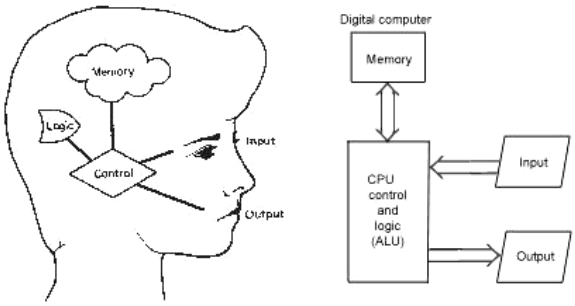


"Cogito ergo sum"

prof. dr ing jaroslav e. poliščuk

ekspertni sistemi

- ***metode za razvoj ekspertnih sistema***
- ***arhitektura ekspertnih sistema***
- ***izgradnja ekspertnih sistema***

Prof. dr ing Jaroslav E. Poliščuk
EKSPERTNI SISTEMI

Recezenti:
1. Prof. dr Petar Hotomski, profesor Tehničkog fakulteta "Mihajlo Pupin" u Zrenjaninu, Univerzitet u Novom Sadu

2. Prof. dr Vladan Devedžić, profesor Fakulteta organizacionih nauka u Beogradu, Univerzitet u Beogradu

Kompjuterska obrada knjige:
Prof. dr ing Jaroslav E. Poliščuk, ETF Podgorica

E-mail: jaroslav@cg.ac.yu

PREDGOVOR

Knjiga *Ekspertni sistemi* (ES) se sastoji od četrnaest poglavlja. U prvom poglavlju, koje nosi naziv ***Opšte o prirodnoj i vještačkoj inteligenciji,*** su dane opšte postavke iz navedene oblasti, kao uvod u naredna poglavlja. Nastojalo se što potpunije definisati karakteristike prirodne inteligencije, kako bi se olakšalo izlaganje vještačke inteligencije (VI).

Drugo poglavlje ***Znanje i zaključivanje*** obuhvata razvrstavanje znanja, način izražavanja znanja pomoću *iskaza*. Objašnjen je *mehanizam zaključivanja,* korištenjem deduktivnih i induktivnih logičkih argumenata.

Pitanje ***Automatskog dokazivanja teorema*** (ADT), sa stanovišta VI, je veoma značajano, jer u sebi sadrži bitne komponente inteligentnog ponašanja. Za ADT postoji opšti algoritam, dok je za metodu rezolucije dokazana njezina potpunost. Univerzalni rezolucijski sistem zaključivanja (dokazivač) omogućio je izgradnju programskog jezika PROLOG i mehanizama zaključivanja u ES, što omogućava izgradnju skeletnih sistema (*shell*) ES. Efikasnost takvih sistema dovedena je do nivoa praktične primjene.

Problem procesiranja znanja i njegovo prikazivanje su osnova na kojoj se zasniva kompletna teorija vještačke inteligencije, pa tako i teorije izgradnje ekspertnih sistema, odnosno ***Arhitektura ekspertnih sistema.*** Ova problematika je posebno istraživana sredinom 70-tih godina, tako da je do danas razvijen čitav niz metoda i programskih alata.

Produkcioni sistemi se sastoje od skupa parova tipa "uslov – akcija", koji se nazivaju produkciona pravila ili pravila zaključivanja, baze podataka stanja sistema smještenih u radnu memoriju i procedura za izvršavanje produkcionih pravila, odnosno mehanizma zaključivanja. Produkcioni sistemi se najčešće primjenjuju za razvoj ES i sistema kognitivne arhitekture.

Predikatski račun omogućava rad i sa objektima i sa propozicijama i predstavlja standardnu metodu predstavljanja znanja, odnosno izgradnje ***Ekspertnih sistema zasnovanih na matematičkoj logici***. Predstavlja formalni logički jezik sa vlastitom sintaksom (strukturom izraza) i gramatikom, koji omogućava uvođenje relacija među objekte i primjenu mehanizma zaključivanja kod procesiranja znanja.

Semantičke mreže opisuju kategorije objekata (čvorovi u mreži) i relacije među njima (veze). Glavni oblik zaključivanja je nasleđivanje. Opisuju se prototipovi, a moguće je opisati i izuzetke (nemonotona logika). Razvoj semantičkih mreža doveo je do nastanka posebne logičke strukture, koja je nazvana ramovi znanja. ***Ramovi znanja*** (*FRAMES*) su kompletne i zaokružene logičke strukture, nalik na klase u objektno orijentisanom programiranju, koje vezuje

jedinstvo vremena, radnje i objekta, slično scenskim prikazima. Predstavljaju pogodno sredstvo za prikazivanje podataka i relacija.

Eksperti se često oslanjaju na uopštene pojmove i uopštena razmišljanja u procesu rješavanja pojedinih problema. U takvim slučajevima eksperti opisuju probleme koristeći neodređene, neizvjesne ili višesmislene termine, a za to su pogodni ***Fazi ekspertni sistemi***.

Neuronske mreže i ekspertni sistemi pokazuju dobre rezultate prilikom predviđanja i modeliranja sistema gdje fizički procesi nisu jasni ili su veoma kompleksni. Prednost NM leži u visokoj elastičnosti prema poremećajima u ulaznim podacima i u sposobnosti učenja.

Iako se ***Genetički algoritmi*** (GA) intenzivnije izučavaju od 1975. godine, činjenica je da oni u poslijednje vrijeme doživljavaju sve veću popularnost. Evoluciona paradigma koju eksploatišu GA postaje široko prihvaćena, kao put rješavanja određenih klasa problema.

Agenti, multiagenti i inteligentni agenti predstavljaju softver koji ima sposobnosti da samostalno, bez intervencije korisnika, izvršava postavljeni zadatak i izvještava korisnika o završetku zadatka ili pojavi očekivanog događaja.

Osnovne pogodnosti ***Savremenih inteligentnih sistema*** su jednostavna i prirodna komunikacija, mogućnost rada sa opštim i problemski usmjerenim znanjem, mogućnost objašnjavanja svojih akcija, adaptivnost i sposobnost učenja.

Izgradnja ES nije nimalo lak posao i u pojedinim slučajevima ga je skoro nemoguće obaviti ukoliko se ne raspolaže inteligentnim alatima i posebnim sredstvima, koji taj posao olakšavaju i skraćuju njegovo vremensko trajanje.

Osim autorovog iskustva u projektovanju ES, knjiga je, uglavnom, rađena na osnovu dostupne literature i obrađuje savremeni pristup projektovanju ES. Izvori, na osnovu kojih je nastala ova knjiga, jednim dijelom su navedeni u samom tekstu knjige, dok je cjelovit pregled dan u okviru pregleda korištene literature. Zahvaljujem se autorima čiji su manji dijelovi radova, u djelimično izmijenjenom obliku, ušli u sastav ove knjige. Ukoliko njihovo autorstvo nije na svakom mjestu jasno naglašeno, taj propust će rado biti naknadno otklonjen. U pojedinim slučajevima bilo je nemoguće razgraničiti autorstvo pojedinih tekstova, jer se isti ili sličan tekst mogao naći u različitim izvorima.

Knjiga je namjenjena studentima postdiplomskog specijalističkog studija Elektrotehničkog fakulteta u Podgorici, a može poslužiti kao potrebna literatura svima koji se bave razvojem savremenih IS.

Posebnu zahvalnost autor duguje **prof. dr Vladanu Devedžiću i prof. dr Petru Hotomskom,** koji su, pored detaljno obavljene recenzije knjige, dali niz korisnih savjeta, kako bi knjiga bila što bolja.

U Podgorici, maj 2004. godine *Autor*

S A D R Ž A J

U ovom poglavlju:

- ***Pojam inteligencije***
- ***Karakteristike inteligencije***
- ***Pojam i definicija VI***
- ***Razvojni put VI***
- ***Podjela VI***
- ***Karakteristike VI***

1

OPŠTE O PRIRODNOJ I VJEŠTAČKOJ INTELIGENCIJI

1.1. POJAM INTELIGENCIJE

U *Aristotel*-ovoj (384-322) knjizi "Logika" je ispitivano da li se za neku pretpostavku može reći da je istinita, zato što se odnosi na druge stvari za koje se zna da su istinite. Ako se zna da su "svi ljudi smrtni" i da je "Sokrat čovjek", može se zakljuciti da je "Sokrat smrtan". U *Rene Dekart*-ovoj (1596-1650) knjizi "Meditacije" se razdvaja proces mišljenja od fizičkog svijeta. Čak je i za njegovo vlastito postojanje tražio potvrdu preko mišljenja: "*Cogito ergo sum*" ("Mislim, dakle postojim").

Vilhelm Lajbnic (1646-1716) uvodi sistem formalne logike i konstruiše mašinu za automatsko računanje. *Leonard Ojler* (1707-1783) uvodi teoriju grafova koja predstavlja jedno od osnovnih oruđa u vještackoj inteligenciji. *La Mettrie* (XVIII vijek) u knjizi "*L'Homme Machine*", posmatrajući mehaničke naprave kao što su mehanička patka i svirač flaute, iznosi pretpostavku da bi se jednog dana mogao napraviti mehanički čovjek koji govori. U devetnaestom vijeku se susreću prva mehanički programabilna računarska mašina *Carsla Bebix*-a (1792-1871), prvog programera *Adu* (*Bajron*) *Lovelsi* (1815-1837), *Xorxa Bul*-a i njegovu binarnu algerbu koja predstavlja osnovu današnjih digitalnih računarskih mašina.

U dvadesetom vijeku jedan od prvih radova vezanih za mašinsku (vještačku) inteligenciju, prije svega u odnosu na današnje digitalne računare, je rad Alana

Tjuringa *(Alan Turing)* "Računarske mašine i inteligencija", odnosno test za "mjerenje inteligencije". Mašina predstavlja jedini do sada priznati standard u ovom domenu.

Izraz "vještacka inteligencija" se koristi od sredine pedesetih godina. Za uvođenje ovog izraza smatra se da je najzaslužniji *John Mac Carty*. Prvi put izraz vještačka inteligencija se čuo u ljeto 1956. godine na sastanku tadašnjih pet vodećih naučnika iz oblasti računarskih nauka. Sasatanak je održan na *Dartmouth Collegu*, u Hanoveru, Novi Hemšir. Sasatanku su prisustovali: Klod Šenon, Marvin Minski, Xon Mekarti, Alen Novel i Herbert Sajmon. Sam izraz je uveden da bi se što više naglasile i što lakše objasnile, mogućnosti budućih računara i računarskih programa.

Intelekt je oduvjek bio vezan za čovjeka, jedino živo biće na našoj planeti koje posjeduje to svojstvo. Za pojam intelekta u literaturi se mogu naći slijedeća tumačenja: "Intelekt je sposobnost shvatanja značenja, pravilno razumjevanje i bistrina uma, sposobnost mišljenja, oštroumnost, pamet, itd". Vrhunac pojma inteligencije je kad se navede rješavanje problema, jer vrhunsko rješavanje problema neminovno zahtjeva inteligenciju, što ovom pojmu daje empirijsku notu. Jednu od relativno prihvatljivih definicija pojma inteligencije dao je profesor *Christopher Evans* u svojoj čuvenoj knjizi "Moćni mikro" *("The Mighty Micro"*) (1979):

Inteligencija je sposobnost sistema da se prilagodi promjenama u svijetu i što je ta sposobnost veća, odnosno profinjenija snaga prilagođavanja, sistem je inteligentniji.

U navedenoj definiciji se spominje sistem, što daje dosta široko značenje, mada se u podsvjesti misli na čovjeka. Otuda je potreban poseban napor da se sve to premjesti i sve te karakteristike pripišu neživoj tvorevini - mašini. Da bi se ovaj problem donekle shvatio, potrebno je barem pokušati dati osnovne karakteristike inteligencije. Ukoliko se karakteristike inteligencije pravilno definišu, utoliko će biti lakše izlaganje osnova vještacke inteligencije.

1.2. KARAKTERISTIKE INTELIGENCIJE

U narednom tekstu će se pokušati objasniti nekoliko osnovnih karakteristika inteligencije, koje su danas opšte prihvaćene, i to:

- imitacija dijaloga,
- rješavanje svih varijanti problema,
- rješavanje netrivijalnih zadataka,
- učenje,
- ekstrapolacija.

Konačan broj karakteristika i njihove definicije još su daleko od današnjih saznanja i one treba da budu predmet daljih istraživanja.

Imitacija dijaloga

Imitacija dijaloga je karakteristika koja potiče od čuvenog engleskog matematičara Tjuringa (*Turing*). On je predložio da se sistem, čovjek ili računar, smatra inteligentnim ako se ne može uočiti, u toku konverzacije, s kim se vodi dijalog, sa čovjekom ili sa računarom.

Tjuringov test je naziv ovog testa inteligencije, za koji su bili razvijeni posebni dijalogni programi. U toku provođenja testa sa bolesnicima, većinu sagovornika je bilo teško ubijediti da "razgovaraju" sa računarom, odgovarali su "da ih on tako dobro razumije". Međutim, vrlo brzo se došlo do zaključka da je ovakav test samo imitacija intelekta i da je jedan od potrebnih, ali ne i dovoljnih uslova za inteligenciju. Dalji razvoj dijalognih programa, kao i razvoj Tjuringovog testa prije tridesetak godina, vezivan je za rješavanje određenih logičkih ili računarskih problema.

Posebno se isticala mogućnost igranja šaha, što je u ne tako davnoj prošlosti zadiralo u domen naučne fantastike. Duži niz godina postoji veći broj programa koji igraju veoma dobar šah već i na kućnom rečunaru. Time se pokazalo da je područje vještačke inteligencije vrlo ćudljivo područje nauke, neki problemi za koje se smatralo da su nesavladivi postali su jednostavni i lagani, dok su drugi, za koje se mislilo da su jednostavni i lagani, postali praktično nesavladivi.

Rješavanje svih varijanti problema

Rješavanje svih varijanti problema predstavlja slijedeću karakteristiku inteligencije. Ovdje se misli na rješavanje svih varijanti nekog problema, ali ne lošije od čovjeka. Ova karakteristika, nekada veoma popularna, danas se smatra

nedovoljnom i nepotpunom. Postavlja se pitanje o kakvom problemu i kakvim varijantama se govori. Probleme ne treba tražiti u području numerike, jer onda o intelektu nema ni govora. Neki autori predlažu da treba preći u mnogo "intelektualnije" područje, muziku.

Rješavanje netrivijalnih zadataka

Odmah na početku se postavlja pitanje šta su trivijalni, a šta netrivijalni zadaci. Uobičajena matematička definicija kaže da je **trivijalni zadatak** onaj kod koga je način rješavanja poznat. Pri tome nije važno da li rješenje postoji ili ne, odnosno da postavke dovode do apsurda. Iz definicije trivijalnog zadatka proizilazi definicija **netrivijalnog zadatka**: to je zadatak kod koga se mora pronaći način rješavanja. Ovdje se ne misli na klasu problema koji su rješavani, odnosno nisu rješavani, nego na onoga ko mora rješiti zadatak, a ne poznaje algoritam rješavanja.

PRIMJER*: *Množenje dva broja.

Za učenike starijih razreda množenje dva broja je trivijalni zadatak, pošto su učili tablicu množenja, a za prvoškolca ovo je netrivijalni zadatak, jer on ne poznaje način rješavanja zadatka.

Klasični primjer inteligencije dat je u anegdoti iz đačkog života *Karl Friedrich Gauss*-a. Kao prvoškolac je dobio "nerješiv" zadatak da sabere brojeve od jedan do sto. Gauss je vrlo brzo i elegantno došao do rješenja: uvidio je da je zbir prvog i zadnjeg broja 101, drugog i predzadnjeg takođe 101 i tako sve do zbira zadnjeg para brojeva 50 i 51. Budući da parova ima 50, proizvod 50 puta 101 daje rješenje 5050.

Ekstrapolacija

Ekstrapolacija, odnosno aproksimativno odlučivanje na osnovu niza faktora, je jedna od karakteristika koju je korisno objasniti i preko primjera. Neka je dan uzrok neke pojave i njegova posljedica: pritisak na prekidač izaziva paljenje svjetla, guranje bureta izaziva njegovo kotrljanje, pojava problema zahtjeva njegovo otklanjanje, itd. Kod trivijalnih problema rješenje je očigledno, što nije slučaj kod zdravstvenih tegoba relativno nedefinisanog oblika ili proizvodnje nekog dijela ili sklopa.

***PRIMJER*: Donošenje odluke o kupovini automobila.**

Može se napraviti tabela konkurentnih karakteristika dva ili više automobila različitih proizvođača u istoj klasi i sa približno jednakom cijenom. U tabeli će se naći: potrošnja, komfor, snaga, servis, veličina putnog i prtljažnog prostora, dizajn, itd. Može se napraviti bodna lista, gdje svaka karakteristika ima svoju težinsku vrijednost, čime se dobija sistem jednačina u kojima je upoređivanje karakteristika poznato kao proces identifikacije. Nakon identifikacije treba dobiti rješenje, ali rješenje, praktično nikada, nije jednoznačno i definitivno. Odluka je tada i sama niz odluka, odnosno proces koji nazivamo jednim imenom ekstrapolacija.

Učenje

Učenje je jedna od najvažnijih i najtežih karakteristika spoznaje, i samim tim, inteligencije uopšte. Pri tome se ne misli da je učenje memorisanje činjenica.To bi bilo isto kao kada bi se reklo da je vožnja automobila pritiskanje papučice gasa. Doduše, bez gasa automobil se ne može voziti, ali samo sa gasom to je još besmislenije.

Memorisanje činjenica jeste nužna i neophodna karakteristika učenja, ali daleko od toga da bude i dovoljna. **Aktiviranjem čula** počinje učenje, odnosno uspostavljanje kontakta sa vanjskim svijetom. Kod živih bića ovo je stalan, neprekidan proces, koji počinje sa rađanjem i prestaje sa smrti. **Sposobnost pamćenja** je slijedeća karakteristika procesa učenja. Samo prosto memorisanje podataka nije karakteristika inteligencije, jer ukoliko bi to bilo tačno onda i papir i crijep sa klinastim pismom i magnetni medij bi bili inteligentni.

1.3. POJAM I DEFINICIJE VJEŠTAČKE INTELIGENCIJE

Teorija i praksa vještačke inteligencije (VI) su još u svom pionirskom dobu i vrijeme pravih otkrića tek treba da dođe. Potrebno je bilo dugo vremena da se pojam vještačke inteligecije *[Artifical Intelligence (AI)]* probije i odomaći. Primjena metoda VI će, vjerovatno, igrati značajnu ulogu u organizacijama budućnosti, u prvom redu u cilju pripremanja odluka u procesu upravljanja.

Vještačka inteligencija je naučno područje, koje ima cilj da mašine, kao što je računar, imaju sposobnost inteligentnog ponašanja. Može se očekivati, da će organizacije koje ovladaju ovom oblasti, značajno povećati svoju produktivnost, odnosno obezbjediti prednost na tržištima budućnosti.

Definicije vještačke inteligencije

(1) **Vještačka inteligencija je nauka koja čini da mašine obavljaju stvari koje bi zahtjevale inteligenciju kada bi ih obavljao čovjek** *(M. Minsky, 1968.);*

(2) **Vještačka inteligencija je dio nauke o računarima usmjeren na stvaranje i proučavanje računarskih programa koji ispoljavaju svojstva koja se identifikuju kao inteligentna u ljudskom ponašanju: znanje, zaključivanje, učenje, rješavanje problema, razumjevanje jezika i dr.** *(A. Barr, 1983.);*

(3) **Vještačka inteligencija je disciplina kreiranja mašina koje podražavaju ljudsko ponašanje ili inteligenciju; mašine koje su senzitivne i misle** *(M. Carrico, J. Girard, J. Jones, 1989.);*

(4) **Veštačka inteligencija je disciplina koja izučava mehanizme inteligentnog ponašanja kroz analizu, razvoj i evaluaciju veštačkih tvorevina u koje se ugrađuju ti mehanizmi** *(V. Devedžić, 2002.).*

1.4. RAZVOJNI PUT VJEŠTAČKE INTELIGENCIJE

Značajniji događaji u istoriji VI prikazani su u tabeli 1.1.

TABELA 1.1.

PERIOD	KLJUČNI DOGAĐAJI
1930. - 1940. Korjeni	Formalna logika, kognitivna psihologija (*), ideja kompjuterizacije
1945. – 1954.	Razvijeni računari, administrativno ponašanje, kibernetika i samoorganizirajući sistemi, (Turing, A., "Computing Machinery and Intelligence", *Mind 49*, 1950.)
1955. - 1960 Počeci istraživanja VI.	Narastajući broj računara, jezik obrade informacija I (IPL-I), rješavanje opštih problema, psihologija obrade informacija

NASTAVAK TABELE 1.1.

Godine razvoja i preusmjeravanja 1961. - 1970. Traganje za rješavaocima opštih problema	Rješavanje ljudskih problema (Nowel, A., Simon, H., Am.Psych.Ass.) LISP (McCarthy, John, AI Lab., Stanford University, [1979-… (**)]) Heuristici ELIZA (ES, Weizenbaum, Joseph , MIT, [1960.- …]) Robotika Šahovski programi DENDRAL (ES, Stanford University [1965.-83.] – hemija)
Godine specijalizacije i uspjeha 1971. - 1980. Razvoj sistema zasnovanih na znanju.	MYCIN (ES, Stanford University, [1972.-1980.] – medicina) HEARSAY-II (ES, Erman, D.et all, Cornegie-Mellon, [1980.- …] - prirodni jezici) EMYCIN (ES, Essential MYCIN, Stanford University *&* Texas Instruments, [1974.-1979.]) – medicina) PROLOG (Colmerauer, A., Marselj, [1971-…] – programski jezik) Inženjerstvo znanja
Godine promjene 1981. - 1992.	PROSPECTOR (ES, Stanford Res. Inst, [1982- …] – geologija) Japanski projekti pete generacije Peta generacija računara Šesta generacija programskih jezika
1992. - … . Dalja istraživanja	Softverski agenti Otkrivanje znanja u BP (*Knowledge Discovery in Databases*, KDD) VI na Internetu

(*) *Kognitivna psihologija* - usmjerenje u psihologiji zasnovano na pretpostavci da funkcije mozga predstavljaju neki vid kompjuterizacije.

(**) Tri tačke (**...**) označavaju da se softverski proizvod, najvjerovatnije novija verzija, istog proizvođača pod istim imenom predstavlja na Internetu.

1.5. KARAKTERISTIKE VJEŠTAČKE INTELIGENCIJE

Računari predstavljaju jedinu tvorevinu ljudske ruke koji su u stanju da pretenduju na mogućnost sticanja elemenata i karakteristika inteligencije. Pri tome treba zanemariti sociološka, kulturološka, psihološka i druga razmatranja i sagledavati tehničke pretpostavke za ostvarenje navedenog cilja. Prema Evansovim definicijama karakteristike VI, koje bi računar morao da ima pa da, barem rudimentarno, bude inteligentan, su:

T 1.1. (bez zadnjeg reda) djelimično preuzeta iz knjige Ristić, Z. i dr.: "Ekspertni sistemi", 1993.
T 1.1. (zadnji red) preuzet iz knjige Devedžić, V.: "Inteligentni informacioni sistemi", 2000.

- prijem podataka,
- spremanje podataka,
- brzina obrade podataka,
- efikasnost računarskih programa,
- promjenljivost računarskih programa,
- mogućnost učenja,
- ekstrapolacija i rješavanje netrivijalnih zadataka.

Prijem podataka

Prijem podataka je prva karakteristika u ovom navođenju. Razvoj VI je uslovljen i čovjekovom težnjom da uspostavi maksimalno moguću i lagodnu komunikaciju sa mašinom. Idealno bi bilo da se računaru kaže glasom šta se želi i da on dalje posao obavlja sam. Nažalost, danas u najvećem broju slučajeva se mora na vrlo dugotrajan i mukotrpan način, preko tastature, reći kako se nešto radi, pa tek onda tražiti rezultate.

Spremanje podataka

Spremanje podataka je jedna od karakteristika VI, jer prijem podataka iz vanjskog svijeta i njihova obrada nemaju nikakvog smisla ako se ne mogu memorisati. Iz tog razloga računar mora imati odgovarajuće veliki prostor za njihovo spremanje. Prije spremanja podataka neophodna je odgovarajuća obrada, filtriranje i analiza, jer podatak se ne može uzimati "sirov".

Brzina obrade podataka

Brzina obrade podataka je karakteristika VI, bez obzira da li se radi o samom spremanju, klasifikaciji, sortiranju ili nekoj drugoj operaciji na podacima. Pri tome treba reći da računar mora imati odgovarajuću arhitekturu.

Efikasnost računarskih programa

Pod programiranjem se podrazumjeva niz upravljačkih naredbi potrebnih za rad sistema, pri čemu se misli na kodirane algoritme obrade podataka. Efikasnost računarskih programa podrazumjeva optimalno rješenje postavljenog problema, odnosno najbrži i najpouzdaniji način sa najmanjim utroškom prostora u memoriji računara.

Efikasnost programa bioloških sistema je vrhunska, jer je priroda imala dovoljno vremena za eksperimentisanje i odbacivanje neuspješnih rješenja. Pored toga, u prirodi se vodi bespoštedna borba za opstanak u neprijateljskoj okolini, gdje su kazne za neefikasno djelovanje veoma rigorozne.

Promjenljivost računarskih programa

Promjenljivost računarskih programa je neophodna, jer se programi moraju oblikovati prema zahtjevima sredine. Da bi program mogao izdržati visoke zahtjeve koje pretpostavljena inteligencija ima, on mora posjedovati minimalno dva svojstva:

- samoispravljanje grešaka,
- mogućnost promjene vlastite strukture.

Mogućnost učenja

Mogućnost učenja je veoma bitna karakteristika računara. Današnji računari rade tačno ono što je programom predviđeno. Bez obzira na vanjsku manifestaciju, rad računara se svodi, uglavnom, na korištenje memorije i dovoljno mnogo pitanja - skretnica tipa "AKO ... → TADA". Odstupanje od programiranog ponašanja znači grešku ili sistema ili programa. Međutim, ako se računaru omogući da uči, u bilo kojem obliku, tada se mogu očekivati optimalni programi, primjereni danom problemu.

Ekstrapolacija i rješavanje netrivijalnih zadataka

Ekstrapolacija i rješavanje netrivijalnih zadataka je danas u samom središtu svih istraživanja. Za ova istraživanja su angažovana ogromna materijalna sredstva i na njima rade vrhunski specijalistički timovi. Cilj je dobijanje inteligentnog računara, koji bi obrađivao probleme samostalno, metodom ekstrapolacije i time stvorio sve pretpostavke za rješavanje netrivijalnih zadataka.

1.6. PODJELA VJEŠTAČKE INTELIGENCIJE

Danas se primjenjuje čitav niz različitih klasifiacija VI. Najznačajnije oblasti VI su:

- obrada i razumjevanje prirodnih jezika,
- interpretacija i računarska obrada video oblika,

- robotika,
- sistemi zasnovani na znanju, u prvom redu sistemi za podršku odlučivanju *(Decision Support Systems)* i ekspertni sistemi,
- softverski agenti,
- otkrivanje znanja u BP (*Knowledge Discovery in Databases*, KDD), i
- VI na Internetu.

Obrada i razumjevanje prirodnih jezika

Prirodni jezički sistemi u ljudskom razvoju zauzimaju veoma visoko mjesto, možda i najviše. Osnova svakog sistema inteligencije je komunikacija, pri tome nije bitno da li se radi o izgovorenoj ili pisanoj riječi. Sjećanje ljudske vrste ne dopire do samih početaka nastanka jezika, mada se o tome može dobiti mutna predstava prateći dječije učenje jezika ili način komunikacije primitivnih plemena.

Iz navedenog, sasvim je razumljivo što je u središtu svih istraživanja VI pitanje komunikacije, posebno komunikacije koja je maksimalno bliska čovjeku, a to je prirodni živi jezik. Sva shvatanja pisaca naučne fantastike, kao i naša gledišta o odnosu čovjeka sa inteligentnim računarom, uvijek podrazumjevaju govornu komunikaciju.

Interpretacija i računarska obrada video oblika

Najrasprostranjeniji i najbrži oblik spoznaje svijeta, mada ne i najdublji, je putem vida. Opstanak u svakodnevnom životu tijesno je vezan za ovu karakteristiku većine živih bića. Paralelizam oku u VI predstavlja interpretacija i računarska obrada video oblika okoline, što ujedno predstavlja i kontakt sa tom istom okolinom. U okviru razvoja ove oblasti VI, izdvojila su se **slijedeća područja**:

- obrada slike različitim metodama uzoraka,
- prepoznavanje i definisanje vizuelnih oblika i njihova interpretacija,
- vezivanje identifikovanih oblika i njihovo poređenje sa bazama znanja,
- mogućnost učenja kroz uzimanje podataka iz vanjskog svijeta,
- odgovarajuća obrada vizuelnih signala i brzo reagovanje u realnom vremenu.

Robotika

Mjerenje stepena razvijenosti neke države se kroz istoriju stalno mijenjalo. Nekada je kao parametar uzimana proizvodnja čelika ili potrošnja energije. U posljednje vrijeme sve više se probijaju parametri koji prate informatičku

revoluciju, kao što su instalisani računarski sistemi, a prije svega broj praktično upotrebljenih industrijskih robota.

Robotika se bavi proučavanjem i razvojem mašina koje obavljaju mahaničke manipulacije. VI pomaže računarskom kontrolisanju pokreta, uz korištenje specijalnog zaključivanja. Pri tome je veoma teško postići glatke, povezane, skladne pokrete kakvi su prirodni.

Sistemi zasnovani na znanju

Sistemi zasnovani na znanju, gdje spadaju sistemi za podršku odlučivanju i ekspertni sistemi, predstavljaju jedan od oblika praktične primjene VI. Opšte karakteristike računarskih programa ovih sistema su:

- simboličko predstavljanje, korištenje heuristike, predstavljanje znanja,
- nestruktuirani problemi se rješavaju uz prisustvo logičkog nesklada između raspoloživih podataka,
- sposobnost "usavršavanja".

Heuristika je skup pravila domišljatog nagađanja, koja usmjeravaju i ograničavaju područja traganja za rješenjem.

Sistemi za podršku odlučivanju su IS, koji su slični i komplementarni standardnim IS, i imaju za cilj da podržavaju procese donošenja odluka. Oni predstavljaju simbiozu IS, primjene funkcionalnih znanja i tekućeg procesa analize odlučivanja.

Ekspertni sistemi su računarski programi kojima se emulira rješavanje problema na način kako to čine eksperti.

Počeci istorije razvoja ekspertnih sistema ukratko su prikazani u tabeli 1.2. Istraživanja u domenu ekspertnih sistema su započela 1960. godine. Nekoliko ES je razvijeno u periodu između 1965. i 1975. godine. Od 1975. godine broj projektovanih, razvijenih i implementiranih ekspertnih sistema naglo raste. Danas se pouzdano ne zna koliko je izgrađeno takvih specifičnih računarskih programa. Još uvijek se najveći broj ekspertnih sistema razvija i koristi u okvirima najvećih i najpoznatijih univerzitetskih ustanova tehnološki najrazvijenijih zemalja svijeta. Zbog svoje velike praktične primjene podliježu strogoj kontroli i nedostupni su široj javnosti.

TABELA 1.2.

SISTEM / DOMEN	DATUM	RAZVOJ/TRAJANJE	PODRUČJE
DENDRAL	1965.	(Stanford University, [1965.- 83.])	Zaključivanje o hemijskim strukturama
HEARSAY	1965.	(Erman,D.et all, Carnegie. – Mellon, [1965.-1980.])	Prirodno – jezička interpretacija za podskup jezika
MACSYMA	1965.	(Schelter, W., MIT, [1965.- 1991.])	Izvođenje kompleksne matematičke analize
AGE	1973.	(Stanford Res. Ins. [1976.-1982])	Sredstva za generisanje ekspertnih sistema
MYCIN	1972.	(Stanford University, [1972.-1980.])	Dijagnoza bolesti krvi
TEIRESIAS	1972.	(Stanford University, [1974.-1977.])	Sredstva za transformaciju znanja
PROSPECTOR	1972.	(Stanford Res. Ins., [1974.-1977.])	Sredstva za identifikaciju i otkrivanje nalazišta minerala
DPS5	1974.	(Carnegie – Mellon, [1974.- …])	Sredstvo za izgradnju ES
CADUCEUS	1975.	(University of Pittsburgh, [1975.- …])	Sredstvo za dijagnozu u internoj medicini
R1	1978.	(Carnegie - Mellon, [1978.- 1982.])	Konfigurator za DEC računarske mašine

U proteklih 25 godina razvijene su hiljade ES, za rješavanje različitih problema i za različite primjene. Područja primjene ES su mnogobrojna. Ovdje će biti navedena neka od njih:

- medicinske nauke
- Web dizajn
- humanističke nauke
- kompjuterske nauke
- proizvodnja
- vladine ustanove
- transport
- istraživanje i razvoj
- zaštita životne sredine
- finansijske usluge
- pravne nauke
- vojska
- marketing
- poljoprivreda
- naučna istraživanja
- projektovanje
- geologija i rudarstvo
- podrška odlučivanju
- trgovina
- inženjering, itd.

T 1.2. Djelimično preuzeto iz knjige Ristić, Z. i dr.: "Ekspertni sistemi", 1993 (izuzev zadnjeg reda).

Ekspertni sistemi podržavaju objektno orijentisani pristup programiranju, sa operativnim sistemima Windows, Mac, UNIX, Linux i dr., kao i interaktivni Web dizajn. Podaci za ES se mogu uzimati iz baza podataka (baza znanja), kao i drugih izvora.

PRIMJER:

Kompanija EXSYS sa svojim softverskim alatima (shell) omogućava razvoj ES u raznim područjima ljudske djelatnosti. Neki od uspješno razvijenih ES namjenjeni su:

- *za dijagnostiku civilnih i vojnih vazduhoplova i podršku pri opravci,*
- *praćenje imigracionih i naturalizacionih kretanja,*
- *vršenje vladinih usluga i procjena,*
- *odabir instrumenata za prognozu vremena,*
- *ekspertni sistemi za instalaciju računarskog hardvera i dr.*

1.7. PRIKAZ RADA EKSPERTNOG SISTEMA

Jedan od najstarijih poznatih ekspertnih sistema MYCIN razvijen je još 1972. godine na Univerzitetu Stanford u SAD. Ovaj ES razvijen je na kompjuteru PDP-11 i sadrži preko 500 pravila i relacija, a predviđen je kao moćan alat liječnika kod dijagnosticiranja infektivnih bolesti krvi. Na primjeru MYCIN-a se može dobro analizirati rad ES. Da bi se lakše pratio dijalog između korisnika, odnosno liječnika, i ES, odnosno MYCIN-a, koriste se slijedeće skraćenice:

- KOR: liječnik - korisnik,
- EKS: ekspertni sistem.

Dijalog između korisnika i ES

Nakon uključivanja računara i pozivanja ES, javlja se MYCIN i počinje dijalog.

EKS: POZDRAVLJA VAS MYCIN
Želite li instrukcije (DA ili NE)?

KOR: DA

EKS: Ekspertni sistem MYCIN je predviđen za konsultacije po pitanjima infektivnih oboljenja. Mislim, da Vi želite dobiti informaciju o liječenju nekog pacijenta. Zato ću Vam postaviti neka pitanja o pacijentu. Ako niste sasvim sigurni u Vaš odgovor, tada možete u zagradama dati koeficijent Vaše sigurnosti (broj od 1 do 10). Broj 10 odgovara apsolutnoj uvjerenosti.

Ako želite izmjeniti odgovor na neko od prethodnih pitanja, dajte naredbu MIJENJATI, iza koje mora slijediti broj pitanja na koje želite izmjeniti odgovor. Ne primjenjujte često tu naredbu, jer poslije nje sistem pristupa verifikaciji od samog početka (makar i ne zadavao korisniku nova pitanja), što sve produžuje vrijeme dobijanja rješenja.

Kao moguću reakciju na pitanje sistema Vi možete dati slijedeće informacije:

1. **NEIZ (NEIZVJESNO) - ako ne znate odgovor na pitanje koje Vam je dao sistem;**
2. **? - ako želite da Vam sistem da mogući odgovor;**
3. **?? - ako želite da Vam sistem da SVE moguće odgovore na zadano pitanje;**
4. **PR (PRAVILO) - ako želite da Vam sistem pokaže pravilo na osnovu koga je dao odgovor na postavljeno pitanje;**
5. **ZAŠTO - ako želite da Vam sistem detaljnije objasni pravila na osnovu kojih je došao do odgovora na tekuće pitanje;**
6. **KAKO-N - ako želite objašnjenje kako je sistem došao do cilja N nakon pitanja ZAŠTO;**
7. **U - I - ako želite prekid konsultacije, da bi sistemu zadali dodatna pitanja i saznanja.**

Ako ste u toku konsultacije zaboravili dopuštene komande ili djelovanje, odaberite komandu POMOĆ (*HELP*) i sistem će Vam odmah pomoći.

- Kraj instrukcije.

Primjer korištenja MYCIN-a

Nakon što je korisnik dobio ovakve i slične upute (koje govore da korisnik ni u kom slučaju ne mora da bude računarski specijalista), može se prići praktičnom korištenju MYCIN-a. U narednom tekstu dan je primjer tog korištenja.

EKS: 1. Ime i prezime pacijenta?
KOR: Savo Savić

EKS: 2. Pol?
KOR: Muški

EKS: 3. Starost (godine)?
KOR: 50

EKS: 4. Da li su dobijene kulture mikroorganizama iz inficiranog mjesta?
KOR: Da

EKS: 5. Kakva je to infekcija?

KOR: Primarna, bakterijama

EKS: 6. Molim datum prvih simptoma infekcije (INFEKCIJA-1) - (mjesec, dan, godina)
KOR: Maj, 10, 2003.

EKS: 7. Iz kojeg mjesta je dobijena kultura?
KOR: Krv

EKS: 8. Kada je dobijena ova kultura (mjesec, dan, godina)?
KOR: Maj, 14, 2003.

EKS: 9. Mikroorganizam identifikovan?
KOR: Ne

EKS: 10. Da li je mikroorganizam u obliku štapića, koke, itd?
KOR: Štapić

EKS: 11. Kako se ponaša mikroorganizam (gram pozitivno ili negativno)?
KOR: Gram negativno

EKS: 12. Organizam raste anaerobno?
KOR: Da

EKS: 13. Gdje se infekcija širi?
KOR: Želučano - probavni trakt.

Ovakav oblik dijaloga se nastavlja sve dok sistem ne prikupi dovoljno informacija o pacijentu, odnosno njegovoj bolesti. Dalji dijalog će biti preskočen i biće prikazan njegov završetak.

EKS: 45. Terapeutska preporuka se zasniva na tome da je infekcija zaista primarno bakteriološka i da je mikroorganizam:

1. PSEUDOMONAS - AERUGINOSA
2. KLEBSIELLA - PNEUMONIAE
3. E. COLLI
4. BACTEROIDES - FRAGILIS
5. ENTEROBACTER
6. PROTEUS - NON - MIRABILIS

Kako ocjenjujete stanje pacijenta (koristeći skalu od 0 do 4, gdje je veći broj lošije stanje)?
KOR: 3

EKS: 46. Da li je pacijent Savo Savić po obavljenoj pretrazi pokazao alergološku reakciju na bilo koji antibiotik?
KOR: Ne

EKS: 47. Kolika je težina pacijenta (kg)?
KOR: 70

EKS: 48. PREPORUKA 1. Moja prva preporuka je slijedeća:
Da bi riješili slučajeve <1, 2, 3, 5, 6>:
Propisati: GENTAMYCIN
Doza: 119 mg (1,7 mg/kg) svakih 8 sati intramuskularno ili intravenozno tokom 10 dana.
Komentar: Modifikovati dozu u slučaju nezadovoljavajućeg efekta.
Da bi riješili slučaj <4>:
Propisati: CLINDAMYCIN
Doza: 595 mg (8,5 mg/kg) svakih 6 sati intramuskularno ili intravenozno tokom 14 dana.

Komentar: Ako se pojavi dijareja ili neka druga želučano - probavna smetnja, provjeriti odmah kod pacijenta prisustvo pseudomembanskog kolita.

Nakon ovog MYCIN pita korisnika za daljnje, detaljnije, razmatranje i, eventualno, nove činjenice. Ovaj mali primjer pokazuje način komunikacije i oblik rada ES. Tokom dijaloga se prikupljaju činjenice, uzimaju detaljni i manje detaljni podaci i dolazi do traženog rješenja.

Nakon ovoga, odmah se postavlja pitanje koliko je jedan ES pouzdan, odnosno koliko mu se zaista može vjerovati. Vršena su brojna ispitivanja kvaliteta ES, prvo MYCIN-a a kasnije i drugih, i sva ispitivanja su dala više nego izuzetne rezultate.

2

U ovom poglavlju:

- ***Značenje izraza znanje***
- ***Vrste znanja***
- ***Iskazi***
- ***Zaključivanje***

ZNANJE I ZAKLJUČIVANJE

2.1. ZNAČENJE IZRAZA ZNANJE

Neki autori poistovjećuju znanje sa informacijom, a neki pokušavaju da značenje jednog od ta dva pojma odrede posredstvom onog drugog.

PRIMJER: ***Informacija je prirast znanja***.

Može se tvrditi da nema opravdanja za poistovjećivanje znanja i informacije**: nije svaka informacija znanje, niti je svako znanje informacija!**

Pojam informacije

Na osnovu mnogih teorija, da se nešto smatra informacijom, potrebno je da:

(1) Kazuje nešto što je prethodno bilo nepoznato primaocu;
(2) Govori o nečemu što je prethodno bilo više neizvjesno za primaoca;
(3) Utiče na količinu ili strukturu znanja primaoca;
(4) Bude upotrebljeno u primaočevom odlučivanju;
(5) Proizvodi zamišljene, razmatrane ili stvarno preduzete akcije primaoca;
(6) Primaocu smanjuje neizvjesnost;
(7) Primaocu pomaže da identifikuje kontekstualna značenja riječi u rečenici;
(8) Isključuje neka od alternativnih stanja stvari;
(9) Mijenja vjerovanja primaoca, naročito na distribucije vjerovatnoća sa primaočeve tačke gledišta.

Pojam znanja

Postoji više različitih tumačenja pojma znanja, pa je izraz znanje, kao i izraz informacija, višeznačan. Do 60-tih godina smatrano je da je: **znanje opravdano istinito vjerovanje**.

Neka osoba **O** zna **p** ako i samo ako:
(1) **O** vjeruje u **p**;
(2) **p** je istinito;
(3) **O** ima odgovarajuće svjedočanstvo za **p**.

(pod svjedočanstvom se podrazumjeva neki konačni skup razloga na osnovu kojih se prihvata **p**).

Navedena tri uslova smatrana su nužnim i dovoljnim da bi se **p** proglasilo znanjem. Međutim, uslijedile su tvrdnje da prvi navedeni uslov nije nužan, a drugi i treći uslov nisu dovoljni uslov znanja.

Osnovne odlike znanja

Navedeni nužni i dovoljni uslovi znanja ipak ukazuju na neke osnovne odlike znanja, kao što su:

- struktuiranost,
- koherentnost,
- relativna trajnost.

Osnovne odlike informacija su:

- djelimičnost,
- fragmentarnost,
- izrazita privremenost,
- kratkotrajnost.

Ovim razlikama informacije i znanja se može dodati: **informacija je proces, a znanje je stanje.**

2.1.1. Vrste znanja

Znanje se može razvrstati na više načina, i to:
- zdravorazumsko i naučno znanje,
- teorijsko i empirijsko,
- individualno i socijalno,
- eksplicitno i implicitno (neizrecivo) znanje,
- subjektivno i objektivno,
- deklarativno i proceduralno, i druga.

Zdravorazumsko znanje se stiče iskustvom i ono je:
- nedovoljno sistematizovano,
- nedovoljno kritično prema vlastitim tvrdnjama,
- ne njeguje razboritu sumnju u svoja tvrđenja,
- ne nalaže strogo navođenje adekvatnog i relevantnog svjedočenja u prilog svom tvrđenju,
- ne zahtjeva strogo dokazivanje i provjeravanje iskaza,
- trpeljivo prema logičkom neskladu između tvrđenja koja ulaze u njegov sastav.

Zdravorazumsko znanje je pretežno praktične prirode, oskudjeva u objašnjavalačkoj moći.

Naučno znanje se stiče pomoću naučnih metoda, pretežno je teorijske prirode te mu je svojstvena objašnjavalačka snaga. Naučno znanje se stiče, organizuje, dokazuje, provjerava i razvija zaključivanjem.

Implicitno (neizrecivo) znanje je nepodložno izražavanju posredstvom nekakvih jezičkih oblika. Riječ je o subjektivnom znanju koje je nedostupno ili teško dostupno drugim osobama, što ima nepoželjne posljedice za crpljenje znanja od eksperata metodama i tehnikama inženjeringa znanja.

Tri svijeta znanja

Mogu se uočiti tri svijeta znanja:
- fizički svijet: **svijet objektivne stvarnosti**,
- svijet naših svjesnih iskustava: **subjektivna znanja**,

- svijet logičkih sadržaja u knjigama, biblotekama, računarskim memorijama i sl.: **svijet objektivnih znanja**.

Čovjekovo znanje je refleksivno, čovjek ne samo da saznaje nego i saznaje da saznaje, pa otuda on može znati da zna i, što je isto važno, znati da ne zna. Takođe, može ne znati da zna i ne znati da ne zna.

2.1.2. Iskaz i sud

Izražavanje znanja

Naučno znanje se izražava rečenicama. Rečenicama se izražava ne samo neko naučno tvrđenje: naučna činjenica, hipoteza, naučni zakon, nego i svjedočanstvo koje se navodi za to tvrđenje. Osim toga, rečenicama se pored smisla može pripisati jedno veoma bitno svojstvo: **istinitost** ili **lažnost**.

Rečenica

Pod rečenicom se podrazumjeva takva kombinacija riječi (znakova) nekog (prirodnog ili vještačkog) jezika, izgrađena prema pravilima toga jezika, koja može da bude upotrebljena za tvrđenja, postavljanje pitanja, izražavanje sumnji, izdavanje naloga, ukoravanje, itd. i koja mora imati smisao.

***PRIMJERI* rečenica:**

(a) Da li ste provjerili ove podatke?
(b) Treba ponoviti ovo ispitivanje.
(c) Uran je radioaktivni hemijski element.

O prve dvije rečenice se ne može govoriti kao o istinitim, odnosno lažnim, bar ne onako kao o trećoj rečenici. Rečenice koje izražavaju pitanja, sumnju, savjete, zapovjesti, prekore, nadanja, bojazni, itd. ništa izričito ne tvrde, pa se ne postavlja pitanje njihove istinitosti, odnosno lažnosti. O trećoj rečenici se može misliti i govoriti kao o istinitoj.

Iskazi

Iskazi su rečenice o kojima se može misliti (i govoriti) kao o istinitim (odnosno lažnim).

Sud

Za označavanje smisla rečenice o kojoj se može misliti kao o istinitoj ili lažnoj, u logici se koristi izraz sud. Za razliku od rečenice, sud se ne sastoji od riječi, sud je njima samo izražen. Ista rečenica može izražavati različite sudove u različitim situacijama njene upotrebe.

Iskaz je jezički oblik izražavanja suda. Ako je sud istinit, onda je istinit i iskaz kojim je taj sud izražen.

2.1.3. Struktura iskaza i suda

Predikatski iskazi (sudovi)

Veliku klasu iskaza (sudova) čine iskazi kojima se nečemu ili nekome pripisuje (ili odriče) neko svojstvo, npr. "*Zemlja je okrugla*".

Ovakvi iskazi (sudovi) sadrže tri osnovna elementa:

(1) **Subjekat (S)** - dio iskaza koji ukazuje na predmet o kojem se misli;
(2) **Predikat (P)** - dio iskaza koji govori o nekom svojstvu koje se subjektu pripisuje ili odriče;
(3) **Veza** (je, nije, su, nisu) - koja izražava vezu između subjekta i predikata.

Struktura ovih iskaza (sudova) se može izraziti formulom:

S je (nije) P

gdje su S i P promjenljive i mogu se zamjeniti raznim objektima, odnosno svojstvima.

Relacioni iskazi

Relacioni iskazi tvrde ili poriču postojanje nekog odnosa, relacije, između objekata ili pojava. Struktura ovih sudova se može predstaviti binarnom relacijom:

R (a, b)

PRIMJERI: $X < Y$
Magelanovi oblaci su sataliti naše galaksije.

Cetinje je sjeverno od Budve.

Promjenljive su **a** i **b**, a **R** je relacija. **R** može izražavati količinski odnos, prostorni odnos, poslovni odnos, itd.

Za određivanje značenja izraza **relacija**, potrebno je poći od pojma **skupa**. Skup koji ima dva člana {a, b} se zove **par** (dvojka). Ovaj skup je istovjetan sa skupom {b, a}. Skup $R(a_1, a_2, \ldots, a_n)$ je **n-torka**. Ako se skup od dva člana "uredi", {a, b}, onda to nije više skup istovjetan sa skupom {b, a}. Sada je riječ o **"uređenim" parovima**.

PRIMJER: *12 > 7 nije isto što i 7 > 12.*

Relacija dužine n, n prirodni broj, je skup uređenih n-torki. To je podskup skupa svih mogućih uređenih n-torki.

Egzistencijalni iskazi

Egzistencijalni iskazi tvrde ili poriču postojanje nekog objekta, pojave, procesa, stanja, itd.

PRIMJER: *Ne postoji perpetum mobile.*

2.2. ZAKLJUČIVANJE

2.2.1. Struktura logičkog argumenta i zaključivanje

Naučno znanje se stiče, povezuje u cjelinu, dokazuje, provjerava i razvija zaključivanjem.

Logički argument je skup od najmanje dva iskaza sa određenim međusobnim odnosom i sastoji se od:

- iskaza koji se izriče,
- iskaza kojima su izraženi razlozi za dati iskaz (svjedočanstvo ili premise za taj iskaz).

Zaključivanje je uspostavljanje takvog odnosa među iskazima, posredstvom koga se na osnovu poznate (ili pretpostavljene) istinitosne vrijednosti nekog (ili nekih) iskaza procjenjuje istinitosna vrijednost nekog drugog iskaza.

Zaključivanje se može shvatiti i kao proces izvođenja jednog iskaza ili suda iz drugog (drugih). **Premise logičkog argumenta** su iskazi koji čine osnovu za izvođenje nekog iskaza. **Zaključak** je iskaz koji se izvodi iz drugih iskaza.

PRIMJER:

Premise: Ako se metal tare, onda se zagrijava.
Ako se metal zagrijava, onda se širi.

Zaključak: Ako se metal tare, onda se širi.

2.2.2. Ispravnost logičkog argumenta

Da bi se ocjenio neki logički argument (odnosno zaključivanje), neophodno je ispitati odnos između premise (svjedočanstava) i zaključka (iskaza koji se tvrdi i obrazlaže).

Ispravnost logičkog argumenta, odnosno zaključivanje, tiče se odnosa između premise i zaključka, a ne istinitosti premise ili zaključka.

Ispravnost ili neispravnost je svojstvo logičkog argumenta, odnosno zaključivanja, a istinitosna vrijednost je svojstvo iskaza koji sačinjavaju logički argument. Ispravnost ili neispravnost logikog argumenta se ne dokazuje istinitošću ili neistinošću zaključka, jer i u ispravnom i u neispravnom logičkom argumentu zaključak može da bude istinit (a takođe i lažan).

PRIMJER:

a) (I) Svi metali su hemijski elementi.
(I) Srebro je metal *(ISPRAVAN)*

(I) Srebro je hemijski element.

b) (I) Svi metali su hemijski elementi.
(I) Srebro je hemijski element. *(NEISPRAVAN)*

(I) Srebro je metal.

2.2.3. Vrste logičkih argumenata

Obzirom na zavisnost istinitosne vrijednosti zaključka od istinitosne vrijednosti premisa, razlikuju se dvije osnovne vrste logičkih argumenata:

(1) Logički argumenti u kojima zaključak iz premisa slijedi sa logičkom nužnošću - **deduktivni logički argumenti**;

(2) Logički argumenti u kojima zaključak iz premisa ne slijedi sa logičkom nužnošću, nego se iz njih izvodi sa nekom (većom ili manjom) vjerovatnoćom - **induktivni logički argumenti**.

2.2.3.1. Deduktivni logički argumenti

U deduktivnom logičkom argumentu, odnosno zaključivanju, zaključak iz premisa slijedi sa logičkom nužnošću (izvjesnošću). Suština je u tome da ako su premise istinite i logički argument ispravan, zaključak mora da bude istinit. Deduktivni logički argument se smatra ispravnim ako je odnos između premisa i zaključka takav da zaključak nužno mora da bude istinit ako su istinite premise.

Drugim riječima, **deduktivni logički argument je ispravan (valjan) ako i samo ako nije moguće da premise budu istinite, a zaključak lažan.**

Ako se u obzir uzme jedino istinitosna vrijednost premisa i zaključka, onda se dedukcija može shvatiti kao implikacija, te se može reći: **deduktivni logički argument je valjan, logički besprijekoran, ako premise implikuju zaključak**.

Sasvim je jasno da pri ispravnom, valjanom zaključivanju, iz istinitih premisa slijedi istinit zaključak, ali može biti da se sa ispravnim zaključivanjem iz lažnih premisa može izvesti ne samo lažan nego i istinit zaključak.

Odnosi između istinitosne vrijednosti iskaza

Sažet pregled odnosa između istinitosne vrijednosti iskaza, koji sačinjavaju logički argument, i ispravnosti deduktivnog logičkog argumenta (zaključivanja), prikazan je u tabeli 2.1:

- iz istinitih premisa se ne dobija uvijek istinit zaključak, nego samo onda kada je logički argument ispravan (1. red),
- bilo koji logički argument istinitih premisa i lažnog zaključka je neispravan (2. red),
- istinit zaključak se ne dobija jedino ispravnim zaključivanjem iz istinitih premisa (1. i 3. red),
- lažne premise i neispravno zaključivanje ne vode uvijek lažnom zaključku (3. i 4. red).

TABELA 2.1.

Istinitosna vrijednost iskaza u logočkom argumentu		**Deduktivni logički argument može da bude**	
PREMISE	**ZAKLJUČAK**	**ISPRAVAN**	**NEISPRAVAN**
istinite	**istinit**	**DA**	**DA**
istinite	**lažan**	**NE**	**DA**
lažne	**istinit**	**DA**	**DA**
lažne	**lažan**	**DA**	**DA**

Opšta pravila dedukcije

Da bi se obezbjedili uslovi da iz istinitih premisa uvijek slijedi istinit zaključak, uvode se opšta pravila dedukcije, koja imaju slijedeći opšti oblik:

$$\mathbf{A_1 \wedge A_2 \wedge A_3 \wedge ... \wedge A_n \supset B},$$

gdje su A_1, A_2, ... , A_n premise dedukovanog logičkog argumenta (koje sačinjavaju antecedente navedene implikacije), a B zaključak argumenta (konsekvent u navedenoj implikaciji).

U cilju formulisanja opštih pravila dedukcije moguće je razdvojiti sadržaj od forme (strukture) iskaza (koji sačinjavaju logički argument), kao i sadržaj od forme (strukture) logičkog argumenta. Na primjer, forma logičkog argumenta navedenog u primjeru sa zagrijavanjem metala može se predstaviti:

p ⊃ q		**Ako p, onda q**
q ⊃ r	**ili**	**Ako q, onda r**
p ⊃ r		**Ako p, onda r**

U ovom poglavlju:

- ***Automatsko zaključivanje***
- ***Cilj automatskog dokazivanja teorema***
- ***Formalizacija dokazivanja teorema***

3

AUTOMATSKO DOKAZIVANJE TEOREMA

3.1. AUTOMATSKO ZAKLJUČIVANJE

Automatsko dokazivanje teorema (ADT) se zasniva na metodama automatskog zaključivanja. Deduktivnim sistemima pripada centralno mjesto u oblasti automatskog zaključivanja i to:

- metodi rezolucije, za potpuno automatsko dokazivanje teorema, i
- interaktivno dokazivanje teorema na osnovi prirodnog izvođenja.

Moguće primjene ovih sistema su u domenu:

- upitnih dijaloških sistema,
- situacionog upravljanja i odlučivanja, i
- domenu automatskog kreiranja kombinatornih rasporeda.

Posebnu oblast primjene rezolucijskog dokazivanja teorema predstavlja jezik logičkog programiranja PROLOG.

Ključni problemi razvoja VI i automatskog zaključivanja su:

- mogućnost formalizacije neformalno postavljenih zadataka, i
- nalaženje dovoljno efikasnih strategija i procedura pretraživanja u mnoštvu varijanti.

Na zaključivanju se temelje sposobnost učenja, otkrivanja primjera i kontraprimjera, uočavanje analogija, prepoznavanje objekata i drugo. Posebno mjesto u tome pripada logičkom zaključivanju, tj. izvođenju posljedica iz danog skupa premisa.

Materijal izložen u u ovom poglavlju se zasniva na knjizi Hotomski, P., "Sistemi veštačke inteligencije", 1995. (detaljnije informacije o ADT čitalac može naći u navedenoj knjizi).

3.2. CILJ AUTOMATSKOG DOKAZIVANJA TEOREMA

Cilj ADT je projektovanje i implementacija računarskih programa koji dokazuju, ili pomažu pri dokazivanju, teoreme. Prema učešću korisnika u procesu dokazivanja na računaru, programi za ADT se dijele na:

- nezavisne od učešća korisnika ili čisto automatske, gdje program samostalno dokazuje teoremu kada i ako uspije, i
- na interaktivne dokazivače, gdje programi pronalaze dijelove dokaza, a zatim, u interaktivnom izvođenju dokaza, korisnik te dijelove kombinuje sa sopstvenom intuicijom, i eventualno, preusmjerava program u cilju kompletiranja dokaza.

ADT je našlo primjene (osim matematike) u oblastima kao što su: provjera korektnosti programa, generisanje programa, upitni jezici za relacione BP, projektovanje elektronskih kola, izgradnja ES i sistema zasnovanih na znanju. Kao jezik, tj. formalna reprezentacija u kojoj se dokazuju teoreme, koristi se iskazni račun, predikatski račun prvog reda (najčešće se primjenjuje), kao i logike višeg reda.

Od tehnika ADT najviše se izučavaju rezolucijska pravilima izvođenja. Primjena klasične rezolucije, međutim, predpostavlja da se formula iz predikatskog računa prvog reda prevede na oblik sastavaka, čime se znatno gubi na prirodnom obliku formule koji je blizak korisniku.

Dosadašnji rezultati u oblasti ADT ni izdaleka još nisu postigli cilj, jer se pokazalo da je oblast izuzetno kompleksna. U posljednje vrijeme, ipak, se pojavilo nekoliko programa, koji su doprinijeli dokazivanju novih teorema srednje težine.

3.3. FORMALIZACIJA DOKAZIVANJA TEOREMA

Logički sistem, u okviru koga se ostvaruje formalizacija dokazivanja teorema, predstavljaju formalne teorije. Formalne teorije se grade sa ciljem svođenja na minimum neodređenosti i nepreciznosti koje se javljaju uslijed upotrebe prirodnog jezika, intuicije i drugih subjektivnih kriterijuma, čime se omogućava objektivizacija deduktivnih metoda. Ta objektivizacija se sastoji u tome što se o ispravnosti neke definicije ili nekog dokaza odlučuje samo na osnovu njihove strukture, odnosno forme, a ne na osnovu sadržaja koji im se može pridružiti.

Bitna razlika između formalnih i sadržajnih matematičkih teorija je u postojanju formalnih pravila za prepoznavanje objekata teorije. Takav sistem formalnih pravila predstavlja tzv. **sintaksu formalne teorije**. Formalna teorija τ je određena kada su zadani slijedeći skupovi:

1. $S(\tau)$ - najviše prebrojiv skup osnovnih simbola (azbuka teorije);
2. $F(\tau)$ - skup formula kao podskup skupa svih riječi sa osnovnim simbolima teorije τ. Uz to je dan i efektivan postupak za odlučivanje da li je neka riječ formula ili nije. To se postiže definicijom koja iz skupa svih riječi dane azbuke izdvaja one riječi koje se smatraju formulama teorije;
3. $A(\tau)$ - skup aksioma kao podskupa skupa formula. Ako je dan efektivan postupak za odlučivanje da li je neka formula aksioma ili nije, onda je to aksiomatska teorija;
4. $\Re(\tau)$ - konačan skup pravila izvođenja. Svako pravilo izvođenja je izvjesna relacija u skupu formula teorije τ. Ako su formule $A_1, A_2, \ldots, A_{n-1}, A_n$ u relaciji α, onda se kaže da je A_n direktna posljedica redom danih formula $A_1, A_2, \ldots, A_{n-1}$ po pravilu izvođenja α i piše se:

$$\frac{A_1, A_2, \ldots, A_{n-1}}{A_n}$$

Ovi skupovi potpuno određuju formalnu teoriju, pa se teorija τ može definisati kao uređena četvorka: $(S(\tau), F(\tau), A(\tau), \Re(\tau))$.

U formalnim teorijama razlikuju se dva tipa izvođenja:

- izvođenje iz aksioma, i
- izvođenje iz hipoteza.

Definicija 1. Konačni niz formula

$$B_1, B_2, \ldots, B_m$$

formalne teorije τ u teoriji se zove **izvođenje** (dedukcija, dokaz), ako svaka formula B_i $(1 \le i \le m)$ tog niza ispunjava uslov:

1. B_i je aksioma, ili
2. B_i je direktna posljedica nekih prethodnih formula iz niza po izvjesnom pravilu izvođenja u τ.

Definicija 2. Formulu B_m formalne teorije τ u teoriji se zove **teorema**, sa oznakom:

$$\vdash_{\tau} B_m \qquad \text{ili} \qquad \vdash B_m,$$

ako postoji bar jedan niz $B_1, B_2, \ldots, B_m$, čije je izvođenje u teoriji τ. Kaže se da je taj niz izvođenje teoreme B_m.

Definicija 3. Neka je H neki skup formula neke formalne teorije τ i neka je A određena formula iste teorije. Kaže se: formula A je posljedica skupa formula H,

ako postoji konačan niz formula B_1, B_n, ... , B_m (B_m je jednako A), čija svaka formula ispunjava uslov:

1. B_1 je aksioma, ili
2. B_1 je iz skupa H, ili
3. B_1 je direktna posljedica nekih prethodnih formula niza po izvjesnom pravilu izvođenja teorije τ.

Može se pisati:

$$H \underset{\tau}{\vdash} A \qquad \text{ili} \qquad H \vdash A;$$

gdje se elementi skupa H zovu **hipoteze** (premise, pretpostavke). Niz B_1, B_2, ... , B_m se zove izvođenje formule A iz skupa hipoteza H.

Na osnovu navedenih definicija, za dokaz da je formula A teorema, ili posljedica skupa formula H, dovoljno je odrediti bar jedno izvođenje B_1, ... , B_m, gde je B_m formula A. Određivanje takvog niza formula za zadanu teoremu A, često zahtjeva jaku intuiciju u pogledu odabiranja njegovih elemenata. Međutim, konstrukcija traženog niza može se izvršiti, bar u principu, mehaničkim putem bez oslanjanja na intuiciju, što je objašnjeno u narednom tekstu.

Metoda potpunog pretraživanja (Graf dokaza teorema)

Neka $R(S) \overset{\text{def}}{=} S \cup P(S)$, gde je S neki skup formula teorije τ, a P(S) skup svih direktnih posljedica formula iz S po pravilima izvođenja teorije τ, i neka je za skup aksioma A teorije τ:

$$R_0(A) \overset{\text{def}}{=} A$$
$$R_{n+1}(A) \overset{\text{def}}{=} R(R_n(A)) \text{ , za } n \geq 0.$$

Obzirom da je izvođenje teoreme A konačan niz, za neko konačno $k \geq 0$ biće $A \in R_k(A)$. Metoda potpunog pregleda se sastoji u uzastopnom generisanju skupova $R_i(A)$, i = 1, 2, ... , k. Ako je skup *A* konačan, ovim postupkom se može odrediti minimalan dokaz, tj. izvođenje sa minimalnim brojem formula ili pravila izvođenja. Preciznije rečeno, dokaz se vrši pomoću pojmova **teorije grafova**.

