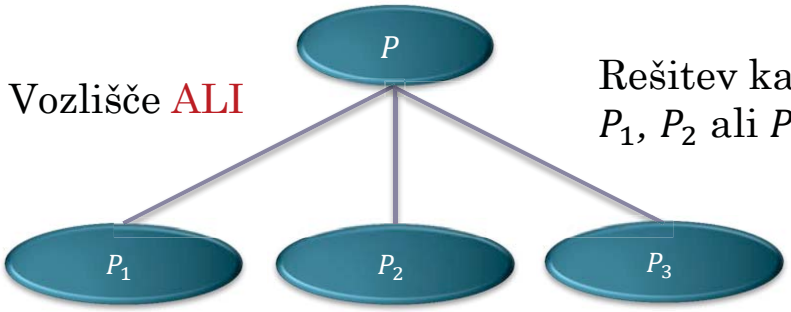
- Če dan problem P_i ni rešljiv z eno operacijo, ga razgradimo na podprobleme.
- Razgradnjo nadaljujemo toliko časa, dokler ne naletimo na podprobleme, ki so rešljivi z eno operacijo (to so osnovni problemi).
- Razgradnjo problema na podprobleme predstavimo z grafom oz. drevesom IN/ALI.
- Listi drevesa IN/ALI so vedno osnovni podproblemi, ki so trivialno rešljivi.

Razgradnja problema na podprobleme 2/2

- Razgrajeni problem rešujemo tako, da najprej rešimo osnovne probleme po nekem načrtu reševanja.
- Nato rešujemo podprobleme v najglobljih vmesnih vozliščih drevesa IN/ALI v skladu z načrtom reševanja vse do korena drevesa, ki predstavlja rešitev celotnega problema.
- Če drevo IN/ALI vsebuje le vozlišča IN, je možna le ena rešitev problema, sicer jih je več.
- Rešitev problema ni več pot, ampak vsako poddrevo drevesa IN/ALI, ki vsebuje le vozlišča IN.
- Podproblemi vozlišč IN naj bi bili med sabo neodvisni (ni pomemben vrstni red njihovega reševanja)

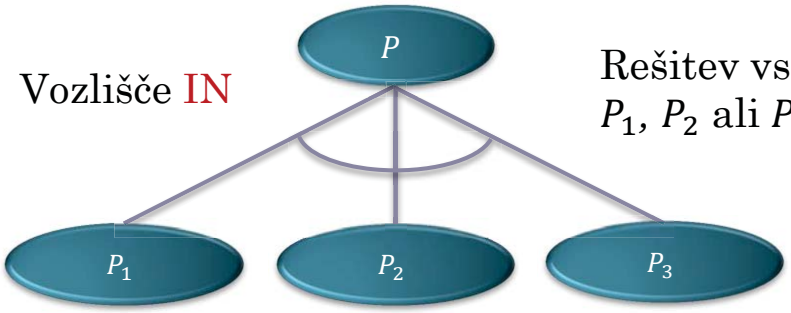
VOZLIŠČA IN/ALI GRAFOV OZ. DREVES

Vozlišče ALI



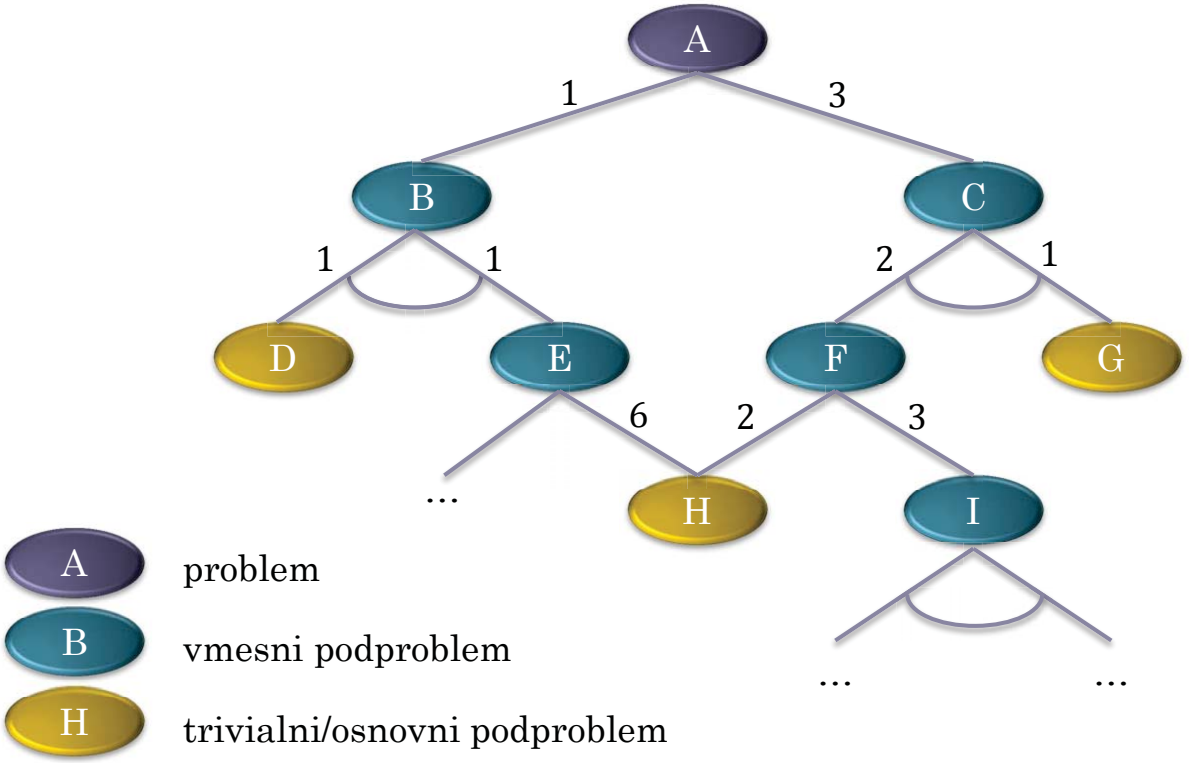
Rešitev kateregakoli podproblema P_1, P_2 ali P_3 reši problem P

Vozlišče IN

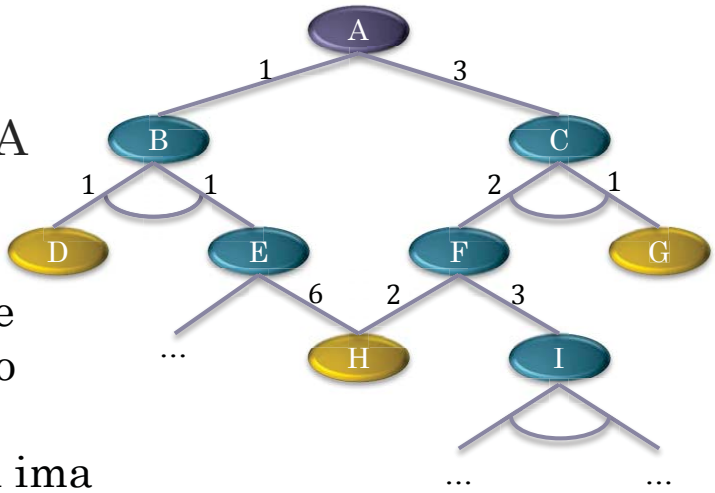


Rešitev vseh podproblemov P_1, P_2 ali P_3 reši problem P

UVAJANJE CEN RELACIJ MED (POD)PROBLEMI

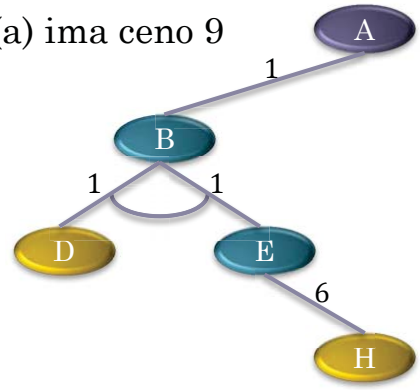


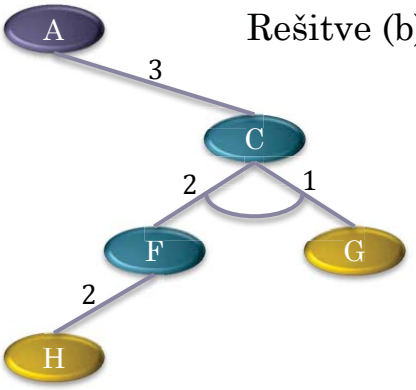
Rešitev je načrt reševanja problema



- Končna rešitev problema je načrt reševanja, ki je vsako poddrevo grafa IN/ALI, ki vsebuje le vozlišča IN in ima liste s trivialnimi podproblemi

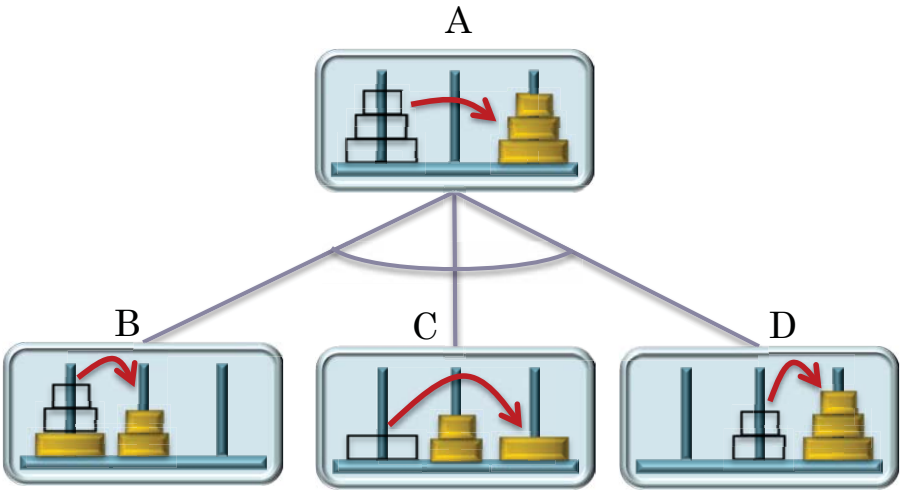
Rešitve (a) ima ceno 9





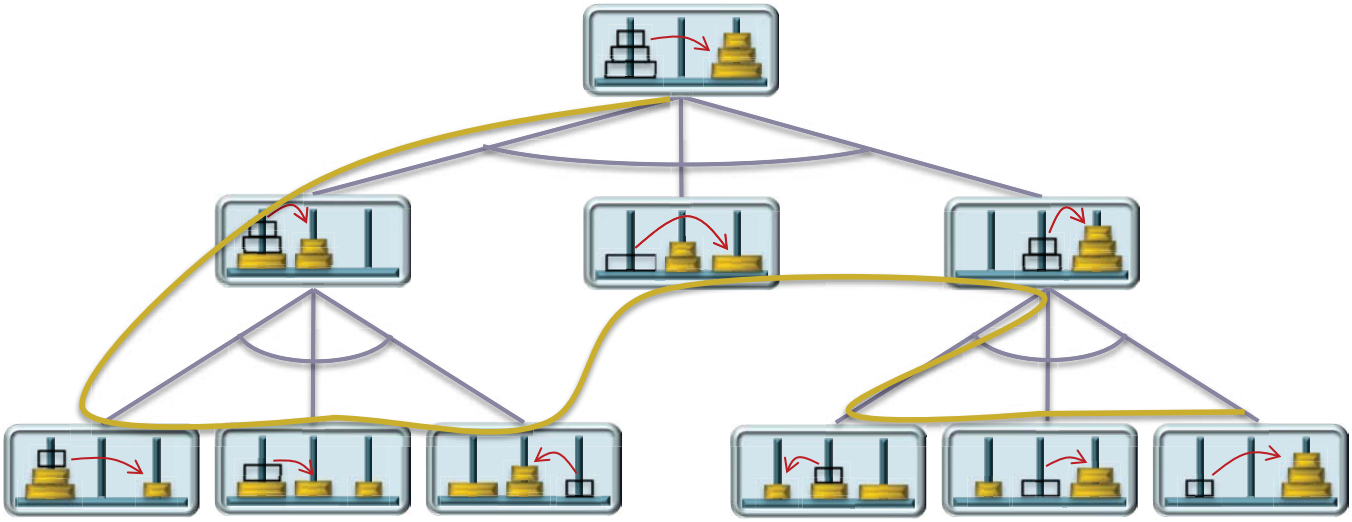
Rešitve (b) ima ceno 8

Zgled problema Hanojskega stolpa 1/4



- Problem A razgradimo na tri podprobleme B, C in D.
- Podproblem C je trivialen podproblema A in D pa je potrebno ponovno razgraditi.
- Podproblemi B, C in D niso neodvisni ampak se morajo reševati po vrsti od leve proti desni.

Zgled problema Hanojskega stolpa 2/4



- Drevo tokrat vsebuje samo vozlišča IN in zato celo drevo že predstavlja tudi en sam možen načrt reševanja problema .
- Načrt reševanja izvajamo podobno kot izvajamo preiskovanje drevesa v globino.

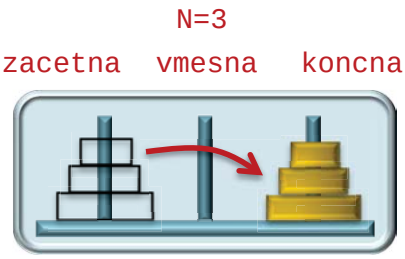
Zgled problema Hanojskega stolpa 3/4

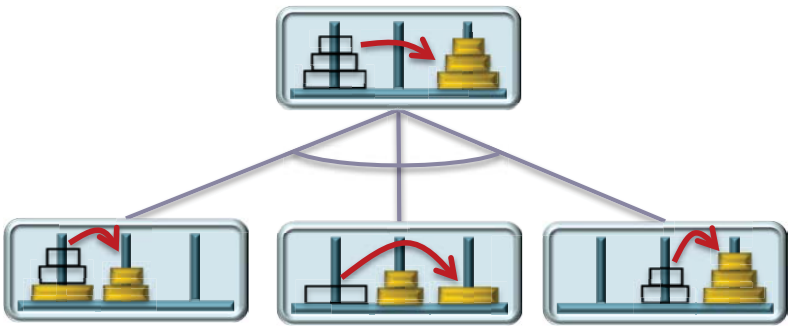
- Pri tem zgledu se razgradnjo vmesnih (pod)problemov vedno izvede na tri podprobleme, ki jih lahko opišemo:
  - Premakni N-1 diskov stolpa z dane začetne na dano začasno vmesno palico.
  - Premakni zadnji največji disk z dane začetne na dano končno palico (trivialni/osnovni problem).
  - Premakni N-1 diskov stolpa z dane začasne palice na končno palico (kjer je že največji disk).
- Če je na dani začetni palici samo še en disk, ga samo še premaknem na dano končno palico (trivialni problem)

Zgled problema Hanojskega stolpa 3/4

- Opisano razgradnjo lahko izvedemo z rekurzivnim klicem funkcije `resi`, ki razgradi dani problem na tri podprobleme.

```
function resi (N, zacetna, vmesna, koncna)
   if N = 0
      return
   else
      resi (N-1,zacetna, koncna, vmesna)
      premakni_disk (zacetna, koncna)
      resi (N-1,vmesna, zacetna, koncna)
   endif
end
```





Iskanje optimalnega načrta reševanja problema 1/2

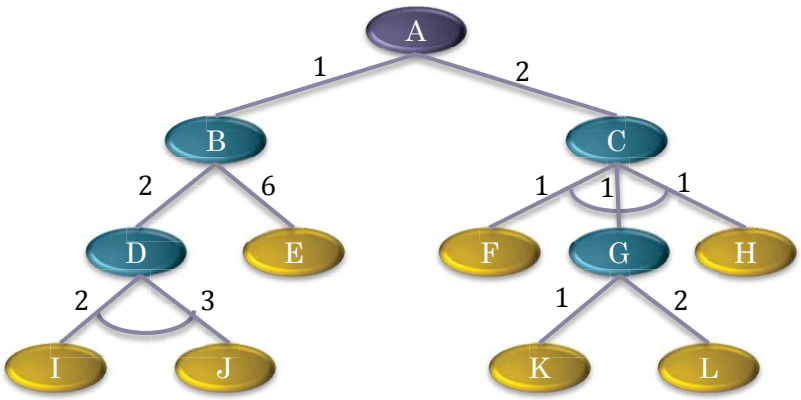
- V primeru, da predstavimo razgradnjo problema na podprobleme z grafom, ki vsebuje tudi vozlišča ALI obstaja več rešitev oziroma načrtov reševanja.
- Optimalni načrt iščemo s strategijami iskanja po drevesu IN/ALI, ki je podobno strategijam iskanja po drevesih pri predstavitvah reševanja s prostorom stanj problema.
- Iskanje pričnemo v korenu drevesa, ki predstavlja celoten problem in nadaljujemo do listov drevesa, ki predstavljajo trivialne probleme.
- Med preiskovanjem razširjamo vozlišča in obiščemo vse njihove naslednike pri vozliščih IN in vsaj enega naslednika pri vozliščih ALI.

Iskanje optimalnega načrta reševanja problema 2/2

- Dano vozlišče se šteje za rešeno:
  - če je to vozlišče ALI in je rešen vsaj en naslednik tega vozlišča;
  - če je to vozlišče IN in so rešeni vsi nasledniki tega vozlišča;
  - če je je vozlišče osnovno in je rešen pripisan trivialni problem.
- Pri vozliščih ALI je pomembna strategija izbire naslednika.
- Ko na ta način rešimo vsa odprta vozlišča, smo rešili celoten problem in pridobili drevo, ki predstavlja enega od možnih načrtov reševanja.
- V iskanje je zato potrebno vključiti cene, na podlagi katerih lahko ocenimo, kateri načrt je boljši ali slabši.

Zgled iskanja načrtov reševanja 1/2

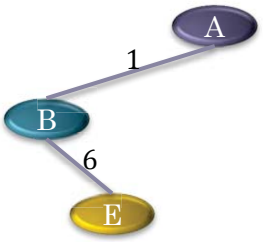
- Vzemimo, da je razgradnja problema predstavljena s spodaj podanim IN/ALI drevesom.



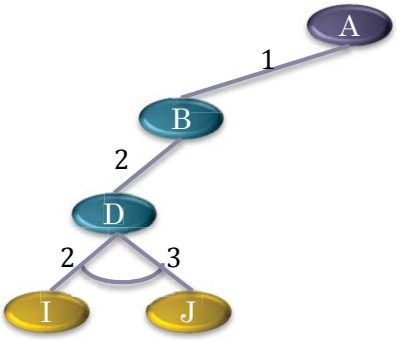
- Ugotovimo lahko, da obstaja štiri rešitve oz. načrti reševanja celotnega problema.

ZGLED ISKANJA NAČRTOV REŠEVANJA 1/2

1)

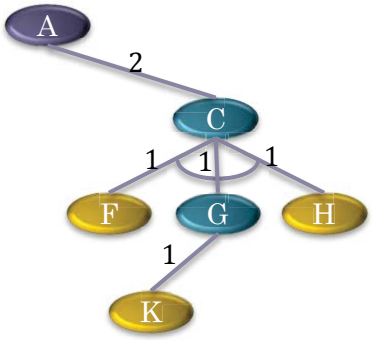


2)

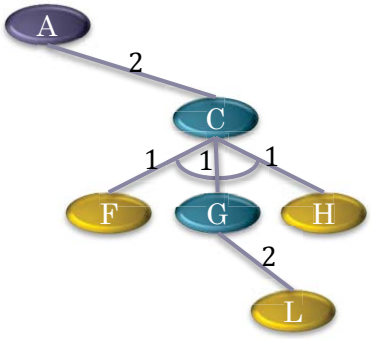


Načrt/drevo z najnižjo ceno

3)



4)



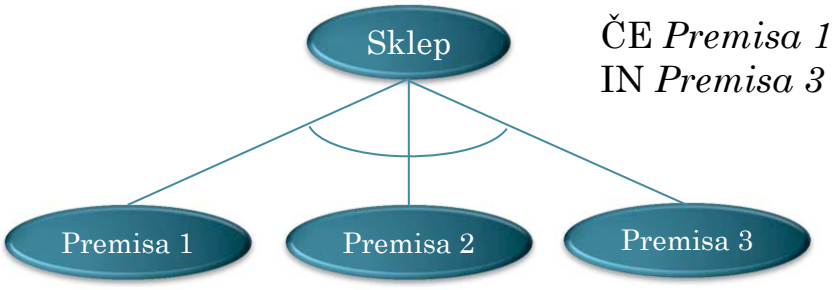
STRATEGIJE PREISKOVANJA IN/ALI DREVES

- Strategija iskanja temelji na načelu - *„Preišči in razširi vozlišče z najnižjo ocenjeno skupno ceno reševanja“.*
- Uporabljamo algoritem AO* (t.j. posplošitev algoritma A* za IN/ALI drevesa – torej, ne le ALI drevesa, kjer je rešitev pot in ne drevo).
- Vzemimo, da $\tilde{h}\,(P)$ označuje hevristično oceno dejanske cene $h(P)$ reševanja vozlišča (problema).
- V listih drevesa je $\tilde{h}\,(P)$ enaka ceni reševanja predhodnega vozlišča.
- Pri ALI vozliščih je $\tilde{h}\,(P) = \min_i \{c(P, P_i) + \tilde{h}(P_i)\}$, kjer $c(P, P_i)$ označuje ceno odpiranja podproblema.
- Pri IN vozliščih je $\tilde{h}\,(P) = \sum_i c(P, P_i) + \tilde{h}(P_i)$.

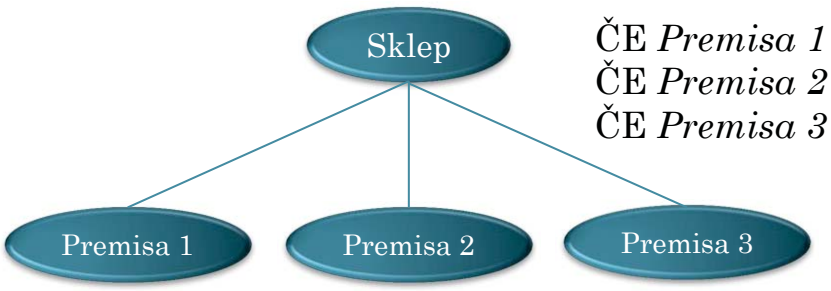
Širša uporaba IN/ALI dreves

- Drevesa, ki predstavljajo prostor stanj problema lahko obravnavamo kot drevesa s samimi ALI vozlišči.
- Pri drevesih s samimi ALI vozlišči je rešitev problema pot od začetnega do končnega vozlišča.
- Drevesa IN/ALI so posplošena drevesa in se uporabljajo pri
  - reševanju problema z razgradnjo problema na podprobleme;
  - predstavitvi znanja s produkcijskimi (ČE … POTEM) pravili;
  - izvajanju sklepanja iz pravil in podanih dejstev (veriženje pravil naprej ali veriženje pravil nazaj).
  - …

Predstavitev produkcijskih pravil z IN/ALI drevesi



ČE *Premisa 1* IN *Premisa 2*
IN *Premisa 3* POTEM *Sklep*



ČE *Premisa 1* POTEM *Sklep*
ČE *Premisa 2* POTEM *Sklep*
ČE *Premisa 3* POTEM *Sklep*

VPRAŠANJA

- Na kakšne način predstavimo razgradnjo problema na podprobleme?
- Kaj predstavljajo vozlišča v IN/ALI grafih oziroma drevesih.
- Kaj predstavlja rešitev problema, ki je razgrajen na podprobleme?
- Koliko rešitev problema vsebuje drevo s samimi IN vozlišči?
- Opišite zgled razgradnje problema Honojskega stolpa na podprobleme.
- Kako z IN/ALI drevesi predstavimo produkcijska pravila.

Umetni inteligentni sistemi, UN2-1-IZB-vsi 2013/2014
Avtorske pravice pridržane © 2013 - Univerza v Ljubljani, Fakulteta za elektrotehniko

EKSPERTNI SISTEMI

VSEBINA

- Uvod
- Zgradba ekspertnih sistemov
- Baza znanja
- Mehanizem sklepanja
- Orodja za izgradnjo ekspertnih sistemov

KAJ SO EKSPERTNI SISTEMI?

- Poskušajo posnemati človeške miselne procese.
- So osredotočeni na uporabo strokovnega znanja.
- Temeljijo na zamisli produkcijskih sistemov.
- Vključujejo bazo znanja in mehanizem sklepanja.
- Izgradnja navadno zahteva samo vnos baze znanja.

SESTAVINE EKSPERTNIH SISTEMOV

- **Baza znanja** – vsebuje znanje, specifično za dano področje reševanja problemov
- **Baza dejstev** – vsebuje dejstva, ki so ugotovljena med reševanjem danega problema
- **Baza pravil** – vsebuje odnose med možnimi dejstvi
- **Mehanizem sklepanja** – program, ki aktivira znanje v bazi znanja.
- **Uporabniški vmesnik** – omogoča komunikacijo med uporabnikom in ekspertnim sistemom.

ZGRADBA EKSPERTNIH SISTEMOV

PROBLEMI, KI JIH REŠUJEJO EKSPERTNI SISTEMI

- Problemi, ki jih tipično rešujejo strokovnjaki na ožjem strokovnem področju.
- Pogosto na področju medicine, servisnih dejavnosti, svetovanje, podajanje informacij itd.
- Udejanjanje inteligentnih avtonomnih agentov ali agentov v več-agentnih sistemih.
- Povsod, kjer inteligentni sistem vključuje veliko odločanja na osnovi baze znanja.



KDAJ UPORABITI EKSPERTNE SISTEME?

- Če razvijamo sistem za reševanje problema, ki vključuje veliko pogojnih vejitev, odvisnih od vhodnih podatkov.
- Če je sprejemanje odločitev odvisno od zapletenih pogojev, ki so v določenih primerih med sabo še soodvisni.
- Če se lahko pravila, na katerih temelji sprejemanje odločitev sčasoma tudi spremenijo.
- Če imamo namen posodabljati in izboljševati delovanje sistema s čim manj posegov v izvorno programsko kodo.
- Če naročnik ne zahteva rešitve, ki je v celoti prilagojena prav njihovim potrebam.

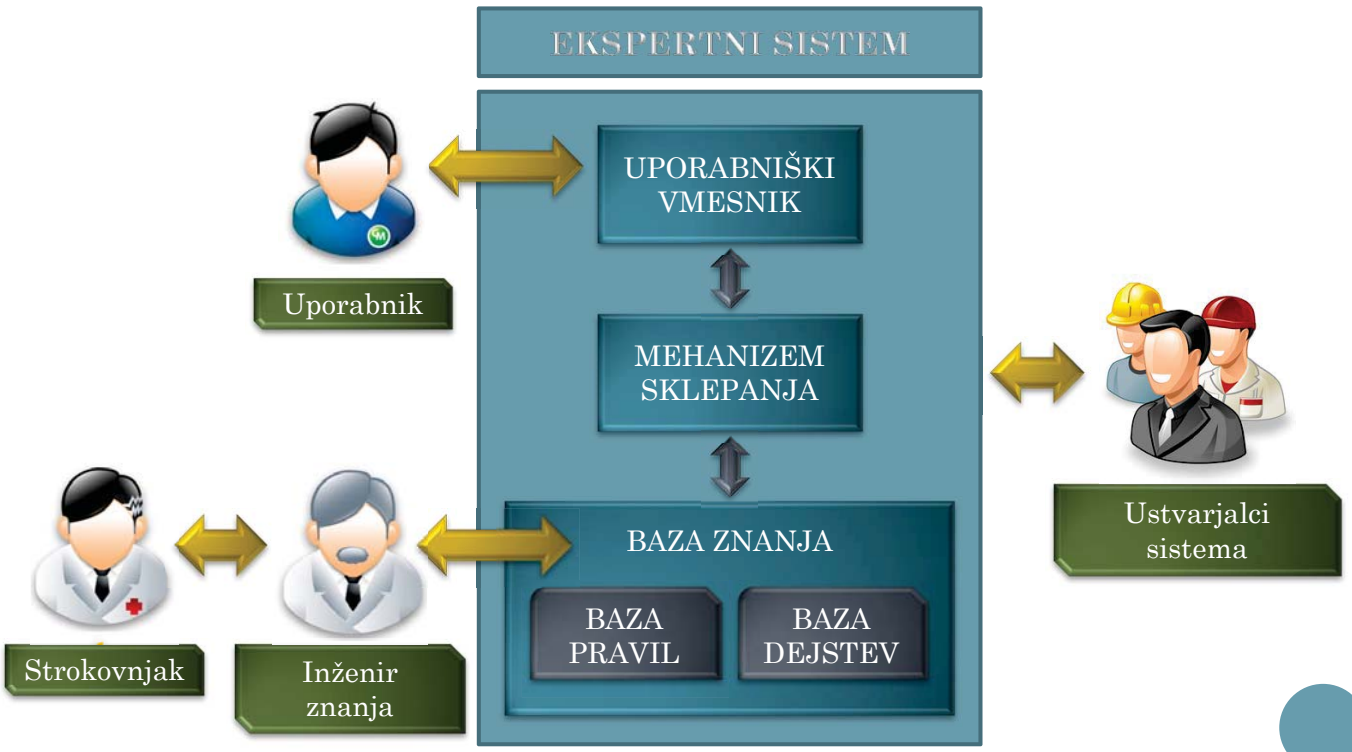
PREDNOSTI EKSPERTNIH SISTEMOV

- Nizki stroški razvoja sistema za reševanje danih problemov.
- Vso znanje sistema je zapisano v bazi znanja.
- Možnost razumljive razlage sprejetih odločitev tudi za uporabnika, ki ni računalniški strokovnjak.
- Uporaba enakega sistema za različne probleme.

OMEJITVE EKSPERTNIH SISTEMOV

- Rezultati so popolnoma odvisni od točnosti znanja.
- Težavno vnašanje baze znanja.
- Ne poznajo omejitev danega področja uporabe.
- Nimajo sposobnosti učenja iz izkušenj z uporabniki.
- Neuporabni pri nepričakovanih razmerah delovanja.
- Omejeni na manj obsežne baze znanja.

OSEBE, VPLETENE V DELOVANJE EKSPERTNI SISTEMA



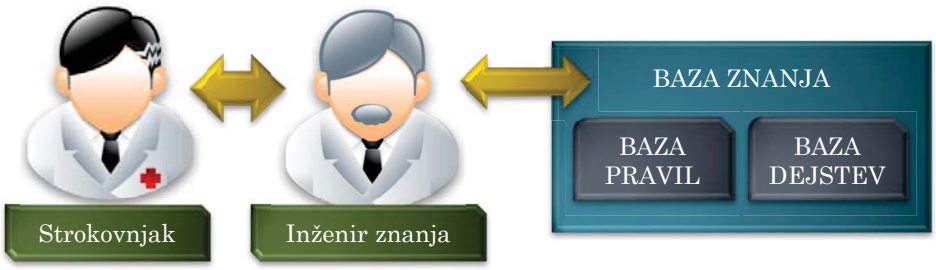
BAZA ZNANJA EKSPERTNEGA SISTEMA

- Baza znanja predstavlja formalno opisane zamisli, ki omogočajo mehanizmu sklepanja izpeljavo rešitve za dani problem.
- Večinoma je znanje opisano s produkcijskim sistemom, ki vključuje množico izjav ČE ... POTEM.

ČE *si lačen* POTEM *želiš jesti*

(če obstaja dejstvo, da *si lačen*, se to prilega premisi „*si lačen*“; pogoj je izpolnjen, zato se izpelje sklep „*želiš jesti*“)

DIALOG MED INŽENIR ZNANJA IN STROKOVNJAKOM

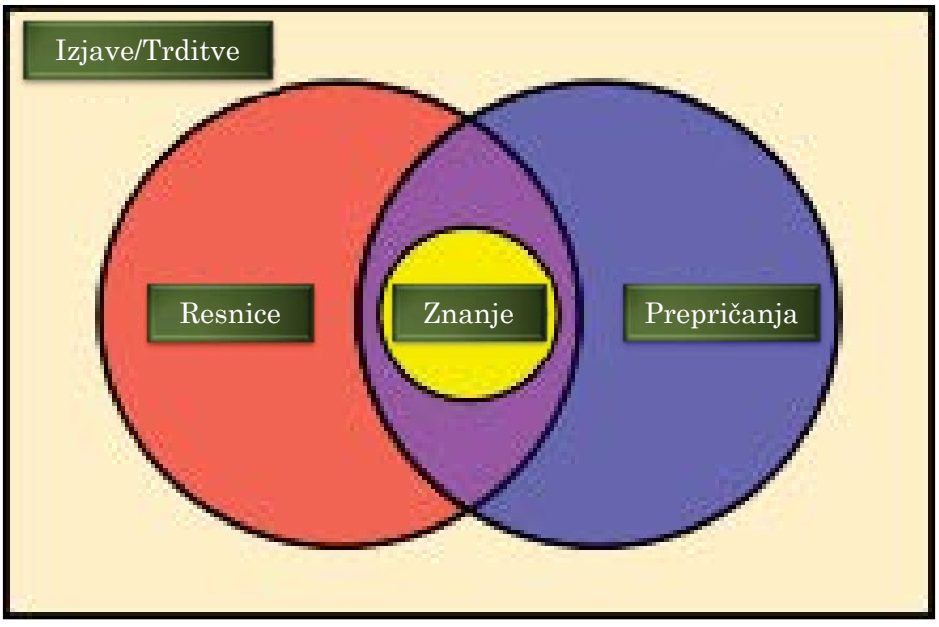


- Inženir znanja mora vzpostaviti dialog s strokovnjakom izbranega področja uporabe.
- V izbranem formalnem računalniškem deklarativnem jeziku vnaša znanje neposredno v bazo znanja.
- Strokovnjak ovrednoti delovanje sistema in daje pripombe inženirju znanja, ki bazo znanje dopolnjuje.

Predstavitev znanja v ekspertnih sistemih

- Ustrezna predstavitev znanja je ključna za uspeh ekspertnega sistema.
- Ekspertni sistemi večinoma temeljijo na predstavitvi znanja, ki temelji na logičnem sklepanju.
- Formalni študij znanja se izvaja v okviru *epistemologije*, ki se ubada z naravo, strukturo in izvorom znanja.
- Epistemologija je filozofska disciplina, ki raziskuje izvore, možnosti, meje, objektivno vrednost in predmet spoznanja.

Epistemologija/Gnoseologija



- Epistemologija obravnava prostor izjav/trditev, resnic, prepričanj in znanja.

VRSTE ZNANJA - APRIORNO ZNANJE

- Izraz izvira iz latinske besede „*kar predhodi*“.
- Je neodvisno od konkretnih izkustev in zaznav.
- Je univerzalno resnično.
- Ne more biti zavrnjeno brez kontradikcije.

APOSTERIORNO (SPO)ZNANJE

- Izraz izvira iz latinske besede „*kar sledi*“.
- Je izpeljano iz konkretnih izkustev in zaznav.
- Ni vedno zaupanja vredno.
- Je lahko zavrnjeno na podlagi novih zaznav.

Proceduralno, deklarativno in nemo znanje

- Proceduralno znanje je znanje, kako se nekaj naredi (npr. *nameščanje računalniškega programa*).
- Deklarativno znanje je vedenje o tem ali je nekaj resnično ali ni (npr. *zgodila se je prometna nesreča*).
- Implicitno znanje so veščine, ki se ne dajo izraziti z jezikom (npr. *znanje, kako premakniti nogo*).

Meta-znanje in piramida znanja

- Znanje o znanju.
- S pomočjo meta-znanja bi lahko ekspertni sistem izbiral znanje, ki je najbolj primerno za dano področje uporabe.

Predstavitev znanja

- Razčlenjevalna drevesa
- Logična pravila.
- Semantična (pomenska) omrežja.
- Konceptni grafi.
- Programske skripte.
- …

Predstavitev znanje, ki temelji na pravilih

- Znanje je predstavljeno s pravili, ki se aktivirajo z dejstvi ali drugimi pravili.
- Aktivirana pravila lahko postanejo dejstva, ki aktivirajo druga pravila.
- Rezultat so aktivacije pravil, ki določajo možen končni cilj verižnega sklepanja iz pravil.

PRODUKCIJSKI SISTEM

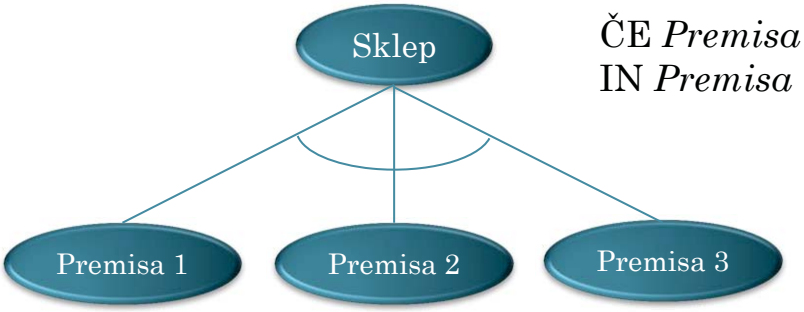
- **Baza pravil** je opisana z izjavami:

  ČE *premisa* POTEM *sklep*

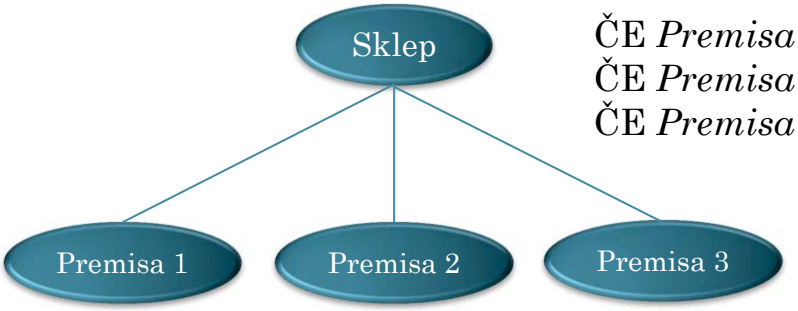
- Te izjave imenuje *produkcijska pravila.*
- **Baza dejstev** je seznam dejstev, ki potrjujejo pravilnost premis, in dejstev, ki izhajajo iz sklepov izjav.
- **Mehanizem sklepanja** je postopek, ki določa izbiro pravil in izvajanja sklepov, glede na stanje celotne baze znanja.

PREDSTAVITEV PRODUKCIJSKIH PRAVIL

- Ponazoritev pravil z IN/ALI drevesi.



ČE *Premisa 1* IN *Premisa 2*
IN *Premisa 3* POTEM *Sklep*



ČE *Premisa 1* POTEM *Sklep*
ČE *Premisa 2* POTEM *Sklep*
ČE *Premisa 3* POTEM *Sklep*

MEHANIZEM SKLEPANJA

- Izbira pravil za izvajanje sklepov se opravi na podlagi ene izmed dveh temeljnih strategij.
  - Veriženje pravil naprej (angl. *forward chaining*).
  - Veriženje pravil nazaj (angl. *backward chaining*).

VERIŽENJE PRAVIL NAPREJ

- Iskanje pravil, ki morejo potrditi pravilnost premis z dejstvi iz baze dejstev.
- Takšno pravilo izvedemo in sklep vpišemo kot novo dejstvo v bazo dejstev.
- Iskanje in izvajanje pravil poteka tako dolgo
  - dokler nek sklep ni označen kot želeni cilj izvajanja, oziroma,
  - ko iz dejstev v bazi dejstev ne moremo izvesti več nobenega sklepa.

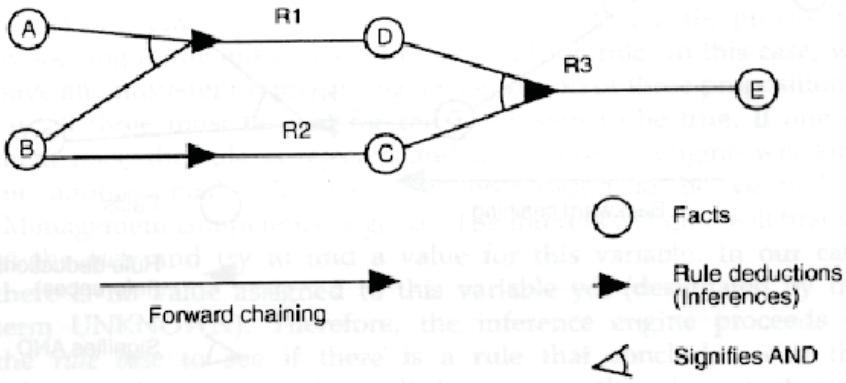
PONAZORITEV VERIŽENJA PRAVIL NAPREJ

BAZA PRAVIL

R1: IF A AND B THEN D
R2: IF B THEN C
R3: IF C AND D THEN E

BAZA DEJSTEV

A,B



VERIŽENJE PRAVIL NAZAJ

- Iskanje pravil, ki imajo sklep označen kot želeni cilj izvajanja.
- Za takšna pravila se nato preveri, če podana dejstva potrjujejo resničnost njihovih premis.
- Če takšnih dejstev ni, preverimo, ali lahko premise potrdimo z izpeljevanjem iz drugih podanih pravil glede na podana dejstva.

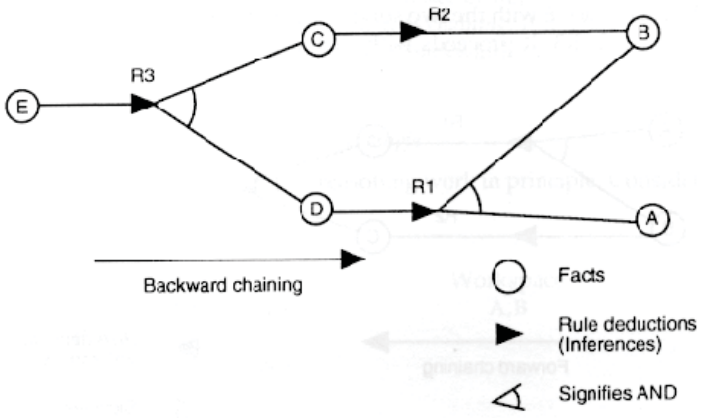
PONAZORITEV VERIŽENJA PRAVIL NAZAJ

BAZA PRAVIL

R1: IF A AND B THEN D
R2: IF B THEN C
R3: IF C AND D THEN E

BAZA DEJSTEV

A,B

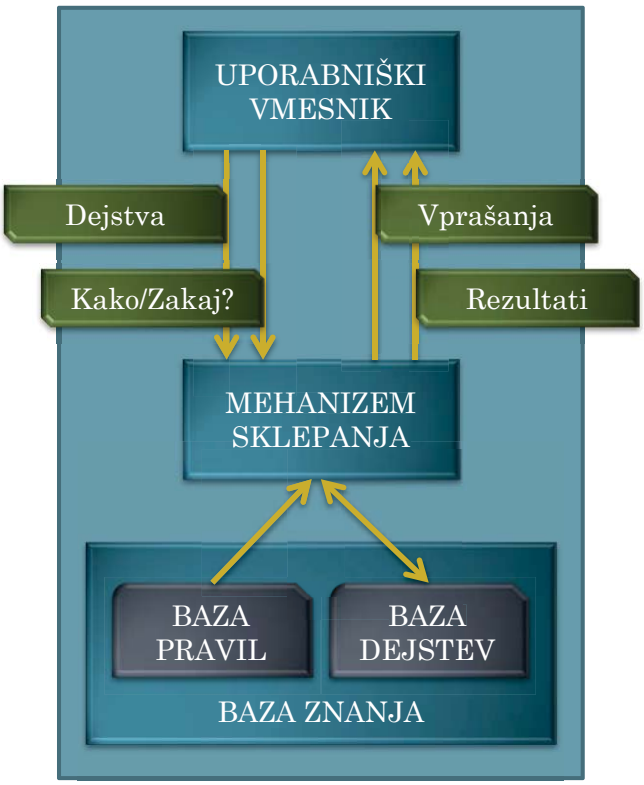


RAZLIKE MED OBEMA STRATEGIJAMA

- Veriženje naprej je strategija z iskanjem v širino.
- Veriženje nazaj je strategija z iskanjem v globino.