Postupak je slijedeći. Prvo se definiše nivo formule C u danom izvođenju:

(1) Ako $C \in A$, onda C ima nulti nivo,
(2) Ako je S neki skup formula koje pripadaju danom izvođenju i formula $D \in S$ ima najviši nivo u S, označen sa j ($j \geq 0$), a formula C, koja ne pripada skupu S, se izvodi iz D i možda još nekih formula skupa S po pravilu izvođenja ϕ, onda C ima nivo j+1.

Kako formula C može imati više izvođenja, formula C može imati nekoliko različitih nivoa (po jedan u svakom izvođenju). Izvođenjima koja su određena trojkom (A, $\Re$, Φ), gde je A-skup aksioma, $\Re$-skup pravila izvođenja, Φ-skup terminalnih formula, pridružuje se orijentisan graf čiji su čvorovi formule iz A^* ($A^* = \cup R_i(A)$). Pri tome, formuli C koja ima m različitih izvođenja odgovara na grafu m različitih čvorova, a svakom izvođenju odgovara po jedan podgraf takvog grafa.

Graf (G,s) se interpretira u domenu dokazivanja teorema na slijedeći način: Preslikavanje $f:G\rightarrow A^*$ svakom čvoru $n\in G$ pridružuje formulu $f(n)\in A^*$; skup čvorova s(G'), gde $G'\subseteq G$, preslikava se u skup formula $R(\{f(n) \mid n\in G'\})$ određen primjenom svih pravila izvođenja iz skupa $\Re$ na formule skupa $\{f(n) \mid n\in G'\}$, tj. s(G') je skup sljedbenika. Pri takvoj interpretaciji grafa (G,s) izvođenje formule f(m) sadrži sve formule f(n) za koje je $m\geq n$.

Sada se zadatak dokazivanja teorema može predstaviti kao uređena četvorka (G,s,F,g), gde je $F\subseteq G$ skup terminalnih čvorova, a $g:G\rightarrow C$ (C je skup realnih brojeva) funkcija ocjene kojom se izražava složenost izvođenja. Po metodi potpunog pretraživanja generišu se svi čvorovi na svakom nivou, pa se za $n\in G_i$ funkcija g(n) obično izražava brojem i, tj. nivoom čvora n. Jasno je da zbog brojnosti skupova G algoritam potpunog pretraživanja daje ograničene, gotovo neznatne, mogućnosti za praktičnu realizaciju.

Meta-teorija i interpretacije formalne teorije

Definicija 4. Ako za ma koju formulu teorije τ postoji način da se odluči da li je teorema te teorije ili nije, kaže se da je **teorija τ odlučiva**.

Ispitivanje odlučivosti formalne teorije pripada meta-teoriji te teorije.

Neodlučivost i ograničenja

Predikatski račun se može izgraditi kako na bazi sintaksnih, tako i na bazi semantičkih koncepcija. **Semantička koncepcija** se oslanja na sadržajnu istinitost ili neistinitost interpretiranih formula na nekom domenu interpretacije, dok se **sintaksna koncepcija** izgrađuje kao formalna teorija zasnovana na aksiomama i pravilima izvođenja.

Ne postoji opšta procedura za utvrđivanje da li je neka formula valjana ili nije. Međutim, za formule koje su valjane postoji mehanička procedura pomoću koje se u konačnom broju koraka potvrđuje da je formula valjana. Postojanje takve procedure daje osnovu da se predikatski račun smatra **poluodlučivim**.

4

U ovom poglavlju:

- ***Pojam ekspertnih sistema***
- ***Ekspertni sistemi bazirani na pravilima***
- ***Arhitektura ekspertnih sistema***

ARHITEKTURA EKSPERTNIH SISTEMA

4.1. POJAM EKSPERTNIH SISTEMA

U praksi se često javlja potreba za nizom specifičnih znanja danih u cjelini, brzo sigurno i povezano. Drugim riječima, želi se da u problematičnim situacijama i kod donošenja složenih odluka pomogne dobar stručnjak, vrhunski specijalista ili, kako se drugačije kaže, ekspert. Pomoć eksperta je dobro došla u složenim situacijama bilo koje oblasti ljudskog rada: medicini, pravu, građevinarstvu, industriji, marketingu, itd.

Osnovna svojstva eksperta

Osnovna svojstva eksperta su da:

- primjeni, na optimalni način, svoja znanja u rješavanju problema. Pri tome se podrazumjeva uzimanje u obzir činjenica i predviđanje relevantnih posljedica;
- objasni i obrazloži svoje odluke i prijedloge;
- komunicira sa drugim ekspertima i proširuje svoja znanja, prestruktuira i reorganizuje shvatanja i znanja;
- formira i napušta određene zaključke, što dokazuje da je pronikao u suštinu određenih pojava i našao nove zakonitosti koje među njima vladaju;
- određuje najbrži način dolaska do rješenja i njegove praktične primjene;
- u specifičnim situacijama intuitivno (heuristički), na osnovu svih dosadašnjih iskustava i događaja ocjeni gdje se nalazi rješenje problema.

Imati pored sebe eksperta nije ni najmanje jednostavno, eksperata nema previše, nisu na raspolaganju u svakom trenutku i nisu ni jeftini. Osim toga ni jedan ekspert ne može da posjeduje sva znanja.

Ekspertni sistemi

Današnji stepen razvoja moderne informatičke nauke sve više omogućava da se stalno može raspolagati ekspertnim uslugama. Pri tome se misli na ekspertne sisteme (ES).

Pod ES se podrazumjeva takva vrsta programske podrške ili softvera na računaru, koja u većoj ili manjoj mjeri zamjenjuje čovjeka - eksperta. ES je u stanju da, na osnovu unesenih podataka i ugrađenih logičkih algoritama (pravila zaključivanja) i tako nastale baze znanja, efikasno pomogne korisniku u rješavanju specifične problematike.

Dodatne definicije ES

U literaturi se može naći veći broj sličnih definicija pojma ES. Tako, jedna definicija opisuje ES kao:

"Računarski sistem koji uključuje organizovano znanje, koje se tiče nekog specifičnog područja ljudske ekspertize (medicinska dijagnostika, identifikacija hemijskih jedinjenja, finansijsko planiranje, geološke prospekcije, itd.), u dovoljnom stepenu da može da vrši ulogu iskusnog i ekonomski racionalnog konsultanta u tom području".

Za ES se može reći da predstavljaju: "Program opšte namjene za rješavanje problema, koji imitira ljudsku inteligenciju" ili "Intelektualnu podršku visokog nivoa, koja služi isto kao i ljudski ekspert".

U slijedećoj definiciji, osim cilja, se objašnjava i struktura ES:

"Ekspertni sistemi koriste formalne načine predstavljanja znanja koje čovjek – ekspert posjeduje i metode logičkog zaključivanja, da putem odgovarajućih računarskih programa obezbjede ekspertni savjet ili mišljenje o problemu za koji je korisnik zainteresovan".

Područje primjene ES

Ekspertni sistemi imaju za cilj da obezbjede odgovor na probleme koji zahtjevaju rasuđivanje, prepoznavanje i poređenje oblika, akviziciju novih koncepata, zaključivanje, ukratko, oni daju odgovor na pitanja koja zahtjevaju inteligenciju. ES se mogu efikasno primjenjivati u područjima gdje se mišljenje o problemu svodi na logičko rasuđivanje, a ne na izračunavanje, i gdje svaki korak u rješavanju problema ima veći broj alternativnih mogućnosti.

Tipovi znanja ES

Ključni faktor za dobre performanse ES je kvalitet znanja koje je u njega ugrađeno. Znanje se čuva u bazi znanja ES i generalno se razlikuju dva tipa znanja:

- prvi tip znanja je ono znanje koje se zove **činjenicama danog domena**, odnosno znanje koje je široko poznato i nalazi se napisano u udžbenicima, časopisima i slično;
- drugi tip znanja je **heurističko znanje**, ono znanje koje čovjek - ekspert gradi na osnovu iskustva i koje kombinovano sa prvim tipom znanja čini čovjeka ekspertom.

Osim znanja, ES zahtjeva i **postupak zaključivanja - metod rasuđivanja**, korišten da napravi spregu između znanja koje se čuva u računaru i problema koji postavlja korisnik. On, takođe, zahtjeva način za **predstavljanje znanja** u računaru, znanja koje ES treba da posjeduje, i to, prije svega, u obliku logičkih struktura sa kojima računar može lako da manipuliše, kao i skup odgovarajućih struktura podataka.

Pitanja vezana za razvoj ES

Kod razvoja ES se javlja niz veoma krupnih pitanja, na koje treba dati odgovor.

Prvi problem, koji se susreće kod ES, je način **predstavljanja znanja**. Kako predstaviti znanje iz danog domena u obliku pogodnih struktura podataka, tako da se efikasno može iskoristiti u rješavanju problema?

Drugo, postavlja se pitanje kako koristiti znanje, kako **projektovati mehanizam zaključivanja** da bi se znanje efikasno koristilo u rješavanju problema?

Treće, postavlja se pitanje **akvizicije znanja**, to jest, kako izvući znanje iz glava eksperata i staviti ga u računar? Da li je moguće automatizovati korak akvizicije znanja i obezbjediti neposrednu komunikaciju eksperta i računara i nesmetan prenos znanja od eksperta ka računaru? U ovom trenutku, akvizicija znanja predstavlja ključno pitanje u razvoju metoda vještačke inteligencije.

Podjela ES prema vrsti korisnika

Ekspertni sistemi se razlikuju prema vrsti korisnika. Neki ES, kao što su sistemi medicinske dijagnostike, uključuju **znanje grupe eksperata u cilju korištenja od strane jednog eksperta iz iste grupe**. Drugim riječima, ljekari kreiraju sistem za ljekare.

Neki ES prenose **znanje jedne grupe eksperata grupi ili pojedincu koji to nisu**. U ovu grupu spadaju sistemi finansijskog planiranja. Upotreba ove grupe ES se danas smatra najkontraverznijom.

Načini korištenja ES

Postoje tri osnovna načina korištenja ES:

- prvi način, gdje korisnik traži odgovor na zadani problem,
- drugi način, gdje je korisnik instruktor koji dodaje znanje u postojeći ES,
- treći način, gdje je korisnik učenik koji uči od ES, na taj način povećavajući svoje znanje.

Pri tome se ES razmatra kao dio vještačke inteligencije i koristi sve tehnike primjenjene u tom području nauke.

Optimalnost baze znanja

Treba naglasiti da ES nije prosta BP ili neka vrsta automatizovanog priručnika. Osim primjene niza podataka i logičkih pravila (na primjer, poznati tip zaključaka "AKO ... $\rightarrow$ TADA"), ES takođe koriste:

- dostignuća iz područja tzv. dijalognih i prirodnih jezičkih sistema,
- dostignuća računarske animacije i robotike,
- razne načine interpretacije problema i donošenja odluka, itd.

Stvorena baza znanja ES mora postići optimum između niza potpuno kontradiktornih zahtjeva da bi, barem minimalno, zadovoljila korisnika. S jedne strane broj podataka, činjenica i logičkih odluka mora biti što je moguće veći, a nasuprot tome vrijeme dobijanja određenog rješenja ili prijedloga mora biti što je moguće manje. Pri tome je potrebno odabrati najvjerovatnije rješenje, ali predložiti i moguće alternative.

Sposobnost prihvatanja novih znanja

Treba znati da ES nije predviđen da daje konačna i neopoziva rješenja, već samo da pomaže u njihovom nalaženju. Takođe se podrazumjeva da ES mora biti sposoban da "**uči**" **i** "**prihvata**" **nova znanja**, shodno sa razvojem područja u kome je "specijalista". Ponekad ne postoji mogućnost davanja konkretnog rješenja, pa je potrebno koristiti **statistiku** i tzv. **heuristiku**, odnosno **intuitivno znanje**.

Pri svemu ovome treba imati u vidu i vrlo težak zahtjev što jednostavnijeg, bržeg i lakšeg odnosa čovjek - računar, sa maksimalnim izbjegavanjem nedoumica ili nejasnih zaključaka. ES, po potrebi, mora da "**pita**" korisnika za dodatne podatke. Ukratko, ES mora posjedovati ako ne sve karakteristike i znanja eksperta, ono barem dobar dio njih.

4.2. EKSPERTNI SISTEMI BAZIRANI NA PRAVILIMA

4.2.1. Mehanizam logičkog zaključivanja

U narednom izlaganju biće razmotren način funkcionisanja mehanizma za zaključivanje, koji obavlja proces logičkog rasuđivanja nad **bazom znanja**, predstavljenom u obliku produkcionih pravila, i **bazom činjenica** koja opisuje stanje sistema.

Inicijalizacija mehanizma za zaključivanje

Da bi mehanizam za zaključivanje mogao da započne postupak logičkog rasuđivanja, potrebno je prvo postaviti cilj rasuđivanja i inicijalizirati sve činjenice potrebne za testiranje uslova, odnosno izvršavanje odgovarajućih akcija. Inicijalizacija se vrši uz pomoć vrijednosti, koje se nalaze u bazi činjenica, ili se

može raditi **interaktivno**, s tim što tada mehanizam zaključivanja postavlja pitanja korisniku tokom postupka korištenja ES. Za efikasnu realizaciju dijaloga sa korisnikom, potrebno je da mehanizam zaključivanja ponudi korisniku i skup legalnih odgovora, koji su dozvoljeni za inicijalizaciju pojedinih činjenica, i na taj način vodi korisnika u pravcu da daje odgovore koje sistem razumije.

Optimizacija meta pravilima

Mehanizam zaključivanja treba, takođe, da sadrži i takozvana meta znanja, koja ustvari predstavljaju pravila, meta pravila, koja upravljaju postupkom logičkog rasuđivanja u smislu njegove optimizacije. Ova meta pravila se, prije svega, odnose na grupisanje, odnosno utvrđivanje pravila i način njihovog pretraživanja, da bi se obezbjedilo minimalno moguće vrijeme pretraživanja pravila.

Pristupi rasuđivanju

Obzirom na postavljeni cilj logičkog rasuđivanja, odnosno cilj konsultacije koja se očekuje od ES, moguća su dva opšta pristupa:

- direktno rasuđivanje,
- inverzno rasuđivanje.

Direktno i inverzno rasuđivanje je detaljnije opisano u poglavlju 14, koje nosi naslov “Izgradnja ekspertnih sistema”.

4.2.2. Razmatranje neizvjesnosti u znanju

Djelovanje u uslovima neizvjesnosti

U životu eksperti često djeluju u uslovima neizvjesnosti. Kao prvo, korisnik ekspertize dolazi sa ulaznim podacima za koje baš nije siguran da su potpuno tačni. Takođe, sam ekspert ponekad ne može biti potpuno siguran u znanja koja prikuplja iz okoline, a koja treba da mu pomognu u rješavanju danog problema. Nekada, postoji neizvjesnost i o stepenu relevantnosti dane ekspertize za neki problem.

Svi ovi izvori neizvjesnosti se moraju uzeti u obzir tokom postupka rasuđivanja i konačan savjet, koji ekspert daje, mora se kvalifikovati određenim stepenom izvjesnosti.

Faktor izvjesnosti

Standardni pristup razmatranju navedenih izvora neizvjesnosti je putem tzv. **faktora izvjesnosti**, koji uzimaju numeričke vrijednosti iz neke pogodno odabrane skale vrijednosti, kao na primjer: od 0 do 100, od 0 do 10 ili od -1 do 1. Ovi faktori predstavljaju mjeru sa kojom mehanizam zaključivanja određuje stepen izvjesnosti ili povjerenja u vrijednosti koje se tiču uslova i zaključaka pravila odlučivanja. Faktor izvjesnosti 0 predstavlja potpuno odsustvo izvjesnosti, odnosno saopštenje da je vrijednost neke činjenice u bazi činjenica potpuno nepoznata. Vrijednost 100 govori da je činjenica poznata sa apsolutnom izvjesnošću.

PRIMJER: *Može se razmotriti primjer da u sistemu pravila za određivanje količine prodaje postoji i pravilo:*

```
IF PRIV_RAST > 0.04 AND NEZAPOSLENOST < 0.075
THEN EK_SITUACIJA = "dobra"
```

KOMENTAR: Ekonomska prognoza je dobra, ukoliko je stopa privrednog rasta veća od 4% i stopa nezaposlenosti manja od 7,5%.

UVOĐENJE JEDNOG FAKTORA IZVJESNOSTI PREMISA

Činjenice `PRIV_RAST` i `NEZAPOSLENOST` ne moraju da budu poznate sa apsolutnom izvjesnošću. Može se pretpostaviti da je činjenica `PRIV_RAST` poznata sa izvjesnošću 80. Saopštenje da je neka činjenica poznata u okviru nekog faktora izvjesnosti se piše:

PRIV_RAST CF 80,

gdje CF *(Certainty Factor)* označava faktor izvjesnosti.

U odnosu na gornje pravilo, može se pretpostaviti da je `NEZAPOSLENOST` poznata sa apsolutnom izvjesnošću, odnosno CF jednako 100, što se po pravilu ne piše, već odsustvo CF implicira apsolutnu izvjesnost.

U ovom slučaju uz zaključak `EK_SITUACIJA = "dobra"` se može pridružiti izvjesnost 80.

UVOĐENJE DVA FAKTORA IZVJESNOSTI PREMISA

Ukoliko je i druga činjenica NEZAPOSLENOST poznata sa izvjesnošću manjom od 100, na primjer 90, tada se može postaviti pitanje kako kombinovati ove dvije izvjesnosti koje se primjenjuju u premisi, odnosno kako izračunati izvjesnost zaključka.

Metoda minimuma

Može se koristiti nekoliko različitih pristupa, a standardni pristup predstavlja uzimanje minimalne vrijednosti od dvije premise.

Razmatran je uslov iz navedenog primjera, koji se formira logičkom konjunkcijom, i ovo razmatranje važi samo za ovaj slučaj. Logika uzimanja minimalne vrijednosti, u danom primjeru 80, je bazirana na rasuđivanju da je "jačina nekog lanca određena jačinom njegove najslabije karike". Ovo je tzv. metoda minimuma, koji se izračunava kao:

MIN(CF1, CF2).

Metoda produkta

Moguće je pretpostaviti da se izvjesnost zaključka dobija kao proizvod dvije izvjesnosti u premisi:

CF1 * CF2/100

to jest, 80 * 90/100 = 72, što se naziva metodom produkta.

Metoda sredine

Moguće je koristiti i tzv. metodu sredine, koja se formira kao:

(MIN(CF1, CF2) + (CF1 * CF2)/100)/2.

Ova vrijednost predstavlja kompromisnu vrijednost, koja se nalazi između vrijednosti dobijene metodom minimuma, koja praktično daje maksimalnu vrijednost za faktor izvjesnosti, i metode produkta, koja daje minimalnu vrijednost.

Bonczek - Eagin-ova metoda

Ukoliko se želi voditi računa o apsolutnim vrijednostima faktora izvjesnosti i ukoliko je vrijednost većeg od njih preko 50, tada je moguće koristiti tzv. *Bonczek*

- *Eagin*-ovu metodu, koja daje vrijednost bližu vrijednosti dobijenoj metodom minimuma. Ukoliko je navedena vrijednost ispod 50, tada se ovom metodom dobija vrijednost koja je bliža vrijednosti koja se dobije pomoću metode produkta. *Bonczek - Eagin*-ova metoda se definiše:

(CF1 * CF2/100) * (2 - MAX(CF1, CF2)/100).

Rezultati kada su premise logička konjunkcija činjenica

Za navedeni primjer metodom minimuma, metodom produkta, metodom sredina i *Bonczek - Eagin*-ovom metodom se dobijaju, respektivno, slijedeći rezultati: 80, 72, 76 i 79,2.

Fleksibilni mehanizam zaključivanja bi morao da posjeduje mogućnost izbora bilo koje od metoda izračunavanja faktora izvjesnosti zaključka. Očigledno, za dani izbor, mehanizam zaključivanja sam treba da izvede sva izračunavanja i da rezultat zaključivanja prezentira korisniku, zajedno sa izračunatim faktorom izvjesnosti rezultata. Izvršena razmatranja vrijede za slučaj kada se premise formiraju kao logička konjunkcija činjenica.

Premise logička disjunkcija činjenica

Sada će biti razmatran slučaj kada su premise logička disjunkcija činjenica i to jedino interesantan slučaj kada obe činjenice uzimaju tačnu vrijednost. Ta okolnost sugeriše da bi se izvjesnost zaključka mogla da podigne na veću vrijednost od minimuma.

Premisa tipa:

```
PRIV_RAST < 0.02 OR NEZAPOSLENOST > 0.08
```

se može koristiti u pravilu koje, za slučaj da su premise tačne, zaključuje da je ekonomska prognoza loša.

KONFIRMATIVNO ODREĐIVANJE FAKTORA IZVJESNOSTI

U slučaju da su obe činjenice tačne, tada se može primjeniti neka od metoda tzv. konfirmativnog određivanja faktora izvjesnosti. Logika je da tačnost obe činjenice potvrđuje te činjenice, odnosno povećava izvjesnost koju možemo pridružiti rezultatu zaključivanja.

Metoda maksimuma

Nasuprot pravilu minimuma, ovdje se koristi metoda maksimuma, odnosno:

MAX(CF1, CF2)

ili, za navedeni primjer:

MAX(90, 80) = 90.

Metoda sume vjerovatnoća

Metoda sume vjerovatnoća konfirmativnu izvjesnost određuje pomoću izraza:

(CF1 + CF2) - (CF1 * CF2)/100,

što za razmatrani primjer daje vrijednost 98 za faktor izvjesnosti rezultata.

Metoda sredina

I ovdje se analogno formuliše metoda sredina kao:

(MAX(CF1, CF2) + (CF1 + CF2) - (CF1 * CF2)/100)/2.

Bonczek - Eagin-ova metoda

Ova metoda glasi:

MAX(CF1, CF2) + ((CF1 * CF2)/100 * (1 - MAX(CF1, CF2)/100)).

Rezultati:

Navedene metode za razmatrani primjer daju, respektivno, slijedeće vrijednosti faktora izvjesnosti: 90, 98, 94 i 97,2.

Faktor izvjesnosti pravila

U ovim razmatranjima je pretpostavljeno da se izvjesnost zaključaka formira samo na osnovu vrijednosti izvjesnosti koje se izračunavaju za premise. Međutim, i uz čitavo pravilo se može pridružiti odgovarajući stepen izvjesnosti. Ako se u razmatranom primjeru pravila pridruži faktor izvjesnosti od 90, kao izraz povjerenja tom faktoru, onda se to piše kao:

```
IF PRIV_RAST > 0.04 AND NEZAPOSLENOST < 0.075
THEN EK_SITUACIJA = "dobra" CF 90.
```

Faktor izvjesnosti zaključka

Može se postaviti pitanje kako se izračunava izvjesnost zaključka. Ova izvjesnost se izračunava na identičan način, kao kod izračunavanja izvjesnosti premisa koje su formirane konjunkcijom činjenica. Prema tome, kako se izvjesnost premisa može izračunati na četiri različita načina, a i izvjesnost zaključka na takva ista četiri različita načina, onda je moguće koristiti ukupno 16 raznih mogućnosti za izračunavanje izvjesnosti zaključka.

Formiranje meta pravila

U situacijama kada se paralelno gornjem pravilu u bazi znanja nalazi i pravilo:

```
IF PRIV_RAST > 0.02 AND NEZAPOSLENOST < 0.05
THEN EK_SITUACIJA = "dobra" CF 80,
```

može doći do jednovremenog "okidanja" oba pravila.

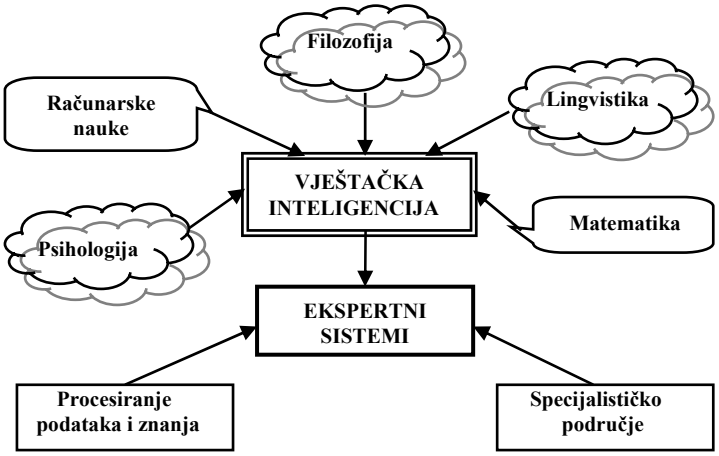
Faktori izvjesnosti se mogu koristiti u formiranju meta pravila, koja odlučuju koje se od dva pravila ovdje može koristiti da se dođe do zaključka i da se koristi. To može da bude, na primjer, pravilo koje daje veću vrijednost izvjesnosti zaključka, odnosno da se koristi **konfirmativni način određivanja izvjesnosti**, kao u slučaju disjunkcije premisa.

4.3. ARHITEKTURA EKSPERTNIH SISTEMA

4.3.1. Opšte o arhitekturi ekspertnih sistema

Interdisciplinarnost ekspertnih sistema

Ekspertni sistemi zadiru u veći broj područja nauke i tehnike, bilo direktno, bilo preko vještačke inteligencije. Na slici 4.1. su prikazana područja koja određuju ES.



Slika 4.1. Područja koja određuju ekspertne sisteme.

Razvojni tim za ES

Ekspertne sisteme razvijaju čitavi timovi stručnjaka, koristeći sva navedena područja nauke i tehnike. Za razmatranje arhitekture ES se može uzeti minimum stručnjaka, koji su u stanju sačiniti ES. Taj minimum se sastoji od: specijaliste iz područja kojem je namjenjen ES - **eksperta** i specijaliste za organizaciju i realizaciju sistema, koji u sebi objedinjuje organizatora baze znanja i sistemskog inženjera za softver. Taj specijalista se može uslovno nazvati **inženjer (tehnolog) znanja**.

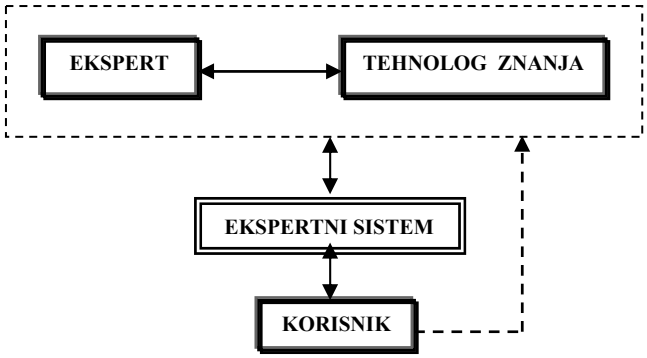
Osnovna šema ES je nezamisliva bez **korisnika**, jer ES ima smisla samo ako se može praktično primjeniti. Testiranje ES, u svim fazama razvoja, se provodi uz pomoć korisnika, što se može prikazati slikom 4.2.

Osnovna arhitektura ekspertnih sistema

Ekspertni sistem je veoma složen programski paket, koji se sastoji od niza manjih programskih cjelina ili modula. Dva osnovna dijela su:

- vezni modul ili interfejs *(interface)*,
- jezgro ekspertnog sistema.

Jezgro ES se sastoji od dva dijela i to:
- baze znanja,
- relacionog modula ili modula za zaključivanje.



Slika 4.2. Razvojni tim za ekspertne sisteme.

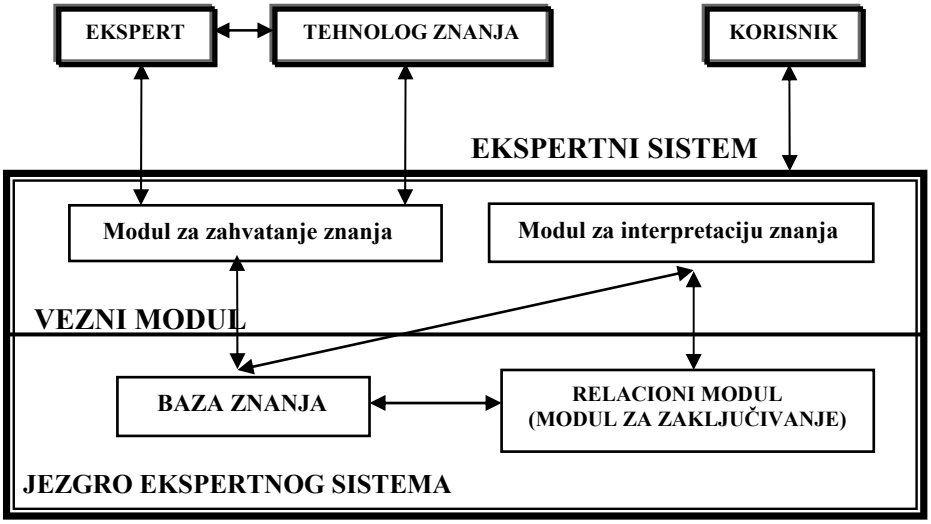
Vezni modul spaja eksperta i tehnologa znanja, sa jedne strane, i korisnika, sa druge strane, sa bazom znanja i relacionim modulom. Iz tog razloga vezni modul se sastoji od dva manja modula (komunikaciona kanala) i to:
- modula za zahvatanje znanja,
- modula za interpretaciju znanja.

Nakon povezivanja ovih modula u jednu cjelinu, dobija se osnovna struktura ES, koja se može prikazati slikom 4.3.

Komunikacija sa ES

Kada je postavljena cjelokupna struktura ES, onda ni ekspert, ni tehnolog znanja, ni korisnik ne pristupaju direktno jezgru ES. Ekspertni sistem sa spoljnjim svijetom komunicira isključivo preko veznog modula i to tako da je za komunikaciju sa ekspertom i tehnologom znanja zadužen modul za zahvatanje znanja, dok je za vezu sa korisnikom zadužen modul za interpretaciju znanja.

Znanja koja sistem dobije preko modula za zahvatanje znanja, raspoređuju se i sređuju u bazi znanja i relacionom modulu. Tek nakon toga jezgro ES je spremno da pruži usluge korisniku sistema preko modula za interpretaciju znanja.



Slika 4.3. Osnovna arhitektura ekspertnih sistema.

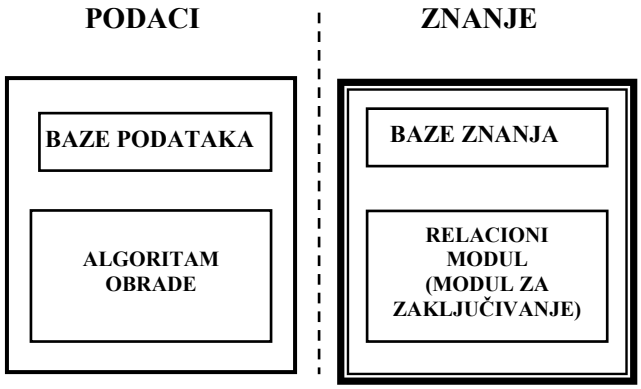
4.3.2. Jezgro ekspertnih sistema

Kvalitetno razumjevanje ES podrazumjeva dobro upoznavanje sa njegovim jezgrom. Iz tog razloga, u najopštijim crtama, će biti prikazano jezgro ES, odnosno princip rada baze znanja i relacionog modula ili modula za zaključivanje.

Baze znanja i procesiranje znanja

Pojavom vještačke inteligencije, a posebno razvojem ES, javljaju se kao svojevrstan razvoj baza podataka i obrade podataka pojmovi baze znanja i procesiranje znanja. Postojeća simetrija između baza podataka i obrade podataka, s jedne strane, i baza znanja i procesiranja znanja, s druge strane, prikazana je na slici 4.4.

Ne može se reći da baze znanja, u užem smislu, nisu takođe neka vrsta baza podataka, a modul za zaključivanje određeni oblik algoritamske obrade.



Slika 4.4. Simetrija između obrade podataka i procesiranja znanja.

Metode za izgradnju jezgra ES

Problem procesiranja znanja i njegovo prikazivanje su osnova na kojoj se zasniva kompletna teorija vještačke inteligencije, pa tako i teorije izgradnje ES. Ova problematika je posebno istraživana sredinom 70-tih godina, tako da se do danas razvio čitav niz metoda i programskih alata, gdje je sve bazirano na matematičkim disciplinama, i to: statistici i teoriji vjerovatnoće, matricama i teoriji grafova, običnoj, višedimenzionalnoj i tzv. "razmazanoj" (*fuzzy*) logici, predikatskom računu, itd.

Vremenom su se izdvojile čitav niz metoda, koje danas dominiraju u izgradnji jezgra ekspertnih sistema, a to su:

- automatsko dokazivanje teorema,
- produkcioni sistemi,
- razvoj ES zasnovan na matematičkoj logici,
- semantičke mreže,
- ramovi znanja (*frames*),
- metode fazi ES,
- metode za izgradnju ES zasnovane na neuronskim mrežama,
- genetički algoritmi i ES,
- agenti, multi agenti i inteligentni agenti,
- inteligentne BP i inteligentni IS.

Iako ove metode imaju niz dodirnih tačaka i određena preklapanja, one predstavljaju zaokružene cjeline.

4.3.3. Vezni modul

Pristup do jezgra u izgrađenom ES je moguć isključivo preko veznog modula. Vezni modul je cjelina od dva svojevrsna komunikaciona kanala i to:

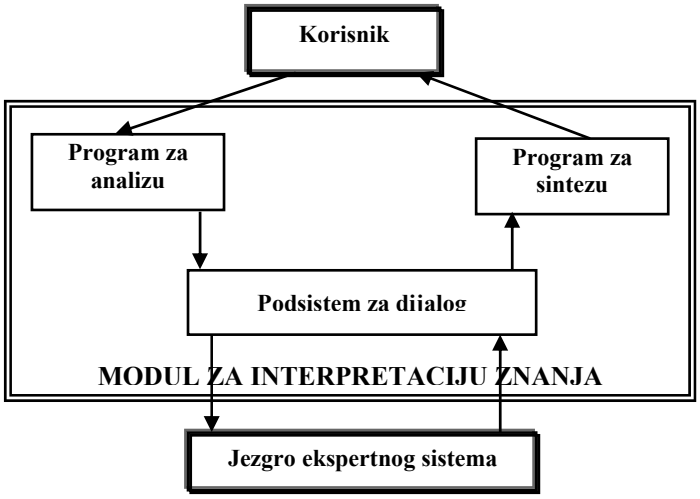
- modula za interpretaciju znanja, predviđenog za korisnika, i
- modula za zahvatanje znanja, koga koriste ekspert i tehnolog znanja.

4.3.3.1. Modul za interpretaciju znanja

Arhitektura modula za interpretaciju znanja

Arhitektura modula za interpretaciju znanja je prikazana na slici 4.5. Modul za interpretaciju znanja se sastoji od slijedećih dijelova:

- programa za analizu saopštenja od strane korisnika sistemu,
- podsistema za dijalog,
- programa za sintezu saopštenja od strane sistema korisniku.



Slika 4.5. Arhitektura modula za interpretaciju znanja.

Program za analizu saopštenja

Program za analizu saopštenja ima zadatak da prima poruke od korisnika. Njime se vrši sintaksna, semantička i morfološka analiza (odnosno, analiza poštovanja pravila jezika, analiza logičkog oblika poruke i analiza oblika poruke), drugim riječima, ovdje se vrši filtriranje poruke.

Podsistem za dijalog

U podsistemu za dijalog prvo se izdvajaju tzv. ključne riječi, uz pomoć rječnika objekata i rječnika relacija, koje se zatim pretvaraju u oblik pogodan za predmetnu oblast. Ovako obrađena informacija se predaje jezgru ES, koje je sada obrađuje prema zadanim pravilima, tj. postavlja dodatna pitanja, traži optimalna rješenja ili daje objašnjenja.

Program za sintezu saopštenja

Rezultat procesiranja znanja se predaje programu za sintezu saopštenja od strane sistema korisniku, koji vrši obradu po sintaksnim, semantičkim i morfološkim pravilima. Pri ovome se podrazumjeva da je čitav ES spreman za rad, da su već uneseni baza znanja i svi mehanizmi zaključivanja.

4.3.3.2. Modul za zahvatanje znanja

Modul za zahvatanje znanja se može razmatrati na niz različitih načina, po područjima nauke i tehnike, sa stanovišta logike i heuristike, sa teorijskog i praktičnog gledišta, itd. Međutim, ova jednostavna pretpostavka je jedan od rubnih kamenova kompletne teorije i prakse ES.

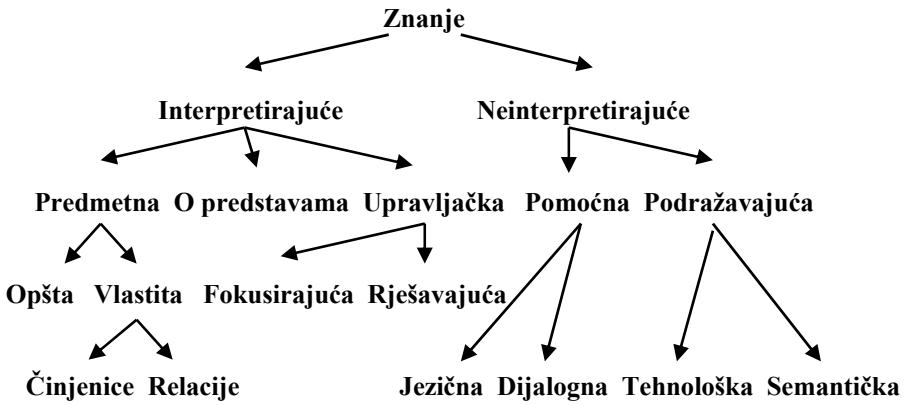
Kompleksnost pojma znanja

Šta je najskuplje u modernom svijetu, šta se najduže i najteže stiče i najviše cijeni? Odgovor je jednoznačan i jasan: ZNANJE! Uostalom, čovjek znanje počinje sticati od rođenja i uči sve do smrti. Kako onda naučiti mašinu da bude savjetnik i pomoćnik u poslovima, gdje se traži vrhunsko, ekspertno znanje? Kako popuniti bazu znanja i kako pravilno definisati modul za zaključivanje, odnosno kako "naučiti" konkretni ES?

Sve ove funkcije treba da obave ekspert i tehnolog znanja, komunicirajući sa jezgrom ES pomoću programa kojim se puni jezgro, odnosno modula za zahvatanje znanja.

Klasifikacija znanja za ES

U slučaju ES klasifikacija znanja izgleda kao na slici 4.6, a rezultat je rada na čitavom nizu projekata. Ovakva klasifikacija znanja je veoma pogodna za zahvatanje, što znači da uspostavlja dobru vezu eksperta i ES.



Slika 4.6. Klasifikacija znanja za ekspertne sisteme.

Postupci zahvatanja znanja za ES

Proces pribavljanja znanja je veoma složen i može se rasčlaniti u nekoliko jasno izdvojenih postupaka. Ti postupci predstavljaju neku vrstu algoritma za sve koji izgrađuju bilo koji ES, odnosno ponavljaju obavljanje određenih zadataka. Zadaci bitni za zahvatanje znanja su:

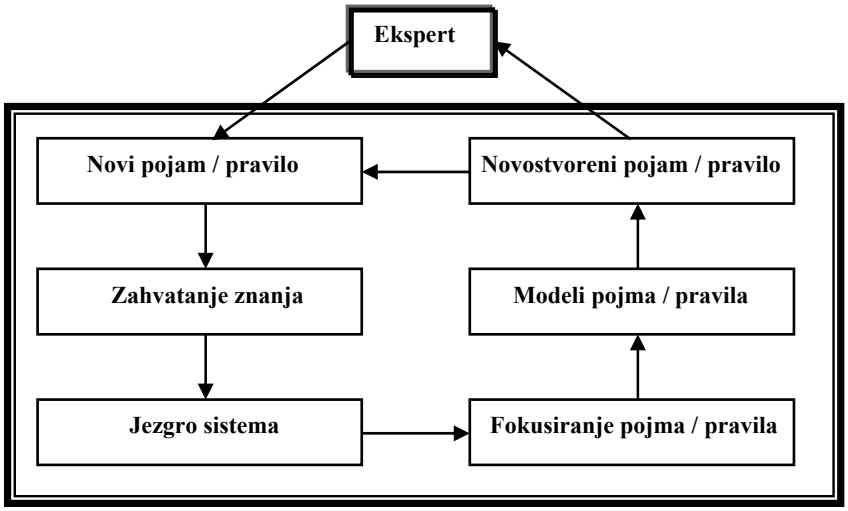
(1) Definisanje neophodnosti proširenja i modifikacije znanja;
(2) Dobijanje potpuno novih znanja o sistemu;
(3) Formiranje novih znanja u obliku koji sistem poznaje;
(4) Usklađivanje starih i novih znanja i prelazak na korak (1).

Navedeni postupci su pokazali da nova generacija ES mora biti zasnovana na dvije osnovne činjenice:

a) Izgradnji modula za zahvatanje znanja, koji omogućava automatizaciju zahvatanja znanja;
b) Automatizaciji zahvatanja znanja ES.

Pored izgradnje modula za zahvatanje znanja i automatizacije zahvatanja znanja, pojam **automatsko učenje** predstavlja treću stepenicu u razvoju i pojedini autori je smatraju jedinim pravim početkom teorije i prakse ES. Ovo mišljenje je bazirano na mogućnosti ES da samostalno i automatski dolazi do potpuno novih zaključaka.

Na slici 4.7. je prikazan mehanizam učenja i samostalnog zaključivanja, o čemu će kasnije još biti riječi.



Slika 4.7. Mehanizam učenja i samostalnog zaključivanja.

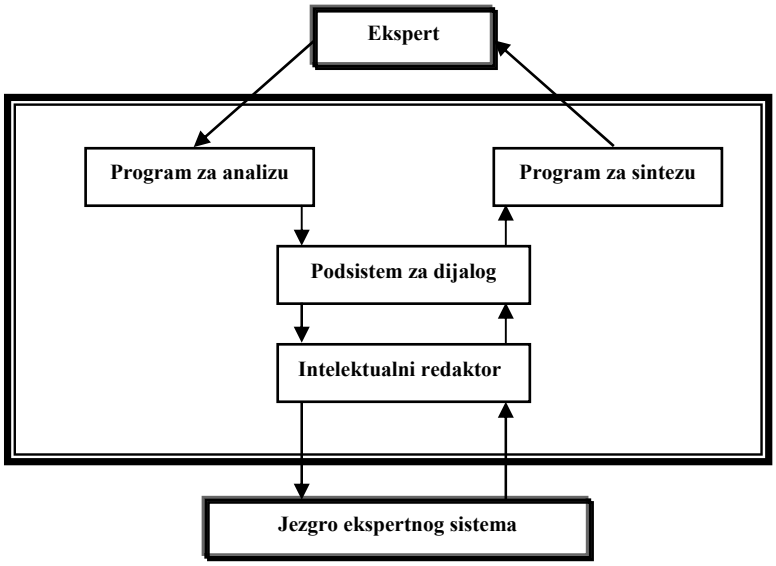
Arhitektura modula za zahvatanje znanja

Modul za zahvatanje znanja mora imati programe za analizu i sintezu saopštenja od strane eksperta, koji se obično nazivaju **mehanizmi uvođenja informacija**, da bi se dobio oblik pogodan za logičku obradu i kasniju predaju novih znanja i pravila jezgru ES.

Međutim, ovdje je od bitnog značenja program koji se može uslovno nazvati **intelektualni redaktor**, slika 4.8.

Intelektualni redaktor

Dužnost intelektualnog redaktora je da prihvati strukturu svih oblika znanja i da stalno obavlja poređenje novih znanja, od strane eksperta, i starih znanja, koja se nalaze u jezgru ES.



Slika 4.8. Arhitektura modula za zahvatanje znanja.

Način na koji intelektualni redaktor obavlja ovu funkciju određuje i sam tip automatskog zahvatanja znanja, odnosno učenja:

- induktivno, ili posebna vrsta analognog,
- jezičko konceptualno,
- eliminaciono, uz pomoć stabla grananja,
- zvjezdasto,
- uz pomoć uzoraka, itd.

Svaka od ovih metoda ima svoje prednosti i nedostatke, a mogu se koristiti i njihove kombinacije.

Mehanizam učenja i samostalnog zaključivanja

Zajedničko za sve oblike i tipove zahvatanja znanja je tzv. **opšti mehanizam učenja i samostalnog zaključivanja** (slika 4.7). Prema toj šemi upravo i radi intelektualni redaktor.

Da li se radi o induktivnom, jezičko konceptualnom i/ili nekom drugom obliku zahvatanja znanja, odlučuju modeli pojmova i/ili pravila. Očigledno, radi se o sistemu sa povratnom vezom, gdje se novostvoreni pojam i/ili pravilo ukomponuju samo ako su prošli dodatni test mehanizma zaključivanja i eksperta.

Jedna od važnih karakteristika ovog načina učenja je i stalni dijalog sa ekspertom. Sistem donosi svoje zaključke i postavlja pitanja svome "učitelju", te pokazuje svoja rješenja. Za ovakav tip konverzije zadužen je podsistem za dijalog.

Podsistem za dijalog

Podsistem za dijalog, u širem smislu, u sebi sadrži i analizu i sintezu saopštenja, isto kao i modul za interpretaciju znanja. U zajednici sa intelektualnim redaktorom predstavlja modul za zahvatanje znanja.

U ovom poglavlju:

- ***Produkciona pravila***
- ***Opšti oblik produkcionih pravila***
- ***Šema produkcionog sistema***
- ***Uređeni produkcioni sistemi***

5

PRODUKCIONI SISTEMI

5.1. PRODUKCIONA PRAVILA

Produkcioni sistem se sastoji od skupa parova tipa "uslov – akcija", koji se nazivaju produkciona pravila ili pravila odlučivanja, baze podataka stanja sistema smještenih u radnu memoriju i procedura za izvršavanje produkcionih pravila, odnosno mehanizma zaključivanja. Struktura ovog sistema je predstavljena na slici 5.1. Produkcioni sistemi se najčešće primjenjuju za razvoj ES i sistema kognitivne arhitekture.

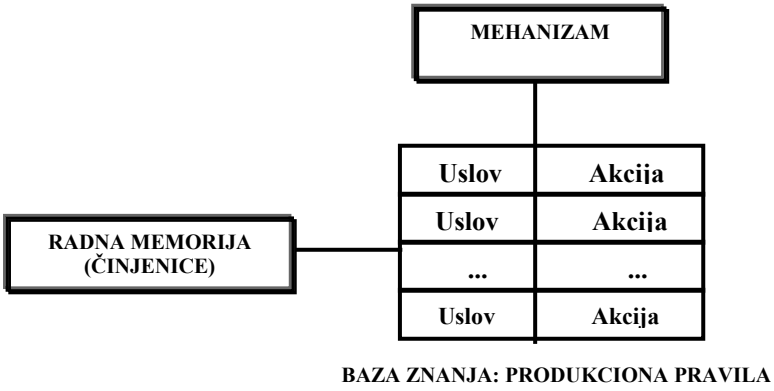
Arhitektura produkcionih sistema

- Interfejs,
- Baza znanja: produkciona pravila (*If-Then* pravila),
- Radna memorija: činjenice,
- Mehanizam zaključivanja.

Baza znanja: produkciona pravila se baziraju na najstarijem obliku tzv. predikatne ili obične logike oblika "AKO ... → TADA" *("IF ... → THEN")*. Svako produkciono pravilo sadrži uslov koji mora biti zadovoljen prije nego što se izvrši akcija.

Radna memorija (činjenice) je skup informacija koje se testiraju da bi se zaključilo o istinitosti nekog uslova.

Mehanizam zaključivanja je jednostavno programska petlja, koja testira istinitost uslova pravila i izvršava akcije kad je uslov tačan.



Slika 5.1. Struktura produkcionog sistema.

5.2. OPŠTI OBLIK PRODUKCIONIH PRAVILA

Opšti oblik produkcionih pravila ima slijedeći oblik:

```
IF situacija s THEN akcija a
IF uslov u THEN posljedica p
IF p THEN s TO DEGREE d
```

Osnovne prednosti produkcionih pravila

- modularnost,
- dopunjivost,
- jednostavnost,
- izmjenljivost .

Mehanizam zaključivanja može imati dva pristupa rasuđivanju

- Olančanje unaprijed (*forward chaining*). Sve činjenice su dostupne na početku zaključivanja. Ovaj pristup je pogodan za sintezu znanja;
- Olančanje unazad (*backward chaining*). Sva ciljna stanja su dostupna na početku zaključivanja. Ovaj pristup je pogodan za dijagnostiku;
- Olančavanje u oba smjera (*bi-directional*).

Algoritam zaključivanja

1. *Match*: Pronalaze se sva primjenljiva pravila (If dio je zadovoljen);
2. *Conflict resolution*: Bira se pravilo koje će se izvršiti;
   - odbaciti pravila koja su već ranije primjenjena,
   - izabrati novo pravilo *(conflict-resolution strategy);*
3. *Act*: Dodati novu činjenicu u bazu znanja.

Strategije za razrješavanje konflikata

No duplication: Izabrati pravilo koje daje nove zaključke;
Do one: Izabrati prvo pravilo čiji je uslov ispunjen;
Do all: Primjeniti sva pravila;
Do the most specific: Primjeniti najspecifičnije pravilo;
Do the most recent: Izabrati pravilo čiji uslov koristi najnovije činjenice.

PRIMJER: *Radna memorija = (zeleno, teško-8kg).*
Baza znanja:

- `P1: IF zeleno,` `THEN povrce-voce`
- `P2: IF upakovano-u-male-kontejnere` `THEN delikates`
- `P3: IF rashladjeno OR povrce-voce,` `THEN kvarljivo`
- `P4: IF tesko-8kg AND jeftino AND NOT kvarljivo,` `THEN osnovna-namirnica`
- `P5: IF kvarljivo AND tesko-8kg` `THEN curka`
- `P6: IF tesko-8kg AND povrce-voce` `THEN lubenica`

PRIMJER: ***Test iz matematike.***

Može se razmatrati slijedeća situacija. Na najavljeni test iz matematike u školu nije došlo pola razreda. Nastavnik zaključuje:

AKO

(1) U gradu i školi nema epidemije, recimo gripe, i
(2) U prethodnim razredima, bez testa, nema izostanaka, i
(3) Ova ocjena bitno određuje ukupan uspjeh učenika;

TADA

(1) Odsutni učenici nisu naučili gradivo,
(2) Misle da će se izvući samo sa neopravdanim časom,
(3) Test će biti drugi put, ali bez najave.

Težinski faktor

Naravno, u ovom zaključivanju može biti i greška. Recimo, pola razreda dolazi u školu autobusom ili vozom, koji taj dan iz opravdanih razloga nisu prevozili putnike. Zbog toga ovakvo zaključivanje obično mora sa sobom nositi i tzv. **težinsku vrijednost**. U takvom kontekstu svakom AKO i svakom TADA može se pridodati težinski faktor u rasponu od, na primjer, +1 do -1.

Treba reći da veličina težinskog faktora, njegov oblik i finoća podjele, zavise isključivo od znanja eksperta i njegove definitivne preporuke. Zadatak inženjera znanja je da ove vrijednosti spremi u bazu znanja. Konkretno, zaključak TADA pod brojem (1) "Odsutni učenici nisu naučili gradivo", ocjenjuje se jednom od mogućnosti:

1 - "sigurno nisu",
0,5 - "vjerovatno nisu",
0 - "ne zna se",
-0,5 - "vjerovatno jesu", i
-1 - "sigurno jesu".

Na potpuno isti način se rangiraju i upiti AKO.

Na osnovu ovakvog logičkog grananja, sa jednim ili više AKO te jednim ili više TADA, i pridodanih težinskih faktora, sistem samostalno donosi definitivan odgovor, a upravo u tome se i ogleda mehanizam zaključivanja:

- rangirati zaključke i/ili
- prihvatiti zaključak x i/ili
- odbaciti zaključak y.

Za svaki ciklus, odrediti:

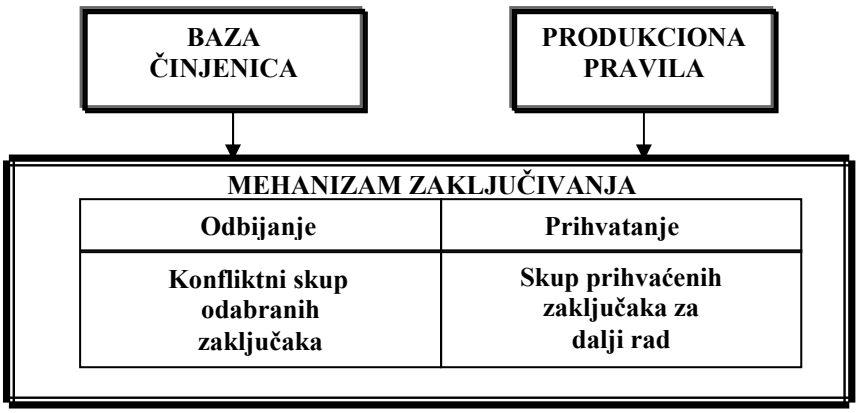
- pravila koja su primjenljiva,
- pravila koja treba deaktivirati,
- izabrano pravilo, ako se izabira pravilo sa najnižim brojem,
- novo stanje radne memorije.

Problemi sa produkcionim pravilima

- velike baze znanja mogu biti spore,
- mala količina informacija na osnovu kojih se donosi zaključak,
- pravila mogu biti redundantna.

5.3. ŠEMA PRODUKCIONOG SISTEMA

Na slici 5.2. je prikazana na nešto drugačiji način šema dijelova produkcionog sistema, gdje su dani svi istraživani objekti (u navedenom primjeru: učenici, test, matematika, biti prisutan) i produkciona pravila koja ih vežu, u užem smislu (AKO ... → TADA).



Slika 5.2. Šema dijelova produkcionog sistema.

Mehanizam zaključivanja je osnovna komponenta u metodi produkcionih pravila, što se može uočiti na osnovu navedenog primjera. Njegov kvalitet zavisi od dobro odabranih pitanja, dobro odabranih odgovora i, što je najvažnije, njihovog

pravilnog ocjenjivanja.

PRIMJER: *Za ilustraciju skupa pravila može se uzeti primjer ES, čiji je zadatak da za dani cijeli broj x da "savjet" kako izgleda njegova predstava sa rimskim ciframa.*

1. Produkciona pravila:

(a) Ako je x nula (nema vrijednosti) tada zatraži od korisnika da unese x;
(b) Ako x nije nula i veće je od 39 tada štampaj "SUVIŠE VELIKA VRIJEDNOST" i napravi x nula;
(c) Ako x nije nula i nalazi se između 10 i 39 tada štampaj "X" i smanji x za 10;
(d) Ako x nije nula i jednako je 9 tada štampaj "IX" i smanji x na 0;
(e) Ako x nije nula i nalazi se između 5 i 8 tada štampaj "V" i smanji x za 5;
(f) Ako x nije nula i jednako je 4 tada štampaj "IV" i smanji x na 0;
(g) Ako x nije nula i nalazi se između 1 i 3 tada štampaj "I" i smanji x za 1;
(h) Ako je x 0 tada štampaj "KRAJ" i napravi x jednako nula (nema vrijednosti).

2. Baza činjenica: *vrijednost promjenljive x.*

3. Mehanizam zaključivanja *pretražuje pravila, koja mogu biti u bilo kom redoslijedu, što odražava neproceduralnost sistema produkcionih pravila. Vrši se testiranje uslova svakog pravila, sve dok se ne naiđe na uslov koji je tačan. Tada se odgovarajuća akcija, vezana za tačan uslov izvršava, ("okida") i eventualno mijenja baza činjenica.*

Na primjer, ako x ima vrijednost 7, tada se izvršava produkciono pravilo (e), odnosno njegova akcija:

- *štampaj "V" i smanji x za 5, se "okida".*

Nakon toga, mehanizam zaključivanja počinje ponovo da pretražuje skup pravila i postupak se ponavlja beskonačno, ili dok se mehanizam zaključivanja na neki način ne isključi. Obično je stanje sistema, u kome ni jedan uslov nije tačan, uslov za prestanak rada ili se izvršava akcija koja eksplicitno zaustavlja interpreter.

5.4. UREĐENI PRODUKCIONI SISTEMI

Da bi se minimizirao potrebni broj testiranja uslova, potrebno je skup pravila urediti u odgovarajući redoslijed, koji će da izbjegne nepotrebno testiranje. Pogodnim uređenjem skupa pravila moguće je ostvariti i paralelizaciju procesa testiranja pravila, čime se može značajno skratiti vrijeme rada. Ovo naročito postaje bitno kada sistem radi sa više hiljada pravila.

Otežavajuća okolnost pri radu sa produkcionim sistemima je da se izbor pravila, koji će se ispitati, vrši **linearnim pretraživanjem** liste pravila i to stalno iz početka. Vrijeme potrebno da se testira svaki uslov varira sa složenošću uslova i u slučaju da je uslov iskazan kao konjunkcija poduslova, postoji velika vjerovatnoća da će neko AND rano u nizu izbaciti dano pravilo.

Umjesto linearnog pretraživanja se može efikasno koristiti neka od metoda pretraživanja stabla. Postavljanjem uslova u **balansno stablo odlučivanja**, može se značajno smanjiti vrijeme pretraživanja pravila. Ovakva stabla se nazivaju i **diskriminacione mreže uslova** za izbor pravila. Akcije pravila su krajnji čvorovi (listovi) stabla odlučivanja.

Uvođenjem diskriminacionih mreža se smanjuje broj testiranja, ali se gubi na jednostavnosti i modularnosti originalnog produkcionog sistema. Postaje komplikovanije ubacivanje novog pravila ili brisanje postojećeg, zbog toga što je potrebno držati diskriminacionu mrežu balansiranom da bi se minimizirao broj testiranja uslova.

Kompajlirani produkcioni sistemi bili bi pokušaj da se objedini modularnost čistog produkcionog sistema i efikasnost diskriminacionih mreža. Ovdje bi se pravila pisala tako da ih program kompajler automatski prevodi u efikasan program, zasnovan na diskriminacionoj mreži. Pogodan način može biti specificiranje uslova u pravila putem forme, koja treba da se poredi sa ulazom i u slučaju potpunog poklapanja uslov poprima tačnu vrijednost.

Standardna konstrukcija pravila

Polazeći od toga da produkciona pravila govore šta treba uraditi ukoliko se zadovolji neki uslov, odnosno neka situacija postoji, standardna konstrukcija za iskazivanje pravila je IF ... THEN konstrukcija, tj.:

IF uslov THEN akcija

Za ilustraciju može da se navede slijedeći primjer.

PRIMJER: *Ukoliko se želi iskazati heuristika koja kaže da se nova količina prodaje nekog artikla određuje tako da ukoliko je tekuća prodaja veća za 15% od tekuće količine, onda se nova količina određuje kao tekuća količina uvećana za razliku između tekuće prodaje i 15% uvećane tekuće količine.*

Koristi se pravilo slijedećeg oblika:

PRAVILO:

```
IF PRODAJA > 1.15 * KOLIČINA
THEN NOVA KOLIČINA = KOLIČINA +(PRODAJA -1.15 * KOLIČINA)
```

KOMENTAR: *U slučaju da je prodaja veća od količine za više od 15%, nova količina se određuje uvećanjem količine za razliku prodaje i 15% uvećane količine.*

Navedeni KOMENTAR služi kao objašnjenje korisniku postupka zaključivanja, na taj način opravdavajući savjet do koga se došlo korištenjem ES.

Produkciona pravila predstavljaju najpopularniju metodu predstavljanja znanja u ES i veoma su pogodna za formalno predstavljanje preporuka, direktiva ili strategija. Drugim riječima, pravila su pogodna kad je znanje iz danog domena rezultat asocijacija koje su sticane godinama rada i iskustva u rješavanju problema u danom domenu. Ove asocijacije se obično, u ovom kontekstu, nazivaju heuristikama.

U ovom poglavlju:

- ***Iskazni račun***
- ***Primjena iskaznih formula***
- ***Tautologije, kontradikcije i istinitosne tabele***
- ***Predikatski račun***
- ***Sintaksa predikata i rečenica***
- ***Pravilo zaključivanja modus ponens***

6

EKSPERTNI SISTEMI ZASNOVANI NA MATEMATIČKOJ LOGICI

6.1. ISKAZNI RAČUN

Među rečenicama koje se upotrebljavaju u svakodnevnom komuniciranju, posebno mjesto zauzimaju one kojima se pridružuje svojstvo da su tačne ili netačne. Takve rečenice se nazivaju **iskazima**.

***PRIMJERI*:**

Isakazi su*: Snijeg je bijel. Kuća je žuta. Moje ime je Ivo.*
Iskazi nisu*: Doviđenja. Dobar dan.*

Definicija:

Simboli iskaznog računa su:

- iskazni simboli: P, Q, R, S, T, ...
- simboli istinitosti: *true*, *false*, ($\top$, $\bot$)
- simboli povezivanja: $\wedge$, $\vee$, $\neg$, $\Rightarrow$, =

Iskazni simboli znače prijedloge, ili rečenice o svijetu, koje mogu biti ili tačne ili netačne. Prijedlozi se obilježavaju sa velikim slovima pri kraju engleskog alfabeta.

Materijal u ovom poglavlju većim dijelom je preuzet sa Interneta htp:/www.iis.ns.ac.yu.htlm, (S. Stankovski, "Iskazni račun"), 2004.

Iskazne rečenice (ili formule) se definišu na sledeći način:

Definicija:
Svaki iskazni simbol i simbol istinitosti je rečenica.
PRIMJER: ***true*****, P, Q,** ***i*** **R** ***su četiri rečenice.***

Negacija rečenice je rečenica.
PRIMJER: **¬ P** ***i*** **¬** ***false su rečenice.***

Konjunkcija **(∧) dvije rečenice je rečenica.**
PRIMJER*:* **P ∧ ¬ P** ***je rečenica.***

Disjunkcija (∨) dvije rečeniče je rečenica.
PRIMJER: P ∨ ¬ P *je rečenica*.

Implikacija jedne rečenice u drugu je rečenica.
PRIMJER: P ⇒ Q *je rečenica*.

Ekvivalencija dvije rečenice je rečenica.
PRIMJER: P ∨ Q ⇔ R je rečenica.

Iskazne rečenice se grade samo konačnom primjenom prethodnih pravila. Legalne rečenice se nazivaju i **dobro formiranim formulama** ili **DFF**.

U izrazu P ∧ Q, P i Q se zovu **konjunkti**. U izrazu P ∨ Q, P i Q se nazivaju **disjunktima**. U implikaciji, P ⇒ Q, P je **premisa** ili **uzrok**, a Q je **posljedica**.

U rečenicama iskaznog računa simboli zagrada (,),[,], se koriste da bi se grupisali simboli u podizraze i tako kontrolisao redoslijed izračunavanja.

PRIMJER: (P ∨ Q) ⇔ R *je u potpunosti različito od* P ∨ (Q ⇔ R), *a to se može dokazati korištenjem istinitosnih tabela.*

Ako se iskazi "Sneg je bijel" i "Ja se zovem Aco" označe redom sa P i Q, onda se formulom (P ∧ Q) označava složeni iskaz "Sneg je bijel i ja se zovem Aco". Primjećuje se da se formulom (P ∧ Q) može označiti i mnogo drugih složenih iskaza (npr.: "Kuća je žuta i pas je sive boje"), pretpostavljajući da P i Q

imaju drugo značenje (u ovom primjeru "Kuća je žuta", odnosno "Pas je sive boje").

U prethodnoj definiciji, a i na dalje, često se koriste slova A, B, C, itd. da označe iskazne, odnosno, predikatske formule. Takve formule su š**ema formule** i predstavljaju veći broj "pravih" formula. Te "prave" formule se od šema formula dobijaju tzv. sistematskom zamjenom. Ova zamjena podrazumjeva da se isto slovo, u šema formuli zamjenjuje uvijek istom formulom sastavljenom od iskaznih, tj. predikatskih, simbola. Tako dobijene formule su **instance šema formula**.

6.1.1. Primjena iskaznih formula

U knjizi sa iskaznim formulama se prvenstveno razmatra njihova istinitosna vrijednost. Simbolima T i $\perp$ redom su označene istinitosne vrijednosti: tačno i netačno. Osnovna pravila za izračunavanje istinitosnih vrijednosti iskaznih formula, na osnovu istinitosne vrijednosti njihovih podformula, dana su TABELOM 6.1.

TABELA 6.1.

A	B	$\neg$ A	A $\wedge$ B	A $\vee$ B	A $\Rightarrow$ B	A $\Leftrightarrow$ B
T	T	$\perp$	T	T	T	T
T	$\perp$	$\perp$	$\perp$	T	$\perp$	$\perp$
$\perp$	T	T	$\perp$	T	T	$\perp$
$\perp$	$\perp$	T	$\perp$	$\perp$	T	T

Istinitosna vrijednost neke formule se može jednoznačno izračunati na osnovu istinitosne vrednosti iskaznih slova koja ulaze u tu formulu.

PRIMJER: *Neka su istinitosne vrijednosti iskaznih slova* P i Q *redom* T i $\perp$. *Tada, prema pravilima iz TABELE 6.1., istinitosne vrednosti formula* P $\wedge$ Q, $\neg$Q, P $\neg$Q i (P $\wedge$ Q) $\Rightarrow$ (P $\Leftrightarrow$ $\neg$Q), *su redom* $\perp$, T, T i T.

Pojmovi interpretacije i tačnosti pri interpretaciji su u uskoj vezi sa istinitosnom vrednošću iskaznih formula.

Definicija

Interpretacija I je preslikavanje koje iskaznim slovima pridružuje vrednosti iz skupa {T,⊥}. Iskazna formula A je tačna pri interpretaciji I (sa oznakom: I x A) ako je, polazeći od istinitosne vrednosti pri interpretaciji I iskaznih slova koja se javljaju u A, izračunato T kao istinitosna vrednost formule A. Konačan skup iskaznih formula $A_1, A_2, \ldots, A_n$ je tačan pri interpretaciji I ako je formula $A_1 \wedge A_2 \wedge \ldots \wedge A_n$ tačna pri interpretaciji I.

Dodjeljivanje istinitostne vrijednosti iskaznoj rečenici se naziva **interpretacijom,** a to je tvrdnja o istinitosti te rečenice u nekom mogućem svijetu.

6.1.2. Tautologije, kontradikcije i istinitosne tabele

Na istinitosnu vrijednost formule utiču samo istinitosne vrednosti iskaznih slova koja se u njoj pojavljuju. Ako u nekoj formuli ima n različitih iskaznih slova, različitih interpretacija ovih iskaznih slova ima ukupno 2n.

PRIMJER: *U TABELI 6.2. su u četiri reda prikazane sve četiri (4 = 2 · 2) različite interpretacije iskaznih slova* P *i* Q.

TABELA 6.2.

	P	Q
I1	T	T
I2	T	⊥
I3	⊥	T
I4	⊥	⊥

Definicija

Formula A je tautologija (kao sinonim se često upotrebljava i termin valjana formula), sa oznakom ×A, ako je tačna pri svakoj interpretaciji. Formula A je kontradikcija ako nije tačna ni pri jednoj interpretaciji.

Tautologije su vrsta formula koje imaju veliku primjenu. To su formule koje su tačne na osnovu svoje forme, odnosno načina na koje su izgradjene korištenjem simbola logičkih veznika, a ne na osnovu neke posebne vrijednosti iskaznih slova od kojih su sastavljene. Tautologije su zato obrasci ispravnog zaključivanja, što će se kasnije ilustrovati primjerima. Pošto je negacija svake tautologije kontradikcija, i to uvijek netačne, formule se često koriste kao pokazatelji stranputica do kojih se došlo u zaključivanju.

PRIMJER: *Formula* $((P\Rightarrow Q)\wedge P)\Rightarrow Q$ *je jedna tautologija. Ovo se neposredno provjerava ispisivanjem sve četiri različite interpretacije iskaznih slova* P *i* Q*, računanjem istinitosnih vrijednosti podformula formule i, konačno, računanjem istinitosne vrednosti same formule koja se razmatra, kao što je prikazano u TABELI 6.3. Sličnim razmatranjem se ustanovljava da je formula* $(P\Rightarrow Q)\wedge(P\wedge\neg Q)$ *kontradikcija.*

Postupak koji je korišten u prethodnom primjeru se naziva **metoda istinitosnih tabela**.

TABELA 6.3.

P	Q	$P\Rightarrow Q$	$(P\Rightarrow Q)\wedge P$	$((P\Rightarrow Q)\wedge P)\Rightarrow Q$
T	T	T	T	T
T	$\perp$	$\perp$	$\perp$	T
$\perp$	T	T	$\perp$	T
$\perp$	$\perp$	T	$\perp$	T

Neke od važnih tautologija su (sa n su označene uvek tačne formule, dok o označava uvek netačne formule):

- $A\Rightarrow A$ (zakon refleksivnosti implikacije),
- $A\vee\neg A$ (zakon isključenja trećeg),
- $\neg(A\wedge\neg A)$ (zakon neprotivrječnosti),

- $\neg\neg A \Rightarrow A$ (zakon dvojne negacije),
- $((A \Rightarrow B) \wedge (B \Rightarrow C)) \Rightarrow (A \Rightarrow C)$ (zakon tranzitivnosti za implikaciju),
- $((A \Leftrightarrow B) \wedge (B \Leftrightarrow C)) \Rightarrow (A \Leftrightarrow C)$ (zakon tranzitivnosti za ekvivalenciju),
- $(A \Rightarrow B) \Leftrightarrow (\neg A \vee B)$ (zakon uklanjanja implikacije),
- $(A \Leftrightarrow B) \Leftrightarrow ((A \Rightarrow B) \wedge (B \Rightarrow A))$ (zakon uklanjanja ekvivalencije),
- $(\neg A \Rightarrow (B \wedge \neg B)) \Rightarrow A$ (zakon svođenja na apsurd),
- $(A \wedge A) \Leftrightarrow A$ (zakon idempotencije za $\wedge$),
- $(A \vee A) \Leftrightarrow A$ (zakon idempotencije za $\vee$),
- $(A \wedge B) \Leftrightarrow (B \wedge A)$ (zakon komutativnosti za $\wedge$),
- $(A \vee B) \Leftrightarrow (B \vee A)$ (zakon komutativnosti za $\vee$),
- $(A \wedge (B \wedge C)) \Leftrightarrow ((A \wedge B) \wedge C)$ (zakon asocijacijativnosti za $\wedge$),
- $(A \vee (B \vee C)) \Leftrightarrow ((A \vee B) \vee C)$ (zakon asocijacijativnosti za $\vee$),
- $(A \wedge (A \vee B)) \Leftrightarrow A$ (zakon apsorpcije),
- $(A \vee (A \wedge B)) \Leftrightarrow A$ (zakon apsorpcije),
- $(A \wedge (B \vee C)) \Leftrightarrow ((A \wedge B) \vee (A \wedge C))$ (zakon distribucije $\wedge$ prema $\vee$),
- $(A \vee (B \wedge C)) \Leftrightarrow ((A \vee B) \wedge (A \vee C))$ (zakon distribucije $\vee$ prema $\wedge$),
- $(\neg(A \wedge B)) \Leftrightarrow (\neg A \vee \neg B)$ (zakon De Morgana),
- $(\neg(A \vee B)) \Leftrightarrow (\neg A \wedge \neg B)$ (zakon De Morgana),
- $(A \wedge (A \Rightarrow B)) \Rightarrow B$ (modus ponens),

- $((A \Rightarrow B) \wedge \neg B) \Rightarrow \neg A$ (modus tolens),
- $(A \vee n) \Leftrightarrow n$ (zakon disjunkcije sa tautologijom),
- $(A \wedge n) \Leftrightarrow A$ (zakon konjunkcije sa tautologijom),
- $(A \vee o) \Leftrightarrow A$ (zakon disjunkcije sa kontradikcijom),
- $(A \wedge o) \Leftrightarrow o$ (zakon konjunkcije sa kontradikcijom).

Za ove formule, metodom istinitosnih tabela, se može utvrditi da jesu tautologije. Istinitosne tabele se koriste i za utvrdjivanje da li se neka formula može zadovoljiti, ili je kontradikcija, tj. ne može se zadovoljiti.

6.2. PREDIKATSKI RAČUN

U iskaznom računu svaki atomski simbol (P, Q, itd) je iskaz o nekakvom složenom problemu. Ne postoji način da se koriste komponente pojedinačnih tvrdnji. Predikatski račun dozvoljava tu mogućnost.

PRIMJER: *Umjesto da iskazni simbol* P *označava cijelu rečenicu "u utorak je padala kiša ", može se kreirati predikat vrijeme, koje opisuje vezu između vremena i datuma vrijeme(utorak, kiša).*

Pomoću pravila zaključivanja radi se sa izrazima, koristeći njihove pojedinačne komponente i izvode nove rečenice.

Predikatski račun takođe dozvoljava i korištenje promjenljivih u izrazima. Promjenljive dozvoljavaju da se kreiraju opšte tvrdnje o klasama objekata.

PRIMER: *Može se tvrditi da sve vrijednosti za X, gde je X dan u nedjelji, rečenica vrijeme(X, kiša) je istinita, na primjer kiša pada svaki dan.*

6.2.1. Sintaksa predikata i rečenica

Prije nego što se definiše sintaksa ispravnih izraza u predikatskom računu, biće definisan alfabet i gramatika za kreiranje simbola ovog jezika. Ovo odgovara leksičkom aspektu definicije programskog jezika. Simboli predikatskog računa,

kao što su znakovi u programskim jezicima, su nesvodljivi sintaksni elementi: ne mogu se rastaviti na komponente pomoću dozvoljenih operacija jezika.

Simboli predikatskog računa su nizovi slova i cifara, koje obavezno počinju slovom. Prazna mjesta i nealfanumerički znaci se ne smiju pojavljivati u nizu znakova. Može se koristiti podcrta "_" za poboljšavanje čitljivosti.

Definicija

Alfabet za formiranje simbola predikatskog računa se sastoji od:

1. Skupa slova, malih i velikih, engleskog alfabeta.
2. Skupa cifara: 0, 1, 2, ... , 9.
3. Podcrte "_".

Simboli u predikatskom računu počinju sa slovom i mogu sadržavati bilo koji od dozvoljenih karaktera.

Dozvoljeni karakteri u alfabetu simbola predikatskog računa su: a R 6 9 p _ z.
Karakteri koji nisu dozvoljeni: % $ & " " / .

PRIMJERI: *Dozvoljeni simboli predikatorskog računa su:*
Ivo; vatra3; mirko_and_marko; ana; XXXXX

PRIMJERI: *Simboli i nizova koji nisu dozvoljeni:*
3pero; "prazna mjesta nisu dozvoljena"; ab%cd; oooo71.patak!!!

Formalni ekvivalentni su: v(i, a) i voli(ivo, ana), imaju istu strukturu. Druga struktura za čitaoca može biti od velike pomoći, jer nagovještava kakav je odnos u izrazu.

Zagrade "( )", zarezi "," i tačke "." se koriste za konstrukciju pravilno formiranih izraza i ne označavaju objekte i relacije u stvarnom svijetu. Oni se nazivaju nepravim simbolima.

Simboli predikatorkog računa mogu predstavljati:
- promjenljive,
- konstante,
- funkcije ili predikate.

Promjenljive su simboli koji se koriste za definisanje opštih klasa objekata i osobina. Promenljive se prikazuje uvijek sa velikim početnim slovom. Mirko, MARKO, KAta su pravilno formirane, dok mirKo i marko nisu pravilno formirane promenljive.

Konstante obilježavaju objekte ili osobine. Konstante moraju počinjati sa malim slovima. Npr.: mirko, drvo, visok, plavo su primjeri pravilno formiranih simbola. Konstante *true* i *false* su rezervisani za istinitostne simbole.

Predikatski račun, takođe, dozvoljava korištenje **funkcija** nad objektima. Simboli funkcija (kao i konstanti) počinju sa malim slovom. Funkcije označavaju preslikavanje jednog ili više elemenata skupa (koji se naziva domenom) u jedinstveni elemenat drugog skupa (koddomen funkcije). Elementi domena i koddomena funkcije su objekti u svijetu govora. Uz uobičajene funkcije, kao što su sabiranje i množenje, mogu se navesti i funkcije koje vrše preslikavanje između nenumeričkih domena.

Svaki funkcijski simbol ima uz sebe i ***arity***, koji pokazuje broj elemenata domena koji se preslikavaju na elemente koddomena. Dakle, otac može označavati funkciju koja ima arity 1, jer preslikava ljude na njihovog (jedinog) muškog roditelja. Operacija plus može biti funkcija sa *arity*-em 2, jer preslikava dva broja na njihovu aritmetičku sumu.

Funkcijski izraz je funkcijski simbol poslije koga dolazi njegov **argument**. Argumenti su elementi iz domena funkcije, broj argumenata je jednak *arity*-u funkcije. Argumenti su unutar zagrada i razdvajaju se zarezima.

PRIMJERI:

f(X,Y); otac(morko); cijena(limun)

su pravilno formirani funkcijski izrazi.

*Ako je plus funkcija čiji je **arity 2**, sa cjelobrojnim domenom, tada je:*

plus(2,3)

*funkcijski izraz čija je vrijednost cio broj 5. Proces zamjene funkcije sa njegovom vrednošću se naziva **izračunavanjem**.*

U predikatskom računu ***term*** je ili konstanta, promenljiva ili funkcijski izraz. Termovi označavaju objekte i osobine u domenu.

PRIMJERI: *Termovi su:*

mačka; množi(2,3); plavo; X; majka(mara); kata

Simboli u predikatorskom računu mogu prestavljati predikate. Simboli predikata, kao i konstanti i funkcija, počinju malim slovom. Imena predikata predstavljaju odnos između nijednog ili više objekata. Broj povezanih objekata je *arity* predikata.

PRIMJERI: *Primeri predikata su:*

voli; jednako; na; blizu; na_stolu

Atomska rečenica (formula) u predikatskom računu je predikat *arity*-a n, iza koga je n termova između zagrada i koji su razdvojeni zarezima. Rečenice predikatskog računa su ograničene tačkom.

PRIMJERI: *Primjeri atomskih rečenica:*

voli (stevan, slatko).; voli (X, X).; drugovi (mirko, marko).; drugovi (otac(marko)), otac(andrija)).; pomaže (rade, branko).

Atomske rečenice se još zovu atomskim izrazima, atomima, ili prijedlozima.

Kada se promjenljiva pojavi kao argument u rečenici, ona označava neodređeni objekat domena. Predikatski račun ima i dva simbola, kvantifikatore promenljivih i "$", koji ograničavaju smisao rečenice koja sadrži promjenljivu. Kvantifikator je praćen promenljivom i rečenicom.

PRIMJERI:

$\forall$ X voli(X, sladoled).
$\exists$Y drugovi(Y, petar).

Univerzalni kvantifikator, $\forall$, ukazuje da je rečenica istinita za svaku vrijednost promjenljive. U gornjem primeru, voli (X, sladoled) je istinita za svaku vrednost X iz definisanog domena. **Egzistencijalni kvantifikator, $\exists$,** pokazuje da je rečenica istinita za neku vrijednost ili više vrijednosti domena.

PRIMJER:

drugovi (Y, petar) je istinita samo za neke objekte Y.

Definisanje rečenica u predikatskom računu

Svaka atomska rečenica je rečenica.

1. Ako je s rečenica, tada je i njena negacija rečenica, $\neg s$.
2. Ako su s_1 i s_2 rečenice, tada je i njihova konjukcija rečenica, $s_1 \wedge s_2$.
3. Ako su s_1 i s_2 rečenice, tada je i njihova disjunkcija rečenica, $s_1 \vee s_2$.
4. Ako su s_1 i s_2 rečenice, tada je i njihova implikacija rečenica, $s_1 \supset s_2$.
5. Ako su s_1 i s_2 rečenice, tada je i njihova ekvivalencija rečenica, $s_1 \equiv s_2$.
6. Ako je X promenljiva a s rečenica, tada je "X s rečenica".
7. Ako je X promenljiva a s rečenica, tada je $X s rečenica.

Interpretacijom u predikatskoj logici simboli konstanti, funkcija i relacija, dobijaju značenje elemenata nekog skupa, odnosno funkcija i relacija nad njima. Taj skup se naziva **domen**. Domen može biti sve što je potrebno: skup osoba, skup brojeva, ili nešto treće.

Definicija

Predikatski račun prvog reda dozvoljava kvantificirane promenljive, koje se odnose na objekte u domenu govora, a ne za predikate i funkcije.

PRIMJER:

" (Voli) Voli(george,kata).

nije pravilno formiran izraz u predikatorskom računu prvog reda. Postoji predikatorski računi višeg reda gde je takav oblik dozvoljen.

PRIMERI :

- Ako sutra ne pada kiša, Tom će ići u planine.
- $\neg$vrijeme(kiša, sutra) $\Rightarrow$ ide(tom, planine).
- Emma je doberman i dobar pas.
- dobarpas(emma) $\wedge$ je(emma, doberman).
- Svi košarkaši su visoki.

- $\forall$ X (košarkaši(X) $\Rightarrow$ visok(X)).
- Neki ljudi vole prokelj.
- $\exists$ X (ljudi(X) $\wedge$ voli(X, prokelj)).
- Kada bi želje bili konji, prosjaci bi jahali.
- jednako(želje, konji) $\Rightarrow$ jahati(prosjaci).
- Niko ne voli poreze.
- $\neg\exists$ X voli(X,porezi).

6.2.2. Pravilo zaključivanja modus ponens

Kao jednostavan primjer korištenja pravila *modus ponens* u predikatskom računu, koje preuzima na sebe tekuće opažanje: "ako pada kiša onda će zemlja biti mokra" odnosno "pada kiša". Ako je sa P označeno "pada kiša", a sa Q "zemlja je mokra" onda je prvi izraz postaje P $\Rightarrow$ Q. Pošto zaista pada kiša (P je istinito), skup aksioma postaje:

$$P \Rightarrow Q$$

$$P$$

Primjenom modus ponensa $(A\wedge(A\Rightarrow B))\Rightarrow B$, činjenica da je zemlja mokra (Q) može se pridodati skupu tačnih izraza.

Modus ponens se takođe može primjeniti na izraze koji sadrže promjenljive. Razmotrimo primjer prostog logičkog zaključka "svi ljudi su smrtni" i "Sokrat je čovjek"; zbog toga "Sokrat je smrtan". "Svi ljudi su smrtni" se može prikazati računom predikata kao:

"X (čovek(X) $\Rightarrow$ smrtan(X)).

"Sokrat je čovjek", kao: čovjek(sokrat).

Zbog toga što je X u izrazu univerzalno vrjednovano, može se zamjeniti svaka vrijednost u domenu za X i još uvijek postoji istinit iskaz unutar pravila zaključivanja univerzalna kvantifikacija. Mijenjanjem sokrat za X u izrazu, zaključuje se izraz:

čovek(sokrat) $\Rightarrow$ smrtan(sokrat).

Sada se može primjeniti modus ponens i izvesti zaključak smrtan(sokrat). Ovo je pridodano skupu izraza koji logički proizilaze iz originalnog iskaza. Algoritam nazvan *unifikacija* se koristi za automatizovano rješavanje problema tj. da odredi da li sokrat može biti ispravno zamjenjen za X u poretku na kom se primjenjuje modus ponens.

6.2.3. Primjer ES izgrađenog predikatskim računom

Predikatski račun omogućava rad i sa objektima i sa iskazima i predstavlja standardnu metodu predstavljanja znanja baziranu na matematičkoj logici. Predstavlja formalni logički jezik sa vlastitom sintaksom (strukturom izraza) i gramatikom, koji omogućava uvođenje relacija među objekte i primjenu mehanizma zaključivanja kod procesiranja znanja.

Programski jezik PROLOG je zasnovan na predikatskom računu i namjenjen je, prvenstveno, području vještačke inteligencije i ES. Postoji važna direktna veza i očigledan lanac: *prirodni jezik* $\rightarrow$ *predikatski račun* $\rightarrow$ *PROLOG*. Zahvaljujući ovoj karakteristici, predikatski račun dobija sve veći značaj kod razvoja ES, što je i prikazano u narednom primjeru.

PRIMJER:

PRIRODNI JEZIK	**PREDIKATSKI RAČUN**
1. *Ivo* i *Ana* su muž i žena	muž-i-žena(*Ivo,Ana*)
Objašnjenje:	*Relacija "muž i žena" postoji između objekata Ivo i Ana*
2. *Ana* živi u Baru	živi-u(*Ana,Bar*)
Objašnjenje:	*Relacija "živi u" postoji između objekata Ana i Bar.*
3. Ako je dvoje ljudi (X_1 i X_2) muž i žena, tada su vjenčani	muž-i-žena(X_1, X_2) $\rightarrow$ vjenčani (X_1, X_2)

Objašnjenje:	*Relacija "muž i žena" između bilo koje dvije osobe (X_1 i X_2) implicira (povlači za sobom ili →) relaciju "vjenčani" između danih osoba X_1 i X_2.*
4. Ako je dvoje ljudi X_3 i X_4 vjenčano i ako osoba X_4 živi u gradu X_5, tada i osoba X_3 također živi u gradu X_5.	vjenčani(X_3, X_4) & živi-u(X_4, X_5) → živi-u(X_3, X_5)
Objašnjenje:	*Relacija "vjenčani" između osoba X_3 i X_4 i relacija "živi u" između osobe X_4 i grada X_5 povlači za sobom da relacija "živi u" postoji također i između osobe X_3 i grada X_5.*

Vezivanje dva relaciona izraza imalo je posljedicu stvaranja sasvim nove relacije, odnosno zaključka. Drugim riječima, predikatski račun nizom relacija (istinitih ili lažnih) implicira relacije koje nisu bile formalno, za određeni objekat, unesene u sistem. Osim toga, u prvoj fazi se prelazi iz područja sasvim konkretnih objekata (ovdje osoba Ivo i Ana) na bilo koje apstraktne objekte (osobe X_1 i X_2), a zatim se ponovo vraća sa apstraktnih objekata na konkretne.

Mehanizam zaključivanja se odvija na slijedeći način:

1. Ivo i Ana su muž i žena.
2. Ana živi u Baru.
3. Ako su dvije osobe muž i žena, one su vjenčane.
4. Ako su dvije osobe vjenčane i ako jedna osoba živi u mjestu X, tada i druga osoba živi u mjestu X.

Ukupan zaključak: Ivo živi u Baru.

U predikatskom računu do istog zaključka se dolazi na slijedeći način:

1. muž-i-žena(Ivo,Ana)
   ↓ 2. živi-u(Ana,Bar)
3. muž-i-žena(X_1, X_2) → vjenčani(X_1, X_2)
   ↓ ↓
4. vjenčani(X_3, X_4) & živi-u(X_4, X_5) → živi-u(X_3, X_5)
   ↑

Ukupni zaključak: živi-u(Ivo,Bar)

Da je u razmatranju bio jedan jedini pogrešni zaključak (relacija između objekata neistinita), ukupni zaključak ne bi bio istinit, ali bi opet bio zaključak.

U ovom slučaju bi se dijalog između korisnika i ekspertnog sistema mogao odvijati na slijedeći način:

EXS: 1. Da li su Ivo i Ana muž i žena?
KOR: Da
EXS: 2. U kom mjestu živi Ana?
KOR: U Baru
EXS: 3. Ivo i Ana su vjenčani?
KOR: Da
EXS: 4. Tada i Ivo živi u Baru.

Sasvim je razumljivo da su i ovdje moguće greške. Postoji vjerovatnoća da je bračni par odvojen zbog posla, liječenja, izbijanja rata i slično. Međutim, ove greške nisu rezultat logičkog zaključivanja, već nepreciznosti relacije koja kaže da je uvijek istina da vjenčane osobe žive u istom mjestu.

7

U ovom poglavlju:

- ***Opšte o semantičkim mrežama***
- ***Primjena semantičkih mreža***
- ***Prednosti i nedostaci semantičkih mreža***
- ***Pravila za upotrebu semantičkih mreža***

SEMANTIČKE MREŽE

7.1. OPŠTE O SEMANTIČKIM MREŽAMA

Hijerarhije inkluzije se mogu lako generalisati dodavanjem novih tipova relacija, odnosno lukova i čvorova, u grafičku predstavu. Ovakva generalizacija se naziva semantičkom mrežom i prvi put je korištena za predstavljanje značenja rečenica u engleskom jeziku. Značaj ovakve predstave znanja je potenciran i činjenicom da neuronske interkonekcije u mozgu, vrlo vjerovatno čine neuronsku mrežu sličnu semantičkoj mreži. Koncept pristupa znanju putem "aktiviranja" čvorova u semantičkoj mreži kretanjem preko lukova, je na neki način analogan aktivnosti mozga putem prenosa električnih signala kroz neuronske mreže.

Semantičke mreže, najkraće rečeno, su oblik baza znanja kod kojih se objekti i relacije među njima prikazuju grafički. Međutim, treba dodati, da se uvijek nakon razrade ovakva mreža obrađuje na takav način da je računar može prihvatiti. To znači da treba primjeniti simboliku koja reprezentuje graf.

Semantičke mreže opisuju kategorije objekata (čvorovi u mreži) i relacije među njima (veza). Glavni oblik zaključivanja je nasleđivanje. Opisuju se prototipovi, a moguće je opisati i izuzetke (nemonotona logika).

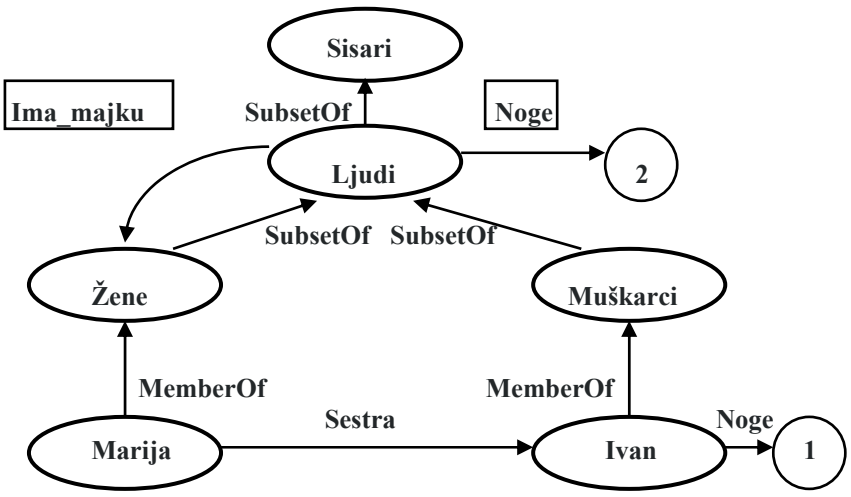
7.2. PRIMJENA SEMANTIČKIH MREŽA

Mogu se definisati slijedeći **tipovi veza** u semantičkim mrežama:

- *SubsetOf* (*a-kind-of*),
- *MemberOf* (*is-a*),
- Relacija između dva objekta (strjelica sa imenom relacije),
- Relacija izmedju svakog elementa klase A i objekta B (ime relacije uokvireno),
- Relacija izmedju svakog elementa klase A i nekog elementa klase B (ime relacije dvostruko uokvireno).

Implementacija semantičkih mreža:

- stvarne vrijednosti,
- *default* vrijednosti,
- procedure (*if needed, if added*).



7.1. Primjer semantičke mreže.

Prevođenje u logiku predikata

```
Ljudi ⊂ Sisari
Marija ∈ Žene
Noge(Ivan, 1)
∀x x ∈ Ljudi ⇒ Noge(x, 2)
∀x ∃y x ∈ Ljudi ⇒ y ∈ Žene ∧ Ima_majku(x, y)
```

Procedure se mogu dodati u mrežu, da bi se izračunale neke vrijednosti (*procedural attachment*).

Procedure za gradnju semantičkih mreža

- dodavanje i brisanje čvorova mreže,
- dodavanje i brisanje lukova mreže,
- pretraživanje,
- *if-added,*
- *if-needed*.

7.3. PREDNOSTI I NEDOSTACI SEMANTIČKIH MREŽA

Prednosti semantičkih mreža

Najveća prednost ovog modela podataka, odnosno ovakvog načina predstavljanja znanja, je njegova fleksibilnost, koja se sastoji u tome da se novi čvorovi i veze mogu dodavati po potrebi. Osim navedene prednosti, veoma bitna prednost je mogućnost naslijeđivanja osobina. Svaki čvor semantičke mreže, koji predstavlja jedinku neke klase, posjeduje osobine opštijih klasa čiji je član.

PRIMJER:

"Krila Jadrana" je brod, brod je plovno sredstvo, te je i "Krila Jadrana" plovno sredstvo.

Navedeno naslijeđivanje osobina smanjuje nepotrebnu redundansu, ali je nepogodno u slučaju izuzetka od pravila. U takvim slučajevima se vezama tipa "je izuzetak" mogu obuhvatiti pojedini izuzeci. Praktično preimućstvo semantičkih

mreža je veoma mala redundansa znanja, jer se svaki objekat, odnosno znanje vezano za njega, pojavljuje samo jednom u mreži. Ovim se postiže značajna ušteda u memorijskom prostoru računara, a povećava se integritet baze znanja.

Takođe, obzirom da su sve asocijacije između objekata eksplicitno povezane kao lukovi u mreži, i pretraživanje baze znanja je u principu vrlo efikasno. Ukoliko se mreža predstavi pomoću lista susjedstva (jedan od načina predstavljanja grafova u memoriji), a ne kao lista parova koji opisuju relacije u grafu, moguće je veoma brzo obaviti pretraživanje.

Osim navedenog, semantičke mreže su:

- pogodne za hijerarhijski uređene podatke,
- podražavaju način prirodne organizacije memorije,
- pogodne za unarne i binarne relacije.

Nedostaci semantičkih mreža

- teškoće pri predstavljanju relacija višeg stepena,
- disjunkcija i negacija se teško predstavljaju,
- teško se izvodi kvantifikacija.

Ranije se smatralo da su semantičke mreže efikasnije nego sistemi bazirani na matematičkoj logici, ali se danas smatraju ekvivalentnima.

Poređenje formalizama za predstavljanje znanja

1. **Ekspresivnost:** Matematička logika i produkcioni sistemi mogu da se koriste nezavisno. Semantičke mreže se obično koriste u hibridnim sistemima;

2. **Semantika:** Matematička logika ima jasnu semantiku. Produkcioni sistemi i mreže su ponekad nejasni;

3. **Izvršavanje:** Produkcioni sistemi su najjednostavniji.

Semantičke mreže su komplikove ako se koriste i procedure. Matematička logika zahtjeva komplikovane procedure za zaključivanje.

7.4. PRAVILA ZA UPOTREBU SEMANTIČKIH MREŽA

Iako pri upotrebi semantičkih mreža ne postoje posebna ograničenja, ipak je dobro se pridržavati određenih pravila.

(1) **Čvorovi** se upotrebljavaju za predstavljanje objekata i opisa:

- objekti mogu biti fizički objekti (plovno sredstvo, brod),
- objekti mogu biti konceptualni entiteti (događaj, ponašanje) ili apstraktne kategorije ("Krila Jadrana"),
- opisi omogućavaju dodatna objašnjanja o objektima (putnički, dodatno objašnjenje o brodu).

(2) **Veze** spajaju objekte i opise i mogu zamjeniti bilo koju relaciju. Najčešće su slijedeće veze:

- veza "JE" često se koristi za predstavljanje veze: klasa - primjerak klase.

PRIMJER:

"Krila Jadrana" je brod, a brod je plovno sredstvo. Prema tome, "Krila Jadrana" je predstavnik šire klase, klase brodova, dok je brod predstavnik još šire klase plovnih sredstava.

- veza "IMA" ukazuje na čvorove koji su u svojini drugih čvorova.

PRIMJER:

Plovno sredstvo ima posadu i kapetana, odnosno veza "IMA" pokazuje odnos dio - poddio.

(3) Veze mogu biti definicijske, kao npr. veza "VOZITI" između čvorova "čovjek" i "automobil".

(4) Određene veze sadrže heurističko znanje, čime obogaćuju mrežu dodatnim putevima.

PRIMJER: *Brod isplovljavajući oslobađa vez u luci.*

U ovom poglavlju:

- ***Opšte o ramovima znanja***
- ***Predstavljanje znanja ramovima znanja***
- ***Struktura ramova znanja***
- ***Generalna struktura ramova znanja***

8

RAMOVI ZNANJA

8.1. OPŠTE O RAMOVIMA ZNANJA

Razvoj semantičkih mreža doveo je do nastanka posebne logičke strukture, koja je nazvana ramovi znanja. Ramovi znanja (*FRAMES*) su kompletne i zaokružene logičke structure, nalik na klase u objektno orijentisanom programiranju, koje vezuje jedinstvo vremena, radnje i objekta, slično scenskim prikazima. Predstavljaju pogodno sredstvo za prikazivanje podataka i relacija.

Svaki ram znanja sačinjava nekoliko tipova informacija. Neke od tih informacija prikazuju kako se ram znanja koristi, dok druge govore šta treba da se desi ili šta raditi ako očekivanja nisu zadovoljena.

Takođe, za ramove znanja se može reći da predstavljaju kolekcije informacija i pravila koja se tiču datog objekta, situacije ili koncepta. Svaki ram znanja sadrži veći broj informacija, a baza znanja se sastoji od većeg broja ramova. Neki ramovi se stalno čuvaju u bazi znanja, dok se neki dinamički pojavljuju i nestaju tokom procesa rješavanja problema.

Ram znanja je mreža čvorova i relacija organizovanih hijerarhijski, gdje čvorovi na vrhu predstavljaju uopštene koncepte (objekte), a niži čvorovi specifične dijelove tih koncepata.

Strukturno, svaki ram znanja se sastoji od dva dijela i to:
- imena rama, i
- liste parova atribut - vrijednost.

Vizuelno, ram se često predstavlja kao imenovana kolekcija **pregradaka** (*SLOTS*), koji predstavljaju atribute, i njihovih **sadržaja** (*FILLERS*), koji nose informacije o datim objektima rama. Na ovaj način je objekat povezan sa skupom činjenica, pravila, procedura, podrazumjevanih (*DEFAULT*) vrijednosti, uputstava, koji ga u potpunosti određuju.

Samo ime "ram znanja" naglašava da se ovom konstrukcijom želi ograditi, uramiti dio znanja iz opšteg univerzuma znanja, odnosno grupisati činjenice i pravila koja su bitna za razmatranu oblast.

8.2. PREDSTAVLJANJE ZNANJA RAMOVIMA ZNANJA

Za predstavljanje znanja postoje dva komplementarna pristupa:
- proceduralni pristup, i
- deskriptivni pristup predstavljanju znanja.

Proceduralna predstava sadrži znanje o nekom objektu koje je dano eksplicitnim navođenjem skupova instrukcija, čija se tačnost mora provjeriti.

Deskriptivna predstava sadrži tvrdnje koje su istinite.

Kod ramova znanja zastupljena je proceduralna predstava u okruženju koje je deskriptivnog karaktera. Oba alternativna pristupa, u okviru jedne prezentacije znanja, doprinose svojim prednostima lakšoj upotrebi, odnosno lakšem održavanju baza znanja. Ova osobina ramova znanja znatno je doprinjela njihovoj sve većoj popularnosti. Za ramove znanja se može reći da posjeduju **dualnu semantiku**, što znači da se proceduralna i deskriptivna predstava može prevoditi jedna u drugu.

Grafički prikaz ramova znanja veže u jednu cjelinu nekoliko logičkih nivoa. Podaci su u ramu smješteni, uslovno rečeno, u neku vrstu pregratka, pri čemu pregradak uvijek sadrži isti tip informacija. Već i sama struktura rama predstavlja vrlo efikasan mehanizam zaključivanja, posebno u području eliminisanja pogrešnih zaključaka.

Korištenje ramova znanja se može objasniti vrlo jednostavnim primjerom.

***PRIMJER*:** *Termin plan sastanka, slika 8.1.*

Podaci za sastanak su: vrijeme, mjesto, tema i voditelj. Na osnovu ovako danih podataka može se formirati ram znanja pod nazivom SASTANAK, sa četiri pregratka (atributa): VRIJEME, MJESTO, TEMA I VODITELJ. U ramu su predviđene četiri vrste podataka, gdje je vrijeme dano kao cjelina zajedno sa datumom. Upisivanje bilo kojeg podatka koji ne spada u ram odmah se uočava.

SASTANAK

VRIJEME:	**11:00/22.09.2003.**
MJESTO:	**Podgorica**
TEMA:	**Ekspertni sistemi**
VODITELJ:	**Savo Savić**

Slika 8.1. Ram znanja sa četiri pregratka.

8.3. STRUKTURA RAMOVA ZNANJA

Ramovi znanja mogu biti:
- jednakog oblika a različitog sadržaja, i
- različitog oblika a istog sadržaja.

Neki pregratci mogu poslužiti kao veza sa drugim ramovima, što znači da se ramovi mogu vezivati u veće cjeline i na njima izvoditi logičke operacije. Kod ramova znanja se mogu ne samo efikasno "upisivati/brisati" pregratci, nego je moguće i njihovo poređenje, sortiranje, pretraživanje, uvijek vezano za dva ili više ramova.

Jedan ram je cjelina koja ne dopušta istovremeno poklapanje sa drugim ramom, nemoguće je da isti voditelj održava dva sastanka na dva mjesta u isto vrijeme.

Sadržaj pregradaka

Pregradak može da sadrži:
- ime drugog rama,
- listu drugih ramova, i/ili
- listu osobina koje se vezuju za pregradak, tj. vrijednosti atributa koje se vezuju za atribut.

Osim toga, pregradak može da sadrži:
- ograničenja koja moraju da zadovolje vrijednosti tog pregratka,
- procedure (**metode**) koje se koriste da se odredi vrijednost pregratka, ako je ta vrijednost potrebna. Ove procedure se nazivaju "**demoni**",
- procedure koje se izvršavaju kad u dati pregradak stigne neka vrijednost. Ove procedure se nazivaju "**aktivne vrijednosti**".

Pregratci pridruženi objektu rama mogu imati **podrazumjevane** (*DEFAULT*) **vrijednosti**, koje se prihvataju ukoliko nisu u suprotnosti sa ostalim informacijama pohranjenim u posmatranom objektu.

PRIMJER: *Podrazumjevana vrijednost za boju snijega je bijela, ukoliko nije došlo do nekih ekoloških poremećaja, što se posebno navodi.*

Ovakve vrijednosti su naročito pogodne u oblastima gdje rijetko dolazi do izuzetaka od pravila.

Relacije inkluzije i ramovi znanja

Ramove znanja je moguće grupisati u hijerarhiju inkluzija, jednostavnim dodavanjem **atributa ISA**, koji za svoju vrijednost ima ram koji je nadklasa danom ramu. Time je omogućeno da se primjeni koncept naslijeđivanja i postigne veća ekonomičnost i bolja organizovanost predstavljanja znanja.

8.4. GENERALNA STRUKTURA RAMOVA ZNANJA

Generalna struktura ramova znanja, predstavljena kao kompleksna linearna lista, ima slijedeći izgled:

```
RAM: ram_1
     pregradak_1: (vrijednost:   )
                  (ISA: ram_2)
                  (DEMON: metoda_2)
                  (DEFAULT: metoda_1)

      pregradak_2: ista struktura

                  .
                  .
                  .

     Metoda_1: pravila_znanja_1

     Metoda_2: algoritam_1

                  .
                  .
                  .
```

Za ram znanja se može dati slijedeći opšti izraz:

$$\{r; (p_1, v_1), (p_2, v_2), \ldots , (p_n, v_n)\},$$

gdje je:

r - ime rama znanja,
p - ime pregratka (atributa),
v - vrijednost pregratka (atributa).

PRIMJER: *Ram znanja MUZIČKI STUB, sa konkretnim podacima, slika 8.2.*

Pregradak KOMPAKT DISK ima pridružena uputstva za uzimanje informacija iz drugih pregradaka (pregratci NAZIV PROIZVOĐAČA i GODINA PROIZVODNJE), dok pregradak MINIMALNA JAČINA ZVUKA sadrži uputstva za korištenje informacija iz drugog rama znanja sa imenom POJAČALO AA-230.

MUZIČKI STUB	
PREGRATCI	**VRIJEDNOST**
(1) VLASNIK:	(1) Ana Tot
(2) NAZIV PROIZVOĐAČA:	(2) ITT
(3) GODINA PROIZVODNJE:	(3) 2003.
(4) VRSTA PRIJEMNIKA:	(4) LM/MW/FM stereo sintisajzer FM- 230
(5) VRSTA POJAČALA:	(5) Integrisani stereo AA-230
(6) GRAMOFON:	(6) DEFAULT: DA
(7) KASETOFON:	(7) DEFAULT: DA
(8) KOMPAKT DISK:	(8) Ako je potreban ovaj podatak, naći godinu proizvodnje i naziv proizvođača i uporediti sa tabelom X (u tabel X su za svakog proizvođača dani tipovi u koje je ugrađen CD i godina od koje se CD ugrađivao u sve modele)
(9) MINIMALNA JAČINA ZVUKA:	(9) Ako je potreban ovaj podatak, pronaći jačinu pojačala AA-230 i uporediti sa tabelom Y (tabela Y sadrži za određene vrste pojačala minimalnu dozvoljenu jačinu zvuka)

Slika 8.2. Primjer rama znanja MUZIČKI STUB.

Koncept naslijeđivanja i ramovi znanja

Kako pregradak može ukazivati na drugi ram znanja, navođenjem imena tog rama i naziva veze koja ih povezuje, omogućava se naslijeđivanje osobina jednog objekta od drugog.

Semantičke mreže i ramovi znanja

Ramovi znanja mogu da se posmatraju kao specifični slučajevi semantičkih mreža. Isto tako oni mogu predstavljati njihove dijelove, odnosno pojedini čvorovi i veze u semantičkoj mreži mogu biti predstavljeni putem ramova znanja.

Kako između semantičkih mreža i ramova znanja postoji visok stepen srodnosti, jedni te isti podaci u bazi znanja mogu biti predstavljeni na bilo koji od ova dva načina.

U ovom poglavlju:

- ***Uvod u fazi logiku***
- ***Formiranje fazi skupova***
- ***Predstavljanje fazi skupova***
- ***Ograničenja fazi skupova***
- ***Operacije fazi skupova***
- ***Izvedeni fazi skupovi***
- ***Fazi zaključivanje***
- ***Pravilo više pretpostavki***
- ***Razvoj fazi ekspertnog sisrema***
- ***Primjer fazi ekspertnog sistema***

9

FAZI EKSPERTNI SISTEMI

9.1. UVOD U FAZI LOGIKU

Eksperti se često oslanjaju na uopštene pojmove i uopštena razmišljanja u procesu rješavanja pojedinih problema. U takvim slučajevima eksperti opisuju probleme koristeći neodređene ili višemislene termine.

PRIMJER: *Ekspertov iskaz: "Kada je motor zagrijan, brzina mu neznatno opada", stvara poteškoću u interpretaciji i korištenju ovakvih i sličnih nejasnih ili nedovoljno određenih iskaza.*

Pohraniti u računar ovakav način razmišljanja predstavlja veliki izazov. Postavlja se pitanje kako na računaru predstaviti i kako odlučivati na osnovu višeznačnih pojmova. Ovakvi, i slični problemi, više ili manje uspješno se rješavaju uz pomoć fazi logike *(fuzzy logic)*.

U narednom tekstu će biti razmatran kratki istorijat fazi logike, njezine osnovne ideje, kao i matematička realizacija fazi logike. Takođe, biće razmatrana primjena fazi logike za relizaciju ekspertnih sistema.

Materijal izložen u tačkama 9.1. i 9.3. najvećim dijelom se zasniva na knjizi Durkin, J., "Expert Systems", 1994.

Fazi sistemi su predloženi od poljskog naučnika Lukaševića (*Lukasiewich*) 1920. godine. Lukašević je izučavao matematički prikaz "*fuzzy*" pojmova, kao što su: visok, star, topao, itd. Njegovo pručavanje je proisteklo iz tumačenja da su svi iskazi proistekli sa dvije vrijednosti [0,1], služeći se prostom Aristotelovom (*Aristotelian*) logikom: tačno ili netačno (*true or false*). Lukašević je proširio sistem logike, proširujući oblast tačnih vrijednosti za sve realne brojeve u intervalu 0 i 1. Koristeći brojeve iz ovog intervala, u predstavljanju vjerovatnoće, da li je razmatrani izraz tačan ili netačan.

PRIMJER: *Vjerovatnoći da je osoba visoka 2 m može se reći "zaista visoka" i dodjeliti vrijednost 0,9. Činjenica je da je osoba "zaista visoka".*

Ovakav pristup nameće formalnoj tehnici razmišljanja o tačnosti i zove se **teorija vjerovatnoće**.

Naučnik *Zadeh* je 1965. godine nastavio rad na teoriji vjerovatnoće, dajući joj formalni oblik matematičke logike. Nije nebitno da je *Zadeh* prvi obratio pažnju inženjera i naučnika na prikupljanje važnih koncepata iz oblasti prirodnog jezika, koji su do tada bili isključivo predmet izučavanja lingvista. Ovaj novi logički alat je omogućio predstavljanje i rad sa "fazi" terminima, dovodeći do nove naučne discipline - fazi logike.

Definicija fazi logike

Grana logike koja koristi stepen članstva (učešća, pripadanja) elementa skupu, prije tačno/ netačno definisanog članstva.

Lingvističke promjenljive

Fazi logika je orijentisana na kvantifikovanje i razmatranje neodređenosti, ili tzv. fazi termina, koji se javljaju u prirodnom jeziku. U fazi logici ovi fazi termini se zovu **lingvističke promjenljive**.

TABELA 9.1.

Lingvističke promjenljive	Tipične vrijednosti
temperatura	toplo, hladno
visina	nizak, srednji, visok
brzina	puzeći, sporo, brzo

Definicija lingvističke promjenljive (fazi vrijednosti)

Izrazi prirodnog jezika, koji opisuju koncepte sa neodređenim (višeznačnim, nejasnim) vrijednostima, su lingvističke promjenljive ili fazi vrijednosti.

PRIMJER: *Iskaz "Ivo je mlad" implicira promjenljivu dob i ima lingvističku vrijednost "mlad".*

U TABELI 9.1. su dani još neki primjeri.

U fazi ES, lingvističke promjenljive se koriste prema fazi pravilima. Fazi pravila daju informacije o lingvističkim promjenljivim na osnovu zaključaka dobijenih na osnovu informacija o promjenljivim, sadržanim u pretpostavkama.

PRIMJER:

Pravilo 1

```
IF        brzina mala
THEN      učini brzinu većom
```

Pravilo 2

```
IF        temperatura mala
AND       pritisak je na sredini
THEN      učini brzinu veoma malom
```

Oblast mogućih vrijednosti lingvističkih promjenljivih se zove **oblast promjenljivih prirodnog jezika**.

PRIMJER: *Zadan je lingvističkoj promjenljivoj "brzina", iz Pravila 1, interval vrijednosti između 0 i 100 km/h. Izraz "brzina je mala" obuhvata određeni interval iz prirodnog jezika.*

Navedena vrijednost u prethodnom primjeru se zove **fazi skupom**.

Fazi skupovi

Tradicionalna teorija skupova posmatra svijet u crno - bijelim pogledima, odnosno objekat se nalazi ili ne nalazi u razmatranom skupu.

PRIMJER:

a) *Posmatra se skup sastavljen od ljudi, mladih ljudi i djece. Tradicionalna teorija skupova postavlja jasnu granicu za ovaj skup i dodjeljuje svakom članu skupa vrijednost 1 , a članovima koji nisu unutar skupa vrijednost 0. Ovakav skup se zove* ***izdvojeni skup****.*

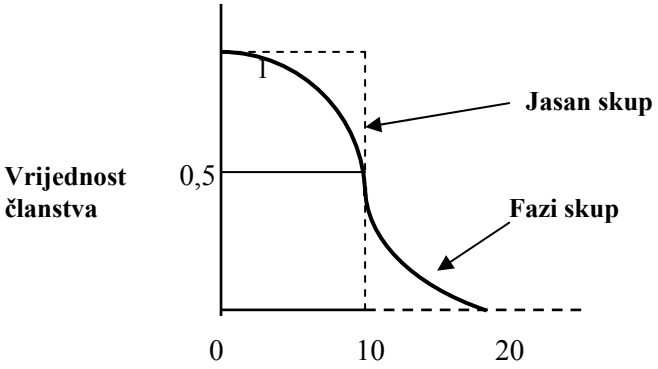
b) *Članovi skupa mogu biti djeca do 10 godina. Koristeći strogo tumačenje, djeca sa 11 godina imaju "isteklu pripadnost".*

Fazi logika omogućava znatno pravilniju interpretaciju pojma "mladi ljudi". Fazi skup označava vrijednost pripadanja intervalu između 0 i 1, koji označava pripadnost članstvu asociranim sa skupom.

PRIMJER: *Ako osoba ima 5 godina, može se označiti vrijednost pripadanja sa 0,9, ali ako ima 13 godina vrijednost pripadanja može biti 0,1.*

U navedenim primjerima "dob" je lingvistička promjenljiva, a "mlad" je fazi skup. Ostali fazi skupovi mogu biti "star", "srednja_dob", itd. Svaki od ovih fazi skupova predstavlja određeno svojstvo, koje je definisano lingvističkom promjenljivom.

Uopšteno, fazi skup obezbjeđuje značajan prelaz preko "granice", što je ilustrovano slikom. 9.1. Osa x, ili univerzalnost govora, predstavlja dob osobe. Osa y predstavlja vrijednost pripadanja fazi skupu. Fazi skup za "mlade" osobe pokriva odgovarajuću vrijednost pripadanja. Sa slike se vidi da je 11-godišnjak skoro bio dijete, i kako se dob osoba povećava one se postepeno odmiču iz ove klasifikacije.



Slika 9.1. Fazi i jasan skup "mladih" ljudi.

Definicija fazi skupa

Neka je X (skup) univerzalnog govora, sa elementima označenim sa x. Fazi skup A iz X karakteriše **funkcija pripadanja $\mu_A(x)$**, koja je posebno povezana sa svakim elementom x stepenom vrijednosti pripadanja skupu A.

Suprotno teoriji vjerovatnoće, koja se nalazi u osnovi označavanja vjerovatnoće svakog danog događaja, a na osnovu prethodne frekvencije događaja, fazi logika predstavlja vrijednost za događaj na osnovu funkcije pripadanja, koja je definisana kao:

$$\mu_A(x) : X \rightarrow [0, 1] \tag{9.1}$$

U fazi logici, događaju ili elementu x, se funkcijom pripadanja μ dodjeljuje vrijednost pripadanja. Ova vrijednost predstavlja stepen sa kojim element x pripada fazi skupu A.

$$\mu_A(x) = stepen\ (x \in A) \tag{9.2}$$

Vrijednost pripadanja za x je ograničena slijedećim izrazom: $\boldsymbol{0 \leq \mu_A(x) \leq 1.}$

Fazi skupovi su proširenje tradicionalne teorije skupova. U generalizaciji koncepata pripadanja je korištena funkcija članstva μ, koja uzima vrijednosti između 0 i 1, što odražava stepen (vrijednost) pripadnosti objekta (elementa x) skupu A.

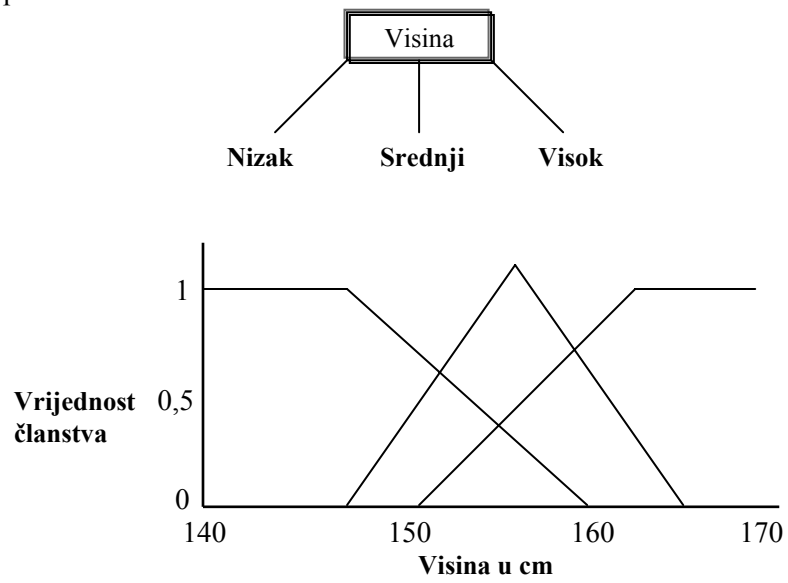
9.1.1. Formiranje fazi skupova

Da bi se fazi skup prikazao pomoću računara, neophodno je definisati njegove funkcije pripadanja. Jedan od načina je mišljenje intervjuisanih osoba, da se na osnovu njihovog shvatanja termina pokuša predstaviti fazi skup.

PRIMJER:
Razmatra se koncept visoke osobe. Intervjuišu se osobe i traži se da se odrede da sa kojim stepenom uvjerenja za neku osobu kažu da je visoka. Nakon dobijanja odgovora za visinu, može se naći srednja vrijednost, formirajući na taj način skup visokih osoba. Nakon toga se može odrediti stepen uvijerenja - vrijednost pripadanja za svaku osobu koja je pripala fazi skupu "visokih" osoba.

Navedeno intervjuisanje se može primjeniti i kod "niskih" i "srednjih" osoba.

Navedeni primjer se može ilustrovati slikom 9.2, gdje su fazi skupovi prikazani linearno za sve tri kategorije individualnih visina. Kada se definiše više fazi skupova kod univerzalnosti govora, tada se oni u literaturi nazivaju fazi podskupovi.



Slika 9.2. Fazi skupovi visina.

Visina od 155 cm je za skup "srednje visokih" osoba sa stepenom uvjerenja od 1, a u isto vrijeme za osobu iz skupa "niskih" ili "visokih" osoba sa stepenom uvjerenja 0,25.

Drugi pristup, koji se sreće u praksi, zasniva se na realizaciji stavova eksperata. Kao i u prethodnoj tehnici intervjuisanja, može se pitati ekspert o njegovom uvjerenju koji objekti pripadaju kojem skupu. Kod ovog pristupa nedostaje stav većeg broja osoba, međutim taj nedostatak je nadomješten ekspertovim shvatanjem problema.

Treći pristup, koji se rjeđe primjenjuje, opisao je *Kosko* (1992). Ovaj postupak se zasniva na primjeni neuronskih mreža, gdje neuronske mreže koriste podatke sistemskih operacija iz procesa obučavanja, a u obliku fazi skupa.

9.1.2. Predstavljanje fazi skupova

U prethodnom razmatranju razmatrani su fazi skupovi primjenom prirodnog jezika. U narednom tekstu biće primjenjeno formalno predstavljanje fazi skupova. Pretpostavka je da je univerzalnim govorom definisan skup X i nad njim definisan fazi skup A. Takođe, pretpostavka je da postoji diskretni skup sa elementima $\{x_1, x_2, \ldots, x_n\}$. Fazi skup A definiše funkciju pripadanja $\mu_A(x)$, koja određuje vrijednosti pripadanja elemenata x_i, iz skupa X, u intervalu (0, 1), jednačina (9.1).

Vrijednost pripadanja određuje do kojeg stepena x_i pripada skupu A, jednačina (9.2). Za diskretni skup elemenata, prikladan način predstavljanja fazi skupa je pomoću vektora:

$$A = (a_1, a_2, \ldots, a_n) \tag{9.3}$$

gdje je:

$$a_i = \mu_A(x_i) \tag{9.4}$$

Za jasnije predstavljanje, vektor često uključuje simbol "/", koji je vezan za vrijednost pripadanja a_i sa koordinatom x_i:

$$A = (a_1/x_1, a_2/x_2, \ldots, a_n/x_n) \tag{9.5}$$

PRIMJER:
Biće razmotren fazi skup za "visoke" osobe, prethodno razmatran na slici 9.2:

VISOK = (0/150; 0,25/ 155; 0,7/ 160; 1/ 165; 1/ 170)

Standardni fazi skup predstavlja uniju vektora prema odgovarajućim dimenzijama, gdje "+" predstavlja Bulovu (*Boolean*) notaciju za unije:

$$A = \mu_1/x_1 + \mu_2/x_2 + \ldots + \mu_n/x_n \tag{9.6}$$

ili

$$A = \sum_{i=1}^{n} \mu_i / x_i$$

Ako je X kontinualna funkcija, onda se skup A može predstaviti kao:

$$A = \int_x \mu_A(x_i) / x_i dx \tag{9.7}$$

Za kontinualni skup elemenata, traži se neka funkcija povezivanja između elemenata i njihovih vrijednosti pripadanja. Tipične funkcije koje se koriste su *sigmoid* (slika 9.1), Gausova (*Gaussian*) ili *pi*. Ove funkcije su monotone i mogu da obezbjede blisko predstavljanje podataka, koji su osnova fazi skupa. Međutim, ove funkcije uzrokuju dodatna izračunavanja podataka u računaru.

U praksi, većina aplikacija se nalazi na linearnom dijelu funkcije za predstavljanje fazi skupa, što je prikazano slikom 9.2. Za predstavljanje linearnog dijela, svaki fazi skup se može kodirati u odgovarajući vektor.

PRIMJER: *Može se kodirati fazi skup "visoki", slika 9.2, vektorom (0/150; 0,25/155; 0,7/160; 1/165; 1/ 170).*

Srednje područje fazi skupova, kao što je npr. "srednji" na slici 9.2. su, obično, predstavljeni odgovarajućom trougaonom funkcijom, kao (0/145; 0,5/150; 1/155; 0,5/160; 0,6/165).

Fazi logika je, često, podržana softverskim alatima, koji omogućavaju kodirane trougaone funkcije da koriste kao vektor kodiran krajnjom i srednjom tačkom.

PRIMJER*: Može se kodiati "srednji" fazi skup sa slike 9.2, koristeći vektor oblika (0/145; 1/155; 0/165).*

9.1.3. Ograničenja fazi skupova

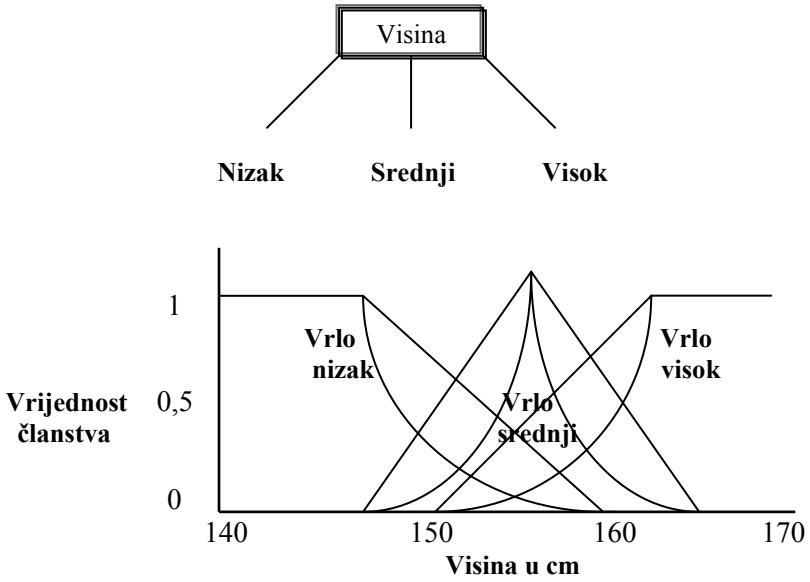
U prethodnim tačkama su opisane metode za formiranje i predstavljanje lingvističkih neodređenosti (nejasnoća) kvantitativnim terminima pri upotrebi fazi skupova. U normalnoj konverzaciji mogu se pridodati dodatne neodređenosti za određeni izraz, koristeći priloge kao što su: "vrlo", "neznatno", "određeni dio" , itd. Prilozi su riječi koji modifikuju glagole, pridjeve, druge priloge ili cijele rečenice.

PRIMJER:

Može se razmotriti prilog koji modifikuje pridjev "Osoba je vrlo visoka". Ako se želi predstaviti ovaj novi fazi skup, koristi se ista grupa osoba da se izjašnjava o visokim osobama, Međutim, u radu sa ovim novim fazi skupom mora se obuhvatiti dodatni prilog "vrlo". Upravo, ovo je ograničenje.

Ograničenja matematički modifikuju postojeći fazi skup, uzimajući u obzir dodatni prilog. Na slici 9.3. su prikazana tri fazi skupa visina, ranije predstavljena slikom 9.2, koja su uzela u obzir termin "vrlo".

Za ilustraciju uticaja modifikacija fazi skupa, uzimajući u obzir granične vrijednosti, može se uzeti u razmatranje osoba visoka 160 cm. U skladu sa slikom 9.3, ova osoba se može razmatrati sa vrijednosti pripadanja 0,7. Međutim, kako je prikazano na slici 9.3 ova osoba se može smatrati vrlo visokom sa vrijednosti pripadanja 0,4. U narednom tekstu će se razmatrati granice koje se koriste u praksi.



Slika 9.3. Fazi skup visina sa "vrlo" ograničenjem.

Koncentracija

Operacija koncentracije utiče na buduće redukcije vrijednosti pripadanja članstva članovima koji imaju niske vrijednosti pripadanja. Ova operacija se definiše:

$$\mu_{CON(A)}(x) = (\mu_A(x))^2 \qquad (9.8)$$

Dobijeni fazi skup "visokih" osoba može se koristiti u kreiranju skupa "vrlo visokih" osoba.

Širenje

Operacija širenja širi fazi elemente povećanjem vrijednosti pripadanja onih elemenata sa malom vrijednošću pripadanja, više nego elemenata sa većom vrijednšću. Ova operacija se definiše:

$$\mu_{DIL(A)}(x) = (\mu_A(x))^{0,5} \tag{9.9}$$

Dani fazi skup je za "srednje" osobe. Može koristiti ovu operaciju i u kreiranju skupa za "više" ili "manje srednje" osobe.

Pojačanje

Operacija pojačanja povećava vrijednost pripadanja u izrazima gdje je ona iznad 0,5, a smanjuje tamo gdje je ispod 0,5. Ova operacija se definiše:

$$\begin{aligned} \mu_{INT(A)}(x) &= 2(\mu_A(x))^2 && \text{za } 0 \le \mu_A(x) \le 0,5 \\ &= 1 - 2(1 - \mu_A(x))^2 && \text{za } 0,5 \le \mu_A(x) \le 1 \end{aligned} \tag{9.10}$$

Za zadani fazi skup "srednjih" osoba, mogu se koristiti ove operacije u kreiranju skupa "sigurno srednjih" osoba.

Kapacitet

Operacija kapaciteta predstavlja proširenje operacije koncentracije:

$$\mu_{POW(A)}(x) = (\mu_A(x))^n \tag{9.11}$$

Zadan je fazi skup "visokih" osoba. Može se koristiti ova operacija za n = 3 u cilju generisanja skupa "sigurno vrlo visokih" osoba.

9.1.4. Operacije fazi skupa

Presjek

U klasičnoj teoriji skupova, presjek dva skupa sadrži one elemente koji su zajednički za oba skupa. U fazi skupovima, element može biti djelimično u oba skupa. Prema tome, kada se razmatra presjek dva skupa, ne može se reći da je element više u presjeku, nego kod nekog od originalnih skupova. Na osnovu navedenog, fazi operacija za kreiranje presjeka dva fazi skupa A i B, definisane nad X, glasi:

$$\mu_{A\wedge B}(X) = \min(\mu_A(x), \mu_B(x)) \qquad \text{za sve } x \in X$$

$$= \mu_A(x) \wedge \mu_B(x) \tag{9.12}$$

$$= \mu_A(x) \cap \mu_B(x)$$

Simbol $\wedge$ (označava logičko "I" ("AND")) se koristi u fazi logici za predstavljanje "min" operacije, odnosno uzimanje minimalne vrijednosti pod određenim uslovima.

Za ilustraciju ove operacije se može razmotriti fazi skup visokih i niskih osoba:

VISOK = (0/150; 0,2/155; 0,5/160; 0,8/165; 1/170)
NIZAK = (1/150; 0,8/155; 0,5/160; 0,2/165; 0/170)

Uzimajući u obzir jednčinu 9.12, presjek ova dva skupa je:

$$\mu_{VISOK\wedge NIZAK}(x) = (0/150; 0{,}2/155; 0{,}5/160; 0{,}2/165; 0/170)$$

Kada se razmotre termini "visok" i "nizak", korisno je analizirati šta u tom slučaju znači "srednji". Za ovaj termin očekuje se da veća vrijednost pripadanja bude u sredini skupa, a manja na granicama skupa.