Veriženje pravil naprej	Veriženje pravil nazaj
•Iskanje iz sedanjosti v prihodnost	•Iskanje iz sedanjosti v preteklost
•Iskanje iz prednikov v naslednike	•Iskanje iz naslednikov v prednike
•Načrtovanje in nadzor sistemov	•Diagnostika sistemov
•Vhodni podatki določajo iskanje	•Želeni sklepi določajo iskanje
•Iskanje sklepov, ki sledijo iz dejstev	•Iskanje dejstev, ki podpirajo sklepe
•Razlaga verige sklepanja ni možna	•Razlaga verige sklepanja je možna

KOMUNIKACIJA Z UPORABNIKOM



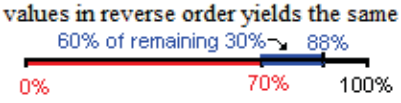
UPOŠTEVANJE NEZANESLJIVOSTI PODATKOV

- Upoštevanje negotovosti pri navajanju dejstev.
- Uporaba verjetnostnih teorij, ki upoštevajo negotovost (Bayes, Shannon, …)
- Uvajanje stopnje gotovosti sklepov.
- Uporaba neizrazitih (angl. *fuzzy*) pravil.
- Minimizacija števila pravil glede na njihov prispevek k želenim ciljem izvajanja.

KOMBINIRANJE STOPENJ GOTOVOSTI

- Pri veriženju pravil je potrebno v premisah kombinirati več vrednosti stopenj gotovosti, ki izhajajo iz negotovih dejstev ali že izpeljanih sklepov.
- Pri stopnjah gotovosti med 0 in 100 se dve vrednosti kombinirat v njun minimum, maksimum, povprečje, produkt, verjetnostno vsoto ipd.

PRIMER KOMBINIRANJA STOPENJ GOTOVOSTI

Example	Method	Result	Justification
IF a AND b CFa=60% CFb=80%	Minimum	Premise is true, CF=60%	*Both* a and b must be true for the premise to be true: the overall certainty is driven by the weakest certainty.
IF a OR b CFa=60% CFb=80%	Maximum	Premise is true, CF=80%	The premise is true if *either* a or b is true: the overall certainty is driven by the strongest certainty.
IF a AND b IF a OR b CFa=60% CFb=80%	Average	Premise is true, CF=70%	A compromise between the Maximum and Minimum methods for either premise.
CFa=70% from 1st source CFa=60% from 2nd source	Probability Sum	Final CFa=88% 70+(60/100)x(100-70)	30% uncertainty remains from the first certainty factor of 70%. The second CF explains 60% of this remaining 30%. (Note: applying the two certainty values in reverse order yields the same result.) 
IF a AND b Then c=5 with 80% confidence (premise CF=90%)	Multiplication	c is 5 with 72% confidence	The confidence assigned to the attribute in the rule conclusion is assumed independent of the premise confidence level, so the two are combined by multiplying as independent probabilities would be combined.

Lupine ekspertnih sistemov

- Obstaja več programskih ogrodij in lupin za razvoj ekspertnih sistemov.
- Vsebujejo že vso programsko arhitekturo, a so brez vgrajenega znanja.
- Primeri odprtokodnih lupin so deli programskih ogrodij CLIPS in FuzzyClips.
  http://clipsrules.sourceforge.net/
  http://en.wikipedia.org/wiki/FuzzyCLIPS
- Primer lupine za razvoj spletnih ekspertnih sistemov je tudi eXpertise2GO.
  http://www.expertise2go.com/

Primer baze znanja ekspertnega sistema

- http://www.expertise2go.com/

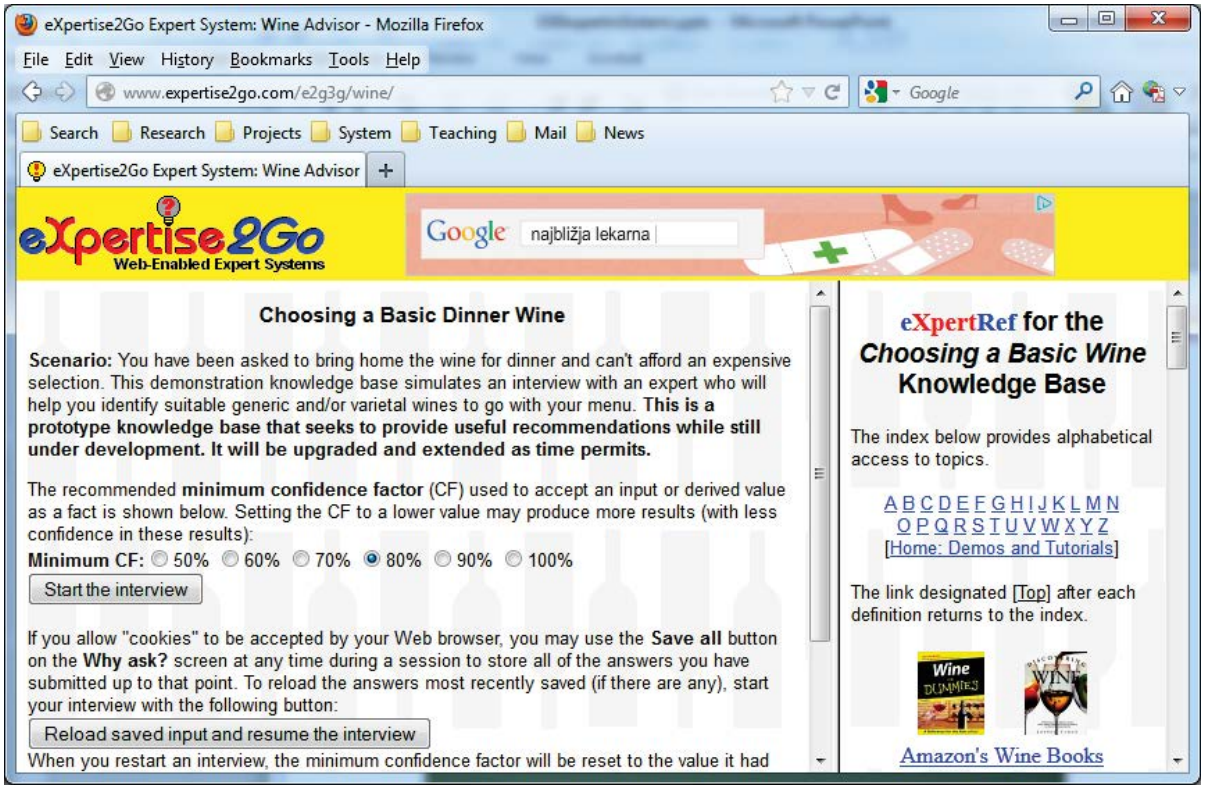
```
RULE [Choosing a full bodied aperitif]
If [this wine] = "to be consumed before a meal (aperitif)" and
[the preferred body] = "full"
Then [a recommended generic wine type] = "dry sherry"

RULE [Choosing a sparkling aperitif]
If [this wine] = "to be consumed before a meal (aperitif)" and
[a sparkling wine is preferred] = true and
[the preferred body] : "medium" "light"
Then [a recommended generic wine type] = "champagne or sparkling white"

RULE [Is this wine for a red meat entree?]
If [this wine] = "to accompany an entree" and
[the entree] : "roast beef" "steak"
Then [a recommended generic wine type] = "red burgundy" and
[the entree category] = "red meat"

RULE [Is this wine for a white/light meat entree?]
If [this wine] = "to accompany an entree" and
[the entree] : "pork" "veal" "chicken" "turkey"
Then [the entree category] = "white/light meat"

RULE [Is the wine for Italian meat & cheese dish?]
If [this wine] = "to accompany an entree" and
[the entree] = "an Italian meat and cheese dish"
Then [a recommended generic wine type] = "chianti"

…
```

Primer baze znanja expertnega sistema

- http://www.expertise2go.com/

```
…
PROMPT [a sparkling wine is preferred] YesNo
"Do you prefer a sparkline wine?:"

PROMPT [the preferred taste] MultChoice CF
"Which taste do you prefer in a wine?"
"sweet"
"medium dry"
"dry"

PROMPT [the preferred body] MultChoice CF
"Which body do you prefer in a wine?"
"light"
"medium"
"full"

PROMPT [the preferred wine color] MultChoice CF
"Which color of wine do you prefer?"
"red"
"white"

MAXVALS [a recommended generic wine type] 2
MAXVALS [a suggested varietal wine] 2

GOAL [a recommended generic wine type]
GOAL [a suggested varietal wine]

MINCF 80
```

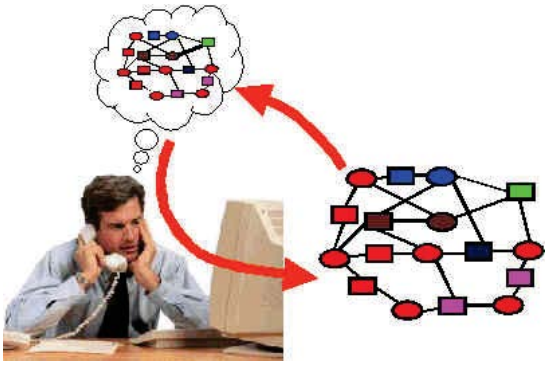

Primer expertnega sistema



VPRAŠANJA

- Kaj so ekspertni sistemi?
- Katere so osnovne sestavine ekspertnih sistemov?
- Kaj so prednosti in omejitve ekspertnih sistemov?
- Opišite produkcijski sistem.
- Opišite veriženje pravil naprej in nazaj.
- Kako opišemo negotovost podatkov?
- Podajte primere ekspertnih lupin.

Umetni inteligentni sistemi, UN2-1-IZB-vsi 2013/2014
Avtorske pravice pridržane © 2013 - Univerza v Ljubljani, Fakulteta za elektrotehniko



PREDSTAVITEV ZNANJA

POVZETEK

- Kako predstaviti znanje
- Razvoj znanja od podatkov do modrosti.
- Implicitno in eksplicitno znanje
- Proceduralno in deklarativno znanje
- Obrazci za predstavitev znanja

KAKO PREDSTAVITI ZNANJE?

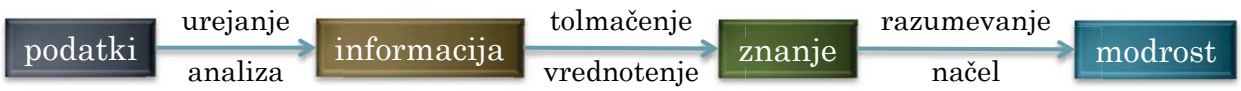
- Znanje je nek splošen, lahko tudi slabo definiran pojem.
- Znanje podaja odgovor na vprašanje „*kako se nekaj naredi*“ in/ali „*kaj je ali ni resnično*“
- Npr., vedeti, „*kako se vozi avto*“ je proceduralno znanje.
- Npr., vedeti, „*da zunaj sije sonce*“, je deklarativno znanje.
- Znanje določa neko sposobnosti/zmožnosti sistema.
- Predstavitev znanja je način, kako je znanje v sistemu zapisano/shranjeno oziroma predstavljeno človeku.

NAČINI PREDSTAVITVE ZNANJA

- Različno znanje zahteva različne načine predstavitve znanja.
- Obstaja več načinov predstavitve znanja, kot so:
  - predstavitve znanja s pravili,
  - predstavitev znanja z uporabo semantičnih omrežij,
  - predstavitev znanja z matematično logiko,
  - predstavitev znanja z uporabo okvirjev,
  - predstavitev znanja s programskimi skripti,
  - …
- Vsaka predstavitev znanja lahko zahteva različen način sklepanja iz podanih dejstev.

Razvoj znanja od podatkov do modrosti

- Razvoj znanja se prične s podatki, ki imajo omejeno uporabnost.
- Z urejanjem in analizo (vzpostavljanje relacij) podatki oblikujejo informacijo.
- Tolmačenje in ovrednotenje informacije privede do znanja.
- Razumevanje osnovnih načel, ki so vgrajena v znanje, vodi v meta znanje ali modrost (znanje o znanju).



Primer razvoja znanja

- Podatki so neka nepovezana dejstva.
  - Na primer: *Dežuje.*
- Informacija se pojavi z razumevanjem in povezovanjem dejstev. Podaja odgovore na vprašanja, „*kdo*“, „*kaj*“, „*kje*“ in „*kdaj*“.
  - Na primer: *Temperatura se je znižala za 10 stopinj in potem je pričelo deževati*
- Znanje se pojavi, ko se ugotovi in razume relacije med vzorci. Podaja odgovore na vprašanje, „*kako*“.
  - Na primer: *Če je vlažnost zraka visoka in se temperatura znatno zniža, se v zraku kondenzirajo kapljice vode, zato prične deževati.*
- Modrost je vrh razumevanja, ki razkriva načela, ki opisujejo vzorcev. Podaja odgovore na vprašanje, „zakaj“.
  - Na primer: *Obstajajo medsebojni vplivi med vlažnostjo, hlapenjem, zračnimi tokovi in spremembami temperature.*

Model znanja

- Stopnja povezanosti in s tem razumevanja se povečuje pri napredovanju od podatkov, preko informacije in znanja do modrosti.
- Ločnica med podatki, informacijo, znanjem in modrostjo ni ostra ampak pogosto zabrisana.
- Podatki in informacija se nanašajo na neko bolj ali manj oddaljeno preteklost.
- Znanje se nanaša na sedanjost in omogoča sistemu delovati v okolju.
- Modrost se nanaša na prihodnost in omogoča napovedovanje prihodnjih dogodkov in okoliščin.

Eksplicitno, tiho in implicitno znanje

- Implicitno ali tiho znanje se težko formalno izrazi/opiše.
- Eksplicitno znanje se da formalno izraziti/opisati.

Implicitno znanje

- Obstaja v človeku, je v njem „utelešeno“.
- Ga je težko formalno opisati.
- Ga je težko deliti z drugimi.
- Ga je težko ukrasti.
- Pridobljeno z izkušnjami in delovanjem in je subjektivno.

Eksplicitno znanje

- Obstaja izven človeka, je vgrajeno.
- Se ga da formalno opisati.
- Se lahko deli, kopira, obdeluje in hrani.
- Enostavno ga je ukrasti oz kopirati.
- Pridobljeno iz okolja in predmetov, kot neka splošna načela, postopki, koncepti.

PREGLED TIPOLOGIJA ZNANJA 1/2

PREGLED TIPOLOGIJA ZNANJA 2/2

- Dejstva so nekaj, kar v resnici obstaja ali se je zgodilo. Dejstva lahko predstavimo kot podatke.
- Koncepti so razredi stvari, besed ali idej, ki so predstavljena s skupnim imenom in si delijo skupne značilnosti.
- Proces je tok dogodkov, ki opisuje, kako nekaj poteka.
- Procedura je zaporedje delovanj in odločitev, ki vodijo k izpolnitvi naloge in razlaga kako se nekaj naredi.
- Princip so navodila, pravila in parametri, ki vodijo samo delovanje in omogočajo napovedovanje dogodkov in izpeljavo možnih posledic.

Preceduralno in deklarativno znanje

- Proceduralno znanje je znanje o tem, kako se nekaj naredi.
- Deklarativno znanje je znanje o tem, kaj je ali ni resnično.
- Proceduralno znanje je lahko implicitno oz. tiho (npr. smučanje ali kuhanje po „občutku“) ali eksplicitno (npr. kuharski recept).
- Proceduralno znanje se včasih razume kot deklarativno (deklarativni opis metod, ki je podoben opisu nekih dejstev).
- Deklarativno znanje je navadno eksplicitno in se ga da formalno predstaviti oz. opisati.

Primeri proceduralnega znanja

- *Da bi določil, ali je Jože starejši kot Miha, najprej ugotovi njuno starost in nato primerjaj njuna leta.*
- *Da bo avto speljal, ga je potrebno najprej prižgati.*
- *Kruh spečeš tako, da testo iz moke izpostaviš visoki temperaturi vsaj za pol ure.*
- …
- *Za peko kruha potrebuješ pečico.*
  (nekaj med proceduralnim in deklarativnim znanjem)

Primeri deklarativnega znanja

- *Zunaj dežuje.*
- *Avto ima štiri kolesa.*
- *Jože je starejši od Miha.*
- *Enakokraki trikotnik ima dve stranici enako dolgi.*
- *Pes je žival*
- *Janez se pelje v Kranj.*
- …

Relacije med vrstami znanja

- Deklarativno znanje je povezano z opisovanjem stanja stvari oz. opisovanjem resničnosti/okolja.
- Deklarativno znanje je navadno vedno eksplicitno izraženo.
- Proceduralno znanje je povezano z delovanjem oz. opisovanjem delovanja in navadno izhaja iz tihega znanja.
- Proceduralno znanje se pogosto razvije iz deklarativnega (npr. najprej vemo, da potrebujemo pečico, da spečemo kruh, šele nato vemo, da moramo kruh vložiti v pečico, da se kruh speče).

Računalniška predstavitev znanja 1/2

- Računalniki zahtevajo dobro definiran problem, da ga obdelajo in tvorijo dobro definirano rešitev problema.
- Predstavitev znanja temelji na naravnem jeziku, računalniku pa se predstavi v nekem formalnem jeziku.
- Računalniško reševanje problemov zahteva:
  - formalno predstavitev znanja
  - pretvorbo neformalnega znanja v formalno znanje, to je pretvorbo implicitnega znanja v eksplicitno znanje.

Računalniška predstavitev znanja 2/2

- Pri razvoju računalništva se je razvilo več oblik formalnih predstavitev znanja.
- Pretvorba neformalnega v formalno znanje pa je v glavnem prepuščena izkušnjam razvijalcem tovrstnih sistemov.
- Za formalno predstavitev znanja moramo znati dejstva preslikati v simbole in tudi simbole nazaj v dejstva.
- Naravni jezik (angleščina, slovenščina, …) je zaenkrat še prezahteven za simbolno predstavitev znanja, ki bi jo lahko uporabljal računalnik.
- Zato se poslužujemo različnih oblik poenostavljenih formalnih predstavitev znanja.

ZGLED PREDSTAVITVE IN UPORABE ZNANJA

Dejstva

- *Floki je pes.*
- `pes('Floki')`
- $\forall$ `pes(X)` $\rightarrow$ `ima(X,'rep')`

Predstavitev

- Dejstvo je predstavljeno v naravnem jeziku (slovenščini).
- Simbolna predstavitve dejstva v jeziku matematične logike.
- Logična predstavitev dejstva, „*Vsi psi imajo rep*.“

- Mehanizem deduktivnega logičnega sklepanja izpelje novo simbolno predstavljeno dejstvo.

- `ima('Floki','rep')`
- *Floki ima rep.*

- Simbolna predstavitev logično izpeljanega dejstva.
- Predstavitev novega dejstva v naravnem jeziku (slovenščini).

OBRAZCI ZA PREDSTAVITEV ZNANJA

- V naravnem jeziku izraženo dejstvo

  „*Janez se pelje v Kranj*.“

  lahko zapišemo v računalniški podatkovni medij kot navaden niz znakov/simbolov.

- V tem primeru računalniški program, ki omogoča iskanje podatkov v podatkovni zbirki, ne more podati odgovor na vprašanje

  „*Kdo se pelje v Kranj*?“

- Zapis podanega dejstva je potrebno obogatiti z dodatno informacijo, ki bo omogočila odgovore na takšna vprašanja.

Zgled obrazca za predstavitev znanja 1/2

- Simbolno predstavitev dejstva je potrebo razširiti z uvedbo dodatnih pomenskih (semantičnih) oznak.

Pomenska kategorija	Pomen
Akcija:	'peljati se'
Nosilec akcije:	'Janez'
Izhodišče:	'?'
Cilj:	'Kranj'
Čas:	'sedanjik'
Sredstvo:	'?'

Zgled obrazca za predstavitev znanja 2/2

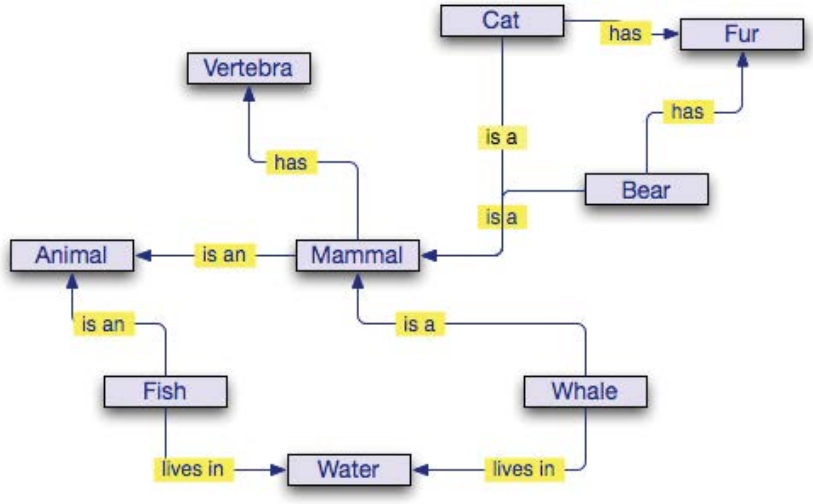
- Uporaba mehanizma sklepanja in drugega obstoječega znanja omogoča dodatno razširitev zapisa.

Pomenska kategorija	Pomen	Dodatna pomenska oznaka	
Akcija:	'peljati se'	(pomenski tipi)	'gibanje'
Nosilec akcije:	'Janez'	(pomenski tipi)	'človek'
Izhodišče:	'?'	(privzet pomen)	'Janezov dom'
Cilj:	'Kranj'	(pomenski tipi)	'mesto', 'mesto na Gorenjskem'
Čas:	'sedanjik'	(pomenski tipi)	'glagolski čas'
Sredstvo:	'?'	(privzet pomen)	'avtomobil'

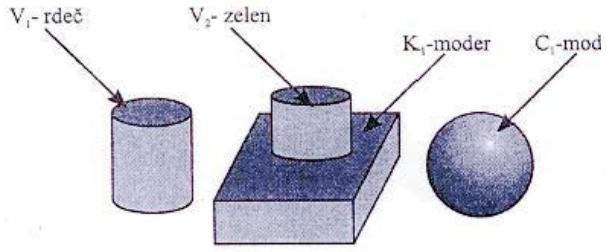
OBRAZCI ZA PREDSTAVITEV ZNANJA

- Logične izjave ČE … POTEM.
- Opisni jezik matematične logike.
- Pomenska (semantična) omrežja.
- (Neizrazita) Petrijeva omrežja
- Okvirji, tj. objekti, ki so opisani z svojo zgradbo in vedenjem.
- Računalniške programske skripte.
- …

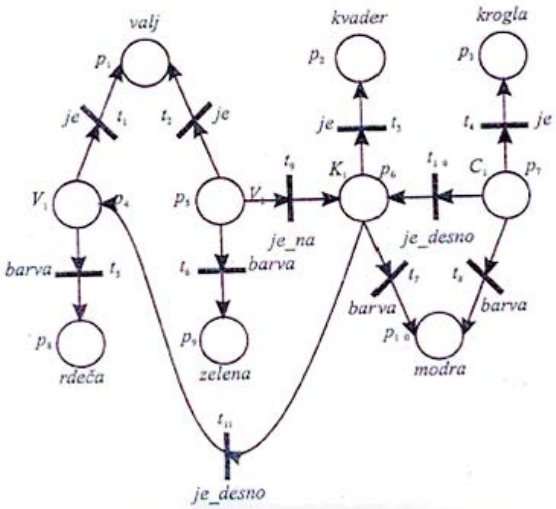
ZGLED SEMANTIČNEGA OMREŽJA



ZGLED PETRIJEVEGA OMREŽJA

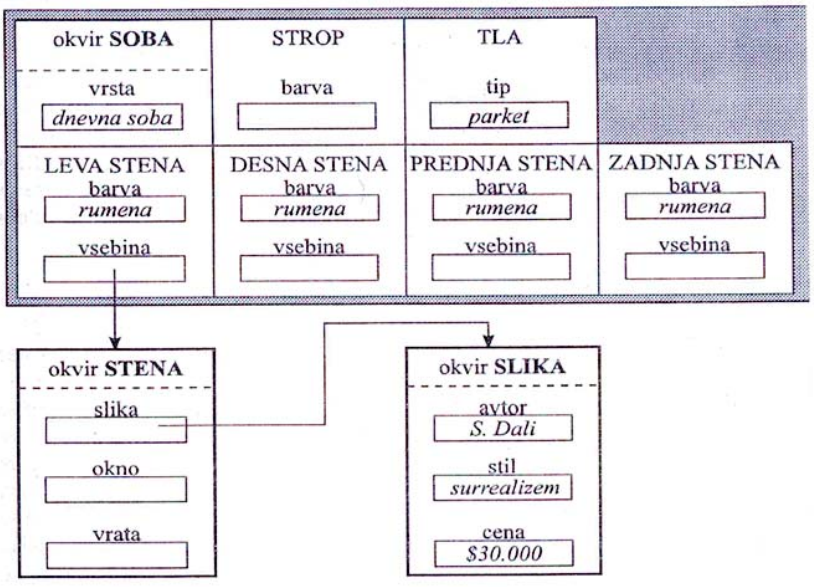


Slika D.1: Preprost prizor.



ZGLED SEMANTIČNIH OKVIRJEV

- Področje t.i. kognitivne lingvistike
  (dodajanje semantičnih okvirjev t.j. znanja besedam v izjavi)



VPRAŠANJA

- Kako poskušamo predstaviti znanje?
- Opišite napredovanje znanja od podatkov do modrosti.
- Kakšna je razlika med implicitnim in eksplicitnim znanjem?
- Kakšna je razlika med proceduralnim in deklarativnim znanjem?
- Katere obrazce za predstavitev znanja poznamo.

Umetni inteligentni sistemi, UN2-1-IZB-vsi 2013/2014
Avtorske pravice pridržane © 2013 - Univerza v Ljubljani, Fakulteta za elektrotehniko

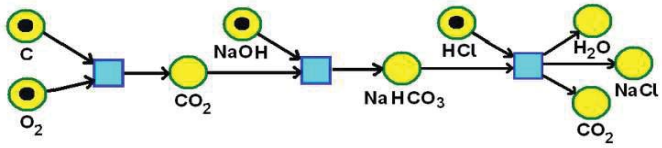
Najnujnejše o Petrijevih omrežjih

Vsebina

Petrijeva omrežja (pravimo jim tudi P/T omrežja po angleškem izrazu *place/transition*) so posplošen matematični formalizem, ki se uporablja za opis in predstavitev sočasnih procesov, ki se pojavljajo pri porazdeljenih (distribuiranih) sistemih.

Grafično predstavitev tega matematičnega formalizma je izumil Carl Adam Petri, ki ga je pri starosti 13 let prvič uporabil za predstavitev kemičnih procesov, kot je ta na spodnji sliki.

Ta omrežja so se izkazala kot zelo uspešno grafično in matematično orodje za modeliranje avtomatiziranih proizvodnih sistemov, komunikacijskih protokolov, z znanjem podprtih sistemov itd.



Slika 1: Primer Petrijevega omrežja, ki opisuje kemični proces.

Petrijevo omrežje je dvostranski usmerjeni graf, ki sestoji iz dveh vrst vozlišč: **mest** in **prehodov**. Grafično mesta predstavimo s krogi, prehode pa s črticami. Odnosi med mesti in prehodi so ponazorjeni z usmerjenimi povezavami.

Definicija 1 Petrijevo omrežje *je petorka* $PN = (P, T, I, O, \mu)$, *kjer je:*

$P = \{p_1, p_2, ..., p_m\}$ *končna množica mest,*

$T = \{t_1, t_2, ..., t_n\}$ *končna množica prehodov,* $P \cap T = \emptyset$,

I : $T \to P^\infty$ *vhodna funkcija – preslikava iz množice prehodov v vrečo mest,*

O : $T \to P^\infty$ *izhodna funkcija – preslikava iz množice prehodov v vrečo mest,*

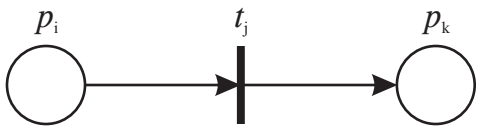
μ : $P \to \mathbb{N}$ *funkcija označitve – preslikava iz množice mest v množico nenegativnih celih števil.*

Izraz *vreča* pomeni množico, v kateri so elementi lahko večkrat navedeni.

Če $p_i \in I(t_j)$, potem obstaja usmerjena povezava, ki povezuje mesto p_i s prehodom t_j, in se mesto p_i imenuje *vhodno mesto* prehoda t_j.

Podobno, če $p_k \in O(t_j)$, potem obstaja usmerjena povezava, ki povezuje prehod t_j z mestom p_k, in se mesto p_k imenuje *izhodno mesto* prehoda t_j.

Slika 2 ponazarja osnovno Petrijevo omrežje z enim prehodom, ki ima eno vhodno ter eno izhodno mesto.



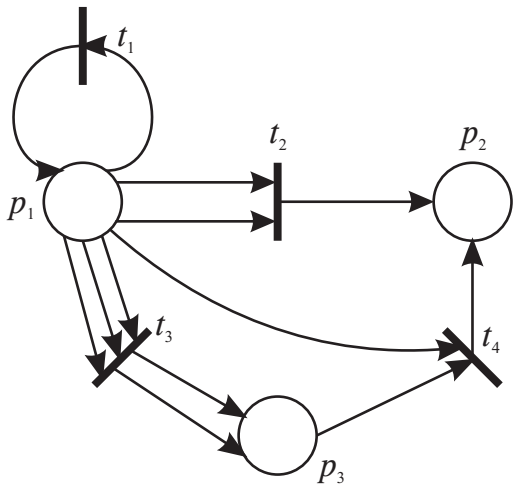
Slika 2: Osnovno Petrijevo omrežje.

Po definiciji 1 sme neko mesto biti večkratno vhodno oziroma izhodno mesto nekega prehoda, kar grafično ponazorimo z večkratnimi usmerjenimi povezavami.

S $\sharp(p_i, I(t_j))$ označujemo večkratnost vhodnega mesta p_i prehoda t_j, to je število pojavljanj mesta p_i v vreči vhodnih mest prehoda t_j.

Podobno je definirana tudi oznaka $\sharp(p_k, O(t_j))$, ki pomeni večkratnost izhodnega mesta p_k prehoda t_j, to je število pojavljanj mesta p_k v vreči izhodnih mest prehoda t_j.

Zgled 1



Slika 3: Primer Petrijevega omrežja.

Petrijevo omrežje na sliki 3 je definirano s:

$$
\begin{array}{ll}
P = \{p_1, p_2, p_3\} & \\
T = \{t_1, t_2, t_3, t_4\} & \\
I(t_1) = \{p_1\} & O(t_1) = \{p_1\} \\
I(t_2) = \{p_1, p_1\} & O(t_2) = \{p_2\} \\
I(t_3) = \{p_1, p_1, p_1\} & O(t_3) = \{p_3, p_3\} \\
I(t_4) = \{p_1, p_3\} & O(t_4) = \{p_2\}
\end{array}
$$

■

Posebna vrsta Petrijevih omrežij so *stroji stanja* (angl. state machine). Stroj stanja je Petrijevo omrežje, v kateri ima vsak prehod natanko eno vhodno mesto in natanko eno izhodno mesto.

Definicija 2 Stroj stanja *je petorka* $SM = (P, T, I, O, \mu)$, *kjer so:*

$P = \{p_1, p_2, ..., p_m\}$ *končna množica mest,*

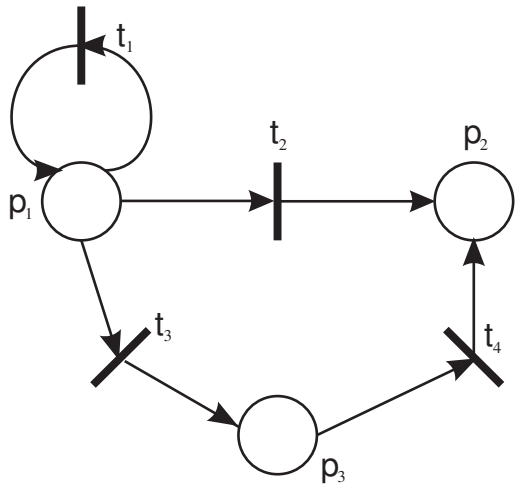
$T = \{t_1, t_2, ..., t_n\}$ *končna množica prehodov,* $P \cap T = \emptyset$,

I : $T \to P$ *vhodna funkcija – preslikava iz množice prehodov v množico mest,*

O : $T \to P$ *izhodna funkcija – preslikava iz množice prehodov v množico mest,*

μ : $P \to \mathbb{N}$ *funkcija označitve – preslikava iz množice mest v množico nenegativnih celih števil.*

Zgled 2



Slika 4: Primer Petrijevega stroja stanj.

Petrijev stroj stanj na sliki 4 je definiran s:

$$
\begin{array}{ll}
P = \{p_1, p_2, p_3\} & \\
T = \{t_1, t_2, t_3, t_4\} & \\
I(t_1) = \{p_1\} & O(t_1) = \{p_1\} \\
I(t_2) = \{p_1\} & O(t_2) = \{p_2\} \\
I(t_3) = \{p_1\} & O(t_3) = \{p_3\} \\
I(t_4) = \{p_3\} & O(t_4) = \{p_2\}
\end{array}
$$

■

Vhodno in izhodno funkcijo I in O je možno modificirati, tako da mestom prirejata njihove vhodne in izhodne prehode:

I : $P \to T^{\infty}$ modificirana vhodna funkcija – preslikava iz množice mest v vrečo prehodov,

O : $P \to T^{\infty}$ modificirana izhodna funkcija – preslikava iz množice mest v vrečo prehodov.

Potem velja:

$$\sharp(t_j, I(p_k)) = \sharp(p_k, O(t_j))$$
$$\sharp(t_j, O(p_i)) = \sharp(p_i, I(t_j)).$$

Prehod t_j je *vhodni prehod* mesta p_k, če je p_k izhodno mesto prehoda t_j, in prehod t_j je *izhodni prehod* mesta p_i, če je p_i vhodno mesto prehoda t_j.

Zgled 3 Za Petrijevo omrežje na sliki 3 veljata modificirani vhodna in izhodna funkcija:

$$\begin{array}{ll} I(p_1) = \{t_1\} & O(p_1) = \{t_1, t_2, t_2, t_3, t_3, t_3, t_4\} \\ I(p_2) = \{t_2, t_4\} & O(p_2) = \emptyset \\ I(p_3) = \{t_3, t_3\} & O(p_3) = \{t_4\} \end{array}$$

■

Definicija 3 *Dualno omrežje Petrijevega omrežja* $PN = (P, T, I, O, \mu)$ *je Petrijevo omrežje* $\underline{PN} = (T, P, I, O, \mu)$*, ki ga dobimo z medsebojno zamenjavo mest in prehodov.*

Graf dualnega Petrijevega omrežja ima v primerjavi z izvirnim grafom Petrijevega omrežja zamenjane kroge in črtice, medtem ko struktura grafa ostane nespremenjena.

Definicija 4 *Inverzno omrežje Petrijevega omrežja* $PN = (P, T, I, O, \mu)$ *je Petrijevo omrežje* $-PN = (P, T, O, I, \mu)$*, ki ga dobimo z medsebojno zamenjavo vhodne in izhodne funkcije.*

Graf inverznega Petrijevega omrežja ima v primerjavi z izvirnim grafom Petrijevega omrežja zamenjane smeri vseh povezav, medtem ko struktura grafa ostane nespremenjena.

Vsakemu mestu Petrijevega omrežja moremo pridružiti *žeton*, ki ga v grafu Petrijevega omrežja ponazorimo s piko v mestu. Razporeditev žetonov v mestih Petrijevega omrežja je določena z *označitvijo* omrežja.

Definicija 5 *Označitev Petrijevega omrežja* μ *je funkcija, ki vsakemu mestu Petrijevega omrežja priredi neko nenegativno celo število:*

$$\mu : \quad P \to \mathbb{N}.$$

Označitev Petrijevega omrežja z m mesti predstavimo z *vektorjem označitve* $\vec{\mu}$ (dim $\vec{\mu} = m \times 1$):

$$\vec{\mu} = (\mu(p_1),\, \mu(p_2), ..., \mu(p_m))^T.$$

Komponente vektorja označitve (označimo jih kot $\mu(p_i)$) so nenegativna cela števila in pomenijo število žetonov v pripadajočih mestih.

Žetoni dajejo Petrijevemu omrežju dinamične lastnosti. Število in razporeditev žetonov se lahko spreminja med *izvajanjem* Petrijevega omrežja. Petrijevo omrežje se izvaja *s proženjem prehodov*. Proženje prehoda t_j povzroči prenos žetonov iz vseh vhodnih mest tega prehoda v izhodna mesta prehoda t_j. Izvajanje Petrijevega omrežja določata dve pravili:

- pravilo omogočenosti in
- pravilo proženja.

Definicija 6 *(***Pravilo omogočenosti** *(angl. enabling rule))*
Prehod t_j je omogočen, če ima vsako njegovo vhodno mesto najmanj toliko žetonov, kolikor je povezav iz vhodnega mesta k prehodu t_j:

$$\mu(p_i) \geq \sharp(p_i, I(t_j)) \quad \textit{za} \ \forall p_i \in P.$$

Definicija 7 (**Pravilo proženja** *(angl. firing rule))*
Prehod t_j se proži le, če je omogočen. Proženje prehoda spremeni razporeditev žetonov v mestih Petrijevega omrežja, zato se spremeni vektor označitve $\vec{\mu}$:

$$\mu'(p_i) = \mu(p_i) - \sharp(p_i, I(t_j)) + \sharp(p_i, O(t_j)) \quad za \ \forall p_i \in P,$$

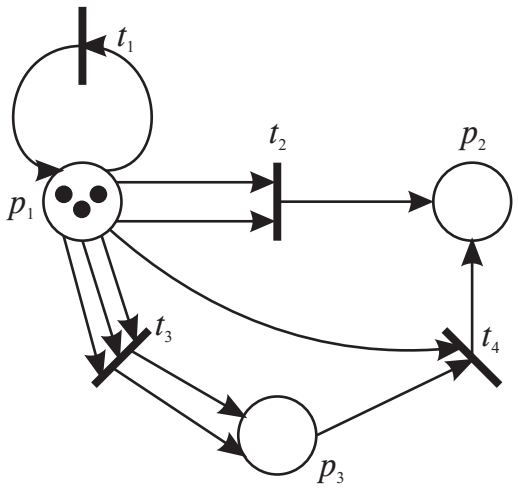
kjer je $\mu'(p_i)$ komponenta vektorja nove označitve Petrijevega omrežja $\vec{\mu}'$.

Zgled 4 Vektor označitve za Petrijevo omrežje s slike 5 je:

$$\vec{\mu} = (\mu(p_1) = 3, \mu(p_2) = 0, \mu(p_3) = 0)^T.$$

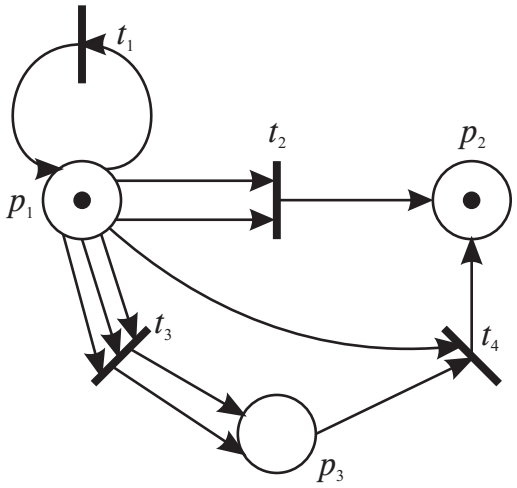
Omogočeni prehodi so t_1, t_2 in t_3. Proženje prehoda t_2 spremeni vektor označitve:

$$\vec{\mu}' = (\mu(p_1) = 1, \mu(p_2) = 1, \mu(p_3) = 0)^T.$$



Slika 5: Označeno Petrijevo omrežje.

Novo razporeditev žetonov v Petrijevem omrežju ponazarja slika 6. ▪



Slika 6: Posledica proženja prehoda t_2.

V vsakem trenutku razporeditev žetonov v mestih Petrijevega omrežja določa njeno stanje. Sprememba stanja Petrijevega omrežja je posledica proženja nekega omogočenega prehoda in je definirana s funkcijo naslednjega stanja.

Definicija 8 *Funkcija naslednjega stanja Petrijevega omrežja* $\delta : \mathbb{N}^m \times T \rightarrow \mathbb{N}^m$ *za označitev* $\vec{\mu}$ *in prehod* $t_j \in T$ *je definirana natanko tedaj, ko za vsako mesto* $p_i \in P$ *velja pravilo omogočenosti:*

$$\mu(p_i) \geq \sharp(p_i, I(t_j)).$$

Potem je:

$$\delta(\vec{\mu}, t_j) = \vec{\mu}',$$

kjer je $\vec{\mu}'$ *nova označitev Petrijevega omrežja. Komponente vektorja nove označitve Petrijevega omrežja so:*

$$\mu'(p_i) = \mu(p_i) - \sharp(p_i, I(t_j)) + \sharp(p_i, O(t_j)).$$

Pravimo, da je označitev $\vec{\mu}'$ *dosegljiva* iz označitve $\vec{\mu}$, če obstaja zaporedje prehodov, ki s svojimi proženji postopoma privedejo označitev omrežja $\vec{\mu}$ do označitve $\vec{\mu}'$. *Dosegljivostna množica* $D(PN)$ Petrijevega omrežja za označitev $\vec{\mu}$, je množica vseh označitev omrežja, ki so dosegljiva iz označitve $\vec{\mu}$.

Definicija 9 *Dosegljivostna množica $D(PN)$ Petrijevega omrežja za označitev $\vec{\mu}$, je najmanjša množica označitev definirana z:*

1. $\vec{\mu} \in D(PN)$ in

2. $\vec{\mu}' \in D(PN)$ in $\vec{\mu}'' = \delta(\vec{\mu}', t_j)$ za nek $t_j \in T \Rightarrow \vec{\mu}'' \in D(PN)$.

Metode za analizo Petrijevih omrežij

Med mnogimi metodami za analizo Petrijevih omrežij sta dve temeljni:

- invariantna metoda in
- metoda dosegljivostnih dreves.