Navedeni primjer ilustruje kako dva fazi skupa mogu biti kombinovana za formiranje novog skupa. Jezički termin, koji se može primjeniti za ovaj novi skup, glasi "srednje visoke" osobe.

Unija

Drugi način kombinovanja skupova je preko njihove unije. Unija dva skupa uključuje one elemente koji pripadaju jednom ili obema skupovima. U ovom slučaju vrijednost članova ne može biti manja od vrijednosti članova u nekom od skupova. U fazi logici se koristi slijedeći oblik jednačine za uniju skupova A i B:

$$\begin{aligned} \mu_{A\vee B}(x) &= \max(\mu_A(x), \mu_B(x)) \qquad \text{za svaki } x \in X \\ &= \mu_A(x) \vee \mu_B(x)) \\ &= \mu_A(x) \cup \mu_B(x) \end{aligned} \tag{9.13}$$

Simbol $\vee$ (označava logičko "ILI" ("OR")) se koristi u fazi logici za predstavljanje "max" operacije, odnosno uzimanje maksimalne vrijednosti pod određenim uslovima.

Za ilustraciju ove operacije se može razmotriti fazi skup visokih i niskih osoba:

VISOK = (0/150; 0,2/155; 0,5/160; 0,8/165; 1/170)
NIZAK = (1/150; 0,8/155; 0,5/160; 0,2/165; 0/170)

Uzimajući u obzir jednčinu 9.13, presjek ova dva skupa je:

$$\mu_{VISOK\vee NIZAK}(x) = (1/150; 0{,}8/155; 0{,}5/160; 0{,}8/165; 1/170)$$

Rezultat predstavlja uniju članova, postižući najveću vrijednost pripadanja na granicama a najmanju vrijednost na sredini skupa. Jezički termin ovog novog skupa glasi "ne srednji".

Komplement (not)

Komplement fazi skupa A je ~A i može se predstaviti slijedećom operacijom:

$$\mu_{\sim A}(x) = 1 - \mu_A(x) \tag{9.14}$$

Zadan je fazi skup "visokih" osoba. Slijedeće operacije se mogu koristiti za predstavljanje skupova "ne visokih" osoba, "srednjih" osoba ili "niskih" osoba:

$\mu_A(x)$ = VISOK = (0/150; 0,2/155; 0,5/160; 0,8/165; 1/170)
$\mu_{\sim A}(x)$ = NE VISOK = (1/150; 0,8/155; 0,5/160; 0,2/165; 0/170)

9.1.5. Izvedeni fazi skupovi

Koristeći ograničenja i operacije fazi skupova, iz postojećih fazi skupova se može izvesti mnoštvo drugih. Pretpostavka je da postoji fazi skup A "visokih" osoba. Može se izvesti fazi skup B "ne vrlo visokih" osoba slijedećom operacijom:

$$\mu_B(x) = 1 - (\mu_A(x))^2$$

Ovaj pristup se može proširiti na slijedeći način. Pretpostavka je da postoji fazi skup A "visokih" osoba i fazi skup B "niskih" osoba. Može se izvesti fazi skup C "ne vrlo visokih" osoba i "ne vrlo niskih" osoba slijedećom operacijom:

$$\mu_C(x) = [1 - (\mu_A(x))^2] \wedge [1 - (\mu_B(x))^2]$$

Uopšteno, mogu se koristiti fazi operacije i ograničenja u izvođenju fazi skupova, koji predstavljaju različite jezičke opise i kombinacije, u izrazima koji se koriste u prirodnom jeziku.

9.1.6. Fazi zaključivanje

Fazi logika razmatra skup kao fazi pravila. Fazi pravila su izrazi koji se odnose na određene jezičke promjenljive, kao što su "visina mu je - visoka". Uopšteno, fazi prijedlog se može predstaviti kao:

```
Prijedlog: X je A
```

gdje je A fazi skup univerzalnog govora X.

Fazi pravila relacija za dva fazi prijedloga dobija oblik:

```
IF X je A   THEN   Y je B
```

Navedeno pravilo određuje vezu ili spajanje dva pravila.

Fazi ES skladišti pravila kao fazi spajanje. To znači, za pravilo *IF A THEN B*, gdje su *A* i *B* fazi skupovi, fazi ES skladišti spajanje (*A, B*) u matricu *M* (često imenovanu u literaturi kao *R,* oznaka za veze). Fazi matrica spajanja *M* raspoređuje fazi skup *A* na fazi skup *B*. Ovo fazi spajanje ili fazi pravilo nazvano je **fazi**

memorija spajanja (FAM). FAM raspoređuje fazi skup *A* na fazi skup *B*, što je nazvano **procesom fazi zaključivanja**.

Kao i ostale neegzaktne tehnike rasuđivanja, korištene u dizajniranju ES, fazi zaključivanje pokušava da uspostavi povjerenje u pravila zaključivanja, dana pravilima pretpostavki. Međutim, dok su pravila sadržana u fazi pravilima fazi skupova, fazi logika mora raspoređivati predpostavljeni skup informacija iz zaključenog skupa informacija. Da bi se ovo ostvarilo, izvodi se fazi zaključivanje realizovano fazi skupom iz informacija vezanih za fazi skup.

Može se izvršiti ilustracija slijedećim primjerom: razmatra se fazi skup A - "Visina mu je - visok" i fazi skup B - "Težina mu je - težak", vezano sa pravilom IF A THEN B:

```
IF Visina is visok THEN težina is težak
```

Mogu se predstaviti obe veličine *A* i B kao odgovarajući vektori i njima pripadajuće relacije u fazi matrici *M*.

U praksi se može koristiti navedeno pravilo za formiranje stepena uvjerenja da neke osobe sa određenom težinom su teške. Fazi zaključivanje obavlja se uzimajući raspoložive informacije kodirane u A' (podskup od A) i uključivanjem fazi skupa B' (podskup od B), koji kvantitativno obuhvataju ovo uvjerenje. Za izvođenje navedenog fazi skupa, fazi zaključivanje počiva na radu sa fazi vektor-matricom.

Dvije najpopularnije fazi tehnike zaključivanja, koje se koriste u praksi, su: **max - min zaključivanje** i **max - ishod zaključivanje**. Da bi se objasnile ove tehnike, potrebno je objasniti rad sa fazi vektor - matricom.

9.1.6.1. Rad sa fazi vektor - matricama

Osnovni rad vektor - matrica omogućava izvođenje vektora *y* za zadani vektor *x* i matricu *A*:

$$\underset{1\times n}{x} \cdot \underset{n\times p}{A} = \underset{1\times p}{y}$$

$$y_j = \sum_{i=1}^{n} x_i a_{ij}$$

Pri radu sa fazi vektor - matricom, se koristi tehnika poznata kao max - min komponovanje, koja je definisana operatorom komponovanja "o". Ovaj operator izvršava max - min operaciju nad zadanim vektorom i matricom. Operacije su slične klasičnim vektor - matrica operacijama, s tim što se množenje parova zamjenjuje maksimumima parova minimuma kolona (redova).

Ova operacija će biti primjenjena na fazi pravilo ili FAM `IF A THEN B`, gdje je *A* fazi skup definisan nad *X,* a *B* fazi skup definisan nad *Y*. Za grubo izračunavanje vektora *A* i *B,* predstavljenih kao:

$$A = (a_1, a_2, \ldots, a_n); \qquad a_i = \mu_A(x_i)$$
$$B = (b_1, b_2, \ldots, b_n); \qquad b_i = \mu_B(x_i)$$

može se definisati fazi matrica *M* tako da je:

$$A \circ M = B \tag{9.15}$$

i izračunati komponenta b_j pomoću:

$$b_j = \max_{1 \le i \le n} \{ \min(a_i, m_{i,j}) \} \tag{9.16}$$

Za ilustraciju će biti predstavljen fazi skup A = (.2.4.61), a fazi matrica *M* je:

$$M = \begin{vmatrix} .1 & .6 & .8 \\ .6 & .8 & .6 \\ .8 & .6 & .5 \\ 0 & .5 & .5 \end{vmatrix}$$

Iz jednačine 9.16 se izračunava fazi skup *B*:

b_1 = max{min(.2,.1), min(.4,.6), min(.6,.8), min(1, 0)
= max{.1,.4,.6, 0}
= 0.6
b_2 = max{.2,.4,.6,.5}
= 0.6
b_3 = max{.2,.4,.5,.5}
= 0.5

9.1.6.2. Pristup fazi zaključivanju

Za razumjevanje savremenog pristupa zaključivanju, korisno je napraviti pregled ranijih istraživanja, u prvom redu originalni rad *Zadeh*-a (1965). Posmatrao je fazi skup kao distribuciju funkcije vjerovatnoće. Ova funkcija raspoređuje elemente nekog univerzalnog skupa u brojeve između 0 i 1, što odražava stepen uvijerenja da neki element pripada određenom fazi skupu.

$$\begin{aligned} A &= \text{distribucija vjerovatnoće} \\ &= \mu_A(x) \\ &= \Pi_A \end{aligned}$$

Vođeno je računa o krajnjim informacijama fazi skupa *B*, dobijenim iz informacija fazi skupa *A*, koji je u relaciji sa fazi skupom *B*. Ovaj pristup je sličan klasičnoj teoriji uslova vjerovatnoće, kod koga je kompozicioni operator korišten za klasične operacije vektor - matrica. Traženi su uslovi distribucije vjerovatnoće matricom $\Pi_{B/A}$, koji komponovani sa distribucijom vjerovatnoće za fazi skup A daju distribuciju vjerovatnoće i za fazi skup B:

$$\Pi_A \circ \Pi_{B/A} = \Pi_B$$

gdje je Π_A vektor *1 x n*, $\Pi_{B/A}$ matrica *n x p* i Π_B vektor *1 x p*.

Koristeći ovaj pristup, uvođenjem informacija za *A* (označene *A′*), dobijaju se informacije za *B* (označene *B′*). Ova tehnika je nazvana **kompoziciona pravila zaključivanja**.

Postavlja se naredni zadatak, a to je kako oformiti $\Pi_{B/A}$ distribucionu matricu. Ovo je rješeno na slijedeći način: izvršena je interpretacija komponente od $\Pi_{B/A}$ matrice kao parovi implikacija fazi skupova *A* i *B*.

PRIMJER:
Navedeni fazi skup u vektorski odgovarajućem obliku, može se prikazati:

$$\Pi_{B/A} = \begin{vmatrix} a_1 \to b_1 & a_1 \to b_2 & . & . & . & . \\ a_2 \to b_1 & & & & & \\ . & & & & & \\ . & & & & & \\ . & & & & & \end{vmatrix}$$

Prikazana matrica je ista kao i ranije objašnjena fazi matrica spajanja *M*. U narednoj tački su opisana dva najčešće korištena fazi zaključivanja, koja se koriste u dizajniranju fazi ES.

9.1.6.3. Max - min zaključivanje

U max - min zaključivanju, kao implikacioni operator, se koristi min i dobija se:

$$m_{ij} = truth(a_i \rightarrow b_j) = min(a_i, b_j) \tag{9.17}$$

Koristeći dva fazi skupa *A* i *B*, se može koristiti jednačina (9.17) za dobijanje matrice *M*. Takođe, može se koristiti jednačina (9.16) za određivanje vektora *B´* iz podskupa *A´* skupa *A*.

PRIMJER:

Predpostavka je postojanje univerzalnog govora definisanog sa X, koji predstavlja "temperaturu", i fazi skup A, definisan nad X, koji predstavlja "normalnu temperaturu". Takođe se predpostavlja univerzalni govor definisan nad Y, koji predstavlja "brzinu" i fazi skup B, definisan na Y, koji predstavlja "srednju brzinu". Pretpostavka je da se raspolaže slijedećim fazi pravilom:

```
IF temperatura je normalna THEN brzina je srednja
```

ili

```
IF A THEN B
```

Radi jasnoće, fazi skupovi se predstavljaju vektorima, a vektor elementi se predstavljaju njihovim korespondirajućim domenskim vrijednostima:

Normalna temperatura = (0/100, .5/125, 1/150, .5/175, .0/200)
Srednja brzina = (0/10, .6/20, 1/30, .6/40, 0/50)

Pristupa se oformljenju matrice *M*, prema jednačini (9.17):

$$M = m_{ij} = min(a_i, b_j) =$$

$$= \begin{vmatrix} min(0.,0.) & min(0.,.6) & min(0.,1.) & min(0.,6) & min(0.,0.) \\ min(.5,0.) & min(.5,.6) & min(.5,1.) & min(.5,.6) & min(.5,0.) \\ min(1.,0.) & min(1.,.6) & min(1.,1.) & min(1.,.6) & min(1.,0.) \\ min(.5,0.) & min(.5,.6) & min(.5,1.) & min(.5,.6) & min(.5,0.) \\ min(0.,0.) & min(0.,.6) & min(0.,1.) & min(0.,.6) & min(0.,0,) \end{vmatrix}$$

$$= \begin{vmatrix} 0. & 0. & 0. & 0. & 0. \\ 0. & 0.5 & 0.5 & 0.5 & 0. \\ 0. & 0.6 & 1. & 0.6 & 0. \\ 0. & 0.5 & 0.5 & 0.5 & 0 \\ 0. & 0. & 0. & 0. & 0. \end{vmatrix}$$

Pretpostavka je da podskup A' ima slijedeću vrijednost:

$A' =$ (0/100, .5/125, 0/150, 0/175, 0/200)

Ovaj podskup predstavlja očitavanje temperature od 125 ^{o}C. Mjerenja raspoređuju vrijednosti pripadanja od 0.5 za fazi skup "normalnih temperatura", što ukazuje na skup B' (uvjeranja u B), čije se vrijednosti žele odrediti.

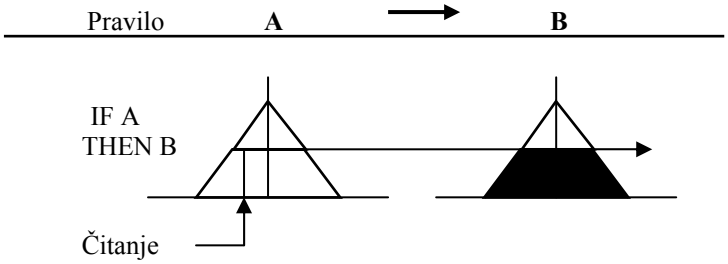
Za A' (gornja vrijednost), koristeći min - max kompoziciju (jednačina 9.16), se dobija:

$$b_j = \max_{1 \le i \le n} \{\min(a_i', m_{ij})\}$$

$b_1 = max[min((0.,0.), min(.5,0.), min(0.,0.), min(0.,0.),min(0.,0.)]$
$b_2 = max[min((0.,0.), min(.5,.5), min(0.,.6), min(0.,.5),min(0.,0.)]$
$b_3 = max[min((0.,0.), min(.5,.5), min(0.,1.), min(0.,.5),min(0.,0.)]$
$b_4 = max[min((0.,0.), min(.5,.5), min(0.,.6), min(0.,.5),min(0.,0.)]$
$b_5 = max[min((0.,0.), min(.5,0.), min(0.,0.), min(0.,0.),min(0.,0.)]$
$B' = (0/10, .5/20,.5/30, .5/40, 0/50)$

Ovo uzrokuje da je fazi skup odsječena verzija B, čija veličina je postavljena preko A'. Može se reći da je ovo uopštena posljedica max - min zaključivanja, što je i ilustrovano slikom 9.4, za trougaono oblikovane fazi skupove. Šta se uobičajeno radi sa navedenim skupom, biće kasnije razmatrano u sklopu teme defazifikacije.

Ključna stvar koja se uočava na navedenom primjeru je rezultat dobijen limitiranjem A' za jednostruku vrijednost. Drugim riječima, tvrđenje da je očitana temperatura 125 ^{o}C daje A' podešeni vektor (0 .5 0 0 0), šta za rezultat daje B' (0 .5 .5 .5 0).



Slika 9.4. Max - min zaključivanje.

U realnim aplikacijama fazi logičkih sistema, za koje postoje vrijednosti mjerenja (x_k = 125 ^oC) sa jednom mjernom vrijednošću x_k, može se koristiti $\mu_A(x_k)$ neposredno za fazi skup predstavljen sa B, sa oznakom $\mu_B(y)$, u dobijanju prouzrokovanog skupa B′:

$$B' = \mu_A(x_k) \wedge \mu_B(y) \tag{9.18}$$

PRIMJER: *U ranije navedenom primjeru, gdje je predpostavljena temperatura od 125 ^oC, slijedi da je* $\mu_A = 0.5$ *i*

B′ = [min(.5,0), min(.5,.6), min(.5,1), min(.5,.6), min(.5,0)] = (0,.5,.5,.5,0)

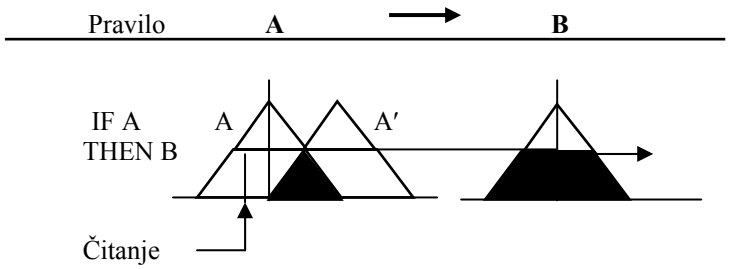
Isti rezultat je dobijen računajući sa fazi matricom spajanja. Slijedi zaključak, kada su ulazne informacije u nejasnoj formi, ne mogu se izračunavati i preračunavati fazi matrice, ali se može (sa stanovišta računara) raditi sa blažim zahtjevima informacija za fazi skup.

U slučaju ulaza sa pravilom predstavljanja fazi očitavanja, može se koristiti jednostavniji pristup. Može se razmotriti pravilo IF A THEN B, a fazi čitanje A imenovati kao A′. Takođe, može se uzeti presjek dva ulaza, $\min(a'_i, a_i)$, za navedeni fazi skup B′. Ovakav pristup je ilustrovan na slici 9.5.

9.1.6.4. Max – proizvod zaključivanje

Max – proizvod zaključivanje se koristi za standardne proizvode, kao operator pri formiranju komponenti za M:

$$m_{ij} = a_i b_j \tag{9.19}$$



Slika 9.5. Min – max zaključivanje za fazi ulaz.

Kod izračunavanja elemenata ove matrice, max – min kompozicija je korištena za određivanje matrice zaključaka B′, od nekog podskupa A′. Kao ilustracija može poslužiti primjer vektora, posmatranog u prethodnim tačkama:

A = (0, .5, 1, .5, 0) B = (0, .6, 1, .6, 0)

$$M = \begin{vmatrix} (0.\cdot,0.) & (0.\cdot.6) & (0.\cdot1.) & (0.\cdot.6) & (0.\cdot0.) \\ (.5\cdot0.) & (.5\cdot.6) & (.5\cdot1.) & (.5\cdot.6) & (.5\cdot0.) \\ (1.\cdot0.) & (1.\cdot.6) & (1.\cdot1.) & (1.\cdot.6) & (1.\cdot0.) \\ (.5\cdot0.) & (.5\cdot.6) & (.5\cdot1.) & (.5\cdot.6) & (.5\cdot0.) \\ (0.\cdot0.) & (0.\cdot.6) & (0.\cdot1.) & (0.\cdot.6) & (0.\cdot0,) \end{vmatrix} = \begin{vmatrix} 0. & 0. & 0. & 0. & 0. \\ 0. & 0.3 & 0.5 & 0.3 & 0. \\ 0. & 0.6 & 1. & 0.6 & 0. \\ 0. & 0.3 & 0.5 & 0.3 & 0. \\ 0. & 0. & 0. & 0. & 0. \end{vmatrix}$$

Ako se uzme A′ = (0, .5, 0, 0, 0), nakon max – min kompozicije se dobija:

$$b_j = \max_{1 \le i \le n} \{\min(a_i', m_{ij})\}$$

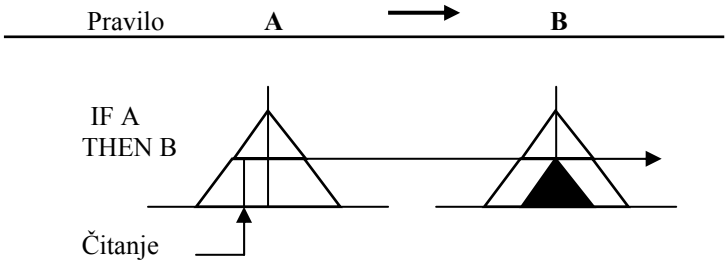
b_1 = *max[min((0.,0.), min(.5,0.), min(0.,0.), min(0.,0.),min(0.,0.)]*
b_2 = *max[min((0.,0.), min(.5,.3), min(0.,.6), min(0.,.5),min(0.,0.)]*
b_3 = *max[min((0.,0.), min(.5,.6), min(0.,1.), min(0.,.5),min(0.,0.)]*
b_4 = *max[min((0.,0.), min(.5,.3), min(0.,.6), min(0.,.5),min(0.,0.)]*
b_5 = *max[min((0.,0.), min(.5,0.), min(0.,0.), min(0.,0.),min(0.,0.)]*
B′= (0, .3, .5, .3, 0)

Tehnika zaključivanja max - proizvod je skalarna verzija za B. Slika 9.6 ilustruje uopštene rezultate upotrebe ove metode za trougaoni fazi skup.

Za primjer se uzima slučaj korišten u prethodnoj tački, A′ je odabrani vektor koji sadrži jedinstvenu vrijednost. Zbog navedenog ograničenja, koje se susreće u praksi, može se zaključiti da je jednostavnije izračunati B′. Ako mjerenje za A iznosi x_k, može se usvojiti vrijednost za B′:

$$B' = \mu_A(x_k)\ \mu_B(y) \tag{9.20}$$

Za razmatrani primjer je: $B' = 0.5 \cdot (0, .6, 1, .6, 0) = (0, .3, .5, .3, 0)$



Slika 9.6. Max – proizvod zaključivanje.

9.1.7. Pravilo više pretpostavki

Do sada su razmatrana fazi pravila koja sadrže po jednu pretpostavku za pravilo IF A THEN B. Međutim, često će se javiti potreba rada po pravilima više pretpostavki: (IF A AND B THEN C). Potrebno je za ovako pravilo oformiti matricu spajanja M.

Jedna od preporuka je praktične prirode. Predpostavlja se da je fazi skup A definisan nad X, skup B nad Y, C i Z. Pristup se sastoji u pojedinačnom definisanju matrice M za svaku od pretpostavki, što vodi do pretpostavljenih zaključaka (M_{AC} i M_{BC}). Nakon toga se pretpostavkama dovodi neka ulazna informacija, A′ i B′. Traženi fazi skup C se izračunava nezavisno pomoću sistema jednačina:

$$A' \circ M_{AC} = C_{A'} \tag{9.21}$$
$$B' \circ M_{BC} = C_{B'} \tag{9.22}$$

Slijedeći korak je rekomponovanje prethodnih skupova iz (9.21) i (9.22). Pristup zavisi da li su pretpostavke spojene konjunktivno "AND" ili disjunktivno "OR". Za pretpostavke spojene sa "AND", fazi logički presjek je korišten za navedene fazi skupove:

$$C' = [A' \circ M_{AC}] \wedge [B' \circ M_{BC}] = C_{A'} \wedge C_{B'} \quad (9.23)$$

U tabeli 9.2 su prikazane vrijednosti za Pravilo više pretpostavki:

Tabela 9.2.

C′	Združivanje pretpostavki	Zaključak
$\text{Min}(a_i, b_j) \wedge \mu_c(z)$	AND	max – min
$\text{Min}(a_i, b_j) \wedge \mu_c(z)$	OR	max – min
$\text{Min}(a_i, b_j) \cdot \mu_c(z)$	AND	max – ishod
$\text{Min}(a_i, b_j) \cdot \mu_c(z)$	OR	max – ishod

dok je operator unije korišten za OR združene pretpostavke:

$$C' = [A' \circ M_{AC}] \vee [B' \circ M_{BC}] = C_{A'} \vee C_{B'} \quad (9.24)$$

Ovaj proces može biti pojednostavljen, kada se unose nove vrijednosti. Može se razmotriti slučaj jedinstvene vrijednosti A, definisane sa x_k, i jedinstvene vrijednosti B, definisane se y_j. Dodjeljujući vrijednosti pripadanja fazi skupu $a_i = \mu_A(x_k)$ i $b_j = \mu_B(y_j)$, tabela 9.2, omogućava proste pristupe u izračunavanju C′ za pravilo više pretpostavki.

9.1.7.1. Defazifikacija

Razmotriće se dvije tehnike za određivanje vrijednosti: uvođenjem fazi skupa B′ i procjenom skupa A definisanog kao A′. U mnogim aplikacijama potrebno je uzeti B′ i prihvatiti nove vrijednosti. Ranije je bio razmotren primjer fazi pravila:

IF Temperatura je normalna THEN Brzina je srednja

i korišteno mjerenje temperatura za navedeni fazi skup pri "normalnoj brzini". U kontrolnim aplikacijama se može ukazati potreba da se znaju specifične brzine, koje treba da se koriste. Za ovo je potrebno uzeti navedeni fazi skup i prihvatiti nove vrijednosti, što je zadatak tehnike **defazifikacije**.

Najpopularnija defazifikaciona tehnika se koristi u **fazi metodi centriranja** i obezbjeđuje jedinstvenu vrijednost y_i iz B′ na slijedeći način:

$$y_i = \frac{\sum_{j=1}^{p} y_j m_{B'}(y_j)}{\sum_{j=1}^{p} m_{B'}(y_j)} \tag{9.25}$$

Ova se tehnika može iskoristiti u razmatranom primjeru, sa jedinstvenom vrijednosti brzine. Kada se koristi sa jednim pravilom dobija se za rezultat ista vrijednost: centrirano B. Defazifikacija je pogodna kada više pravila vode istom zaključku za razmatrani događaj.

9.1.7.2. Višestruka fazi pravila

Sada će biti razmatrani slučajevi sa *n* fazi pravila ili asocijacijama (A_1,B_1), ... , (A_n,B_n). Ovi slučajevi zahtjevaju *n* matrica M_1, ... , M_2 za kodiranje asocijacija ili veza između A_i i B_i. Zahtjeva se od ovakvog skupa fazi pravila dobijanje rezultata vjerovanjem u B, na osnovu jedinstvenih mjerenja A′. Vrši se paralelno prikupljanje A′ za bazu pravila, stvarajući naznačeni fazi skup B_i' za svako pravilo. Nakon toga se vrši sumiranje B_i' skupova u cilju formiranja rezultujućeg mješovitog fazi skupa B′. Pri tome se koristi standardni skup operacija unija, gdje je B definisano na domenu X:

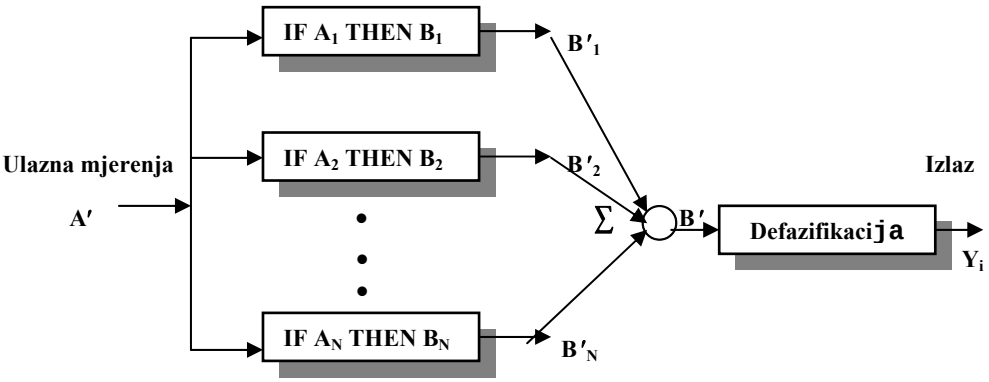
$$\begin{aligned} B' &= B'_1 \cup B'_2 \cup \ldots B'_{n-1} \cup B'_n \\ &= \max (B'_1(x), B'_2(x), \ldots , B'_{n-1}(x)) \quad \text{za sve } x \in X \end{aligned} \tag{9.26}$$

Prateći ove operacije unija, može se defazifikovati rezultat B′, koristeći metodu centriranja. Ovaj proces stvara novi izlaz vrijednostu y_i. Ova operacija je ilustrovana na slici 9.7. Rezultat je da ova operacija kao ulaz uzima vrijednosti za A, a stvara ocjenu za B.

U cilju proširenja ove ideje, može se pretpostaviti da se želi kontrolisati brzina nekog vozila. Slijedeća pretpostavka je da se odluka bazira na temperaturi vozila i pritisku, što se može predstaviti slijedećim pravilima:

IF Temperatura je normalna
OR Pritisak je nizak
THEN Brzina je srednja

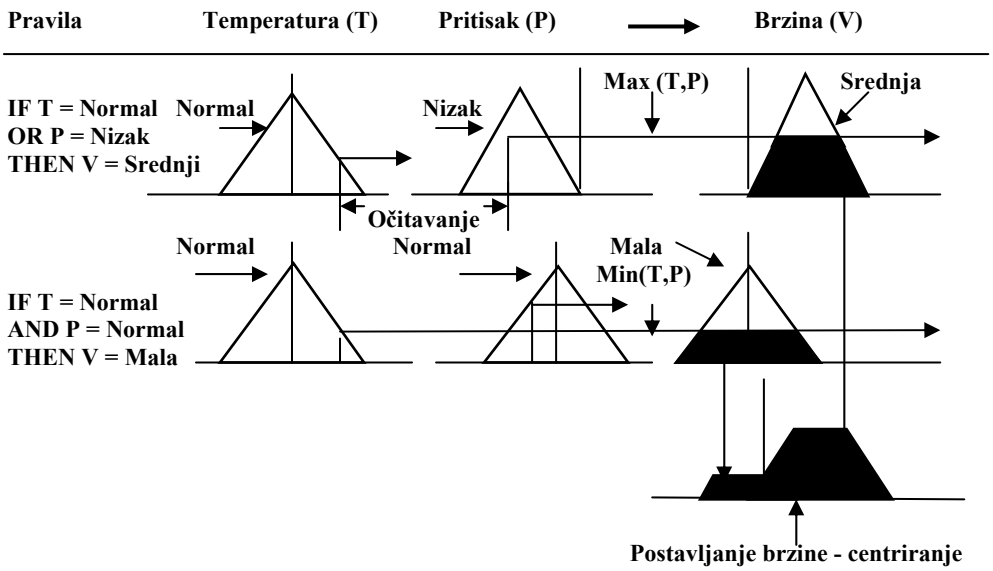
IF Temperatura je normalna
AND Pritisak je normalan
THEN Brzina je niska



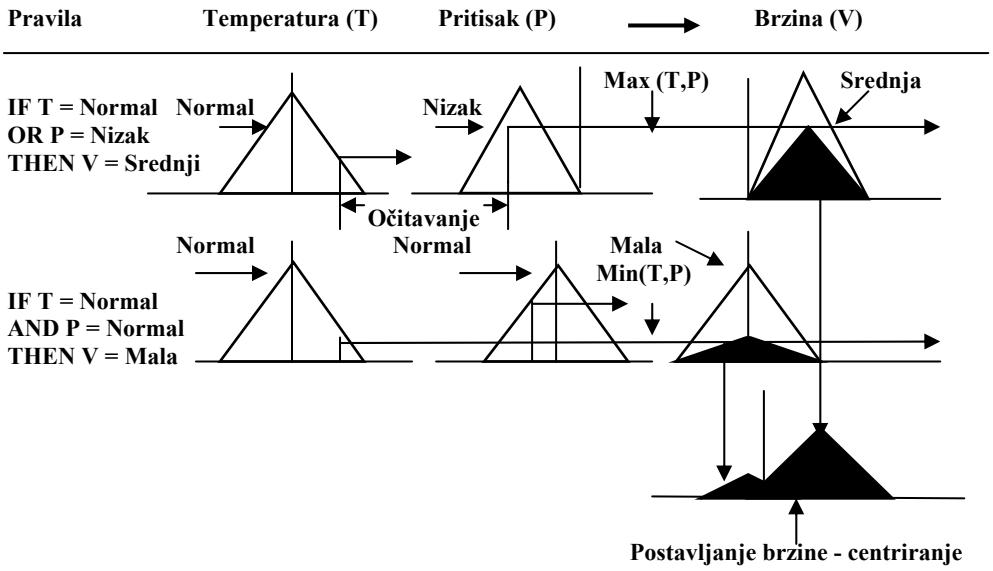
Slika 9.7. Arhitektura sistema fazi pravila.

Slika 9.8. ilustruje proces max – min zaključivanja za dva pravila, dok slika 9.9. prikazuje proces max – proizvod zaključivanje. Čitanje temperature i pritiska se koristi za određivanje njihove korespondirajuće vrijednosti pripadanja u prikazanim fazi skupovima. Bira se najveća vrijednost prema disjunktivnim pravilima, a najmanja prema konjunktivnim pravilima. Nakon toga ove vrijednosti se zatim koriste bilo za cijele novozaključene fazi skupove (max – min zaključivanje) ili za određivanje njihovog stepena (max – proizvod zaključivanje). Navedeni fazi skupovi brzina su zatim sumirani (fazi unija) i određen je centralni rezultat. Centralni rezultat obezbjeđuje željenu vrijednost brzine.

Uopšteno, može se primjeniti proces prikazan na slici 9.7. za bilo koji broj fazi pravila i sa bilo kojim brojem uslova fazi promjenljivih.



Slika 9.8. Max – min zaključivanje za više pravila.



Slika 9.9. Proces max – proizvod zaključivanje.

9.2. RAZVOJ FAZI EKSPERTNIH SISTEMA

9.2.1. Fazi ekspertni sistemi

Fazi ekspertni sistemi omogućavaju uključivanje jezičkih, rasplinutih, sudova u formalne opise sistema. Pokazali su se pogodni za modeliranje složenih, ili u formalnom smislu nedovoljno dobro definisanih problema, te su potencijalni izbor pri modeliranju nelinearnih sistema.

Jedna od osnovnih vrsta su ES zasnovani na pravilima. Takvi ES se zasnivaju na skupu P = {P_i}, i = 1, 2, ... , n, gdje je n broj fazi "AKO ... $\rightarrow$ TADA" pravila, koji čine fazi bazu znanja i što se formalno može prikazati:

$$P_i \equiv Ako\ X\ je\ A_i\ tada\ Y\ je\ B_i,\ i = 1, 2, \ldots, n,$$

gdje su A_i i B_i fazi skupovi, tj. vrijednosti lingvističke promjenljive.

Imajući u vidu da fazi logika predstavlja jedan sasvim drugačiji pristup u projektovanju sistema upravljanja, pa tako i ES, u daljem izlaganju se pošlo od pretpostavke da je neophodno da se sažeto izlože i obrazlože najvažniji elementi teorije upravljanja, što je u krajnjoj liniji i zadatak ES. Razvoj ovih sistema je objašnjen preko razvoja fazi regulatora (*controllers*), korištenjem jednog jednostavnog ilustrativnog primjera. Korištenje ovog jednostavnog primjera treba da doprinese lakšem razumjevanju relativno složenih fazi relacija.

9.2.2. Uvod u upravljanje

Zadatak upravljanja nekim procesom ili sistemom, izražava se zahtjevom da se ponašanje uskladi sa unaprijed zadanim referentnim ponašanjem. Uopšteno, ovaj se problem rješava tako što se prikupe svi raspoloživi podaci o samom sistemu i na osnovu njih, i iskustva iz rješavanja sličnih problema, odabira niz upravljačkih akcija, koje se, eventualno, koriguju preko povratne sprege.

Teorija upravljanja, i pored brojnih različitih rješenja, se u osnovi zasniva na tri osnovne pretpostavke.

- Sistem kojim se upravlja mora biti poznat, šta znači da se može predvidjeti njegov odziv na zadani ulaz. Na osnovu poznavanja sistema, formira se njegov matematički model;

U tački 9.2. je prikazana skraćena verzija primjera iz rada Turajlić, R. S., "Fazi sistemi i fazi upravljanje", 1996. i rada Šaletić, Z. D. i dr., "Rasplinuti sistemi i rasplinuti ES zasnovani na pravilima", 2000.

- Cilj upravljanja mora da bude jasno definisan preko sažetih matematičkih relacija, koje neposredno povezuju promjenljive sistema sa ocjenom njegovih performansi;
- Ponašanje sistema se prati na osnovu mjerenja njegovih izlaza.

Ukoliko su ove pretpostavke ispunjene, struktura i parametri regulatora se mogu odrediti jednom od mnogobrojnih metoda klasične i savremene teorije upravljanja. U praksi je, međutim, situacija često sasvim drugačija. U prvom redu, mehanizam rada sistema je često nedovoljno poznat, a čak i kad to nije slučaj, zahtjev da matematički model bude dovoljno jednostavan za rješavanje, spriječava uključivanje u model svih nelinearnosti ili nestacionarnosti sistema.

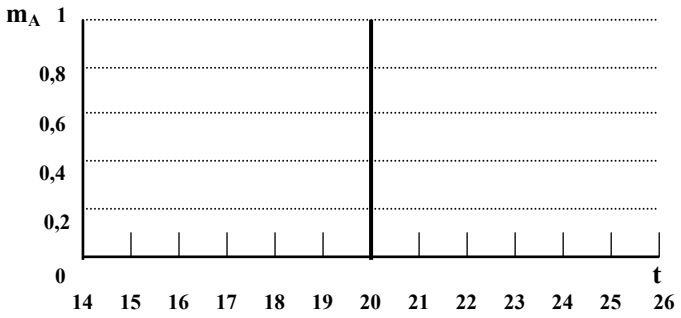
Pored navedenog, cilj upravljanja se, najčešće, iskazuje u obliku "zadovoljavajućeg ponašanja", "male potrošnje energije", "velikog profita" i njegova formulacija obično predstavlja kompromis između stvarnih zahtjeva i jednostavnosti rješenja koje se realizuje. Konačno, mjerenja na procesu su uvijek nedovoljno precizna, što znači da ni ocjena performansi sistema, kao ni korekcija koja se ostvaruje kroz povratnu spregu, ne mora biti adekvatna.

Ovdje treba istaći i činjenicu da su, u pokušaju da se prevaziđu navedeni problemi, vršena intenzivna istraživanja karakteristika ponašanja operatora pri upravljanju procesom. Svi rezultati su ukazivali da se operator ponaša kao izrazito nelinearan regulator, čiji su parametri vremenski promjenljive funkcije. Rješenja navedenih problema su tražena u povećanoj složenosti modela, koji će obuhvatati sve aspekte ponašanja sistema, ili u drugačijim algoritmima upravljanja, koji će biti robusni u odnosu na nepreciznost modela.

9.2.3. Fazi skupovi

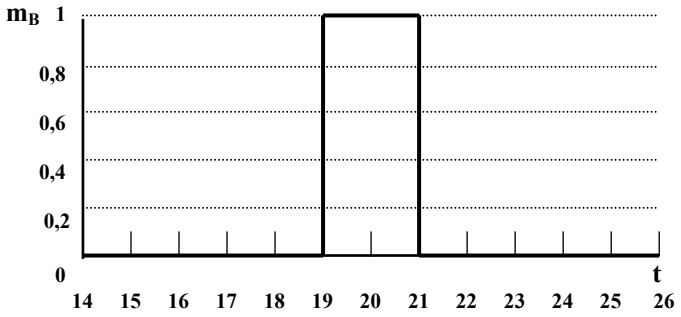
Objašnjenje svih koraka u projektovanju fazi regulatora biće ilustrovano jednostavnim ilustrativnim primjerom, koji treba da olakša razumjevanje relativno složenih fazi relacija. Pretpostavka je da je potrebno projektovati sistem upravljanja koji treba da održava **prijatnu temperaturu**. Realizacija ovakvog sistema podrazumjeva da se, kao prvo, precizira značenje pojma prijatna temperatura i da se on izrazi u okviru nekog matematičkog formalizma. Usvojeni formalizam će se, kasnije, koristiti u projektovanju sistema i ocjeni njegovih performansi.

Najbolje rješenje navedenog problema je da se, kroz razgovor sa osobama koje treba da koriste sistem koji se projektuje, definiše šta oni podrazumjevaju kao prijatna temperatura. Pretpostavka je da se kao rezultat ispitivanja dobije odgovor da je to temperatura od 20 ºC. Na ovaj način se dobija skup promjenljivih A, koji predstavlja prijatnu temperaturu, čija je karakteristična funkcija $m_A(t)$ binarna i može se predstaviti dijagramom na slici 9.10.



Slika 9.10. Prikaz prijatne temperature - referantni skup je u tački.

U najjednostavnijem slučaju, algoritam upravljanja ovakvim sistemom bi predstavljao sekvencu grijanja i hlađenja sistema, koja bi se realizovala u zavisnosti od izmjerene vrijednosti signala greške, imajući u vidu veliku inertnost termičkih procesa, kao i uticaj poremećaja. Izvjesno je da bi ovakvo upravljanje sadržavalo veliki broj uključivanja i isključivanja grijača, što bi nesumljivo nepovoljno uticalo na vijek njegove ispravnosti.

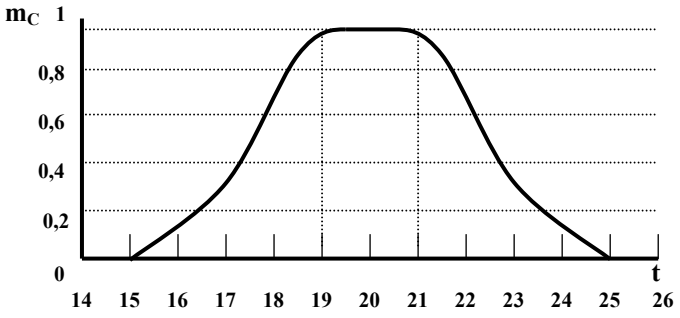


Slika 9.11. Prikaz prijatne temperature - referantni skup je u intervalu.

U cilju smanjivanja broja uključivanja i isključivanja grijača, moglo bi se postaviti pitanje da li pojam prijatna temperatura obuhvata stvarno samo 20 ºC i nijednu drugu vrijednost. U ponovljenom razgovoru sa potencijalnim korisnicima sistema, vjerovatno bi se moglo ustanoviti da oni ne insistiraju baš da temperatura bude 20 ºC i da su spremni da prihvate da se ona kreće između 19 ºC i 21 ºC. Na taj način se došlo do drugog skupa *B*, koji takođe označava pojam prijatna temperatura, a čija je karakteristična funkcija $m_B(t)$ takođe binarna, slika 9.11.

Sistem upravljanja, realizovan na osnovu ovakvog proširenog skupa, realizovao bi isti algoritam kao i prethodni sistem, samo bi obzirom na ublažavanje postavljenih zahtjeva broj uključivanja grijača bio manji, čime bi se produžio vijek trajanja sistema. Postavlja se pitanje da li je neophodno ovako "oštro" upravljanje sistemom. Odgovor je svakako negativan. Sistemom se može upravljati u zatvorenoj sprezi, pomoću bilo kog kompenzatora, uz uslov da postoji odgovarajući izvršni organ koji omogućava kontinualnu promjenu snage grijača.

Problem se i dalje javlja u definiciji referentnog pojma. Na osnovu slike 9.11., tvrdnja da je temperatura od 19 ºC u potpunosti prijatna, a od 18,99 ºC u potpunosti neprijatna, nema mnogo smisla. Izvjesno je da bi se svi potencijalni korisnici sistema složili kako temperatura od 18 ºC ili 22 ºC nije baš prijatna, ali se u nekom periodu može prihvatiti, da je temperatura 17 ºC ili 23 ºC manje prihvatljiva, ali još uvijek podnošljiva, dok se temperatura manja od 15 ºC ili veća od 25 ºC ne može izdržati.



Slika 9.12. Prikaz prijatne temperature - referantni skup je fazi interval.

Ova posljednja lingvistička definicija pojma prijatna temperatura predstavlja problem za klasičnu teoriju upravljanja, zato što ona ne uključuje matematički

formalizam za njeno predstavljanje kao referentnog skupa. Ovakav iskaz se više ne može opisati binarnim skupom koji je namjenjen odgovorima "da - ne" tipa. Dani iskaz zahtjeva kontinualnu **multivalentnu logiku**, koja bi omogućila opis svih stanja između "apsolutnog da" i "apsolutnog ne". Rješenje je da se ovako definisani referentni zahtjevi opišu pomoću **fazi skupova**, kojima se proširuje osnovni pojam karakteristične funkcije. To se ostvaruje tako što se uključuje stepen pripadnosti, koji pokazuje mjeru u kojoj se neka vrijednost slaže sa pridruženim pojmom. **Funkcija pripadnosti** $m_C(t)$ fazi skupa C, koji predstavlja pojam prijatna temperatura, prikazana je na slici 9.12.

Ovakva izmjena definicije referentnog skupa zahtjeva drugačiji pristup projektovanju algoritma upravljanja, o čemu će biti govora.

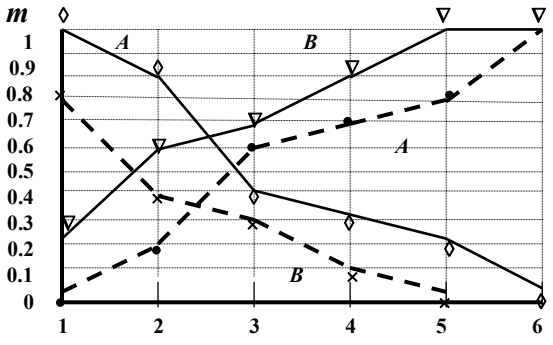
9.2.4. Fazi relacije nad fazi skupovima

Uvođenje pojma fazi skupa zahtjeva da se razvije i odgovarajuća fazi matematika. U ovoj knjizi će biti navedene samo osnovne definicije i operacije, koje su neophodne za razumjevanje fazi upravljanja.

Fazi skup *A* je definisan nad univerzalnom *X* funkcijom pripadnosti:

$$m_A(x): X \rightarrow [0,1].$$

Vrijednost $m_A(x)$ predstavlja stepen pripadnosti i to tako što $m_A(x) = 1$ označava da je element *x* u potpunosti kompatibilan sa danim pojmom *A*, $m_A(x) = 0$ označava da element *x* ni na koji način ne odgovara pojmu *A*, dok sve ostale vrijednosti $m_A(x)$ označavaju da se element *x* i pojam *A* u izvjesnoj mjeri slažu.



Slika 9.13. Fazi skupovi i njihovi komplementi.

Komplement fazi skupa A je $\bar{A}$ i on se definiše funkcijom pripadnosti:

$$\overline{m}_A(x) = 1 - m_A(x)$$

Područje fazi skupa A je:

$$sup(A) = \{x \in X \mid m_A(x) > 0\}$$

Grafički prikaz funkcija pripadnosti dva fazi skupa (A i B) i njihovih komplemenata, prikazan je na slici 9.13.

U pogledu osnovnih operacija nad fazi skupovima (presjeka i unije), predloženo je više definicija:

- **presjek fazi skupova**

$$A \cap B: m_{A \cap B}(x) = m_A(x)\ tm_B(x)$$

gdje se t-norma definiše na različite načine, od kojih su najčešći:

1. $xty = min\ (x, y)$
2. $xty = xy$
3. $xty = max(0,\ x+y-1)$

- **unija fazi skupova**

$$A \cup B: m_{A \cup B}(x) = m_A(x)\ sm_B(x)$$

gdje se, između ostalog, kao s-norma koristi:

1. $xsy = max\ (x, y)$
2. $xsy = x + y - xy$
3. $xsy = min\ (1, x + y)$

Kao prirodna posljedica ovih definicija, dobija se i definicija inkluzije dva fazi skupa:

- **inkluzija fazi skupova**

$$A \subset B: m_{A \subset B}(x) = sup\ \{c \in [0,1] \mid m_A(x)tc \leq m_B(x)\}$$

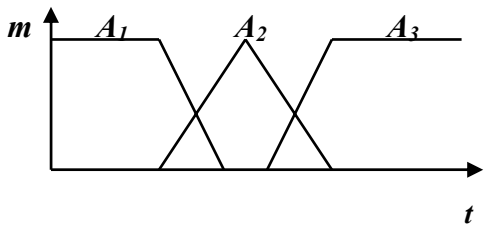
Funkcije pripadnosti presjeka, unije i inkluzije se, takođe, mogu grafički prikazati, što nije učinjeno u ovoj knjizi zbog ograničenog prostora.

9.2.5. Fazi upravljanje

Značenje nekog pojma se precizira uz pomoć atributa. Npr., temperatura može biti *velika* ili *mala*. Pri tome je jasno da se ne može postaviti neka određena vrijednost u kojoj ona prestaje da bude *mala* i postaje *velika*. Drugačije rečeno, granica između ova dva atributa je elastična i zavisi od lične procjene i okolnosti u kojoj se taj pojam posmatra. Ako se želi detaljnija specifikacija ovog pojma, onda se može govoriti da je temperatura: *veoma mala, mala, normalna, velika* i *veoma velika.* Broj pridruženih atributa doprinosi jasnijoj specifikaciji pojma, ali istovremeno umanjuje i njegovu opštost. Međutim, problem granica važenja pojedinog atributa ostaje i dalje otvoren.

Polazeći od rečenog o fazi skupovima, jasno je da oni mogu da igraju ulogu elastičnih granica između pojedinih atributa. Svaki od ovih skupova pridružuje se po jednom atributu i određuje jednu oblast, koja ima jasno semantičko značenje. Ova se ideja može ilustrovati slikom 9.14, gdje su predstavljeni mogući oblici funkcija pripadnosti tri fazi skupa koja određuju pojam temperature.

Ova tri skupa predstavljaju jedan od mogućih okvira spoznaje danog pojma.



Slika 9.14. Okvir spoznaje pojma temperatura.

Posmatrajući uopšteno, može se reći da familija fazi skupova

$$A = \{A_1, A_2, \ldots, A_n\}$$

predstavlja okvir spoznaje nekog pojma X, ako su ispunjeni slijedeći uslovi:

- svaki element pojma X je pridružen bar jednom fazi skupu sa nenultim stepenom pripadnosti:

$$\forall x \in X: \exists m_{Ai}(x) > \varepsilon, \varepsilon > 0,$$

što predstavlja nivo pokrivanja.

Elementi A su unimodalni skupovi, što znači da postoji oblast elemenata x koja ima jasno semantičko značenje i u kojoj samo jedan od fazi skupova A_i ima izrazito veliku funkciju pripadnosti. Izuzetno značajnu ulogu u obradi ulaznih informacija, koje koristi sistem upravljanja, ima definicija okvira spoznaje. Ulazna veličina koja nosi informaciju o određenom pojmu može biti numerička vrijednost, interval ili fazi skup.

Nezavisno od konkretnog oblika u kome je zadana konkretna ulazna veličina, ona se na neki način poredi sa fazi skupovima koji čine okvir spoznaje. Za rezultat poređenja dobija se vektor, čiji elementi predstavljaju mjeru u kojoj dana ulazna veličina odgovara pojedinom atributu. Ova mjera se dalje izražava kao stepen aktivnosti odgovarajućeg fazi skupa.

9.2.6. Osnovni principi projektovanja fazi regulatora

Zadatak realizacije fazi upravljanja se može shvatiti kao svojevrsno modeliranje operatora koji učestvuju u procesu. U strategiji upravljanja, koju koristi operator, uočavaju se postupci koji po svojoj prirodi pripadaju fazi logici (npr. "održavati izlaz *blisko* zadanoj trajektoriji uz *prilično* mali utrošak energije").

Pored toga, osnovni ali ne i jedini, izvor znanja pri formiranju upravljačkog algoritma je protokol upravljanja, koji se sastoji od skupa pogodbenih iskaza "AKO ... $\rightarrow$ TADA". Prilikom formiranja jezičkog protokola za projektovanje fazi regulatora, postavljaju se dvije osnovne vrste pitanja:

- pitanja o ponašanju operatora, npr.: *Šta bi radili u takvoj i takvoj situaciji?*
- pitanja o ponašanju procesa, npr.: *Zašto nastaje takva i takva situacija?*

Koraci projektovanja fazi regulatora

U cilju objašnjenja osnovnog pristupa u projektovanju fazi regulatora, može se posmatrati projektovanje rashladnog uređaja. Pretpostavka je da je izvršni organ na ulazu u proces motor, sa mogućnošću promjene brzine. Stepen hlađenja sistema neposredno zavisi od brzine obrtanja motora. Temperatura sistema se mjeri termo-spregom. Projektovanje upravljanja razmatranim sistemom se vrši u pet osnovnih koraka.

1. Izbor promjenljivih

Ovaj korak je, najčešće, određen samim procesom kojim se želi upravljati, kao i raspoloživim izvršnim organima i mjernom opremom.

U razmatranom primjeru ulaz regulatora je **temperatura (y)**, mjerena u sistemu, dok je izlaz regulatora **brzina motora (u)**.

2. Formiranje okvira spoznaje

U ovom koraku se definišu okviri spoznaje usvojenih ulaznih i izlaznih veličina, a u skladu sa ranije navedenim principima formiranja okvira spoznaje nekog pojma.

Za razmatrani rashladni uređaj je usvojeno:

- **temperatura** = *(hladno, svježe, prijatno, toplo, vruće)* $\rightarrow$ **Y = (Y_1, Y_2, Y_3, Y_4, Y_5),**
- **brzina motora** = *(stop, sporo, srednje, brzo, veoma brzo)* $\rightarrow$ **U = (U_1, U_2, U_3, U_4, U_5).**

3. Izbor fazi pravila

Na osnovu logike rada rashladnog uređaja i iskustva u njegovom korištenju, mogu se definisati slijedeća pravila:

1. Ako je *y* ***hladno***, onda je *u* ***stani***.
2. Ako je *y* ***svježe***, onda je *u* ***sporo***.
3. Ako je *y* ***prijatno***, onda je *u* ***srednje***.
4. Ako je *y* ***toplo***, onda je *u* ***brzo***.
5. Ako je *y* ***vruće***, onda je *u* ***veoma brzo***.

Ovako definisana pravila određuju fazi relaciju:

$$R = \bigcup_{i=1}^{5} R_i, R_i(y_p, u_k) = m_{Yi}(y_p) tm_{Ui}(u_k)$$

gdje $\cup$ označava bilo koju *s*-normu sa značenjem "ili". Dobijena fazi relacija, u stvari, povezuje okvire spoznaje ulaza i izlaza regulatora.

Za svaki par (y, u) jednoznačno je određena relacija $R(y, u)$, koja definiše odgovarajuću implikaciju iz fazi pravila.

4. Odlučivanje

Suština rada fazi regulatora ogleda se u tome da na osnovu mjerenja ulaza i definisanih fazi pravila, on odlučuje o vrijednosti izlaza. Ilustracije radi postavlja se slijedeće pitanje. Ako se na ulazu registruje fazi skup Y_a, koje pravilo regulator treba da primjeni, odnosno kako se dobija odgovarajući izlazni fazi skup U_a?

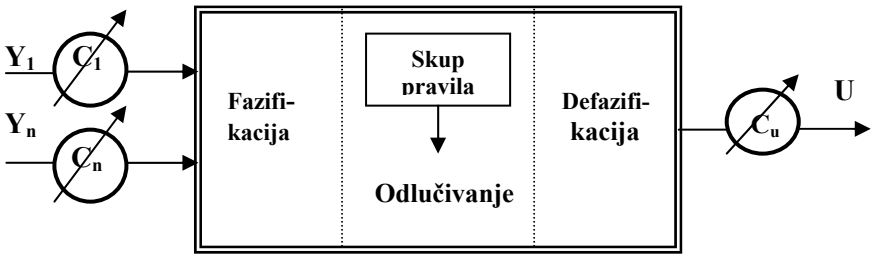
Ovaj problem je rješen tako da se sva pravila uvijek uključuju istovremeno, s tim što svako od njih važi do određenog stepena. Pri tome, stepen važenja pravila zavisi od funkcija pripadnosti fazi skupova ulaza.

5. Defazifikacija

U procesu odlučivanja dobija se izlazni fazi skup, koji određuje upravljanje kojim će regulator djelovati na proces. Međutim, izvršni organi procesa mogu da reaguju samo na konkretne fizičke signale koji na njih djeluju. Priroda izvršnih organa zahtjeva da se fazi slika izlaza regulatora izoštri do te mjere da se u žiži nađe samo jedna numerička vrijednost (u_0) koja predstavlja upravljačku promjenljivu koja djeluje na izvršni organ. Ovaj postupak izoštravanja označava se kao defazifikacija.

Predložene su dvije osnovne metode defazifikacije (koje neće biti razmatrane u ovoj knjizi):

- srednja vrijednost maksimuma,
- centar gravitacije.



Slika 9.15. Fazi regulator.

Na osnovu prethodnog izlaganja jasno je da se u okviru fazi regulatora obavlja prvo postupak fazifikacije mjernih signala, zatim se vrši odlučivanje i na kraju se, pomoću defazifikacije, određuje upravljački signal, slika 9.15.

Sva upravljačka pravila imaju kvalitativnu prirodu i treba da važe za široku klasu problema. Zbog toga se, na ulazu i izlazu regulatora, vrši skaliranje signala. Time se u okviru istih pravila prilagođava semantika osnovnih pojmova.

9.3. Primjer fazi ekspertnog sistema

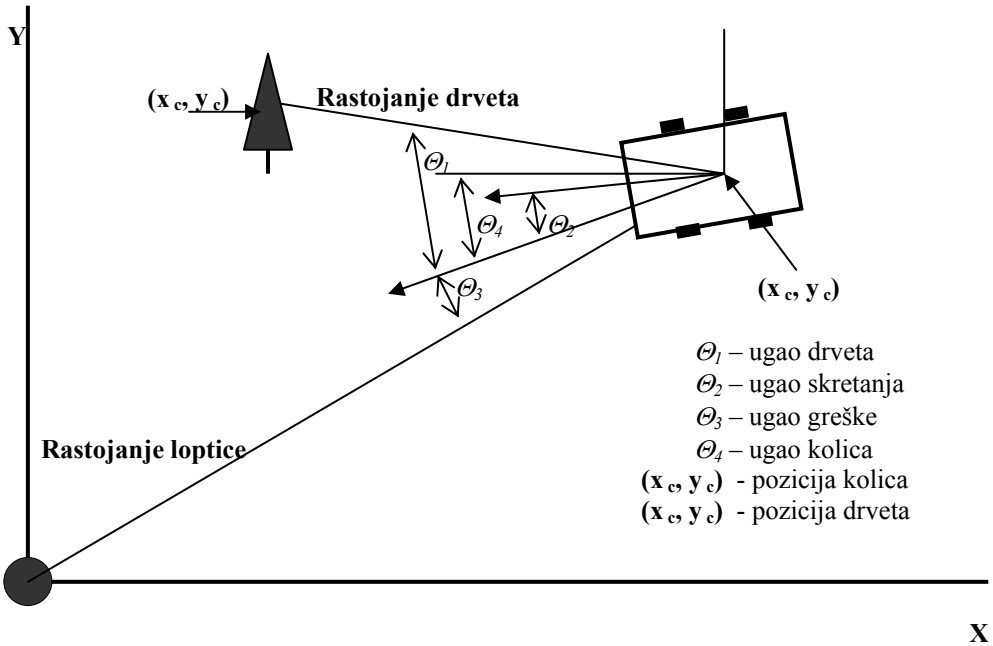
Za ilustraciju razvoja fazi ES biće razmotren problem navigacije kolica za golf. Zadatak je da se automatizuje kretanje golf kolica, u skladu sa potrebama igrača golfa. Postoji nekoliko zadataka koje treba riješiti prilikom razvoja traženog fazi ES:

1. Definisati problem;
2. Definisati lingvističke promjenljive;
3. Definisati fazi skupove;
4. Definisati fazi pravila;
5. Izgradnja fazi ES;
6. Testiranje fazi ES;
7. Poboljšanja fazi ES.

9.3.1. Definisanje problema

Kod razvoja ES prvobitno je potrebno odrediti izvor znanja. Najčešći izvor znanja je ekspert, koji poznaje razmatranu problematiku. Ekspert je ukazao da je osnovni problem efikasne i sigurne navigacije golf kolica, iz određene polazne pozicije, kretanje golf loptice. Efikasnost se postiže minimiziranjem rastojanja koje se prelazi i vremena prelaženja. Za izvršavanje ovih zadataka je potrebno obezbjediti kontrolu fazi sistema za navedene veličine. Slika 9.16. ilustruje razmatrani navigacioni problem, koji se u osnovi svodi na problem anuliranja greške. Golf kolica moraju ići ka lopti, svodeći na nulu grešku između ugaonog pravca kolica i pravca prema lopti. Može se uzeti da je približno zaustavljanje lopte od kolica na udaljenosti od 3 m.

Osim navedenog, golf kolica treba ubrzati do određene maksimalne brzine, nakon toga ih usporiti i eventualno zaustaviti, kada se približe lopti. Minimiziranje



Slika 9.16. Geometrija navigacije kolica za golf.

greške između lokacije golf kolica i lokacije lopte svodi se na promjenu brzine kolica. Ukoliko se na putu kolica nađe drvo, kolica se moraju usporiti i usmjeriti oko drveta, a nakon toga moraju postići optimalnu brzinu i usmjeriti prema lopti.

9.3.2. Definisanje lingvističkih promjenljivih

Potrebno je, na osnovu savjeta eksperta za igranje golfa, definisati lingvističke promjenljive. Određuju se lingvističke promjenljive, koje predstavljaju univerzalnost jezika i fazi skupova, i koje će predvidjeti moguće situacije.

Na osnovu razmatranog, fazi logički sistem se mora zadržati unutar tri osnovna problema:

1. Kontrole navigacije pravca golf kolica;
2. Kontrole brzine kolica;

2. Kontrole navigacije kolica u cilju izbjegavanja drveća.

Osim navedenog, ekspert za golf treba da odgovori na uopštene diskusije kako ove probleme treba riješiti. Razmatrajući prvi problem, ekspert treba da da tumačenje strategije navigacije kolica prema lopti: "Kada je pravac kretanja kolica udaljen od pravca prema lopti, podesiti pravac kolica u pravcu lopte". Na isti način će se dobiti ekspertna preporuka za kontrolu brzine kolica: "Kada su kolica udaljena od lopte, učiniti brzinu kolica većom"; "Kada su kolica blizu lopte, usporiti kolica", a za pravac kolica i brzinu: "Kada su kolica blizu drveta i idu prema njemu, usporiti kolica i pomjeriti pravac kretanja kolica od drveta".

Iz navedene diskusije mogu se definisati lingvističke promjenljive univerzalnog jezika, a takođe tražiti od eksperta za golf da definiše njihovu oblast:

LINGVISTIČKE PROMJENLJIVE	PODRUČJE		
Ugao greške	**- 180**	**prema**	**180^o**
Ugao drveta	**- 180**	**prema**	**180^o**
Ugao upravljanja	**- 45**	**prema**	**45^o**
Brzina	**0**	**prema**	**5 m/s**
Ubrzanje	**- 2**	**prema**	**1 m/s^2**
Rastojanje lopte	**0**	**prema**	**600 m**
Rastojanje drveta	**0**	**prema**	**1000 m**

9.3.3. Definisanje fazi skupova

Slijedeći zadatak je definisanje fazi skupova za sve varijante univerzalnog govora. Da se ovo uradi potrebno je ponovo konsultovati eksperta i zatražiti da da listu tipičnih pridjeva za svaku lingvističku promjenljivu. Na slici 9.17. je predstavljen rječnik problema.

Sugestije za dizajn: Potrebno je popraviti rječnik termina koji se koriste u sistemu koji uključuje sve lingvističke promjenljive i njima pridružene pridjeve.

Slijedeća pitanja za eksperta se odnose na informacije koje će omogućiti definisanje fazi skupova za svaki pridjev definisan slikom 9.17.

Ugao greške	Ugao drveta	Ugao skretanja	Brzina	Ubrzanje	Udaljenost lopte	Udaljenost drveta
mnogo negativno	mnogo negativno	krajnje desno	nula	pojačati kočenje	nula	blizo
malo negativno	malo negativno	malo desno	veoma sporo	ublažiti kočenje	veoma mala	
nula	nula	nula	sporo	granica	mala	
malo pozitivno	malo pozitivno	malo lijevo	srednje	nula	srednja	
mnogo pozitivno	mnogo pozitivno	krajnje lijevo	brzo	malo ubrzanje	daleko	
				zaustavljanje		

Slika 9.17. Fazi promjenljive sa pridjevima.

Primjer pitanja:

1. "Koja se brzina smatra malom?"
   Ekspert na ovo pitanje daje neodređeni odgovor: "od 0 do 3 m/s".
2. "Koji su stepeni vrijednosti različitih brzina 'spore' – 'do kojeg stepena uvjerenja se misli da je 1 m/s sporo'?"

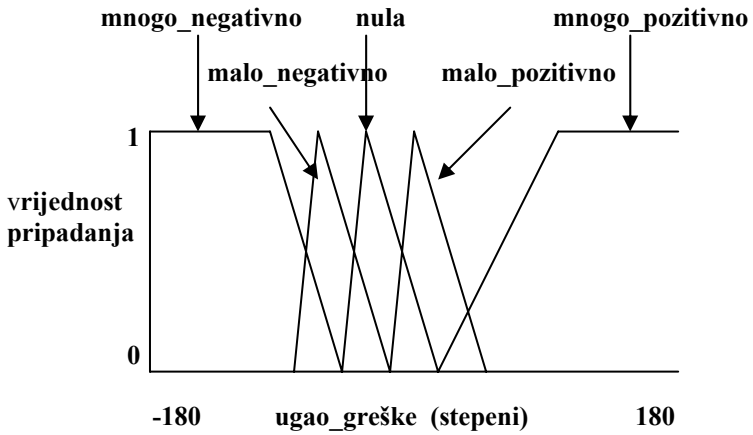
Može se koristiti isto pitanje za ostale vrijednosti brzine. Nakon toga se vrši izbor funkcije, odnosno da li je svrsishodno predstavljati vrijednosti brzina sa njima korespondirajućim vrijednostima uvjerenja. Na taj način se dobija **prvi fazi skup**.

Fazi prikazivanje ili funkcije pripadanja mogu imati različite oblike u zavisnosti od ekspertovog posmatranja različitih domenskih vrijednosti za vrijednosti uvjerenja. U praksi, za dio linearnih funkcija trougaonog ili trapezoidnog oblika, obezbjeđen je adekvatan obuhvat prema ekspertovim uvjerenjima, što istovremeno pojednostavljuje izračunavanja.

Ovaj proces se može nastaviti u cilju definisanja ostalih fazi skupova sa lingvističkom promjenljivom "brzina". U praksi, na osnovu oformljenja prvog fazi skupa, ekspert može da definiše ostale fazi skupove pomjerajući se ka izabranim funkcijama putem univerzalnih promjenljivih. Međutim, ovaj pristup može imati svoje nedostatke.

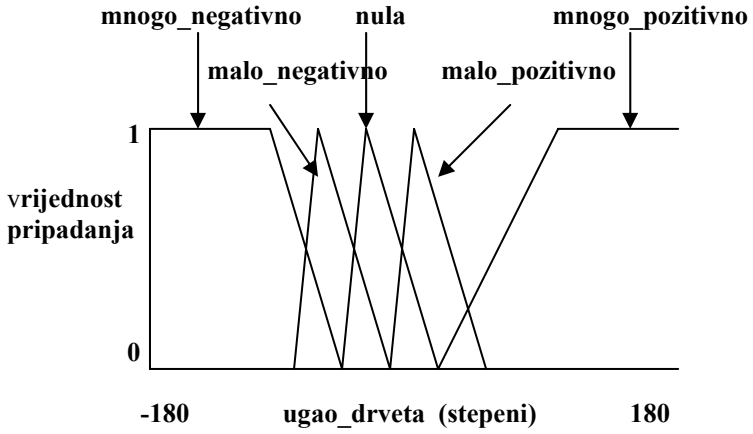
Sugestije za dizajn: Omogućiti ekspertima pomjeranje postojećeg fazi skupa funkcija, koristeći univerzalnost jezika, na ostale pridjeve.

Na slikama od 9.18. do 9. 24. su prikazani fazi skupovi za sve fazi promjenljive prikazane slikom 9.17.



mnogo_pozitivno		malo_pozitivno		nula		malo_negativno		mnogo_negativno	
X	Y	X	Y	X	Y	X	Y	X	Y
30	0	0	0	-30	0	-60	0	-180	1
90	1	30	1	0	1	-30	1	-60	1
180	1	60	0	30	0	0	0	-30	0

Slika 9.18. Fazi skup ugla greške.

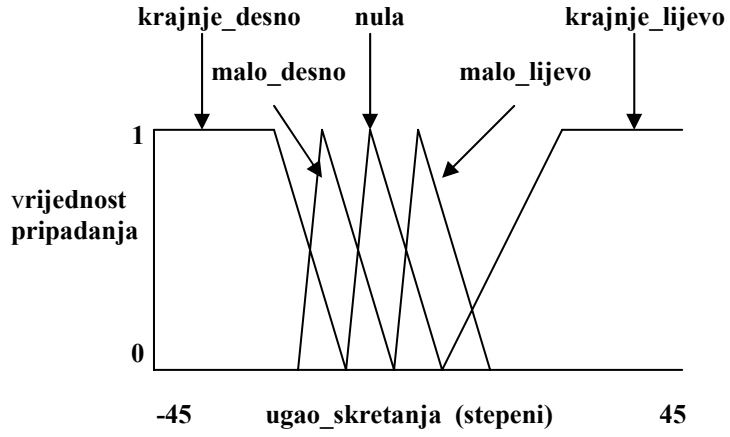


mnogo_pozitivno		malo_pozitivno		nula		malo_negativno		mnogo_negativno	
X	Y	X	Y	X	Y	X	Y	X	Y
30	0	0	0	-30	0	-60	0	-180	1
90	1	30	1	0	1	-30	1	-60	1
180	1	60	0	30	0	0	0	-30	0

Slika 9.19. Fazi skup ugla drveta.

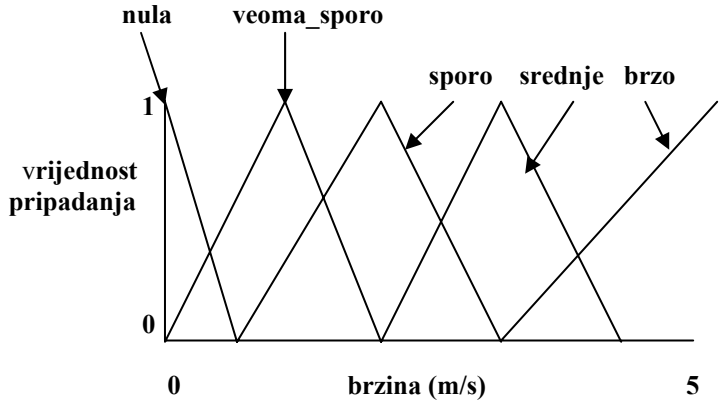
Jedna od ključnih stvari je da se osigura dovoljno nalijeganje kod definisanja fazi skupova, čime se osigurava da se za svaku moguću vrijednost obezbjedi neka od vrijednosti pripadanja fazi skupu. U suprotnom je moguće da uvijek postoji dio fazi sistema koji nije obuhvaćen nekom od mogućih vrijednosti.

Sugestije za dizajn: Održavati dovoljno preklapanje pridjeva u fazi skupovima.



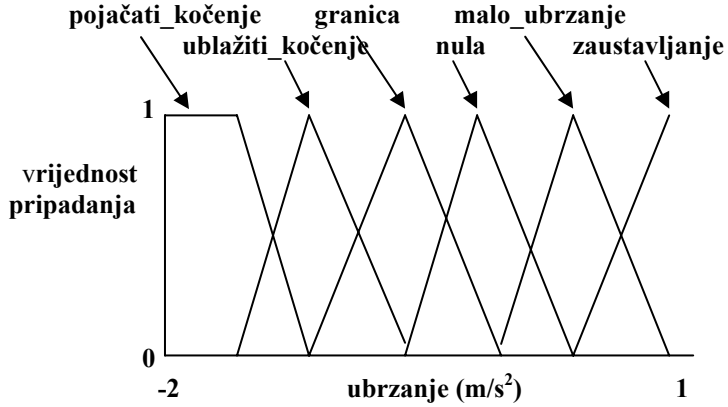
kajnje_lijevo		malo_lijavo		nula		malo_desno		krajnje_desno	
X	Y	X	Y	X	Y	X	Y	X	Y
7,5	0	0	0	-7,5	0	-15	0	-45	1
22,5	1	7,5	1	0	1	-7,5	1	-15	1
45	1	15	0	7,5	0	0	0	-7,5	0

Slika 9.20. Fazi skup ugla skretanja.



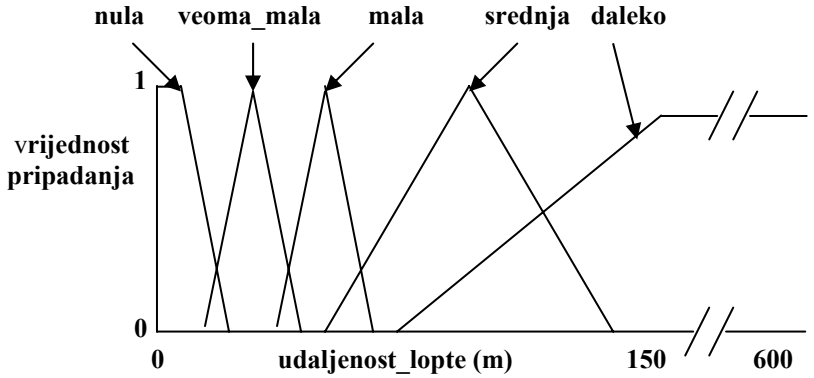
nula		veoma_sporo		sporo		srednje		brzo	
X	Y	X	Y	X	Y	X	Y	X	Y
0	1	0	0	0,5	0	1,5	0	0	0
0,5	0	0,7	1	1,5	1	2,5	1	2,5	0
		1,5	0	2,5	0	3,5	0	5,0	1

Slika 9.21. Fazi skup brzina.



pojačati_kočenje		ublažiti_kočenje		granica		nula		malo_ubrzanje		zaustavljanje	
X	Y	X	Y	X	Y	X	Y	X	Y	X	Y
-2	1	-1,5	0	-1	0	-0,5	0	0	0	0,5	0
-1,5	1	-1	1	-0,5	1	0	1	0,5	1	1	1
-1	0	-0,5	0	0	0	0,5	0	1	0		

Slika 9.22. Fazi skup ubrzanja.



nula		veoma_mala		mala		srednja		daleko	
X	Y	X	Y	X	Y	X	Y	X	Y
0	1	3	0	12	0	24	0	60	0
3	1	12	1	24	1	60	1	150	1
9	0	18	0	36	0	96	0	600	0

Slika 9.23. Fazi skup udaljenosti lopte.

blizo	
X	Y
0	1
50	0
1000	0

Slika 9.24. Fazi skup udaljenosti drveta.

9.3.4. Definisanje fazi pravila

Naredni zadatak je definisanje fazi pravila. Da se ovo izvrši potrebno je konsultovati eksperta za golf o tri primarna problema:

1. Usmjeravanju kolica prema lopti;
2. Kontrolisanju brzine golf kolica;
3. Navigaciji kolica u cilju izbjegavanja drveća.

Osim navedenog, ekspert treba da koristi ranije definisane fazi pridjeve. Ukoliko se pojave i drugi pridjevi tokom intervjua, oni se dodaju prethodnoj listi i definiše se fazi skup pojedinačno za svaki pridjev. Ti pridjevi se navode u narednim fazi pravilima, u kojima se, takođe, koristi nekoliko fazi granica i fazi operator NOT. Upotrebu fazi granica i fazi operatora treba naknadno razmotriti, kada fazi sistem bude razvijen.

Inicijalni skup pravila ne treba da bude veliki. Treba da uključi fazi skupove i pridjeve koji se nalaze u rječniku, bez bilo kakvih dodatnih priloga. Za vrijeme testiranja sistema, mogu se dodati prilozi u cilju poboljšanja performansi sistema.

PRAVILA ZA UPRAVLJANJE GOLF KOLICIMA

```
RULE 1S – održavanje pravca upravljanja
IF ugao_greške je nula
AND rastojanje_stabla je NOT SOMEWHAT blizo
AND ugao_stabla je NOT SOMEWHAT nula
THEN načiniti ugao_upravljanja nulom

RULE 2S – promjeniti pravac upravljanja malo udesno
IF ugao_greške je malo_pozitivan
AND  rastojanje_drveta je NOT SOMEWHAT blizo
AND ugao_drveta je NOT SOMEWHAT nula
THEN načiniti ugao_upravljanja malo_desno

RULE 3S – promjena pravca upravljanja malo lijevo
IF ugao_greške je malo_negativan
AND rastojanje_drveta je NOT SOMEWHAT blizo
AND ugao_drveta je NOT SOMEWHAT nula
THEN načini ugao_upravljanja malo_lijevo
```

```
RULE 4S – promjena pravca upravljanja malo desno
IF ugao_greške mnogo_pozitivan
AND brzina je velika
THEN načini ugao_upravljanja malo_lijevo

RULE 5S – promjena pravca upravljanja jako desno
IF ugao_greške je jako_pozitivan
AND brzina je NOT brza
THEN učini ugao_upravljanja jako_desno

RULE 6S – promjena pravca upravljanja malo lijevo
IF ugao_greške je mnogo_negativan
AND brzina je velika
THEN učini ugao_upravljanja malo_lijevo

RULE 7S – promjena pravca upravljanja jako lijevo
IF ugao_greške je jako_negativan
AND brzina je NOT brza
THEN učini ugao_upravljanja jako_lijevo
```

PRAVILA ZA UBRZANJE

```
RULE 1A – ublažiti kočenje
IF ugao_greške je mnogo_pozitivan
AND brzina je velika
THEN učiniti ubrzanje ublažiti_kočenje

RULE 2A – ublažiti kočenje
IF ugao_greške je mnogo_negativan
AND brzina je velika
THEN učiniti ubrzanje ublažiti_kočenje

RULE 3A – zaustavljanje
IF udaljenost_lopte je velika
AND brzina je NOT VERY velika
THEN učiniti ubrzanje zaustavljanje

RULE 4A – skup ubrzanja do nule
IF udaljenost_lopte je velika
AND brzina je VERY velika
THEN učiniti ubrzanje nula
```

```
RULE 5A – malo ubrzanje
IF rastojanje_lopte je veliko
AND brzina je NOT velika
THEN učiniti ubrzanje malo_ubrzanje

RULE 6A – postaviti ubrzanje na nulu
IF rastojanje_lopte je srednje
AND brzina je velika
THEN učiniti ubrzanje nula

RULE 7A – pojačati kočenje
IF rastojanje_lopte je malo
AND brzina je velika
THEN učiniti ubrzanje pojačati_kočenje

RULE 8A – malo ubrzanje
IF rastojanje_lopte je malo
AND brzina je nula
THEN učiniti ubrzanje malo_ubrzanje

RULE 9A – pojačati kočenje
IF rastojanje_lopte je veoma_malo
AND brzina je velika
THEN učiniti ubrzanje pojačati_kočenje

RULE 10A – ublažiti kočenje
IF rastojanje_lopte je veoma_malo
AND brzina je srednja
THEN učiniti ubrzanje ublažiti_kočenje

RULE 11A – granica
IF rastojanje_lopte veoma_malo
AND brzina je veoma_spora
THEN učiniti ubrzanje na granici

RULE 12A – postaviti ubrzanje na nulu
IF rastojanje_lopte je veoma_malo
AND brzina je veoma_spora
THEN učiniti ubrzanje nula

RULE 13A – malo ubrzanje
IF rastojanje_lopte je veoma_malo
AND brzina je NOT nula
```

```
THEN učiniti ubrzanje pojačati_kočenje

RULE 14A – pojačati kočenje
IF rastojanje_lopte je nula
AND brzina je NOT nula
THEN učiniti ubrzanje pojačati_kočenje

RULE 15A – granica
IF rastojanje_lopte malo
AND brzina je srednja
THEN učiniti ubrzanje na granici

RULE 16A – postaviti ubrzanje na nulu
IF rastojanje_lopte je malo
AND brzina je spora
THEN učiniti ubrzanje nula
```

PRAVILA ZA IZBJEGAVANJE DRVETA

```
RULE 1T – malo odstupanje lijevo za izbjegavanje drveta
IF rastojanje_drveta je SOMEWHAT blizo
AND ugao_drveta je SOMEWHAT nula
AND ugao_drveta je SOMEWHAT malo_pozitivno
THEN učiniti ugao_skretanja malo_lijevo

RULE 2T – malo odstupanje desno za izbjegavanje drveta
IF rastojanje_drveta je SOMEWHAT blizo
AND ugao_drveta je SOMEWHAT nula
AND ugao_drveta je SOMEWHAT malo_negativno
THEN učiniti ugao_skretanja malo_desno

RULE 3T – pojačati kočenje za izbjegavanje drveta
IF rastojanje_drveta je VERY blizo
AND ugao_drveta je nula
THEN učiniti ubrzanje pojačati_kočenje
```

Sugestije za dizajn: Koristiti priloge u pravilima za podešavanje sistemskih performansi.

9.3.5. Izgradnja fazi ekspertnih sistema

Nakon izrade fazi skupova i pravila, slijedeći zadatak je izgradnja fazi ES. Ovaj zadatak uključuje kodiranje fazi skupova, pravila i procedura za izvršavanje fazi logičkih funkcija za fazi zaključivanje. Za izvršavanje ovih zadataka postoje dva pristupa:

- izgradnja fazi sistema koristeći programske jezike,
- korištenje fazi logički razvojni skeletni sistem (*shell*).

Većina istraživača fazi logičkih ekspertnih sistema koristi programski jezik C. Ovaj programski jezik nudi strukture podataka koje su vođene procedurama implementiranja fazi logike. Međutim, izgrađujući fazi logički sistem, koristeći viši programski jezik, potrebno je da istraživač kodira ne samo znanje problema, fazi skup i fazi pravila, nego i proceduru fazi logike.

Fazi logički *shell* obezbjeđuje cjelovito okruženje za izgradnju fazi logičkog ES. Dizajner je jedino odgovoran za kodiranje znanja problema. Zadatak se često izvršava koristeći sintaksu prirodnog jezika, prema pravilima i grafičkim metodama za definisanje fazi skupova. Postoji čitav niz fazi logičkih razvojnih skeletnih sistema (*shell*), pregled kojih, zbog ograničenog prostora, nije dan u ovoj knjizi. Pretpostavka je da je razmatrani ES razvijen pomoću *CubiCalc* (*Hyperlogic Corp., Escondido, CA*). *CubiCalc* je prozorski definisan razvojni alat, koji dozvoljava brzi razvoj prototipova za fazi logičke ES. Takođe, omogućava simulaciju za jednostavno testiranje razvijenog ES.

9.3.6. Testiranje fazi ES

Nakon izgradnje fazi ES potrebno ga je testirati da se provjeri da li ispunjava specifikacije postavljene zadatkom. U razmatranom primjeru testiranje simulira upotrebu *CubiCalc* simulacione jedinice. Svi testovi se izvršavaju upotrebom max – proizvod tehnike zaključivanja. Broj test situacija razmatranog primjera će zavisiti od:

- lokacije golf lopte,
- lokacije drveća,
- lokacije golf kolica, i
- orijentacije golf kolica.

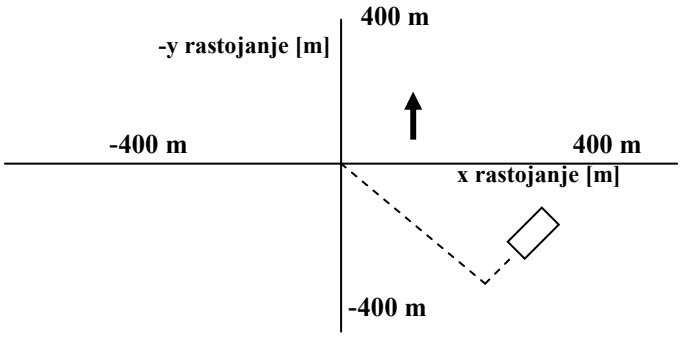
Mogu se uzeti u razmatranje dvije situacije:

1. U blizini nema drveta;
2. U blizini se nalazi samo jedno drvo.

U oba test slučaja se pretpostavlja da se golf lopta nalazi na koordinatama (0,0).

U blizini nema drveta

Prvi test slučaj razmatra situaciju kada drvo nije na direktnoj putanji golf kolica. Kolica su usmjerena u pravcu lopte pod uglom -45^{o}. Simulacioni test je prikazan na slikama 9.25. i 9.26.



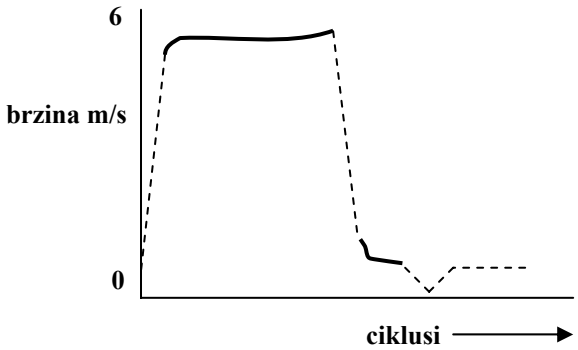
Slika 9.25. Navigacija golf kolica, test slučaj 1.

Slika 9.25. pokazuje putanju golf kolica tokom testa i uočava se da postoji mala poteškoća u pristizanju lopte. Nakon početnog manevra u dostizanju cilja, kolica prate pravu liniju prema lopti. Nema odstupanja od prave linije, tako da udaljeno drvo nema uticaja. Golf kolica se zaustavljaju približno na 3 m od lopte.

Slika 9.26. prikazuje brzinu kolica tokom testa. Iz stanja mirovanja, kolica brzo postižu brzinu do dozvoljene maksimalne vrijednosti od 5 m/s. Kada se kolica približe lopti, brzo se zaustavljaju. Na ciljnom položaju kolica počinju da osciluju. O ovoj neželjenoj pojavi će kasnije biti govora.

Za proučavanje funkcionisanja fazi sistema, simulacija se zaustavlja u cilju provjere fazi pravila. Na poziciji gdje je simulacija zaustavljena, parametri su slijedeći.

- kolica su 28,5 m od lopte,
- kolica su pomjerena neposredno prema lopti,
- ubrzanje kolica je 4,5 m/s^2,
- ugao greške je neznatan,
- drvo je dovoljno udaljeno.



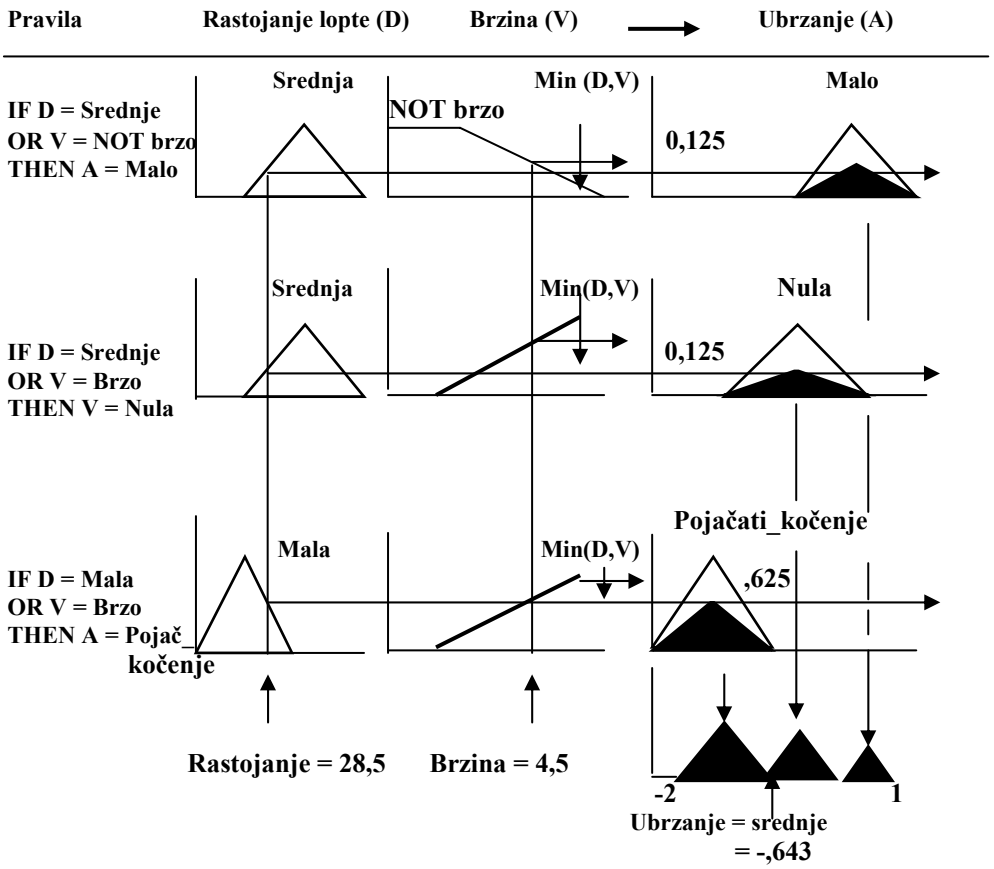
Slika 9.26. Brzina golf kolica, test slučaj 1.

Prema navedenim parametrima odvijaju se tri pravila: PRAVILA 5A, 6A i 7A. Za ocjenu PRAVILA 5 pripadajuće veze funkcije "srednje" za "odstojanja_lopte" i "brzo" za "brzinu", moraju biti ocjenjene. Vrijednost "rastojanje_lopte" od 28,5 m se klasifikuje kao "srednji" za stepen od 0,125 (vidjeti sliku 9.23). "Brzina" od 4,5 m/s je razmatrana kao "brzo" za stepen od 0,8 (vidjeti sliku 9.21), a sa iste slike "NOT brzo" je za stepen od 0,2. Kada AND sabere ove dvije vrijednosti, operator minimuma daje 0,125 kao istiniti antecedent. Koristeći tehniku zaključivanja max – proizvod, ova vrijednost će biti korištena za određivanje pripadajuće veze funkcije za zaključak PRAVILA 5A "malo_ubrzanje"

Kada je PRAVILO 6A ocjenjeno, koristeći tačnu vrijednost za "srednji" i "brz", koje su ranije određene, antecedent ponovo ima kombinovanu tačnu vrijednost od 0,125. Pripadajuća vrijednost za "nulu" ubrzanja je skalarna sa ovom vrijednosti.

Kada se ocjenjuje PRAVILO 7A, rastojanje od 28,5 m je srazmjerno "malo" sa stepenom od 0,625, a "brzina" je "brz" sa stepenom od 0,8. Na osnovu toga "pojačati_kočenje" postaje skalarna sa vrijednosti 0,625.

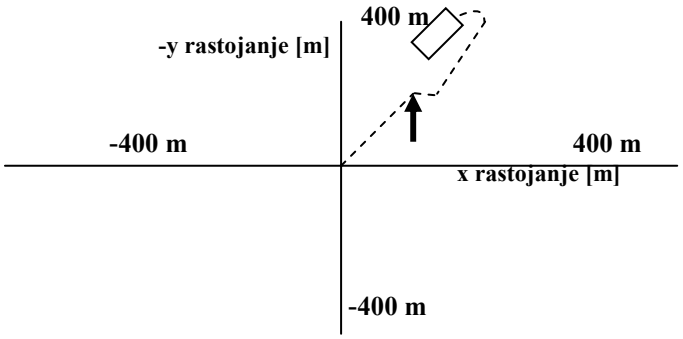
Paralelno uključivanje navedenih triju pravila je prikazano na slici 9.27. Tri navedena fazi skupa, koja se odnose na "ubrzanje" su kombinovano prikazana, korištenjem operacije unije. Srednja vrijednost ovih rezultata je izračunata uključujući promjene u ubrzanju golf kolica. Mada je finalna vrijednost negativna, simulator smanjuje brzinu golf kolica i proces se nastavlja.



Slika 9.27. Probno fazi zaključivanje za test slučaj 1.

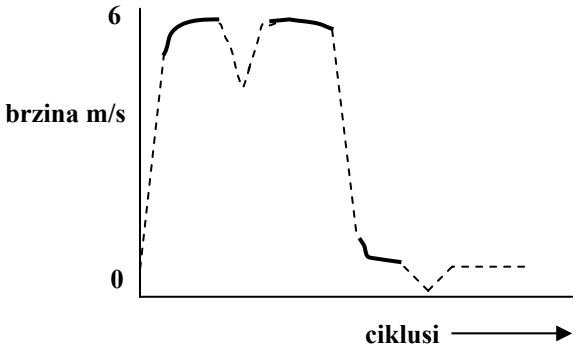
Drveće na putu

U narednom test slučaju, razmatra se situacija u kojoj je drvo na putanji golf kolica. Kolica su zakrenuta u odnosu na loptu za ugao 60^o. Simulirani test rezultati su prikazani na slikama 9.28. i 9.29.



Slika 9.28. Navigacija golf kolica, test slučaj 2.

Slika 9.28. pokazuje da golf kolica moraju manevrom dostići cilj. Kolica nakon toga slijede pravu liniju prema lopti, dok se ne približe drvetu. Kolica obilaze drvo, a nakon toga pravom linijom nastavlkaju prema lopti. Kao i u slučaju 1, kolica se zaustavljaju približno 3 m od lopte.



Slika 9.29. Brzina golf kolica, test slučaj 2.

Na slici 9.29. prikazano je da kolica ponovo nastavljaju da se ubrzavaju do maksimalne dozvoljene brzine, dok se ne približe drvetu. Kod drveta usporavaju, kontinualno obilazeći drvo. Takođe, kao i u testu za slučaj 1, brzina kolica oscilira kada se približavaju lopti.

9.3.7. Podešavanje fazi ES

Podešavanje fazi ekspertnog sistema počinje poređenjem sistemskog odgovora sa inicijalnim očekivanjima. Eventualna odstupanja postaju predmet dodatnih analiza. U mnogim projektima fazi ES, vrijeme utrošeno u određivanje fazi skupova je malo u odnosu na vrijeme potrebno za podešavanje sistema. Najčešće, prva serija fazi skupova i pravila obezbjeđuju suštinska rješenja problema. Ovo je, možda, jedna od principijelnih prednosti fazi logike, dok postizanje tačnih rješenja zahtjeva dodatne napore.

Analiza razmatranih testova pokazuje da je postignuto više inicijalnih uslova. To znači da golf kolica uspješno dolaze do lopte i izbjegavaju sudar sa drvetom. Međutim, provjera brzine kolica crtanjem ukazuje da može biti poteškoća. Slike 9.26. i 9.29. ukazuju na problem kada su kolica veoma blizo lopte, zbog oscilovanja kolica između nulte brzine i određene vrijednosti. Za ispravljanje ovog problema potrebno je provjeriti pravilo kontrole brzine, naročito ono koje kontroliše brzinu kolica kada “zaista blizo” prilaze lopti. Na ovu situaciju se odnose PRAVILA 9A do 13A. PRAVILO 13A uzrokuje povećanje brzine kada se brzina približava nuli, što uzrokuje oscilacije. Za zaštitu od oscilacija, eliminiše se ovo pravilo.

10

U ovom poglavlju:

- ***Pojam neuronskih mreža***
- ***Vještačke neuronske mreže***
- ***Obučavanje neuronskih mreža***
- ***Realizacija neuronskih mreža***
- ***Područja primjene neuronskih mreža***
- ***Primjer ekspertnog sistema sa neuronskim mrežama***

NEURONSKE MREŽE I EKSPERTNI SISTEMI

10.1. POJAM NEURONSKIH MREŽA

Iako se neuronske mreže (NM) intenzivno izučavaju od 1940-tih, one nisu imale značajniju praktičnu primjenu sve do 1980-tih godina, kada su algoritmi postali podesni za opštu upotrebu (aplikacije). Danas se NM primjenjuju za rješavanje sve većeg broja svakodnevnih problema značajne složenosti. U programiranju se mogu koristiti kao "generator" (*engine*) koji je u stanju da vrši različita prepoznanja i klasifikacije i koji ima sposobnost da izvrši generalizaciju prilikom odlučivanja pri nestruktuiranim ulaznim podacima. NM nude idealno rješenje za raznovrsno klasifikovanje problema, kao što je:

- prevođenje teksta u govor,
- prepoznavanje slova,
- rješavanje problema za koje ne postoji algoritamsko rešenje i slično.

Pokazuju dobre rezultate prilikom predviđanja i modeliranja sistema, gdje fizički procesi nisu jasni ili su veoma kompleksni. Prednost NM leži u visokoj elastičnosti prema poremećajima u ulaznim podacima i u sposobnosti učenja. NM često uspješno rješavaju probleme koji su previše kompleksni za konvencionalne tehnologije (na primjer, problemi koji nemaju algoritamsko rješenje ili za koje je

Materijal u ovom poglavlju većim dijelom je preuzet sa Interneta htp://www.iis.ns.ac.yu.htlm, (S. Stankovski,"Neuronske mreže"),2004.i rada Kosko, B.,"Neural Networks and Fuzzy Systems",1992.

algoritam previše komplikovan da bi se napravio) i često su dobra pratnja problemima koji se rješavaju.

Velika prednost NM se nalazi u mogućnosti paralelne obrade podataka, naročito tokom izračunavanja komponenti koje su nezavisne jedne od drugih. Neuronske mreže su sistemi sastavljeni od više jednostavnih elemenata (neurona) koji obrađuju podatke paralelno. Funkcije koje su NM u stanju da obrađuju su određene strukturom mreže, jačinom konekcije, a obrada podataka se izvodi u neuronima. Svaki elemenat operiše samo lokalnim informacijama, radi asinhronizovano, kao da nema sistemskog sata.

Postoje dve kategorije neuronskih mreža:

- vještačke, i
- biološke neuronske mreže.

Nervni sistem živih bića je bioloških predstavnik NM. Vještačke NM su po strukturi, funkciji i obradi informacija slične biološkim NM, mada se radi o vještačkim tvorevinama. Neuronska mreža u računarskim naukama predstavlja veoma povezanu mrežu elemenata koji obrađuju podatke. Sposobne su da izađu na kraj sa problemima koji se tradicionalnim pristupom teško rješavaju, kao što su govor i prepoznavanje oblika. Jedna od važnijih osobina NM je njihova sposobnost da uče na ograničenom skupu primjera.

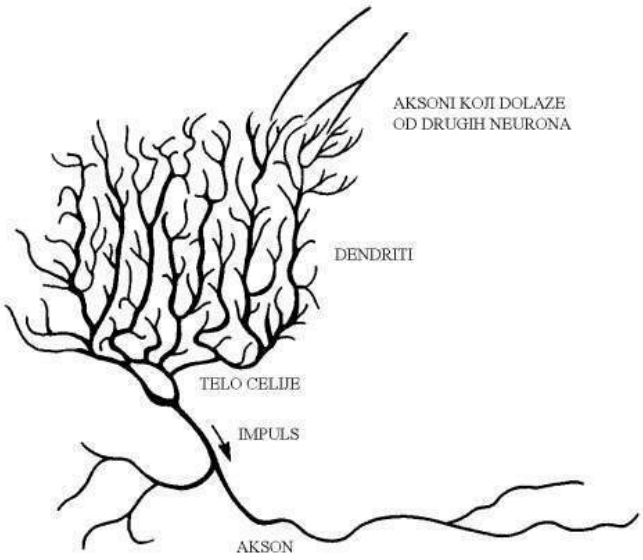
U ovoj knjizi, kada se govori o NM, se prvenstveno misli na "vještačke neuronske mreže" (*Artificial Neural Networks - ANN*), zbog toga što se uglavnom govori o modelima NM, realizovanim na računarima. Biološke NM su daleko komplikovanije od svojih matematičkih modela, koji se koriste za vještačke NM. Neuronske mreže predstavljaju sistem sastavljen od veoma velikog broja jednostavnih elemenata za obradu podataka. Ovakvi sistemi su sposobni za prikupljanje, memorisanje i korištenje eksperimentalnog znanja. Ne postoji jedinstvena definicija NM, međutim, može se izdvojiti slijedeća definicija:

Neuronske mreže su sistemi sastavljeni od više jednostavnih procesora (jedinica, neurona), gdje svaki od njih ima lokalnu memoriju u kojoj pamti podatke koje obrađuje. Te jedinice su povezane komunikacionim kanalima (vezama). Podaci koji se ovim kanalima razmjenjuju su obično numerički. Jedinice obrađuju samo svoje lokalne podatke i ulaze koje primaju preko konekcije. Ograničenja lokalnih operatora se mogu otkloniti putem treninga.

Osnovna jedinica nervog sistema je **nervna ćelija** ili **neuron**. Ona ima četiri osnovna dijela:

- ulazni dio ćelije,
- tijelo ćelije,
- izlazni dio ćelije, i
- sinapse.

Ulazni dio ćelije sadrži skup razgranatih niti nazvanih ***dendriti***. **Tijelo ćelije** obrađuje signale koje dobija od dendrita, na taj način dobijajući izlazni impuls koji se proslijeđuje na sve krajeve razgranate niti nazvane ***aksonom***, koji predstavlja **izlazni dio ćelije**. Mjesto gde se akson dodiruje sa dendritima neke druge ćelije se naziva **sinapsa**. To je mjesto gde se impulsi prenose od jedne do druge nervne ćelije. Biološki neuron je prikazan na slici 10.1.



Slika 10.1. Biloški neuron.

Učenje se kod bioloških sistema obavlja putem regulisanja sinaptičkih veza, koje povezuju aksone i dendrite neurona. Učenje tipičnih događaja putem primjera se ostvaruje preko treninga ili otkrića do tačnih setova podataka ulaza - izlaza, koji treniraju algoritam ponavljanjem podešavajući propusne (težinske) koeficijente

veza (sinapse). Ove veze memorišu znanje neophodno za rješavanje specifičnog problema.

Većina NM ima neku vrstu pravila za "obučavanje", čime se koeficijenti veza između neurona podešavaju na osnovu ulaznih podataka. Drugim riječima, NM "uče" preko primjera (kao što djeca uče da prepoznaju konkretni predmet, objekat, proces ili pojavu preko odgovarajućih primjera) i posjeduju sposobnost za generalizaciju nakon trening podataka.

10.2. VJEŠTAČKE NEURONSKE MREŽE

Vještačke neuronske mreže su vrsta računara koji se potpuno razlikuju od tradicionalnih, klasičnih, računara sa *Von Neumann*-ovom arhitekturom. Kod klasičnog računara sa *Von Neumann*-ovom arhitekturom jedan centralni procesor sekvencijalno obavlja instrukcije dane programom. Pri tome procesor može da obavlja stotinu i više osnovnih naredbi, kao što su sabiranje, oduzimanje, množenje, punjenje, pomjeranje i drugo.

U NM, gde su procesorske jedinice povezane određenom topologijom, postoji struktura paralelnog distribuiranog procesiranja (PDP). Istovremeno rade više procesorskih jedinica, da bi rezultati njihove obrade PDP strukturom prešli na druge jedinice, itd. Procesorske jedinice u jednoj NM su jednostavne i mogu obavljati samo jednu ili eventualno nekoliko računskih operacija i međusobno su povezane tako da u jednoj NM postoji mnogo više veza nego procesorskih jedinica. Svaka jedinica je povezana sa više susjednih jedinica. Obično je taj broj iznad stotinu, pa čak i do hiljadu. Broj ovih veza između procesorskih jedinica ustvari predstavlja snagu NM. Rad NM sa više veza je kompleksniji, ali takva mreža može da odgovara na kompleksnije zadatke koji joj se zadaju. Analogan broju veza u NM, koji predstavlja snagu procesiranja ove mreže, je broj instrukcija koje centralni procesor u klasičnom računaru obrađuje u sekundi.

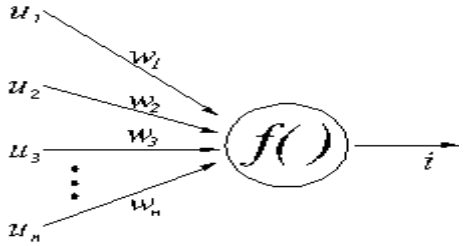
Postoji još jedna bitna razlika u odnosu na klasične računare: NM se ne "programiraju". Dok kod klasičnog računara programer unosi u računar program, kojim je tačno određen rad računara u svakom trenutku, NM se ne programiraju već se "obučavaju". Prije nego što se počnu primenjivati, ulaže se dosta vremena za obučavanje, učenje ili treniranje NM. Proces obučavanja se najčešće zasniva na

ažuriranju **težinskih koeficijenata veza**, a ponekad samo vrijednosti ulaznih procesorskih jedinica. Težinski koeficijenti veza ili težine veza su koeficijenti koji su dodjeljeni u svakom trenutku vezama NM. Za vrijeme obučavanja ovi se koeficijenti ažuriraju. Brzina njihovog ažuriranja može također biti mjerilo moći procesiranja NM.

Neuronske mreže obezbjeđuju efikasan pristup širokom spektru primjena. Njihova uspješna primjena u prepoznavanju, preslikavanju i klasifikaciji oblika omogućila je dalji razvoj NM. Takođe, ove mreže se mogu primjenjivati u transformisanju slike u neki drugi oblik podataka, ili preslikavanju vizuelne slike u komande robota. Kao što je već napomenuto, rad sa nepotpunim podacima, omogućava da upravo mreža upotpunjuje takve podatke, pošto je prethodno za to obučavana. Neuronske mreže se mogu primjeniti u različitim oblastima. Na primjer, u medicini: kod analize krvnih ćelija; u akustici: kao klasifikator zvučnih slika ili kod prepoznavanja govora, gdje je neophodna identifikacija i klasifikacija riječi ili sekvenci riječi, itd.

10.2.1. Težinski koeficijenti veza

Sinapse, kojima biološki neuroni regulišu prohodnost određene putanje između aksona i dendrita, kod vještačkih neurona se ostvaruju preko prilagodljivih težinskih koeficijenata (*weight*) ili težina veza. Kada se na ulaz neurona dovedu neke vrijednosti i pomnože težinskim koeficijentima, dobijaju se ulazni podaci. Zbir ulaznih vrijednosti neurona pomnoženih sa odgovarajućim težinskim koeficijentima se propušta kroz aktivacionu funkciju i ta vrijednost predstavlja izlaz iz neurona. Iako neuroni imaju prilično jednostavne (linearne) funkcije, kada se povežu u višeslojnu mrežu, u stanju su da obrade veoma složene (nelinearne) funkcije. Na slici 10.2. je prikazan model vještačkog neurona.



Slika 10.2. Model vještačkog neurona.

gdje su:

$u_{1...n}$ – ulazni podaci,
$w_{1...n}$ – težinski koeficijenti,
$f( )$ – aktivaciona funkcija, i
i – izlazni podatak.

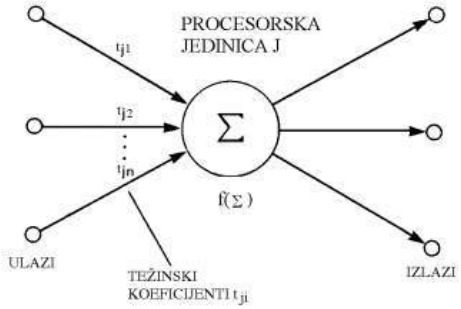
Neuroni na skrivenim i izlaznim slojevima pored težinskih koeficijenata koriste i koeficijent "threshold" ("bias") u računanju mrežnih ulaznih vrijednosti. Koeficijent *threshold* se može tretirati kao dodatni težinski koeficijent na ulazu, koji ima konstantnu težinu jedan. Rukovanje ovim koeficijentom je slično kao i sa svakim drugim težinskim koeficijentom.

Prirodni neuroni su znatno komplikovaniji od vještačkih. Iako su vještački neuroni, izvedeni u VLSI tehnologiji, znatno brži od prirodnih, visok stepen međusobne povezanosti, njihov ogroman broj i još veći broj veza između njih, čine biološke nervne sisteme nedostižnim za današnju tehnologiju i nepotpuno razumljivim za današnju nauku. Uz to, mala je vjerovatnoća da će principijelna šema stotine milijardi veza biti za dogledno vrijeme analizirana. Šta više, još uvijek nije poznato kako da se protumače težinski koeficijenti, čak i u mrežama od samo nekoliko neurona.

10.2.2. Osnovna struktura neuronske mreže

Kao što je već rečeno, NM se sastoji od više jednostavnih procesorskih jedinica. Jedna od takvih procesorskih jedinica je prikazana na slici 10.3. Na lijevoj strani se nalaze ulazi. Signali dolaze od drugih procesorskih jedinica do procesorske jedinice **j** preko ulaznih veza. Svaka veza ima dodjeljen težinski koeficijent tj., pri čemu indeksi označavaju da se signal kreće od **i**-te do *j*-te procesorske jedinice. Elementarna procesorska jedinica signale t_{i1}, t_{i2}, ... , t_{in}, koji dolaze od drugih procesorskih jedinica, sabira i množi sa odgovarajućim težinskim koeficijentima. Na taj način procesorska jedinica pravi težinsku sumu nad ulazima i koristi nelinearnu prag (*threshold*) funkciju *f* za izračunavanje izlaza. Rezultat izlaza se prosljeđuje preko izlaznih veza do drugih procesorskih jedinica.

Ulazni sloj se sastoji od ulaznih procesorskih jedinica. Svakoj jedinici se dodjeljuje određena vrijednost u zavisnosti od vrijednosti pojedinih elemenata niza, koji predstavlja ulazni oblik. Sa svojih izlaza, ove procesorske jedinice prosljeđuju signale do slijedećeg sloja prema topologiji dane mreže.



Slika 10.3. Vještački neuron.

Ove se jedinice nazivaju i **detektorima osobina**, jer odgovaraju posebnim osobinama koje se mogu pojaviti u ulaznom obliku. Izlazni sloj procesorskih jedinica na svojim izlazima daje vrijednosti koje se koriste kao izlazni oblici.

Neuronsku mrežu sačinjavaju:

- arhitektura (topologija) mreže, odnosno šema vezivanja neurona,
- prenosna funkcija neurona, i
- zakoni učenja.

Arhitekturu vještačke neuronske mreže predstavlja specifično uređenje i povezivanje neurona u obliku mreže. Po arhitekturi, neuronske mreže se razlikuju prema broju neuronskih slojeva. Obično svaki sloj prima ulaze iz prethodnog sloja, a svoje izlaze šalje narednom sloju. Prvi sloj se naziva ulaznim, posljednji izlaznim, ostali slojevi se obično nazivaju skrivenim slojevima. Jedna od najčešćih arhitektura neuronskih mreža je mreža sa tri sloja. Prvi sloj (ulazni) je jedini sloj koji prima signale iz okruženja.

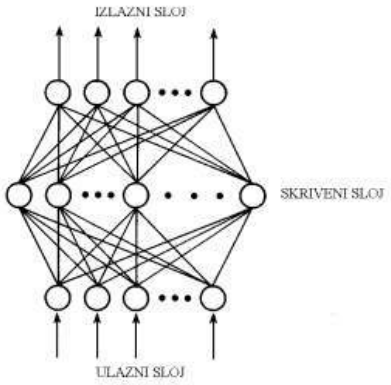
Prvi sloj **prenosi signale** slijedećem sloju (skrivenom sloju), koji obrađuje ove podatke i izdvaja osobine i šeme iz primljenih signala. Podaci koji se smatraju važnim se upućuju izlaznom sloju, posljednjem sloju mreže. Na izlazima neurona trećeg sloja se dobijaju konačni rezultati obrade. Složenije NM mogu imati više skrivenih slojeva, povratne petlje i elemente za vremensko odlaganje, koji su dizajnirani da omoguće što efikasnije odvajanje važnih osobina ili šema sa ulaznog nivoa.

Učenje NM se svodi na učenje iz primjera, kojih treba da bude što više da bi mreža mogla da se ponaša tačnije u kasnijoj eksploataciji. Proces učenja dovodi do korigovanja sinaptičkih težina. Kada uzorci koji se predstavljaju mreži ne dovode više do promene ovih koeficijenata, smatra se da je mreža obučena.

Postoji tri tipa obučavanja:

- Nadgledano obučavanje: mreži se predstavljaju ulazni podaci i očekivani izlazni podaci;
- Obučavanje ocjenjivanjem: mreži se ne predstavljaju očekivani izlazni podaci, nego joj se poslije izvjesnog vremena predstavlja ocjena prethodnog rada. Primjer mreže koja uči da balansira štap. Kad god štap padne, mreži se proslijeđuje ocjena prethodnog rada, na primjer, u obliku ugaonog odstupanja štapa od ravnoteže;
- Samoorganizacija: mreži se predstavljaju isključivo ulazi.

Srednji sloj sadrži procesorske jedinice, čija obrada podataka u potpunosti odgovara onoj opisanoj na slici 10.4.



Slika 10.4. Vještačka neuronska ćelija sa tri puna povezana sloja.

10.3. OBUČAVANJE NEURONSKIH MREŽA

10.3.1. Opšte o obučavanju neuronskih mreža

U svim biološkim NM veze između pojedinačnog dendrita i aksona mogu biti pojačane ili oslabljene. Na primjer, veze mogu postati pojačane ako se više signala šalje kroz njih, ili mogu biti oslabljene ako se signali kroz njih rjeđe šalju. Pojačavanje određenog neuralnog prolaza, ili veze između dendrita i aksona, rezultuje u povećanoj vjerovatnoći da će signal biti prenesen kroz tu putanju, daljim pojačavanjem tog puta. Putevi između neurona koji su rijetko korišteni polako atrofiraju, ili se umanjuju, praveći manju vjerovatnoću da će signal biti kroz njih prenesen.

Slična situacija se pojavljuje i kod vještačkih neurona. Podaci iz trening skupa se periodično propuštaju kroz NM. Dobijene vrijednosti na izlazu mreže se upoređuju sa očekivanim. Ukoliko postoji razlika između dobijenih i očekivanih podataka, prave se modifikacije na vezama između neurona u cilju smanjivanja razlike trenutnog i željenog izlaza. Ulazno - izlazni skup se ponovo predstavlja mreži zbog daljih podešavanja težina, pošto u prvih nekoliko koraka mreža obično daje pogrešan rezultat. Poslije podešavanja težina puta za sve ulazno izlazne šeme u trening skupu, mreža nauči da reaguje na željeni način.

Neuronska mreža je obučena ako može tačno da rješava zadatke za koje je obučavana. NM je sposobna da izdvoji važne osobine i šeme u klasi trening primjera. Nakon obučavanja sa određenom vjerovatnoćom, NM može da generalizuje nove ulazne podatke za koje nije obučavana. Na primer, generalizaciju možemo vidjeti na primjeru mreže obučavane da prepoznaje serije slika: ako na ulaz takve mreže dovedemo slike za koje mreža nije obučavana, ona do izvjesne mjere može uspješno da klasificira takve slike.

Najčešće korišten algoritam za obučavanje NM je propagacija greške unazad *(backpropagation)*, razvijen nezavisno od strane naučnika: *Paul Werbos*-a (1974), *David Parker*-a (1984/1985), *David Rumelhart*-a, *Ronald Williams*-a i drugih (1985). *Backpropagation* uči šeme poredeći izlaz NM sa željenim izlazom i računa greške za svaki čvor u mreži. Neuronska mreža podešava težine veza prema vrijednostima greške dodjeljenim za svaki čvor. Izračunavanje počinje od izlaznog sloja, preko skrivenih slojeva, prema ulaznom sloju. Nakon modifikacije parametara, na mrežu se dovode novi ulazi. Obučavanje se prekida tek kada mreža bude u stanju da daje izlaze sa zadovoljavajućom tačnošću.

Proces obučavanja NM započinje zadavanjem slučajnih vrijednosti težinskim koeficijentima veza i dovođenjem oblika na ulazni sloj. Nakon toga se mreža aktivira i upoređuju se zadani i izlazni oblici (zahtjevani, oni koji treba da se dobiju). Obučavanje se vrši tako da se ažuriraju težinski koeficijenti sa ciljem da se u slijedećoj iteraciji dobija izlaz koji je bliži zadanoj vrijednosti. U trenutku kad se postigne zadovoljavajući rezultat sa jednim ulaznim oblikom, na ulaz se dovodi drugi, itd. Kada se završi sa svim oblicima iz **obučavajućeg skupa 1**, na ulaz mreže se dovodi ponovo prvi ulazni oblik. Ova se procedura nastavlja sve dok se ne dođe do zadovoljavajućih rezultata za sve oblike iz obučavajućeg skupa. Kada je obučavanje mreže završeno, težinski koeficijenti veza ostaju nepromenjeni. Tek tada, nakon obučavanja, mreža se može primjeniti za predviđeni zadatak.

Mada je u najviše slučajeva mreža sa adaptivnim težinskim koeficijentima veza, postoje mreže čije su težine fiksne i za vrijeme obučavanja. Tada se mijenjaju samo aktivacioni nivoi pojedinih neurona. Opisani proces se naziva obučavanje sa nadgledanjem (*supervised*). Dobijeni izlaz se upoređuje sa zadanim i onda se na osnovu dobijene razlike obavlja ažuriranje pojedinih težinskih koeficijenata.

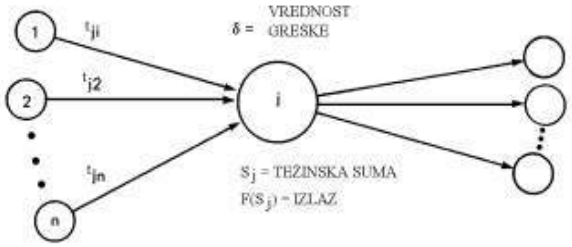
Nasuprot takvom načinu obučavanja postoji i obučavanje bez nadgledanja (*unsupervised*). Izlazna vrijednost se ne upoređuje sa zadanom vrednošću. Ovom metodom mreža klasifikuje ulazne oblike u više grupa. Broj jedinica u izlaznom sloju odgovara broju različitih grupa, što znači da u jednom datom trenutku postoji samo jedan aktivni izlaz.

10.3.2. Propagacija greške unazad

Propagacija greške unazad (*back - propagation*) je od svih paradigmi neuronskih mreža najviše korištena. Propagacija unazad uspješno rješava probleme preslikavanja oblika: za dani ulazni oblik mreža proizvodi asocirani izlazni oblik. Propagacija greške unazad se bazira na relativno jednostavnom principu: kada mreža da pogrešan rezultat, težinski koeficijenti se ispravljaju tako da greška bude manja, a kao rezultat toga, slijedeći odgovori mreže su bliže tačnim. U principu, propagacija unazad koristi tri ili više slojeva procesorskih jedinica. Na slici 10.5. je prikazana tipična topologija troslojne mreže sa propagacijom unazad.

Mreža sa propagacijom unazad mora imati najmanje dva sloja. Ukoliko ima više slojeva, pored ulaznog i izlaznog, ostali su skriveni. Najčešće se koristi mreža sa potpuno povezanim slojevima, što znači da je svaki sloj povezan sa prethodnim i slijedećim.

Na slici 10.5. je predstavljena jedna osnovna procesorska jedinica u mreži sa propagacijom unazad. Ona je okarakterisana težinskom sumom ulaza S_i, izlaznom vrijednošću a_i i vrijednošću greške d_i, koja se koristi pri ažuriranju težinskih koeficijenata veza. Aktivacioni nivo (*activation level*) koji je dodjeljen svakoj od jedinica procesiranja u mreži sa propagacijom unazad predstavlja vrijednost izlaza iz odgovarajuće jedinice.



Slika 10.5. Procesorska jedinica mreže sa propagacijom unazad.

Neuronske mreže sa propagacijom unazad se obučavaju sa nadgledanjem. Mreži se predstavlja jedan par obučavajućeg skupa (ulazni oblik uparen sa zadanim izlazom). Poslije svakog predstavljanja, težine se ažuriraju tako da se smanjuje razlika između izlaza mreže i zadanog (ciljnog) izlaza. Obučavajući skup, se predstavlja, sve dok se mreža ne obuči, ili prekine proces obučavanja. Nakon što se obučavanje završi, vrši se testiranje preformanse mreže.

Algoritmi za obučavanje mreže sa propagacijom greške unazad sadrže korak propagacije unaprijed, za kojim slijedi korak propagacije unazad. Oba koraka se primjenjuju za svako predstavljanje para obučavajućeg skupa. Korak propagacije unaprijed počinje sa prikazom ulaznog oblika na ulaznom nivou mreže i nastavlja tako što aktivacioni nivoi vrše propagaciju unaprijed izračunavanja kroz skrivene nivoe. U svakom sukcesivnom nivou svaka procesorska jedinica sumira svoje ulaze i primjenjuje prag funkciju u izračunavanju izlaza. Jedinice izlaznog nivoa daju izlaz mreže.

Korak propagacije unazad počinje poređenjem izlaznog oblika mreže sa ciljnim vektorom. Tako se izračunava razlika ili greška. Korak propagacije unazad tada izračunava vrijednosti greške za skrivene jedinice i mijenja njihove ulazne težine, polaze i od izlaznog nivoa kroz sve sukcesivne skrivene nivoe. U ovom

koraku propagacije unazad, mreža ispravlja svoje težine tako da smanjuje uočenu grešku. Vrijednost greške pridružena svakoj procesorskoj jedinici se koristi za vrijeme izvršenja procedure korekcije težina tokom učenja. Velika vrijednost označava da treba sprovesti veću korekciju nad ulaznim težinama, a njen znak odražava pravac u kome treba mijenjati težine.

10.3.3. Propagacija unaprijed

Korak propagacije unaprijed se inicijalizira kada se oblik predstavlja mreži. Svaka ulazna jedinica odgovara jednom elementu vektora ulaznog oblika, i svaka jedinica dobija vrijednost tog elementa. Pošto su aktivacioni nivoi prvog sloja procesorskih jedinica postavljeni, ostali slojevi nastavljaju korak propagacije unaprijed i na taj način određuju aktivacione nivoe drugih slojeva. Prvi korak propagacije unapred je izračunavanje težinske sume. Jedinica *j* izračunava težinsku sumu svojih ulaza:

$$S_j = \sum_i^n a_i \, t_{ji} \tag{10.1}$$

gde su:
a_i = aktivacioni nivo jedinice *i,*
t_{ji} = težinski koeficijenat veze koja spaja jedinice *i* i *j*.

Jedinica *i* pripada jednom sloju ispred sloja jedinice *j*. Pošto se izračuna suma (10.1), nalazi se funkcionalno preslikavanje sume $f(S_j)$. Funkcija *f* je sigmoidna funkcija i data je na slici 10.6(a). Njena jednačina glasi:

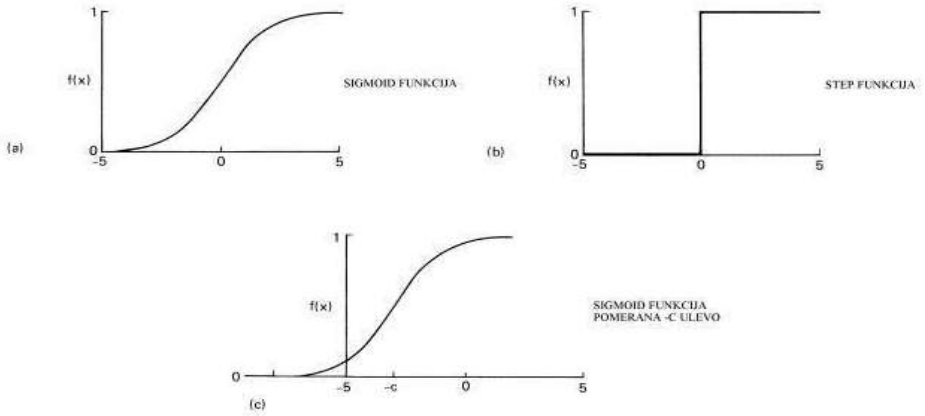
$$f(x) = \frac{1}{1+e^{-x}} \tag{10.2}$$

Funkcija (10.2) predstavlja "mekšu" varijantu step funkcije prikazane na slici 10.6.(b). Koristeći funkciju (10.2) dobija se:

$$f(S_j) = \frac{1}{1+e^{-Sj}} = \frac{1}{1+e^{-\Sigma a_i t_{ji}}} \tag{10.3}$$

Vrijednost izračunata pomoću formule (10.3) postaje aktivacioni nivo jedinice *j* i proslijeđuje se na sve njene izlaze. Ulazni sloj jedinica predstavlja specijalni

slučaj. Ove jedinice ne sabiraju nikakve vrijednosti, već jednostavno proslijeđuju ka slijedećem nivou one vrijednosti koje primaju od vektora ulaznog oblika.



Slika 10.6. Prag funkcije.

Neke mreže sa propagacijom unazad koriste ***bias*** (engleska riječ *bias* se može prevesti kao *uticaj*) jedinicu u svakom, izuzev izlaznog sloja. Ova jedinica ima konstantnu aktivacionu vrijednost. Svaka *bias* jedinica je povezana sa svim jedinicama u slijedećem višem sloju, a težine tih veza se ažuriraju tokom propagacije unazad. *Bias* jedinice obezbjeđuju konstantan član u težinskim sumama jedinica narednog sloja. Kao rezultat toga dobijaju se poboljšane konvergentne karakteristike mreže. Konstantan član u sumi S_k omogućava translaciju sigmoidne funkcije u ljevo ili u desno, kao što je prikazano na slici 10.6(c).

***PRIMJER*:**
Neka je izlazna vrijednost bias jedinice $a_0 = 1.0$, a težinski koeficijenat

$$\mathbf{C = t_{k0}} \tag{10.4}$$

Ako se sa ***z*** *obilježi težinska suma svih ulaza izuzev bias ulaza, dobija se*

$$\mathbf{z} = \sum_{i=1}^{n} \mathbf{a_i t_{j\,i}} \tag{10.5}$$

Uzimajući u obzir posljednje dvije jednačine, jednačina (10.1) postaje

$$\mathbf{S}_j = \mathbf{z} + \mathbf{C} \tag{10.6}$$

pa izlaz jedinice $\boldsymbol{f(S_j)}$ *predstavlja pomjerenu funciju* $\boldsymbol{f}$ *za* $\boldsymbol{c}$*, kao na slici 10.6(c). Na ovaj način bias jedinice obezbeđuju prag za svaku ciljnu jedinicu.*

10.3.4. Propagacija unazad

Na slici 10.7. su prikazani koraci propagacije unazad. Izračunavaju se greške za svaku od jedinica počev od jedinica u izlaznom sloju sve do ulaznog sloja, vraćajući se unazad kroz mrežu. Korekcija greške počinje u izlaznom sloju pošto je mreži predstavljen ulazni oblik i završen korak propagacije unaprijed. Izlaz svake jedinice izlaznog sloja se upoređuje sa zadanom vrijednošću specificiranom u obučavajućem skupu, a zatim se na osnovu toga za svaku jedinicu izračunava vrijednost greške. Nakon toga se ažuriraju težinski koeficijenti veza između jedinica u izlaznom sloju i sloju koji se nalazi neposredno ispred njega.

Nakon toga se izračunava greška za svaku jedinicu sloja, koji se nalazi neposredno ispred izlaznog, što se može vidjeti i na slici 10.7(c). Na kraju se ažuriraju težinski koeficijenti veza koji idu u skriveni sloj. Proces se nastavlja sve dok posljednji nivo težina ne bude podešen, odnosno dok se ne dođe do ulaznog sloja.