Invariantna metoda (angl. invariant method) temelji na opisu dinamičnih lastnosti Petrijevih omrežjih z matričnimi enačbami.

Za Petrijevo omrežje brez lastnih zank (lastna zanka nastane, kadar je neko mesto omrežja vhodno in hkrati izhodno mesto istega prehoda) z m mesti in n prehodi definiramo incidenčno matriko:

$$\mathbf{A} = [a_{ij}] \quad (dim\, \mathbf{A} = n \times m).$$

Elementi incidenčne matrike so:

$$a_{ij} = a_{ij}^{+} - a_{ij}^{-},$$

kjer sta:

a_{ij}^{+} število povezav, ki povezujejo prehod t_i z njegovim izhodnim mestom p_j, $(p_j \in O(t_i))$ in

a_{ij}^{-} število povezav, ki povezujejo prehod t_i z njegovim vhodnim mestom p_j, $(p_j \in I(t_i))$.

Petrijevo omrežje z lastnimi zankami pa je možno preoblikovati v omrežje brez lastnih zank, tako da v omrežje dodamo eno mesto in en prehod ter primerno povežemo.

Ko se proži prehod t_i, število a_{ij}^{+} pomeni število žetonov, ki se bodo prenesli na izhodno mesto p_j, število a_{ij}^{-} pa število žetonov, ki se bodo odstranili iz vhodnega mesta p_j.

Število a_{ij} potem pomeni spremembo v številu žetonov v mestu p_j, ko se prehod t_i proži enkrat.

Pravimo, da je pri trenutni označitvi Petrijevega omrežja $\vec{\mu}$ prehod t_i omogočen, če velja:

$$a_{ij}^{-} \leq \mu(p_j) \quad \text{za} \;\; j = 1, 2, ..., m.$$

Dinamične lastnosti Petrijevega omrežja, to je spremembo v razporeditvi žetonov, ki je posledica proženja prehoda, opisuje matrična enačba stanja:

$$\vec{\mu}_k = \vec{\mu}_{k-1} + \mathbf{A}^T \mathbf{e}_k,$$

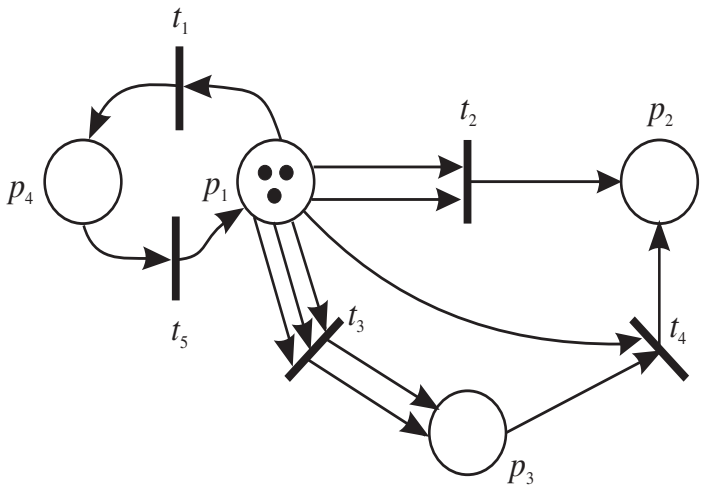
kjer sta:

$\vec{\mu}_k$ vektor označitve ($dim\ \vec{\mu}_k = m \times 1$), neposredno dosegljive iz označitve μ_{k-1} po proženju prehoda t_i in

$\mathbf{e}_k$ vektor proženja ($dim\ \mathbf{e}_k = n \times 1$), ki ima vse svoje komponente enake 0, le i-ta komponenta ima vrednost 1 (ta je v povezavi s prehodom t_i, ki se proži).

Incidenčna matrika je primerna za računanje omogočenih prehodov in novih označitev Petrijevega omrežja po proženju izbranega prehoda.

Zgled 5 Petrijevo omrežje z lastnimi zankami iz slike 5 preoblikujemo v omrežje brez lastnih zank



Slika 7: Petrijevo omrežje brez lastnih zank

Incidenčna matrika podanega omrežja je

$$\mathbf{A} = [a_{ij}] = \begin{bmatrix} -1 & 0 & 0 & 1 \\ -2 & 1 & 0 & 0 \\ -3 & 0 & 2 & 0 \\ -1 & 1 & -1 & 0 \\ 1 & 0 & 0 & -1 \end{bmatrix}$$

pri čemer je

$$[a_{ij}^{+}] = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \quad [a_{ij}^{-}] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Vektor označitve za Petrijevo omrežje s slike 7 je:

$$\vec{\mu} = (\mu(p_1) = 3, \mu(p_2) = 0, \mu(p_3) = 0, \mu(p_3) = 0)^T.$$

Omogočeni prehodi so t_1, t_2 in t_3 (prve tri vrstice matrike $[a_{ij}^{-}]$ so namreč po vseh komponentah manjše ali enake komponentam vektorja označitve $\vec{\mu}_0 = \vec{\mu} = (3, 0, 0, 0)^T$

Proženje prehoda t_2 nam v skladu z enačbo

$$\vec{\mu}_k = \vec{\mu}_{k-1} + \mathbf{A}^T \mathbf{e}_k,$$

kjer je izbor omogočenega prehoda podan v vektorju

$$\mathbf{e}_k = (0, 1, 0, 0, 0)^T,$$

spremeni vektor označitve v:

$$\vec{\mu}_1 = (\mu(p_1) = 1, \mu(p_2) = 1, \mu(p_3) = 0, \mu(p_3) = 0)^T.$$

Metoda dosegljivostnih dreves (angl. method based on the reachability tree) temelji na določanju vseh možnih označitev Petrijevega omrežja, ki so dosegljive iz začetne označitve $\vec{\mu}_0$.

Če začnemo z začetno označitvijo omrežja $\vec{\mu}_0$, potem lahko pridobimo toliko novih označitev omrežja, kolikor je omogočenih prehodov, ki se prožijo.

Dosegljivostno drevo zgradimo, če prožimo vse omogočene prehode v vseh možnih označitvah omrežja, ki so dosegljive iz začetne označitve $\vec{\mu}_0$.

Vsako vozlišče drevesa predstavlja označitev omrežja generirano iz začetne označitve $\vec{\mu}_0$ v korenu drevesa.

Veje drevesa so označene s prehodom, katerega proženje transformira eno označitev omrežja v novo.

Dosegljivostno drevo lahko postane neskončno veliko iz dveh razlogov: če se neka označitev omrežja ponavlja ali če Petrijevo omrežje ni omejeno.

Petrijevo omrežje je neomejeno, kadar število žetonov v njegovih mestih lahko naraste preko vseh meja.

Da bi bilo dosegljivostno drevo uporabno sredstvo za analizo Petrijevih omrežij, mora ostati končno veliko.

Pri gradnji dosegljivostnega drevesa zato odstranimo ponavljajoče se označitve.

Če na poti od začetne označitve $\vec{\mu}_0$ do označitve $\vec{\mu}$ obstaja označitev $\vec{\mu}'$, ki je identična označitvi $\vec{\mu}$, potem vozlišče drevesa, ki pripada označitvi $\vec{\mu}$, označimo kot vozlišče dvojnik in se v njegovi smeri gradnja drevesa zaključi.

Z namenom, da bi tudi v primeru neomejenih Petrijevih omrežij ohranili končno velikost dosegljivostnega drevesa, vpeljemo znak ω za neskončno število žetonov.

Če na poti od začetne označitve $\vec{\mu}_0$ do označitve $\vec{\mu}$ obstaja označitev $\vec{\mu}'$, katere komponente so manjše ali enake istoležnim komponentam označitve $\vec{\mu}$, potem z znakom ω nadomestimo tiste komponente označitve $\vec{\mu}$, ki so strogo večje od istoležnih komponent označitve $\vec{\mu}'$.

Listi dosegljivostnega drevesa so dveh vrst:

- *končna vozlišča* (angl. terminal nodes) so vozlišča, ki pripadajo takšnim označitvam omrežja, v katerih ni noben prehod omogočen,
- *vozlišča dvojniki* (angl. duplicate nodes) so vozlišča, ki predstavljajo označitve omrežja, ki so se v drevesu že pojavile.

Dosegljivostno drevo Petrijevega omrežja z začetno označitvijo $\vec{\mu}_0$ zgradimo z naslednjim algoritmom (T. Murata, 1989)*:

*Murata T.: Petri Nets: Properties, Analysis and Applications, Proc. IEEE, Vol. 77, No. 4, 541-580, 1989.

Postopek

1. korak Generiraj korensko vozlišče dosegljivostnega drevesa, ga označi z začetno označitvijo $\vec{\mu}_0$ in poimenuj z imenom *novi*.
2. korak Dokler obstaja še kakšno vozlišče z imenom *novi*, ponavljaj:
   - 2.1. Izberi vozlišče drevesa, ki je poimenovano z imenom *novi* in ima označitev $\vec{\mu}$;
   - 2.2. Če je označitev $\vec{\mu}$ identična kakšni označitvi na poti od korenskega vozlišča do vozlišča z označitvijo $\vec{\mu}$, potem to vozlišče poimenuj z imenom *dvojnik* in skoči na korak 2.1.
   - 2.3. Če je označitev $\vec{\mu}$ takšna, da noben prehod Petrijevega omrežja ni omogočen, potem to vozlišče poimenuj z imenom *končno* in skoči na korak 2.1.

2.4. Dokler obstaja še kakšen omogočen prehod t Petrijevega omrežja pri označitvi $\vec{\mu}$, ponavljaj:

2.4.1. Določi novo označitev omrežja $\vec{\mu}'$ s proženjem prehoda t.

2.4.2. Če na poti od korenskega vozlišča do vozlišča z označitvijo $\vec{\mu'}$ obstaja vozlišče z označitvijo $\vec{\mu''}$, tako da je $\mu'(p_i) \geq \mu''(p_i)$ za $\forall p_i \in P$ in $\vec{\mu'} \neq \vec{\mu''}$, potem nadomesti $\mu'(p_i)$ z ω za vse p_i, za katere velja $\mu'(p_i) > \mu''(p_i)$.

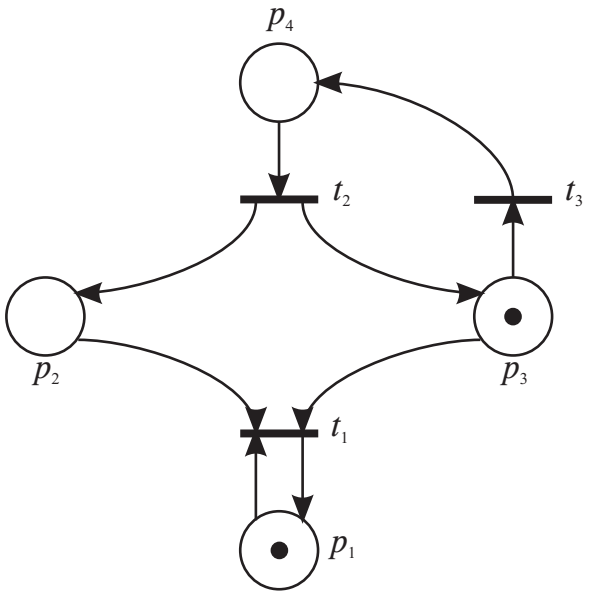
2.4.3. Za novo označitev drevesa $\vec{\mu}'$ generiraj novo vozlišče drevesa, ga poveži z vozliščem, ki je označeno s $\vec{\mu}$, in ga poimenuj z oznako *novi*, povezavi pa pridruži ime omogočenega prehoda t, katerega proženje je pretvorilo označitev $\vec{\mu}$ v označitev $\vec{\mu}'$.

2.4.4. Skoči na korak 2.4.1.

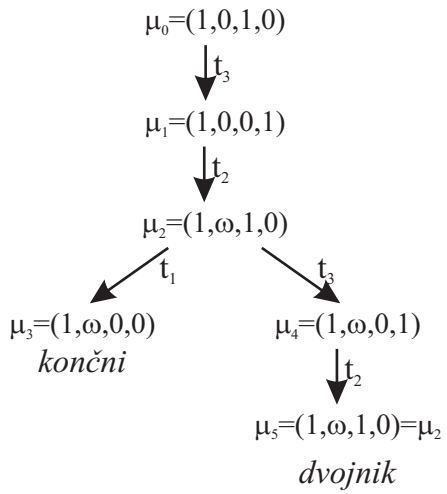
2.5. Skoči na korak 2.1.

Konec postopka.

Zgled 6 Dosegljivostno drevo Petrijevega omrežja s slike 8 pri začetni označitvi $\vec{\mu}_0 = (1, 0, 1, 0)$ je prikazano na sliki 9.



Slika 8: Petrijevo omrežje.



Slika 9: Dosegljivostno drevo.

Opomba: Vektor označitve $\vec{\mu}_2 = (1, 1, 1, 0)$ se zaradi izpolnjevanja pogoja v koraku 2.4.2. glede na vektor $\vec{\mu}_0 = (1, 0, 1, 0)$ spremeni v $\vec{\mu}_2 = (1, \omega, 1, 0)$.

■

Metoda dosegljivostnih dreves pokaže vrsto lastnosti Petrijevih omrežij:

- Petrijevo omrežje je *omejeno* natanko tedaj, če označitev v nobenem vozlišču dosegljivostnega drevesa ne vsebuje znaka ω.
- Petrijevo omrežje je *varno* natanko tedaj, če so v vseh vozliščih dosegljivostnega drevesa komponente vektorjev označitev le 0 in 1.
- Nek prehod v Petrijevem omrežju je *mrtev*, če se nikjer ne pojavi kot oznaka veje dosegljivostnega drevesa.
- Če je označitev $\vec{\mu}$ *dosegljiva* iz začetne označitve $\vec{\mu}_0$, potem obstaja vozlišče dosegljivostnega drevesa z označitvijo $\vec{\mu}'$, tako da je $\vec{\mu} \leq \vec{\mu}'$.

Gradnja in preiskovanje dosegljivostnega drevesa poteka po pristopu *najprej v širino*, kar pomeni, da drevo nastaja po nivojih.

Rast drevesa se zaključi v njegovih listih, ki so poimenovani kot *končno* oziroma *dvojnik*.

Na $(i+1)$-vem nivoju drevesa so vozlišča z označitvami, ki so neposredno generirane s proženjem vseh omogočenih prehodov iz vozlišč na i-tem nivoju, ki so bila poimenovana z oznako *novi*.

Velikost dosegljivostnega drevesa, to je število vozlišč in nivojev, je odvisna od velikosti Petrijevega omrežja in od njegove začetne označitve.

Stroji stanja kot posebna vrsta Petrijevih omrežij, v katerih ima vsak prehod natanko eno vhodno in izhodno mesto, imajo dosegljivostno drevo, v katerem se nikoli ne pojavi znak ω.

Stroj stanja je torej *omejeno* Petrijevo omrežje, kar pomeni, da v njem število žetonov ostane omejeno.

Poleg tega je stroj stanja tudi *ohranljivo* Petrijevo omrežje, kar pa pomeni, da v njem število žetonov ostaja celo konstantno.

Viri

N. Pavešić, *Razpoznavanje vzorcev: uvod v analizo in razumevanje vidnih in slušnih signalov*, 3. popravljena in dopolnjena izd., 2 zv., (Dodatek C) Založba FE in FRI, 2012.

Murata T.: Petri Nets: *Properties, Analysis and Applications*, Proc. IEEE, Vol. 77, No. 4, 541-580, 1989.

Peterson J.L.: *Petri Net Theory and the Modeling of Systems*, Prentice-Hall, Inc., Englewood Cliffs, 1981.

Parkelj Mojca: *Z znanjem podprt sistem umetnega vida*, magistrska naloga, 3. poglavje, FE Ljubljana, 1996.

Vprašanja

Podajte osnovno definicijo Petrijevega omrežja.

Kaj opisujemo oz. predstavljamo s Petrijevimi omrežji?

Kakšne vrste Petrijevo omrežje je *stroj stanj*?

Kako poteka izvajanje Petrijevega omrežja?

Kateri dve pravili določata izvajanje Petrijevega omrežja?

Na čem temelji invariantna metoda analize Petrijevega omrežja?

Na čem temelji metoda dosegljivostnih dreves?

Obrazec za prikaz znanja, ki temelji na teoriji Petrijevih omrežij

Vsebina

Obrazec za prikaz znanja, ki temelji na teoriji Petrijevih omrežij, krajše obrazec KRP (angl. **K***nowledge* **R***epresentation Scheme Based on* **P***etri Net Theory*) omogoča prikaz *deklarativnega* znanja o objektih, njihovih lastnostih in medsebojnih odnosih, pa tudi *proceduralnega* znanja, to je opisov postopkov za reševanje problemov. Oblikujemo ga tako, da posameznim elementom Petrijevega omrežja dodamo pomen:

- *mestom* Petrijevega omrežja pridružimo pojme, ki se nanašajo na objekte ali dejstva,
- *prehodom* pa pojme, ki se nanašajo na odnose med objekti oziroma dejstvi.

Definicija 1 Splošni obrazec za prikaz znanja KRP *je sedmerka* $KRP = (P, T, I, O, \mu, \alpha, \beta)$, *kjer so* P, T, I, O, μ *elementi Petrijevega omrežja, in je potem:*

$P = \{p_1, p_2, ..., p_m\}$ *končna množica mest,*

$T = \{t_1, t_2, ..., t_n\}$ *končna množica prehodov,* $P \cap T = \emptyset$,

$I: \ T \to P^\infty$ *vhodna funkcija – preslikava iz množice prehodov v vrečo mest,*

$O: \ T \to P^\infty$ *izhodna funkcija – preslikava iz množice prehodov v vrečo mest,*

$\mu: \ P \to \mathbb{N}$ *funkcija označitve – preslikava iz množice mest v množico nenegativnih celih števil,*

$\alpha: \ P \to D$ *bijektivna funkcija, ki vsakemu mestu iz* P *pridruži pojem, ki predstavlja objekt ali dejstvo,*

β : $T \to \Sigma$ *surjektivna funkcija, ki vsakemu prehodu iz* T *pridruži pojem, ki opisuje odnos med objekti oziroma dejstvi,*

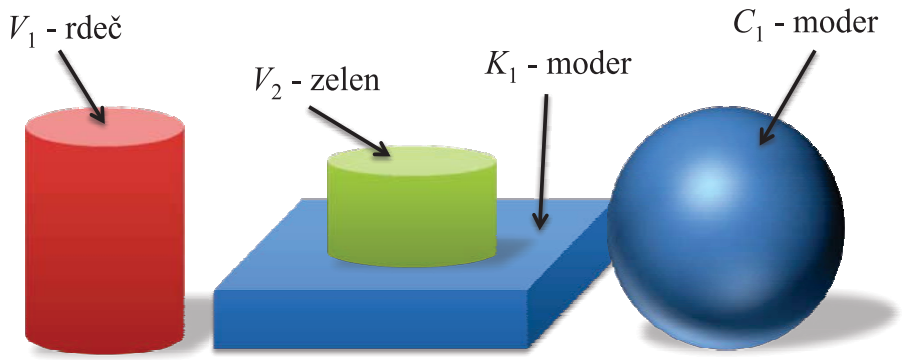
D *množica pojmov, ki se nanašajo na objekte ali dejstva,*

Σ *množica pojmov, ki se nanašajo na odnose med objekti oziroma dejstvi.*

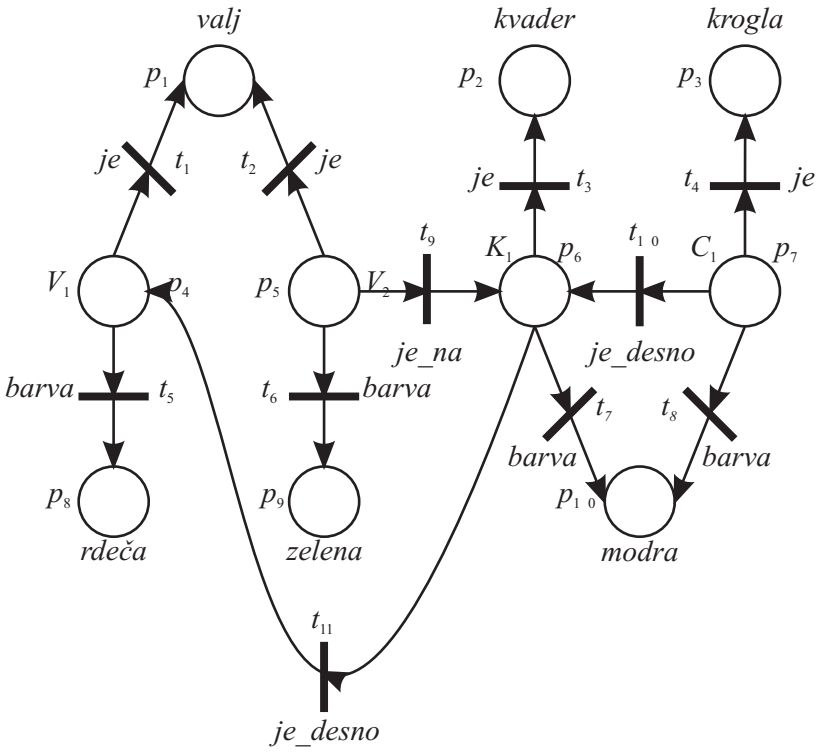
S funkcijama α in β dobijo mesta in prehodi Petrijevega omrežja pomen. Inverzni funkciji α^{-1} in β^{-1} uporabimo, kadar želimo pojmom iz D in Σ prirediti mesta oziroma prehode Petrijevega omrežja.

Funkcija α je bijektivna, zato je možno s funkcijo α^{-1} vsakemu pojmu iz D prirediti mesto v Petrijevem omrežju. O inverzni funkciji β^{-1} ne moremo govoriti, če le-ta ni bijektivna.

Zgled 1 Na sliki 1 je preprost prizor, ki sestoji iz dveh valjev, kvadra in krogle. Poznamo tudi barve teles: valj V_1 je rdeč, valj V_2 je zelen, kvader K_1 in krogla C_1 sta modra.



Slika 1: Preprost prizor.



Slika 2: Obrazec KRP preprostega prizora.

Obrazec KRP je ponazorjen na sliki 2 in je definiran s:

$$
\begin{aligned}
P &= \{p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9, p_{10}\},\\
T &= \{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9, t_{10}, t_{11}\},\\
D &= \{\text{valj}, \text{kvader}, \text{krogla}, V_1, V_2, K_1, C_1, \text{rdeča}, \text{zelena}, \text{modra}\},\\
\Sigma &= \{\text{je}, \text{barva}, \text{je\_na}, \text{je\_desno}\},
\end{aligned}
$$

$$
\begin{array}{ll}
I(t_1) = \{p_4\} & O(t_1) = \{p_1\}\\
I(t_2) = \{p_5\} & O(t_2) = \{p_1\}\\
I(t_3) = \{p_6\} & O(t_3) = \{p_2\}\\
I(t_4) = \{p_7\} & O(t_4) = \{p_3\}\\
I(t_5) = \{p_4\} & O(t_5) = \{p_8\}\\
I(t_6) = \{p_5\} & O(t_6) = \{p_9\}\\
I(t_7) = \{p_6\} & O(t_7) = \{p_{10}\}\\
I(t_8) = \{p_7\} & O(t_8) = \{p_{10}\}\\
I(t_9) = \{p_5\} & O(t_9) = \{p_6\}\\
I(t_{10}) = \{p_7\} & O(t_{10}) = \{p_6\}\\
I(t_{11}) = \{p_6\} & O(t_{11}) = \{p_4\}
\end{array}
$$

$\alpha : p_1 \rightarrow$ valj
$\alpha : p_2 \rightarrow$ kvader
$\alpha : p_3 \rightarrow$ krogla
$\alpha : p_4 \rightarrow V_1$
$\alpha : p_5 \rightarrow V_2$
$\alpha : p_6 \rightarrow K_1$
$\alpha : p_7 \rightarrow C_1$
$\alpha : p_8 \rightarrow$ rdeča
$\alpha : p_9 \rightarrow$ zelena
$\alpha : p_{10} \rightarrow$ modra

$\beta : t_1 \rightarrow$ je
$\beta : t_2 \rightarrow$ je
$\beta : t_3 \rightarrow$ je
$\beta : t_4 \rightarrow$ je
$\beta : t_5 \rightarrow$ barva
$\beta : t_6 \rightarrow$ barva
$\beta : t_7 \rightarrow$ barva
$\beta : t_8 \rightarrow$ barva
$\beta : t_9 \rightarrow$ je_na
$\beta : t_{10} \rightarrow$ je_desno
$\beta : t_{11} \rightarrow$ je_desno

$\mu(p_i) = 0$ za $\forall p_i \in P$

Inverzni funkciji α^{-1} in β^{-1} sta potem:

$\alpha^{-1} :$ valj $\rightarrow p_1$
$\alpha^{-1} :$ kvader $\rightarrow p_2$
$\alpha^{-1} :$ krogla $\rightarrow p_3$
$\alpha^{-1} : V_1 \rightarrow p_4$
$\alpha^{-1} : V_2 \rightarrow p_5$
$\alpha^{-1} : K_1 \rightarrow p_6$
$\alpha^{-1} : C_1 \rightarrow p_7$
$\alpha^{-1} :$ rdeča $\rightarrow p_8$
$\alpha^{-1} :$ zelena $\rightarrow p_9$
$\alpha^{-1} :$ modra $\rightarrow p_{10}$

$\beta^{-1} :$ je $\rightarrow \{t_1, t_2, t_3, t_4\}$
$\beta^{-1} :$ barva $\rightarrow \{t_5, t_6, t_7, t_8\}$
$\beta^{-1} :$ je_na $\rightarrow \{t_9\}$
$\beta^{-1} :$ je_desno $\rightarrow \{t_{10}, t_{11}\}$

■

Za sam prikaz odnosov med objekti na prizoru funkcija označitve nima pomena. Svojo vlogo ima šele v postopkih sklepanja z obrazcem KRP.

Množico pojmov D, ki predstavljajo objekte ali dejstva, lahko prikažemo kot unijo treh disjunktnih podmnožic D_1, D_2 in D_3:

$$D = D_1 \cup D_2 \cup D_3,$$

kjer so množice D_1, D_2 in D_3 definirane kot:

D_1 množica pojmov, ki se nanašajo na posamezne predstavnike ali primerke iz področja problema,

D_2 množica pojmov, ki se nanašajo na razrede ali kategorije, katerim pripadajo primerki iz množice D_1 in pa tudi drugi pojmi iz D_2, in predstavljajo višji nivo abstrakcije,

D_3 množica pojmov, ki se nanašajo na notranje lastnosti ali vrednosti teh lastnosti, ki jih imajo elementi iz množic D_1 in D_2.

Podobno tudi množico Σ lahko prikažemo kot unijo treh disjunktnih podmnožic Σ_1, Σ_2 in Σ_3:

$$\Sigma = \Sigma_1 \cup \Sigma_2 \cup \Sigma_3,$$

kjer so množice Σ_1, Σ_2 in Σ_3 definirane kot:

Σ_1 množica pojmov, ki se nanašajo na odnose med pojmi iz množic D_1 in D_2, in se uporabljajo za hierarhično ureditev teh pojmov,

Σ_2 množica pojmov, ki se nanašajo na vrste lastnosti, katerih vrednosti so elementi iz množice D_3,

Σ_3 množica pojmov, ki se nanašajo na odnose med pojmi iz množic D_1 in D_2, in se ne uporabljajo za hierarhično ureditev teh pojmov.

Zgled 2 Množici D in Σ iz **Zgleda 1** lahko razčlenimo na:

$$
\begin{aligned}
D_1 &= \{V_1, V_2, K_1, C_1\} \\
D_2 &= \{\text{valj, kvader, krogla}\} \\
D_3 &= \{\text{rdeča, zelena, modra}\} \\
\Sigma_1 &= \{\text{je}\} \\
\Sigma_2 &= \{\text{barva}\} \\
\Sigma_3 &= \{\text{je\_na, je\_desno}\}
\end{aligned}
$$

■

Osnovne operacije v obrazcu za prikaz znanja KRP

Za obrazec KRP so definirane naslednje operacije (S. Ribarić, 1991):

- superpozicija dveh obrazcev,
- brisanje mesta v obrazcu in
- brisanje prehoda v obrazcu.

Omenjene operacije na obrazcu za prikaz znanja KRP dajejo možnost upravljanja z bazo znanja, to je dodajanje novega znanja in spreminjanje obstoječega znanja.

Superpozicija dveh obrazcev - superpozicija(KRP_a, KRP_b)

Dana sta obrazca za prikaz znanja KRP_a in KRP_b:

$$KRP_a = (P_a, T_a, I_a, O_a, \mu_a, \alpha_a, \beta_a) \quad \text{in}$$
$$KRP_b = (P_b, T_b, I_b, O_b, \mu_b, \alpha_b, \beta_b)$$

pri pogoju $T_a \cap T_b = \emptyset$.

Rezultat superpozicije obeh obrazcev je nov obrazec KRP:

$$KRP = (P, T, I, O, \mu, \alpha, \beta),$$

kjer so:

$P = P_a \cup P_b$,

$T = T_a \cup T_b$,

$D = D_a \cup D_b$,

$\Sigma = \Sigma_a \cup \Sigma_b$,

$$I = I_a \cup I_b \Rightarrow I(t_i) = \begin{cases} I_a(t_i), & \text{če } t_i \in T_a \\ I_b(t_i), & \text{če } t_i \in T_b, \end{cases}$$

$$O = O_a \cup O_b \Rightarrow O(t_i) = \begin{cases} O_a(t_i), & \text{če } t_i \in T_a \\ O_b(t_i), & \text{če } t_i \in T_b, \end{cases}$$

$$\mu(p_i) = \begin{cases} \mu_a(p_i), & \text{če } p_i \in P_a \wedge p_i \notin P_a \cap P_b \\ \mu_b(p_i), & \text{če } p_i \in P_b \wedge p_i \notin P_a \cap P_b \\ \min\{\mu_a(p_i), \mu_b(p_i)\}, & \text{če } p_i \in P_a \cap P_b, \end{cases}$$

$$\alpha = \alpha_a \cup \alpha_b \Rightarrow \alpha(p_i) = \begin{cases} \alpha_a(p_i), & \text{če } p_i \in P_a \\ \alpha_b(p_i), & \text{če } p_i \in P_b, \end{cases}$$

$$\beta = \beta_a \cup \beta_b \Rightarrow \beta(t_i) = \begin{cases} \beta_a(t_i), & \text{če } t_i \in T_a \\ \beta_b(t_i), & \text{če } t_i \in T_b. \end{cases}$$

Brisanje mesta v obrazcu – briši(d_j)

Dan je obrazec za prikaz znanja KRP:

$$KRP = (P, T, I, O, \mu, \alpha, \beta).$$

Iz obrazca KRP želimo zbrisati mesto $p_i \in P$, katerega lastnost je d_j, to je:

$$\alpha^{-1}: \quad d_j \to p_i.$$

Rezultat brisanja mesta p_i iz obrazca KRP je nov obrazec KRP':

$$KRP' = (P', T', I', O', \mu', \alpha', \beta'),$$

kjer so:

$P' = P - \{p_i\}$,
$D' = D - \{d_j\}$,
$I'(t_k) = I(t_k) - \{p_i\}$ za $\forall t_k \in T$, za katerega $p_i \in I(t_k)$,
$O'(t_k) = O(t_k) - \{p_i\}$ za $\forall t_k \in T$, za katerega $p_i \in O(t_k)$,

$(I'(t_k) = \emptyset \wedge O'(t_k) = \emptyset) \Rightarrow t_k$ je osamljen prehod in zato:

$$T' = T - \{t_k\} \quad \text{in} \quad \Sigma' = \Sigma - \{\beta(t_k)\},$$

$\mu'(p_k) = \mu(p_k)$ za $p_k \in P'$,
$\alpha'(p_k) = \alpha(p_k)$ za $p_k \in P'$ in
$\beta'(t_k) = \beta(t_k)$ za $t_k \in T'$.

Brisanje prehoda v obrazcu – briši(t_j)

Dan je obrazec za prikaz znanja KRP:

$$KRP = (P, T, I, O, \mu, \alpha, \beta).$$

Iz obrazca KRP želimo zbrisati prehod $t_j \in T$. Rezultat brisanja prehoda t_j iz obrazca KRP je nov obrazec KRP':

$$KRP' = (P', T', I', O', \mu', \alpha', \beta'),$$

kjer je:

$T' = T - \{t_j\}$,

$(\beta(t_j) \neq \beta(t_i) \quad \text{za} \quad \forall t_i \in T') \Rightarrow \Sigma' = \Sigma - \{\beta(t_j)\}$,

$I'(t_k) = I(t_k) \quad \text{za } \forall t_k \in T'$,

$O'(t_k) = O(t_k) \quad \text{za } \forall t_k \in T'$,

$(p_k \notin I'(t_k) \wedge p_k \notin O'(t_i) \quad \text{za} \quad \forall t_k, t_i \in T') \Rightarrow p_k$, je osamljeno mesto in zato:

$$P' = P - \{p_k\} \quad \text{in} \quad D' = D - \{\alpha(p_k)\},$$

$\mu'(p_k) = \mu(p_k) \quad \text{za} \quad p_k \in P'$,

$\alpha'(p_k) = \alpha(p_k) \quad \text{za} \quad p_k \in P'$ in

$\beta'(t_k) = \beta(t_k) \quad \text{za} \quad t_k \in T'$.

Hierarhične lastnosti obrazca za predstavitev znanja KRP

S Petrijevimi omrežji je možno hierarhično modelirati sisteme, kar pomeni, da je sisteme možno predstaviti na različnih nivojih splošnosti znanja.

Tako lahko neko podomrežje Petrijevega omrežja nadomestimo s samo enim mestom ali prehodom, ki predstavlja višji nivo splošnosti. V tem primeru gre za proces *posploševanja* (angl. process of abstraction).

Lahko pa tudi neko mesto ali prehod Petrijevega omrežja zamenjamo z nekim podomrežjem, ki predstavlja nižji nivo abstrakcije. Tedaj pa gre za proces *plemenitenja* (angl. process of refinement).

Omenjena lastnost Petrijevih omrežij ponuja možnost hierarhičnega predstavljanja znanja v obrazcu za prikaz znanja KRP. To pomeni, da lahko:

- vsako mesto $p_i \in P$ v KRP, ki predstavlja neko dejstvo ali objekt, nadomestimo s podobrazcem KRP', ki bolj podrobno prikazuje znanje o tem dejstvu ali objektu,
- vsak prehod $t_j \in T$ v KRP, ki predstavlja nek odnos med dvema dejstvoma oziroma objektoma, nadomestimo s podobrazcem KRP', ki bolj podrobno prikazuje znanje o tem odnosu.

Proces posploševanja z mestom – nadomesti (KRP', d_j)

Dan je obrazec za prikaz znanja KRP:

$$KRP = (P, T, I, O, \mu, \alpha, \beta).$$

V obrazcu KRP nadomestimo podobrazec KRP' z mestom p_i, katerega lastnost je d_j:

$$\alpha^{-1} : d_j \to p_i.$$

Mesto p_i, ki nadomesti podobrazec KRP', pridobi modificirano vhodno in izhodno funkcijo:

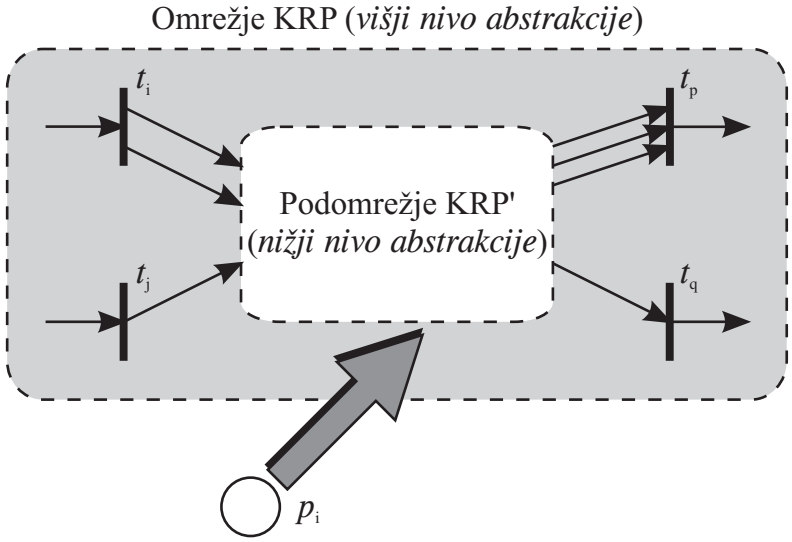
$$\begin{aligned} I : P \to T^\infty &\quad \to \quad I(p_i) := I'(KRP') \\ O : P \to T^\infty &\quad \to \quad O(p_i) := O'(KRP'), \end{aligned}$$

kjer sta $I'(KRP')$ in $O'(KRP')$ modificirani vhodna in izhodna funkcija podomrežja KRP':

$$\begin{aligned} I' : KRP' &\to T^\infty \\ O' : KRP' &\to T^\infty. \end{aligned}$$

$I'(KRP')$ je določena z vrečo prehodov, ki vodijo iz obrazca KRP na nižji nivo posplošitve (abstrakcije) v KRP', $O'(KRP')$ pa z vrečo prehodov, ki vodijo iz KRP' na višji nivo posplošitve v KRP.

Zgled 3 Primer posplošitve podomrežja KRP' z mestom p_i je prikazan na sliki 3. Za KRP' s slike 3 velja:



Slika 3: Posplošitev podomrežja KRP' z mestom.

$$\begin{aligned} I'(KRP') &= \{t_i, t_i, t_j\} \\ O'(KRP') &= \{t_p, t_p, t_p, t_q\}. \end{aligned}$$

Proces plemenitenja mesta – nadomesti(d_j, KRP')

Dan je obrazec za prikaz znanja KRP:

$$KRP = (P, T, I, O, \mu, \alpha, \beta).$$

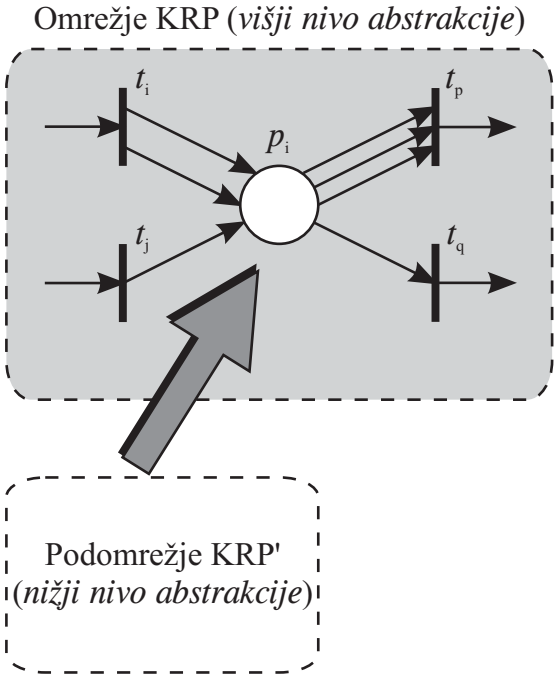
V obrazcu KRP nadomestimo mesto p_i, katerega lastnost je d_j:

$$\alpha^{-1} : d_j \to p_i$$

s podobrazcem KRP'. Podobrazec KRP', ki nadomesti mesto p_i, pridobi modificirano vhodno in izhodno funkcijo:

$$\begin{aligned} I' : KRP' \to T^\infty &\Rightarrow I'(KRP') := I(p_i) \\ O' : KRP' \to T^\infty &\Rightarrow O'(KRP') := O(p_i). \end{aligned}$$

Zgled 4 Primer plemenitenja mesta p_i s podomrežjem KRP' je ponazorjen na sliki 4. ∎



Slika 4: Plemenitenje mesta s podomrežjem KRP'.

Postopki sklepanja v obrazcu za predstavitev znanja KRP

Postopki sklepanja v obrazcu za prikaz znanja KRP so:

- generiranje odgovorov na vprašanja,
- postopek iskanja presečišč aktivnosti,
- dedovanje,
- razpoznavanje.

Generiranje odgovorov na vprašanja

Ena od pomembnih lastnosti omrežja za prikaz znanja KRP je, da zna priklicati svoje shranjeno eksplicitno ali implicitno znanje in odgovoriti na vprašanja uporabnika. Vprašanja poizvedujejo po objektih oziroma dejstvih in njihovih medsebojnih odnosih. Predpostavimo, da uporabnik postavlja vprašanja v skladu s sintakso programskega jezika *PROLOG*. Vprašanje: "*Kaj je (***X***) v odnosu* **relacija** *z objektom ali dejstvom* **argument**?" bi potem imelo obliko:

$$? - \text{relacija}(X, \text{argument}).$$

Odgovor na postavljeno vprašanje omrežje KRP generira z naslednjim algoritmom:

Postopek

1. korak

$$p_k = \alpha^{-1}(\text{argument})$$
$$\tau = \{t_j \in T | \beta(t_j) = \text{relacija}\}$$

Če p_k ne obstaja ali $\tau = \emptyset$, generiraj odgovor "NE".

2. korak Za $\forall t_j \in \tau$ določi množico $M = \{p_i \in P | p_k \in O(t_j) \wedge p_i \in I(t_j)\}$.
Če $M = \emptyset$, generiraj odgovor "NE".

3. korak Za $p_i \in M$ generiraj odgovor $X = \alpha(p_i)$.

Konec postopka.

Zgled 5 Zanima nas, kateri objekt (če sploh kakšen) stoji na objektu K_1 na prizoru, ki je prikazan na sliki 1. Vprašanje zapišemo v formalni obliki:

$$? - \text{je\_na}(X, K_1).$$

Sledimo algoritmu na str. 28:

Postopek

1. korak

$$p_k = p_6, \quad \text{ker } \alpha^{-1}(K_1) = p_6$$
$$\tau = \{t_9\}, \quad \text{ker } \beta(\text{je\_na}) = t_9$$

2. korak $M = \{p_5\}, \quad \text{ker } p_6 \in O(t_9) \wedge p_5 \in I(t_9)\}$

3. korak $X = V_2, \quad \text{ker } \alpha(p_5) = V_2$

Konec postopka.

Sistem generira odgovor: $X = V_2$.

Bolj zapleteno vprašanje: "*Kaj je (***X***) v odnosu* $\mathbf{relacija}_2$ *z objektom ali dejstvom* **argument** *in ima lastnost* $\mathbf{relacija}_1$?" pa bi potem imelo obliko:

$$? - \text{relacija}_1(X), \text{relacija}_2(X, \text{argument}).$$

kjer vejica pomeni konjunkcijo izjav.

Odgovor na zadnje postavljeno vprašanje omrežje KRP generira z naslednjim algoritmom:

Postopek

1. korak

$$p_l = \alpha^{-1}(\text{argument})$$
$$\tau = \{t_j \in T | \beta(t_j) = \text{relacija}_1\}$$

   Če p_l ne obstaja ali $\tau = \emptyset$, generiraj odgovor "NE".

2. korak Za $\forall t_j \in \tau$ določi množico $M = \{p_k \in P | p_k \in O(t_j) \wedge p_k \in I(t_j)\}$.
   Če $M = \emptyset$, generiraj odgovor "NE".

3. korak

$$\tau' = \{t_j \in T \mid \beta(t_j) = \text{relacija}_2\}$$

   Če $\tau' = \emptyset$, generiraj odgovor "NE".

4. korak Za $\forall t_j \in \tau'$, določi množico $M' = \{p_k \in M | p_l \in O(t_j) \wedge p_k \in I(t_j)\}$.
   Če $M' = \emptyset$, generiraj odgovor "NE".

5. korak Za $\forall p_k \in M'$ generiraj odgovor $X = \alpha(p_k)$.

Konec postopka.

Postopek iskanja presečišč aktivnosti

Postopek *iskanja presečišč aktivnosti* je postopek sklepanja, ki ga uporabljamo za iskanje medsebojnih odnosov med dvema pojmoma.

Postopek temelji na gradnji dveh dosegljivostnih dreves za dve različni začetni označitvi omrežja KRP, ki ustrezata danima pojmoma, in na iskanju identičnih vozlišč v teh dveh drevesih.

Pravimo, da je najdeno presečišče aktivnosti, če s primerjavo obeh dosegljivostnih dreves najdemo vozlišča z enako označitvijo.

Če sledimo zaporedju proženih prehodov in spremembam označitve omrežja v obeh dosegljivostnih drevesih od korenov do presečišč aktivnosti, lahko določimo vse relacije, ki povezujejo dana pojma.

Naj bosta $d_1 \in D$ in $d_2 \in D$ dva pojma, katerih medsebojni odnos iščemo. Postopek iskanja presečišč aktivnosti je opisan z naslednjim algoritmom:

$Postopek$

1. korak

$$p_i = \alpha^{-1}(d_1),$$
$$p_j = \alpha^{-1}(d_2).$$

2. korak Naj bo $\overrightarrow{\mu}_{01}$ začetna označitev omrežja KRP, ki ustreza pojmu d_1, $\overrightarrow{\mu}_{02}$ pa začetna označitev, ki ustreza pojmu d_2:

$$\mu_{01}(p_k) = \begin{cases} 1, \text{ če } k = i \\ 0, \text{ če } k \neq i, \ k = 1, 2, ..., m, \end{cases}$$

$$\mu_{02}(p_k) = \begin{cases} 1, \text{ če } k = j \\ 0, \text{ če } k \neq j, \ k = 1, 2, ..., m. \end{cases}$$

3. korak Generiraj dosegljivostni drevesi RT_1 in RT_2 omrežja KRP za začetni označitvi $\overrightarrow{\mu}_{01}$ in $\overrightarrow{\mu}_{02}$.

4. korak Vozlišča dosegljivostnih dreves RT_1 in RT_2, ki se ujemajo, so presečišča aktivnosti.

5. korak Vsa zaporedja vozlišč dosegljivostnih dreves RT_1 in RT_2, od korenov pa do presečišč aktivnosti, skupaj s pripadajočimi povezavami (in s pridruženimi prehodi) določajo relacije med danima pojmoma d_1 in d_2.

6. korak V vsakem zaporedju vozlišč od korenov dreves RT_1 in RT_2 pa do presečišč aktivnosti določi:

   - vsakemu vozlišču v zaporedju priredi:

$$d_j = \alpha(p_i) \quad \text{za} \quad i \in \{1, 2, ..., m\} \quad \text{in} \quad \mu(p_i) \neq 0,$$

   - vsaki povezavi v zaporedju priredi:

$$\sigma_s = \beta(t_k) \quad \text{za} \quad k \in \{1, 2, ..., m\}.$$

7. korak Za obe dosegljivostni drevesi RT_1 in RT_2 tvori zaporedje lastnosti mest in prehodov:

$$d_j \sigma_s d_1 \sigma_1 ...$$

   Množica vseh takih zaporedij lastnosti določa odnose med danima pojmoma d_1 in d_2.

Konec postopka.

Zgled 6 Zanima nas, v kakšnem odnosu sta objekta K_1 in V_2 na prizoru, ki je prikazan na sliki 1.

$$d_1 = K_1 \quad \text{in} \quad d_2 = V_2$$

Sledimo algoritmu na str. 31:

Postopek

1. korak

$$\begin{aligned} p_i &= p_6, \quad \text{ker} \ \alpha^{-1}(K_1) = p_6, \\ p_j &= p_5, \quad \text{ker} \ \alpha^{-1}(V_2) = p_5. \end{aligned}$$

2. korak Začetni označitvi:

$$\begin{aligned} \overrightarrow{\mu}_{01} &= (0,0,0,0,0,1,0,0,0,0), \\ \overrightarrow{\mu}_{02} &= (0,0,0,0,1,0,0,0,0,0). \end{aligned}$$

3. korak Dosegljivostni drevesi RT_1 in RT_2 za začetni stanji $\overrightarrow{\mu}_{01}$ in $\overrightarrow{\mu}_{02}$ sta prikazani na sliki 5.

4. korak Poiščemo presečišča aktivnosti, to je vozlišča dosegljivostnih dreves RT_1 in RT_2, ki se ujemajo:

$$(n_1, m_4),\ (n_2, m_5),\ (n_3, m_6),\ (n_4, m_7),\ (n_5, m_8),\ (n_6, m_9).$$

5. korak Poiščemo vsa zaporedja vozlišč od korenov pa do presečišč aktivnosti:

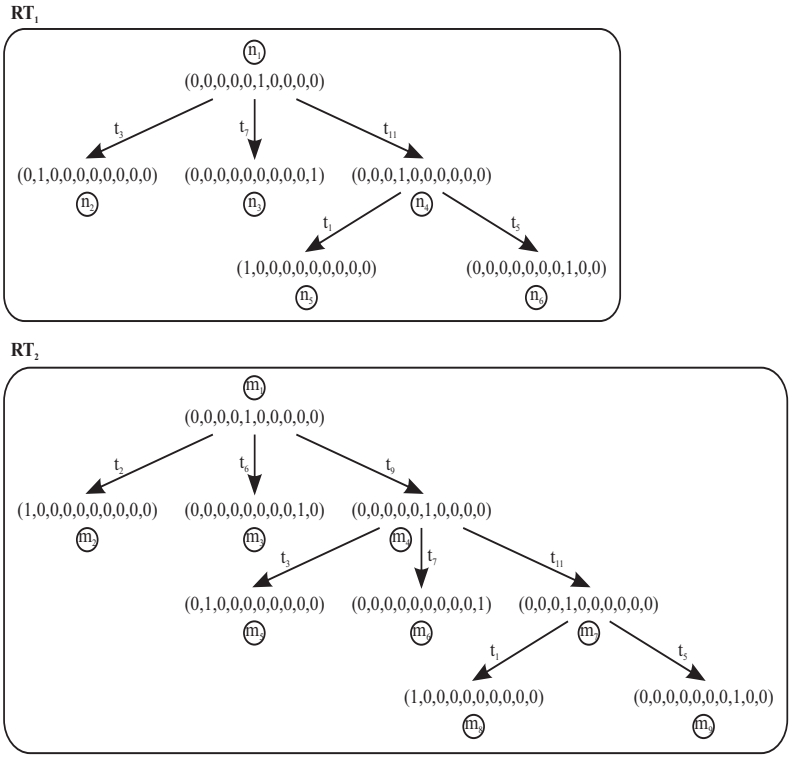
$(RT_1 :\ \underline{n}_1)$	$(RT_2 :\ m_1\, t_9\, \underline{m}_4)$
$(RT_1 :\ n_1\, t_3\, \underline{n}_2)$	$(RT_2 :\ m_1\, t_9\, m_4\, t_3\, \underline{m}_5)$
$(RT_1 :\ n_1\, t_7\, \underline{n}_3)$	$(RT_2 :\ m_1\, t_9\, m_4\, t_7\, \underline{m}_6)$
$(RT_1 :\ n_1\, t_{11}\, \underline{n}_4)$	$(RT_2 :\ m_1\, t_9\, m_4\, t_{11}\, \underline{m}_7)$
$(RT_1 :\ n_1\, t_{11}\, n_4\, t_1\, \underline{n}_5)$	$(RT_2 :\ m_1\, t_9\, m_4\, t_{11}\, m_7\, t_1\, \underline{m}_8)$
$(RT_1 :\ n_1\, t_{11}\, n_4\, t_5\, \underline{n}_6)$	$(RT_2 :\ m_1\, t_9\, m_4\, t_{11}\, m_7\, t_5\, \underline{m}_9)$

6. korak Poiščemo pomene vozlišč in povezav dosegljivostnih dreves.

7. korak Odgovor algoritma:

$(RT_1 : K_1)$	$(RT_2 : V_2$ je_na $K_1)$
$(RT_1 : K_1$ je kvader$)$	$(RT_2 : V_2$ je_na K_1 je kvader$)$
$(RT_1 : K_1$ barva modra$)$	$(RT_2 : V_2$ je_na K_1 barva modra$)$
$(RT_1 : K_1$ je_desno $V_1)$	$(RT_2 : V_2$ je_na K_1 je_desno $V_1)$
$(RT_1 : K_1$ je_desno V_1 je valj$)$	$(RT_2 : V_2$ je_na K_1 je_desno V_1 je valj$)$
$(RT_1 : K_1$ je_desno V_1 barva rdeča$)$	$(RT_2 : V_2$ je_na K_1 je_desno V_1 barva rdeča

Konec postopka.



Slika 5: Dosegljivostni drevesi RT_1 in RT_2.

■

Dedovanje

Dedovanje je oblika sklepanja v obrazcu za prikaz znanja KRP, s pomočjo katere dosežemo znanje, ki ni eksplicitno shranjeno v obstoječi bazi znanja.

Hierarhična strukturiranost baze znanja omogoča, da se določene skupne lastnosti več posameznih pojmov izločijo in pripišejo nekemu drugemu bolj splošnemu pojmu.

Na ta način se izoblikuje hierarhija pojmov, v kateri je omenjeni bolj splošni pojem *prednik* vseh posameznih pojmov, ki imajo skupne lastnosti.

Posamezni pojmi pa so potem *nasledniki* bolj splošnega pojma.

S hierarhično strukturiranostjo baze znanja dosežemo bolj ekonomično shranjevanje znanja in se izognemo večkratnemu navajanju istih lastnosti posameznih pojmov.

Dedovanje definiramo kot proces določanja lastnosti danega pojma $d \in D$, pri čemer ne iščemo samo lastnosti, ki so neposredno povezane s pojmom d, pač pa tudi lastnosti tistih pojmov, ki ležijo višje v hierarhiji pojmov.

Postopek dedovanja v obrazcu za prikaz znanja KRP temelji na izgradnji drevesa *dedovanja*.

Drevo dedovanja je podobno dosegljivostnemu drevesu, pri čemer so lahko njegovi listi poimenovani tudi z oznako *zamrznjeno vozlišče* (angl. frozen node).

Zamrznjena vozlišča drevesa dedovanja pripadajo takšnim označitvam omrežja KRP, pri katerih dospeli žeton v neko mesto v tem mestu zamrzne.

Zamrznjeni žeton pa ne dovoljuje proženja sicer omogočenih prehodov iz tega mesta. Zamrznejo tisti žetoni, ki nastanejo s proženjem prehodov, katerih lastnosti pripadajo množicama Σ_2 in Σ_3, torej opisujejo nehierarhične odnose med pojmi in njihove lastnosti.

Gradnja drevesa dedovanja se prične v korenskem vozlišču, ki pripada pojmu iz množice D, katerega dedne lastnosti iščemo. V korenu drevesa dedovanja se prožijo vsi omogočeni prehodi, katerih lastnosti so elementi množic Σ_1, Σ_2 in Σ_3, in pri tem v njihovih izhodnih mestih generirani žetoni zamrznejo, če so nastali s proženjem prehodov, katerih lastnosti pripadajo množicama Σ_2 in Σ_3.

Vozlišča drevesa dedovanja predstavljajo pojme, ki so v nekem dednem odnosu z danim pojmom. Vsako zaporedje vozlišč od korena drevesa dedovanja do kateregakoli lista, ki ni zamrznjen, je pot dedovanja.

Navadno vnaprej določimo, do katere globine naj se tvori drevo dedovanja, in tako odločimo, do katerega prednika naj pojem še deduje lastnosti.

Tako postanejo listi drevesa dedovanja tudi vsa vozlišča, ki pripadajo izbranemu nivoju, do katerega se drevo dedovanja še tvori.

Proces dedovanja za dani pojem $d \in D$ in dano globino dedovanja l v obrazcu KRP opisuje naslednji algoritem:

Postopek

1. korak V omrežju KRP določi mesto p_i, ki pripada danemu pojmu $d \in D$:

$$p_i = \alpha^{-1}(d).$$

2. korak Določi začetno označitev omrežja KRP:

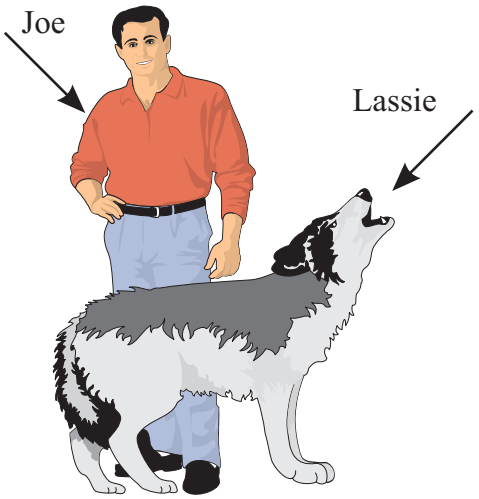
$$\mu_0(p_k) = \begin{cases} 1, \text{ če } k = i \\ 0, \text{ če } k \neq i, \ k = 1, 2, ..., m \end{cases}$$

3. korak Generiraj l nivojev drevesa dedovanja pri začetni označitvi $\vec{\mu}_0$ omrežja KRP.

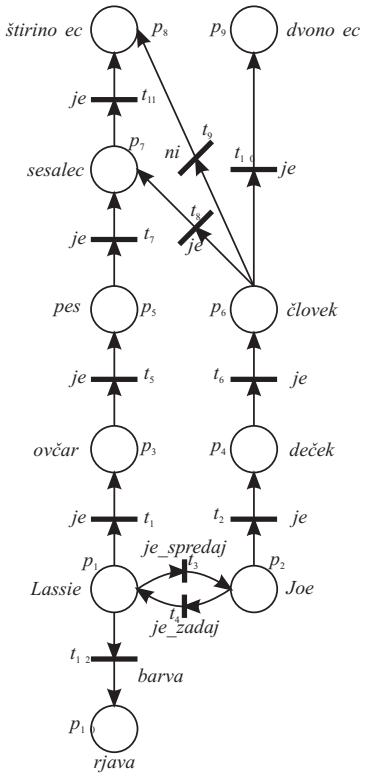
4. korak V drevesu dedovanja poišči vse poti dedovanja, ki vodijo od korena drevesa do listov na l-tem nivoju, ki niso zamrznjeni. Poti dedovanja določajo dedovane lastnosti pojma d.

Konec postopka.

Zgled 7 Prizor na sliki 6 opisuje obrazec KRP na sliki 7.



Slika 6: Prizor iz zgodbe Lassie se vrača.



Slika 7: Omrežje KRP.

S postopkom dedovanja pridobimo znanje o pojmu Lassie $\in D$. Naj bo globina dedovanja $l = 4$.

Sledimo algoritmu na str. 42:

Postopek

1. korak

$$p_i = p_1, \quad \text{ker } \alpha^{-1}(\text{Lassie}) = p_1.$$

2. korak Začetna označitev:

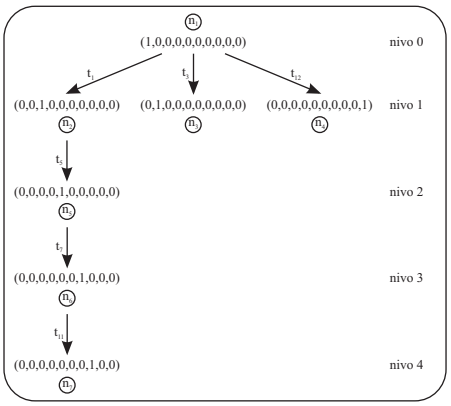
$$\overrightarrow{\mu}_0 = (1, 0, 0, 0, 0, 0, 0, 0, 0, 0).$$

3. korak $l = 4$ nivojev drevesa dedovanja pri začetni označitvi $\overrightarrow{\mu}_0$ omrežja KRP je prikazanih na sliki 8.

4. korak Poiščemo poti dedovanja in jim priredimo njihov pomen:

$(n_1\, t_1\, n_2\, t_5\, n_5\, t_7\, n_6\, t_{11}\, n_7)$	(Lassie je ovčar je pes je sesalec je štirinožec)
$(n_1\, t_3\, n_2)$	(Lassie je_spredaj Joe)
$(n_1\, t_{12}\, n_3)$	(Lassie barva rjava)

Konec postopka.



Slika 8: Drevo dedovanja.

■

V procesu dedovanja lahko pride do protislovij, če

- dani pojem podeduje lastnosti svojega prednika, ki zanj ne veljajo, ker predstavlja izjemo,
- ima dani pojem več prednikov in pride do spornega dedovanja.

Protislovne situacije dedovanja rešujemo s postopkom PIDO (angl. Principle of the Inferential Distance Ordering). Zasnova postopka PIDO je naslednja:

V situaciji protislovnega dedovanja dani pojem deduje lastnosti bližnjega prednika. Pojem A je *bližji* predniku B kakor predniku C natanko tedaj, ko za pojem A obstaja pot dedovanja do prednika C skozi B.

Zgled 8 S postopkom dedovanja bi za obrazec KRP na sliki 7 in globino dedovanja $l = 4$ pridobili naslednje znanje o pojmu Joe $\in D$:

1. Joe je deček je človek je dvonožec.
2. Joe je deček je človek je sesalec je štirinožec.
3. Joe je deček je človek ni štirinožec.
4. Joe je_zadaj Lassie.

Pridobljeno znanje je protislovno, saj si trditve (1) in (2) ter (2) in (3) nasprotujejo. Ni jasno ali je Joe štirinožec ali ne? Problem protislovja reši postopek PIDO, katerega rezultat je naslednje znanje:

1. Joe je deček je človek je dvonožec.
2. Joe je deček je človek je sesalec.
3. Joe je deček je človek ni štirinožec.
4. Joe je_zadaj Lassie.

■

Razpoznavanje

Naloga postopka razpoznavanja v obrazcu za prikaz znanja KRP je poiskati pojem v bazi znanja, ki najbolje ustreza danemu opisu.

Opis iskanega pojma je podan z množico njegovih lastnosti S.

Te lastnosti pa niso nujno samo neposredne lastnosti iskanega pojma, pač pa so lahko tudi podedovane od njegovih prednikov v hierarhiji pojmov.

Problem razpoznavanja v obrazcu opišemo takole:

Dana je množica lastnosti S neznanega pojma X. Množica S, ki ji pravimo tudi *opis pojma* X, je sestavljena iz naslednjih elementov:

$$S = \left\{ s_i : \left\{ \begin{array}{ll} s_i \in D_2 \cup D_3 & \\ s_i \notin D_2 \cup D_3 & \\ s_i = (r,d) & r \in \Sigma,\ d \in D \end{array} \right\} \right\}.$$

- Če $s_i \in D_2 \cup D_3$, potem lastnosti s_i določimo pripadajoče mesto p_k v obrazcu KRP:

$$p_k = \alpha^{-1}(s_i),$$

- če $s_i \notin D_2 \cup D_3$, potem lastnost s_i nima pripadajočega mesta v obrazcu KRP,
- če $s_i = (r,d)$, potem lastnost s_i govori o odnosu (relaciji) r med neznanim pojmom X in pojmom d.

Postopek razpoznavanja v obrazcu za prikaz znanja KRP temelji na inverznem obrazcu $-KRP$, ki ga dobimo z medsebojno zamenjavo vhodne in izhodne funkcije v obrazcu KRP:

$$KRP = (P,T,I,O,\mu,\alpha,\beta) \ \Rightarrow \ -KRP = (P,T,I,O,\mu_R,\alpha,\beta).$$

Graf obrazca $-KRP$ ima v primerjavi z izvirnim grafom obrazca KRP zamenjane smeri vseh povezav.

Postopek razpoznavanja zahteva modifikacijo funkcije označitve:

$$\mu_R : \quad P \to \mathbb{Z}.$$

Funkcija označitve postane preslikava iz množice mest v množico celih števil. Posledica te spremembe je, da imajo mesta sedaj lahko tudi negativno število žetonov. Razlog za uvedbo modificirane funkcije označitve je možnost obstoja izjem v bazi znanja. Z izjemo označujemo pojem, katerega lastnosti so v nasprotju s splošno veljavnimi lastnostmi razreda, kateremu dani pojem pripada:

Človek je sesalec.
Sesalci so štirinožci.
Človek ni štirinožec.

Izjemo v bazi znanja predstavimo s prehodom, ki mu damo nikalen pomen (na primer "*ni*": človek ni štirinožec).

Modifikacija funkcije označitve povzroči tudi modifikacijo pravila proženja prehodov.

Proženja prehoda t_j, ki pripada povezavi izjeme, povzroči zmanjšanje števila žetonov v njegovem izhodnem mestu p_k za 1:

$$\mu'_R(p_k) = \mu_R(p_k) - 1,$$

kjer je $\mu_R(p_k)$ število žetonov v mestu p_k pred proženjem prehoda t_j.

Nek prehod t_k je omogočen, čeprav ima njegovo vhodno mesto p_i negativno označitev (na primer $I(t_k) = \{p_i\}$ in $\mu_R(p_i) = -1$) in njegovo proženje povzroči zmanjšanje števila žetonov v njegovem izhodnem mestu p_k ($O(t_k) = \{p_k\}$) za 1.

Dosegljivostnemu drevesu inverznega obrazca $-KRP$ pravimo *drevo razpoznavanja*.

Tvorimo ga na enak način kot dosegljivostno drevo z upoštevanjem dveh sprememb:

1. upoštevamo modifikacijo pravila proženja prehodov,
2. upoštevamo, da prehodi, ki ustrezajo relacijam iz množice Σ_3, nikoli ne postanejo omogočeni in se nikoli ne prožijo, ne glede na število žetonov v njihovih vhodnih mestih.

Za tiste lastnosti s_i iz množice S, ki imajo obliko $s_i = (r, d)$, tvorimo le poddrevo razpoznavanja s selektivnim proženjem le tistih prehodov, ki ustrezajo relaciji R.

Za tvorbo poddrevesa razpoznavanja ne velja zgoraj navedena omejitev 2.

Postopek razpoznavanja v obrazcu za predstavitev znanja KRP za dano množico lastnosti S in izbrano globino razpoznavanja l, do katere naj se tvori drevo oziroma poddrevo razpoznavanja, opisuje naslednji algoritem:

Postopek

1. korak Danemu obrazcu KRP določi inverzni obrazec $-KRP$.
2. korak Za $\forall s_i \in S$, ki $s_i \in D_2 \cup D_3$ določi mesta p_j v obrazcu $-KRP$:

$$p_j = \alpha^{-1}(s_i).$$

Takih mest je lahko več, naj bo b njihovo število. Vsako od teh mest $p_j,\ j = 1, 2, ..., b$ dobi žeton:

$$\mu_0(p_j) = 1$$

in tako določa b začetnih označitev omrežja:

$$\overrightarrow{\mu}_{01}, \overrightarrow{\mu}_{02}, ..., \overrightarrow{\mu}_{0b}.$$

Za $\forall s_i \in S$, ki $s_i \notin D_2 \cup D_3$ funkcija α^{-1} ni definirana.

3. korak Za $\forall s_i \in S$, kjer je $s_i = (r, d)$, določi mesta p_j in prehode t_j v obrazcu $-KRP$:

$$p_j = \alpha^{-1}(d) \quad \text{in} \quad \tau = \{t_k \in T \mid \beta(t_k) = r\}.$$

Naj bo c število takih mest p_j in vsako od njih ($j = 1, 2, ..., c$) dobi žeton:

$$\mu'_0(p_j) = 1$$

in tako določa c začetnih označitev omrežja:

$$\overrightarrow{\mu}'_{01}, \overrightarrow{\mu}'_{02}, ..., \overrightarrow{\mu}'_{0c}.$$

4. korak Generiraj l nivojev drevesa razpoznavanja za vsako začetno označitev $\overrightarrow{\mu}_{0j}$ ($j = 1, 2, ..., b$). Dobimo b dreves razpoznavanja.

Generiraj l nivojev poddrevesa razpoznavanja za vsako začetno označitev $\overrightarrow{\mu}'_{0k}$ ($k = 1, 2, ..., c$) in s selektivnim proženjem le tistih prehodov iz množice τ. Dobimo c poddreves razpoznavanja.

5. korak Za vsako od b dreves razpoznavanja izračunaj vsoto vektorjev označitev v vseh vozliščih drevesa razen korenskega:

$$\mathbf{z}_j = \sum_{k=1}^{m_j} \overrightarrow{\mu}_{jk}, \quad j = 1, 2, ..., b,$$

kjer je m_j število vozlišč j-tega drevesa razpoznavanja razen korena.

6. korak Za vsako od c poddreves razpoznavanja izračunaj vsoto vektorjev označitev v vseh vozliščih poddrevesa razen korenskega:

$$\mathbf{a}_j = \sum_{k=1}^{m'_j} \overrightarrow{\mu}'_{jk}, \quad j = 1, 2, ..., c,$$

kjer je m'_j število vozlišč j-tega poddrevesa razpoznavanja razen korena.

7. korak Določi vektorje:

$$Z = \sum_{j=1}^{b} \mathbf{z}_j, \quad A = \sum_{j=1}^{c} \mathbf{a}_j, \quad E = Z + A.$$

8. korak V vektorju $E = (e_1, e_2, ..., e_m)$ (m je število mest obrazca KRP) poišči indeks i_{max} komponente z največjo vrednostjo:

$$e_{i_{max}} = \text{Max}\{e_i\}, \quad i = 1, 2, ..., m.$$

Določi mesto $p_j \in P$ za $j = i_{max}$ (to je mesto, ki v celotnem postopku pridobi največ žetonov) in določi pojem, ki temu mestu pripada:

$$d_{rec} = \alpha(p_j).$$

Pojem d_{rec} je iskani pojem X, ki najbolje ustreza opisu S.

Konec postopka.

Zgled 9 Dana je baza znanja z obrazcem za prikaz znanja KRP na sliki 7. Naj je neznani pojem X podan z množico lastnosti S:

$$S = \{\text{štirinožec}, \text{rjava}, \text{udomačen}, (\text{je\_spredaj}, \text{Joe}), (\text{je\_na}, \text{tla})\}.$$

S postopkom razpoznavanja iščemo pojem, ki se najbolje ujema z opisom S. Naj bo globina razpoznavanja $l = 3$.

Sledimo algoritmu na str. 55:

Postopek

1. korak Inverzni obrazec $-KRP$ je ponazorjen na sliki 9.

2. korak

$$\begin{aligned} s_1 &= \text{štirinožec} \in D_2 \cup D_3 \Rightarrow \alpha^{-1}(\text{štirinožec}) = p_8, \\ s_2 &= \text{rjava} \in D_2 \cup D_3 \Rightarrow \alpha^{-1}(\text{rjava}) = p_{10}, \\ s_3 &= \text{udomačen} \notin D_2 \cup D_3 \Rightarrow \alpha^{-1}\text{ni definirana}, \end{aligned}$$

začetni označitvi:

$$\begin{aligned} \overrightarrow{\mu}_{01} &= (0,0,0,0,0,0,0,1,0,0), \\ \overrightarrow{\mu}_{02} &= (0,0,0,0,0,0,0,0,0,1). \end{aligned}$$

3. korak

$s_4 = (\text{je\_spredaj}, \text{Joe})$	$\text{je\_spredaj} \in \Sigma_3, \text{Joe} \in D$
	$\beta^{-1}(\text{je\_spredaj}) = t_3$
	$\alpha^{-1}(\text{Joe}) = p_2$
$s_5 = (\text{je\_na}, \text{tla})$	$\text{je\_na} \notin \Sigma, \text{tla} \notin D \Rightarrow \alpha^{-1}\text{ni definirana},$
	$\beta^{-1}\text{ni definirana}$
začetna označitev:	$\overrightarrow{\mu}'_{01} = (0,1,0,0,0,0,0,0,0,0).$

4. korak Generiramo $l = 4$ nivojev dreves razpoznavanja RT_1 in RT_2 za začetni označitvi $\overrightarrow{\mu}_{01}$ in $\overrightarrow{\mu}_{02}$ ter $l = 4$ nivojev poddrevesa razpoznavanja RT_3 za začetno označitev $\overrightarrow{\mu}'_{01}$. Drevesa so prikazana na sliki 10.

5. korak Za vsako od dreves razpoznavanja izračunamo vsoto vektojev označitve:

$$\begin{aligned} \mathbf{z}_1 &= (1,0,1,0,1,0,1,0,0,0), \\ \mathbf{z}_2 &= (1,0,0,0,0,0,0,0,0,0). \end{aligned}$$

6. korak Za vsako od poddreves razpoznavanja izračunamo vsoto vektojev označitve:

$$\mathbf{a}_1 = (1,0,0,0,0,0,0,0,0,0).$$

7. korak Določimo vektorje:

$$\begin{aligned} Z = \mathbf{z}_1 + \mathbf{z}_2 &= (2,0,1,0,1,0,1,0,0,0), \\ A = \mathbf{a}_1 &= (1,0,0,0,0,0,0,0,0,0), \\ E = Z + A &= (3,0,1,0,1,0,1,0,0,0). \end{aligned}$$

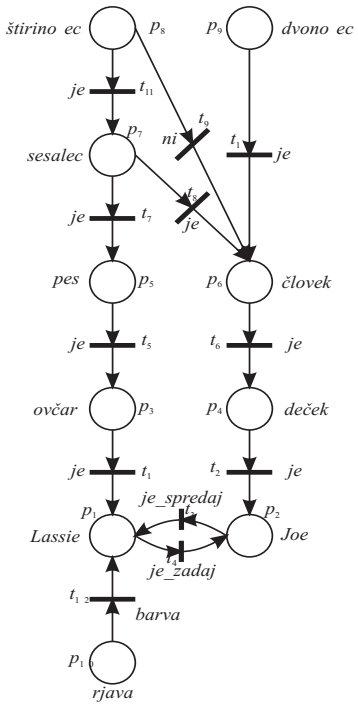
8. korak Ker $e_{i_{max}} = 3$ in $i_{max} = 1$, je p_1 mesto, ki v postopku razpoznavanja pridobi največ žetonov.

Rezultat razpoznavanja: $\alpha(p_1) =$ Lassie.

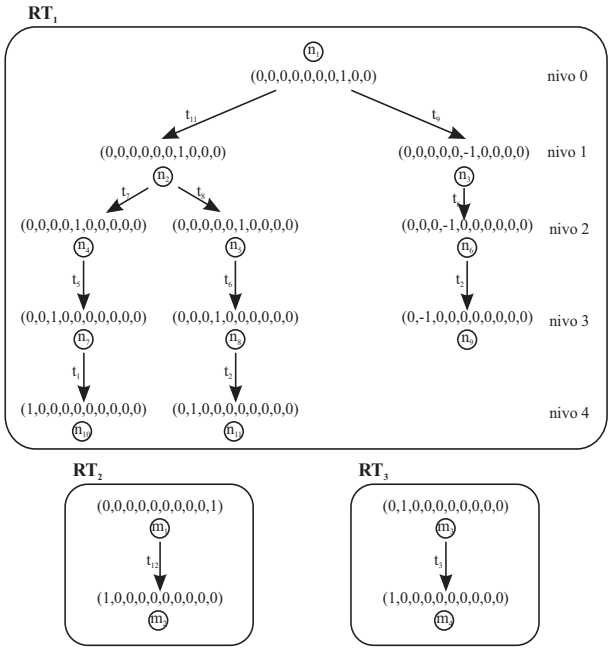
Iskani pojem, ki najbolje ustreza danemu opisu S, je:

$$X = \text{Lassie}.$$

Konec postopka.



Slika 9: Inverzni obrazec $-KRP$.



Slika 10: Drevesi razpoznavanja RT_1 in RT_2 ter poddrevo razpoznavanja RT_3.

■

Viri

N. Pavešić, *Razpoznavanje vzorcev: uvod v analizo in razumevanje vidnih in slušnih signalov*, 3. popravljena in dopolnjena izd., 2 zv., (Dodatek D) Založba FE in FRI, 2012.

Parkelj Mojca: *Z znanjem podprt sistem umetnega vida*, magistrska naloga, 4. poglavje, FE Ljubljana, 1996.

Ribarić S.: it Knowledge Representation Scheme Based on Petri Net Theory, Int. Journal of Pattern Recognition and Artificial Intelligence, Vol. 2, No. 4, 691–700, 1988.

Vprašanja

Kako je definiran splošni obrazec za prikaz znanja s Petrijevimi omrežji (KRP)?

Katere osnovne operacije v obrazcu za prikaz znanja KRP poznamo?

Kako hierarhično predstavimo znanje v omrežju KRP?

Kako se izvede proces plemenitenja in proces posploševanja znanja v omrežju KRP?

Katere postopke sklepanja poznamo v omrežju KRP?

Kakšna vprašanja lahko postavljamo pri generiranju odgovorov z omrežjem KRP?

Kaj iščemo pri iskanju presečišč aktivnosti v omrežju KRP?

Kakšna oblika sklepanja z omrežjem KRP je dedovanje?

Kaj je naloga postopka razpoznavanja v omrežju KRP?

Najnujnejše o matematični logiki

Vsebina

IZJAVNI RAČUN

Izjavni račun obravnava povezave nad **izjavami**. Pri tem imamo posamezne izjave za **celote** in se ne zanimamo za njihovo notranjo logično zgradbo.

Kaj so izjave?

Pravimo, da je trdilni ali nikalni stavek smiseln, če se v okviru predmetov in pojmov, o katerih stavek govori, lahko odločimo, ali je njegova vsebina **resnična** ali **neresnična**.

Vsi smiselni stavki, ki trdijo isto, določajo **izjavo**.

Primeri smiselnih izjav:

1. *Enakostranični trikotniki imajo enaki stranici.*
2. *Ptiči imajo krila.*
3. *Danes je sončno.*
4. *Kvadrat je pravokotnik.*

Primeri nesmiselnih izjav:

1. *Zapri vrata!*
2. *Je zunaj mrzlo?*
3. *Vrednost x je večja od 2.*

Izjave z logičnimi povezavami sestavljamo v nove izjave. Najbolj pogosto uporabljamo naslednje povezave:

negacija $^-$ ali $\neg$,

disjunkcija $+$ ali $\vee$,

konjunkcija $\cdot$ ali $\wedge$,

implikacija $\Rightarrow$ ali $\supset$,

ekvivalencija $\equiv$ ali $\Leftrightarrow$.

Resničnost delnih izjav bolj ali manj vpliva na resničnost sestavljene izjave.

Resničnost sestavljenih izjav lahko razberemo iz pravilnostne tabele.
(1 pomeni resničnost, 0 pa neresničnost izjave).

A	$\neg$ A
1	0
0	1

A	B	$A \vee B$	$A \wedge B$	$A \Rightarrow B$	$A \Leftrightarrow B$
1	1	1	1	1	1
1	0	1	0	0	0
0	1	1	0	1	0
0	0	0	0	1	1

- Če izjavi A in B povežemo z veznikom "ali", dobimo disjunkcijo: $A \vee B$. Iz tabele vidimo, da je disjunkcija resnična takrat, kadar je resnična ali prva ali druga ali pa obe izjavi, torej vsaj ena od obeh izjav.

- Če izjavi A in B povežemo z veznikom "in", dobimo konjunkcijo: $A \wedge B$. Iz tabele vidimo, da je konjunkcija resnična takrat, kadar sta oba člena resnični izjavi.

- Iz izjave A sledi izjava B: $A \Rightarrow B$, če lahko iz resničnosti A sklepamo na resničnost B. Izjava $A \Rightarrow B$ (beri: "če A, potem B") se imenuje implikacija.
  (Če A ni resničen, potem je resničnost ali neresničnost B ne vpliva na pravilnost implikacije)

  Zgled: *Telo miruje* $\Rightarrow$ *Vsota vseh na telo delujočih sil je nič*
  V implikaciji $A \Rightarrow B$ je A predpostavka (premisa, hipoteza, antecedens), B je posledica (zaključek, konsekvens).

- Izjavi A in B sta ekvivalentni (ali: ekvivalenca $A \Leftrightarrow B$ je resnična), če sta A in B vselej hkrati resnični ali hkrati lažni (glej tabelo). ($A \Leftrightarrow B$ beri: "A,če in samo če B").
  Zgled: *Produkt števil* a *in* b *je pozitiven* $\Leftrightarrow$ *Števili* a *in* b *sta enako predznačeni.*

- Izjavi, ki je vedno resnična ne glede na naravo delnih izjav, rečemo **tavtologija**.

  Zgled: *Vrane so črne ali niso črne.*

- Izjavi, ki je vedno neresnična ne glede na naravo delnih izjav, rečemo **kontradikcija**.

  Zgled: *Miha je poročen z Marijo in Marija ni poročena z Mihom.*

Primer: Dokaži, da je ekvivalenca: $\neg A \vee B \Leftrightarrow A \Rightarrow B$.

A	B	$\neg A$	$\neg A \vee B$	$A \Rightarrow B$
1	1	0	1	1
1	0	0	0	0
0	1	1	1	1
0	0	1	1	1

V tabeli je stolpec $\neg A \vee B$ identičen stolpcu za $A \Rightarrow B$.

Logično sklepanje

Sklepanje omogoča nadomestiti dve izjavi (*premisi*) z eno samo izjavo (*sklep*).

Pravila sklepanja:

1. Konjunkcija:

Premisa 1: A

Premisa 2: B

Sklep: $A \wedge B$

2. Modus ponens (pogojna eliminacija):

Premisa 1: A

Premisa 2: $A \Rightarrow B$

Sklep: B

3. Resolucija:

Premisa 1: $A \vee B$

Premisa 2: $A \vee \neg B$

Sklep: A

Opazimo, da v primeru da sta obe premisi pravilni, je tudi sklep pravilen.

Dokazovanje pravilnosti sklepov z resolucijo je priljubljeno v UI, ker številni jeziki UI (npr. Prolog), za dokazovanje dopuščajo le povezavi $\vee, \wedge$ ter $\neg$.

A	B	$\neg B$	$A \vee B$	$A \vee \neg B$	$(A \vee B) \wedge (A \vee \neg B)$
1	1	0	1	1	1
1	0	1	1	1	1
0	1	0	1	0	0
0	0	1	0	1	0

Vprašanja

- Kaj so izjave?
- Podajte primere smiselnih izjav izjavnega računa.
- S katerimi logičnimi povezavami sestavljamo izjave?
- Kateri izjavi pravimo tavtologija in kateri kontradikcija?
- Kaj omogoča logično sklepanje?
- Navedite tri glavna pravila sklepanja.

PREDIKATNI RAČUN

Izjavo x *je večji od 2* lahko obravnavamo s predikatnim računom, ne pa z izjavnim računom, kjer je v izjavnem računu potrebno eksplicitno navesti, za katero število x velja izjava.

V izjavnem računu obravnavamo izjavo kot celoto in se ne menimo za njeno notranjo zgradbo.

V naravnem jeziku je izjava sestavljena iz besed, ki so razporejene po pravilih sintakse in katerih pomen določa sporočilo izjave. Besede, ki označujejo kaj trdimo ali zanikamo v izjavi, imenujemo *predikat* in besede, ki označujejo o čemer trdimo ali zanikamo, imenujemo *objekt*.

V predikatnem računu se izjava sestoji iz dveh delov: predikata in argumenta(ov).

Izjavo:

- *Objekt a ima lastnost P.* zapišemo $P(a)$
- *Objekta a_1, a_2 sta v odnosu Q.* zapišemo $Q(a_1, a_2)$
- *Objekti $a_1, ..., a_n$ so v odnosu R.* zapišemo $R(a_1, ..., a_n)$.

P imenujemo *enomestni predikat*, Q *dvomestni predikat* in R *n-mestni predikat*.

Na primer:

- izjavo *Jana je Slovenka.* zapišemo kot

  SLOVENKA(JANA)

- izjavo *Peter je v sobi.* zapišemo kot

  V(PETER, SOBA)

- izjavo *x je večji od y.* zapišemo kot

  VEČJI(x, y)

Predikata: SLOVENKA(JANA) in V(PETER, SOBA) vsebujeta konstante (JANA, PETER, SOBA), zato predstavljata logični izjavi, ki sta lahko resnični ali neresnični.

V predikatu VEČJI(x, y) je vsebovano veliko izjav, ker sta x in y spremenljivki. Predikat VEČJI(x, y) je resničen, če zamenjamo x s številom ki je večje od števila, s katerim smo zamenjali y. V nasprotnem pa je neresničen.

Predikat je torej preslikava argumentov v *RESNIČNO* ali *NERESNIČNO*

OPOMBA: V predikatnem računu pišemo predikate in konstante z velikimi črkami, spremenljivke pa z malimi črkami.

Logične povezave

Predikatni račun uporablja iste logične povezave, kot izjavni račun. Uporaba logičnih povezav omogoča povezovanje *osnovnih* predikatov.

Na primer, spodnje izjave zapišemo kot:

- *Peter živi v Ljubljani, v Sloveniji.*

  ŽIVI(PETER, LJUBLJANA) $\wedge$ LEŽI (LJUBLJANA, SLOVENIJA)

- *Peter je v sobi ali v kuhinji.*

  V(PETER, SOBA) $\vee$ V(PETER, KUHINJA)

- *Če je lastnik avtomobila Peter, potem je avtomobil rdeč.*

  LASTNIK(PETER, AVTOMOBIL) $\Rightarrow$BARVA(AVTOMOBIL, RDEČ)

- *Če Peter ne bere knjigo, potem piše pismo.*

  $\overline{\text{BERE}}$(PETER, KNJIGA) $\Rightarrow$ PIŠE (PETER, PISMO)

Kvantifikator

Kvantifikator določa meje spremenljivk znotraj katerih je predikat, ki vsebuje spremenljivke, resničen oziroma neresničen.

V predikatnem računu uporabljamo dva kvantifikatorja:

- EKSISTENCIALNI $\exists$ (''obstaja'', ''eksistira'')
- UNIVERZALNI $\forall$ (''vsak'')

Če velja predikat P vsaj za en element x dane množice, potem to zapišemo kot $(\exists x)P$, kar preberemo *Obstaja vsaj en x tako, da je P.*

Na primer, stavek ''V kuhinji je nekaj na mizi.'' zapišemo kot

$(\exists x)$[V(x, KUHINJA) $\wedge$ NA(x, MIZA)]

Trditev, da velja predikat P za vsak element x dane množice, zapišemo kot $(\forall x)P$, kar preberemo ''Vsak x ima lastnost P''.

Na primer, stavek
''Če x ni enako y, potem je x večji ali manjši od y.'' zapišemo kot

$(\forall x)(\forall y)[\overline{\text{ENAKO}}(x,y) \Rightarrow$VEČJI$(x,y) \vee$MANJŠI$(x,y)]$

Če spremenljivke niso vezane s kvantifikatorji, so *proste* in jih lahko zamenjamo s katerimi koli konstantami.