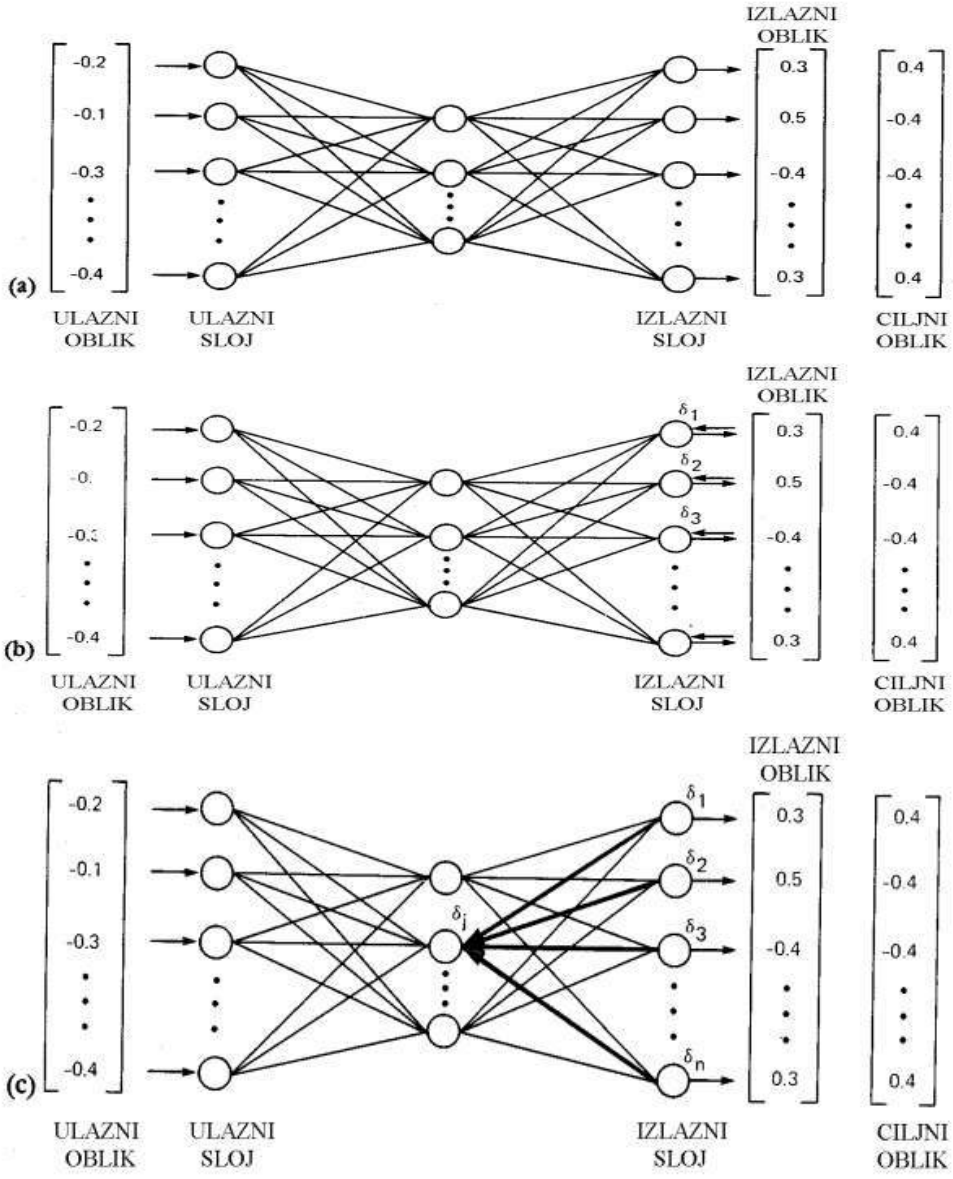
Obrazac za izračunavanje greške je jednostavan za jedinice iz izlaznog sloja, a nešto složeniji za jedinice iz ostalih (skrivenih) slojeva. Ako je jedinica i izlazna, tada je vrednost njene greške:

$$\delta_i = (z_i - a_i)\, f'(S_i) \tag{10.7}$$

gdje su:

z_i = zadana vrijednost izlaza za jedinicu i,
a_i = vrijednost izlaza jedinice i,
$f'(S_i)$ = prvi izvod funkcije f,
S_i = težinska suma ulaza u jedinicu i.

Razlika određuje iznos greške, a prvenstveno njen znak. Prvi izvod funkcije f, $f'(S_i)$, mijenja korekciju greške; ukoliko funkcija f (uzimajući u obzir da je f sigmoidna funkcija) brže raste, tada treba izvršiti veću korekciju i obrnuto.



Slika 10.7. Korak propagacije unazad.

Ako je procesorska jedinica i skrivena, greška se izračunava pomoću obrasca

$$\delta_i = S_{k=1,m}\, \delta_k\, t_{ki}\, f'(S_i) \qquad (10.8)$$

gdje je:

m = broj procesorskih jedinica koje prihvataju izlaze jedinice i.

Prvi izvod funkcije f, i u ovom slučaju, određuje veličinu korekcije greške. Koristeći grešku δ_i ažuriraju se težinski koeficijenti veza. Obrazac za ažuriranje težinskih koeficijenata veze koja spaja jedinice k i i, pri čemu je jedinica k ispred jedinice i, je dat sa:

$$\delta_{tik} = h\, \delta_{iak} \qquad (10.9)$$

gdje su:

h = konstanta (brzina) obučavanja,

δ_i = greška jedinice i,

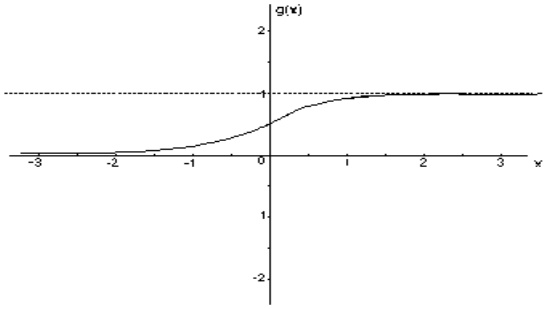
ak = aktivacioni nivo jedinice k.

Jednačina (10.9) predstavlja **generalisano delta pravilo**. Konstanta (brzina) obučavanja se bira najčešće u opsegu 0.25 i 0.75. Ona određuje brzinu obučavanja mreže, ako je veća, obučavanje je brže i obrnuto. Međutim, prevelike vrijednosti ovog koeficijenta djeluju destabilišujuće na mrežu i mogu da uzrokuju neobučavanje mreže. S druge strane, ako je isuviše mala, obučavanje je sporo. Ponekad se koristi efikasnija metoda, **metoda promjenljivog koeficijenta**. Vrijednosti koeficijenta koje su veće na početku se mogu smanjivati tokom obučavanja, omogućavajući tako mreži bolje karakteristike obučavanja.

10.3.5. Aktivacione funkcije

Aktivacione funkcije neurona na skrivenim slojevima su potrebne da bi mreža bila u stanju da nauči nelinearne funkcije. Bez nelinearnosti, neuroni skrivenih slojeva ne bi imali veće mogućnosti od obične perceptronske mreže, koja se sastoji samo od ulaza i izlaza. Kombinovanjem linearnih funkcija se ponovo dobija linearna funkcija. Zbog toga se na izlazu neurona nalazi aktivaciona funkcija koja je najčešće nelinearna. Ova nelinearnost čini mreže sa više slojeva naročito moćnim. Gotovo svaka nelinearna funkcija može da se koristi, mada se za *backpropagation algoritam* najčešće koriste *sigmoidne funkcije* kao što su *logistička*, *arcustangens* ili *Gausova funkcija*.

Za neurone na izlaznom sloju se mogu birati aktivacione funkcije koje odgovaraju raspodjeli ciljnih vrijednosti. Granične aktivacione funkcije, kakva je logistička, su naročito korisne kada su ciljne vrijednosti ograničene. Ali ako ciljne vrijednosti nemaju ograničene vrijednosti, bolje je da se koristi aktivaciona funkcija koja nije ograničena. Ako su ciljne vrijednosti pozitivne, ali nemaju gornju granicu, najbolje je koristiti eksponencijalnu aktivacionu funkciju. Postoji izvjesna prirodna povezanost između izlaznih aktivacionih funkcija i različite raspodjele šuma koji se izučava statistički u kontekstu generalizacije izlaznog modela, slika 10.8.



Slika 10.8. Aktivaciona funkcija.

10.4. REALIZACIJA NEURONSKIH MREŽA

Neuronska mreža se može realizovati na dva načina:

- hardverski, i
- softverski.

Kod hardverske realizacije vještački neuroni su fizički međusobno povezani, oponašajući veze između bioloških neurona. Neuroni se realizuju kao jednostavna integrisana kola. Kod softverske realizacije NM se obično simuliraju na tradicionalnim računarima, u kojima je veza između čvorova logička (virtuelna). Svaki od ovih načina realizacije NM ima svoje prednosti i mane.

Prednost hardverske realizacije je u tome što može da koristi mogućnost paralelnog procesiranja informacija, ukoliko se svakom neuronu u mreži dodjeli po jedan procesor. Prednost softverske realizacije NM, na standardnom PC računaru, je u tome što se lakše uspostavljaju, a kasnije i mijenjaju, veze između pojedinih

neurona u mreži. U praksi se softverska realizacija koristi za testiranje, a konkretna realizacija koja se primenjuje u praksi može biti realizovana i hardverski, čime se dobija na brzini.

10.4.1. Podjela neuronskih mreža

Postoji veliki broj različitih realizacija neuronskih mreža, a samim tim postoji i mnogo podjela. Može se izvršiti slijedeća klasifikacija NM:

- prema broju slojeva,
- prema vrsti veza između neurona,
- prema vrsti obučavanja neuronskih mreža,
- prema smjeru prostiranja informacija, i
- prema vrsti podataka.

Najopštija podjela NM je prema broju slojeva. Mreže se mogu podjeliti na:

- jednoslojne, i
- višeslojne.

Danas se uglavnom izučavaju i primjenjuju višeslojne NM, koje pored ulaznih i izlaznih slojeva sadrže neurone na srednjim, skrivenim, slojevima.

10.4.2. Mogućnosti neuronskih mreža

Teorijski se neuronske mreže mogu obučiti za izračunavanje svake izračunljive funkcije. One mogu uraditi sve što može da uradi savremeni digitalni računar. Međutim, u praksi NM najbolje rezultate pokazuju na:

- području klasifikacije,
- području aproksimacije funkcija,
- na problemima mapiranja, sa nepreciznim podacima,
- na problemima koji imaju dosta dostupnih podataka za trening,
- na problemima koji zahtjevaju brzu primjenu odgovarajućeg pravila u zavisnosti od ulaznih podataka,
- preciznoj prostornoj aproksimaciji vektora.

Neuronske mreže ne mogu da stvore informaciju koju ne sadrže trening podaci.

10.4.3. Razlike između NM i klasičnih računara

Neuronska mreža se razlikuje od klasičnih računara (PC računara, radnih stanica i *mainframe* računara) u formi i funkcionisanju. Dok neuronska mreža koristi veliki broj jednostavnih procesora da bi obavila njene kalkulacije, klasični računari koriste jedan ili, u rjeđim slučajevima, svega nekoliko veoma kompleksnih procesorskih jedinica. Neuronska mreža ne posjeduje centralno lokalizovanu memoriju, niti se programira sekvencama instrukcija, kao svi klasični računari. Klasični računari koji rade na binarnoj logičkoj osnovi, koriste algoritamski način obrade podataka, sekvencijalni, sa veoma niskim stepenom paralelizacije. U algoritamskom načinu obrade podataka računar obrađuje jednu po jednu informaciju ili u boljem slučaju obrađuje manji broj informacija u isto vrijeme.

Za razliku od ovog pristupa obrade podataka, NM procesira istovremeno više informacija, tj. najbolja varijanta za NM je da svaki neuron predstavlja jedan procesor. Razvoj NM je doveo do novih arhitektura računara koji se umnogome razlikuju od računara kakvi su danas rasprostranjeni. Ako bi se posmatrao primjer prepoznavanja slova, algoritamsko rješenje bi zahtjevalo da se zadano slovo uporedi sa svim slovima u bazi podataka, slovo po slovo, dok NM može da uporedi zadano slovo istovremeno sa svim slovima, a rješenje je slovo sa najvećom vjerovatnoćom. Ovo je moguće jer se memoriji pristupa uz pomoć sadržaja, a ne adrese.

Kod klasičnih računara su elementi obrade informacija i elementi memorisanja informacija potpuno odvojene komponente. Kod neuronske mreže memorisanje i obrada predstavljaju jednu kompaktnu cjelinu. Podaci koji su vezani za rad NM nemaju nikakav smisao bez jedinica obrade. Neuronska mreža se razlikuje od klasičnih računara po načinu na koji se "programira". Umjesto programa napisanih kao serije instrukcija, kao što je slučaj kod klasičnih računara, upotrebljava se obučena NM, gdje arhitektura i težinski koeficijenti određuju njenu funkciju. Koeficijenti se podešavaju tokom obučavanja na ograničenom skupu karakterističnih primjera. Kada se mreža obuči do zadovoljavajuće granice, vrijednosti veza se mogu memorisati i koristiti u kasnijem radu.

Kod klasičnih računara softver mora biti gotovo savršen da bi radio. Razvoj softvera zahtjeva iscrpan dizajn, testiranje i postepeno usavršavanje čine ga dugim i skupim procesom. NM omogućavaju evolutivni razvoj softvera, tj. NM se mogu naknadno adaptirati realnim i novo nastalim uslovima. Neuronske mreže imaju

sposobnost da mijenjaju svoju strukturu i funkciju, za razliku od klasičnih algoritama koji nemaju toliku fleksibilnost.

Decentralizovana obrada i memorisanje omogućavaju NM da nastavi funkcionisanje i u uslovima kada se dio mreže ošteti, jedan dio neurona prestane da funkcioniše ili se neke veze pokidaju. Oštećena mreža će i dalje biti u stanju da funkcioniše, ali sa smanjenom tačnošću. Mreža je, takođe, tolerantna i na prisustvo šuma u ulaznom signalu. Svaki memorisani uzorak je delokalizovan, tj. smješten je u cijelu mrežu.

10.4.4. Načini implementacije neuronskih mreža

Neuronske mreže su obično simulirane na klasičnim računarima. Prednost ovog pristupa je u tome što se računari mogu lako reprogramirati da promjene arhitekturu ili pravilo učenja simulirane NM. Računanje u NM je uglavnom paralelno, tako da brzina obrade simulirane NM može biti znatno uvećana korištenjem paralelnih procesora.

Neuronske mreže i klasično programiranje mogu se posmatrati kao fundamentalno različiti, ali komplementarni prilazi obradi informacija. NM su zasnovane na transformacijama, dok je programiranje zasnovano na algoritmima i pravilima.

10.5. PODRUČJA PRIMJENE NEURONSKIH MREŽA

U početku su NM koristili naučnici računarskih i kognitivnih nauka koji su pokušavali da modeliraju čulni sistem živih organizama. Danas NM predstavljaju veoma atraktivnu oblast istraživanja i postoje brojne oblasti u kojima se koriste.

Neuronske mreže se mogu koristiti za:

- prepoznavanje video oblika,
- prepoznavanje rukopisa,
- prepoznavanje govora,
- finansijske i ekonomske simulacije,
- predviđanje poremećaja na tržištu,
- upravljanje tehnološkim sistemima,

- upravljanje proizvodnim procesima,
- analizu tehničkih rješenja,
- medicini, naročito kod psihijatrijskih oboljenja,
- kompresiju podataka u memorijama računara,
- istraživanja u naftnoj industriji,
- pomoć prilikom kriminoloških istraživanja,
- analizu medicinskih testova,
- ispitivanje EEG i EKG signala,
- pronalaženje optimalnog rješenja,
- upravljanje robotima,
- analiziranje podataka pri pirolizi i spektroskopiji,
- u bioračunarskim sistemima,
- podršku pri vremenskoj prognozu, i
- u drugima oblastima.

Na osnovu navedenog prikaza korištenje NM, može se primjena neuronskih mreža razvrstati na tri karakteristične oblasti:

- procesiranje senzorskih informacija,
- analiza podataka, i
- kontrola upravljanja.

10.6. PRIMJER EKSPERTNOG SISTEMA SA NEURONSKIM MREŽAMA

Ekspertni sistem, koji daje najbolji efekat u upravljanju valjanjem čeličnih traka u valjaonicama, se realizuje kombinacijom klasičnih matematičkih modela i neuronskih mreža. Proces valjanja čeličnih traka spada u klasu vrlo složenih tehnoloških procesa. Istraživanja su pokazala da NM uspješno preuzimaju funkciju modela, posebno funkciju adaptivnih modela. U navedenoj kombinaciji, NM i matematički model, NM se primjenjuje kao:

- parametarska mreža,
- korektivna mreža, ili
- mreža za sintezu.

10.6.1. Svojstva neuronskih mreža

U cilju detaljnijeg definisanja osnovnih pojmova u neuronskim mrežama, mogu se definisati slijedeća svojstva:

1. **Skup procesirajućih elemenata** (čvorovi ili neuroni), i to:
   - ulazni (primaju ulazne signale u neuronsku mrežu),
   - izlazni (predstavljaju rezultate rada NM),
   - skriveni (nemaju neposrednu vezu sa okolinom, nego preko ulaznih i izlaznih čvorova).

2. **Aktiviranje i izlazna funkcija:**
   Izlazna funkcija f_i svakom čvoru u trenutku t pridružuje, na osnovu aktiviranja čvora $\alpha_i\ (t)$ izlazni signal:

   $$o_i\ (t) = f_i\ (\alpha_i\ (t)).$$

3. **Struktura povezanosti čvorova:**
   U svim tipovima NM, čvorovi su brojnim vezama povezani jedan sa drugim. Obzirom da je u većini slučajeva ulaz u svaki čvor težinska suma izlaza čvorova koji su sa njim povezani, pogodno je svakoj vezi između dva čvora pridružiti težinu koja se može podešavati, uglavnom u fazi obučavanja mreže.
   Struktura povezanosti NM se može predstaviti kvadratnom matricom težina W, dim $W = N \times N$, gdje je N broj čvorova.

4. **Pravilo propagacije signala:**
   Pravilom propagacije signala se definiše način kombinovanja izlaza čvorova $o(t)$ i matrice veza W u cilju formiranja ulaznih signala čvorova. Ako se sa $net = [net_1, net_2, \ldots, net_N]^T$ označi vektor ulaza u čvorove, jedno od najčešćih pravila propagacije je

   $$net = W\, o(t),$$

   gdje je $o(t) = [o_1(t), o_2(t), \ldots, o_N(t)]$ vektor izlaza čvorova.

5. **Aktivaciona funkcija:**
   Na osnovu stanja aktivacije u prethodnom trenutku, aktivacionih funkcija i ukupnog ulaza, generiše se tekuće stanje aktivacije:

$$\alpha_j(t) = F_j(\alpha_j(t-1), f_j(\alpha_j), net_j(t)),$$

gdje je sa F_j označena aktivaciona funkcija, koja je najčešće oblika odskočne funkcije ili semilinearne, odnosno zvonolike funkcije.

6. **Algoritam obučavanja:**
   Algoritam obučavanja određuje način promjene težine matrice povezanosti u cilju željenog ponašanja NM.

Prvenstveno u zavisnosti od strukture povezanosti čvorova i načina propagacije signala kroz mrežu, razlikuje se veći broj tipova NM. Za ilustraciju rada su odabrani višeslojni perceptron i Kohonenova NM.

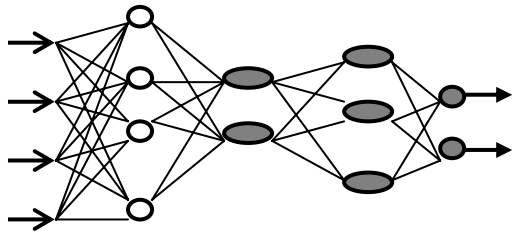
10.6.2. Višeslojni perceptron

Višeslojni perceptron je tip NM sa prostiranjem signala u jednom smjeru, sa jednim ili više slojeva između ulaznih i izlaznih neurona. Za višeslojni perceptron se mogu pisati slijedeće relacije:

$$f_i(x) = x, \qquad net_i = \sum_{j=1}^{N} W_{ji} x_j - \theta_i \qquad \alpha_i(t) = F(net_j(t)),$$

$$F(x) = 1 / (1 + \exp(-x)), \qquad o_i(t) = 1 / (1 + \exp(\textstyle\sum_j W_{ji} x_j - \theta_i))$$

Na slici 10.9. je prikazan višeslojni perceptron.



Slika 10.9. Višeslojni perceptron.

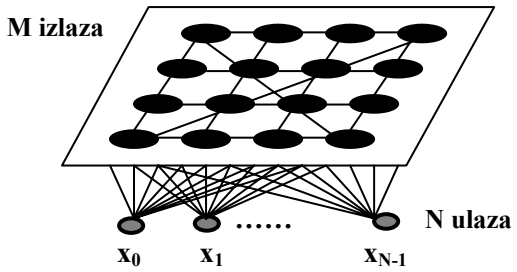
Obučavanje višeslojnog perceptrona se obavlja algoritmom propagacije greške unazad. Suština algoritma se odnosi na minimizaciju ciljne funkcije:

$$E = \Sigma_p \Sigma_I (t_i^p - o_i^p)^2,$$

gdje je *t* željeni, a *o* stvarni izlaz mreže.

10.6.3. Kohonenova neuronska mreža

Kohonenova neutronska mreža ima samo jedan sloj. Računa se rastojanje ulaznog vektora do vektora težinskih koeficijenata, odnosno do izlaznog vektora. Maksimalna aktivacija je kada se postigne nulto rastojanje. Kohenenova mreža omogućava samoobučavanje, sa nazivom samoorganizujuće preslikavanje. U suštini, ova mreža vrši kvantizaciju N dimenzionalnog ulaza u M kategorija pridruženih izlaznim neuronima, raspoređenim u linearnu rešetku. Cilj je da se kroz adaptaciju težina ostvari izomorfizam između ulaznih signala i nivoa aktivacije izlaznih neurona. Na slici 10.10. je prikazana Kohonenova vještačka neuronska mreža.

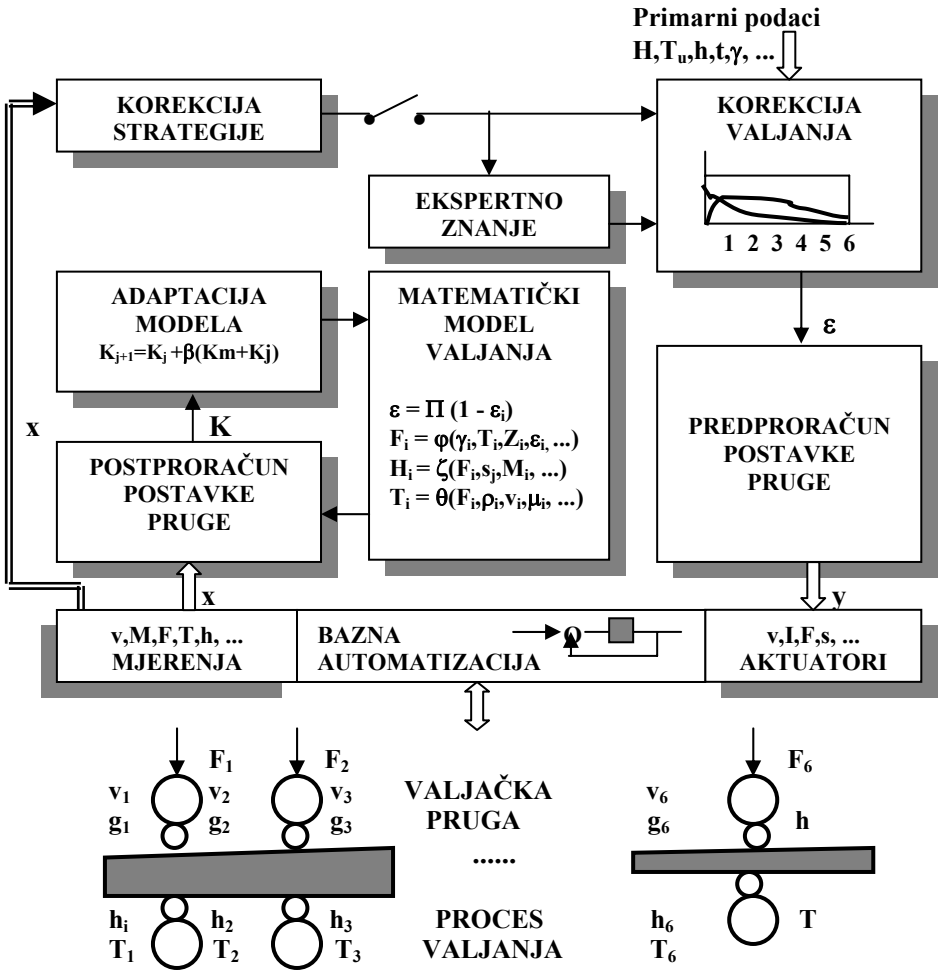


Slika 10.10. Kohonenova vještačka neuronska mreža.

Nakon obučavanja, težine Kohonenove NM teže formiranju centara klastera koji pokrivaju prostor ulaznih signala, tako da aproksimiraju njihovu funkciju gustine vjerovatnoće. Osim navedenog, težine su tako organizovane da su topološki bliski čvorovi osjetljivi na ulaze koji su slični.

10.6.4. Svojstva matematičkog modela

Matematički model valjanja čeličnih traka, kao instrument predikcije ponašanja trake pri valjanju, je uvijek samo manje ili više tačan opis procesa, a pokazalo se da bez adaptacije parametara ne može pouzdano funkcionisati. Uobičajena funkcionalna šema računarskog upravljanja valjanjem čeličnih traka izgleda kao na slici 10.11.



Slika 10.11. Funkcionalna šema računarskog upravljanja valjanjem čeličnih traka.

Na osnovu primarnih podataka o ulaznom materijalu i ciljnih parametara, u bazi podataka se obavlja izbor strategije valjanja koja se zasniva na ekspertnom znanju o povoljnoj raspodjeli redukcija, sila valjanja ili potrebnih snaga motora, po valjačkim stanovima. Prije ulaska početka trake u prvi valjački stan, vrše se pomoću matematičkog modela valjanja, tačni proračuni redukcija, debljina, temperatura, otpora materijala na deformaciju, sila valjanja, momenta motora, itd.. Kao krajnji rezultat se dobije skup referensi za aktuatore (brzine motora, pozicije vretena, pojačanja petlji regulacije, itd.), koje predstavljaju predproračun postavke pruge.

Mjerene vrijednosti (brzine, debljine, temperature, sile, zazori valjaka, itd.) se tokom valjanja trake koriste u tzv. postproračunu, gdje se one propuštaju kroz inverzni model valjanja i na taj način se dobija ostvarena postavka pruge.

Adaptacija modela

Adaptacija modela se vrši od trake do trake, u okviru jedne strategije valjanja, tj. jednog asortimana proizvoda. To se vrši na osnovu razlike između predproračunate i postproračunate (ostvarene) postavke pruge, korekcijom odabranog skupa koeficijenata modela, prema algoritmu:

$$K_{i+1} = K_i + \beta\,(K_m - K_i)$$

,

Gdje su: K_{i+1} – nova vrijednost predproračunatog koeficijenta,
K_i – stara vrijednost predproračunatog koeficijenta,
K_m – vrijednost postproračunatog koeficijenta,
β – pojačanje, faktor koji određuje brzinu konvergencije adaptacije.

Prikazani matematički model valjanja daje zadovoljavajuće rezultate, ali je potrebno istaći nedostatke i probleme pri njihovoj implementaciji i eksploataciji:

- Period ispitivanja sistema (odnosno preciznog podešavanja modela) traje po nekoliko mjeseci i zahtjeva valjanje svih asortimana proizvoda i 20 do 30 hiljada traka (300 do 450 hiljada tona čelika). To je potrebno, prije svega, zbog involviranja ekspertnog znanja, sažetog u tzv. strategiji valjanja, čije se korekcije računaju. Praksa je pokazala da čovjek mora odlučiti o tome da li će ponuđenu korekciju prihvatiti ili odbiti;
- Postizanje zadovoljavajuće brzine konvergencije algoritma adaptacije modela, obzirom na učestale promjene asortimana proizvoda pri valjanju, zahtjeva mnogo rada i vremena;

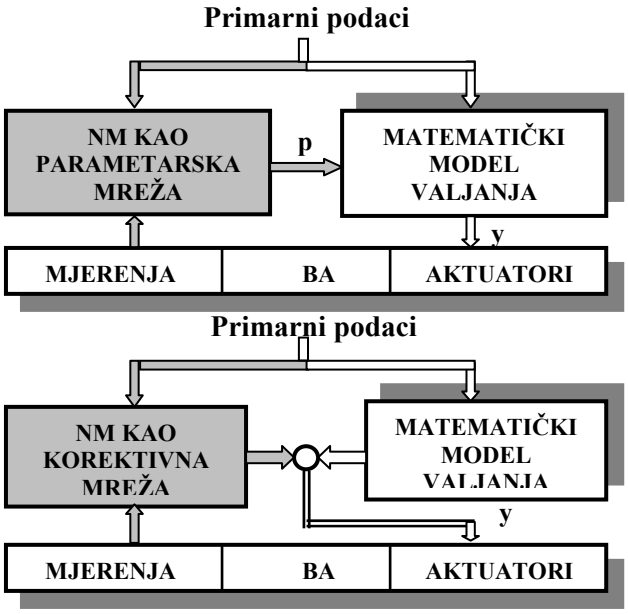
- Proizvođači traže smanjenje greške u postizanju ciljnih parametara valjanja (debljina, temperatura i profil trake), šta i jest osnovni zadatak modela valjanja. Problem sa dozvoljenim tolerancijama je prošlost, a svaki promil smanjenja greške povećava konkurentnost proizvođača.

Navedeni razlozi su osnovni zbog kojih se istraživanja kreću u smjeru primjene NM u računarskom upravljanju valjanjem čeličnih traka.

10.6.5. Neuronske mreže u ES za upravljanje valjanjem čeličnih traka

Dosadašnja istraživanja mogućnosti i efekta primjene NM u računarskom upravljanju valjanjem čeličnih traka, pokazala su da je najbolji slijedeći pristup: kombinacija klasičnih matematičkih modela i NM.

Na slici 10.12. je prikazan princip funkcionisanja slijedeće dvije kombinacije:
- model sa NM kao parametarskom mrežom,
- model sa NM kao korektivnom mrežom.



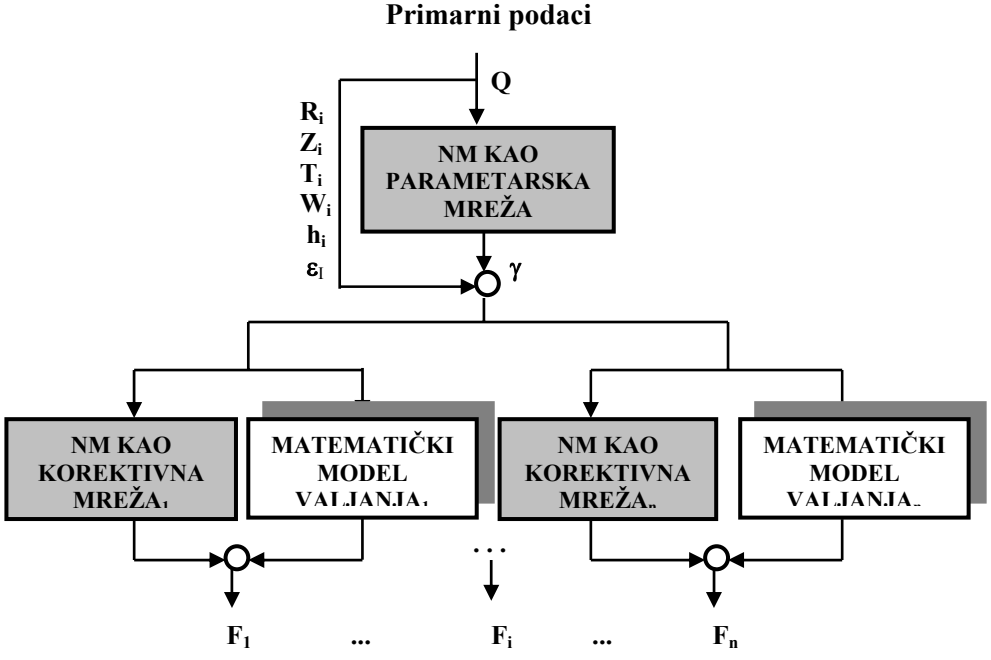
Slika 10.12. Princip funkcionisanja modela (za obe kombinacije) sa NM.

Formulacija, npr. NM kao korektivna mreža, ne označava tip NM, nego njenu funkciju u kombinaciji sa matematičkim modelom. Kako se matematički model valjanja može posmatrati kao da se sastoji od više dijelova, koji su u jačoj ili slabijoj interakciji:

- model proračuna redukcija,
- temperaturni model,
- model za proračun otpora materijala na deformaciju,
- model sile valjanja,
- model oblika trake,
- itd.,

moguće je praviti najpogodnije kombinacije dijelova modela i NM.

Uočeno je da je za model sile valjanja, koji je najbitniji dio matematičkog modela valjanja, pogodna NM kao parametarska mreža, ali se bolji rezultati postižu odgovarajućom serijskom kombinacijom: NM kao parametarska mreža u kombinaciji sa matematičkim modelom i NM kao korektivnom mrežom, slika 10.13.



Slika 10.13. Primjena neuronskih mreža u modelu sile valjanja.

U neuronsku mrežu, kao parametarsku mrežu, se dovode na ulaze podaci o hemijskom sastavu čelika *Q* (procenat sadržaja ugljenika, silicijuma, mangana, fosfora, sumpora, itd.), a na izlazu se dobija samo jedan koeficijenat - γ. Vrijednost koeficijenta γ ulazi u kombinaciju sa modelom NM kao korektivnom mrežom. Ostali ulazni podaci za svaki valjački stan su:

- prečnik valjka R,
- sila zatezanja Z,
- temperatura trake T,
- širina trake W,
- debljina trake h,
- redukcija ε,
- itd.

Rezultat je zadana i očekivana sila valjanja F

Obučavanje NM traje, obično, oko dvije nedjelje. Pošto se radi o probnim valjanjima, koristi se pristup da računarski sistem istovremeno vrši proračun postavke pruge i to pomoću modela sa klasičnom adaptacijom i modela sa NM.

Na aktuatore se, preko bazne automatizacije (BA), proslijeđuje postavka pruge valjanja sa NM samo ukoliko je razlika između rezultata dva proračuna unutar tzv. prozora tolerancija. Ukoliko je razlika između rezultata proračuna pomoću modela sa klasičnom adaptacijom i NM van prozora tolerancija, zbog sigurnosti se proslijeđuje i koristi postavka proračuna pomoću modela sa klasičnom adaptacijom. Dokazano je da model sa NM ostvaruje smanjenje greške za oko 12% u odnosu na model sa klasičnom adaptacijom.

Kada su u pitanju toplo valjanje traka, odlični rezultati se ostvaruju primjenom relativno proste kombinacije: matematički model sa NM kao korektivnom mrežom. Smanjenje greške za međustansku temperaturu trake je oko 20%, a za završnu temperaturu i do 40%, u odnosu na model sa klasičnom adaptacijom parametara.

Kada je u pitanju NM kao mreža za sintezu, istraživanja su pokazala da je ta konfiguracija najpovoljnija za model profila trake, međutim još ne postoje konkretni rezultati o takvoj primjeni matematičkog modela i NM u procesu upravljanja valjanjem čeličnih traka.

Prikazani primjer je zasnovan na radovima Batak, D. N. i dr., "Neuralne mreže u računarskom upravljanju valjanjem čeličnih traka", 1996. i Schmitter D. E.: "Neural nets - types, configurations and pitfalls", 1995.

U ovom poglavlju:

- ***Uvod u genetičke algoritme***
- ***Načini kodiranja***
- ***De Jongov test***
- ***Testiranje***

11

11.1. UVOD U GENETIČKE ALGORITME

I pored toga što se genetički algoritmi (GA) počinju intenzivnije izučavati od 1975. godine, činjenica je da oni u posljednje vrijeme doživljavaju sve veću ekspanziju. Evoluciona paradigma, koju eksploatišu GA, postaje široko prihvaćena kao put za rješavanje određenih klasa problema.

Iako su, naročito u Njemačkoj, i prije 70. godina predlagana rješenja raznovrsnih problema koja emuliraju ponašanje, odnose i veze kao u prirodi, ipak se početak GA vezuje za 1975. godinu i *Holland*-ovu knjigu "*Adaptation in natural and artifiicial systems*". *Holland* je GA razvio radi proučavanja procesa prilagođavanja kod prirodnih sistema i radi razvoja sistema vještačke inteligencije, koji oponašaju modele prilagođavanja.

Treba istaći da je u prethodno pomenutoj knjizi napravljen i veliki pomak u matematičkim osnovama GA:

- definicija šema,
- hipoteza gradivnih blokova,
- teorema o implicitnoj paralelnosti,

i da GA povećanje popularnosti u velikoj mjeri duguju i postojanju matematičkog aparata, čijim se korištenjem pojedini rezultati mogu i predvidjeti i objasniti.

Materijal izložen u ovom poglavlju dijelom se zasniva na radovima Reynolds, D. i dr.: "Stohastic modelling of Genetic Algorithms", 1996. i Filipović, V. i dr., "Uticaj binarnog kodiranja na genetske algoritme za nalaženje ekstremnih vrednosti", 2000.

Genetički algoritmi se primenjuje kod raznih optimizacionih problema, kako iz problema iz domena teorije:

- raspoređivanje poslova,
- problem trgovačkog putnika,

tako i onih praktičnih:

- finansijsko modeliranje,
- projektovanje gasovoda,
- mašinsko učenje,
- teorija igara,
- neuronske mreže,
- prepoznavanje oblika, itd.

Ideja GA se zasniva na procesima oponašanaja prirodne selekcije.

U osnovi procesa selekcije koji se odvija u prirodi su slijedeće činjenice:

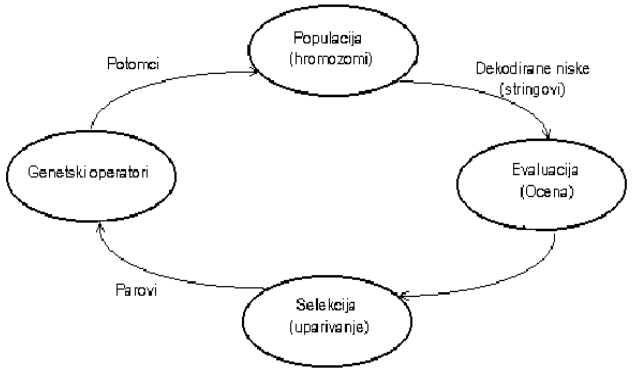
- jedinke bolje prilagođene okolini preživljavaju i imaju jači uticaj na formiranje slijedećih pokoljenja,
- jedinke jedne generacije u populaciji formiraju novu generaciju, na taj način što se osobine novih jedniki dobijaju kombinacijom genetičkog sadržaja roditelja,
- s vremena na vrijeme dolazi do mutacije, tj. do slučajne izmjene genetičkog sadržaja jedne jedinke.

Elementi genetičkih algoritama:

- pretraživački prostor, tj. skup svih mogućih rješenja,
- populacija, tj. skup aktuelnih kandidata za rješavanja; elementi populacije su jedinke (čvorovi pretraživanja, tj. tačke u pretraživačkom prostoru),
- prostor niski (stringova), tj. prostor reprezentacija jedinki, kao i funkcije za preslikavanje pretraživačkog prostora u prostor niski i obratno,
- skup genetičkih operatora za generisanje novih stringova, a samim tim i novih jedinki,

- ocjenjivačka funkcija, koja utvrđije povoljnost (korisnost) određene jedinke,
- stohastička kontrola genetičkih operatora.

Odnos elemenata GA može biti iskazan slijedećim dijagramom:



Slika 11.1. Odnos elemenata genetičkog algoritma.

Koraci genetičkog algoritma:

1) **Inicijalizacija** - slučajno se generiše inicijalna populacija pretraživačkih čvorova (što se svodi na generisnje čvorova u prostoru niski);
2) **Izračunavanje ocjenjivačke funkcije** (funkcije objekcije tj. *fitness* funkcije) za svaki čvor populacije;
3) **Operacija izbora tj. selekcije**, kojima se novi pretraživački čvorovi generišu slučajno, ispitivanjem vrijednosti ocjenjivačke funkcije za pretraživački čvor;
4) **Primjena genetičkih operatora** (ukrštanje, mutacija) na čvorove tj. na jedinke;
5) Ponavljanje koraka **2)**, **3)** i **4)** sve dok se ne ispuni kriterijum konvergencije, odnosno neki unaprijed fiksirani broj puta.

Karakteristike genetičkog algoritma:

- GA koriste kodiranje parametara, a ne same parametre,

- GA ne pretražuju jedan čvor pretraživačkog prostora, već populaciju čvorova (robustnost),
- GA koriste probabilistička pravila prelaska,
- GA (u svojoj čistoj formi) tokom procesa nalaženja rješenja ne eksploatišu eventualne dodatne informacije o prirodi problema (npr. glatkost funkcije kod problema optimizacije funkcije, itd.).

U realizaciji GA, radi povećavanja njihove efikasnosti, koriste se raznovrsne modifikacije:

- pri formiranju niske koja reprezentuje jedinku (haploidna i diploidna reprezentacija),
- pri kodiranju, odnosno preslikavanju pretraživačkog prostora u prostor niski (npr. kanoničko kodiranje, Grej kodovi, itd.),
- pri računanju ocjenjivačke funkcije (razna skaliranja),
- pri izboru genetičkih operatora (npr. za diploidnu reprezentaciju se uvodi dominacija),
- pri utvrđivanju vjerovatnoća koje određuju primjenu genetičkih operatora,
- pri utvrđivanja kriterijuma konvergencije (npr. da li se najbolja jedinka promjenila tokom fiksiranog broja generacija,
- da li se više od pola jedinki u populaciji poklapa sa najboljom, da li se prosječna vrijednost jedinki malo razlikuje od najbolje, itd.).

Dok je originalni GA baziran na potpunoj smjeni generacija, u nekim modifikacijama se dopušta da roditelj, naravno ukoliko je njegova ocjena zadovoljavajuća, nastavlja da živi uporedo sa svojim potomcima. U konkretnim realizacijama GA se često eksploatiše međurješenje: najbolje prilagođena jedinka iz prethodne generacije nastavlja svoj život, i to je poznato pod imenom "elitist strategija". Nadalje, u raznim realizacijama se parametri koji određuju stohastičku kontrolu genetičkih operatora mijenjaju tokom samog rada algoritma.

Familija strategija rada GA, koje se razlikuju po vrijednosti stohastičkih parametara se nazivaju reproduktivnim planom. Termin reproduktivni plan potiče od *De Jong*-a.

11.2. NAČINI KODIRANJA

Genetički algoritmi su nastavili da se razvijaju, i mnogi autori su predlagali razna poboljšanja, tj. razne nove reproduktivne planove. Neke od predloženih izmjena su bile skoncentrisane i efikasno primjenljive samo na problem kojim se autor - predlagač bavio, dok su drugi prijedlozi bili opšte primjenljivi.

U praksi se dešava (empirijski je utvrđeno) da se određena kodiranja za neke tipove problema mnogo bolje "ponašaju", tj. da su rezultati primjene GA sa takvim kodiranjem osjetno bolji. Ta činjenica ima svoje teoretsko razjašnjenje, koje se zasniva na hipotezi gradivnih blokova. Gradivni blokovi su šeme malog reda a velikog *fitness*-a (ocjene), koje se eksponencionalno šire kroz prostor niski. Ispostavlja se da su bolja kodiranja kod kojih bliskim tačkama pretraživačkog prostora odgovaraju kodovi tj. niske koji se razlikuju na malom broju bitovnih mjesta, odnosno niske sa malim Hemingovim rastojanjem.

Prijedloga i poboljšanja kod načina kodiranja ima mnogo, ali u knjizi su opisani: kanoničko kodiranje, Grej kodiranje i dinamičko kodiranje parametara.

11.2.1. Kanoničko kodiranje

Ovo je istorijski najstariji način kodiranja. Za problem optimizacije funkcija preslikavanje realnog broja iz nekog intervala u nisku bitova (dužine n) se obavlja na slijedeći način: interval se podjeli na ekvidistantne tačke (broj tačaka je 2^n), svakoj tački se pridruži binarni cio broj koji označava redni broj te tačke u intervalu, i realnom broju se dodjeli redni broj najbliže donje tačke podjele.

11.2.2. Grej kodiranje

Samo kodiranje se vrši na sličan način kao u prethodnoj tački, s tim što se tačkama podjele ne dodjeljuje binarni ekvivalent rednog broja, već Grej kod koji odgovara rednom broju tačke u intervalu.

Grej kod je dobio naziv po *Frenk Grej*-u, koji je 1953. patentirao njegovo korištenje za određene kodere i dekodere. Grej kodiranje će svaki od cijelih brojeva iz skupa $\{1, 2, \ldots, 2^{n-1}\}$ predstaviti kao binarnu nisku dužine n na takav način da susjedni cijeli brojevi imaju Grej reprezentacije, koje se razlikuju u jednoj bitovnoj poziciji. Tako je jedno Grej kodiranje (i ne samo jedno) za skup {0,1,2,3,4,5,6,7} dano sa 000,001,011,010,110,111,101,100. Ovakav način kodiranja se najčešće koristi i naziva se binarno reflektovani Grej kod.

Prosta šema za generisanje Grej koda sekvenci glasi: "počni sa svim nulama, sukcesivno mijenjaj najdesniji bit, čijom se izmjenom može proizvesti nova niska".

11.2.3. Dinamičko kodiranje parametara

U ovom slučaju šema kodiranja ne ostaje konstantna tokom genetičke pretrage. Tako dobijeni rezultat može biti precizniji od 1/2, jer se na početku koristi vrlo gruba preciznost, sve dok GA ne prokonvergira. U tom trenutku se rezultujuća populacija (odnosno podinterval koji ona zauzima) ponovo kodira u isti skup bitova, tako da je pretraga sužena na ovaj podinterval, pa se GA ponovno startuje. Ovaj bi se proces mogao nastaviti sve dok rezultat ne dobije željenu
preciznost. Ustvari, dinamičko kodiranje parametara to i radi, i to autonomno i paralelno za svaku realnu vrijednost. Na ovaj način, žrtvovanje rezoluciji pretage ne mora dovesti do istovjetnog gubljenja preciznosti kod rezultata pretrage.

Kod ovog načina kodiranja mora postojati operator zumiranja.

11.2.4. Teoretska procjena

Poglavlje o kodiranju ne bi bilo kompletno ukoliko se ne navede i *Vous*-ov rezultat, gdje gradivni blokavi ne moraju biti invarijante problema koji se rješava GA. Dakle, rezultati GA za svako fiksno kodiranje pri rješavanju neke klase problema bi, ukoliko se usrednje po skupu svih instanci te klase problema, trebalo da daju otprilike iste rezultate.

Vous je, u drugom svom radu, još pokazao da je Grej kodiranje specijalan slučaj kodiranja indukovanog fiksnom matricom.

11.3. DE JONGOV TEST

Godine 1975, u vrijeme izdavanja *Holland*-ove knjige, *De Jong* završava svoju doktorsku disertaciju: "*An Analysis of the Behavior of a Class of Genetic Adaptive Systems*". Taj će rad imati važno, moglo bi se reći centralno, mjesto u daljem razvoju GA. Disertacija je postala prekretnica u daljem razvoju GA kako zbog povezanosti sa *Holland*-ovom teorijom šema, tako i zbog pažljivih i preciznih računarskih eksperimenata, te pažljivo razmotrenih zaključaka.

U to vrijeme *De Jong* se interesovao za primjenu GA u dizajniranju struktura podataka, dizajniranju algoritama i adaptivne kontrole operativnog sistema. Uprkos privlačnosti ovih ezoteričnijih problema, *De Jong* je uvidio značaj pažljivo kontrolisanog eksperimentisanja u problemskom domenu koji nije nesređen. On je tako ogolio GA, okruženje i performanse do same osnove. Takvo uproštavanje mu je omogućilo da izvrši eksperimente koji su postali bazna linija za sva dalja proučavanja GA i dalje radove.

De Jong-ov skup uključuje test funkcije sa slijedećim karakteristikama:

- neprekidne / prekidne,
- konveksne / nekonveksne,
- jednomodalne / višemodalne,
- kvadratne / nekvadratne,
- malodimenzionalne / mnogodimenzionalne,
- deterministčke / stohastičke.

De Jong-ovo test - okruženje sadrži slijedećih pet funkcija:

1. Sferna funkcija: $f_1(x_1, x_2, x_3) = \sum_{i=1}^{3} x_i^2$

pri čemu je $-5.12 \le x_i \le 5.12$.

2. Rozenbrokova funkcija: $f_2(x) = 100(x_1^2 - x_2)^2 + (1 - x_1)^2$

pri čemu je $-2.048 \le x_i \le 2.048$

3. Stepenasta funkcija: $f_3(x_1, ..., x_5) = \sum_{i=1} [x_i]$

pri čemu je $-5.12 \le x_i \le 5.12$

4. Četvorna funkcija sa šumom: $f_4(x_1, ..., x_{30}) = \sum_{i=1}^{30} i x_i + Gaus(0,1)$

pri čemu je $-1.28 \le x_i \le 1.28$

5. Šekelova funkcija: $f_5(x_1, ..., x_{25}) = 0.0002 + \sum_{j=1}^{25} \frac{1}{j + \sum_{i=1}^{2} (x_i - a_{ij})^6}$

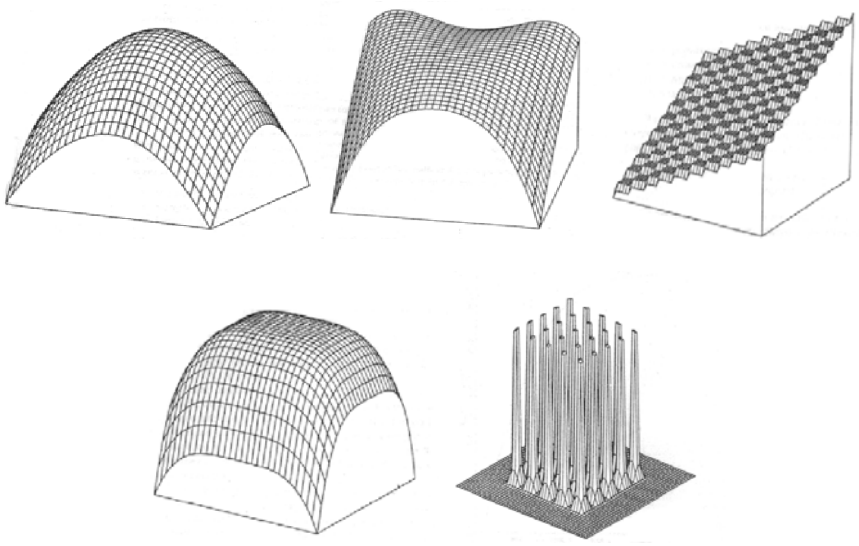
pri čemu je $-65.536 \le x_i \le 65.536$

Na slikama koje slijede su predstavljeni grafikoni dvodimenzionalnih ekvivalenata prethodno definisanih funkcija. *De Jong* je, radi kvantfikovanja efektivnosti svakog od različitih GA, uveo dvije mjere, jednu za mjerenje konvergencije, a drugu za mjerenje performansi toka. Prvu je nazvao *off-lajn performansa*, a drugu *on-lajn performansa* - po analogiji sa *off-lajn* i *on-lajn* aplikacijama. Novouvedene performanse se definišu na sledeći način.

- *Off-lajn* performansa (konvergencija) strategije *s* u okruženju *e*:

$$x_e^*(s) = \frac{1}{T}\sum_{t=1}^{T} f_e^*(t),$$

gdje je sa $f_e^*(t)$ označena vrijednost najboljeg u skupu $\{f_1(t), f_2(t), \ldots, f_e(t)\}$.



Slika 11.2. Grafikoni dvodimenzionalnih ekvivalenata *De Jong*-ovih funkcija.

- *On-lajn* performansa (tok) strategije *s* u okruženju *e*:

$$x_e(s) = \frac{1}{T}\sum_{t=1}^{T} f_e(t),$$

gdje je sa $f_e(t)$ označena vrijednost funkcije objekcije za okruženje *e* tokom probe *t*.

Of-lajn performansa je prosjek dosad najboljih performansi u raznim generacijama. *De Jong* je predložio i opštiju verziju ovog kriterijuma, koja dopušta neuniformnu dodjelu težina raznim pokušajima. Međutim, u daljem radu se isključivo radi sa uniformnom varijantom. *On-lajn* performansa je

prosjek svih funkcijskih evaluacija, zaključno sa tekućim pokušajem. I za ovu mjeru je bila predložena i neuniformna varijanta, ali je u radu korištena uniformna. Pored ove dvije karakteristike, *De Jong* je ispitivao i broj izgubljenih gena tokom probe *t*. Ova veličina ukazuje na širinu pretrage kroz pretraživački prostor.

Ispitivane su razne varijacije genetičkog algoritma. U svojoj disertaciji, *De Jong* je razmatrao odnos slijedećih reproduktivnih planova:

- R1 - Prosti model,
- R2 - Elitist model,
- R3 - Model očekivane vrijednosti,
- R4 - Elitistički model očekivane vrijednosti,
- R5 - Model faktora gomilanja,
- R6 - Model uopštenog ukrštanja.

De Jong je u svojim testovima koristio isključivo kanoničko kodiranje. Treba napomenuti da, iako se pomoću GA lijepo rješavaju određeni problemi optimizacije funkcija, ne treba smatrati da se GA samo zato koriste. Nedavno je izašao rad jednog od vodećih autoriteta u ovoj oblasti, *Keneta De Jonga*, pod naslovom: "Genetički algoritmi nisu optimizatori funkcija", u kome autor kritikuje taj, dosta često viđen, simplifikovan prilaz problematici GA, ukazuje na razlike koje se pojavljuju između GA za optimizaciju funkcija i GA za rješavanje drugih problema i ističe nedostatak čvršće matematičke teorije vezane za predložena poboljšanja.

11.4. TESTIRANJE

Evaluacija raznih varijacija GA i određivanje performansi je vršeno na algoritamski orijentisanom GA sistemu. Dizajn i osobine GA sistema, komparativ-ne prednosti i eventualni nedostaci u odnosu na preko Interneta dostupan PD softver, kao i u odnosu na komercijalni GA softver, su tema za sebe. Rasprava o tim pitanjima zahtjeva mnogo više prostora. Međutim, mišljenje je da bi ovo poglavlje bilo nekompletno ukoliko se ne navedu neke najvažnije karakteristike razvijenog GA sistema:

- **Fleksibilnost** - koristeći razvijeni softver, a zahvaljujući pažljivo izvršenom koncipiranju i realizovanju, kreirani su za genetski zasnovane algoritme programi i "razmatrani" problemi raznovrsnih tipova, od kojih su neki vrlo egzotični (npr. genetski algoritam na dva nivoa: normalnom i meta, a radi odredjivanja samih stohastičkih veličina koje definišu rad GA; NP teški kombinatorni problemi, itd.).

- **Ponovno korištenje razvijenog koda (*reusibility*)** - kod razvijen radi rješavanja jednog od problema se može, bez ikakvih izmjena, čak i bez ponovnog prevođenja, koristiti i u druge svrhe (npr. računanje vrijednosti *De Jong*-ovih performansi, koji može da se koristi, i koristi se, za ocjene performansi novih varijanti GA, itd.).

- **Nezavisnost od operativnog sistema i računara** - softer može biti razvijen u programskom jeziku C, uz strogo poštovanje ANSI standarda, tako da ga GA sistem može lako koristiti na najraznovrsnijim platformama. Nezavisnost od platforme se ne može u potpunosti postići kada npr. treba eksploatisati eksplicitnu paralelnost GA, ali i tada ostali bitni elementi u svim drugim modulima ostaju nepromjenjeni.

- **Efikasnost** - način koncipiranja i način realizacije ovog softvera omogućili su, i ako je softver veoma fleksibilan, njegovu potpunu nezavisnost od platforme, pri čemu njegove performanse nisu bitno degradirane.

Veličine stohastičkih parametara (generacijski jaz, vjerovatnoće ukrštanja i mutacije) su pri testiranju postavljene na vrednosti koje je *De Jong* (uz *Greffenstate*-ova preciziranja) utvrdio kao najbolje (potpuna smjena generacija tj. G=1, vjerovatnoća ukrštanja 0.6, vjerovatnoća mutacije 0.01).

Rezultati dobijeni poređenjem ukazuju da je, za ovaj skup test - funkcija, Grej kodiranje bolje od kanoničkog (u smislu poboljšanja *off-lajn* i *on-lajn* performansi) prosječno za desetak procenata, pri čemu je kod "normalnijih" funkcija to poboljšanje primjetnije.

12

U ovom poglavlju:

- *Vrste sistema baziranih na agentima*
- *Multiagenti*
- *Razlike između OO i agentne metodologije*
- *Primjena agenata*
- *Inteligentni agenti*

AGENTI, MULTIAGENTNI I INTELIGENTNI AGENTI

12.1. OPŠTE O AGENTIMA

Agenti predstavljaju softver koji ima sposobnosti da samostalno, bez intervencije korisnika, izvršava postavljeni zadatak i izvještava korisnika o završetku zadatka ili pojavi očekivanog događaja. Agent se može definisati na slijedeći način:

Agent je računarski softver, koji u interakciji sa okruženjem, ima sposobnost da fleksibilno i samostalno reaguje u skladu sa ciljevima koji su mu postavljeni.

U ovoj definiciji se ističu tri ključna zahteva:

- interakcija sa okruženjem,
- autonomnost, i
- fleksibilnost.

Interakcija sa okruženjem, u ovom kontekstu, znači da su agenti sposobni da reaguju na ulaze dobijene od senzora iz okruženja i da moge da izvodi akcije koje mijenjaju okruženje u kome agenti djeluju. Okruženje u kome agenti djeluju može biti fizičko (realan svijet) ili softversko (računar na kome su instalirani ili Internet), za razliku od klasičnih ES, koji informacije o okruženju dobijaju preko posrednika (korisnika) koji unosi parametre sistema. Klasični ES nisu bili u

Materijal izložen u ovom poglavlju najvećim dijelom se zasniva na knjizi Devedžić, V., "Inteligentni informacioni sistemi", 2000. i Internetu http://www.research.ibm.com/iagents/paps/rc20835.pdf.

mogućnosti da djeluju na okruženje (bar ne neposredno) ili su to, takođe, činili preko posrednika (korisnika), koji je u zavisnosti od dobijenog odgovora reagovao na okruženje.

Autonomnost znači da je sistem u stanju da reaguje bez intervencije korisnika (ili drugih agenata) i da ima kontrolu nad sopstvenim akcijama i unutrašnjim stanjem. Takav sistem treba, takođe, da bude sposoban da uči iz iskustva. Mogućnost interakcije sa okruženjem i autonomnost računarskih sistema nije nova ideja. Postoje već mnogi takvi sistemi, kao što su programi za kontrolu realnih sistema koji nadgledaju okruženje realnog svijeta i izvode akcije kao odgovor na promjene sistema u realnom vremenu. Tu se mogu ubrojiti i programi koji nadgledaju softversko okruženje i izvode akcije kojima djeluju na okruženje u slučaju promjene radnih uslova (antivirus programi).

Navedeni primjeri imaju odlike interakcije sa okruženjem i autonomnosti, ali se ovi sistemi ne mogu smatrati agentima sve dok nemaju mogućnost fleksibilnog ponašanja kada se nađu u situacijama koje nisu planirane prilikom dizajniranja.

Da bi se softverski sistem smatrao **fleksibilnim** mora da ispunjava slijedeće uslove:

- agent treba da primjeti promjene u okruženju i da donese odluku o mogućim akcijama dovoljnom brzinom da bi takva akcija bila od značaja za sistem u kome djeluje,
- agenti ne treba samo jednostavno da reaguju kao odgovor na signale iz okruženje, oni treba da budu sposobni da uočavaju povoljne prilike i u tim situacijama preuzimaju inicijativu u skladu sa svojim ciljevima,
- agenti treba, po potrebi, da su sposobni da stupe u komunikaciju sa drugim agentima i/ili ljudima da bi riješili sopstveni problem ili pomogli jedni drugima u njihovim aktivnostima.

12.1.1. Vrste sistema baziranih na agentima

Agenti se međusobno razlikuju u zavisnosti od domena primjene, međutim oni se ipak mogu klasifikovati u nekoliko karakterističnih klasa kao što su:

1. Kooperativni agenti (*Collaborative Agents*);
2. Interfejs agenti (*Interface Agents*);
3. Mobilni agenti (*Mobile Agents*);

4. Informacioni / Internet agenti (*Information / Internet Agents*);
5. Reaktivni agenti (*Reactive Agents*);
6. Inteligenti agenti (*Smart Agents*);
7. Hibridni agenti (*Hybrid Agents*).

Kooperativni agenti

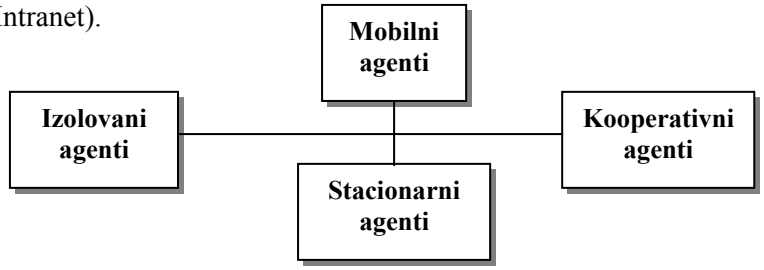
Kooperativni agenti omogućavaju međusobno povezivanje i obradu postojećih višestrukih sistema (npr. ekspertni sistemi ili sistemi za podršku odlučivanju). Najčešće se upotrebljavaju za rješavanje problema koji su po prirodi decentralizovani ili za rješavanje problema koji su suviše kompleksni da bi se rješavali jednim centralizovanim agentom. Primjenom kooperativnih agenata može se prevazići problem ograničenih resursa i smanjiti rizik otkazivanja sistema, koji je prisutan u centralizovanim sistemima. Ovakvi agenti poboljšavaju modularnost sistema, brzinu, pouzdanost, fleksibilnost i ponovno iskorišćenje (*re-usability*) sistema

Interfejs agenti

Uloga interfejs agenata je da krajnjem korisniku olakšaju upravljanje sistemom na kome radi. Ovi agenti se mogu upotrijebiti za izradu adaptivnog korisničkog interfejsa (*Adaptive user interfaces*). Ulaga agenata u ovakvom sistemu je da tokom vremena prate navike korisnika i pretpostavljaju njegove buduće akcije. Interfejs agenti imaju zadatak da prikazuju informacije za koje pretpostavljaju da u datom trenutku interesuju korisnika.

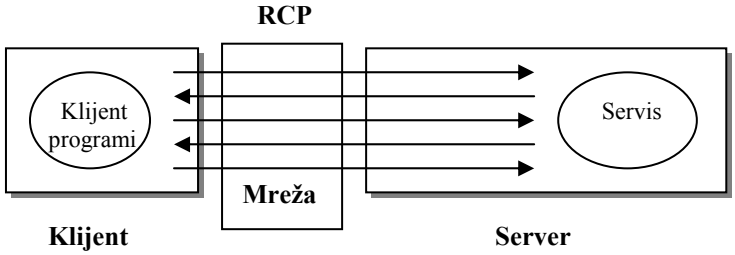
Mobilni agenti

Za ovu vrstu agenata je karakteristično da imaju sposobnost da se fizički kreću od jednog prema drugom serveru (hostu) agenata preko računarske mreže (Internet / Intranet).



Slika 12.1. Blok šema prostornog agenata.

Mobilni agenti su se pokazali posebno korisnim u situacijama kada je potrebno smanjiti obim (troškove) komunikacije između povezanih računara. Umjesto obimne razmjene informacija između računara, šalje se agent na izvor informacija i tamo vrši obradu podataka.



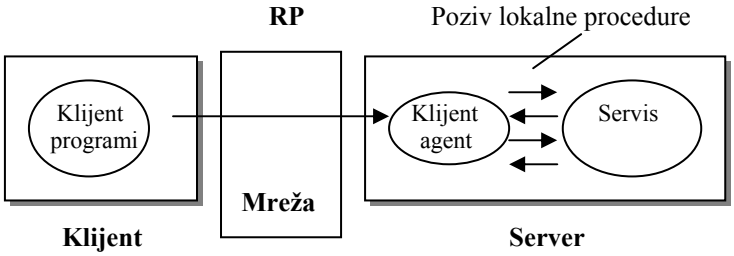
Slika 12.2. Poziv udaljene procedure (*Remote Procedure Call*, **RPC**).

Nakon završene obrade, agent šalje rezultate serveru. Ovi agenti se, takođe, mogu upotrebljavati da bi se prevazišao problem ograničenih lokalnih resursa. Ukoliko su resursi jednog računara zauzeti, agent može da potraži računar sa slobodnim resursima i tamo obraditi podatke.