Funkcije

n-mestna funkcija na področju predmetov in pojmov (krajše *individumov*) $\mathcal{P}$ je vsako pravilo, ki priredi vsaki n-terki individumov področja $\mathcal{P}$ natanko en individum (konstanto) tega področja.

y je **funkcija** spremenljivk $x_1, x_2, ..., x_n$ zapišemo kot:

$$y = f(x_1, x_2, ..., x_n).$$

Zgled: Vzemimo funkcijo ene spremenljivke

$$y = u(x),$$

kjer je u UČITELJ. Vzemimo, da spremenljivko x lahko nadomestimo z imenom kateregakoli učenca v razredu, v katerem je učitelj g. Janez. Vzemimo, da sta Jana in Peter učenca v tem razredu. Če zamenjamo x z JANA ali s PETER, bo v obeh primerih y= g. JANEZ.

(Vrednost predikata je RESNIČNO ali NERESNIČNO. Vrednost funkcije je konstanta.)

Pravila sklepanja

V predikatnem računu uporabljamo ista pravila sklepanja kot v izjavnem računu, to je: Konjunkcija, Modus ponens in Resolucija.

Na primer:

Premisa 1: RIBA(POSTRV)

Premisa 2: $(\forall x)$[RIBA(x)$\Rightarrow$PLAVA(x)]

Sklep: PLAVA(POSTRV)

Iz stavkov: *Postrv je riba.* in *Vsaka riba lahko plava.* ustvari pravilo sklepanja Modus ponens stavek (sklep) *Postrv plava.*

Resničnost sklepov dokazujemo s pravilnostnimi tabelami ali s pomočjo algoritma, ki temelji na uporabi pravila resolucije. Tovrstnem dokazovanju pravimo ''Dokazovanje s protislovjem'' ali ''Reduciranje do absurda''.

Oglejmo si kako dokažemo pravilnost sklepa *Postrv plava.* z algoritmom, ki temelji na uporabi pravila resolucije.

a) Predpostavimo, da sklep ni pravilen
$\overline{\text{PLAVA}}$(POSTRV)

b) Vse premise zapišemo v *konjuktivni obliki* (lahko so negirani le predikati, ne pa tudi njihove logične povezave; predikati so povezani le s povezavo ''*ali*'') in jih razbijemo v *klavzule* (klavzula vsebuje predikate in negirane predikate, ki so lahko povezani le s povezavo ''*ali*'').
Vse klavzule in negirani sklep iz koraka **a)** zapiši na ''seznam klavzul''.

V našem primeru je prva premisa: RIBA(POSTRV) že klavzula.

Drugo premiso: $(\forall x)$[RIBA$(x)\Rightarrow$PLAVA(x) lahko
zapišemo kot: $(\forall x)[\overline{\text{RIBA}}(x)\vee\text{PLAVA}(x)]$
(glej primer: $\overline{A} \vee B \equiv A \Rightarrow B$)

Ker je ta izraz resničen za vsak x, lahko zamenjamo x s POSTRV.
Dobimo: $\overline{\text{RIBA}}(POSTRV)\vee$PLAVA$(POSTRV)$

(Procesu zamenjave spremenljivk s konstantami pravimo *unifikacija* (poenotenje).)

V našem primeru vsebuje ''seznam klavzul'':
$\overline{\text{PLAVA}}$(POSTRV),
RIBA(POSTRV) in
$\overline{\text{RIBA}}$(POSTRV) $\vee$ PLAVA(POSTRV).

c) Izberi s seznama klavzul iz koraka **b)** par klavzul tako, da vsebuje ena klavzula predikat, ki je negiran v drugi klavzuli (npr. $A \vee B$ in $\overline{A} \vee B$). Izbrani klavzuli briši s seznama klavzul.

V našem primeru izberemo par:
RIBA(POSTRV)
$\overline{\text{RIBA}}$(POSTRV) $\vee$ PLAVA(POSTRV)

d) Uporabi izpeljano pravilo Modus ponens

Premisa 1: A

Premisa 2: $\neg A \vee B$

Sklep: B

s premisami iz koraka **c)**.
V našem primeru je sklep: PLAVA(POSTRV)

e) Klavzulo - sklep iz koraka **d)** postavi na seznam klavzul iz koraka **b)** in ponavljaj koraka **c)** in **d)** dokler ne dobiš klavzule, ki je v kontradikciji

z negiranim sklepom iz koraka **a)**.
Do kontradikcije (protislovja) pridemo, ko se klazuli v pravilu resolucije izničita:

Premisa 1: A

Premisa 2: $\neg A$

Sklep: prazna klavzula

V našem primeru imamo na seznamu PLAVA(POSTRV), ki je kontradiktorna negiranemu sklepu $\overline{\text{PLAVA}}$(POSTRV), zato:

Premisa 1: PLAVA(POSTRV)

Premisa 2: $\overline{\text{PLAVA}}$(POSTRV)

Sklep: prazna klavzula

Sklep *Postrv lahko plava.* je torej resničen.

Zaključek

Najprej smo obravnavali izjavni račun. Nismo se menili za notranjo zgradbo izjav, temveč le za povezave med njimi. Na vsa temeljna logična vprašanja, kot npr., kdaj je kakšna izjava logično resnična ali neresnična, ali, kako ugotovimo logične posledice dane skupine izjav, lahko dobimo odgovor s *pomočjo pravilnostnih tabel*.

Za razliko od izjavnega računa, kjer so atomarne izjave še neizoblikovane, v predikatnem računu upoštevamo še *notranjo zgradbo* izjav. Vpeljali smo področje individumov, ki mu pravimo *svet pogovora*; vpeljali smo *predikate*, *konstante*, *spremenljivke* in dva *kvantifikatorja*.

Simbolični jezik, ki smo ga tako zgradili, imenujemo *jezik prvega reda*. (V jezikih višjega reda so možni predikati, ki imajo za argumente spet predikate, ali pa se kvantifikatorja ne nanašata samo na spremenljivke, temveč tudi na funkcije in predikate.)

Jezik prvega reda je dovolj ''bogat'', da se v njem lahko izrazimo na različnih področjih.

PROLOG (PROgrammation en LOGique) sloni na ideji, da jezik prvega reda uporabimo kot programski jezik.

Vprašanja

- V čem se predikatni račun razlikuje od izjavnega računa?
- Kako je določen predikat v predikatnem računu?
- Katere logične povezave uporablja predikatni račun?
- Katera dva kvantifikatorja pozna predikatni račun?
- V čem je razlika med funkcijo in predikatom?
- Katera pravila sklepanja uporablja predikatni račun?
- Kako dokazujemo resničnost sklepov?

Najnujnejše iz Prologa

- Računalniški programski jezik.
- Se razlikuje od običajnih postopkovnih jezikov.
- Programer prevaja opis problema iz naravnega jezika v formalno definicijo problema.
- V prologu opisujemo problem (kaj je potrebno napraviti) v jeziku, ki računalniku pove, kako naj reši zadano nalogo.
- Beseda **prolog** je kratica za "**pro***gramming* in **log***ic*, torej programiranje v jeziku logike.

Program v postopkovnem jeziku je (po Wirthu):

Program = Algoritem + Podatkovne strukture

Program v prologu je (po Kowalskem):

Program = Logične izjave + Kontrola izvajanja

Osnova prologa je podmnožica predikatnega računa I. reda, to so Hornovi stavki:

Če veljajo **pogoj1** *in* **pogoj2** *in ...* **pogojn** *potem velja* **sklep**.

V sintaksi prologa so stavki zapisani obratno:

Sklep *velja, če veljajo* **pogoj1** *in* **pogoj2** *in ...* **pogojn**.

Prologovi stavki so v resnici nekoliko posplošeni Hornovi stavki, ker dovoljujejo tudi negirane pogoje.

Primer Hornovovega stavka v prologovi sintaksi:

```
sklep :-
  pogoj1,
  pogoj2,
  ...
  pogojn.
```

- Temu stavku pravimo *pravilo*.
- Stavku brez pogojev pravimo *dejstvo*.
- Z dejstvi in pravili opisujemo relacije med objekti.
- Uporabnik v dialogu s prologovim tolmačem zapisuje dejstva in pravila in sprašuje o veljavnosti določenih trditev.
- Če trditev prologov tolmač izpelje iz danih pravil in dejstev, potem trditev obvelja za resnično.

Primer programa, sestavljenega iz treh dejstev in enega pravila:

```
otrok(mojca,ana).
otrok(miha,ana).
otrok(ana,marija).

roditelj(X,Y) :-
  otrok(Y,X).
```

Prologov tolmač se nam javi z:

```
?-
```

Preverjanje trditve s prologovim tolmačem:

1) Preverjanje dejstva:

```
?- otrok(ana,marija).
true
?- otrok(mojca,marija).
false
```

2) Preverjanje nepopolnega dejstva:

```
?- otrok(X,marija).
X = ana
true
?- otrok(miha,X).
X = ana
true
?- otrok(X,Y).
X = miha
Y = ana;

X = mojca
Y = ana;

X = ana
Y = marija;
false
```

Ali lahko računamo s prologom?

Primer Evklidovega algoritma iskanja največjega skupnega delitelja:

Programski jezik C:

```
int delitelj(int a, int b)
{
  while(a != b)
    if(a > b)
      a -= b;
    else
      b -= a;

  return a;
}
```

Programski jezik prolog:

```
delitelj(A,A,A).

delitelj(A,B,Delitelj) :-
   A > B,
   A1 is A - B,
   delitelj(A1,B,Delitelj).

delitelj(A,B,Delitelj) :-
  delitelj(B,A,Delitelj).
```

Preverjanje nepopolnih in popolnih dejstev:

```
?- delitelj(24,30,X).
X = 6
?- delitelj(24,30,6).
true
?- delitelj(24,30,5).
...
```

Druga različica Evklidovega algoritma v prologu:

```
delitelj(A,A,A).

delitelj(A,B,Delitelj) :-
   A > B,
   A1 is A - B,
   delitelj(A1,B,Delitelj).

delitelj(A,B,Delitelj) :-
   B > A,
   B1 is B - A,
   delitelj(B1,A,Delitelj).
```

2) Preverjanje nepopolnega dejstva:

```
?- delitelj(24,30,5).
false
```

Kakšna je sintaksa prologa?

- Prologovi stavki so zaključeni s *piko*.
- Sestavni del stavka je *literal* (sintaktični konstrukt za preproste izjave).
- Prologovi stavki so lahko *dejstvo*,*pravilo*,*ukaz* ali *vprašanje*.

1) Dejstvo sestavlja en sam pozitivni literal.

```
dezuje.                % Hja, dezuje!
otrok(miha,ana).     /* Miha je Anin otrok! */
enaka(X,X).
miha je modelar.
moc([],0).
sledenje_vkljuceno.
```

2) Pravilo sestavlja glava in telo, ločena z operatorjem.

```
glava :- telo.
neumen :- lep.

izlet :-
  soncno,
  avto.

premagal(X,Y) :-
  predal(Y,X);
  (tocke(X,Tx),
   tocke(Y,Ty),
   Tx > Ty).
```

Literalom v telesu pravila pravimo tudi *cilji*.

3) Ukaz je pravilo brez glave.

```
:- cilj1,cilj2;cilj3,cilj4.

:- consult('moja.pl').

:- write('Dober dan!').

:- start.
```

4) Vprašanje je v prologu podobno ukazu.

```
?- dezuje.

?- dezuje,izlet.

?- otrok(miha,ana).

?- otrok(miha,X).

?- otrok(X,Y).

?- star(miha,X), X > 10.
```

Pri vprašanju je le zamenjan operator `:-` z operatorjem `?-`.

Literal, predikat in operator

- Literal sestavlja predikatni simbol in n argumentov v oklepaju, ločenih z vejico.
- Predikatni simbol določa relacijo med objekti.
- Število argumentov (n $\geq$ 0) določa mestnost predikata.
- Predikatni simbol določa relacijo med objekti.
- Predikat je enolično določen z imenom predikata in mestnostjo.

V pravilu

```
premagal(X,Y) :-
  predal(Y,X);
  (tocke(X,Tx),
   tocke(Y,Ty),
   Tx > Ty)).
```

nastopajo predikati `premagal/2`, `tocke/2` in `>/2`.

Predikat `>/2` je operator, ki ga zaradi berljivosti pišemo med argumetoma.

Ime predikata mora biti *atom*, argumenti pa *termi* (izrazi).

Procedura

- Proceduro sestavljajo vsi stavki, ki določajo isti predikat.

Program

```
mama(ana,miha).
mama(ana,mojca).
mama(marija,ana).

stara_mama(X,Y) :-
  mama(X,Z),
  roditelj(Z,Y).

roditelj(X,Y) :-
  mama(X,Y).

roditelj(X,Y) :-
  oce(X,Y).
```

vsebuje tri procedure za predikate `stara_mama/2`, `roditelj/2` in `mama/2`.

Izrazi

izrazi			
enostavni izrazi			strukture
spremenljivke	konstante		
	atom	število	
`X, Rezultat, _x1, H_A`	`miha, x_13, «-:`	`1, 213, 2.7, 3.14`	`oseba(EMSO, ime(Ime,Priimek))`

Atomi:

- Alfanumerični: `miha  x_13  miha_hocevar`
- Posebni: `:-  <->  ::-`
- V narekovajih: `'Miha'  'A = B + C'`

Števila: `331, 7.5877`

Spremenljivke: `X, _x13, O1, Ime_Priimek`

Strukture: `funktor(arg1,arg2, ...  argn)`

`oseba(EMSO,ime(Ime,Priimek),datum(Dan,Mesec,Leto))`

Operatorji

- Operatorji so funktorji eno ali dvomestnih struktur, zapisani brez uporabe oklepajev.
- Enomestni operator stoji pred argumetom (*prefiksno*) ali za argumetom (*postfiksno*).
- Dvomestni operator stoji med argumetoma (*infiksno*).
- Operatorji povečujejo preglednost programa.

Vnaprej določeni operatorji: `+ - = / *`

```
+(+(*(5,*(X,X)),*(100,X)),/(10,+(X,1)))
5*X*X + 100*X + 10/(X+1)
```

Leva in desna asociativnost.:
Kateri operator veže argumente močneje.

```
a + b + c        (a + b) + c
```

Prioriteta.:
Moč vezave argumenta neodvisno od asociativnosti.

```
a + b * c        a + (b * c)
```

Spremenite v prologove stavke!

Sonja igra klavir.

```
igra(sonja,klavir).
```

Irena igra vse instrumente, ki imajo klaviaturo.

```
igra(irena,Instrument) :-
    ima(Instrument,klaviatura).
```

Ali Igor igra kakšen instrument?

```
igra(Oseba) :-
    instrument(Instrument),igra(Oseba,Instrument).
?-igra(igor).
```

Katere instrumente igra Igor?

```
?-igra(igor,Instrument).
```

Boris igra instrumente, ki jih igrata Mojca in Sonja.

```
igra(boris,Instrument) :-
    instrument(Instrument),
    igra(mojca,Instrument),
    igra(sonja,Instrument).
```

Kdo zna igrati kitaro in harmoniko?

```
?-igra(Oseba,kitaro),igra(Oseba,harmoniko).
```

Spremenite v prologove stavke!

Boris igra vse instrumente, ki jih igrata Mojca ali Sonja.

```
igra(boris,Instrument) :-
    instrument(Instrument),
    (igra(mojca,Instrument);
    igra(sonja,Instrument)).
```

Ali zna kdo igrati kitaro in harmoniko?

```
igra_kitaro_in_harmoniko :-
    igra(Oseba,kitara),
    igra(Oseba,harmonika).
?-igra_kitaro_in_harmoniko.

?-igra(_,kitara),igra(_,harmonika).
```

Znebite se dvoumnosti!

```
otrok(leo,lili).          % Leo je Lilin otrok.
otrok([leo,tea],lili).    % Leo in Tea sta Lilina otroka.
otrok(lili,2).            % Lili ima dva otroka.
```

```
otrok(leo,lili).           % Leo je Lilin otrok.
otroci([leo,tea],lili).    % Leo in Tea sta Lilina otroka.
stevilo_otrok(lili,2).     % Lili ima dva otroka.
```

Prevedite prologove stavke v naravni jezik!

```
ima_rad('Ana','Niko').
```

Ana ima rada Nika.

```
znak(==>,implikacija).
```

'==>' je znak za logično implikacijo.

```
znak(&,konjunkcija).
```

'&' je znak za logično konjunkcijo.

```
znak(V,disjunkcija).
```

Karkoli je znak za logično disjunkcijo.

```
znak(*,Zvezdica).
```

Zvezdica je znak za karkoli.

```
porabi(Ana,1000) :- dohodek(Ana,OD), OD > 1000.
```

Nekdo lahko porabi 1000, če ima dohodek večji kot 1000.

```
porabi(niko,_100) :- dohodek(niko,_200), _200 > _100.
```

Niko lahko porabi manj kot je njegova plača.

```
ima_rad(Ana,Niko).
```

Vsakdo ima rad vsakega.

Prevedite prologove stavke v naravni jezik!

```
?-ima_rad(Ana,Niko).
```

Kdo ima rad koga?

```
?-ima_rad(Ana,_).
```

Kdo ima rad nekoga?

```
?-ima_rad(_,_).
```

Ima kdorkoli rad kogarkoli?

```
?-not(ima_rad(Ana,'Niko')).
```

Ali nima nihče rad Nika?

Izboljšajte berljivost prologovih stavkov!

```
stara_mama(X,Y) :- mama(X,Z),(mama(Z,Y);oce(Z,Y)).
```

```
stara_mama(Mama,Vnuk) :-
    mama(Mama,Otrok),
    roditelj(Otrok,Vnuk).
roditelj(Roditelj,Otrok) :-
    mama(Roditelj,Otrok);
    oce(Roditelj,Otrok).
```

Izboljšajte berljivost prologovih stavkov!

```
med(C,A,B) :- levo(A,C),desno(B,C);
              levo(B,C),desno(A,C).
```

```
med(C,A,B) :-
    levo(A,C),
    desno(B,C).
med(C,B,A) :-
    levo(A,C),
    desno(B,C).
```

```
pot_domov(Pisarna,Dom) :- pes(Pisarna,Postaja1),
    (vlak(Postaja1,Postaja2);bus(Postaja1,Postaja2)),
    pes(Postaja2,Dom).
```

```
pot_domov(Pisarna,Dom) :-
    pes(Pisarna,Postaja1),
    prevoz(Postaja1,Postaja2),
    pes(Postaja2,Dom).
prevoz(Postaja1,Postaja2) :-
    vlak(Postaja1,Postaja2);
    bus(Postaja1,Postaja2)).
```

Seznami

- Seznam je posebna struktura v obliki urejene množice elementov.
- Vrstni red elementov je pomemben.
- Seznam je zgrajen iz glave in repa.
- Glava je prvi seznam elementa.
- Rep je nujno seznam in to seznam brez prvega elementa.

```
[obj1,obj2, ... objn]         % Osnovni zapis

[obj1 | [obj2, ... objn]]     % Z loceno glavo in repom

[]                            % Prazen seznam

[o1, o2, o3 | [o4, ... on]]   % Posplosena uporaba oznake |
```

Primeri seznamov, razbitih na glavo in rep.

```
[tine,tone,metka]     [tine              | [tone,metka]]
[1,2,3,5|X]           [1                 | [2,3,5|X]]
[X|Y]                 [X                 | Y]
[f(mesto,[1,2,3])]    [f(mesto,[1,2,3]) | []]
```

Razbijanje seznama na glavo in rep.

```
[obj1,obj2, ... objn] = [Glava|Rep].
```

Seznami

Posplošena uporaba oznake | pri zapisu seznamov

```
[a,b,c] = [a|[b,c]] = [a|[b|[c]]] = [a|[b|[c|[]]]]
[a,b,c] = [a,b|[c]] = [a,b,c,[]] = [a|[b,c|[]]]
```

Razbijte sezname na glavo in rep!

```
[a,b,c] = [G|R].
  G = a
  R = [b, c]

[[a,b,c],e|[d,f]] = [[G|R1]|R2].
  G = a
  R1 = [b, c]
  R2 = [e, d, f]

[] = [G|R].
  false

[X,[Y]] = [G|R].
  X = G
  R = [[Y]]
```

Nizi

- Nizi so seznami pozitivnih celih števil.
- Pozitivna cela števila ustrezajo ASCII kodnim znakom.

Prolog ne dela razlike med zapisoma

```
"Prolog"
[80,114,111,108,111,103]
```

Procedura `name(Atom,Seznam)` omogoča pretvorbo atoma v seznam ASCII kodnih znakov!

```
?- name('Uvod v prolog',S).
  S = [85,118,111,100,32,118,32,112,114,111,108,111,103]

?- name(A,"Uvod v prolog").
  A = 'Uvod v Prolog'

?- name(A,[112,114,111,108,111,103]).
  A = prolog    % Brez presledka brez enojnih navednic
```

Konstruktor

- Konstruktor `=..` je operator , ki omogoča sestavljanje poljubnih struktur.
- Glava seznama postane funktor strukture z argumenti v repu seznama.

```
?- X =.. [glava,o2,o3,o4].   % Operator =.. mora
  X = glava(o2, o3, o4)      % biti zapisan brez presledka

?- funktor(o1,o2,o3) =.. X.
  X = [funktor,o1,o2,o3]
```

Primeri uporabe konstruktorja:

```
?- med(a,b,c) =.. X.
  X = [med,a,b,c]

?- X =.. [a].
  X = a

?- X =.. [naslov,kajuhova,5],
   Y =.. [oseba,tone,X].

  X = naslov(kujuhova,5)
  Y = oseba(tone,naslov(kujuhova,5))
```

Prilagajanje

- Uporaba pri izpeljevanju cilja iz vprašanja.
- Iskanje stavka, katerega sklepni del "ustreza" cilju iz vprašanja.
- Glava stavka ustreza cilju, če se mu lahko *prilagodi*.
- Prilagajanje je operacija izenačitve dveh izrazov.
  - Dve konstanti se prilagodita, če sta identični.
  - Dve strukturi se prilagodita, če imata isti funktor in če se lahko vsi argumenti med seboj paroma prilagodijo.
  - Neopredeljena spremenljivka se lahko prilagodi čemurkoli.
  - Po prileganju postane spremenljivka opredeljena in njena vrednost je identična prilagojenemu izrazu.
  - Dve neopredeljeni spremenljivki postaneta po prileganju identični.

Operator = je operator, ki izvede prilagoditev dveh izrazov.

```
?- X = 1 + 2.
  X = 1 + 2

?- X + Y = 1 + 2.
  X = 1                % +(X,Y) = +(1,2).
  Y = 2

?- f(X) = f(g(1)).
  X = g(1)

?- X = f(X).
  X =  f(f(f(f(f(f(f(f( ....   % Neskon\c na zanka!
```

Kako se prilagodijo naslednji izrazi?

```
?- datum(D,M,1997) = datum(D1,maj,Y1).
    D = D1      M = maj        Y1 = 1997

?- trikot(t(1,1),A,t(2,3)) = trikot(X,t(4,Y),t(2,Y)).
    Y = 3       X = t(1, 1)    A = t(4, 3)

?- X = [a,b,c].
    X = [a,b,c].

?- [a,b|X] = [a,b,c].
    X = [c]     % Ker [a,b,c] = [a,b|[c]].

?- [X|[Y]] = [a,b,c].
    false       % Ker [a,b,c] = [a|[b,c]] in Y != b,c

?- [X,Y] = [a,[b,c]].
    X = a       Y = [b,c]

?- [X|Y] = [a].
    X = a       Y = []     % Ker [a] = [a|[]]

?- [2|X] = X.
    X = [2,2,2,2,2, ...    % Ker X = [2|[2|[2 ...
```

Aritmetika

- Operator = je operator, ki izvede le prilagoditev (unifikacijo) dveh izrazov (!).
- Operator `is` je procedura, ki omogoča izračun aritmetičnega izraza.
- V aritmetičnem izrazu lahko nastopajo vsa števila in vgrajeni operatorji.
  - to so operatorji `+`,`-`,`*`,`/`,`div` in `mod`.
- V aritmetičnem izrazu lahko nastopajo vgrajene matematične funkcije.
  - od različice prologa: `sin(Stevilo)`,`log(Stevilo)`, ...

```
?-  X = 1 + 2.
  X = 1 + 2

?-  X is 1 + 2.
  X = 3

?-  3 = 1 + 2.
  false

?-  3 is 1 + 2.
  true

?-  X = 1 + 2, Y is 3*X.
  X = 1 + 2
  Y = 9
```

Primerjanje aritmetičnih izrazov

Poleg procedure `is` poznamo še procedure za primerjanje vrednosti aritmetičnih izrazov.

operator	pomen
>	večji kot
<	manjši kot
>=	večji ali enak
=<	manjši ali enak
=:=	enaka vrednost
=/=	različna vrednost

Operator =:= moramo ločiti od operatorja ==, ki služi primerjanju nearitmetičnih izrazov.

```
?-  2*3 = 7-1.
  false           % Ker se *(2,3) in -(7,1) ne prilegata!

?-  2*3 is 7-1.
  false           % Ker se *(2,3) in 6 ne prilegata!

?-  6 is 7-1.
  true            % Ker sta 6 in 6 identicna!

?-  2*3 =:= 7-1.
  true            % Ker sta 6 in 6 enaki vrednosti!

?-  2*3 >= 7-1.
  true            % Ker je 6 vecje ali enako 6!
```

Kakšen bo odgovor na naslednja vprašanja?

```
?- X is 17 * 3 - 5 + log(1.0).
     X = 46

?- 5 is X - 2.
     [WARNING: Unbound variable in arithmetic expression]

?- 15 is 3 * 5.
     true

?- X = 1 + 2.
     X = 1 + 2

?- X == 1 + 2.
     false  % To je splosno primerjanje izrazov

?- X =:= 1 + 2.
     false  % Ker je X neopredeljen

?- X = 1 + 2, 1 + 2 * 3 = X * 3.
     false  % Ker je X * 3 = (1 + 2) * 3

?- X = 5 + 3, Y is 2 * X.
     X = 5 + 3    Y = 16

?- X = a + Y, Y = b + Z, Z = c + d.
     Z = c + d
     Y = b + (c + d)
     X = a + (b + (c + d))
```

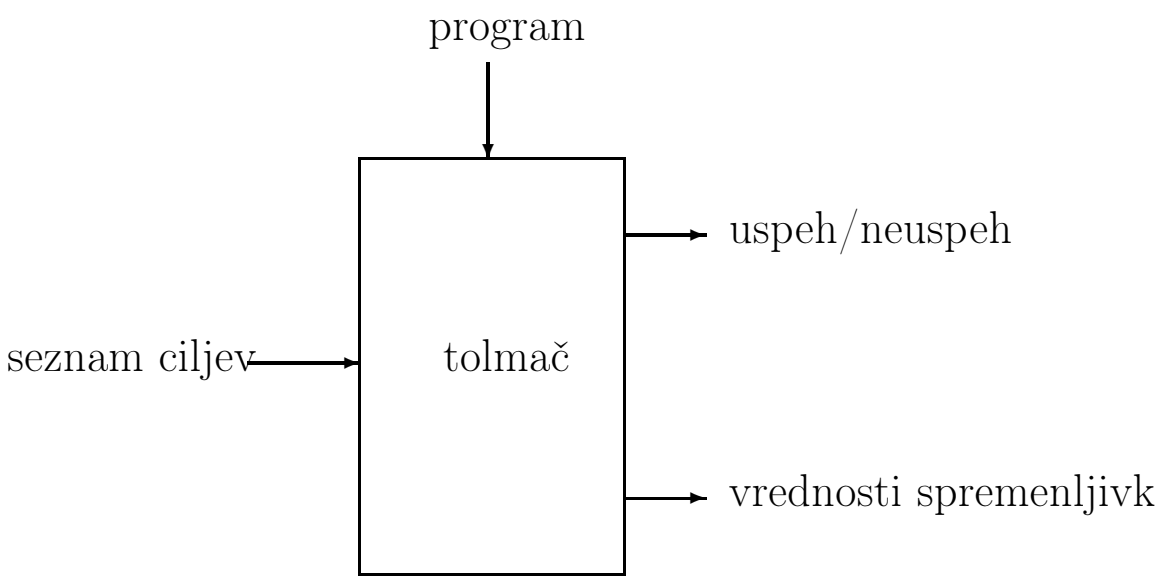
Leksikalni doseg spremenljivk in stavkov

Tolmačenje spremenljivk in procedur v različnih delih programa.

- Vse procedure so globalne.
- Vrstni red zapisa procedur ni pomemben,
  pomemben pa je vrstni red zapisa stavkov v proceduri.
- Vse spremenljivke so lokalne v stavku.
- Prolog ne pozna globalnih spremenljivk.
- Posebej se obravnava anonimna spremenljivka `_`.
- Anonimna spremenljivka je lokalna sama sebi.

```
naslov(Ime,Naslov) :-
  oseba(Ime,_,_,Naslov).

?- otrok(_,miha).    % Ali ima miha otroka?

?- otrok(_,_).       % Ali ima kdo otroka?

?- igra(_,kitara),igra(_,harmonika).

  % Ali igra kdo kitaro in kdo harmoniko?
```

Pravila izvajanja prologovega tolmača



- Uspešna izpolnitev seznama ciljev vsebuje tudi vrednosti spremenljivk iz vprašanja, ki so se uspešno prilagodile.
- Vrednost anonimnih spremenljivk tolmač ne izpisuje.
- Tolmač izpolnjuje seznam ciljev v vprašanju pri danem programu po posebnih pravilih.

Pravila izvajanja prologovega tolmača

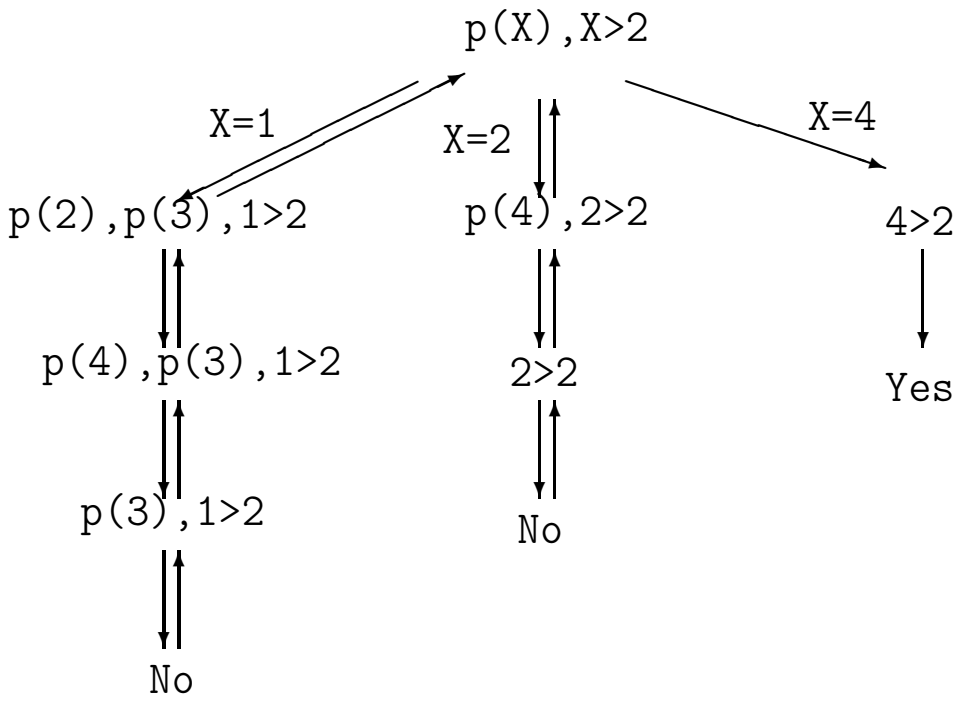
1. Cilje tolmač izpolnjuje od **leve proti desni**.
2. Cilje izpolni tako da:
   (a) poišče prvi stavek (**od zgoraj navzdol**),
   katerega glava se lahko prilagodi cilju;
   (b) iz stavka naredi različico stavka s preimenovanjem
   vseh spremenljivk v stavku, da prilagoditev ne bi
   spremenila programa;
   (c) cilj prilagodi glavi te različice stavka, telo opredeljene
   različice stavka pa postavi na začetek seznama ciljev.
3. Če cilj ne uspe se tolmač vrača do prejšnjega cilja, ki ga je že prilagodil, ter ga poskuša ponovno prilagoditi v skladu s prejšnjo točko, la da preiskuje naprej od stavka, katerega glavi se je cilje prej prilagodil.
4. Če ne uspe prilagoditi prvega cilja, potem izpolnitev seznama ciljev ni uspela. Če seznam ciljev postane prazen, pa je izponitev uspela.

Če prologov tolmač ne izvrši četrte točke,
pride do *zaciklanja*.

```
delaj :- delaj.    % Program!

?- delaj.          % Vprasanje, ki zacikla tolmaca
```

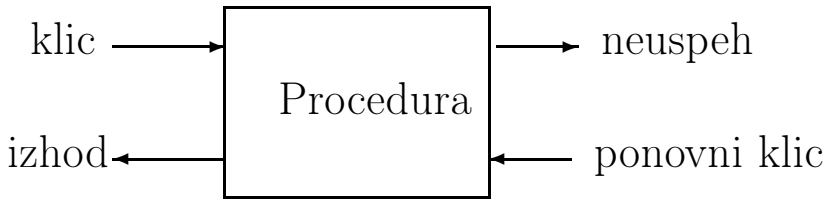
Primer izvajanja prologovega tolmača

```
p(1) :- p(2),p(3).          %%%%%%%%%%%%%%%%%%%%%%%%
p(2) :- p(4).               % Program (procedura)! %
p(4).                       %%%%%%%%%%%%%%%%%%%%%%%%

?- consult('program.pl')    % Nalaganje programa

?- p(X),X>2.                % Vprasanje.
```



Sledenje izvajanja prologovih programov

- Iskanje napak v programu je olajšano zaradi možnosti kontrole izvajanja (`trace, spy(procedura,st_argumentov`).
- Sintaktične napake odkrije tolmač , paziti pa moramo na tipkarske napake.
- Hude so semantične napake, ki so posledica nezadostnega razumevanja reševanega problema.



V prologovem tolmaču so zgornji izrazi dejansko:

`call, exit/done, redo, fail`.

Primer sledenja izvajanja prologovih programov

```
otrok(miha,ana).
otrok(lili,ana).
roditelj(X,Y) :- otrok(Y,X).

?- spy(roditelj/2),trace,roditelj(ana,X).
  *  Call:   roditelj(ana, G1248) ?
       Call:   otrok(G1248, ana) ?
       Exit:   otrok(miha, ana) ?
  *  Exit:   roditelj(ana, miha) ?
 X = miha ;
       Redo:   otrok(G1248, ana) ?
       Exit:   otrok(lili, ana) ?
  *  Exit:   roditelj(ana, lili) ?
 X = lili ;
 No

?- spy(roditelj/2),trace,roditelj(joze,X).
  *  Call:   roditelj(joze, G1272) ?
       Call:   otrok(G1272, joze) ?
       Fail:   otrok(G1272, joze) ?
  *  Redo:   roditelj(joze, G1272) ?
  *  Fail:   roditelj(joze, G1272) ?
No
```

Osnovni principi programiranja v prologu

Rekurzija

- Definicija pojma pri spodnji meji vrednosti rekurzijske spremenljivke (*robni pogoj*) z uporabo le znanih pojmov.
- Definicija pojma pri neki vrednosti rekurzijske spremenljivke z uporabo znanih pojmov in uporabo pojma samega pri manjši vrednosti rekurzijske spremenljivke.

```
fak(0,1).                    %%%%%%%%%%%%%%%%%%%%%%%%%
fak(N,F) :-                  % Racunanje faktoriele! %
  N > 0,                     %%%%%%%%%%%%%%%%%%%%%%%%%
  N1 is N - 1,
  fak(N1,F1),
  F is N*F1.

?- fak(2,F).                 % fak(+N,-F)/2

otrok(lili,ana).
otrok(miha,lili).
otrok(sonja,lili).

naslednik(Nas,Pred) :-       %%%%%%%%%%%%%%%%%%%%%%%%
  otrok(Pred,Nas).           % Relacija naslednik! %
naslednik(Nas,Pred) :-       %%%%%%%%%%%%%%%%%%%%%%%%
  otrok(Vmes,Pred),
  naslednik(Nas,Vmes).

?- naslednik(X,ana).
```

Interaktivna rekurzija

```
delaj :-
  write('Vas ukaz: '),
  read(Ukaz),
  (Ukaz = konec;
  izvrsi(Ukaz), delaj).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

delaj :-
  write('Vas ukaz: '),
  read(Ukaz),
  odloci(Ukaz).

odloci(konec).
odloci(Ukaz) :-
  izvrsi(Ukaz),
  delaj.



izvrsi(Ukaz) :-
  write('Vtipkali ste: '),
  write(Ukaz),
  name(NL,[10]),
  write(NL).
```

Rekurzija in seznami

```
% Definicija brisi(+Element,+Seznam1,-Seznam2).
% Procedura brisi/3 izbrise element iz seznama.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
brisi(_,[],[]).
brisi(E,[E|Rep],Rep).
brisi(E,[G|Rep],[G|Rep1]) :-
  brisi(E,Rep,Rep1).

?- brisi(1,[0,1,2],S).

% Definicija procedure element(?Element,+Seznam).
% Procedura element/2 preveri prisotnost elementa v seznamu.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
element(X,[X|_]).
element(X,[_|Rep]) :-
  element(X,Rep).

?- element(1,[0,1,2]).

% Definicija procedura ni_element(?Element,+Seznam).
% Procedura ni_element/2 preveri odsotnost elementa v seznamu.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
ni_element(_,[]).
ni_element(X,[G|Rep]) :-
  X \== G,
  ni_element(X,Rep).

?- ni_element(4,[0,1,2]).
```

Rekurzija in seznami

```
% Definicija stik(?Seznam1,?Seznam2,?Seznam3).
% Procedura stik/3 zdruzi dva seznama v nov seznam.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
stik(_,Seznam,Seznam).
stik([G|R],Seznam,[G|R1]) :-
    stik(R,Seznam,R1).

% Procedura stik/3 tudi razstavlja sezname

?- stik(S1,S2,[a,b,c]).
   S1 = []
   S2 = [a,b,c];
   S1 = [a]
   S2 = [b,c];
   ...
% In isce podsezname

podseznam(P,S) :- stik(_,S1,S),stik(P,_,S1).

?- podseznam(P,[a,b,c]).
   P = [];
   P = [a];
   P = [a,b];
   P = [a,b,c];
   P = [];
   P = [b];
   P = [b,c];
   ...
```

Transformacije vseh elementov seznama

```
kvadriraj([],[]).
kvadriraj([G|Rep],[KvG|KvRep]) :-
  KvG is G*G,
  kvadriraj(Rep,KvRep).

transformiraj([],_,[]).
transformiraj([G|Rep],Operacija,[Gt|Rept]) :-
  Transformacija =.. [Operacija,G,Gt],
  call(Transformacija),
  transformiraj(Rep,Operacija,Rept).


kv(X,X*X).

kvadrat(X,KvX) :- KvX is X*X.

mod3(X,M) :- M is X mod 3.

?-transformiraj([1,2,3,4,5,6],kv,X).
  X = [1 * 1,2 * 2,3 * 3,4 * 4,5 * 5,6 * 6]

?-transformiraj([1,2,3,4,5,6],kvadrat,X).
  X = [1,4,9,16,25,36]

?-transformiraj([1,2,3,4,5,6],mod3,X).
  X = [1,2,0,1,2,0]
```

Kontrola avtomatskega vračanja

- Kontrolo izvajamo s pomočjo vgrajene procedure *rez* - `!`.
- Prvi klic reza vedno uspe, vsi nadaljnji klici pa ne.
- `!` prepreči večkratno izpolnitev cilja v glavi stavka.

```
% Definicija procedure element(?Element,+Seznam).
% Procedura element/2 preveri prisotnost elementa v seznamu.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

element(X,[X|_]) :- !.
element(X,[_|Rep]) :-
  element(X,Rep).

?- element(f(X),[f(1),f(2),f(3)]).
   X = 1;
   No

% Izvajanje s prekinitvijo avtomatskega vracanja
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

a.  e.  d.  g.

p :- a,b,c.
p :- d.
b :- e,!,f.
b :- g.

?- p.
```

Večsmernost procedur

- Na vhodu konstante, izhod je odgovor 'da' ali 'ne'.
- Na vhodu nekaj konstant, na izhodu prilagojene preostale spremenljivke.
- Na vhodu spremenljivke, na izhodu prilagojene spremenljivke.
- Procedura `is` je strogo enosmerna.
- Rez `!` pokvari večsmernost.
- Večsmernost je včasih nezaželena.

```
element(X,[X|_]).
element(X,[_|Rep]) :-
  element(X,Rep).

?- element(a,[b,c,X,d]).
   X = a


strogi_element(X,[X1|_]) :-
  X == X1.
strogi_element(X,[_|Rep]) :-
  strogi_element(X,Rep).

?- element(a,[b,c,X,d]).
   No.
```

Proceduri `assert` in `retract`

- Procedura `assert(S)` doda stavek `S` k programu.
- Moramo paziti na prioriteto operatorja `:-`.
- Procedura `retract(S)` izloči stavek `S` iz programa.
- Če `S` vsebuje spremenljivke se izloči prvi prilagojeni stavek.

```
?- assert((lepo :- soncno)), assert(soncno).
   Yes
?- lepo.
   Yes

?- retract(otrok(ana,X)).
   X = miha;
   No

%
% Uporaba za globalne spremenljivke
%

spremeni(Ime,X) :-
  retract(spremenljivka(Ime,_)),
  assert(spremenljivka(Ime,X)).
```

Procedure `findall`, `bagof` in `setof`

- Procedure poiščejo urejen seznam vseh vrednosti, ki izpolnjujejo cilj iz vprašanja.

```
otrok(joze,ana). otrok(miha,ana).
otrok(lili,ana). otrok(lili,andrej).

?- findall(X,otrok(X,ana),S).
   S = [joze,miha,lili]

?- setof(X,otrok(X,ana),S).
   S = [joze,lili,miha]

?- findall(X,otrok(X,Y),S).
   S = [joze,miha,lili,lili]

?- bagof(X,otrok(X,Y),S).
   S = [joze,miha,lili]
   Y = ana;

   S = [lili]
   Y = andrej
```

Delo z vhodom, izhodom in datotekami

```
?- consult(Datoteka).
?- reconsult(Datoteka).  % Ni vgrajena v SWI!!!
?- see(Datoteka).   % Odprtje in prenos vhoda na datoteko
?- see(user).       % Prenos vhoda na uporabnika
?- seen.            % Zaprtje in prenos na uporabnika
?- seeing(X).       % Info. o trenutnem vhodu
?- tell(Datoteka).  % Odprtje in prenos izhoda na datoteko
?- tell(user).      % Prenos izhoda na uporabnika
?- told.            % Zaprtje in prenos na uporabnika
?- telling(X).      % Info. o trenutnem izhodu
?- open/4.          % Odprtje datoteke - toka.
?- close(Datoteka). % Zaprtje datoteke - toka
?- get0(C).         % Prilagoditev C znaku na vhodu.
?- get(C).          % Prilagoditev C vidnemu znaku na vhodu.
?- put(C).          % Izpis znaka na izhod.
?- read(I).         % Prilagoditev I izrazu na vhodu.
?- write(I).        % Izpis izraza na izhod.

izpisi_znak(Dat) :-
  see(Dat),
  get0(Znak),
  put(Znak),
  see(user).
```

Delo z vhodom, izhodom in datotekami

```
vpisi_znak(Dat) :-
  get0(Znak),
  tell(Dat),
  put(Znak),
  tell(user).

procesiraj(Dat) :-
  seeing(OldStream),
  see(Dat),
  repeat,            % Repeat procedura !
    read(T),
    process(T),
    T == end_of_file,
  !,  % Sicer ga avtomatsko vracanje zaznka.
  seen,
  see(OldStream).


daj_roditelja :-
  write('Cigavega otroka zelis?'),
  read(Otrok),
  roditelj(Roditelj,Otrok),
  write('Roditelj od ',write(Otrok),
  write(' je ',write(Roditelj),
  write('!'),nl.
```

Viri?