Životni ciklus mobilnih agenata se sastoji od:

- stanja pokretanja,
- stanja izvršavanja, i
- stanja ispunjenja postavljenog zadatka, nakon čega se agent uništava (oslobađaju se zauzeti resursi).

U trenutku kada se agenti kreću od jednog računarskog sistema ka drugom, stanje izvršavanja se zaustavlja i memoriše se trenutno stanje obrade. Kada se agent prenese na drugi računar, obrada se nastavlja sve dok se ne izvrši postavljeni zadatak



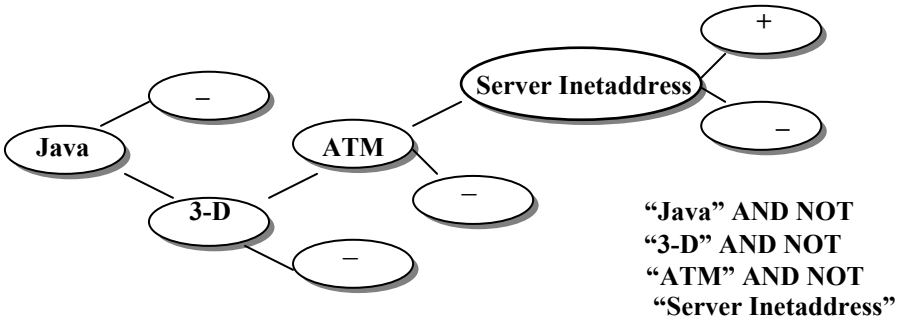
Slika 12.3. Daljinsko programiranje (*Remote Programming*, **RP**)

U sistemima gdje se primjenjuju mobilni agenti, posebna pažnja se mora posvetiti bezbjedonosnim problemima kao što su:

- spriječavanje neovlaštenih osoba / agenta da prikupljaju podatake,
- spriječavanje neovlaštenih osoba / agenata da mijenjaju podatake na sistemima,
- spriječavanje neovlaštenih osoba / agenata da upotrebljavaju resurse sistema,
- spriječavanje pokretanja agenata sa nejasnim ili zlonamjernim ciljevima.

Informacioni / Internet agenti

Zadatak informacionih agenata je da se suoče sa savremenim kompleksnim informacionim okruženjem. Zadatak takvih agenata se sastoji od pronalaženja informacija na lokalnom hard disku, preko niza sistema za pretraživanje višestrukih baza na udaljenim serverima ili pronalaženje informacija na Intranetu ili Internetu.



Slika 12.4. Primjer stabla odlučivanja dobijenog korištenjem agenta *InfoFinder*-a.

Glavni zadatak informacionih agenata je da aktivno tragaju za informacijama u zavisnosti od interesovanja korisnika u svom informacionom okruženju, obavještavajući korisnike o novim sadržajima koji zadovoljavaju postavljene kriterijume. Informacioni agenti pronalaze, analiziraju, obrađuju i objedinjavaju informacije sa više nezavisnih izvora. Sistemi informacionih agenata aktivno tragaju za podacima za koje oni vjeruju da su interesantni korisniku, umjesto da informacije dobijaju jednostavnim propuštanjem kroz pasivni filter.

Istraživanje oblasti informacionih agenata predstavlja izazov za razvoj naredne generacije informacionih okruženja.

Reaktivni agenti

Reaktivni agenti ne planiraju svoje akcije, njihove akcije zavise isključivo od trenutnih događanja u sistemu. U ovakvim agentima se najčešće primenjuju tradicionalne tehnike vještačke inteligencije, kao što je monotono zaključivanje. Do sada su se reaktivni agenti najčešće primenjivali u računarskim igrama.

Hibridni agenti

Ova vrsta agenata je bazirana na jednoj ili više vrsta agenata sa prethodne liste.

12.2. MULTIAGENTI

Sistemi u kojima se upotrebljava više agenata radi rješavanja zajedničkog problema se nazivaju multiagentni sistemi. U ovakvim sistemima neophodno je da agenti imaju mogućnost međusobne komunikacije u cilju razmjene iskustva ili "pregovaranja", da bi se našlo optimalno rješenje. Agenti koji se upotrebljavaju u multiagentnim sistemima mogu biti jednaki po karakteristikama ili se mogu razlikovati prema specijalnostima.

Multiagentni sistemi su idealni za predstavljanje problema koji imaju više različitih metoda za rješavanje problema i/ili višestruke perspektive. Omogućavaju izradu paralelnih računarskih sistema, pomažu pri radu sa vremenski ograničenim zaključivanjem i robusnim sistemima, ako su odgovornosti podjeljene. U sistemima izrađenim na ovaj način, umjesto da procesom upravlja jedan kompleksan agent, upravljanje se dijeli na više agenata koji prema svojim specijalnostima preuzimaju nadležnost nad kontrolom složenog procesa. Upotrebom multiagentnih sistema se povećava bezbjednost sistema u situacijama otkazivanja jednog od agenata, čitav sistem može biti automatski rekonstruisan ili zaustavljen na kontrolisan način.

Prilikom dizajniranja multiagentnih sistema potrebno je definisati broj agenata, kritičnu količinu vremena za obavljanje zadataka, dinamiku pristizanja ciljeva, troškove komunikacije, cijenu neuspjeha, uticaj korisnika, neodređenost okruženja.

Na nivou svakog agenta potrebno je definisati:

- početna stanja u domenu,
- moguće akcije drugih agenata,
- izlazne akcije agenta.

Sa povećanjem broja agenata, koji sarađuju na rješavanju zajedničkog problema, javljaju se problemi kao što su:

- kooperativnost (dizajnirati agente tako da zajednički rade na zajedničkim ciljevima),
- koordinacija (upravljati agentima tako da izbjegnu štetne interakcije, a korisne interakcije iskoriste),
- pregovaranje (dolaženje do dogovora koji su prihvatljivi svim objektima / agentima koji učestvuju u rješavanju problema).

Kod razvoja multiagentnih sistemima javlja se potreba da se standardizuje komunikacija između agenata, što obezbjeđuje komunikaciju između agenata koji su razvijeni nezavisno jedan od drugoga (različiti programeri). Time se omogućava planiranje akcija i upravljanje resursima cjelokupnog sistema.

12.2.1. Razlike između objektno orijentisane i agentne metodologije izrade programa

U objektno orijentisanim programskim jezicima, objekti se definišu kao entiteti koji enkapsuliraju neka stanja, imaju mogućnost da izvode akcije ili metode i međusobno razmenjuju informacije. Pod principom enkapsulacije se, u objektno orijentisanim programskim jezicima, smatra da se entitetu (objektu) pristupa kao cjelini, ne uzimajući u obzir unutrašnju strukturu i ne utičući direktno na nju. Objektu se pristupa isključivo preko javnih metoda i funkcija, i na taj način se izbjegavaju konfliktne situacije u kojima bi korisnik, ili drugi objekat, greškom poremetio unutrašnju strukturu objekta. Kada se na ovaj način pravi poređenje između objekata i agenata može se uočiti dosta sličnosti.

Pored sličnosti postoje i brojne razlike. Jedna od značajnih razlika između agenta i objekta predstavlja stepen autonomije prilikom izvršavanja metode. Objekti predstavljaju pasivne entitete, koji slijede naredbe programa u kome su primjenjeni, dok su agenti samostalni entiteti koji slijede sopstvenu logiku u skladu sa sopstvenim ciljevima.

PRIMJER: *U objektno orijentisanim programskim jezicima ukoliko postoji definisan objekat A, sa javno definisanom metodom M_1, ostali objekti mogu pozivati ovu metodu po potrebi, pri čemu objekat A nema uticaj da li će se metoda M_1 izvršiti ili ne.*

Prilikom izrade agenata, ne može se uvijek pretpostaviti šta će biti zajednički zadatak na kome će se agenti upotrjebiti, kao što je slučaj sa objektima. U multi-agentnim sistemima se mogu često naći agenti koje su izradili drugi pojedinci ili firme i koji nisu mogli predvidjeti u kakvim sistemima će se agenti upotrebljavati. Agenti predstavljaju entitete koji djeluju u skladu sa sopstvenim ciljevima.

***PRIMJER*:** *Ukoliko agent B pokuša da pozove metodu M_1 agenta A, može se desiti da izvršavanje metode M_1 nije trenutno u interesu agenta A. Kako metoda M_1 pripada agentu A, on donosi konačnu odluku da li će se metoda M_1 izvršiti ili ne.*

U multiagentnom sistemu bi se prije moglo reći da agenti međusobno upućuju zahtjeve za izvršenjem metoda, umjesto da direktno pozivaju metode kao što je slučaj sa objektima u objektno orijentisanim programskim jezicima.

Postoji još jedna ključna razlika između ove dvije tehnologije. U objektno orijentisanom programiranju se svi objekti izvršavaju u jednom *tread*-u (koraku) aplikacije, u kojem su upotrebljeni, dok u multiagentnom sistemu svaki agent predstavlja poseban *tread*. Naravno, agenti se mogu implementirati objektno orijentisanim programskim jezicima.

12.2.2. Primjena multiagenata

Oblast autonomnih agenata i multiagentnih sistema je veoma raznolika i predstavlja oblast koja se ubrzano širi. Metodologija izrade programa, bazirana na agentima, pruža niz efikasnih alata i tehnika koje imaju potencijal da značajno unaprjede tehniku izrade softvera, počevši od idejnog rješenja pa sve do konkretne implementacije.

Predstavlja spoj više naučnih oblasti kao što su:

- distribuirana obrada podataka,
- objektno orijentisani sistemi,
- softverski inženjering,
- vještačka inteligencija,

- ekonomija,
- sociologija, i
- organizacione nauke.

Tehnologija agenata dobija sve više na značaju i oni se sve više upotrebljavaju za rješavanje realnih problema i komercijalnim aplikacijama. Agenti imaju veoma širok spektar primjena, od veoma jednostavnih sistema kao što su filteri za elektronsku poštu, programa za presretanje i uklanjanje računarskih virusa pa sve do veoma kompleksnih, kao što je softver za kontrolu avio saobraćaja.

Oblasti u kojima se trenutno najčešće primenjuju aplikacije na bazi agenata se mogu svrstati u slijedeće:

- proizvodnja,
- kontrola procesa,
- telekomunikacioni sistemi,
- kontrola avio saobraćaja,
- upravljanje transportom,
- meteorologija,
- filtriranje i sakupljanje informacija,
- upravljanje tokovima informacija,
- elektronska trgovina,
- upravljanje poslovnim procesima,
- medicina,
- industrija zabave,
- kompjuterske igre, i
- drugim oblastima.

U toku protekle dvije decenije je otkriven značajan broj poboljšanja u dizajnu i implementaciji autonomnih agenata, kao i načinu na koji oni stupaju u interakciju. Tehnologija na bazi agenata sve više nalazi primjenu u komercijalnim proizvodima i softveru koji se primjenjuje u realnom okruženju.

Trenutno je veoma važno riješiti dva problema:

- nedostatak jasno definisane sistemske metodologije za razvoj agenata u multiagentnim okruženjima, i
- nedostatak široko rasprostranjenih, dostupnih i standardizovanih razvojnih aplikacija za izradu multiagentnih sistema.

Većina dosadašnjih aplikacija su dizajnirane na bazi metodologije pozajmljene iz objektno orijentisanih programskih jezika. Trenutno ne postoji metodologija koja definiše kako najbolje struktuirati multiagentni sistem, kako uskladiti individualne i/ili kolektivne ciljeve agenata u međusobnoj komunikaciji ili koja je najbolja struktura individualnog agenta u takvom sistemu.

12.3. INTELIGENTNI AGENTI

12.3.1. Pojam inteligentnih agenata

Inteligentni agenti predstavljaju softver koji automatski može da izvrši zadatak koji mu postavi osoba ili drugi softver (agent). Kada se jednom podese oni izvršavaju svoje zadatke, automatski, bez dalje intervencije korisnika.

Najčešće se upotrebljavaju za:

- automatsko traganje za informacijama,
- pružaju odgovore na postavljena pitanja u domenu svog znanja,
- informišu korisnike o interesantnim događajima (o pojavi novog članka na Internetu, prikazuje informacije o eventualnoj pojavi problema na putu između početne i krajnje destinacije, da li se zadani pojam pojavljuje negdje na web-u i slično),
- obezbjeđuju trenutne i presonalizovane vijesti,
- omogućavaju inteligentno obučavanje korisnika,
- pronalaze robu po najpovoljnijim cijenama,
- obezbjeđuju automatske servise, kao što je provjera promjena na web stranicama ili pojava "prekinutih" linkova.

Inteligentni agenti (IA) dobijaju informacije iz svog okruženja. "Opažaju" okruženje, odlučuju o svojim akcijama i izvršavaju ih. IA su implementirani kao programi (funkcije), koji preslikavaju opažanja u akcije.

Prisutna je nedovoljna usaglašenost oko definicije pojma IA. Termini koji su u upotrebi:

- inteligentni agenti,
- adaptivni interfejsi,
- personalni agenti,

- autonomni agenti,
- mrežni agenti, i drugi.

Osnovna ideja u razvoju IA je:
- da inteligentni sistemi pomažu svim kategorijama krajnjih korisnika,
- poređenje sa ostalim tehnologijama proisteklim iz vještake inteligencije,
- ”nešto između” krajnjeg korisnika i drugih programa drugih agenata i drugih programa.

Inteligentni agenti pomažu korisnicima na razne načine:
- prikrivaju složenost teških zadataka,
- obavljaju zadatke u ime korisnika,
- podučavaju korisnika,
- nadziru događaje i procedure,
- pomažu u međusobnoj saradnji korisnika

Korisnik **ne mora** da “posluša” agenta.

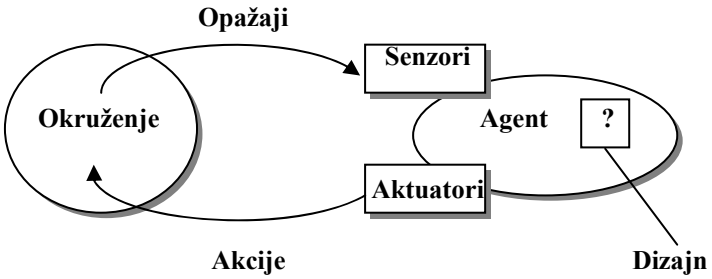
Potreba za IA:
- informacije je sve teže locirati, prikupljati, filtrirati, ocjenjivati i integrisati,
- sve je teže koordinirati dobijanje informacija iz heterogenih izvora.

12.3.2. Opšti opis inteligentnih agenata

Inteligentni agent je:
- autonomni softverski entitet,
- opaža svoje okruženje putem senzora,
- dejstvuje na svoje okruženje putem aktuatora,
- može da izvrši neki zadatak,
- ima sposobnosti navigacije i komunikacije.

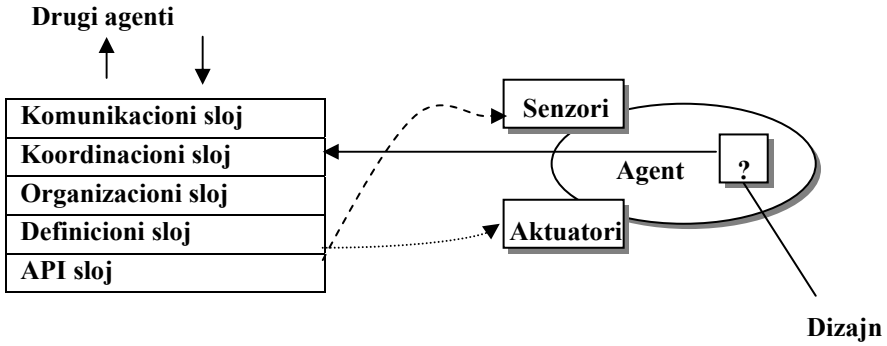
Agent može da bude čovjek, robot ili softverski proizvod. Svaki agent posjeduje neko znanje.

Opis generičkog agenta se može predstaviti slijedećom slikom:



Slika 12.5. Grafički prikaz generičkog agenta.

Opšti unutrašnji izgled IA prikazan je na slici 12.6.



Slika 12.6. Opšti unutrašnji izgled IA.

Koordinacioni sloj omogućava:
- rad koordinacione tehnike u kordinaciji sa drugim agentima,
- razmjenu tehnika znanja i ekspertize sa drugim agentima,
- poboljšanje efikasnosti grupe u timskom radu, i dr.

Organizacioni sloj određuje:
- kojoj grupi agenata pripada koji agent,
- ulogu tog agenta u grupi,
- kojih je drugih agenata "svjestan" taj agent, i dr.

Definicioni sloj određuje:

- mehanizam odlučivanja,
- mehanizam učenja,
- ciljeve, činjenice, resurse, i dr.

12.3.3. Osobine inteligentnih agenata

Osobine IA su:

- autonomnost,
- sposobnost komuniciranja,
- sposobnost učenja,
- inicijativa i blagovremeni odziv,
- fleksibilnost, i
- prilagodljivost.

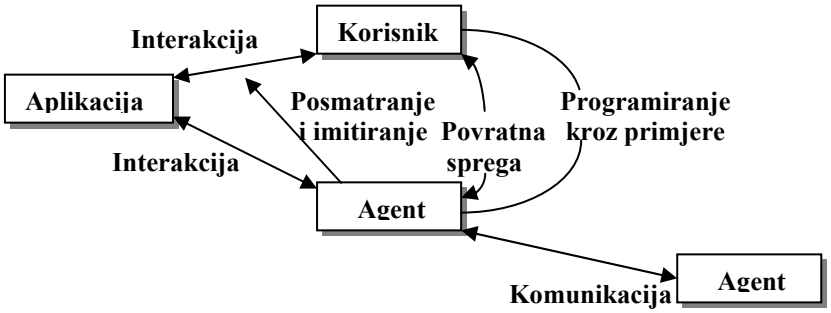
Autonomnost IA nije običan interfejs između korisnika i aplikacije. To je mogućnost rada u heterogenim okruženjima, mogućnost da se IA ”pusti”da radi preko noći (da bude dugo nenadgledan). IA je tipično dugotrajni (a ne jednokratni), kontinualan proces.

Sposobnost komuniciranja agenta je preduslov agentovog postojanja, odnosno njegovo postojanje je uslovljeno njegovom mogućnosti da komunicira sa drugim agentima. Jezik komunikacije mora da bude poznat, agent mora da čita poruke napisane tim jezikom, kao i da prihvata ograničenja koja nameće semantika poruka.

Sposobnost učenja, mogućnost prikupljanja novog znanja tokom rada, je uslovljeno posjedovanjem minimalnog početnog znanja. Uslovi za prikupljanje novog znanja su:

- repetitivnost zadataka koje IA rješava,
- različita repetitivnost za različite korisnike.

Sposobnost učenja se može predstaviti slijedećom slikom.



Slika 12.7. Grafički prikaz sposobnosti učenja.

12.3.4. Struktura inteligentnih agenata

Strukturu inteligentnih agenata sačinjavaju hardver, koji se sastoji od računarskog sistema i specijalnog hardvera (npr. kamera ili audio ulaza) i računarskih programa, koji sadrže i softver za razdvajanje IA od hardvera. Zadatak ovakve arhitekture je izvršavanje IA i komunikacija sa okruženjem.

Računarski program je ustvari funkcija kojom je implementirano preslikavanje **'sekvenca opažaja → akcija'** i može se prikazati PAGE deskripcijom IA:

- ***P****ercepts* (opažaji),
- ***A****ctions* (akcije),
- ***G****oals* (ciljevi-mjere performansi),
- ***E****nvironment* (okruženje).

Ciljevi ne moraju da budu predstavljeni unutar agenta.

PRIMJER *PAGE deskripcije:*

- tip IA - medicinski dijagnostički sistem,
- opažaji - simptomi, laboratorijski nalazi, … ,
- akcije - pitanja, testovi, terapije, … ,
- ciljevi - ozdravljenje, minimizacija troškova, … ,
- okruženje - pacijent, bolnica, laboratorija, ...

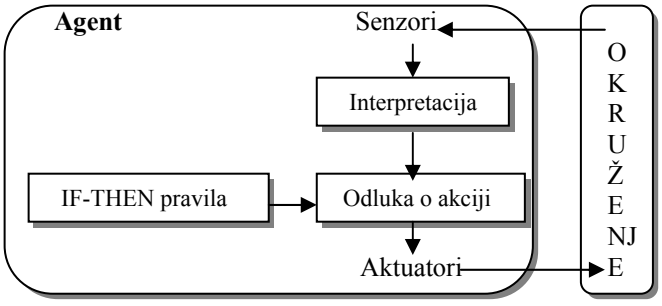
12.3.5. Programi inteligentnih agenata

U narednom tekstu će biti opisani:
- najjednostavniji opšti program IA,
- prosti refleksni agenti,
- refleksni agenti sa stanjem,
- agenti sa ciljevima,
- agenti zasnovani na znanju.

PRIMJER: *Najjednostavniji opšti program IA.*

function Skeleton-Agent(*opažaj*) returns *akcija*
    static: *memorija*, opis okruženja
    memorija ← Ažuriraj(*memorija*, *opažaj*)
    akcija← Odaberi-Najbolju-Akciju(*memorija*)
    memorija← Ažuriraj(*memorija*, *akcija*)
    return *akcija*

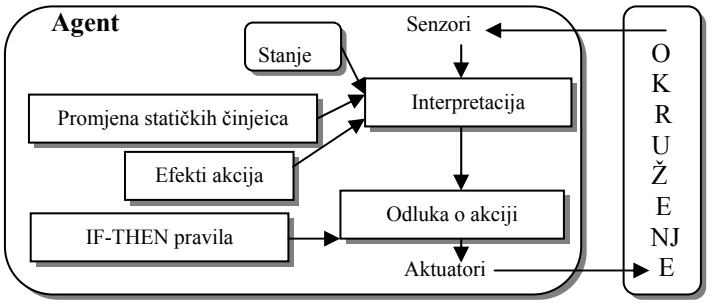
Prosti refleksni agenti



Slika 12.8. Grafički prikaz prostog refleksnog agenta.

function Prost-Refleksni-Agent(*opažaj*)
    returns *akcija*
    static: *pravila*, skup IF-THEN pravila
    stanje ← Interpretiraj(*opažaj*)
    pravilo ← Prepoznavanje(*stanje*, *pravila*)
    akcija ← THEN-akcija(*pravilo*)
    return *akcija*

Refleksni agenti sa stanjem

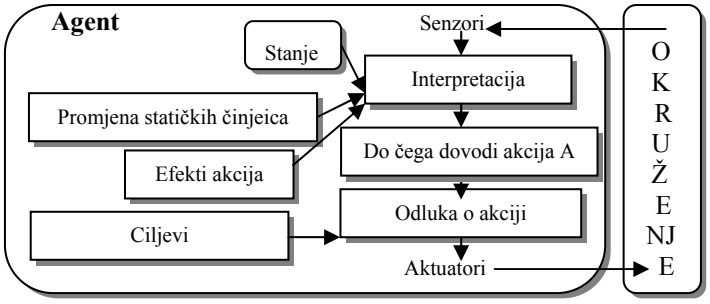


Slika 12.9. Grafički prikaz refleksnog agenta sa stanjem.

```
function Refleksni-Agent-sa-Stanjem(opažaj)
        returns akcija
        static:  pravila, skup IF-THEN pravila
                 stanje, trenutno stanje u okruženju
        stanje ← Ažuriraj(stanje, opažaj)
        pravilo ← Prepoznavanje(stanje, pravila)
        akcija ← THEN-akcija(pravilo)
        stanje ← Ažuriraj(stanje, akcija)
        return akcija
```

Agenti sa ciljevima

Ovakvi agenti odgovaraju zadacima pretraživanja i planiranja. Prisutna je neophodnost informacije o ciljevima kod agenata sa ciljevima. Vrši se razmatranje **potencijalnih stanja** (”Do čega dovodi akcija A?”).



Slika 12.10. Grafički prikaz agenta sa ciljevima.

```
function Agent-za-Rješavanje-Problema(o) returns akcija
    inputs: o, opažaj
    static: sa, sekvenca akcija, inicijalno prazna
            stanje, trenutno stanje u okruženju
            c, cilj, tj. skup stanja u kojima je cilj zadovoljen,
                    inicijalno prazan
            problem, formulacija problema
    ! Primjer ovakvog IA: putovanje od mjesta A do mjesta B
    ! Rješavanje problema (Problem solving) je
    ! samo jedna moguća primjena ovakvih IA !!!
    stanje ← Ažuriraj(stanje, o)
    if sa je prazna sekvenca then
            c ← Formulacija-Cilja(stanje)
            problem ← Formulacija-Problema(stanje, c)
                    ! Odluka o tome koja stanja i akcije se razmatraju
            sa ← Pretraživanje(problem)
    akcija ← Preporuka(sa, stanje)
    sa ← Ostatak(sa, stanje)
    return akcija
```

Agenti zasnovani na znanju

Ovi agenti posjeduju znanje o svom okruženju i mogućnost zaključivanja o svojim akcijama. Prihvataju nove zadatke u obliku eksplicitno zadanih ciljeva. Prilagođavaju se promenama u okruženju ažuriranjem svog znanja i povećavaju svoje znanje komunikacijom sa okruženjem ili učenjem.

Postavlja se pitanje "Šta treba da bude sadržaj znanja ovih IA?". Odgovori su:

- trenutno stanje u okruženju,
- načini indirektnog zaključivanja o okruženju, polazeći od opažaja,
- mogućnosti promjene statičkih činjenica,
- koji krajnji cilj treba ostvariti,
- kakav efekat postižu akcije ako se izvrše pod različitim okolnostima.

Uloga baze znanja:

- predstavlja centralnu komponentu ovakvih agenata,
- iskazi se nalaze u bazi znanja (činjenice o okruženju),
- sadrže jezik za predstavljanje znanja,

- dodavanje novih iskaza u bazu znanja,
- zadavanje upita o poznatim činjenicama,
- zaključivanje nad bazom znanja vrši mehanizam zaključivanja.

function KB-Agent (*opažaj*) returns *akcija*
  static: *KB*, baza znanja, može da sadrži prethodno znanje
    t, brojač, inicijalno 0
  Dodaj(*KB*, Generiši-Iskaz-o-Opažaju(*opažaj*, *t*))
  akcija ← Upit(*KB*, Generiši-Upit-o-Akciji(*t*))
  Dodaj(*KB*, Generiši-Iskaz-o-Akciji(*akcija*, *t*))
  t ← *t* + *1*
  return *akcija*

Okruženja inteligentnih agenata

Primjeri okruženja mogu da budu:
- pacijent, bolnica, laboratorija,
- slike dobijene sa orbitalnih satelita,
- pokretna traka,
- rafinerija nafte,
- grupa studenata.

Okruženja mogu da budu: "stvarna" i "vještačka" (računarska) okruženja.

Simulator okruženja 1:

procedure Okruženje(*stanje*, Ažuriraj, *agenti*, *kraj*)
inputs: *stanje*, početno stanje u okruženju
  Ažuriraj, funkcija za modifikaciju okruženja
  agenti, skup agenata
  kraj, predikat za ispitivanje završetka

Simulator okruženja 2:

repeat
  for each *agent* in *agenti* do
    opažaj [agent] ← Učitaj-Opažaj(*agent*, *stanje*)
  end

```
    for each agent in agenti do
            akcija [agent] ← Program [agent] (opažaj [agent])
    end
    stanje ← Ažuriraj(akcije, agenti, stanje)
until kraj(stanje)
function    Okruženje-Sa-Mjerenjem-Performansi
            (stanje, Ažuriraj, agenti, kraj, Izmjeri-Performanse)
            returns ocjene
    local variables: ocjene, vektor ocjena za agente, inicijalno 0
    repeat
            ! Obe for each petlje kao u proceduri Okruženje
            stanje ← Ažuriraj(akcije, agenti, stanje)
            ocjene ← Izmjeri-Performanse(akcije, agenti, stanje)
    until kraj(stanje)
```

Uloga promenljive *stanje:*

- simulator okruženja ažurira kompletno stanje (funkcija "Ažuriraj" u prethodna dva primjera),
- IA nema potpun pristup stanju okruženja,
- IA ažurira stanje okruženja samo na osnovu svojih opažaja,
- funkcija "Ažuriraj" u programu Refleksni-Agent-sa-Stanjem je funkcija agenta.

Email agent

U narednom tekstu će biti razmatran agent **Maxims** (MIT, USA). Preporuke u vezi sa email porukama:

- davanje prioriteta,
- brisanje,
- proslijeđivanje,
- sortiranje,
- arhiviranje.

Email agent može da bude refleksni agent sa stanjem. Tehnika učenja agenta: *memory-based reasoning,* se sastoji u slijedećem:

- posmatranje akcija korisnika,
- memorisanje parova **situacija - akcija** (opis situacije: *From:*, *To:*, *Cc:*, *Subject:*, … ; opis akcije: *Save* , *Reply* , *Delete* , … .),

kao i poređenje novoprispjele poruke sa memorisanim parovima.

Metrika koja se koristi pri poređenju:
- svakom parametru iz opisa situacije dodjeljuje se određeni težinski faktor,
- **rastojanje** u prostoru poruka može biti:
  - razlika vrijednosti pojedinih parametara za novoprispjelu poruku i opise svih ranijih situacija,
  - svaka takva razlika množi se odgovarajućim težinskim faktorom,
  - akcija: ona koja odgovara minimalnom rastojanju.

Određivanje težinskih faktora vrši **Maxims:**
- periodično ispitivanje korelacije pojedinačnih parametara i korisnikovih akcija,
- detektovane korelacije se koriste kao težinski faktori.

Određivanje faktora povjerenja u pogledu preporuke koja se daje korisniku:
- da li bi iste preporuke bile dane za susjedna rastojanja,
- kolika je razlika minimalnog rastojanja i susjednih rastojanja,
- koliko je primjera memorisano (tj. kolika je preciznost izračunatih korelacija).

Pragovi faktora povjerenja:
- *do it* prag
  - ako je faktor povjerenja iznad njega, akcija se vrši automatski,
  - o izvršenoj akciji piše se izvještaj, koji korisnik može uvijek da zahtjeva na uvid;
- *tell me* prag
  - ako je faktor povjerenja iznad njega, akcija se vrši samo uz potvrdu korisnika;
- korisnik postavlja vrijednosti pragova:
  - ako **Maxims** nekontrolisano briše poruke, *do it* prag se postavlja na maksimalnu vrijednost,
  - ako **Maxims** rijetko daje preporuku, *tell me* prag se postavlja na nižu vrijednost.

Obučavanje **Maxims**-a:
- bez obučavanja **Maxims** bi sporo sticao znanje,
- hipotetički primjeri koje kreira korisnik:
  - variranje pojedinih parametara u istom primjeru,

  - zadavanje i lakih i teških primjera.
- slanje zadanog primjera drugim srodnim agentima preko email-a,
  - postepeno izgrađivanje povjerenja prema drugim IA.

Komunikacija između agenata

Moguća upotreba jezika za komunikaciju IA:
- KQML - *Knowledge Query and Manipulation Language.*

Agenti posrednici, uslužni agenti (*Facilitators and Mediators*).

Postoji potreba za zajedničkim jezikom za komunikaciju IA:
- zajednička sintaksa (za sada ne postoji)
  - pokušaji: LOOM, KIF, prošireni SQL, ... ,
  - zajednička semantika (ontologije),
  - zajednička pragmatika [ko sa kim komunicira i kako pronaći "sagovornika" (identifikacija), kako inicirati i održavati komunikaciju]

.

Jezik KQML:
- jezik i skup protokola za komunikaciju IA (zajednička pragmatika),
- podržava raznovrsne arhitekture i tipove IA,
- podržava i komunikaciju IA sa drugim klijentima i serverima,
- koristi standardne protokole za razmenu informacija (TCP/IP, email, HTTP, CORBA).

U ovom poglavlju:

- ***Inteligentni sistemi***
- ***Inteligentne BP***
- ***Hibridni IS***
- ***Inteligentne telekomunikacione mreže***

13

SAVREMENI INTELIGENTNI SISTEMI

13.1. OPŠTE O INTELIGENTNIM SISTEMIMA

Postoji veliki broj definicija inteligentnih sistema. Nijedna od definicija nije u potpunosti prihvatljiva. Razlog je u činjenici da pojam inteligencije još uvijek nije u potpunosti objašnjen. Inteligencijom se bavi niz naučnih disciplina, u prvom redu psihologija, filozofija, neurologija i druge (o čemu je u ovoj knjizi već bilo govora), a u posljednjim decenijama inteligencijom, tačnije rečeno vještačkom inteligencijom, se sve više bave i računarske nauke. Računarske nauke proučavaju mogućnost da se pomoću računara ostvari opažanje, rasuđivanje i ponašanje, odnosno da se automatizuju inteligentne akcije.

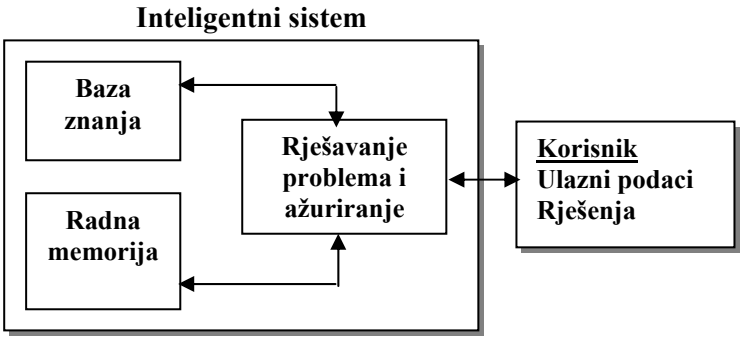
Računarskim programima je svojstveno simboličko predstavljanje znanja o nekom području realnog sistema, odnosno stvaranje uproštene vizije realnog sistema. Računarski programi manipulišu pretežno nenumeričkim simbolima i to je jedno od osnovnih svojstava tih programa. Pri tome često koriste heuristiku, što im omogućava da rješavaju probleme za čije rješenje ne postoje algoritmi (poznati niz koraka koji vodi rješenju problema). **Heuristika** je skup pravila domišljatog nagađanja, koja usmjeravaju i ograničavaju područja traganja za rješenjem. Grčki glagol *heuriskein* znači otkriti.

Struktura inteligentnih sistema se može prikazati kao na slici 13.1. U **bazi znanja** se nalazi kodirano znanje sistema, predstavljeno pomoću jedne ili više tehnika za predstavljanje znanja. Tu se nalazi logički integrisana kolekcija

---

Materijal izložen u tačkama 13.1. i 13.2. najvećim dijelom se zasniva na knjizi Devedžić, V., "Inteligentni informacioni sistemi", 2000.

međusobno povezanih elemenata znanja, koji se koriste u procesu rješavanja problema.



Slika 13.1. Struktura inteligentnih sistema.

Radna memorija sadrži podatke i činjenice unijete spolja u sistem, kao i one do kojih je sistem došao tokom rješavanja problema. Ulaz i izlaz sistema služe za unošenje neophodnih podataka u radnu memoriju, izvještavanje o djelimičnim i konačnim rješenjima problema, pokretanje i zaustavljanje procesa rješavanja problema, i sl.

Proces rješavanja problema počinje unošenjem u radnu memoriju podataka za definisanje početnog stanja sistema, a završava se kada sistem dođe u ciljno stanje. Komunikacija može biti sa korisnikom ili sa drugim sistemima u okruženju: sistemima za upravljanje bazama podataka (SUBP), senzorima, relacijama, raznim serverima, i slično. Proces rješavanja problema se opisuje kao rasuđivanje ili zaključivanje.

Inteligentni sistemi se mogu podjeliti u tri grupe:

- **autonomni sistemi**, koji samostalno planiraju i izvode akcije u realnom okruženju,
- **sistemi za podršku**, koji učestvuju u pripremi, ali samostalno ne moraju da djeluju,
- **savjetodavni sistemi**, koji posreduju u prikupljanju informacija, uz pomoć kojih je uspješnije odlučivanje .

Pogodnosti inteligentnih sistema su:

- komunikacija sa njima mora biti jednostavna i prirodna,

- moraju biti prilagođeni radu sa opštim i problemski usmjerenim znanjem,
- moraju biti u stanju da objasne svoje akcije, moraju biti adaptivni, učiti i poboljšavati svoje sposobnosti.

Ostale pogodnosti inteligentnih računarskih sistema su:
- da posjeduju adaptivno ciljno ponašanje,
- da uče na osnovu iskustva,
- koriste veliki obim znanja,
- posjeduju sposobnost samostalnog zaključivanja,
- komuniciraju na što prirodniji način,
- tolerišu greške pri komunikaciji,
- odgovaraju u realnom vremenu.

13.2. INTELIGENTNE BAZE PODATAKA

13.2.1. Otkrivanje znanja u bazama podataka (KDD)

Uobičajeno pitanje koje se postavlja kako sa stalnim narastanjem podataka u bazama podataka (BP) te podatke koristiti na najoptimalniji način. Pored koncepata podatak i informacija, javlja se i novi koncept: znanje. Neminovno se traže odgovori na sljedeća pitanja:
- kako otkrivati znanja u BP,
- objašnjenje popularnosti otkrivanja znanja u BP,
- odnos otkrivanja znanja u BP i ostale discipline.

Otkrivanje znanja u bazama podataka (*Knowledge Discovery in Databases*, **KDD**) se susreće sa slijedećim izazovima:

1. **Uvećavanje baza podataka:**
   - smeštanje podataka je sve jeftinije,
   - vjerujuje se da se uvćanjem podataka mogu da sačuvaju neke potencijalno vrijedne informacije,
   - dupliranje informacija na svakih 20 mjeseci (1989).

2. **Sirovi podaci su rijetko od koristi.**

3. **Analiza podataka je često samo manuelna:**
   - analitičari u ulozi sofisticiranih ”procesora upita”,
   - problemi nastaju uslijed povećanja količine podataka.

4. **Automatizacija analize podataka:**
   - otkrivanje znanja u BP,
   - traganje kroz podatke.

Otkrivanja znanja u BP (KDD)

1. **Traganje kroz podatke, (*Data Mining*, DM):**
   - integracija različitih metoda mašinskog učenja i otkrivanja znanja,
   - integracija sistema zasnovanih na znanju i objektno-orijentisanih sistema,
   - integracija sistema zasnovanih na znanju i statističkih metoda,
   - otkrivanje kauzalnosti između podataka,
   - korištenje domenskog znanja u procesu KDD,
   - sumiranje i interpretacija otkrivenog znanja,
   - uključivanje otkrivenog znanja u proces KDD,
   - odgovaranje na upite visokog nivoa: optimizacija upita i otkrivanje pravila za transformaciju upita,
   - otkrivanje evolucije podataka,
   - učenje koncepata,
   - evaluacija otkrivenog znanja,
   - otkrivanje neegzaktnih pojmova u strukturiranim podacima,
   - tretman otkrivenog neizvesnog znanja,
   - indukcija meta znanja o KDD,
   - izbegavanje grešaka u BP,
   - softverska okruženja za KDD,
   - vizuelizacija KDD,
   - etička i pravna pitanja u vezi sa KDD, i dr.

PRIMJERI KDD:

1. American Airlines pretražuje BP o korisnicima ”Frequent Flyer” programa. Istaknuti korisnici njihovih usluga dobijaju promotivne nagrade.

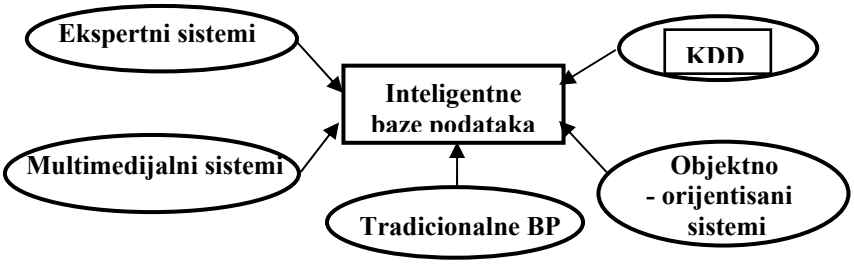
2. General Motors koristi BP o automobilskim nesrećama. Na osnovu toga se vrši razvoj dijagnostičkog ES za razne modele automobila.

3. Banke koriste podatke o kreditima i zajmovima svojih klijenata, što poboljšava metode predikcije poslovanja.

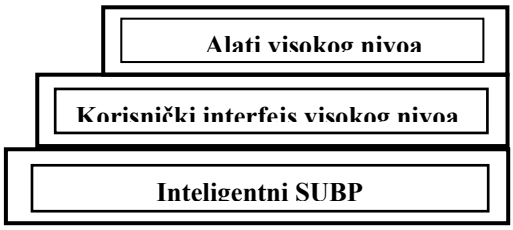
KDD i ostale discipline

1. **Baze podataka:**
   - smještanje podataka, modifikacije, upiti.

2. **Ekspertni sistemi:**
   - indukcija pravila na osnovu primjera eksperta,
   - razlika KDD od indukcije kod ES (primjeri eksperta su kvalitetniji nego "prosječni" podaci u BP; primeri eksperata obično pokrivaju *tipične* slučajeve).



Slika 13.2. KDD i ostale discipline.



Slika 13.3. KDD i ostale discipline.

3. **Inteligentne baze podataka:**
   - alati visokog nivoa,
   - korisnički interfejs visokog nivoa,
   - inteligentni SUBP:
     - objektno-orijentisane multimedijalne informacije,
     - deduktivne baze podataka,
   - korisnički interfejs visokog nivoa:
     - inteligentna interakcija sa korisnicima,
     - model zadatka i okruženja,
   - alati visokog nivoa:
     - agenti za inteligentno pretraživanje,
     - kontrola kvaliteta podataka,
     - kontrola integriteta podataka,
     - … .

4. **Statistika:**
   - omogućuje analizu podataka,
   - zahteva učešće korisnika u analizi podataka,
   - daje rezultate koje je često teško interpretirati,
   - nije pogodna za analizu kompleksnijih struktura podataka,
   - ne omogućuje korištenje domenskog znanja.

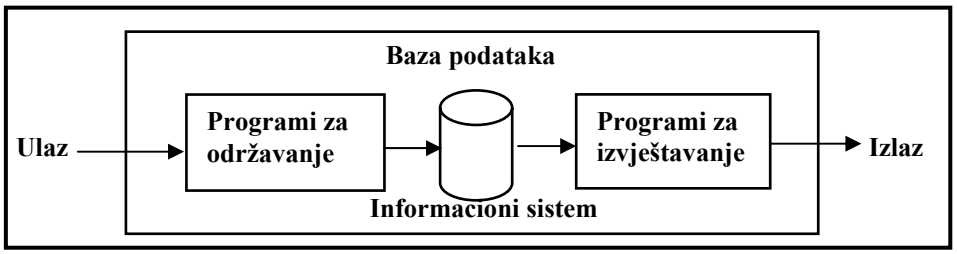
5. **Mašinsko učenje:**
   - kolekcija primjera (instanci) u datoteci (bazi) - tipično nekoliko stotina primjera,
   - primjer (instanca) -vektor fiksne dužine (slog),
   - algoritam učenja na osnovu skupa primjera generiše neki iskaz (npr. opis nekog pojma),
   - klasične pretpostavke:
     - statični primjeri, kompletne informacije, bez grešaka.

6. **Naučno otkriće** (*Scientific discovery*):
   - jasniji cilj otkrića nego kod KDD,

- kontrolisaniji proces nego kod KDD,
- veća zasnovanost na eksperimentima,
- visoka fokusiranost eksperimenata,
- mogućnost redizajniranja eksperimenata.

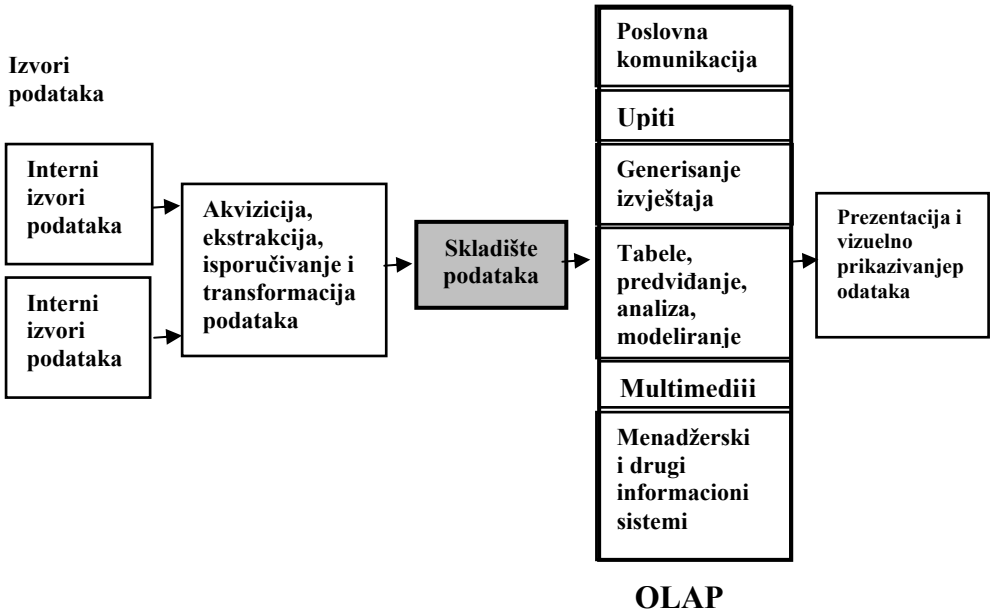
7. **Skladišta podataka** (*Data Warehouses*):
   - prikupljanje i prečišćavanje podataka u cilju *on-line* analize i podrške odlučivanju,
   - integracija nasleđenih BP,
   - OLAP (*On-Line Analytical Processing*):
     - o multidimenzionalna interaktivna analiza podataka,
     - o superiornost u odnosu na SQL,
     - o prilagođavanje podataka korisniku kroz generalizaciju i eksplicitno prikazivanje konteksta.

Klasični pretraživači (poput Alta Viste) se sastoje od tri komponente: indeksa, programa za pretraživanje i programa za kreiranje indeksa. Programi za kreiranje indeksa, tzv. "roboti" ili "pauci" (*spiders, crawlers*), automatski skeniraju razne Web servere i stranice i formiraju indekse URL-adresa ključnih riječi, linkova i tekstova. Oni takođe prate i linkove na pojedinim stranicama da bi pronašli druge relevantne stranice, a takođe se povremeno vraćaju na servere i stranice da bi provjerili da li se na njima nešto promjenilo.



Slika 13.4. Opšta struktura informacionih sistema.

Kada korisnik zada upit, program za pretraživanje pretražuje indeks koji su "roboti" napravili i koji se stalno osvježava. U rezultujućem izvještaju se prikazuju sranice iz indeksa, koje sadrže ključne riječi iz korisnikovog upita. Prikazane stranice su često rangirane na način koji omogućava tzv. OLAP procesiranje podataka (OLAP – *On Line Analytical Procesing*), što je prikazano na slici 13.5.



Slika 13.5. Skladište podataka i OLAP procesiranje.

Poređenjem slika 13.4. i 13.5. se može uočiti da centralizovano skladište podataka vrši svojevrsnu reorganizaciju, transformaciju i objedinjavanje original-nih podataka. Originalni podaci potiču iz BP informacionog sistema i iz drugih izvora, eksternih u odnosu na posmatrani IS. Gledano iz ugla OLAP procesiranja, svi podaci naizgled potiču iz jedinstvene BP i već su organizovani na način koji odgovara funkcijama OLAP sistema. Cilj uvođenja skladišta podataka je da oformi repozitorijum podataka, u kojima su operativni podaci dostupni aplikacijama.

Izvori podataka za inteligentne BP

- odlučivanje uz prisustvo neizvesnosti,
- prikupljanje znanja za ekspertne sisteme,
- prepoznavanje *pattern*-a,
- neuronske mreže,
- vizuelni prikaz podataka.

13.2.2. Definicije

Otkrivanje znanja

Netrivijalna ekstrakcija informacija iz podataka, i to informacija koje su:

- implicitne,
- prethodno nepoznate, kao i
- potencijalno korisne.

Informacija mora da bude u obliku *pattern*-a koji su razumljivi za korisnika (*pattern* – šablon, opšte rješenje opšteg problema u danom kontekstu).

Pattern

- F - skup podataka,
- L - jezik,
- C - mjera izvesnosti.
- *Pattern:*
  - iskaz S u jeziku L o odnosima između podataka podskupa F_S skupa F,
  - iskaz S ima izvjesnost *c*,
  - iskaz S mora da bude u izvesnom smislu jednostavniji od nabrajanja svih podataka u F.

Pattern mora da bude iskazan nekim jezikom visokog nivoa:

```
IF Starost < 25 AND Položen-vozački-ispit = False
THEN Krivac-u-nesreći = Yes   (CF = 0.2)
```

Pattern-i se mogu koristiti kao ulaz u druge programe (npr. ES ili za optimi-zaciju upita).

Znanje

- znanje je sadržano u *pattern*-u koji je značajan i dovoljno izvjestan,
- meru značaja i kriterijum izvesnosti specificira korisnik.

Otkriveno znanje

- izlaz iz programa koji ispituje neki skup podataka u BP i generiše *pattern*-e.

Izvjesnost

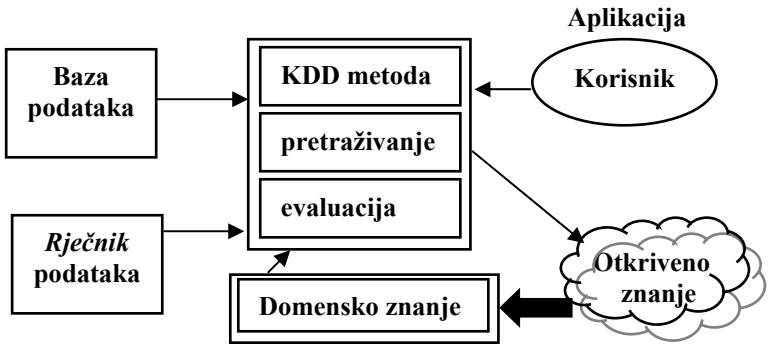
- stepen povjerenja u otkriveno znanje:
  - bez dovoljne izvesnosti *pattern* nije znanje,
  - otkriveno znanje rijetko važi za sve podatke;
- aspekti izvjesnosti:
  - integritet podataka,
  - veličina uzorka podataka nad kojim je vršeno KDD,
  - stepen podrške od domenskog znanja.

Mjera značaja otkrivenog znanja

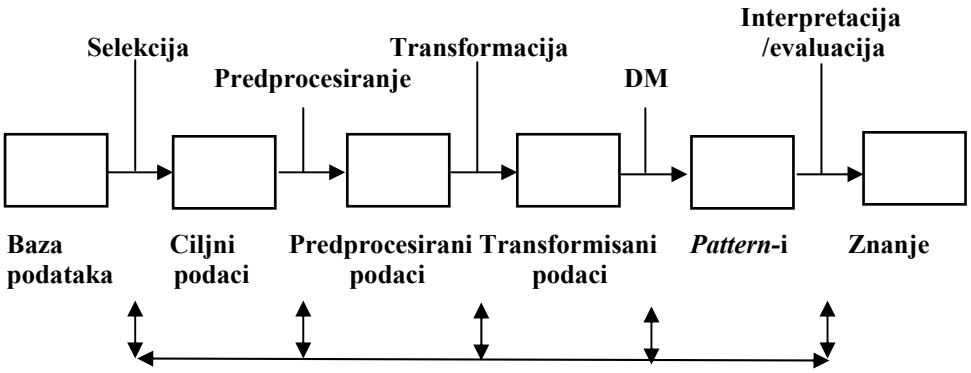
- znanje predstavljaju samo **interesantni *pattern***-i,
- *pattern* je značajan (interesantan) ako je nov, koristan i netrivijalan:
  - nov *pattern* - nov za korisnika, a ne za sistem,
  - koristan *pattern* - koristan za taj zadatak (pomaže da se zadatak bolje riješi),
  - netrivijalan *pattern* - dovoljno složen (ne predstavlja samo prostu statistiku).

13.2.3. Otkrivanje znanja u BP - KDD kao proces

Okruženje procesa otkrivanja znanja u BP prikazano je na slikama 11.6. i 11.7.



Slika 13.6. Okruženje procesa KDD.



Slika 13.7. Podaci i aktivnosti u procesu KDD.

13.2.4. Traganje kroz podatke

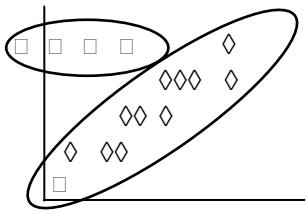
Traganje kroz podatke (*Data Mining*, DM) će biti opisano u narednom tekstu.

DM zadaci

- DM zadaci - vrste i ciljevi DM aktivnosti,
- Klasifikacija:

- o određivanje pripadnosti podatka nekoj od unaprjed definisanih klasa,
- o bilo koje polje može da posluži za definisanje novih klasa - polje *Region*, klase *Sever*, *Jug*, *Istok*, *Zapad,*
- o klase moge da opišu i domensko znanje;

- Određivanje klastera:
  - o formiranje klasa na osnovu podataka (kod klasifikacije klase su unaprjed poznate),



  - o korištenje raznih mjera sličnosti podataka,
  - o korištenje gustina verovatnoće;
- Sumiranje:
  - o određivanje zajedničkih karakteristika klasa u kompaktnom obliku:

PRIMER: S*vi olimpijski pobjednici u trci na 100 m su bili tamnoputi i imali su manje od 28 godina.*

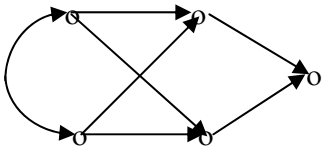
  - o mogućnost generisanja pravila sumiranja,
  - o sofisticiranije tehnike omogućuju generisanje funkcionalnih zavis-nosti između u podataka.

Modeliranje zavisnosti

- Modeliranje zavisnosti:
  - o opisivanje važnih zavisnosti među podacima,
  - o zavisnost postoji ako se može izvršiti predikcija jednog podatka na osnovu drugog ($A \rightarrow B$),
  - o strukturni nivo modela: specifikacija **podataka** među kojima postoji zavisnost,
  - o kvantitativni nivo modela: **jačina zavisnosti** na nekoj brojnoj skali.

Zavisnost velike jačine:

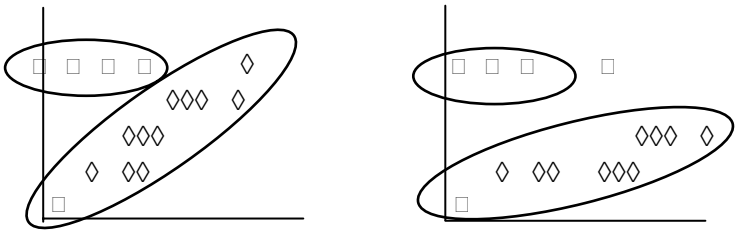
- obično ne daje nove ili interesantne *pattern*-e,
- ukazuje na unutrašnju strukturu domena,
- automatska detekcija takvih zavisnosti može da koristi drugim DM zadacima:
  - funkcija zavisnosti obično je probabilistika ($A \rightarrow B$, $CF = 0.93$),
  - kolekcija međusobno povezanih zavisnosti čini graf zavisnosti (*dependency graph*):



  - promjena vrijednosti jednog polja može se pratiti kroz graf zavisnosti u cilju objašnjenja promjene.

Detekcija promjene ili odstupanja

- vrste promjena i odstupanja:
  - podaci koji ne pripadaju nijednoj klasi / klasteru,
  - promjena vrijednosti skupa podataka tokom vremena,
  - značajna promjena srednje vrijednosti, gradijenta, …
  - odsustvo podudarnosti modela i opservacija;



- detekcija se vrši u odnosu na neke referense,
- omogućava detekciju *pattern*-a koji su potencijalno nezanimljivi,
- važno pitanje: kada je odstupanje dovoljno veliko?

- primjena statističkih mjera,
- korištenje domenskog znanja,
- učeše korisnika.

Diskriminacija

- identifikacija osobina koje su dovoljne za razlikovanje dvije klase: osobe koje su se sunčale su preplanule.

Određivanje referensi

- utvrđivanje osobina i tipičnih vrijednosti koje mogu da služe kao referense za poređenje: tipičan biznismen ima strani automobil, mobilni telefon, … .

Analiza sekvenci

PRIMER: *Analiza vremenskih nizova.*

Određivanje asocijativnih pravila

- utvrđivanje korelacije između dva ili više polja.

PRIMJER**:** *Od svih slogova koji sadrže A, B i C, 72% takođe sadrži i D i E (kao vrijednosti nekih polja).*

- korelacije moraju da budu iznad nekog praga.

Analiza veza *(Link analysis)*

- utvrđivanje korelacije između slogova,

PRIMJER: *Koji se artikl kupuje uz neke druge.*

- sličnost sa određivanjem asocijativnih pravila (ponekad se poistovećuju).

Analiza prostornih zavisnosti

PRIMJERI: *Analiza podataka u GIS sistemima, astrofizičkih, ekoloških i drugih podataka.*

Određivanje karakterističnih putanja *(Mining path traversal patterns).*

PRIMJER: *Pattern-i putanja u korištenju WWW-a, odnosno korisnikovog ponašanja u tome, omogućavaju bolji dizajn sistema uz poznavanje tih pattern-a. Osnovna teškoća: neka mjesta su na putanji zbog svoje lokacije, a ne zbog sadržaja.*

13.2.5. Algoritmi za traganje kroz podatke

- DM algoritmi - procedure za ekstrakciju *pattern*-a iz podataka,
- identifikacija *pattern*-a,
- predstavljanje i opisivanje *pattern*-a,
- obično se ne radi o algoritmima specifičnim samo za DM:
  - preuzimanje algoritama iz drugih disciplina,
  - adaptiranje tih algoritama za DM.

Identifikacija pattern-a

- otkrivanje kolekcija (klasa) slogova koji imaju nešto zajedničko,
- numerički algoritmi:
  - maksimiziranje sličnosti unutar klase,
  - minimiziranje sličnosti između različitih klasa,
  - Euklidske mjere rastojanja za izračunavanje sličnosti,
  - dobri su samo za numeričke podatke,
  - ne mogu da koriste domensko znanje (npr. znanje o obliku klastera).

Određivanje konceptualnih klastera

- koristi sličnost atributa, kao i konceptualnu koheziju (definisanu domen-skim znanjem),
- može da se koristi i za strukturirane podatke.

Interaktivno određivanje klastera

- kombinovanje mogućnosti računara i znanja i vizuelnih mogućnosti korisnika.

Predstavljanje i opisivanje pattern-a

- pravila,
- stabla odlučivanja,
- linearni modeli,
- neuronske mreže,
- genetički algoritmi,
- korištenje statističkih metoda.

Predstavljanje i opisivanje pattern-a

- korištenje domenskog znanja,
- metod najbližeg susjeda,
- korištenje već rješavanih slučajeva,
- *Bayes*-ove mreže.

Ne postoji univerzalno dobar DM algoritam.

13.2.6. Algoritam Apriori

- Algoritam Apriori,
- Algoritam DHP,
- Generički DM algoritam,
- Algoritam CDP, i
- Algoritam MPTP,

Namjena traganja kroz podatke (DM zadatak) je određivanje asocijativnih pravila.

Suština je otkrivanje pravila oblika $A \Rightarrow B$, tj.

$$A_1 \wedge A_2 \wedge \ldots \wedge A_m \Rightarrow B_1 \wedge B_2 \wedge \ldots \wedge B_n$$

Smisao pravila je da slog koji sadrži skup elemenata A sadrži i skup elemenata B,

$$A \cap B = 0,$$

Za pravila se vezuju faktori povjerenja i podrške.

Faktor povjerenja

c - *c* % onih slogova koji sadrže A sadrže i B. Označava jačinu implikacije $A \Rightarrow B$.

Faktor podrške

s - *s* % slogova u bazi sadrže $A \cup B$. Označava frekvenciju implikacije $A \Rightarrow B$.

Jaka asocijativna pravila

Jaka asocijativna pravila su pravila sa visokim faktorima povjerenja i podrške.

Jaki skupovi elemenata

Jaki skupovi elemenata su skupovi asociranih elemenata ($\{A\ B\} \rightarrow A \Rightarrow B$) sa *s* iznad neke unaprijed zadane vrijednosti. Broj elemenata u takvim skupovima može da bude i veći od 2.

- Prevođenje jakih skupova elemenata u asocijativna pravila je direktno;
- Algoritam Apriori služi za otkrivanje jakih skupova elemenata.

Nadovezivanje skupova skupova

Nadovezivanje skupova skupova ima operator $\otimes$ za skupove skupova od po jednog elementa.

- $S_1 \otimes S_1$ (u opštem slučaju),
- $\{\{A\}, \{B\}, \{C\}\} \otimes \{\{A\}, \{B\}, \{C\}\} = \{\{A\ B\}, \{A\ C\}, \{B\ C\}\}$,

Za skupove skupova od po *k* elemenata:

- $S_k \otimes S_k = \{ X \cup Y \mid X,Y \in S_k, \mid X \cap Y \mid = k - 1 \}$
- $S_2 = \{\{A\ C\}, \{B\ C\}, \{B\ E\}, \{C\ E\}\}$, $S_2 \otimes S_2 = \{ B\ C\ E \}$.

PRIMJER: *Primjer principa rada algoritma Apriori. Baza podataka D.*

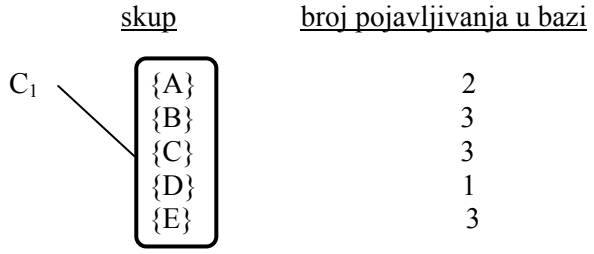
ID	elementi
10	A C D
20	B C E

30	A B C E
40	B E

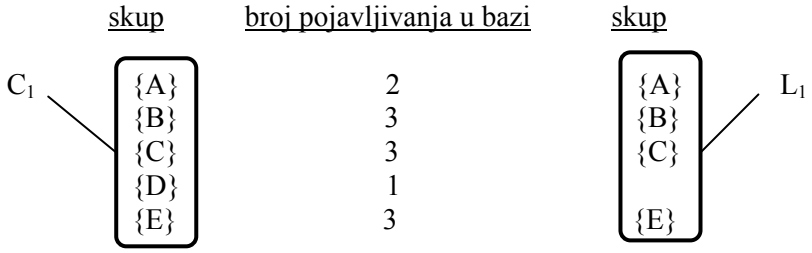
Minimalni zahtevani faktor podrške s = 50%.