- Prolog Programming for Artificial Intelligence (4th Edition)(Bratko, 2011).
- Naloge iz programiranja v Prologu z rešitvami (Kononenko, 1997).
- `http://www.swi-prolog.org/Links.html`

Vprašanja:

- V čem se Prolog razlikuje od postopkovnih programskih jezikov?
- Podajte primer Hornovovega stavka v Prologovi sintaksi.
- Opišite ključne značilnosti sintakse programskega jezika Prolog.
- Kaj določa proceduro v Prologu?
- Spremenite primer izjave v naravnem jeziku v Prologov stavek in obratno.
- Pojasnite razliko med seznami in nizi v Prologu?
- Pojasnite pomen "prilagajanja" v Prologu.
- Kakšen je leksikalni doseg spremenljivk in stavkov v Prologu?
- Pojasnite pravilo izvajanja/sklepanja Prologovega tolmača.
- Kako se v Prologu določi rekurzijo?
- S čim izvajamo kontrolo vračanja pri izvajanju procedur?
- S katero proceduro programsko dodajamo ali odstranjujemo stavke v Prologu?

Najnujnejše o končnih neizrazitih množicah in neizraziti logiki

Vsebina

Teorija neizrazitih množic (angl. fuzzy set theory) obravnava s strogim matematičnim jezikom nenatančno opisane pojme in pojave.

Navadno množico sestavljajo objekti, ki zadoščajo nekemu predikatu. Če predikat za nek objekt drži, je objekt element množice, sicer pa ni.

Definicija 1 *Če je X končna osnovna/univerzalna množica elementov x, potem je* **navadna množica** *A v X množica urejenih parov:*

$$A = \{(x, \psi_A(x))\}, \quad \forall x \in X,$$

kjer je $\psi_A(x) \in \{0, 1\}$ dvovrednostna funkcija (enomestni predikat):

$$\psi_A(x) = \begin{cases} 1, \text{ če } x \in A \\ 0, \text{ če } x \notin A. \end{cases}$$

Navadno množico pogosto pišemo krajše:

$$A = \{x\},$$

kjer so v množici A le tisti x, za katere je $\psi_A(x) = 1$.

Neizrazito množico v osnovni množici elementov dobimo, če namesto predikata vpeljemo *pripadnostno funkcijo*, ki za vsak element osnovne množice zavzame vrednost med 0 in 1. Ta funkcija izraža *stopnjo pripadnosti* elementa v neizraziti množici. Le-ta je večja, če je vrednost pripadnostne funkcije bližje vrednosti 1.

Definicija 2 *Če je X osnovna množica elementov x, potem je* **neizrazita množica** *A v X množica urejenih parov:*

$$A = \{(x, \mu_A(x))\}, \quad \forall x \in X,$$

kjer je $\mu_A(x) \in [0, 1]$ stopnja pripadnosti elementa x v neizraziti množici A.

Neizrazito množico lahko bolj pregledno zapišemo kot:

$$A = \bigcup_{\forall x \in X} \frac{\mu_A(x)}{x}.$$

Zgled 1 Naj bo osnovna množica:

$$X = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}.$$

Praštevila v množici X so enolično določena, zato je množica praštevil A v množici X navadna množica:

$$A = \{(1,0), (2,1), (3,1), (4,0), (5,1), (6,0), (7,1), (8,0), (9,0), (10,0)\}$$

oziroma krajše:

$$A = \{2, 3, 5, 7\}.$$

Majhna števila v množici X niso enolično določena. Ker lahko definiramo več različnih navadnih množic majhnih števil v osnovni množici, je bolj primerno, če majhna števila v X zapišemo z neizrazito množico na primer kot:

$$A = \{(1,1), (2,1), (3,0.9), (4,0.7), (5,0.4), (6,0.2), (7,0), (8,0), (9,0), (10,0)\}$$

ali bolj pregledno:

$$A = \left\{\frac{1}{1}, \frac{1}{2}, \frac{0.9}{3}, \frac{0.7}{4}, \frac{0.4}{5}, \frac{0.2}{6}, \frac{0}{7}, \frac{0}{8}, \frac{0}{9}, \frac{0}{10}\right\}.$$

■

Definicija 3 Moč *(kardinalnost) končne neizrazite množice A v X je:*

$$\text{card}(A) = \sum_{x \in X} \mu_A(x),$$

relativna moč pa:

$$\text{rcard}(A) = \frac{\text{card}(A)}{\text{card}(X)}.$$

Definicija 4 *Neizrazita množica A v X je* **normirana**, *če velja:*

$$\mu_A(x) = 1, \quad \textit{za vsaj en } x \in X.$$

Definicija 5 Rez α *neizrazite množice A v X je navadna množica vseh tistih $x \in X$, za katere je:*

$$\mu_A(x) \geq \alpha,$$

kjer je α realno število med 0 in 1.

Definicija 6 Strogi rez *α neizrazite množice A v X je navadna množica vseh tistih $x \in X$, za katere je:*

$$\mu_A(x) > \alpha,$$

kjer je α realno število med 0 in 1.

Definicija 7 Jedro *neizrazite množice A v X je navadna množica vseh tistih $x \in X$, za katere je:*

$$\mu_A(x) = 1.$$

Jedro neizrazite množice A je torej njen rez pri vrednosti $\alpha = 1$.

Definicija 8 Nosilec *neizrazite množice A v X je navadna množica vseh tistih $x \in X$, za katere je:*

$$\mu_A(x) > 0.$$

Nosilec neizrazite množice A je torej njen strogi rez pri vrednosti $\alpha = 0$.

Definicija 9 Izrazita vrednost *(angl. crisp value) je neizrazita množica, ki ima nosilec s samo enim elementom.*

Znanih je več postopkov pridobivanja izrazite vrednosti iz neizrazite množice, ki ima več kot en element s stopnjo pripadnosti večjo od nič. Najbolj pogosto uporabimo naslednja postopka:

- **Težiščni postopek**
  Vzemimo, da ima neizrazita množica A v končni osnovni množici X pripadnostno funkcijo $\mu_A(x)$. Izrazita vrednost x' množice A je:

$$x' = \frac{\sum_{i=1}^{\mathrm{card}(X)} x_i \mu_A(x_i)}{\sum_{i=1}^{\mathrm{card}(X)} \mu_A(x_i)}.$$

- **Postopek največje vrednosti**
  Vzemimo, da ima neizrazita množica A v osnovni množici X pripadnostno funkcijo $\mu_A(x)$. Izrazita vrednost x' neizrazite množice A je element množice A z največjo stopnjo pripadnosti množici. Če ima pripadnostna funkcija $\mu_A(x)$ največjo vrednost pri več različnih elementih, vzamemo, da je izrazita vrednost x' množice A aritmetična srednja vrednost tistih x, pri katerih je pripadnostna funkcija $\mu_A(x)$ največja.

Zgled 2 Vzemimo, da je dana neizrazita množica:

$A = \{(1,\,1), (2,\,1), (3,\,0.9), (4,\,0.7), (5,\,0.4), (6,\,0.2), (7,\,0), (8,\,0), (9,\,0), (10,\,0)$

Izrazita vrednost neizrazite množice A je:

- težiščni postopek:

$$x' = \frac{1 \cdot 1 + 2 \cdot 1 + 3 \cdot 0.9 + 4 \cdot 0.7 + 5 \cdot 0.4 + 6 \cdot 0.2}{1 + 1 + 0.9 + 0.7 + 0.4 + 0.2} = 2.76.$$

- postopek največje vrednosti:

$$x' = 1/2\,(1 + 2) = 1.50.$$

■

Definicija 10 *Neizrazita množica A v X je* **izbočena/konveksna**, *če velja:*

$$\mu_A(\lambda x_1 + (1 - \lambda)x_2) \geq \min\{\mu_A(x_1), \mu_A(x_2)\}, \quad x_1, x_2 \in X,\ \lambda \in [0, 1].$$

Pripadnostna funkcija izbočene neizrazite množice je nad definicijskim območjem X povsod izbočena.

Definicija 11 Komplement *neizrazite množice A v X je neizrazita množica $\overline{A}$ s pripadnostno funkcijo:*

$$\mu_{\overline{A}}(x) = 1 - \mu_A(x), \quad \forall x \in X.$$

Velja: $\overline{(\overline{A})} = A$.

Definicija 12 Zmnožek *neizrazite množice A v X s številom r je neizrazita množica s pripadnostno funkcijo:*

$$\mu_{rA}(x) = r \cdot \mu_A(x), \quad \forall x \in X.$$

Definicija 13 m**-ta potenca** *neizrazite množice* A *v* X *je neizrazita množica s pripadnostno funkcijo:*

$$\mu_{A^m}(x) = (\mu_A(x))^m, \quad \forall x \in X,\ m \text{ pozitivno realno število.}$$

Nad neizrazitimi množicami $A \subseteq X$ in $B \subseteq X$ lahko definiramo:

Definicija 14 *Neizraziti množici* A *in* B *sta* **enaki** $(A = B)$*, če je:*

$$\mu_A(x) = \mu_B(x), \quad \forall x \in X.$$

Definicija 15 *Neizrazita množica* A *je* **vsebovana** $(A \subset B)$ *v neizraziti množici* B*, če je:*

$$\mu_A(x) \leq \mu_B(x), \quad \forall x \in X.$$

Definicija 16 Algebrajski zmnožek $(A \cdot B)$ *neizrazitih množic* A *in* B *je neizrazita množica s pripadnostno funkcijo:*

$$\mu_{A \cdot B}(x) = \mu_A(x) \cdot \mu_B(x), \quad \forall x \in X.$$

Definicija 17 Algebrajsko povprečje $(povp(A, B))$ *neizrazitih množic* A *in* B *je neizrazita množica s pripadnostno funkcijo:*

$$\mu_{povp(A,B)}(x) = \frac{1}{2}[\mu_A(x) + \mu_B(x)], \quad \forall x \in X.$$

Definicija 18 Presek $(A \cap B)$ *neizrazitih množic* A *in* B *je neizrazita množica s pripadnostno funkcijo:*

$$\mu_{A \cap B}(x) = \min\{\mu_A(x), \mu_B(x)\}, \quad \forall x \in X.$$

Definicija 19 Unija $(A \cup B)$ *neizrazitih množic* A *in* B *je neizrazita množica s pripadnostno funkcijo:*

$$\mu_{A \cup B}(x) = \max\{\mu_A(x), \mu_B(x)\}, \quad \forall x \in X.$$

Definicija 20 Operacija α $(A \,\alpha\, B)$ *neizrazitih množic* A *in* B *je neizrazita množica s pripadnostno funkcijo:*

$$\mu_{A \alpha B}(x) = \begin{cases} 1, & \text{če } \mu_A(x) \leq \mu_B(x) \\ \mu_B(x), & \text{če } \mu_A(x) > \mu_B(x), \end{cases}$$

za $\forall x \in X$*.*

Zgled 3 Naj bo osnovna množica:

$$X = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}.$$

Vzemimo, da sta v množici X dve neizraziti množici: neizrazita množica majhnih števil:

$A = \{(1,1), (2,1), (3,0.9), (4,0.7), (5,0.4), (6,0.2), (7,0), (8,0), (9,0), (10,0)\}$

in neizrazita množica velikih števil:

$B = \{(1,0), (2,0), (3,0), (4,0), (5,0.3), (6,0.5), (7,0.9), (8,0.9), (9,1), (10,1)\}.$

- Algebrajski zmnožek neizrazitih množic A in B v X je neizrazita množica:

$$C = \{(1, 1\cdot 0), (2, 1\cdot 0), (3, 0.9\cdot 0), (4, 0.7\cdot 0), (5, 0.4\cdot 0.3), (6, 0.2\cdot 0.5), (7, 0\cdot 0.9), (8, 0\cdot 0.9), (9, 0\cdot 1), (10, 0\cdot 1)\}$$

oziroma:

$$C = \{(1,0), (2,0), (3,0), (4,0), (5,0.12), (6,0.1), (7,0), (8,0), (9,0), (10,0)\}.$$

- Algebrajsko povprečje neizrazitih množic A in B v X je neizrazita množica:

$$D = \Big\{(1, \frac{1}{2}[1+0]), (2, \frac{1}{2}[1+0]), (3, \frac{1}{2}[0.9+0]), (4, \frac{1}{2}[0.7+0]), (5, \frac{1}{2}[0.4+0.3]), (6, \frac{1}{2}[0.2+0.5]), (7, \frac{1}{2}[0+0.9]), (8, \frac{1}{2}[0+0.9]), (9, \frac{1}{2}[0+1]), (10, \frac{1}{2}[0+1])\}$$

oziroma:

$$D = \{(1,0.5), (2,0.5), (3,0.45), (4,0.35), (5,0.35), (6,0.35), (7,0.45), (8,0.45), (9,0.5), (10,0.5)\}.$$

- Presek neizrazitih množic A in B v X je neizrazita množica:

$$E = A \cap B = \{(1, \min\{1,0\}), (2, \min\{1,0\}), (3, \min\{0.9,0\}), (4, \min\{0.7,0\}), (5, \min\{0.4,0.3\}), (6, \min\{0.2,0.5\}), (7, \min\{0,0.9\}), (8, \min\{0,0.9\}), (9, \min\{0,1\}), (10, \min\{0,1\})\}$$

oziroma:

$$E = \{(1,0), (2,0), (3,0), (4,0), (5,0.3), (6,0.2), (7,0), (8,0), (9,0), (10,0)\}.$$

- Unija neizrazitih množic A in B v X je neizrazita množica:

$$F = A \cup B = \{(1, \max\{1,0\}), (2, \max\{1,0\}), (3, \max\{0.9,0\}), (4, \max\{0.7,0\}), (5, \max\{0.4,0.3\}), (6, \max\{0.2,0.5\}), (7, \max\{0,0.9\}), (8, \max\{0,0.9\}), (9, \max\{0,1\}), (10, \max\{0,1\})\}$$

oziroma:

$$F = \{(1,1), (2,1), (3,0.9), (4,0.7), (5,0.4), (6,0.5), (7,0.9), (8,0.9), (9,1), (10,1)\}.$$

- Operacija α nad neizrazitimi množicami A in B v X je neizrazita množica:

$$G = A \,\alpha\, B = \{(1,0), (2,0), (3,0), (4,0), (5,0.3), (6,1), (7,1), (8,1), (9,1), (10,1)\}.$$

■

Zgled 4 Vzemimo, da sta dani neizraziti množici $A \subseteq X$ in $B \subseteq Y$:

$A = \{(0,0), (1,0.3), (2,0.6), (3,1), (4,1), (5,1), (6,0.6), (7,0.3), (8,0), (9,0)\}$

in

$B = \{(0,0), (1,0), (2,0.3), (3,0.6), (4,1), (5,1), (6,1), (7,0.6), (8,0.3), (9,0)\}$

v končnih osnovnih množicah $X = Y = \{0,1,2,3,4,5,6,7,8,9\}$.

$C = A \,\alpha\, B$ je neizrazita množica s pripadnostno funkcijo $\mu_C(x,y)$:

$$\mu_C(x,y) = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0.3 & 1 & 1 & 1 & 1 & 1 & 0.3 & 0 \\ 0 & 0 & 0.3 & 0.6 & 1 & 1 & 1 & 0.6 & 0.3 & 0 \\ 0 & 0 & 0.3 & 0.6 & 1 & 1 & 1 & 0.6 & 0.3 & 0 \\ 0 & 0 & 0.3 & 0.6 & 1 & 1 & 1 & 0.6 & 0.3 & 0 \\ 0 & 0 & 0.3 & 1 & 1 & 1 & 1 & 0.6 & 0.3 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

■

Zgled 5 Vzemimo, da sta dani neizraziti množici $A \subseteq X$ in $C \subseteq X \times Y$, kjer sta $X = Y = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ končni osnovni množici. Pripadnostni funkciji $\mu_A(x)$ in $\mu_C(x, y)$ sta:

$$\mu_A = \{0,\ 0.3,\ 0.6,\ 1,\ 1,\ 1,\ 0.6,\ 0.3,\ 0,\ 0\}$$

in

$$\mu_C(x,y) = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0.3 & 1 & 1 & 1 & 1 & 1 & 0.3 & 0 \\ 0 & 0 & 0.3 & 0.6 & 1 & 1 & 1 & 0.6 & 0.3 & 0 \\ 0 & 0 & 0.3 & 0.6 & 1 & 1 & 1 & 0.6 & 0.3 & 0 \\ 0 & 0 & 0.3 & 0.6 & 1 & 1 & 1 & 0.6 & 0.3 & 0 \\ 0 & 0 & 0.3 & 1 & 1 & 1 & 1 & 0.6 & 0.3 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

$B = A \,\alpha\, C$ je neizrazita množica s pripadnostno funkcijo:

$$\mu_B(y) = \cap_x \left(\mu_A(x) \,\alpha\, \mu_C(x, y)\right) = \{0,\ 0,\ 0.3,\ 0.6,\ 1,\ 1,\ 1,\ 0.6,\ 0.3,\ 0\}.$$

Neizrazita množica B je torej:

$$B = \{(0,0), (1,0), (2,0.3), (3,0.6), (4,1), (5,1), (6,1), (7,0.6), (8,0.3), (9,0)\}.$$

Presek in unija neizrazitih množic A in B v osnovni množici X imata naslednje lastnosti:

- asociativnost

$$(A \cup B) \cup C \equiv A \cup (B \cup C)$$
$$(A \cap B) \cap C \equiv A \cap (B \cap C)$$

- distributivnost

$$(A \cup B) \cap C \equiv (A \cap B) \cup (B \cap C)$$
$$(A \cap B) \cup C \equiv (A \cup B) \cap (B \cap C)$$

- absorpcija

$$A \cap (A \cup B) \equiv A$$
$$A \cup (A \cap B) \equiv A$$

- idempotenca

$$A \cup A = A$$
$$A \cap A = A$$

- komutativnost

$$A \cup B = B \cup A$$
$$A \cap B = B \cap A$$

- De Morganova zakona

$$\overline{A \cup B} = \overline{A} \cap \overline{B}$$
$$\overline{A \cap B} = \overline{A} \cup \overline{B}$$

Velja:

- unija neizrazite množice z njenim komplementom ni nujno osnovna množica:

$$A \cup \overline{A} \neq X$$

in

- presek neizrazite množice z njenim komplementom ni nujno prazna množica:

$$A \cap \overline{A} \neq \emptyset.$$

Definicija 21 Razdalja Minkovskega *med neizrazitima množicama A in B v X je:*

$$D_M(A,B) = \left[\sum_{i=1}^{\text{card}(X)} |\mu_A(x_i) - \mu_B(x_i)|^s \right]^{\frac{1}{s}},$$

kjer je $s \geq 1$.

Najbolj pogosto uporabljamo naslednje razdalje Minkovskega:

- $s = 1$; Hammingova razdalja:

$$D_H(A,B) = \sum_{i=1}^{\text{card}(X)} |\mu_A(x_i) - \mu_B(x_i)|,$$

- $s = 2$; Evklidova razdalja:

$$D_E(A,B) = \sqrt{\sum_{i=1}^{\text{card}(X)} [\mu_A(x_i) - \mu_B(x_i)]^2},$$

- $s \to \infty$; Čebiševljeva razdalja:

$$D_{\check{C}}(A,B) = \max_{\forall x \in X} \{|\mu_A(x) - \mu_B(x)|\}.$$

Definicija 22 Jezikovni izraz *(angl. linguistic term)* T_i *je trditev, ki je zapisana z besedo ali z več smiselno povezanih besed nekega naravnega jezika. Trditev je:*

- pravilna *za vsak* x *iz (enorazsežnega) razmaka* $[x^i_{min}, x^i_{maks}]$ *v urejeni končni osnovni množici* X *oziroma*
- ni pravilna *za vsak* x *zunaj razmaka* $[x^i_{min}, x^i_{maks}]$ *v* X.

V urejeni končni osnovni množici X je število jezikovnih izrazov poljubno naravno število med 1 in card(X). Število izrazov v X določa njeno *zrnatost* (granulacijo). Za le-to pravimo, da je "groba", če je število T_i v X blizu 1, oziroma da je "fina", če je število T_i v X blizu card(X).

Zgled 6 Vzemimo, da nas zanima telesna temperatura človeka. Merimo jo s toplomerom in vzemimo, da jo odčitavamo z natančnostjo ene desetinke stopinje Celzija. V tem primeru je osnovna množica elementov množica realnih števil:

$$X = \{35.0, 35.1, 35.2, \ldots, 41.9, 42.0\}.$$

Vzemimo, da na osnovi strokovnega ali splošnega znanja o področju uporabe (telesna temperatura človeka) razdelimo osnovno množico elementov X v podmnožice (razmake) s trditvami:

$$\{\textit{znižana}, \textit{normalna}, \textit{povišana}, \textit{zmerna\_vročina},$$
$$\textit{visoka\_vročina}, \textit{zelo\_visoka\_vročina}\}.$$

Izraz *znižana* je trditev, ki drži za vsako temperaturo iz razmaka [35.0, 36.0], izraz *normalna* drži za vsako temperaturo iz razmaka [36.1, 37.0], izraz *povišana* za vsako temperaturo iz razmaka [37.1, 38.0], izraz *zmerna_vročina* za vsako temperaturo iz razmaka [38.1, 39.0], izraz *visoka_vročina* za vsako temperaturo iz razmaka [39.1, 40.5] in izraz *zelo_visoka_vročina* za vsako temperaturo iz razmaka [40.6, 42.0].

Zdravnik bo tako odčitano telesno temperaturo 39.0° C zapisal (označil) kot *zmerna_vročina*.

▪

Navadne množice v X, ki smo jih definirali z jezikovnimi izrazi T_i, lahko zapišemo tudi kot neizrazite množice.

Definicija 23 Neizraziti jezikovni izraz F_i *v osnovni množici* X *je neizrazita množica:*

$$F_i = \{(x, \mu_{T_i}(x))\}, \quad \forall x \in X,$$

kjer je T_i i*-ti izraz v* X*.*

Definicija 24 Jezikovna ali neizrazita spremenljivka *je spremenljivka, ki črpa vrednosti iz zaloge neizrazitih jezikovnih izrazov* $\{F_i : i = 1, 2, ..., N_T\}$ *v osnovni množici* X*.*

Jezikovno spremenljivko poznamo, če so dani:

H	ime spremenljivke,
X	urejena končna osnovna množica elementov,
$T(H)$	množica jezikovnih izrazov $\{T_i : i = 1, 2, ..., N_T\}$ in
$F(H)$	množica neizrazitih jezikovnih izrazov $\Big\{\{(x, \mu_{T_i}) : i = 1, 2, ..., N_T\}\Big\}$.

Zgled 7 Vzemimo:

H je telesna toplota (človeka)

$X = \{35.0, 35.1, 35.2, \ldots, 41.9, 42.0\}$

$T(\textit{telesna\_toplota}) = \{\textit{znižana}, \textit{normalna}, \textit{povišana}, \textit{zmerna\_vročina}, \textit{visoka\_vročina}, \textit{zelo\_visoka\_vročina}\}$

$F(\textit{telesna\_toplota})$ je navadna množica šestih neizrazitih množic:

$$\begin{aligned}
\textit{znižana} = \{&(35.0, 1), (35.1, 1), ..., (35.7, 1),\\
&(35.8, 0.9), (35.9, 0.9), (36.0, 0.7),\\
&(36.1, 0.2), (36.2, 0.1), (36.3, 0),\\
&(37.0, 0), ..., (42.0, 0)\}\\
\textit{normalna} = \{&(35.0, 0), ..., (35.8, 0), (35.9, 0.1),\\
&(36.0, 0.3), (36.1, 0.5), (36.2, 0.8),\\
&(36.3, 0.9), (36.4, 1), ..., (36.8, 1),\\
&(36.9, 0.9), (37.0, 0.7), (37.1, 0.4),\\
&(37.2, 0.3), (37.3, 0.1), (37.4, 0),\\
&..., (42.0, 0)\}
\end{aligned}$$

$$
\begin{aligned}
\textit{povišana} = \{&(35.0,0),...,(36.8,0),(36.9,0.1),\\
&(37.0,0.3),(37.1,0.5),(37.2,0.7),\\
&(37.3,0.9),(37.4,1),...,(37.8,1),\\
&(37.9,0.9),(38.0,0.7),(38.1,0.5),\\
&(38.2,0.3),(38.3,0.1),(38.4,0),\\
&...,(42.0,0)\}\\
\textit{zmerna\_vročina} = \{&(35.0,0),...,(37.8,0),(37.9,0.1),\\
&(38.0,0.3),(38.1,0.5),(38.2,0.7),\\
&(38.3,0.9),(38.4,1),...,(38.8,1),\\
&(38.9,0.9),(39.0,0.7),(39.1,0.5),\\
&(39.2,0.3),(39.3,0.1),(39.4,0),\\
&...,(42.0,0)\}\\
\textit{visoka\_vročina} = \{&(35.0,0),...,(38.8,0),(38.9,0.1),\\
&(39.0,0.3),(39.1,0.5),(39.2,0.7),\\
&(39.3,0.9),(39.4,1),...,(40.3,1),\\
&(40.4,0.9),(40.5,0.7),(40.6,0.5),\\
&(40.7,0.3),(40.8,0.1),(40.9,0),...,(42.0,0)\}
\end{aligned}
$$

$$
\begin{aligned}
\textit{zelo\_visoka\_vročina} = \{&(35.0,0),...,(40.2,0),(40.3,0.1),\\
&(40.4,0.3),(40.5,0.5),(40.6,0.7),\\
&(40.7,0.9),(40.8,1),...,(42.0,1)\}
\end{aligned}
$$

H črpa vrednosti iz $F(H)$. Jezikovna spremenljivka *telesna toplota* lahko zavzame neizrazite vrednosti *znižana, normalna, povišana, zmerna_vročina, visoka_vročina* in/ali *zelo_visoka_vročina*.

Iz $F(H)$ lahko razberemo, da je možnost 0.7, da bo zdravnik zapisal odčitano telesno temperaturo 39.0° C kot *zmerna_vročina* oziroma, in da je možnost 0.3, da jo bo zapisal (označil) kot *visoka_vročina*. ∎

Množico jezikovnih izrazov jezikovne spremenljivke $T(H)$ lahko "obogatimo" s prislovi, kot so na primer: bolj, manj, zelo, malo, precej ipd. Če je jezikovni izraz T_i dan z neizrazito množico – neizrazitim jezikovnim izrazom $F_i = \{(x, \mu_{T_i})\}$, lahko dobimo neizrazite jezikovne izraze jezikovnih izrazov s prislovom z operatorjema *zgoščevanja* in *raztezanja*.

Definicija 25 Operator zgoščevanja CON *naredi neizrazito množico* F_i *bolj izrazito. Velja:*

$$\mu_{prislov\_T_i} = \mu_{CON(F_i)} = (\mu_{T_i})^2.$$

Definicija 26 Operator raztezanja DIL *naredi neizrazito množico* F_i *manj izrazito. Velja:*

$$\mu_{prislov\_T_i} = \mu_{DIL(F_i)} = \sqrt{\mu_{T_i}}.$$

Z operatorjema zgoščevanja in raztezanja in operacijami nad neizrazitimi množicami lahko dobimo pripadnostne funkcije *sestavljenih* neizrazitih izrazov, na primer *ne_zelo_star*, *ne_zelo_star*,*ne_zelo_star_in_ne_zelo_mlad* ipd.

Zgled 8 Vzemimo neizraziti jezikovni izraz:

$$star = \{(60,\, 0.2), (65,\, 0.4), (70,\, 0.6), (75,\, 0.8), (80,\, 1)\}.$$

Neizraziti jezikovni izraz $star$ s prislovom $zelo$ je neizrazita množica:

$$zelo\_star = CON(star) =$$

$$= \{(60,\, 0.04), (65,\, 0.16), (70,\, 0.36), (75,\, 0.64), (80,\, 1)\},$$

neizraziti sestavljeni jezikovni izraz ne_zelo_star pa je neizrazita množica:

$$ne\_zelo\_star = 1 - CON(star) =$$

$$= \{(60, 0.96), (65, 0.84), (70, 0.64), (75, 0.36), (80, 0)\}.$$

▪

Jezikovne spremenljivke omogočajo prevedbo opisov objektov, podanih z besedami ali s stavki naravnega jezika, v numerične opise.

Definicija 27 Jezikovni vzorec *je navadna množica n ali manj vrednosti, ki so jih v danem trenutku ali situaciji zavzele jezikovne spremenljivke H_1, H_2, ..., H_n.*

Definicija 28 Neizraziti odnos (relacija) *R med elementi neizrazite množice v osnovni množici X in neizrazite množice v osnovni množici Y je neizrazita množica v kartezičnem zmnožku univerzalnih množic X in Y:*

$$R = \{ ((x, y), \mu_R(x, y)) \} \quad \forall (x, y) \in X \times Y$$

oziroma:

$$R = \bigcup_{\forall (x,y) \in X \times Y} \frac{\mu_R(x, y)}{(x, y)},$$

kjer sta:

(x, y) *urejeni par kartezičnega (premega) zmnožka $X \times Y$ (element osnovne množice $X \times Y$ v kateri je definirana R) in*

$\mu_R(x, y)$ *stopnja pripadnosti elementa (x, y) v neizraziti množici R.*

Pripadnostna funkcija neizrazite množice R (odnosa elementov dveh neizrazitih množic, vzemimo A v X in B v Y) je dvorazsežna funkcija. Če poznamo pripadnostno funkcijo $\mu_R(x, y)$, lahko določimo pripadnostni funkciji neizrazitih množic v X in Y. Pripadnostna funkcija neizrazite množice A v X je:

$$\mu_A(x) = \max_{y \in Y} \{\mu_R(x, y)\}, \quad x \in X,$$

pripadnostna funkcija neizrazite množice B v Y pa:

$$\mu_B(y) = \max_{x \in X} \{\mu_R(x, y)\}, \quad y \in Y.$$

Če pa poznamo pripadnostni funkciji $\mu_A(x)$ in $\mu_B(y)$, lahko določimo pripadnostno funkcijo odnosa $\mu_R(x, y)$ kot:

$$\mu_R(x, y) = \min_{(x,y) \in X \times Y} \{\mu_A(x), \mu_B(y)\}.$$

Zgled 9 Dani sta osnovni množici:

$$X = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\} \text{ in}$$
$$Y = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}.$$

Vzemimo, da je pripadnostna funkcija $\mu_R(x, y)$ neizrazitega odnosa R med $x \in X$ in $y \in Y$ (dvorazsežna) funkcija:

$$\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0.3 & 0.3 & 0.3 & 0.3 & 0.3 & 0.3 & 0.3 & 0 \\
0 & 0 & 0.3 & 0.6 & 0.6 & 0.6 & 0.6 & 0.6 & 0.3 & 0 \\
0 & 0 & 0.3 & 0.6 & 1 & 1 & 1 & 0.6 & 0.3 & 0 \\
0 & 0 & 0.3 & 0.6 & 1 & 1 & 1 & 0.6 & 0.3 & 0 \\
0 & 0 & 0.3 & 0.6 & 1 & 1 & 1 & 0.6 & 0.3 & 0 \\
0 & 0 & 0.3 & 0.6 & 0.6 & 0.6 & 0.6 & 0.6 & 0.3 & 0 \\
0 & 0 & 0.3 & 0.3 & 0.3 & 0.3 & 0.3 & 0.3 & 0.3 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}.$$

Pripadnostna funkcija neizrazite množice A v X je:

$$\mu_A(x) = \max_y\{\mu_R(x, y)\} = \{0, 0.3, 0.6, 1, 1, 1, 0.6, 0.3, 0, 0\},$$

pripadnostna funkcija neizrazite množice B v Y pa:

$$\mu_B(y) = \max_x\{\mu_R(x, y)\} = \{0, 0, 0.3, 0.6, 1, 1, 1, 0.6, 0.3, 0\}.$$

Neizraziti množici A in B sta torej:

$A = \{(0,0), (1,0.3), (2,0.6), (3,1), (4,1), (5,1), (6,0.6), (7,0.3), (8,0), (9,0)\}$

in

$B = \{(0,0), (1,0), (2,0.3), (3,0.6), (4,1), (5,1), (6,1), (7,0.6), (8,0.3), (9,0)\}.$

Pripadnostno funkcijo odnosa $\mu_R(x,y)$ izračunamo iz pripadnostnih funkcij $\mu_A(x)$ in $\mu_B(y)$ kot:

$$\mu_R(x,y) = \min_{X\times Y}\{\mu_A(x),\, \mu_B(y)\} \quad x = 0,1,...,9,\ \ y = 0,1,...,9.$$

Izračunajmo le diagonalne elemente pripadnostne funkcije odnosa $\mu_R(x,y)$ iz pravkar izračunanih pripadnostnih funkcij $\mu_A(x)$ in $\mu_B(y)$. Le-ti so:

$\min\{0,0\}$, $\min\{0.3,0\}$, $\min\{0.6,0.3\}$, $\min\{1,0.6\}$, $\min\{1,1\}$, $\min\{1,1\}$, $\min\{0.6,1\}$, $\min\{0.3,0.6\}$, $\min\{0,0.3\}$, $\min\{0,0\}$,

to je:

0, 0, 0.3, 0.6, 1, 1, 0.6, 0.3, 0, 0.

■

Temeljne lastnosti neizrazitih relacij R so:

- refleksivnost (povratnost), če je:

$$\mu_R(x,x) = 1, \quad \forall (x,x) \in X \times Y,$$

- simetričnost (vzajemnost), če je:

$$\mu_R(x,y) = \mu_R(y,x), \quad \forall x \in X,\ \forall y \in Y \quad \text{in}$$

- maks-min tranzitivnost (prehodnost), če je:

$$\mu_R(x,y) \geq \max_{z\in Z}\{\min_{x\in X,\, y\in Y}\{\mu_R(x,z),\, \mu_R(z,y)\}\}.$$

Podobnostna neizrazita relacija je vedno povratna, vzajemna in prehodna. Neizraziti relaciji, ki je le povratna in vzajemna, pravimo relacija bližine ali tolerančna relacija.

Definicija 29 Neizrazita pogojna trditev (implikacija) *ČE X je A, POTEM Y je B, je neizrazita množica (relacija):*

$$R = \{ ((x, y), \mu_R(x, y)) \},$$

kjer so:

X *in* Y *jezikovni spremenljivki,*

A *in* B *neizraziti množici iz zalog vrednosti spremenljivk* X *in* Y*,*

$\mu_R(x, y)$ *funkcija pripadnosti neizrazite implikacije:*

$$\mu_R(x, y) = \Phi[\mu_A(x), \mu_B(y)], \quad x \in X, \, y \in Y.$$

Znanih je več funkcij Φ. Na primer:

- Mamdanijeva

$$\Phi[\mu_A(x), \mu_B(y)] = \min\{\mu_A(x), \mu_B(y)\},$$

- Larsenova

$$\Phi[\mu_A(x), \mu_B(y)] = \mu_A(x) \cdot \mu_B(y),$$

- Lukasiewiczova

$$\Phi[\mu_A(x), \mu_B(y)] = \min\{1, (1 - \mu_A(x) + \mu_B(y))\},$$

- Kleenova in Diensova

$$\Phi[\mu_A(x), \mu_B(y)] = \max\{(1 - \mu_A(x)), \mu_B(y)\},$$

- omejeni zmnožek

$$\Phi[\mu_A(x),\, \mu_B(y)] = \max\{0,\, (\mu_A(x) + \mu_B(y) - 1)\},$$

- Zadehova

$$\Phi[\mu_A(x),\, \mu_B(y)] = \max\left\{\min\{\mu_A(x),\, \mu_B(y)\},\, (1 - \mu_A(x)\right\},$$

- standardna

$$\Phi[\mu_A(x),\, \mu_B(y)] = \begin{cases} 1, \text{ če } \mu_A(x) \leq \mu_B(y) \\ 0, \text{ če } \mu_B(y) > \mu_A(x), \end{cases}$$

- Godelianova

$$\Phi[\mu_A(x),\, \mu_B(y)] = \begin{cases} 1, & \text{če } \mu_A(x) \leq \mu_B(y) \\ \mu_B(y), & \text{če } \mu_B(y) > \mu_A(x). \end{cases}$$

Izbira funkcije Φ je odvisna od primera uporabe.

Zgled 10 Kot trditvi (predikata) v neizraziti pogojni trditvi *ČE X je A, POTEM Y je B*, sta dani neizraziti množici:

$A = \{(0,0), (1,0.3), (2,0.6), (3,1), (4,1), (5,1), (6,0.6), (7,0.3), (8,0), (9,0)\}$

in

$B = \{(0,0), (1,0), (2,0.3), (3,0.6), (4,1), (5,1), (6,1), (7,0.6), (8,0.3), (9,0)\}.$

Pripadnostna funkcija $\mu_R(x,y)$ neizrazite pogojne trditve *ČE X je A, POTEM Y je B*, po Mamdaniju je dvorazsežna funkcija:

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.3 & 0.3 & 0.3 & 0.3 & 0.3 & 0.3 & 0.3 & 0 \\ 0 & 0 & 0.3 & 0.6 & 0.6 & 0.6 & 0.6 & 0.6 & 0.3 & 0 \\ 0 & 0 & 0.3 & 0.6 & 1 & 1 & 1 & 0.6 & 0.3 & 0 \\ 0 & 0 & 0.3 & 0.6 & 1 & 1 & 1 & 0.6 & 0.3 & 0 \\ 0 & 0 & 0.3 & 0.6 & 1 & 1 & 1 & 0.6 & 0.3 & 0 \\ 0 & 0 & 0.3 & 0.6 & 0.6 & 0.6 & 0.6 & 0.6 & 0.3 & 0 \\ 0 & 0 & 0.3 & 0.3 & 0.3 & 0.3 & 0.3 & 0.3 & 0.3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Neizrazita pogojna trditev *ČE X je A, POTEM Y je B*, je neizrazita množica:

$$\begin{aligned} R = \{&((0,0),\, 0), ((0,1),\, 0), ((0,2),\, 0),\\ &((0,3),\, 0), ((0,4),\, 0), ((0,5),\, 0),\\ &((0,6),\, 0), ((0,7),\, 0), ((0,8),\, 0),\\ &((0,9),\, 0), ((1,0),\, 0), ((1,1),\, 0),\\ &((1,2),\, 0.3), ((1,3),\, 0.3), ((1,4),\, 0.3),\\ &((1,5),\, 0.3), ((1,6),\, 0.3), ((1,7),\, 0.3),\\ &((1,8),\, 0.3), ((1,9),\, 0), \ldots, ((9,9),\, 0)\}. \end{aligned}$$

■

Neizrazita pogojna trditev pogosto nastopa v (nekanonični) obliki *ČE* X_1 *je A IN* X_2 *je B, POTEM Y je C*.

Zgled 11 Kot trditvi (predikata) v premisnem delu neizrazite pogojne trditve *ČE* X_1 *je A IN* X_2 *je B, POTEM Y je C*, sta dani neizraziti množici:

$$A = \{(1, 0.1), (2, 0.2), (3, 0.4), (4, 0.7)\}$$

in

$$B = \{(7, 0.6), (8, 0.8), (9, 0.6)\}.$$

Ker sta premisi povezani z veznikom IN, je jezikovni vzorec v premisnem delu (X_1 *je* A, X_2 *je* B) presek neizrazitih množic A in B:

$$\begin{aligned} (X_1 \textit{ je } A,\, X_2 \textit{ je } B) = \{&((1,7),\, 0.1), ((1,8),\, 0.1), ((1,9),\, 0.1),\\ &((2,7),\, 0.2), ((2,8),\, 0.2), ((2,9),\, 0.2),\\ &((3,7),\, 0.4), ((3,8),\, 0.4), ((3,9),\, 0.4)\\ &((4,7),\, 0.6), ((4,8),\, 0.7), ((4,9),\, 0.6)\}. \end{aligned}$$

Če še vzamemo, da je trditev v sklepnem delu neizrazite pogojne trditve neizrazita množica:

$$C = \{(0.0,\, 0.1), (0.2,\, 0.4), (0.4,\, 0.6), (0.6,\, 1)\},$$

je pripadnostna funkcija $\mu_R((x_1, x_2), y)$ neizrazite pogojne trditve *ČE* X_1 *je A IN* X_2 *je B, POTEM Y je C*, po Mamdaniju naslednja dvorazsežna funkcija:

$$\begin{bmatrix} 0.1 & 0.1 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.1 & 0.1 \\ 0.1 & 0.2 & 0.2 & 0.2 \\ 0.1 & 0.2 & 0.2 & 0.2 \\ 0.1 & 0.2 & 0.2 & 0.2 \\ 0.1 & 0.4 & 0.4 & 0.4 \\ 0.1 & 0.4 & 0.4 & 0.4 \\ 0.1 & 0.4 & 0.4 & 0.4 \\ 0.1 & 0.4 & 0.6 & 0.6 \\ 0.1 & 0.4 & 0.6 & 0.7 \\ 0.1 & 0.4 & 0.6 & 0.6 \end{bmatrix}.$$

■

Če poznamo neizraziti relaciji med elementi neizrazite množice A in neizrazite množice B ter med elementi neizrazite množice B in neizrazite množice C, lahko s *sestavljanjem relacij* določimo neizrazito relacijo med elementi neizrazitih množic A in C.

Definicija 30 *Če sta dani neizrazita relacija* R_1 *med elementi neizrazite množice v osnovni množici* X *in neizrazite množice v osnovni množici* Y

$$R_1 = \{\, (\,(x, y),\, \mu_{R_1}(x, y)\,)\,\} \quad \forall (x, y) \in X \times Y$$

in neizrazita relacija R_2 *med elementi neizrazite množice v osnovni množici* Y *in neizrazite množice v osnovni množici* Z

$$R_2 = \{\, (\,(y, z),\, \mu_{R_2}(y, z)\,)\,\} \quad \forall (y, z) \in Y \times Z,$$

je **maks–min sestav (kompozitum)** *relacij* R_1 *in* R_2 *neizrazita relacija* R_3 *med elementi neizrazite množice v osnovni množici* X *in neizrazite množice v osnovni množici* Z*:*

$$R_3 = \{\, (\,(x, z),\, \mu_{R_3}(x, z)\,)\,\} \quad \forall (x, z) \in X \times Z,$$

kjer je pripadnostna funkcija neizrazite relacije R_3 enaka:

$$\mu_{R_3}(x,z) = \max_y \left\{ \min_{x,z} \{\mu_{R_1}(x,y),\, \mu_{R_2}(y,z)\} \right\} \quad \forall x \in X,\, \forall y \in Y,\, \forall z \in Z.$$

(Oznaka za maks-min sestav relacij R_1 in R_2 je navadno $R_1 \circ R_2$ ali $R_3 = R_1 \circ R_2$.)

Zgled 12 Vzemimo, da je pripadnostna funkcija $\mu_{R_1}(x,y)$ neizrazite relacije R_1 med $x \in X$ in $y \in Y$ funkcija:

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix},$$

pripadnostna funkcija $\mu_{R_2}(y,z)$ neizrazite relacije R_2 med $y \in Y$ in $z \in Z$ pa funkcija:

$$\begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}.$$

Pripadnostna funkcija $\mu_{R_3}(x,z)$ neizrazite relacije R_3 med $x \in X$ in $z \in Z$ je *maks–min* sestav pripadnostnih funkcij neizrazitih relacij R_1 in R_2. To je:

$$\begin{bmatrix} \max\{\min\{a_{11}, b_{11}\}, \min\{a_{12}, b_{21}\}\} & \max\{\min\{a_{11}, b_{12}\}, \min\{a_{12}, b_{22}\}\} \\ \max\{\min\{a_{21}, b_{11}\}, \min\{a_{22}, b_{21}\}\} & \max\{\min\{a_{21}, b_{12}\}, \min\{a_{22}, b_{22}\}\} \end{bmatrix}.$$

Če nadomestimo množenje dveh elementov matrik z iskanjem manjšega izmed dveh elementov, seštevanje elementov pa z iskanjem največjega elementa, je izračun $\mu_{R_3}(x,z)$ podoben izračunu zmnožka matrik $\mu_{R_1}(x,y)$ in $\mu_{R_2}(y,z)$. ∎

Če je znanje o področju uporabe podano z neizrazitimi pogojnimi trditvami (implikacijami), je posplošeno pravilo sklepanja *modus ponens* najprimernejši postopek logičnega sklepanja.

Definicija 31 Posplošeni modus ponens *je pravilo neizrazitega logičnega sklepanja:*

$$\begin{array}{ll} 1.\,premisa & : \check{C}E\ X\ je\ A,\ POTEM\ Y\ je\ B \\ 2.\,premisa & : X\ je\ A' \\ \hline Sklep & : Y\ je\ B', \end{array}$$

kjer je $B' = \Psi[A, A', B]$ *neizrazita množica v* Y *s pripadnostno funkcijo:*

$$\mu_{B'}(y) = \max_x \{\min\{\mu_{A'}(x), \mu_R(x,y)\}\}.$$

Pri tem je $\mu_R(x,y)$ *pripadnostna funkcija neizrazite pogojne trditve ČE* X *je* A*, POTEM* Y *je* B*.*

Zgled 13 Vzemimo, da sta dani pripadnostna funkcija $\mu_{A'}(x)$ neizrazite množice A' in pripadnostna funkcija $\mu_R(x,y)$ neizrazite pogojne trditve *ČE* X *je A, POTEM Y je B*:

$$\mu_{A'}(x) = \{0.05, 0.1, 0.2, 0.4\}$$

in

$$\mu_R(x,y) = \begin{bmatrix} 0.1 & 0.1 & 0.1 \\ 0.2 & 0.2 & 0.2 \\ 0.4 & 0.4 & 0.4 \\ 0.6 & 0.7 & 0.6 \end{bmatrix}.$$

Pripadnostno funkcijo $\mu_{B'}(y)$ neizrazite množice B' določimo kot *maks–min* sestav pripadnostnih funkcij neizrazite množice A' in neizrazite pogojne trditve *ČE* X *je A, POTEM Y je B*:

$$\begin{aligned} \mu_{B'}(y) &= \mu_{A'}(x) \circ \mu_R(x,y) = [\,0.05, 0.1, 0.2, 0.4\,] \circ \begin{bmatrix} 0.1 & 0.1 & 0.1 \\ 0.2 & 0.2 & 0.2 \\ 0.4 & 0.4 & 0.4 \\ 0.6 & 0.7 & 0.6 \end{bmatrix} \\ &= [\,0.4, 0.4, 0.4\,]. \end{aligned}$$

Viri

N. Pavešić, *Razpoznavanje vzorcev: uvod v analizo in razumevanje vidnih in slušnih signalov*, 3. popravljena in dopolnjena izd., 2 zv., (Dodatek B) Založba FE in FRI, 2012.

Vprašanja

Pojasnite razliko med končno navadno (izrazito) in neizrazito množico.
Podajte zgled končne neizrazite množice.
Podajte nosilec in jedro neizrazite množice
$A = \{(1,1), (2,1), (3,0.9), (4,0.7), (5,0.4), (6,0.2), (7,0), (8,0), (9,0), (10,0)\}$
Je zgornja množica normirana?
Podajte še njeno izrazito vrednost po težiščnem postopku.
Podajte še komplement zgornje neizrazite množice.
Katere operacije na dveh neizrazitih množicah poznamo?
Podajte lastnosti preseka in unije dveh neizrazitih množic.
Kako merimo podobnost med dvema neizrazitima množicama?
Kako je definiran neizrazit jezikovni izraz?
S katerima operatorjema lahko obogatimo množico jezikovnih izrazov?
Kako je definiran jezikovni vzorec? Podajte primer takšnega vzorca.
Kako je definirana neizrazita relacija med elementi dveh neizrazitih množic?
Kako je definirana neizrazita pogojna trditev (implikacija)?
Kako je definirano pravilo neizrazitega logičnega sklepanja?

UMETNA NEVRONSKA OMREŽJA

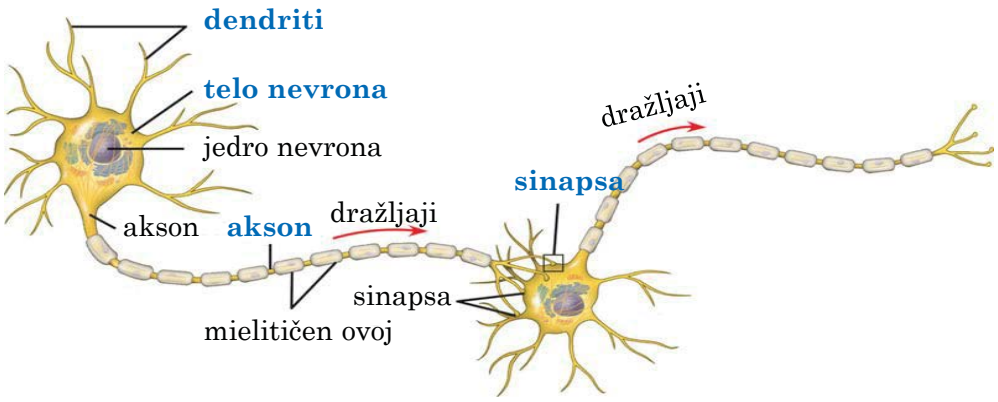
TEME PREDAVANJA

- Model nevronskega sistema
- Vrste nevronskih omrežij
- Predkrmiljeno več-plastno omrežje
- Uporaba umetnih nevronskih omrežij
- Zgled hierarhičnega nevronskega modela

UMETNA NEVRONSKA OMREŽJA

- So model naravnih bioloških nevronskih sistemov
- Temeljijo na paralelnem obdelovanju podatkov
- Imajo sposobnost učenja in prilagajanja
- Z zelo enostavnimi sestavinami tvorijo zelo vsestransko obnašanje
- So zmogljiv sistem za reševanje problemov

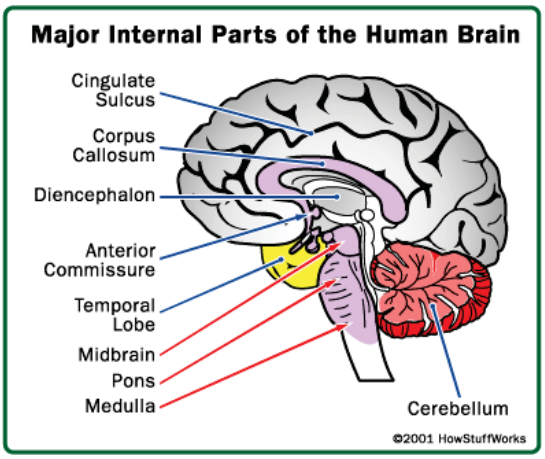
MODEL NEVRONSKEGA SISTEMA

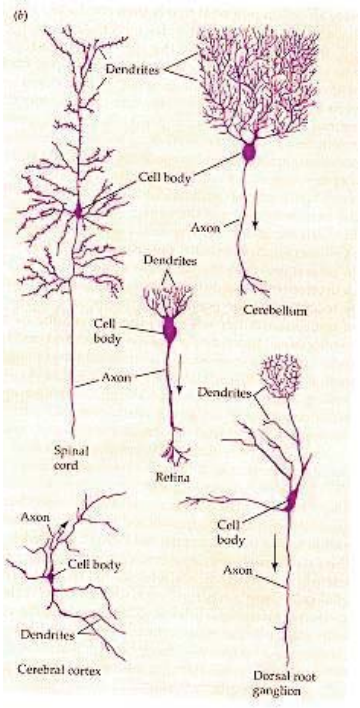


- Nevronski sistem tvori množica podobnih elementov - nevronov, ki so vseskozi povezani.
- Nevron sestavljajo vhodni dendriti, telo nevrona z jedrom in izhodni aksoni.
- Dražljaji iz aksona ene celice do dendritov druge celice potujejo preko sinaps.

Človekov nevronski sistem

- Različen razvoj nevronskih celic v različnih delih možganov in hrbtenjače.





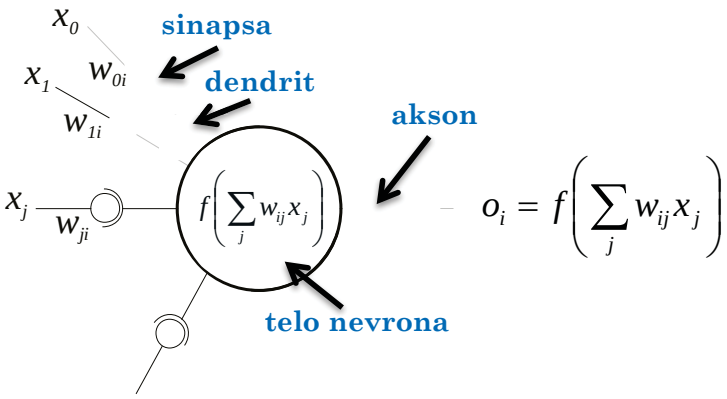
Primerjava z računalnikom

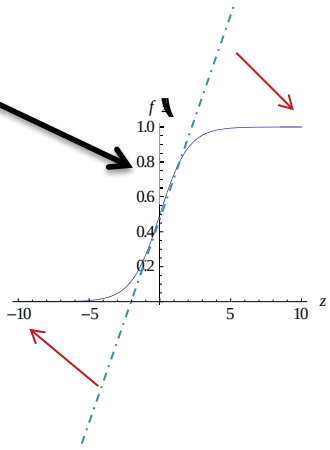
- Ključna razlika med računalnikom in možgani je način obdelave podatkov

	procesnih enot	velikost enot	poraba energije	hitrost obdelave	način obdelave	odporn. na napake	učenje	inteligenca in zavest
	10^{11} celic 10^{14} sinaps	10^{-6} m	30 W	100 Hz	paralelno porazdeljeno	da	da	ponavadi
	10^{8} transistorjev	10^{-6} m	30W	10^{9} Hz	zaporedno centralizirano	ne	malo	ne še

McCulloch-Pittsov model nevrona

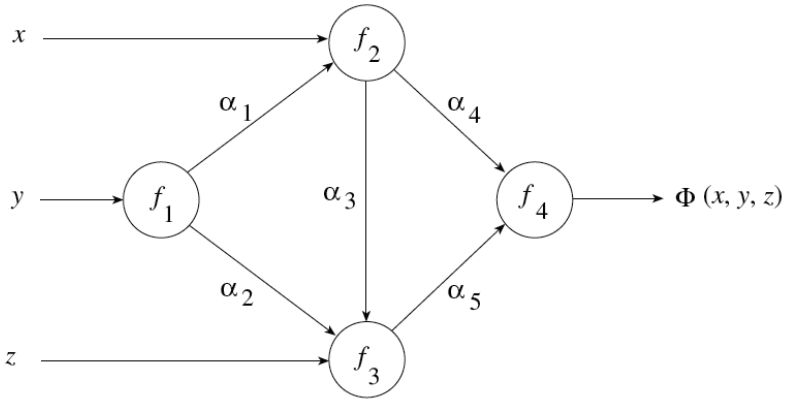
- Jakost izhodnih dražljajev je "stisnjena" linearna funkcija vhodnih dražljajev.





- Groba poenostavitev, ki omogoča razvoj matematičnega modela umetnega nevronskega omrežja.

Umetno nevronsko omrežje

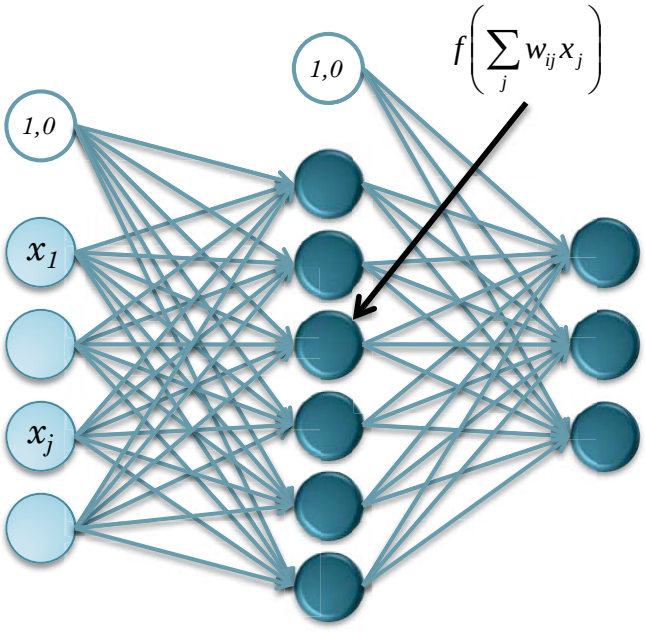


- Funkcijski model nevronskega omrežja.

VRSTE UMETNIH NEVRONSKIH OMREŽIJ

- Več-slojna predkrmiljena nevronska omrežja.
- Samoorganizirajoča nevronska omrežja
- Povratno-zančno nevronsko omrežje.
- Hopfieldovo, Hammingovo Bolzmanovo nevronsko omrežje
- Asociativna nevronska omrežja.
- Nevronska omrežja, ki temeljijo na adaptivni resonančni teoriji.
- Hierarhično Bayesovo omrežje

PREDKRMILJENO VEČ-SLOJNO UMETNO NEVRONSKO OMREŽJE



$$y_k(\mathbf{x}) = f\left(\sum_i w_{ki} f\left(\sum_j w_{ij} x_j\right)\right)$$

preslikava vektorskih naključnih spremenljivk

$$g : X \rightarrow Y$$

- Izhodi omrežja so nelinearne funkcije linearnih kombinacij nelinearnih funkcij linearnih kombinacij …

ZGOŠČEN ZAPIS FUNKCIJE NEVRONA

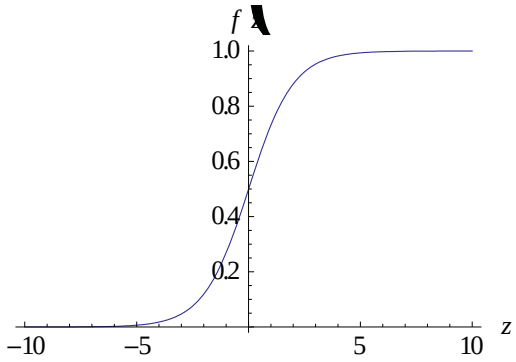
- Zapis funkcije i-tega nevrona navadno zapišemo v vektorski obliki

$$\mathbf{x} = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_j \\ \vdots \end{bmatrix} \qquad \mathbf{w}_i = \begin{bmatrix} w_{i1} \\ w_{i2} \\ \vdots \\ w_{ij} \\ \vdots \end{bmatrix} \qquad z_i = \mathbf{w}_i^T \mathbf{x} = \sum_j w_{ij} x_j$$

$$f(z_i) = f(\mathbf{w}_i^T \mathbf{x}) = f\left(\sum_j w_{ij} x_j\right)$$

NELINEARNA FUNKCIJA NEVRONA

- Nelinearna elementarna funkcija, ki je monotono naraščajoča in odvedljiva, kot je na primer sigmoidalna funkcija.



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

$$f'(z) = f(z)(1-f(z))$$

$$f(z_i) = f(\mathbf{w}_i^T \mathbf{x}) = \frac{1}{1+e^{\mathbf{w}_i^T \mathbf{x}}}$$

IZHOD OMREŽJA KOT IZRAŽAVA PO PARAMETRIČNIH TEMELJNIH FUNKCIJAH

- Izvajamo izražavo funkcije tako, da določamo tako utežne koeficiente kot tudi parametre parametričnih temeljnih funkcij

$$y_k(\mathbf{x}) = f\left(\cdots + w_{ki} f(\mathbf{w}_i, \mathbf{x}) + \cdots + w_{k2} f(\mathbf{w}_2, \mathbf{x}) + w_{k1} f(\mathbf{w}_1, \mathbf{x}) + w_{k0}\right)$$

- Modelirana funkcija ni znana, na razpolago so le njeni odtipki/vzorci na poljubnih mestih (ni enakomernega vzorčenja).

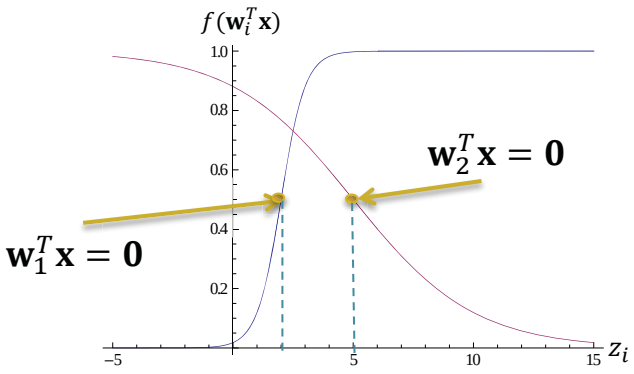
- Pri tipanju funkcije imamo navadno opravka s šumom, zato je izražava v bistvu regresija (modeliranje iz podatkov).

INICIALIZACIJE VREDNOSTI UTEŽI NEVRONOV

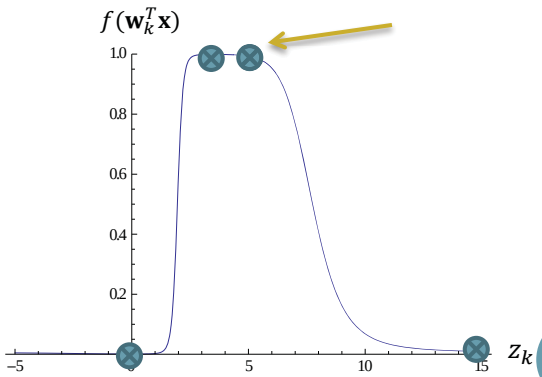
- Ločilna meja nevrona je določena pri

$$\mathbf{w}^T\mathbf{x} = \mathbf{0} \implies f(\mathbf{w}^T\mathbf{x}) = 0{,}5$$

- Začetni položaj ločilne meje vsakega nevrona lahko določimo naključno ali glede na rezultat predhodne analize vzorcev.



$\mathbf{w}_1^T = [2\,, -4]$ $\mathbf{w}_2^T = [-0{,}4\,, -2]$



$\mathbf{w}_k^T = [20\,, 20\,, 25]$

UČENJE Z VZVRATNIM RAZŠIRJANJEM

- Računamo gradient, ki zmanjšuje izhodno napako.
- Napako vzvratno razširjamo po nevronskem omrežju.

$$\varepsilon(\mathbf{x_n}) = \frac{1}{2}\sum_k \varepsilon_k{}^2(\mathbf{x_n}) \qquad \varepsilon_k(\mathbf{x_n}) = y_k(\mathbf{x_n}) - \hat{y}_k(\mathbf{x_n})$$

$$\frac{\partial \varepsilon_k(\mathbf{x_n})}{\partial w_{k\ i}} = \frac{\partial \varepsilon_k(\mathbf{x_n})}{\partial f(z_k)} \frac{\partial f(z_k)}{\partial w_{k\ i}}$$

$$\delta_k(\mathbf{x_n}) = (y_k(\mathbf{x_n}) - \hat{y}_k(\mathbf{x_n})\ f'(z_k)) \qquad w_k(\mathbf{x_n})_i = w_k(\mathbf{x_n})_i - \eta_{\ k}(\mathbf{x_n})\hat{y}_i(\mathbf{x_n})$$

$$\delta_i(\mathbf{x_n}) = f'(z_i)\sum_k \delta_k(\mathbf{x_n}) \qquad w_i(\mathbf{x_n})_j = w_i(\mathbf{x_n})_j - \eta_{\ i}(\mathbf{x_n})\hat{y}_j(\mathbf{x_n})$$

… …

UPORABA UMETNIH NEVRONSKIH OMREŽIJ

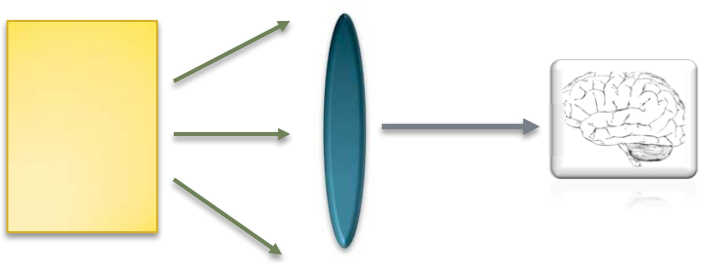
- Razvrščanje
  - Razpoznavanje vzorcev
  - Določanje značilk
  - Poravnavo slik
  - …
- Zmanjševanje šuma
  - Razpoznavanje znanih vzorcev in odstranjevanje šuma
- Napovedovanje
  - Ekstrapolacija iz znanih podatkov

Literatura in vprašanja

- Sistematični pregled umetnih klasičnih nevronskih omrežij: http://page.mi.fu-berlin.de/rojas/neural/index.html.html
- Opišite osnovni model nevronske omrežja.
- Katere vrste klasični umetnih nevronskih omrežij poznamo?
- Kako učimo klasično predkrmiljeno umetno nevronsko omrežje?
- Kakšne probleme rešujemo z umetnimi nevronskimi omrežji?

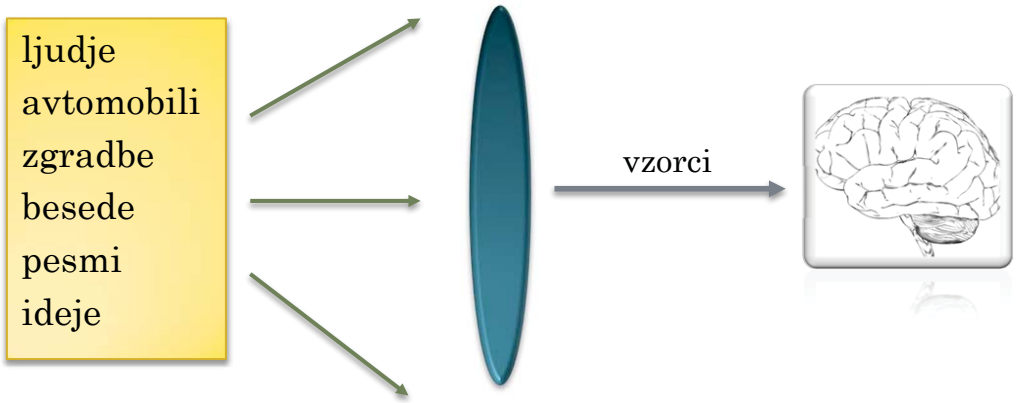
Zgled Hierarhičnega Bayesovega nevronskega modela

- V zadnjem času se precej razvijajo modeli, ki temeljijo na verjetnostni teoriji Bayesovega sklepanja.
- Eden najuspešnejših tovrstnih modelov je hierarhični začasni pomnilnik - HTM



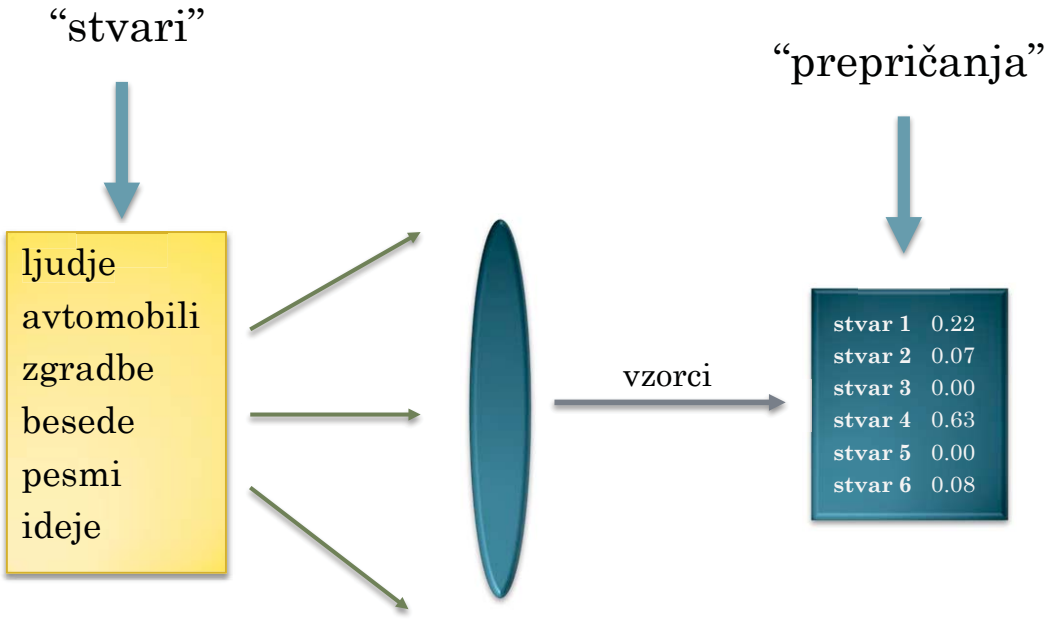
- HTM odkriva stvari, ki se pojavljajo v njegovem svetu.
- HTM sklepa o stvareh na osnovi novih zaznav.

HIERARHIČNI ZAČASNI POMNILNIK (HTM)



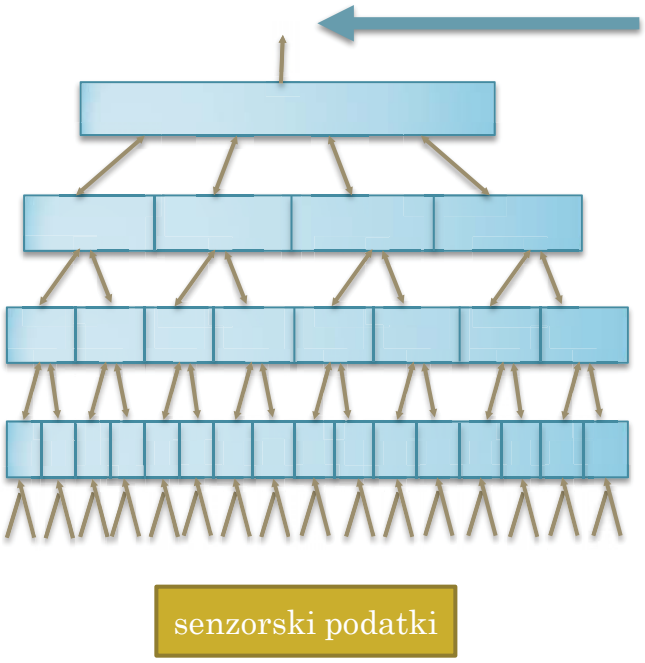
svet zaznave HTM/Cortex

HIERARHIČNI ZAČASNI POMNILNIK (HTM)

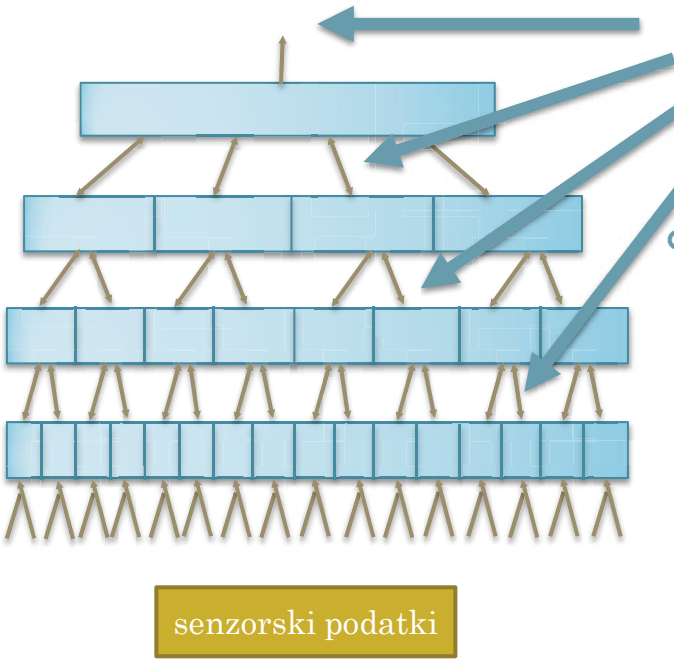


svet zaznave HTM/Cortex

HIERARHIČNI ZAČASNI POMNILNIK (HTM)

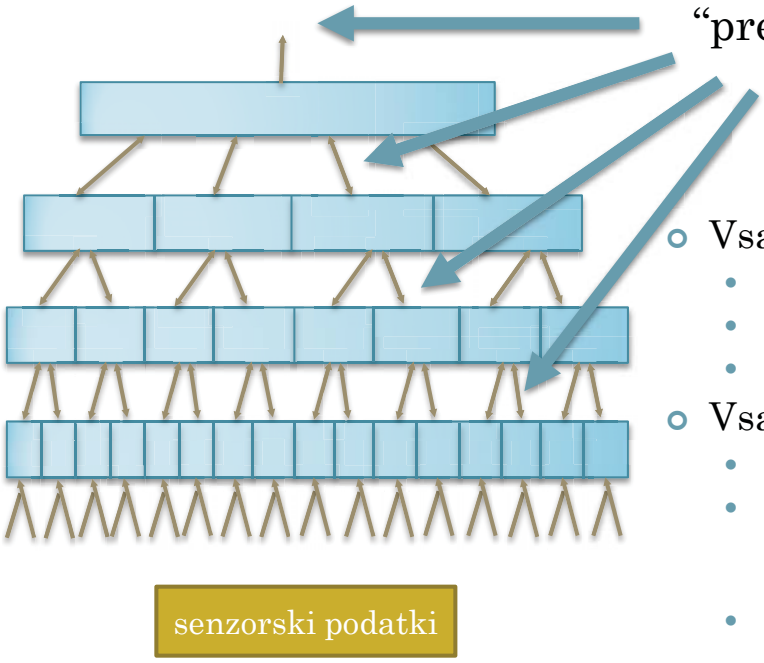


HIERARHIČNI ZAČASNI POMNILNIK (HTM)



- Vsako vozlišče:
  - odkriva „stvari" na svojem vhodu,
  - posreduje prepričanja navzgor in
  - posreduje napovedi navzdol.

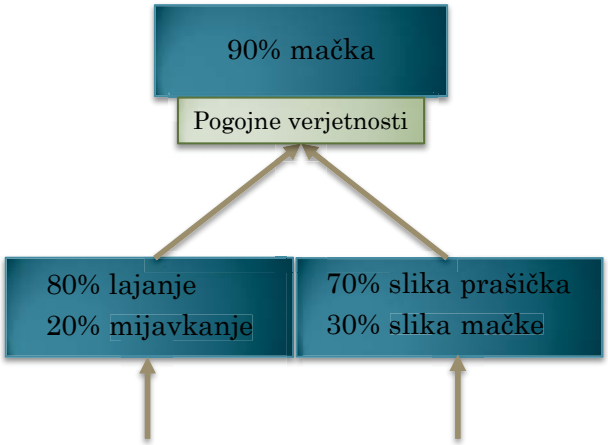
HIERARHIČNI ZAČASNI POMNILNIK (HTM)



- Vsako vozlišče:
  - odkriva stvari na svojem vhodu,
  - posreduje prepričanja navzgor in
  - posreduje napovedi navzdol.
- Vsako vozlišče:
  - shranjuje pogosta zaporedja,
  - spreminjajoči senzorski podatki ustvarjajo stabilna prepričanja navzgor in
  - Stabilna prepričanja ustvarjajo spreminjajoče napovedi senzorskih podatkov navzdol.

ZAKAJ JE HIERARHIJA TAKO POMEMBNA

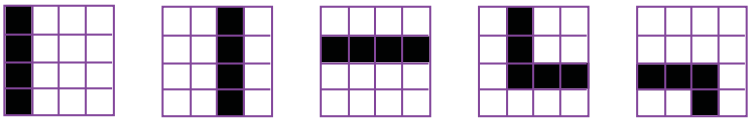
- Deljene predstavitve omogočajo večjo generalizacijo in učinkovitost
- Hierarhija HTM se lahko prilega prostorski in časovni hierarhiji „stvari" v svetu.
- Razširjanje prepričanja omogoča, da se vsa vozlišča hitro uskladijo glede skupnega prepričanja.



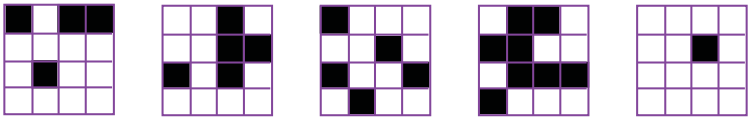
KAKO HTM ODKRIVA STVARI

- Zapomni si pogoste vzorcev (ugotavljanje rojenja) in zavrže redke vzorce (šum).

Pogosti vzorci



Redki vzorci



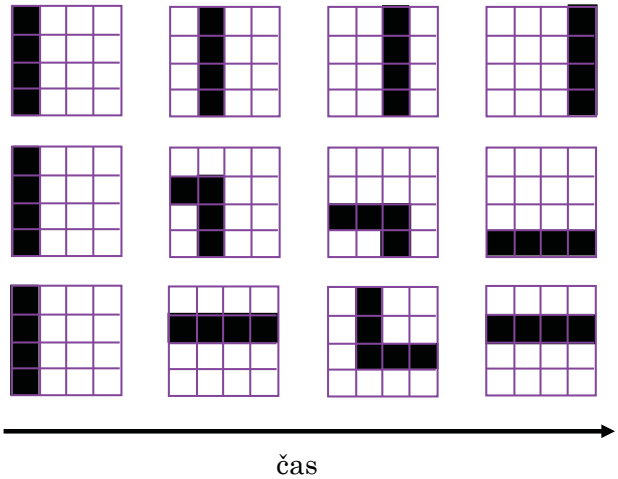
KAKO HTM ODKRIVA STVARI

- Zapomni si pogosta časovna zaporedja vzorcev in zavrže redka zaporedja vzorce (šum).

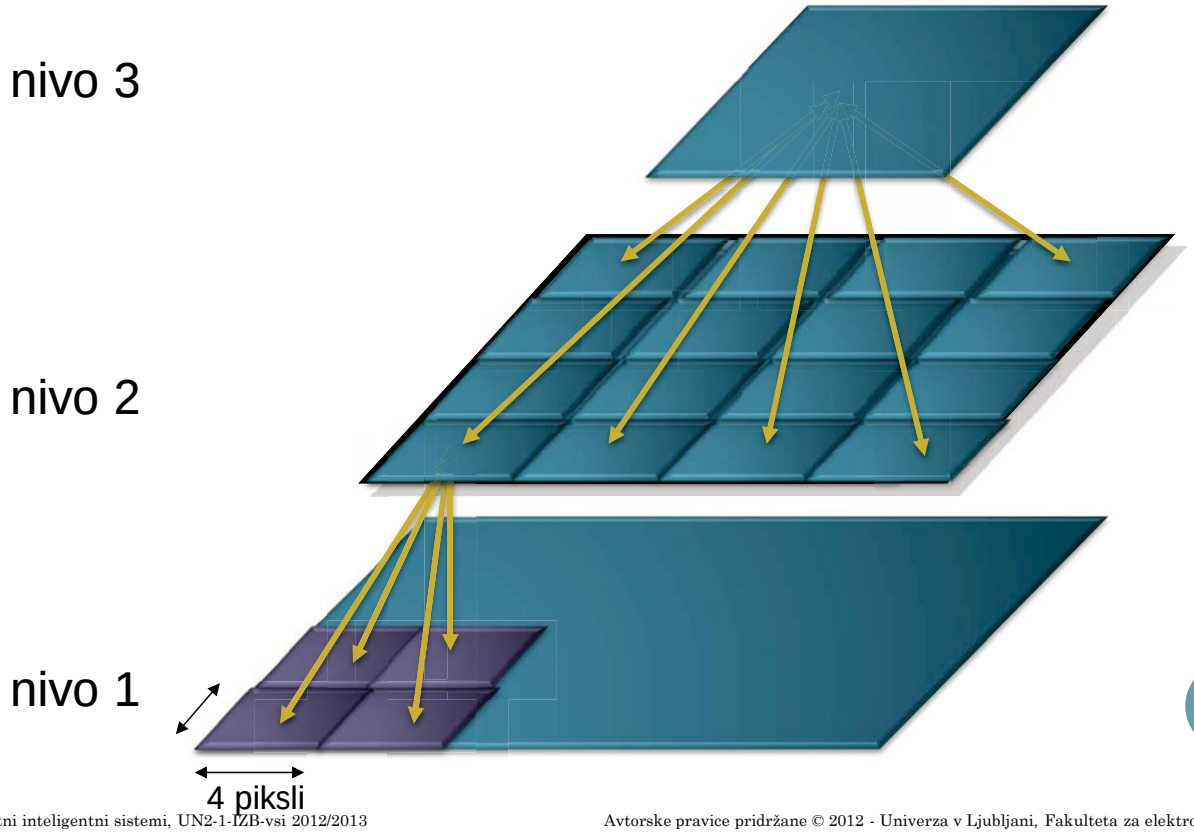
Pogosto zaporedje:
(pripiši stvari)

Pogosto zaporedje:
(pripiši stvari)

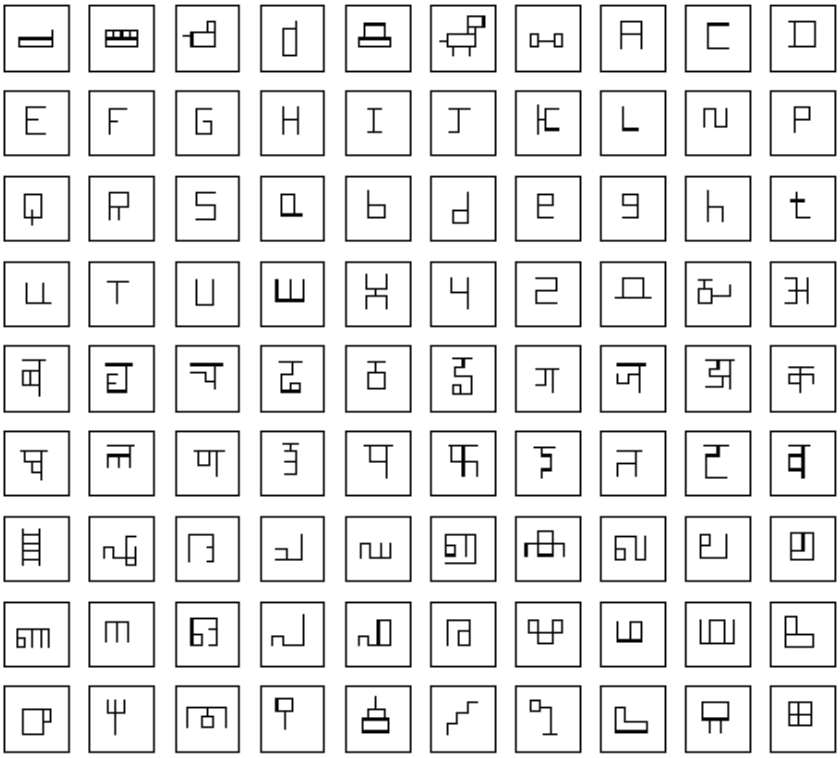
Redko zaporedje:
(zavrzi)



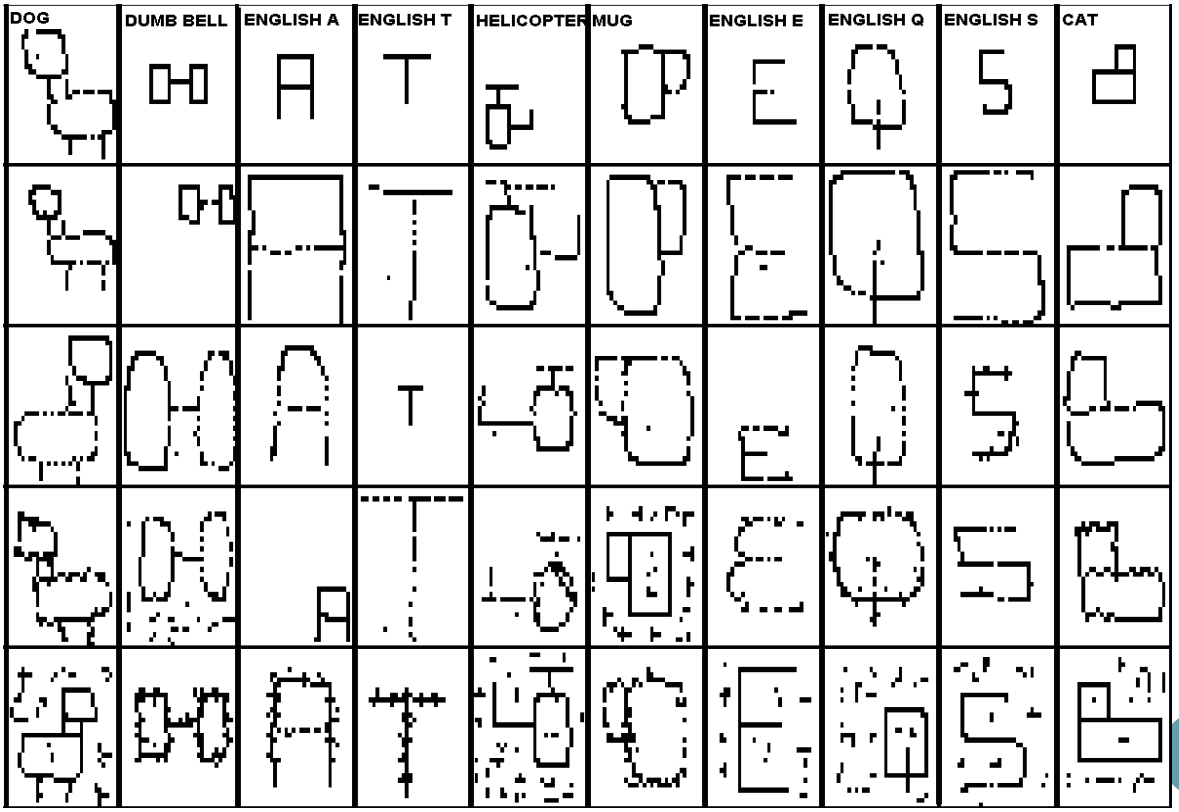
Preprost primer računalniškega vida



Učne slike



Pravilno razpoznane slike



Literatura in vprašanja

- Osnovni opis hierarhičnega začasnega pomnilnika: http://en.wikipedia.org/wiki/Hierarchical_temporal_memory
- Na čem temeljijo hierarhični Bayesovi modeli?
- Opišite osnovne značilnosti hierarhičnega začasnega pomnilnika

GENETSKI ALGORITMI

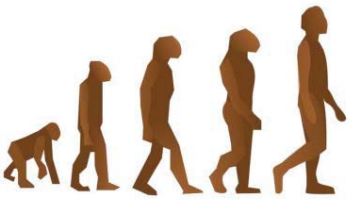
TEME PREDAVANJA

- Uvod
- Oponašanje mehanizmov evolucije
- Določanja genetskega zapisa kromosomov
- Uporaba genetskih algoritmov
- Primeri uporabe genetskega algoritma

GENETSKI ALGORITMI

- Hevristični postopki iskanja in optimizacije
- Oponašanje mehanizmov naravne evolucije
- Spadajo v družino evolucijskih algoritmov
- Osnovne operacije teh algoritmov so (*reprodukcija, križanje, mutacija in selekcija*).

EVOLUCIJA POPULACIJE



- Evolucija populacije posameznikov temelji na kriterijski funkciji njihovih kromosomov, ki ji pravimo funkcija njihove ustreznosti (angl. *fitness function*)
- Višja vrednost funkcije ustreznosti pomeni višjo verjetnost križanja z drugimi posamezniki in s tem prenosa kromosomov v naslednjo generacijo.
- Mutacije omogočajo večjo variabilnost pri iskanju bolj učinkovite populacije.

OPTIMIZACIJA

- Možne rešitve danega problema igrajo vlogo posameznikov neke populacije
- Funkcijo ustreznosti določa okolje, v katerem možne rešitve "živijo" in se reproducirajo.
- Evolucija populacije se nato izvaja z omenjenimi štirimi osnovnimi operacijam.
  (*reprodukcija, križanje, mutacija in selekcija*)

OPTIMIZACIJA PARAMETRIČNIH MODELOV

- Za posameznike v populaciji obravnavamo parametrične modele, pri katerih ima vsak svojo dano vrednosti parametrov.
- Modele predstavimo kot posameznike, katerih genetski zapisi so izpeljani iz vrednosti njihovih parametrov.
- Evolucija sčasoma določi populacijo modelov, ki imajo najvišjo vrednost kriterijske funkcije oziroma funkcije ustreznosti.

KODIRANJE PARAMETROV MODELA V GENSKI ZAPIS KROMOSOMOV

- Dvojiški nizi, ki predstavljajo nize celoštevilskih ali kvantiziranih realnih vrednosti parametrov.

  [00101010|10010101|11101101|00101001|10011110]

- Nizi desetiških števk.

  [42535126|09767283|86192876|76243546|67718231]

- Nizi realnih števil z omejitvami pri križanju.

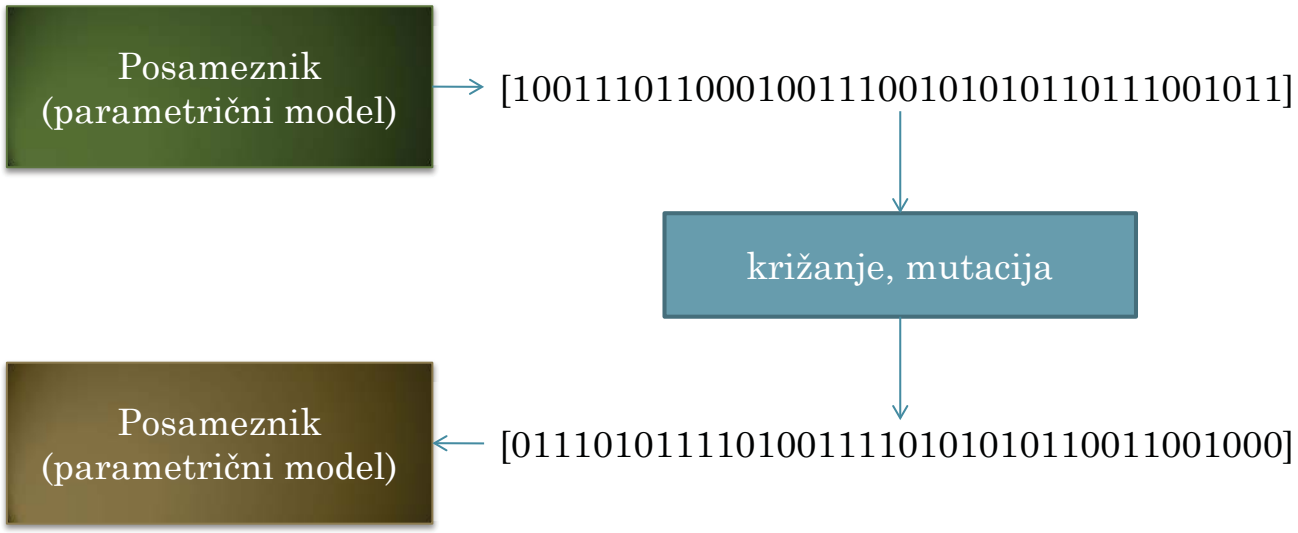
  [1,311; 3,988; 9,123|4,877; 6,712; 0,981|, 123; 9,712; 7,100]

KODIRANJE PARAMETROV MODELA V GENSKI ZAPIS KROMOSOMOV

- Ključen korak za uspešnost genetskih algoritmov je ustrezno prekodiranje parametrov modela v kromosomski niz.
- Križanje in mutacije kromosomov morajo določati spremenjene vrednosti parametrov, ki so že znotraj zaloge vrednosti, smiselnih za obravnavane parametrične modele.
- Pri dvojiškem kodiranju se pogosto uporablja t.i. Grayev kod.

Dec.	Dvo.	Gray
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	111
6	110	101
7	111	100

KODIRANJE PARAMETROV MODELA V GENSKI ZAPIS KROMOSOMOV

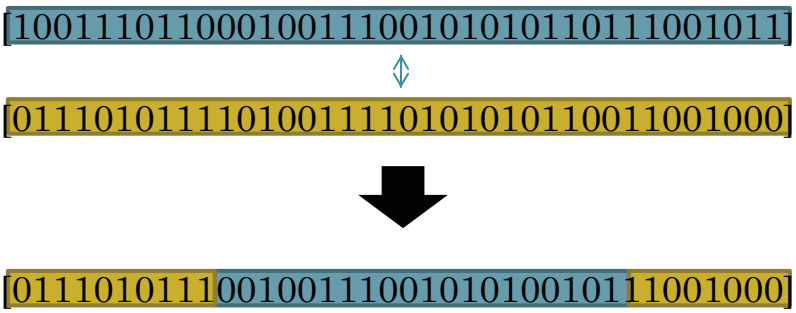


KRITERIJSKA FUNKCIJA EVOLUCIJE

- Kriterijsko funkcijo parametrov modelov samo prevedemo v funkcijo ustreznosti kromosomov posameznikov v populaciji.
- Vrednosti funkcije ustreznosti normiramo na vsoto njenih vrednosti po vsej populaciji.
- S tem pridobimo verjetnost reprodukcije vsakega posameznika s svojim kromosomom.
- Višja vrednost funkcije ustreznosti pomeni višjo verjetnost reprodukcije posameznika.

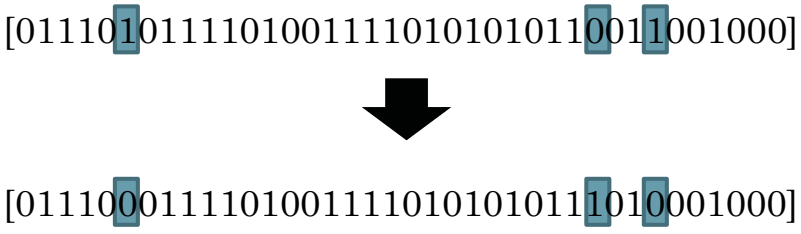
KRIŽANJE

- Kromosoma dveh naključno izbranih posameznikov naključno križamo v enega ali več različnih potomcev.



MUTACIJA

- V kromosomu se z majhno verjetnostjo spremeni kakšen gen.



Primer uporabe genetskega algoritma – Ugotavljanje rojenja vzorcev

- Kromosome tvorijo nizi središčnih vektorjev rojev

$$[\boldsymbol{w}_1, \boldsymbol{w}_2, \dots, \boldsymbol{w}_m] = [w_{11}, w_{12}, \dots, w_{1l}, w_{21}, w_{22}, \dots, w_{2l}, \dots, w_{11}, w_{12}, \dots, w_{ml}]$$

$$J([\boldsymbol{w}_1, \boldsymbol{w}_2, \dots, \boldsymbol{w}_m]) = \sum_{i=1}^{n} \sum_{j=1}^{m} \mu_{ij} d(\boldsymbol{x}_i, \boldsymbol{w}_j)$$

$$\mu_{ij} = \begin{cases} 1 \ , & d(\boldsymbol{x}_i, \boldsymbol{w}_j) = \min\limits_{k=1,\dots,m} d(\boldsymbol{x}_i, \boldsymbol{w}_k) \\ 0 \ , & \text{drugače} \end{cases}$$

- Križanje je dovoljeno le na stikih med vektorji v kromosomih
- Mutacije z majhnimi spremembami poljubne vrednosti

Primer uporabe genetskega algoritma – Iskanje maksimuma izbrane kriterijske funkcije

- http://userweb.elec.gla.ac.uk/y/yunli/ga_demo/
- Iskanje maksimuma funkcije $f_1(x, y)$ s preverjanjem pogojne funkcije $f_2(x, y)$.
- Definicijsko območje obeh funkcij je enako

$$(x, y) \in [0{,}10) \times [0{,}10)$$

- Kromosome kodiramo kot osem-mestno desetiško kodo

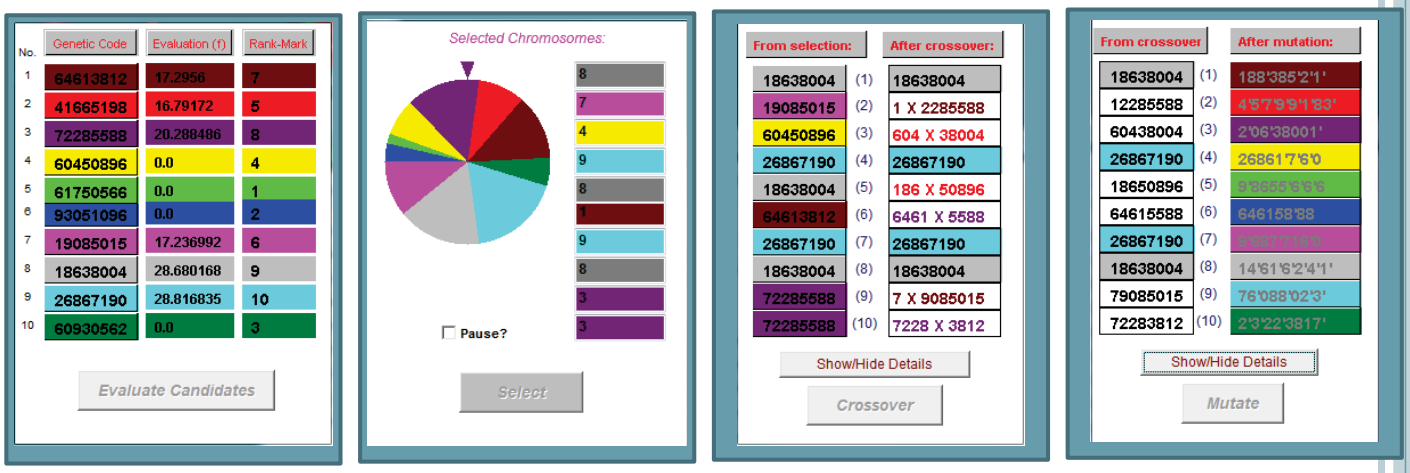
$$(x = 7{,}623\ , y = 3{,}098) \longrightarrow [76233098]$$

- Križanje izvajamo z rezom na enem naključnem mestu

[76233098] [53094145] ⇒ [76233145] [53094098]

PRIMER UPORABE GENETSKEGA ALGORITMA – ISKANJE MAKSIMUMA IZBRANE KRITERIJSKE FUNKCIJE

http://userweb.elec.gla.ac.uk/y/yunli/ga_demo/



UPORABNOST GENETSKIH ALGORITMOV ZA OPTIMIZACIJO PARAMETRIČNIH MODELOV

- Primerni za parametrične modele, pri katerih izračun kriterijske funkcije ni računsko prezahteven.
- Primerni v primerih, ko je kriterijska funkcija zapletena, neodvedljiva po parametrih in ima veliko lokalnih maksimumov, vendar ne sme biti odsekoma konstantna.
- Nimajo jasno definiranega ustavitvenega pravila.
- Drugi podobni algoritmi v določenih primerih dajejo boljše rezultate

PODOBNI ZGLEDI ALGORITMOV

- Simulirano ohlajanje
- Simulacija kolonije mravelj
- Simulacija bakterijskega hranjenja
- Iskanje harmonije
- Inteligentna kapljica vode
- Iskalni algoritem Tabu

  (*uvedba seznama prepovedanih področij možnih rešitev*)
- …

VPRAŠANJA

- Kaj je glavni namen genetskih algoritmov?
- Katere so osnovne operacije teh algoritmov?
- Kako predstavimo posameznike oziroma njihov genetski zapis ter funkcijo njihove ustreznosti?
- Kako poteka optimizacija parametričnih modelov z genetskimi algoritmi?
- Kako kodiramo parametre modela v genski zapis kromosomov?
- Podajte primer optimizacije z genetskim algoritmom.
- Kdaj je optimizacija z genetskimi algoritmi še posebej uporabna?
- Katere so slabosti genetskih algoritmov?
- Podajte nekaj podobnih zgledov optimizacijskih algoritmov.

VEČ-AGENTNI SISTEMI

TEME PREDAVANJA

- Uvod in definicije
- Namerni/hoteč sistem
- Primer abstraktne zgradbe agentov
- Koristnost agenta
- Komunikacija med agenti
- Agentni model BDI
- Primeri več-agentnih platform

Razvoj računalniškega programiranja

- Strojna koda
- Zbirni jezik
- Od platforme neodvisni programski jeziki
- Podprogrami
- Procedure in funkcije
- Abstraktni podatkovni tipi
- Objekti

  …
- Subjekti/Agenti

Drugi pomembni trendi računalništva

- Omrežno računalništvo (grid computing)
- Vseprisotno računalništvo (ubiquitous computing)
- Semantični splet (semantic web)
- Računalništvo v oblaku (cloud computing)
- …

VEČ-AGENTNI SISTEMI IN OMREŽNO RAČUNALNIŠTVO

- Pri obeh gre za razvoj odprtih distribuiranih sistemov.
- Omrežno računalništvo se osredotoča na razvoj

  inter-operabilne infrastrukture in orodja za varno in zanesljivo delitev računalniških virov.
- Več-agentni sistemi se osredotočajo na razvoj

  konceptov, metodologij in algoritmov za avtonomne reševalce problemov, ki se prilagajajo negotovim in dinamičnim okoljem pri doseganju zastavljenih ciljev.

AGENTI - PRVA DEFINICIJA

- Agent je računalniški sistem, ki je sposoben avtonomnega (neodvisnega) delovanja za izpolnjevanje zahtev in potreb uporabnika oz. lastnika.
- Je sposoben samostojnega odločanja o tem, kaj je potrebno storiti za doseganje zastavljenih ciljev.
- Mu ni potrebno nenehno dopovedovati, katere korake mora storiti, da bi dosegel zastavljeni cilj.

VEČ-AGENTNI SISTEM - PRVA DEFINICIJA

- Več-agentni sistem vključuje večje število agentov, ki vzajemno delujejo drug z drugim.
- Sodelujoči agenti na splošno za potrebe uporabnika ne zasledujejo istega cilja.
- Uspešno vzajemno delovanje agentov zahteva njihovo sposobnost sodelovanja, usklajevanja in pogajanja drug z drugim.

VIZIJA UPORABE

- Avtonomne vesoljske in zemeljske sonde
- Preiskovanje medmrežja
- Omrežni komunikacijski sistemi
- Računalniške igre in grafika
- Umetno življenje
- Obrambni sistemi
- Transportni in logistični sistemi

Mikro in makro vizija razvoja

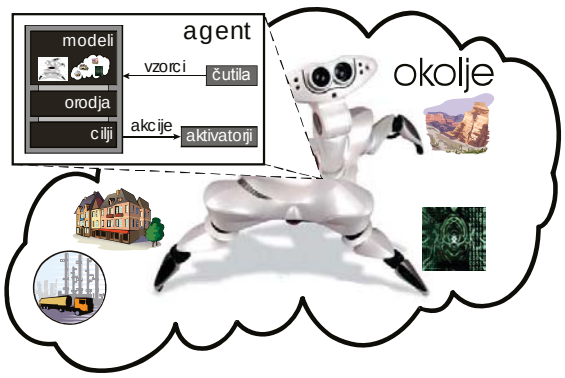
- Razvoj posameznih agentov:

  Kako razviti agenta, ki bo zmožen samostojnega odločanja o potrebnih akcijah za doseganje zastavljenih ciljev?

- Razvoj družbe agentov:

  Kako razviti agente, ki bodo sposobni vzajemnega delovanja (sodelovanja, usklajevanja in pogajanja), še posebej, če si agenti ne delijo istih interesov in ciljev?

Kaj je torej agent?



- Agent je računalniški sistem, ki je sposoben avtonomnega delovanja v nekem okolju, z namenom doseganja zastavljenega cilja.
- Agent je v nenehnem vzajemnem delovanju z njegovim okoljem

zaznaj – odloči – deluj – zaznaj – odloči – deluj – …

PRIMERI TRIVIALNIH AGENTOV

- Termostat:
  - Zastavljeni cilj je vzdrževanje temperature okolja.
  - Delovanje z vklapljanjem in izklapljanjem gretja.
- Poštni bralnik:
  - Zastavljeni cilj je spremljanje pošte in obveščanje.
  - Delovanje z akcijami preko grafičnega vmesnika.

INTELIGENTNI AGENT

- Izkazuje naslednje tri oblike vedenja:
  - Odzivnost na spremembe v okolju.
  - Proaktivnost, ki presega zgolj odzivanje na dogodke in spremembe v okolju, denimo, prepoznavanje priložnosti, za lažje doseganje zastavljenega cilja.
  - Družbenost pri vzajemnem delovanju z drugimi agenti in po potrebi z ljudmi (zmožnost sodelovanja, usklajevanja in pogajanja).

DRUŽBENOST PRI VZAJEMNE DELOVANJU

- Sodelovanje v primeru, ko agent sam ne zmore doseči cilja brez sodelovanja z drugimi agenti.
- Usklajevanje vzajemnih odvisnosti med aktivnostmi različnih agentov, denimo, pri nedeljivih virih.
- Pogajanje za doseganje dogovora pri navzkrižnih interesih, denimo, pri časovnih omejitvah in omejitvah virov.

AGENTI IN OBJEKTI

- Objekti:
  - zaobjema neko notranje stanje,
  - vsebuje metode, ki so operacije na notranjem stanju,
  - komunicira s posredovanjem sporočil.
- Agent:
  - uteleša večjo stopnjo avtonomije (sam odloča, ali bo ali ne bo izvedel zahtevano akcijo),
  - izkazuje kompleksno in prilagodljivo vedenje (odzivnost, proaktivnost, družbenost),
  - niso pasivni izvajalci ukazov.

LASTNOSTI OKOLJA AGENTA

- Dosegljivost proti nedosegljivosti.
- Determiniranost proti nedeterminiranosti.
- Epizodnost proti ne-epizodnosti.
- Statičnost proti dinamičnosti.
- Diskretnost proti zveznosti.

AGENTI KOT NAMERNI/HOTEČ SISTEM

- Pri razlagi človeških aktivnosti uporabljamo izraze, kot:

  Jože je vzel dežnik, ker je bil prepričan, da dežuje in je nameraval/hotel ostati suh.

- Tovrstne izjave se uporabljajo pri napovedovanju in razlagi človeškega obnašanja s pripisovanjem drž/vedenja kot so verujoč, želeč, upajoč, prestrašen, … (folk psychology).

AGENTI KOT NAMERNI SISTEM (INTENTIONAL SYSTEM)

- Izraz „namerni sistem“ je prvi uveljavil filozof Daniel Dennet.
- Namerni sistem prvega reda vsebuje neka prepričanja in želje, vendar ne vsebuje prepričanj in želj o prepričanjih in željah.
- Namerni sistem drugega reda vsebuje prepričanja in želje tudi o prepričanjih in željah.

ALI LAHKO GOVORIMO O NAMERNI/HOTEČI DRŽI PRI STROJIH?

- Pripisovanje prepričanj, svobodne volje, namer, zavesti, želja, in potreb nekemu stroju je smiselno, če takšno pripisovanje izraža enako informacijo pri stroju, kot jo izraža pri človeku.
- To početje je uporabno, če nam omogoči boljše razumevanje zgradbe in delovanja stroja, njegovega preteklega in prihodnjega vedenja ali pomaga pri njegovem popravilu ali izboljšanju.

KAJ LAHKO OPIŠEMO Z NAMERNO DRŽO?

- Primer stikala za luč.

  Stikalo je zelo kooperativne trivialni agent, ki ima zmožnost prevajanja el. toka po želji. Ko verjame/je prepričano, da mi želimo, da tok steče, potem tok prevaja, sicer pa ne.

  Preklop stikala je način komunikacije, s katero izrazimo našo željo.

KAJ LAHKO OPIŠEMO Z NAMERNO DRŽO?

- Pri preprostih sistemih, ki jih dobro razumemo, s takšnimi opisi ne pridobimo nič uporabnega.
- Pri zelo kompleksnih sistemih pa običajni tehnični opisi mehanizma delovanja niso nujno mogoči.
- V teh primerih potrebujemo za opis delovanja globljo abstrakcijo in prispodobe.
- Opis z namerno držo je ena od takšnih abstrakcij.

POST-DEKLARATIVNI PROGRAMSKI ZGLED

- Pri proceduralnem programiranju natančno povemo, kaj naj sistem izvaja.
- Pri deklarativnem programiranju povemo, kaj želimo doseči in podamo relacije med objekti, mehanizmu sklepanja pa prepustimo izpeljavo cilja.
- Pri agentih podamo le visoko-nivojski opis želenega cilja in prepuščamo kontrolnemu mehanizmu, da se sam odloči, kako cilj doseči, pri čemer vemo, da bo deloval v skladu z neko vgrajeno racionalno logiko.

PRIMER ABSTRAKTNE ZGRADBA AGENTOV

- Predpostavimo, da je lahko okolje le v enem od stanj iz končne množice.

$$E = \{e, e', \ldots\}.$$

- Agenti naj imajo na razpolago zalogo akcij, ki spreminjajo stanje okolja.

$$Ac = \{\alpha, \alpha', \ldots\}$$

- Tek agenta v okolju je niz prepletenih stanj okolja in akcij.

$$r : e_0 \xrightarrow{\alpha_0} e_1 \xrightarrow{\alpha_1} e_2 \xrightarrow{\alpha_2} e_3 \xrightarrow{\alpha_3} \cdots \xrightarrow{\alpha_{u-1}} e_u$$

VEDENJE OKOLJA

- Pretvornik stanj okolja, ki tek, ki se konča z neko akcijo, pretvori v novo stanje okolja.

$$\tau : \mathcal{R}^{Ac} \rightarrow \wp(E)$$

- Okolje je lahko odvisno od zgodovine ali ne-deterministično.
- Okolje tako določa trojka $Env = \langle E, e_0, \tau \rangle$

VEDENJE AGENTA

- Agent je preslikava, ki tek, ki se zaključi z stanjem, okolja preslika v novo akcijo.

$$Ag : \mathcal{R}^{E} \rightarrow Ac$$

- Agent torej sprejme odločitev o novi akciji na osnovi zgodovine sistema, ki mu je priča.
- Sistem je par agenta in okolja $\mathcal{R}(Ag, Env)$.

Agentni sistem

- Zaporedje $(e_0, \alpha_0, e_1, \alpha_1, e_2, \ldots)$

  predstavlja tek agenta Ag v okolju $Env = \langle E, e_0, \tau \rangle$

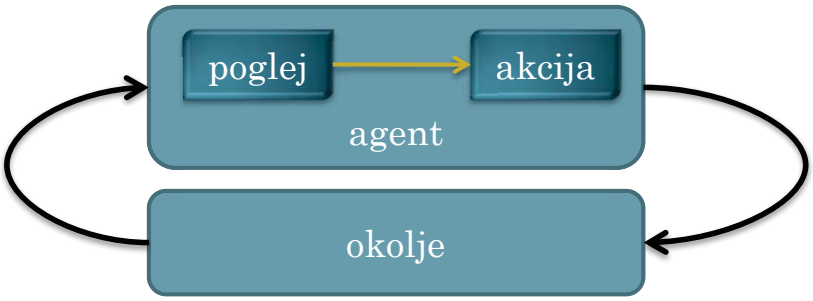
  če:

  1. e_0 predstavlja začetno stanje okolja Env
  2. $\alpha_0 = Ag(e_0)$ in
  3. Za $u > 0$

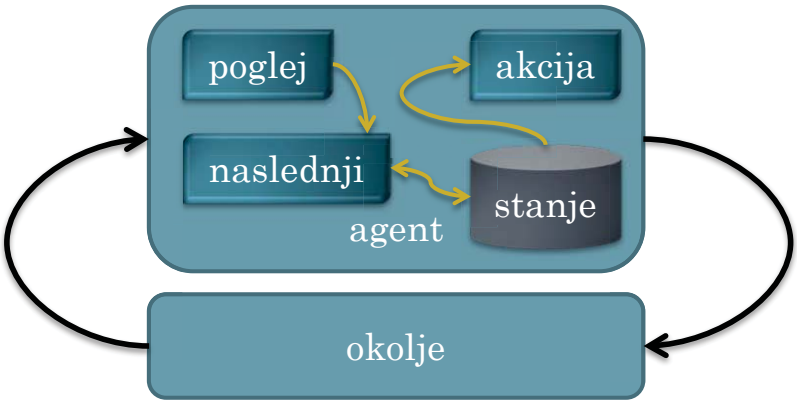
$$e_u \in \tau((e_0, \alpha_0, \ldots, \alpha_{u-1}))$$
$$\alpha_u = Ag((e_0, \alpha_0, \ldots, e_u))$$

Posplošitev agentnega sistema

- Uvajanje zaznavanja okolja



- Uvajanje notranjega stanja agenta



KONTROLNA ZANKA AGENTA

- Pričetek izvajanja v začetnem notranjem stanju
- Ponavljanje naslednjih korakov:
  - Opazuj stanje okolja in ustvari zaznavo skozi *poglej(…)*.
  - Osveži notranje stanje skozi *naslednji (…)*.
  - Izberi akcijo skozi *akcija(…)*.
  - Izvedi izbrano akcijo.

UVAJANJE KORISTNOSTI STANJ

- Ena od možnosti je prirejanje koristnosti posameznim stanjem okolja.
- Naloga agenta je prihod v stanje, ki maksimizira koristnost.
- Določitev naloge je tako določitev preslikave

$$u : E \rightarrow \mathbb{R}$$

ki priredi realno število vsakemu stanju okolja.

Problem določanja koristnosti teka

- Koristnost teka je lahko enaka:
  - Minimalni koristnosti stanja v teku.
  - Največji koristnosti stanja v teku.
  - Vsoti koristnosti stanj v teku.
  - Povprečje koristnosti stanj v teku.
  - …

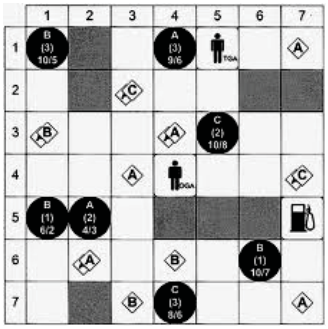
Problem določanja koristnosti teka

- Druga možnost je uvajanja neposredne koristnosti teka.

$$u : \mathcal{R} \rightarrow \mathbb{R}$$

- To zahteva dolgoročni pogled na zgodovino sistema.
- V obeh primerih imamo težave z določanjem vrednosti koristnosti.

PRIMER OPEČNATEGA SVETA



- Simulacija dvorazsežnega sveta z agenti, opekami, ovirami in luknjami.
- Agent se lahko premika v štirih smereh in v primeru če se nahaja zraven opeke, jo lahko porine naprej.
- Agenti lahko zapolnijo luknje z opekami.
- Opečnati svet se spreminja z naključnim pojavljanjem ali izginjanjem lukenj ali ovir.

PRIMER FUNKCIJE KORISTNOSTI TEKA

- Funkcija koristnosti je definirana kot razmerje med številom zapolnjenih lukenj med tekom in številom lukenj, ki so se pojavile med tekom.
- Če agent zapolni vse luknje je koristnost enaka ena, če ne zapolni nobene pa nič.

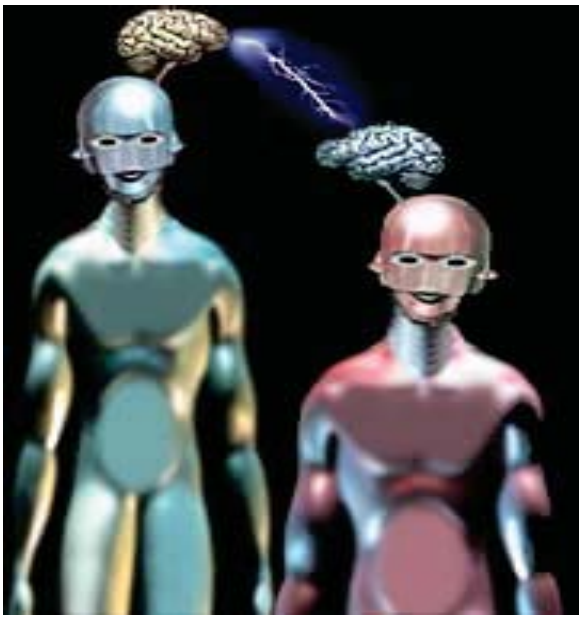
UVAJANJE PRIČAKOVANE KORISTNOSTI TEKA

- Uvedemo verjetnost, da se tek zgodi, če je agent umeščen v dano okolje.
- Pričakovana koristnost agenta je seštevek koristnosti vseh možnih tekov agenta, pomnoženih z verjetnostjo teh tekov.
- Za optimalnega agenta se šteje tisti agent, ki maksimizira svojo pričakovano koristnost.

VPRAŠANJA – 1. DEL

- Kakšna je definicija agenta in več-agentnega sistema?
- Kakšna je vizija uporabe več-agentnih sistemov?
- Podajte primer trivialnih agentov.
- Kakšne oblike vedenja naj bi izkazoval inteligentni agent?
- Kakšna je razlika med programskimi objekti in agenti?
- Ali lahko govorimo o namerni/hoteči drži pri strojih?
- Opišite osnovno kontrolno zanko agenta.
- Opišite problem določanja koristnosti teka agenta.

KOMUNIKACIJA MED AGENTI



KOMUNIKACIJA MED AGENTI

- Komunikacija med agenti izhaja iz Teorija govornih dejanj (speech act theory).
- Ta teorija obravnava jezik kot sredstvo s katerim ljudje dosegajo svoje cilje in namere.
- Izjave obravnava kot svojevrstna fizična dejanja, za katere se zdi, da spremenijo stanje sveta (npr. napoved vojne).
- Predlagani so standardi zapisa med-agentnega jezika, kot je FIPA (*Foundation for Intelligent Physical Agents*)

GOVORNA DEJANJA

- Predpostavlja se več vrst osnovnih govornih dejanj, kot so:
  - informacije: informiranje o stanju, (*Zunaj dežuje*),
  - ukazi: poskus pripraviti poslušalca izvesti nekaj, (*Pripravi čaj!*),
  - obveze: obvezovanje govorca k nekemu dejanju, (*Obljubljam, da …*),
  - izrazi: izražanja govorčevega duševnega stanja (*Hvala!*),
  - deklaracije: deklariranje novega stanja okolja (*Napovedujem vojno!*),

TOPOLOGIJA GOVORNIH DEJANJ

- Govorno dejanje ima dve glavni komponenti:
  - izvršni glagol: zahtevati, informirati, poizvedovati, …
  - vsebina izjave: (npr. „*Vrata so zaprta*“).

- performative = request
  content = “the door is closed”
  speech act = “please close the door”
- performative = inform
  content = “the door is closed”
  speech act = “the door is closed!”
- performative = inquire
  content = “the door is closed”
  speech act = “is the door closed?”

POMENSKA ANALIZA IN SISTEM NAČRTOVANJA

- Kako ugotovimo, da nekdo nekaj zahteva, informira, …
- Eden od sistemov načrtovanja temelji na formalizmu *predpogoj-izbris-dodaj:*
  - *predpogoj(a,s)* je resničen, če je akcijo *a* možno izvesti v situaciji *s*,
  - *izbrisan(p,a,s)* je resničen, če se predpogoj *p* lahko izbriše, ko je akcija *a* izvedena v situaciji *s*,
  - *dodaj(p,a,s)* je resničen, če se predpogoj *p* lahko doda, ko je akcija *a* izvedena v situaciji *s*.

PRIMER POMENSKE ANALIZE „ZAHTEVE“ PRI SISTEMU NAČRTOVANJA

zahteva(g,p,a)

- Pred govornim dejanjem
  - govorec *g* verjame, da poslušalec *p* zmore akcijo *a,*
  - govorec *g* verjame, da poslušalec *p* verjame, da zmore akcijo *a,*
  - govorec *g* verjame, da govorec *g* želi akcijo *a*.
- Po govornem dejanju
  - poslušalec *p* verjame, da govorec *g* želi akcijo *a*.

PRIMERI JEZIKOV ZA KOMUNIKACIJO MED AGENTI

- Obstaja nekaj primerov jezikov za komunikacijo med agenti (angl. *agent communication languages* - ACL).
- Eden prvih in zelo znanih je jezik KQML predlagan v okviru ARPA iniciative.
- Vsebuje dva dela:
  - jezik KQML (*the knowledge query and manipulation language*)
  - format KIF (*the knowledge interchange format*)

PRIMERI IZVRŠNIKOV V KQML

- KQML določa več ključnih komunikacijskih izvršnih glagolov:
  - `ask-if` („*Je res, da …?*“)
  - `perform` („*Prosim, izvedi naslednjo akcijo …*“)
  - `tell` („*Res je ,da …*“)
  - `reply` („*Odgovor je, da …*“)
- S KIF pa izražamo vsebino izjave.

PRIMER KQML/KIF DIALOGA

```
A to B:   (ask-if
            (> (size chip1) (size chip2)))
B to A:   (reply true)
B to A:   (tell (= (size chip1) 20))
B to A:   (tell (= (size chip2) 18))
```


FIPA STANDARD

- Program agentnih standardov FIPA
- Jedro standarda je prav ACL, ki je podoben KQML.
- Vsebuje 20 izvršnih glagolov, vodenje evidence sporočil in prenašanje vsebine izjav.

```
(inform
  :sender       agent1
  :receiver     agent5
  :content      (price good200 150)
  :language     sl
  :ontology     hpl-auction
)
```

IZVRŠNI GLAGOLI PO FIPA STANDARDU

performative	passing info	requesting info	negotiation	performing actions	error handling
accept-proposal			x		
agree				x	
cancel		x		x	
cfp			x		
confirm	x				
disconfirm	x				
failure					x
inform	x				
inform-if	x				
inform-ref	x				
not-understood					x
propose			x		
query-if		x			
query-ref		x			
refuse				x	
reject-proposal			x		
request				x	
request-when				x	
request-whenever				x	
subscribe		x			

PROGRAMSKI AGENTNI MODEL

- Običajno računalniško programiranje temelji na preprostem agentnem modelu.
- Inteligentni agenti pa naj bi temeljili na bolj striktnem programskem modelu, ki predvideva njegova mentalna stanja, kot so prepričanja, želje in namere (angl. *beliefs, desires and intentions - BDI*).
- Ta programski model omogoča višje nivoje abstrakcije, kot običajno programiranje in je bližje človekovemu načinu umovanja.

PROGRAMSKI AGENTNI MODEL BDI

- Temelji na Bratmanovi teoriji razlage človeškega obnašanja z modeliranjem njegovih prepričanj, želja in namer.
- Predvsem gre za mehanizem, ki loči aktivnosti izbire načrta delovanja iz zbirke načrtov in aktivnosti samega izvajanja že izbranih načrtov.
- Agent tako lahko uravnoveša porabljeni čas med „razmišljanjem“ o tem, kaj naj naredi, in dejanskim delovanjem po izbranem načrtu.

SESTAVINE PROGRAMSKEGA AGENTNEGA MODELA BDI

- Prepričanja: Notranja stanje agenta, ki navadno vključujejo mehanizem sklepanja, ki z veriženjem omogoča tvorjenje novih prepričanj o svetu in agentu samem iz obstoječih prepričanj.
- Želje: Ciljno motivirano stanje agenta, ki jih navadno podamo kot neke logične cilje.
- Namere: Preudarno stanje agenta, ki predstavlja neko njegovo namero v obliki načrta delovanja. Namere so želje, ki jih poskuša agent le do neke mere izpolniti.
- Dogodki: Prožijo spreminjanje prepričanj, spreminjanje ciljev ali drugo izbiro načrtov.

SESTAVINE PROGRAMSKEGA AGENTNEGA MODELA BDI

- Prepričanja: Urejena v obliki baze prepričanj, ki je navadno neka baza znanja, ki pa ne odraža nujno resničnega sveta.
- Želje: Urejena v nek seznam logičnih ciljev.
- Namere: Urejene v seznam načrtov, ki jih predstavljajo zaporedja možnih akcij.
- Dogodki: Se ustvarjajo iz okolja ali so proženi notranje, kot posledica sprememb notranjega stanja.

SPLOŠEN TOLMAČ ZA PROGRAMSKI MODEL BDI

- Inicializiraj notranja stanja
- Ponavljaj:
  - *seznam_opcij:* ***tvori_opcije****(vrsta_dogodkov)*
  - *izbrana_opcija:* ***preudari****(seznam_opcij)*
  - ***osveži_namere****(izbrana_opcija)*
  - ***izvedi_akcije****()*
  - ***pridobi_nove_zunanje_dogodke****()*
  - ***zavrzi_neuspešne_drže****()*
  - ***zavrzi_nesmiselne_drže****()*

PROGRAMSKI JEZIKI ZA UDEJANJANJE PROGRAMSKEGA MODELA BDI

- Navadno se izbira med funkcionalnimi oz. deklarativnimi programskimi jeziki.
- Več-agentne platforme, kot je JADE, ne vključujejo višjih programskih jezikov.
- Mehanizem sklepanja pri vodenju baze prepričanj in zasledovanju ciljev lahko udejanjimo s Prologom.
- Obstajajo predlogi in udejanjeni tolmači za več deklarativnih programskih jezikov za programski model BDI.

ABSTRAKTNI PROGRAMSKI JEZIK AGENTSPEAK(L)

- Anand S. Rao (1996). AgentSpeak(L): *BDI Agents speak out in a logical computable language*.
- Razširja predikatni račun z definicijo *želje* agenta, da doseže izpolnitev predikata ali poskuša ugotoviti ali je izpolnjen.
- Predvideva mehanizem hkratne obravnave več akcijskih načrtov, ki jih agent ustvarja ali opušča glede na zabeležene dogodke.
- Primera udejanjenj tega jezika sta Jason in 2APL, ki uporabljata prolog za manipulacijo baze prepričanj, ki ga nadgrajujeta z višjim jezikom pravil.

ABSTRAKTNI PROGRAMSKI JEZIK AGENTSPEAK(L)

- Verjetja so predikati (npr. `location(robot,a)`).
- Cilj je stanje, ki ga agent želi doseči. Obravnava se dve vrsti dogodkov: *dosežek* in *preizkus*.
- Če je `location(robot,a)`verjetje, potem je `!location(robot,a)` dosežek in `?location(robot,a)` preizkus.

ABSTRAKTNI PROGRAMSKI JEZIK AGENTSPEAK(L)

- Prožilni dogodek so dogodki dodajanja/brisanja verjetij ali ciljev.
- Če je `b(t)` atom verjetja, `!b(t)` in `?b(t)` cilja, potem so možni prožilni dogodki:

```
+b(t), -b(t), +!b(t), -!b(t), +?b(t), -?b(t)
```

Primer agentovih načrtov v jeziku AgentSpeak(L)

```
+location(waste,X):location(robot,X) &
                   location(bin,Y)
                   <- pick(waste);
                      !location(robot,Y);
                      drop(waste).              (P1)


+!location(robot,X):location(robot,X) <- true.   (P2)

+!location(robot,X):location(robot,Y) &
                    (not (X = Y)) &
                    adjacent(Y,Z) &
                    (not (location(car, Z)))
                          <- move(Y,Z);
                             +!location(robot,X). (P3)
```


Programski jeziki za udejanjanje programskega modela BDI

- JADEX (*http://vsis-www.informatik.uni-hamburg.de/projects/jadex*)
- *JASON (AgentSpeak(XL))* (*http://jason.sourceforge.net/Jason/Jason.html*)
- 3APL (http://www.cs.uu.nl/3apl/)
- 2APL (AgentSpeak) (*http://apapl.sourceforge.net/*)
- *JACK* Agent Language (JAL) (*http://www.agent-software.com.au/products/jack*)
- …

POMANJKLJIVOSTI IN OMEJITVE PROGRAMSKEGA MODELA BDI

- Nezmožnost učenja iz preteklih drž pri prilagajanju na nove razmere.
- Ni eksplicitnega modeliranja sodelovanja, usklajevanja in pogajanja z drugimi agenti.
- Ni podpore preudarjanju s pogledom v prihodnost, kar lahko vodi agenta v pasti iz katerih ni rešitve.
- Ni jasno, če bi prepričanjem, nameram in željam morda ne bilo dobro dodati še kakšno vedenje oziroma držo.

VPRAŠANJA – 2. DEL

- Zakaj komunikacija med agenti temelji na teoriji govornih dejanj?
- Katera osnovna govorna dejanja se upošteva?
- Podajte primer pomenske analize in sistema načrtovanja.
- Podajte primere jezikov za komunikacijo med agenti.
- Kaj določa FIPA standard?
- Opišite programski agentni model BDI.
- Kaj so osnovne značilnosti abstraktnega programskega jezika AgentSpeak(L)?