PRIMJER: *Primjer principa rada algoritma Apriori.*

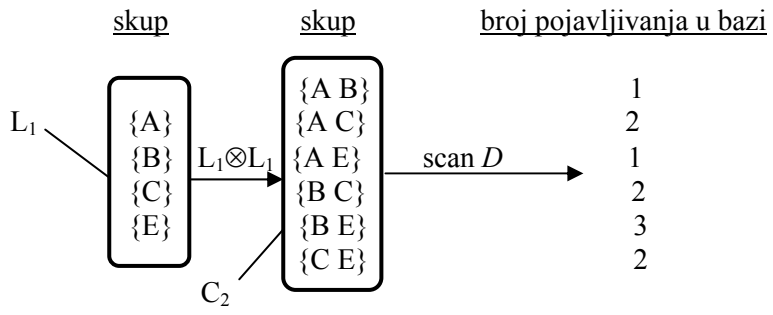
Korak 1 *- pravljenje skupa skupova kandidata, od po jednog elementa (skup C_1).*



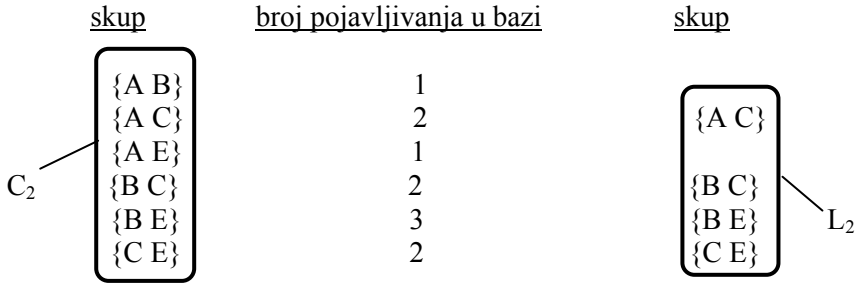
Korak 2 *- pravljenje skupa jakih skupova od po jednog elementa (skup L_1).*



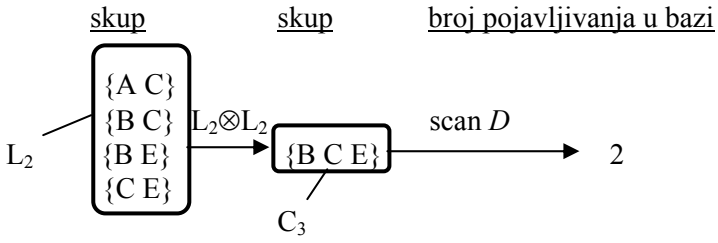
Korak 3 *- pravljenje skupa skupova kandidata, od po dva elementa, pomoću $L_1 \otimes L_1$ (skup C_2).*



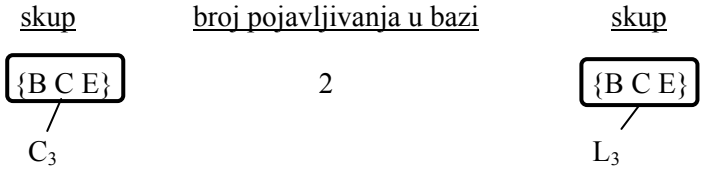
Korak 4 *- pravljenje skupa jakih skupova od po dva elementa (skup L_2).*



Korak 5 *- pravljenje skupa skupova kandidata, od po tri elementa, pomoću $L_2 \otimes L_2$ (skup C_3).*



Korak 6 *- pravljenje skupa jakih skupova od po tri elementa (skup L_3)*




```
function Formiraj-Jake-Skupove(D, s)
                    returns jaki-skupovi
   C1 ← Nabroj-Elemente-iz-Baze(D )
   be ←1, broj elemenata u jakim skupovima
   C1 ← Scan(D, be, C1)
   L1 ← Filtriraj(C1, s)
   jaki-skupovi ← L1
   k ← 1
   L ← L1
while (Broj-Elemenata(L) > 0)
   k ← k + 1
   be ← be + 1
   C ← Nadoveži-Skupove(L, be)
   C ← Scan (D, be, C)
   L ← Filtriraj(C, s)
   jaki-skupovi ← jaki-skupovi ∪ L
end while
return jaki-skupovi
```

Redukcija skeniranja

C_3 se može izračunati i iz $C_2 \otimes C_2$, bez prethodnog izračunavanja L_2 i $L_2 \otimes L_2$.

- $C_3' = C_2 \otimes C_2$, $C_3 = L_2 \otimes L_2$
- $|C_3'| > |C_3|$, u opštem slučaju.

Ako razlika $|C_3'|$ i $|C_3|$ nije velika, ponekad i C_2 i C_3' mogu da stanu u radnu memoriju. Ako i C_2 i C_3' mogu da stanu u radnu memoriju onda se L_2 i L_3 mogu izračunati odjednom:

- slijedeće skeniranje baze podataka koristi se za izračunavanje i L_2 i L_3,
- ušteda: obavlja se jedno skeniranje manje.

Mogućnost određivanja svih L_k u samo dva skeniranja baze podataka:

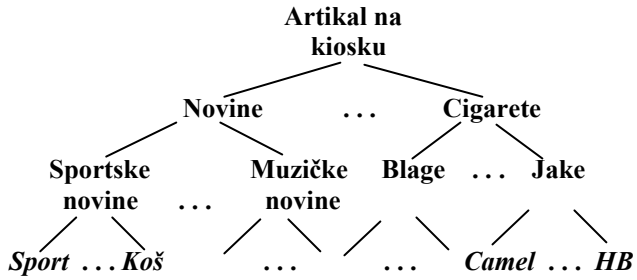
- uslov 1: C_k' se izračunava iz C_{k-1} za $k > 2$,
- uslov 2: svi C_k' za $k > 2$,

mogu da stanu u memoriju istovremeno. **Prvo skeniranje:** određivanje L_1, a **drugo skeniranje:** određivanje L_k, $k > 1$.

Asocijacije između elemenata su obično uočljive tek na višim nivoima apstrakcije:

- "kupovina *Sport*-a $\Rightarrow$ kupovina *Camel* -a", pravilo sa vrlo malim faktorom podrške,
- "kupovina *sportskih novina* $\Rightarrow$ kupovina *jakih cigareta*", pravilo sa većim faktorom podrške,
- "kupovina *novina* $\Rightarrow$ kupovina *cigareta* ", pravilo sa velikim faktorom podrške.

Asocijacije između elemenata obično su uočljive tek na višim nivoima apstrakcije.



Asocijacije na višim nivoima apstrakcije su **jače**, ali **manje konkretne**.

Proširenje algoritma Apriori

Za DM na različitim nivoima apstrakcije:

- skupovi kandidata na nižim nivoima ispituju se samo ako su odgovarajući opšti skupovi jaki,
- na različitim nivoima apstrakcije se mogu usvojiti različiti pragovi faktora podrške.

13.2.7. Domensko znanje u procesu KDD

Pored sirovih podataka, u procesu otkrivanje znanja u bazama podataka (KDD), potrebne su i dodatne informacije. Informacije iz rječnika podataka ukazuju na međusobne veze i organizaciju podataka domenskog znanja, odnosno znanja o:

- sadržaju podataka u bazi podataka,
- domenu baze podataka,
- kontekstu i svrsi KDD procesa.

Domensko znanje fokusira pretraživanje. Nedostatak, koji se može javiti, je da fokusiranjem pretraživanja može da se ne uoči neko vrijedno znanje.

Logističko planiranje: korištenje ograničenja

- "kamioni ne voze preko vode" - kamioni ipak voze preko zaleđenih jezera.

Domensko znanje obično daje ekspert, ali se može i **otkriti** u procesu KDD.

Oblici domenskog znanja

- rječnik podataka je najjednostavniji oblik,
- korelacija polja (vodi ka semantici domena): visina i tjelesna težina su pozitivno korelisane,
- heuristička pravila,
- procedure i funkcije.

Načini korištenja domenskog znanja

- redukcija skupa podataka, nad kojim se vrši pretraživanje,
- optimizacija hipoteza, o znanju koje se otkriva,
- optimizacija upita, kojima se potvrđuju hipoteze,
- testiranje validnosti i ažurnosti znanja, otkrivenog u procesu KDD.

Redukcija skupa podataka

- traženo znanje: "lijek $X \Rightarrow$ efekti u trudnoći".
- redukcija: pravilo ne važi za muškarce, niti za žene mlađe od 12 ili starije od 65 godina.
- odgovarajući upit:

```
CREATE VIEW Reduced-Patient-File AS
SELECT *
FROM Patient-File
WHERE sex <> male AND age >= 12 AND age <= 65
```

Optimizacija hipoteza

- otklanjanje međuzavisnosti premisa hipoteze, predstavljene u obliku pravila,
- hipoteze mogu da sadrže redundantne premise,
- domensko znanje koje formuliše ekspert,
- omogućuje izbacivanje nekih premisa.

PRIMJER: *Hipoteza.*

```
IF starost >= 30                  AND
pol = ŽENSKI                      AND
trudnoća = DA                     AND
bolest = DIJABETES                AND
lijek = XXX                       AND
nivo šećera u krvi = POVIŠEN      AND
…
THEN efekat = POZITIVAN
```

PRIMJER: *Domensko znanje.*

trudnoća = DA ⇒ pol = ŽENSKI
bolest = DIJABETES ⇒ nivo šećera u krvi = POVIŠEN

Optimizacija upita

PRIMJER: *Hipoteza.*

```
IF starost >= 70                    AND
   fizičko stanje = SLABO           AND
   terapija za druge bolesti = DA   AND
   osiguranje = NE                  AND
THEN operacija = NE
```

PRIMJER: *Odgovarajući upit.*

```
SELECT starost, fizičko stanje, terapija za
       Druge bolesti, osiguranje
FROM Pacijenti, Operacije
WHERE operacija = NE
AND Pacijenti.P# = Operacije.P#
```

PRIMER:

- *domensko znanje: hemofilija = DA ⇒ operacija = NE*
- *optimizirani upit*:

```
SELECT starost, fizičko stanje, terapija za
       druge bolesti, osiguranje
FROM Pacijenti
WHERE hemofilija = DA
```

Testiranje validnosti i ažurnosti znanja

Otkriveno znanje može da sadrži:
- redundansu i kontradikcije. Kontradikcije su ozbiljniji problem;
- testiranje pomaže da se odredi:
  - da li je otkriveno znanje **zbilja** kontradiktorno ili samo **naizgled** kontradiktorno,
  - da li je domensko znanje neažurno (ako je otkriveno znanje dovoljno validno).

PRIMJER:

- *1970.-80.: 10% žena među žrtvama srčanog udara,*
- *1980.-95.: 40% žena među žrtvama srčanog udara,*
- *ta dva pravila ne moraju da budu u kontradikciji,*
- *domensko znanje: žene su u periodu 1980.-95. pod većim stresom, sklonije alkoholu, duvanu,….*

13.2.8. Evaluacija otkrivenog znanja

Mjera značaja pattern-a *(Measure of interestingness)*
- kvantitativna mjera za evaluaciju *pattern*-a,
- kombinacija sledećih osobina *pattern*-a:
  - validnosti (stepena izvjesnosti),
  - novine,
  - koristi,
  - jednostavnosti.

Izračunavanje mjere značaja pattern-a na osnovu statističkih pokazatelja
- procenat podataka koji odgovara *pattern*-u i njegova preciznost,
- obično najvažnije, a za neke *pattern*-e i dovoljno,

- ponekad se zahtjeva i neka mjera povjerenja, naročito u slučaju predikcije novih podataka: utvrđivanje profila novih kupaca na osnovu baze podataka o redovnim kupcima.

Izračunavanje mjere značaja pattern-a na osnovu domenskog znanja

- statističke mjere često nisu dovoljne
  - o "povećanje prodaje od 5% u regionu 1",
  - o "povećanje prodaje od 50% u regionu 2",
  - o prvo povećanje može da bude značajnije;
- često se domensko znanje kombinuje sa statističkim izračunavanjem.

Subjektivne mjere značaja pattern-a

- **objektivne mjere**
  - o vjerovatnoća, statistika, …

PRIMJERI: *Faktori povjerenja i podrške.*

- **subjektivne mjere** zavise i od korisnika, pa mogu da se razlikuju za razne korisnike:
  - o neočekivanost - *pattern* je interesantan ako predstavlja "iznenađenje" za korisnika,
  - o primenljivost, mogućnost "prevođenja" u akciju.

PRIMJER: *Primjenljivi pattern.*

(ocjene_ predavača < ocjene_kurseva)

Dekan može nešto da **uradi** na osnovu tog *pattern*-a, tj. da ga korisno **primjeni** *(prevede u akciju)*.

PRIMER: *Neočekivani pattern*.

(ocene_ predavača, visoke) $\wedge$ (ocene_ predavača_X, niske).

Neočekivani *pattern*-i su u **kontradikciji** sa uvjerenjima.

- primenljivost *pattern*-a je teško formalizovati,
- neočekivanost *pattern*-a se može formalizovati pomoću formalnog opisa sistema uvjerenja;
- neočekivani pattern-i su u **kontradikciji** sa uvjerenjima:
  - pretpostavka: primenljivi *pattern*-i su većinom neočekivani (i obrat-no),
  - zato se primenljivost *pattern*-a opisuje posredno, preko neočekivano-sti.

Formalizacija sistema uvjerenja

- sistem uvjerenja - skup logičkih iskaza,
- neizvesnost uverenja - verovatnoća, faktori izvesnosti,…

$d(b \mid E)$ - jačina (stepen) uvjerenja b na bazi neke prethodne evidencije E
$d(b \mid E)$ se često predstavlja preko uslovne vjerovatnoće (*Bayes*-ova teore-ma).

- čvrsta uvjerenja - nova evidencija ih ne mijenja, čak i ako je kontradiktor-na uvjerenjima,
- promjenljiva uvjerenja - nova evidencija mijenja $d(b \mid E)$,
- *pattern* koji je u kontradikciji sa čvrstim uvjerenjima je neočekivan i **uvijek** interesantan (značajan),
- što je veći uticaj *pattern*-a na promjenljiva uvjerenja, to je *pattern* značaj-niji za korisnika,

Formalna mjera neočekivanosti (značaja)

- različita uvjerenja (iskazi) su različitog značaja,
- promjene nekih uvjerenja su značajnije nego promjene drugih,
- αi - uvjerenje u sistemu uvjerenja B
- wi - značaj (težina) uverenja αi,
- konvencija: $\sum_{\alpha_i \in B} w_i = 1$

- značaj *pattern*-a p u odnosu na sistem uverenja B:

$$I(p,B,E) = \sum_{\alpha_i \in B} w_i \, | \, d(\alpha_i \, | \, p,E) - d(\alpha_i \, | \, E) \, |$$

- $I(p, B, E)$ pokazuje koliko se B mjenja pod dejstvom *pattern*-a p,
- važan specijalan slučaj, kada su sva uvjerenja jednako značajna, $w_i = 1/N$,
- $I(p, B, E)$ se ne mijenja ako se neko uverenje α_i zameni svojim komplementom, $\neg \alpha_i$,
- $(0.5 < d (\alpha_i \, | \, E) < 1) \wedge$; "dovoljno" uvjerenje o α_i,
- $(0.5 < d (\alpha_i \, | \, \neg p, E)) \wedge$; kontradiktorni *pattern*,
- $(d (\alpha_i \, | \, p, E) < 1) \Rightarrow I (p, B, E) \leq I (\neg p, B, E)$,
- tumačenje gornjeg izraza: neočekivani *pattern* je značajniji od očekiva-nog,
- dodavanjem novih podataka u bazu može da se promeni stepen nekih uvjerenja o podacima,
- mogućnost traganja za interesantnim *pattern*-ima: ispitivanje promjena uvjerenja zbog novih podataka,
- za skup starih podataka D u bazi i skup novih ΔD, iz izraza za $I(p, B, E)$ slijedi:

$$(d (\alpha_i \, | \, \Delta D, D) \neq d (\alpha_i \, | \, D)) \Rightarrow \exists \, p, I (p, B, E) \neq 0$$

DM proces vođen uvjerenjima (*belief-driven DM*)

- prethodna formula sugeriše šemu; dolaskom novih podataka revidiraju se $d (\alpha_i \, | \, D)$,
- ako je promjena nekog $d (\alpha_i \, | \, D)$ iznad nekog praga, onda postoje neki interesantni *pattern*-i,
- DM proces se u tom slučaju pokreće.

13.2.9. Sistemi i aplikacije

Važne oblasti primjene KDD

- **medicina** - sporedni efekti ljekova, analiza genetskih sekvenci, ... ,

- **finansije -** odobravanje kredita, predikcije na tržištu dionica, … ,
- **društvene nauke -** demografski podaci, prognoza / analiza izbora, … ,
- **astronomija -**analiza satelitskih podataka,
- **pravne nauke i kriminalistika -** utaja poreza, identifikacija ukradenih vozila,
- **marketing -** predikcija prodaje, identifikacija grupa potrošača, identifika-cija grupa kupljenih proizvoda, … ,
- **inženjerske discipline -** analiza CAD baza podataka, ponude poslova,…,
- **osiguranje -** detekcija visokih potraživanja, ulančana potraživanja,
- **poljoprivreda -** klasifikacija oboljenja biljaka,
- **izdavaštvo -** specijalna izdanja časopisa za određeni profil čitalaca.

Kriterijumi za izbor KDD aplikacije

- postojanje objektivne potrebe za KDD,
- postojanje potencijalnih finansijskih efekata,
- nemogućnost otkrivanja *pattern*-a korištenjem samo statističkih metoda,
- dovoljan broj pouzdanih primjera (najmanje 1000),
- organizaciona podrška,
- domensko znanje (KDD će ga upotpuniti!).

Izgledi za uspjeh KDD aplikacije

- najveći su u situacijama kada "neko znanje već postoji, ali nije kompletno",
- količina neizvesnih podataka i šuma treba da bude minimalna.

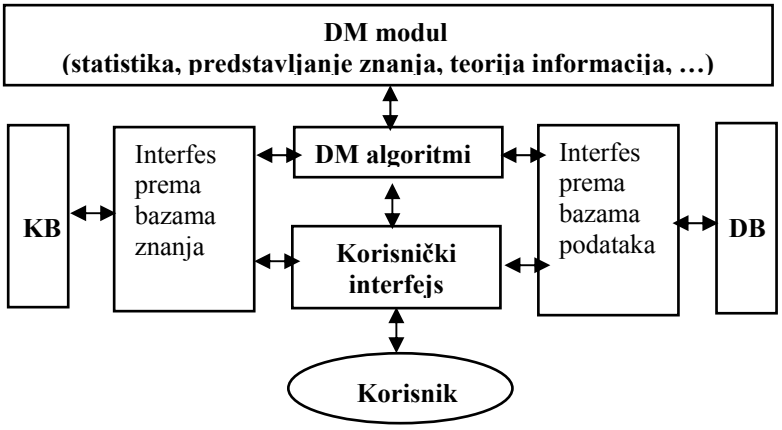
13.2.10. Softverska okruženja za KDD

Mining Kernel System - MKS (University of Ulster, Northern Ireland)

Okruženje za razvoj KDD & DM aplikacija:

- skup biblioteka sa različitim alatima;
- biblioteke raspodjeljene u tri osnovna modula:
  - interfejs (pristup podacima i znanju),
  - o

- o DM modul (statistika, teorija informacija, …),
- o jezgro (DM algoritmi).



Slika 13.8. Okruženje za razvoj KDD i DM aplikacija.

Korisnički interfejs

- opisivanje podataka nad kojima se vrši KDD:
  - o opisi se formiraju preko ekranskih oblika,
  - o početni opisi prevode se u oblik sličan SQL upitima i smještaju se u posebne tekstualne datoteke,
  - o gotovi opisi iz datoteka prosljeđuju se interfejsu prema bazi podataka;
- postavljanje pragova za mjere značaja *pattern*-a.

Interfejs prema bazama podataka

- prevođenje podataka nad kojima se vrši KDD:
  - o početni opisi se dobijaju od korisničkog interfejsa u obliku DSM datoteka (*data-source mapping*),
  - o prevedeni opisi se prosleđuju SUBP-u;
- ulaz / izlaz znanja, i početnog i otkrivenog.

PRIMER: *DSM datoteke.*

```
# MKS Data Source File          ; komentar
TYPE Ingres                     ; format podataka
SOURCE personnel                ; ime baze podataka
MAPPING                         ; kao SELECT…AS… u
                                    SQL-u
person.surname < surname >
person.sex     < sex >
…
WHERE CLAUSE                    ; kao WHERE u SQL-u
person.surname like 'N%'
and person.ni_num = account.ni_num
END
```

DM modul

- biblioteka za statistiku:
  - srednja vrijednost, standardna devijacija, histogram, medijana, kore-lacioni koeficijenti, χ^2, sampling, … ,
  - Euklidske mjere rastojanja,
  - procedure za formiranje klastera,
  - izračunavanje relativnih frekvencija,
  - određivanje diskriminišućih atributa,
  - vizuelno prikazivanje podataka (jednostavne slike);
- biblioteka teorije informacija:
  - entropija,
  - informaciona dobit,
  - informacioni količnik,
  - mjere značaja *pattern*-a,
  - … ;
- predstavljanje znanja - pravila, CF, … ,
- biblioteka teorije verovatnoće:
  - uslovne verovatnoće, *Bayes*-ovski modeli;

- biblioteka za rad sa skupovima:
  - primjena u algoritmima Apriori, DHP, CDP, … ;
- biblioteka teorije evidencije:
  - proširenje *Bayes*-ove teorije vjerovatnoće,
  - predstavljanje parcijalnog znanja,
  - predstavljanje **neznanja** - podataka koji nedostaju.

13.3. HIBRIDNI INFORMACIONI SISTEMI

13.3.1. Opšte o hibridnim inteligentnim sistemima

Većina tradicionalnih informacionih sistema, koji se baziraju na znanju, razvijaju se kao samostalni sistemi, sa minimalnom međusobnom povezanosti. Narastanje količine informacija, zahtjeva razvoj kompleksnijih sistema, koji integrišu znanje i tradicionalno procesiranje. Jedna generacija inteligentnih IS, koji prevazilaze navedene probleme, se razvija uz pomoć hibridne metodologi-je. Svaka inteligentna tehnika ima svoje pogodnosti (npr. sposobnost učenja, analizu odluka, itd.), koje ih čine pogodnim za dotično rješavanje problema ili kao pomoć za druge tehnike.

PRIMJER: *Neuronske mreže su pogodne za prepoznavanje uzoraka, ali nisu pogodne za analizu kako se došlo do rješenja. Fazi logički sistemi su dobri za analizu njihovih rješenja, mada ne mogu automatski iskazati pravila koja su upotrebljena pri dobijanju tih rješenja.*

Navedena ograničenja su bili osnovni razlog za kreiranje hibridnih inteligentnih sistema, kombinujući dvije ili više tehnika u namjeri da prevaziđu ograničenja pojedinačnih tehnika. Prema tome, hibridni sistemi su pogodni za rješavanje različitih aplikacionih domena.

Mnogi kompleksni domeni imaju različite komponente problema, koji zahtjevaju različitu vrstu procesiranja. Iz tog razloga upotreba hibridnih

inteligentnih sistema brzo se razvija u mnogim domenima, uspješno rješavajući aplikacije, kao što su:

- procesna kontrola,
- industrijsko projektovanje,
- marketing,
- medicina,
- razne vrste simulacija, i dr.

U hibridnim inteligentnim sistemima se koriste različite tehnologije:

- fazi logika,
- neuronske mreže,
- genetički algoritmi,
- stabla odlučivanja, itd.

13.3.2. Neuro - fazi sistemi

Fazi logika se upotrebljava u mnogim područjima, a pristupi se temelje na konvencionalnim metodama. Učinkovitost ovih sistema nije adekvatna utrošenim sredstvima. Povećanjem kompleksnosti sistema otežano je definisanje fazi pravila i osnovnih funkcija, koje se upotrebljavaju za opisivanje ponašanja sistema. Prednost fazi logike se uočava u ekonomskom i finansijskom modeliranju, gdje se pravila upotrebljavaju bez detaljnog i eksplicitnog znanja o izvršavanom procesu. Kod neuronskih mreža, ograničeni ili nestruktuirani podaci lako dovode do nekonzistentnih izlaza, što prouzrokuje velike probleme.

Imajući u vidu komplementarnost ovih dvaju tehnologija, moguće ih je integrisati na različite načine. Time se umanjuju njihovi pojedinačni nedostaci. Budući da se fazi sistemi, uglavnom, upotrebljavaju u industrijskim aplikacijama, očito je da razvoj dobrih fazi sistema nije ni malo jednostavan. Obuka neuronskih mreža je glavni razlog zašto se one upotrebljavaju u kombinaciji sa fazi sistemima. Cilj je automatizacija ili podrška procesa razvoja

fazi sistema za određenu namjenu. Prvi razvijani neuro - fazi sistemi su bili u domenu (neuro-) fazi kontrole, dok je kasnije pristup razvoju znatno proširen.

Neuro-fazi sistemi se upotrebljavaju u različitim domenima:
- industrijskoj kontroli,
- analizi podataka,
- podršci odlučivanju, itd.

13.3.3. Neuro - fazi hibridni sistemi

Neuro-fazi hibridni sistemi kombinuju prednosti fazi sistema, koja se očituje u eksplicitnom znanju koje je prihvatljivo i razumljivo za neuronske mreže, i koje se kombinuje sa implicitnim znanjem neuronskih mreža, dobijenim obučavanjem mreža.

Obuka neuronskih mreža predstavlja dobar način za pripremu ekspertnog znanja i dodatno generisanje dodatnih fazi pravila u osnovne funkcije, u cilju smanjenja vremena njihovog kreiranja. Sa druge strane, fazi logika povećava mogućnosti generisanja sistema sa neuronskim mrežama, dobijanjem boljih izlaza za njihovu eksploataciju.

Neuro - fazi arhitektura

Neuro-fazi sistem posjeduje različite komponente tradicionalnih fazi sistema, obezbjeđujući da svaka faza izdvaja sloj sakrivenih neurona u neuronskoj mreži, omogućujući uvećanje sistemskog znanja.

Arhitektura neuro - fazi sistema se sastoji, slika 13.9.:
- fazifikacionog sloja,
- sloja fazi pravila,
- defazifikacionog sloja.

Slika 13.9. Šema neuro - fazi arhitekture.

Fazifikacioni sloj

Svaki neuron u fazifikacionom sloju predstavlja ulaznu osnovnu funkciju za preostala fazi pravila. Jedna od metoda za implementaciju ovog nivoa je prikazivanje osnovnih funkcija kao diskretnih tačaka. Nakon toga se aktivira fazi pravilo:

IF X_1 is A! in X_2 is A_2 .. THEN Y is B,

koje je, možda, određeno distribucijom prethodnog iskaza "X is A".

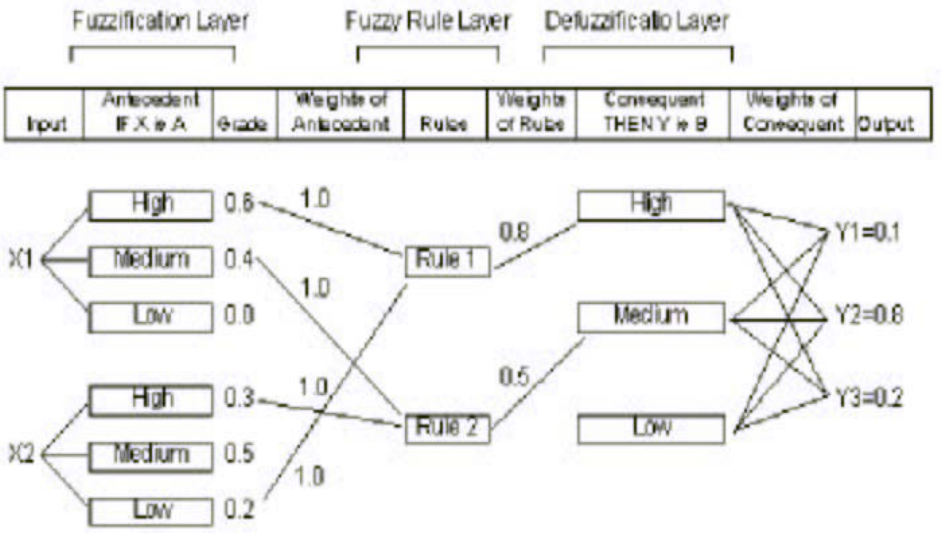
Svaki sakriveni "*node*" (čvor - komponenta u računarskoj mreži) je definisan kao fazi referentna tačka u ulaznom prostoru. Ova metoda lako aproksimira mnogo narednih funkcija i stepen greške je ovisan od broja upotrebljenih diskretnih tačaka.

Drugi, mnogo bolji pristup, je kombinovanje jedne ili dvije "*sigmodial*" funkcije sa linearnim funkcijama za predstavljanje osnovnih funkcija u fazifikacionom i defazifikacionom sloju. Parametri tih neurona su lagano trenirani za besprijekorno oblikovanje konačnog oblika u lokacijama osnovnih funkcija. U većini slučajeva, na tom nivou, broj neurona je stalan, mada je između treninga moguće dodati ili odstraniti te neurone u odnosu na izlaze, koji su stvoreni na primjerima za treniranje.

Na slici 13.10. je prikazan osnovni nivo, koji pokazuje stanje: "X_1 *is High*" je 0,6; "X_1 *is Medium*" je 0,4 i "X_1 *is Low*" je 0,0. Izlaz ovih osnovnih funkcija je povezan sa slojem fazi pravila, specificiran sa fazi pravilima, upotrebljavajući veze stalne jednakosti. Na slici 13.10. svaki mali pravougaonik predstavlja neuron.

Sloj fazi pravila

Ovaj sloj predstavlja osnovu fazi pravila i njegova funkcija će izvršavati fazi logičke operacije. Svaki neuron predstavlja fazi pravilo ″IF X_1 is A_1 and X_2 is A_2 ... THEN Y is B" i određuje spremnost svakoga pripremljenog prijedloga "IF X_1 is A_1 and X_2 is A_2 ... ". Također, pokazuje ispravnost pripreme, i kako su preduslovi svakoga fazi pravila pripremljena.



Slika 13.10. Implementacija fazi pravila.

Neuroni imaju linearne funkcije i njihovi izlazi su povezani sa defazifikacionim slojem sa uravnoteženim vezama. Uravnoteženost tih veza predstavlja relativnu povezanost pravila ostvarenih sa neuronima. Njihove vrijednosti se lako produžavaju u odnosu na eksperta, ali inicijalizirane na 1,0, jer su tako trenirane da pokazuju njihovu aktuelnu povezanost u odnosu na izlaz osnovne funkcije koja se nalazi u defazifikacionom sloju.

Defazifikacioni sloj

Funkcija ovoga sloja je da ocjenjuje pravila. Svaki neuron u ovom sloju predstavlja rezultat prijedloga "THEN Y is B" i njegova osnovna funkcija je lagano implementirana u kombinaciji jedne ili više "sigmoidal" funkcija iz linearnih funkcija. Spremnost svakog posljedičnog prijedloga je računata ili je uočena kao odlična prilagođenost razmatranih fazi pravila, koji imaju jednake posljedične prijedloge.

Sadržaj svake izlazne veze tih neurona predstavlja centralno težište svake izlazne dobijene osnovne funkcije i daju se trenirati. Konačna izlazna vrijednost je izračunata upotrebom metode centralnog težišta.

Treniranje neuro - fazi sistema

Struktura sa slike 13.10. se lako konfiguriše sa inicijalnim vrijednostima specificiranim od strane eksperta, a zatim oblikuje upotrebom algoritma za treniranje "*backpropagation*", slijedećim redoslijedom:

Korak 1: Prisutni primjeri ulaznih podataka izračunavaju vrijednost izla-za;
Korak 2: Izračunavanje greške između izlaza sa aktuelnim ciljem;
Korak 3: Povezane vrijednosti i osnovne funkcije su prilagođene;
Korak 4: Brišu se neupotrebljiva pravila sa istim brojem u osnovnoj funk-ciji "*node*" i dodaju nova;
Korak 5: IF greška > tolerancije THEN goto Korak 1 ELSE stop.

Ako nova greška padne u interval navedene tolerancije, konačne međusobno povezane vrijednosti izvršavaju pohranjivanje pravila u početna fazi pravila i osnovne funkcije. Ukoliko je rezultujuća vrijednost blizu 0, pravilo se odstranjuje iz baze pravila. Oblik i pozicija osnovnih funkcija u fazifikacionom i defazifikacionom sloju se lako ostvaruje, uz prilagođavanje parametara naurona u navedenim slojevima postupkom procesa treninga.

13.4. INTELIGENTNE TELEKOMUNIKACIONE MREŽE

13.4.1. Opšte o inteligentnim mrežama

Telekomunikaciono tržište bilježi brz rast, bez presedana, u posljednjih nekoliko godina. Kupci traže pristup servisima dostupnih brzina sa minimalnim kašnjenjem. Mrežni provajderi moraju biti u stanju, i spremni, da uvedu nove proizvode i servise, ne samo da bi osvojili udio tržišta već i da bi ga zadržali.

Izraz inteligentne mreže se odnosi na telekomunikacione mreže izgrađene na računarski baziranim platformama. Glavni cilj inteligentnih mreža je da pruže široki spektar generičkih servisa, dok kreiraju prilagođene nove servise bazirane na specifičnim zahtjevima u relativno kratkom vremenskom periodu. *IN* koncept (*Intelligent Networks* koncept) je baziran na arhitekturi, gdje su određujuće servisne funkcije distribuirane preko specijaliziranih mrežnih čvorova.

Koncentrisanje najvažnijih funkcija u kontrolnim čvorovima dozvoljava softversku nadogradnju bez uticaja na preostale mrežne čvorove. Kao rezultat novi servisi su omogućeni bez prekida u sistemu.

13.4.2. Arhitektura inteligentnih mreža

Inteligentne telekomunikacione mreže omogućavaju fleksibilno i brzo uvođe-nje novih servisa. Posjeduju centre inteligencije, u kojima je izvršena centralizacija upravljačkih funkcija i u okviru kojih se vrši posluživanje pojedinih poziva.

Osnovni elementi inteligentne mreže su:

- komutacioni čvor servisa SSP (*Service Switching Point*),
- širokopojasni čvor za upravljanje servisom BSCP (*Broadband Service Control Point*),
- sistem za upravljanje servisom SMS (*Service Management System*), i

- inteligentni periferali IP (*Intelligent Peripheral*).

SSP predstavlja odgovarajući komutacioni sistem u mreži. Obezbjeđuje pristup u inteligentnu mrežu, tako što detektuje zahtjev za određenim servisom i proslijeđuje ga prema BSCP, koji po prijemu zahtjeva daje instrukcije SSP-u koje se odnose na obradu poziva. BSCP obezbjeđuje prevodilačke i upravljačke funkcije:

- adresiranje servera,
- uspostavljanje veze, uključujući selekciju propusnog opsega i ostalih parametara servisa.

SSP i BSCP među sobom komuniciraju posredstvom mreže za signalizaciju po odgovarajućem zajedničkom kanalu. SMS je zadužen za administraciju pojedinih servisa i sadrži interfejse za komunikaciju sa operatorima i korisnicima servisa. Na taj način omogućena je modifikacija servisa i promjena njegovih parametara. Širokopojasni inteligentni periferali omogućavaju dobijanje informacija u toku obavljanja određenog servisa i zajedno sa *set-top box* uređajima realizuju odgovarajući grafički interfejs prema korisniku i izbor provajdera servisa.

Arhitektura inteligentnih mreža organizovana je na multiservisnom principu, šta znači da jedan BSCP može da opsluži više servisa i obratno, jedan servis može biti distribuiran u okviru više BSCP u zavisnosti od saobraćajnih zahtjeva. Korisnici imaju mogućnost da posredstvom SMS-a definišu individualne karakteristike sopstvenih servisa, čime se još više povećava komfor koji im pruža inteligentna mreža.

Kompleksna mrežna struktura, sa mnoštvom elemenata koji su u međusobnoj korelaciji, zahtjeva upravljanje mrežom “sa kraja na kraj”, uključujući sve elemente mreže, od *set-top box* uređaja do servera sa informacijama multimedijal-nog tipa, pri čemu se mora obezbjediti podrška različitim operacionim scenarijima, u zavisnosti od konstalacije obuhvaćenih operatora mreža i provajdera servisa. Navedeni zahtjevi se mogu realizovati integracijom koncepata inteligentne mreže sa TMN sistemom za upravljanje širokopojasnom mrežom.

Transportna SDH mreža se sastoji od:

- optičkih medijuma prenosa, i
- SDH uređaja (SDH terminalni multiplekseri, *add-dropp* multiplekseri, *cross-connect* uređaji i SDH linijski uređaji).

SDH multipleksna struktura omogućava neposredno multipleksiranje protočnih signala svih vrsta servisa u STM-N signale protoka 155 Mb/s, 622 Mb/s i 2,5 Gb/s, njihovo jednostavno grananje i rutiranje kroz mrežu. Puna fleksibilnost SDH mreže je obezbjeđena uvođenjem globalnog upravljanja mrežom (TMN).

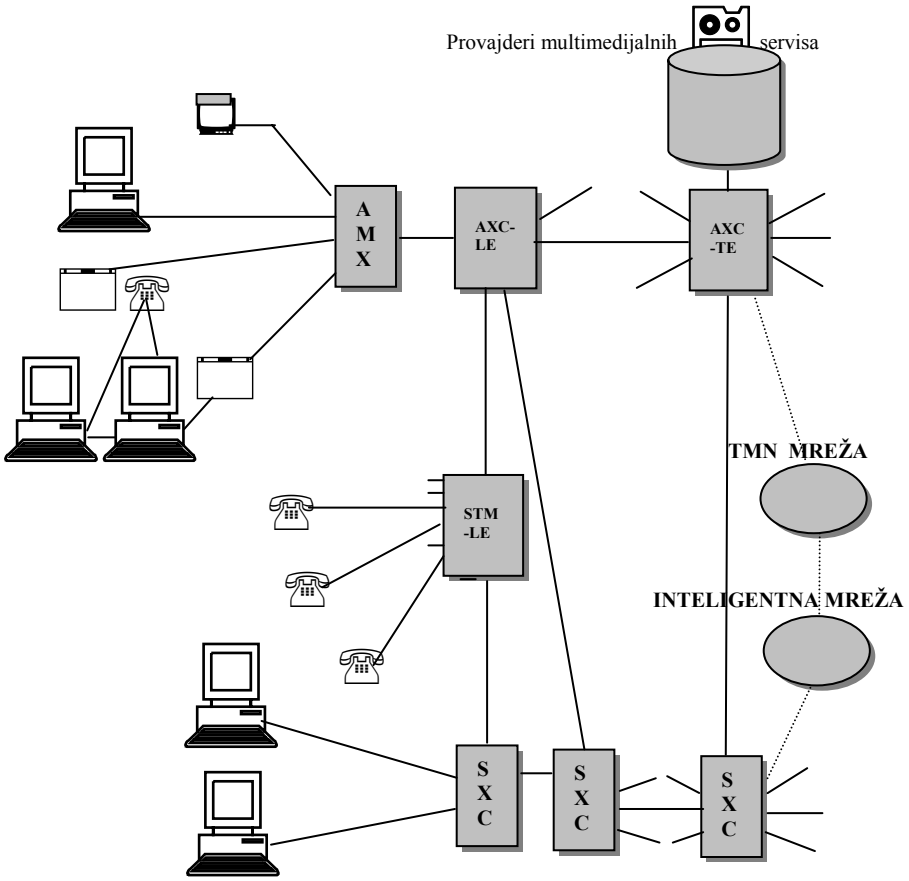
ATM predstavlja novu tehniku prenosa, komutacije i multipleksiranja, kod koje se informacije prenose u kratkim paketima, tzv ćelijama, stalne dužine od 48 bajta, plus 5 bajta za zaglavlje. Rutiranje ćelija se zasniva na principu kanala sa dvojnom identifikacijom. Zbog jednostavnog protokola koji se koristi, transfer ćelija između čvorova u mreži se može obavljati samo hardverski, što za posljedicu ima kratko vrijeme prenosa i visok stepen iskorištenja transmisionih puteva, čak i za protokole od nekoliko stotina Mb/s. Sa druge strane, ATM je zadržao fleksibilnost paketskog načina prenosa informacija i jednostavni metod multipleksiranja, koji ne zavisi od veličine i varijacije protoka bita.

Za razliku od sinhrone tehnike vremenskog multipleksiranja ATM posjeduje značajne prednosti, koje se prije svega odnose na iskorištavanje prenosnog puta pri realizaciji određenog servisa ili grupe servisa, u skladu sa egzaktnim zahtjevima i aktivnošću izvora. Ovo je posebno važno u slučaju da se radi o saobraćaju promjenljivog protoka, što predstavlja osnovnu karakteristiku multimedijalnih servisa. Na slici 13.11 je prikazana ATM mreža i njena veza sa SDH mrežom.

ATM multiservisni multiplekseri AMX (*ATM Multiplexer*) vrše konverziju digitalnih signala u ATM format, multipleksiranje virtuelnih putanja i virtuelnih kanala, kao i rutiranje tranzitnog ATM saobraćaja, a mogu da rade kao komutatori ili *cross-connect* uređaji. Na svojim ulaznim konektorima, ovi uređaji posjeduju različite vrste interfejsa koji realizuju protokole, kao što su:

- nx64 kb/s sinhroni protokol za priključenje kućnih centrala,

- kodeka za video konferencije ili *voice-data* multipleksere,
- X.25 i *Frame-Relay* protokole,
- *Ethernet, Token Ring* i dr.



Slika 13.11. ATM koncept multimedijalne mreže.

Navedeni multiplekseri posjeduju i funkcije rerutiranja, zaštite i sofistirane funkcije upravljanja za administriranje servisa. ATM *cross-connect* uređaji, ATM komutatori i AXC (*Asynchronous Cross-Connect*) uređaji omogućavaju funkcije prespajanja i komutacije, na osnovu podataka funkcija upravljanja i signalnih poruka na lokalnom, regionalnom i magistralnom nivou mreže. Na njih su priključeni digitalni serveri multimedijalnih informacija, kao i SDH *cross-connect* uređaji. AXC uređaji mogu imati funkcije lokalnih komutacionih čvorova AXC-LE (*Asynchronous Cross-Connect Local Exchange*) ili tranzitnih centrala AXC-TE (*Asynchronous Cross-Connect Trunc Exchange)*, što zavisi od potreba i kapaciteta širokopojasne mreže.

Sve veze u ATM dijelu mreže se realizuju kao virtuelni kanali ili virtuelne putanje, a za transport se koristi SDH transmisiona infrastruktura. Sinhroni *cross-connect* uređaji SXC, u zavisnosti od kapaciteta, vrše prespajanje SDH digitalnih signala u informacione strukture, tzv. virtuelne kontejnere VC (*Virtual-Container*). Prihvatanje ATM signala je standardizovano kroz postupke mapiranja ATM ćelija u virtuelne kontejnere različitog formata.

Koncepcija arhitekture multimedijalne širokopojasne telekomunikacione mreže treba da obezbjedi fleksibilnu i modularnu strukturu mreže u skladu sa povećanjem potreba korisnika za novim i širokopojasnim servisima, uz optimalnu i ekomičnu eksploataciju od strane operatora mreže i provajdera servisa.

Prikazani primjer Inteligentne telekomunikacione mreže zasnovan je na radovima Drobnjak, R. i dr., "Predlog arhitekture multimedijalne telekomunikacione mreže", 1996. i Edgar, A.,: "A Generic Service Access Network Platform", 1995.

U ovom poglavlju:

- ***Sticanje znanja za ES***
- ***Metodologija razvoja ES***
- ***Sredstva za izgradnju ES***
- ***Razvoj ekspertnog sistema***
- ***Primjer ekspertnog sistema***

14

IZGRADNJA EKSPERTNIH SISTEMA

14.1. STICANJE ZNANJA ZA EKSPERTNE SISTEME

14.1.1. Opšte o sticanju znanja

Ekspertni sistemi (ES) se projektuju i izgrađuju po nekoj od važećih i u praksi korištenih metodologija. Moguća metodologija razvoja ES sadrži slijedeće korake.

(1) **Analiza**
- identifikacija potencijalne aplikacije,
- ocjenjivanje podesnosti inženjeringa znanja za datu aplikaciju;

(2) **Specifikacija**
- upoznavanje šta će ekspertni sistem raditi,
- rad sa ekspertom, učenje o zadatku u cilju planiranja razvoja sistema;

(3) **Razvoj**
- konceptualni dizajn: spoznaja kako ekspert izvršava zadatak, razvoj konceptualnog modela,
- implementacioni dizajn: spoznaja kako formalizmi zaključivanja, prezen-tacije i kontrole mogu biti upotrebljeni u implementaciji konceptualnog modela,
- implementacija: slijediti implementacioni dizajn u izgradnji baze znanja,
- evaluacija: test sistema u cilju verifikacije da li sistem izvršava zadatke korektno;

(4) **Razvoj u primjeni**
- primjena: instalisanje sistema za rutinsku upotrebu,
- održavanje: pronalaženje grešaka, ažuriranje i poboljšanje vrijednosti sistema.

14.1.2. Ispitivanje eksperta

Postoje dva osnovna pristupa u crpljenju znanja od eksperta:
- ispitivanje eksperta poznatim metodama i tehnikama,
- posmatranje eksperta na djelu.

Crpljenje znanja ispitivanjem eksperta uvijek ima oblik interakcije čovjeka eksperta i inženjera znanja. Koriste se slijedeći oblici interakcije:
- deskripcija,
- introspekcija.

Deskripcijom se predstavljaju "idealni slučajevi", kao što se to čini u udžbenicima, dok informacije o strategijama rješavanja specifičnih problema ostaju neobuhvaćene.

Introspekcija je ispitivanje eksperta u cilju razrješavanja sopstvenih nedoumica iz posmatranog domena.

Metode i tehnike ispitivanja eksperta su slijedeće:
- intervju,
- repertoarske rešetke,
- skale procjenjivanja,
- tehnika kritičnih događaja,
- tehnika uparivanja karakteristika i odluka,
- razlikovanje ciljeva,
- reklasifikacija,
- analiza odlučivanja.

Intervju

Lični razgovor je najprirodniji i najčešći, a ujedno i najefikasniji oblik kontaktiranja među ljudima.

Za primjenu intervjua moraju biti ispunjeni slijedeći uslovi:

- lični kontakt intervjuiste i intervjuisanog,
- plansko i svrsishodno vođenje intervjua,
- usmjerenost intervjua,
- iskrena saradnja između intervjuiste i intervjuisanog,
- neometanost razgovora.

Repertoarske rešetke

Pri korištenju repertoarskih rešetki polazi se od toga da svaka osoba ima svoj lični teorijski obrazac svijeta, da predviđa i nadzire procese i događaje, izgrađujući teorije, provjeravajući svoje hipoteze i vrednujući iskustveno svjedočanstvo. Repertoarske rešetke su sačinjene od elemenata i konstrukata. **Elementi** su ključni primjeri koje daje ekspert, a **konstrukti** su bipolarne karakteristike koje svaki element ima u većoj ili manjoj mjeri.

Skale procjenjivanja

Elementi se ocjenjuju, opisuju i upoređuju s obzirom na stepen u kojem posjeduju svaki konstrukt. U tom procjenjivanju se često upotrebljavaju skale procjenjivanja.

Tehnike kritičnih događaja

Suština ove metode je da ekspert potanko opisuje teške, značajne i zanimljive slučajeve iz svog iskustva, svoje ponašanje, doživljaje i osjećanja. Naročitu pažnju u evociranim mislima i ponašanju eksperta zavređuje ono što je presudno vodilo do uspješnog ili neuspješnog rješavanja danog problema. Tehnika kritičnih događaja je podesna za sticanje znanja u obliku činjenica i heuristika.

Tehnika uparivanja karakteristika i odluka

Ova tehnika od eksperta traži da navede skup mogućih značajnih karakteristika problema i skup mogućih odluka, a potom da upari podskupove karakteristika sa odgovarajućim odlukama.

Razlikovanje ciljeva

Kada se za crpljenje znanja od eksperta primjenjuje metoda razlikovanja ciljeva, od eksperta se zahtjeva da za neki identifikovani cilj navede skupove nužnih i dovoljnih razloga za razlikovanje tog cilja od ostalih.

Reklasifikacija

U reklasifikaciji je smjer obrnut, polazi se od ciljeva unatrag ka činjenicama, karakteristikama, simptomima. Zadatak je eksperta da svaki pojedinačan cilj rasčlani na podciljeve i reklasifikuje na razloge, svjedočanstvo za te ciljeve.

Analiza odlučivanja

Analiza odlučivanja se sprovodi kada se žele dokučiti znanja potrebna za uspješno odlučivanje i procedure odlučivanja u danoj problemskoj situaciji. Kao sredstvo za sprovođenje ove tehnike se koriste pomoćna sredstva, kao što su tabele, stabla odlučivanja i slično.

14.1.3. Posmatranje eksperta na djelu

Posmatranje eksperta na djelu, odnosno tehnika opservacije, omogućava inženjeru znanja da zapazi mnoge bitne detalje koji deskripcijom ostaju neuočeni, ali sa druge strane opservacija nije moguća ukoliko inženjer znanja nije upoznat sa osnovnim domenskim znanjem. Zbog toga se posmatranje eksperta na djelu kombinuje sa ispitivanjem eksperta i/ili proučavanjem pisanih izvora.

14.2. METODOLOGIJA RAZVOJA EKSPERTNOG SISTEMA

Razvoj ekspertnog sistema (ES) se može podjeliti u šest faza:

- preliminarna analiza,
- analiza i dizajn ekspertnog sistema,
- razvoj prototipa ekspertnog sistema,
- razvoj ekspertnog sistema,
- testiranje i implementacija,
- održavanje ES.

14.2.1. Preliminarna analiza

Preliminarna analiza sadrži slijedeće radnje:

- definisanje potencijalnih projekata,
- dobijanje saglasnosti rukovodstva organizacije za potencijalne projekte,
- oformljenje projektantskog tima za razvoj ekspertnog sistema.

Projektantski tim, za svaki projekat ekspertnog sistema, mora obavezno sadržavati slijedeće grupe ljudi:

- rukovodilac projekta,
- glavni projektant (inženjer znanja),
- projektanti (inženjeri znanja),
- eksperti,
- korisnici.

Zadaci projektantskog tima u ovoj fazi razvoja su:

- identifikacija potencijalnih projekata,
- analiza i ocjena svakog potencijalnog projekta,
- formiranje i organizovanje tima projekta,
- afirmacija inženjeringa znanja i projekta kod rukovodstva organizacije, eksperata i korisnika,
- izbor projekta za razvoj.

Svaka grupa ljudi, koja sačinjava projektantski tim, ima tačno definisane zadatke, koji na ovom mjestu neće biti opisivani.

14.2.2. Analiza i dizajn

U drugoj fazi razvoja ekspertnog sistema rukovodilac projekta i projektantski tim moraju izabrani projekat za razvoj svestrano analizirati i procjeniti sa različitih aspekata, ističući sve korisne detalje.

Rukovodilac projekta je direktno angažovan u fazi analize i njegovi glavni zadaci su planerske i organizacione prirode:

- izrada detaljnog plana analize,
- definisanje domena i njegova dublja analiza,
- razvoj detaljnog plana projekta,

- organizovanje projektantskog tima,
- organizovanje više uzastopnih sastanaka sa ekspertom i korisnicima, sa posebnim naglaskom na interakciji ekspert - korisnik.

Rukovodilac projekta mora odgovarajuću pažnju posvetiti problemu softvera i hardvera, koji su potrebni za razvoj sistema. Komunikacija između članova projektantskog tima je važna i u mnogome presudna za dalje faze rada.

Zadaci projektantskog tima u ovoj fazi razvoja ekspertnog sistema su:

- doprinos preciznom definisanju domena,
- upoznavanje sa konkretnim zadacima i analiza tih zadataka,
- intervjuisanje eksperata,
- pronalaženje i sakupljanje knjiga, članaka i drugih pisanih materijala, kako bi se što bolje spoznali kognitivni aspekti zadatka,
- analiza znanja i kognitivnih procesa koje eksperti koriste u rješavanju problema,
- definisanje, zajedno sa ekspertima i korisnicima, kriterijuma za ocjenu uspješnosti ekspertnog sistema.

Kognitivna analiza je sredstvo inženjeringa znanja i pored analize zadatka je primarni posao projektantskog tima u fazi analize i dizajna. Kognitivna analiza je proces sa kojim se produbljuju saznanja šta eksperti znaju i kako razmišljaju o specifičnim zadacima. To nije savršen proces i češće je više vještina nego inženjering.

Postoje empirijske metode koje se mogu upotrijebiti za "prečišćavanje" znanja, koje se dobijaju za vrijeme kognitivne analize. Može se kodirati znanje koje je predviđeno za bazu znanja i razmatrati, zajedno sa ekspertom, slučajeve koje ekspertni sistem pokušava da riješi.

Eksperti, u ovoj fazi razvoja ES, se moraju upoznati sa njihovom ulogom i zadacima, a nakon toga, analizirajući studije slučajeva i čitajući literaturu iz područja inženjeringa znanja, steći kritinu masu znanja o ES, domenu, metodama i tehnikama razvoja ES.

Korisnici su dragocjeni izvor informacija o tome kako oni sada koriste eksperte u svom poslu i šta očekuju od sistema.

Na kraju, može se reći da je faza analize i dizajna faza "učenja" o domenu, načinu rješavanja problema, zadacima i svim metodološkim aspektima inženjeringa znanja.

Faza je završena kada projektantski tim ima oformljen pisani izvještaj za razvoj konkretnog ekspertnog sistema.

14.2.3. Razvoj prototipa ekspertnog sistema

U posljednje vrijeme, zahvaljujući prednostima koje pruža softver četvrte generacije, **prototipski razvoj** je široko prihvaćen već kod izgradnje upravljačkih informacionih sistema, a naročito kod razvoja sistema zasnovanih na znanju: sistema za podršku odlučivanju i ekspertnih sistema. Razlozi za ovakav pristup razvoju su brza izgradnja radnog modela, u ovom slučaju ekspertnog sistema, i simulacija komponenata sistema.

Razvoj prototipa ES podrazumjeva izvršenje slijedećih funkcija:

- ocjena, na osnovu izvršene analize i dizajna, da li ekspertni sistem može biti izgrađen,
- izgradnja prototipa.

Osnovni zadaci projektantskog tima u izgradnji prototipa su:

- razvijanje efikasnog odnosa ekspert - inženjer znanja,
- uzimanje znanja,
- formalizacija uzetog znanja,
- kodiranje znanja i umetanje u program.

Rezultat cjelokupnog rada projektantskog tima, na ovom nivou, je razvoj inicijalnog prototipa sistema.

Inicijalni prototip je prva verzija sistema, koji će rješavati jedan ili nekoliko problema, i koji će projektanti pokazati ekspertima, korisnicima i rukovodstvu projekta, da se izvrši ocjena kako se rješavaju inicijalni slučajevi. Test i dogradnja inicijalnog prototipa, u saradnji sa ekspertima, je nužna da bi se izvršila dorada prototipa i definisao finalni prototip, spreman za prezentaciju i ocjenu.

Prototip je osnova za razvoj cjelokupnog ekspertnog sistema. Eksperti mogu dati značajni doprinos u definisanju, usavršavanju i konačnom oblikovanju pravila, odnosno mogu eksperimentisati sa modelima: entitet - atribut – vrijednost (E-A-V).

Korisnici imaju sporednu ulogu u razvoju prototipa. Njihova uloga se, prije svega, svodi na davanje objašnjenja kako oni koriste heurističke metode u rješavanju problema u njihovom radnom okruženju. Pored toga, njihovi prijedlozi i sugestije za inicijalni i finalni prototip su veoma dragocjeni.

Uloga rukovodioca projekta se svodi na koordinaciju i kontrolu.

Faza razvoja je završena kada projektantski tim i eksperti mogu prezentirati prototip ekspertnog sistema i ponuditi plan razvoja finalnog sistema.

14.2.4. Razvoj ekspertnog sistema

Faza proširenja prototipa

Razvoj ekspertnog sistema je faza prirodnog nastavka i proširenja prototipa. Znanje iskorišteno za prototip je samo uzorak znanja. Sada se uzima, formalizuje, kodira i umeće u bazu znanja svo preostalo znanje koje se može obezbjediti od eksperta. Finalni ekspertni sistem treba da na efikasan način izvršava zadatke za koje je namjenjen i da, u skladu sa softverskim i hardverskim okruženjem, po mogućnosti bude integrisan u IS organizacije.

Veliki ekspertni sistemi zahtjevaju razvoj i upravljanje više složenih potprojekata, koji moraju biti koordinirani, kontrolisani i na kraju integrisani zajedno u kompleksnu aplikativnu cjelinu.

Povezivanje ES sa radnim okruženjem

Zbog sadržine i prirode ove razvojne faze, rukovodilac projekta u razvoj finalnog ES intenzivnije angažuju i uključuje rukovodioce drugih dijelova organizacije, jer sistem se obično svojom djelatnošću prostire u šire radno okruženje.

Korisnicima se mora posvetiti posebna pažnja, moraju biti psihološki i edukativno pripremljeni za prihvat i efikasnu primjenu sistema. Svaka verzija sistema mora biti prezentirana i kritički ocjenjena od strane korisnika. Korisnički interfejs, koji podržava dijalog između korisnika i ekspertnog sistema, mora biti razvijen uz korisnikovo učešće.

Uloga eksperta u završnoj fazi razvoja ES

U ovoj fazi razvoja ekspert, u saradnji sa projektantskim timom, ima naglašenu ulogu, a sastoji se u slijedećem:

- razrješavanje i definisanje stavova, sa rukovodiocem projekta, kakav sistem treba da bude;
- razlaganje prototipa sistema i rekonstruisanje baze znanja;
- razvoj procedura za povezivanje sistema sa bazama podataka i drugim aplikacijama. Za neke dijelove podbaze znanja, informacije se dobijaju iz baze podataka ili drugih programa;
- uzimanje, formatizovanje, kodiranje i unošenje nove količine znanja u bazu znanja, uvođenje dodatnih nivoa apstrakcije u prilagođavanju porasta količine znanja u bazi;
- promjena imena entiteta i atributa u toku eventualne rekonstrukcije baze znanja;
- koncipiranje i razvoj konačnog korisničkog interfejsa;
- razvoj postupaka za ažuriranje baze podataka;
- integracija komponenti sistema u koherentnu programsku cjelinu.

Faza razvoja ES je završena kada je ES spreman za testiranje i implementaciju u stvarnom radnom okruženju.

14.2.5. Testiranje i implementacija ekspertnog sistema

Testiranje i implementacija ES obuhvata slijedeće korake:

- uvođenje ES u radno okruženje korisnika,
- testiranje sistema,
- modifikacija sistema.

Prilikom testiranja ES naglasak je na dva ključna elementa:

(1) Da li je baza znanja dovoljno velika i snažna da omogući rješavanje problema zbog kojih je ES i izgrađen;
(2) Da li postoje inherentne kolizije u bazi znanja.

Testiranje treba da pokaže da li ES uspješno rješava sve slučajeve za koje se vjeruje da ih čovjek - ekspert takođe uspješno rješava.

Smatra se da ako ES uspješno rješava više od 80% slučajeva da je njegova baza znanja dovoljno velika i snažna i da se ES može uvesti u upotrebu.

Ciljevi testiranja i implementacije ES su:
- provjeravanje dejstva sistema u radnom okruženju i njegova nužna modifikacija,
- provjeravanje efikasnosti, brzine i inteligencije sistema i sprovođenje potrebnih modifikacija,
- stručno osposobljavanje korisnika za konačnu upotrebu ES.

Različiti ES podliježu različitim procedurama testiranja.

14.2.6. Održavanje ekspertnog sistema

Održavanje ES podrazumjeva obezbjeđivanje takvog funkcionisanja i takvih učinaka sistema, kakvi su bili predviđeni pri razvoju sistema. Održavanje sistema podrazumjeva i njegovo ažuriranje i unapređivanje, kao i eventualno otklanjanje uočenih nedostataka. ES posjeduje znanje, koje se mijenja i evoluira tokom vremena.

Najčešće, održavanje ES spada u djelokrug:
- inženjera znanja i/ili
- korisnika ES.

Korisnici mogu održavati samo one komponente koje ne zadiru u strukturalne nivoe. Oni sakupljaju i dodaju podatke, činjenice i informacije, a svo drugo održavanje, kao što je dodavanje novih pravila i slično, mora biti u saglasnosti sa inženjerom znanja.

Mogući model održavanja ES može da se sastoji od:

- korisnici prate funkcionisanje sistema i izvještavaju o uočenim greškama,
- eksperti izvještavaju o promjenama u znanju,
- inženjeri znanja vrše modifikaciju, restruktuiranje i proširivanje baze znanja novim elementima.

14.3. SREDSTVA ZA IZGRADNJU EKSPERTNIH SISTEMA

14.3.1. Opšte o sredstvima za izgradnju ES

Složenost i obim rašunarskih programa ES može da bude različita: demonstracioni prototipovi sadrže obično 50 do 100 pravila, dok pojedini komercijalni sistemi sadrže i nekoliko hiljada pravila. Izgradnja ES nije nimalo lak posao i u pojedinim slučajevima ga je skoro nemoguće obaviti ukoliko se ne raspolaže inteligentnim alatima i posebnim sredstvima, koji će taj posao olakšati i skratiti njegovo vremensko trajanje.

Najšire posmatrano, sredstva koja su u funkciji izgradnje ES, se mogu podjeliti u slijedeće grupe:

(1) Programski jezici za razvoj ES;
(2) Jezici inženjeringa znanja;

Programski jezici, bilo da su problemski orijentisani (FORTRAN, Pascal, C, itd.), objektno orijentisani (C++, Java, itd.) ili jezici za rad sa simbolima (LISP, PROLOG, itd.), pružaju najveću slobodu pri izgradnji ES, ali nisu pogodni za predstavljanje znanja ili pristup bazi znanja. Zbog određenih svojstava, za izgradnju ES najčešće se primjenjuje jezik PROLOG. Ovaj jezik će biti ukratko prikazan.

14.3.2. Programski jezik PROLOG

Naziv PROLOG je skraćenica od engleskih riječi *PRO(gramming) in LOG(ic)*, što znači da je riječ o programskom jeziku koji je namjenjen logičkom programiranju. Međutim, PROLOG ima i čitav niz drugih dodatnih imena, kao što su: **jezik vještačke inteligencije, nedeterministički programski jezik, jezik računara pete generacije, deklarativni programski jezik,** itd. Svaki od ovih dodatnih naziva ističe neku važnu osobinu PROLOG-a. To su osobine koje PROLOG čine drugačijim od ostalih rasprostranjenih programskih jezika.

Jezik logičkog programiranja PROLOG je posebna oblast primjene rezolucijskog dokazivanja teorema. Više o ovome u literaturi [88, poglavlje 4].

OSNOVE PROLOG-a

Teorijsku osnovu PROLOG-a čini predikatski račun I reda. Međutim, postupak nalaženja rješenja u PROLOG-u se zasniva na principu rezolucije za automatsko dokazivanje teorema. Riječ je o jednoj modifikaciji principa rezolucije za tzv. Hornove disjunkte.

Osnovni elementi PROLOG-a se nazivaju **Hornovi disjunkti - klauzule** i predstavljaju rečenice oblika:

ako c_1, c_2, ... , c_n onda c

Ovo se može napisati i u obliku:

c ako c_1, c_2, ... , c_n

ili u uobičajenoj notaciji, gdje se umjesto riječi ako koristi specijalni simbol :- (tzv. **vrat - simbol**, jer spaja glavu predikata c sa tijelom: c_1, c_2, ... , c_n)

c:- c_1, c_2, ... , c_n.

Predikati koji nemaju tijelo u PROLOG-u se nazivaju **činjenice**. Predikati bez glave se nazivaju **ciljevi** ili **upiti**.

U opštem slučaju, u matematičkoj logici, pogodbene rečenice mogu imati oblik:

ako p_1, p_2, ... , p_n onda c_1, c_2, ... , c_k.

Ovdje je riječ o klauzuli koja ima više glava i takve klauzule se ne koriste u PROLOG-u. Ako klauzula ima samo jednu glavu ili nema glavu, tada se naziva Hornova klauzula.

Princip rezolucije u PROLOG-u se primjenjuje na Hornove klauzule. Sve činjenice u PROLOG-u se tretiraju kao apsolutne istine, tj. **aksiome**.

Primjena metode rezolucije se sastoji u usaglašavanju postavljenog cilja (upita) sa činjenicama, tj. bazom podataka. Ako je postavljen cilj, PROLOG nastoji da ispuni taj cilj u skladu sa bazom činjenica i bazom pravila. Ako PROLOG u svom nastojanju uspije da ispuni cilj, daje pozitivan odgovor "**YES**", u suprotnom negativan "**NO**".

NAČIN RJEŠAVANJA PROBLEMA U PROLOG-u

U Hornovim klauzulama c_1, c_2, ... , c_n su klauzule (rečenice) koje mogu sadržavati konstante i promjenljive. Elementi c_1, c_2, ... , c_n se nazivaju **potciljevi**.

Glavni cilj može uspjeti (biti ispunjen) ako uspije svaki potcilj u okviru glavnog cilja. Prema tome, glavni cilj:

:- c_1, c_2, ... ,c_n

(koji može biti označen i na slijedeći način: **?- c_1, c_2, ... , c_n** predstavlja konjunkciju potciljeva: **c_1, c_2 , ... , c_n**.

Procedure

Glavni cilj se može shvatiti kao poziv procedure. Procedure se zadaju preko predikata sa odgovarajućim imenom i odgovarajućim brojem argumenata.

Potciljevi se ispunjavaju u procesu nalaženja rješenja, tj. u procesu izračunavanja. Proces rješavanja može da se razgrana različitim putevima i da generiše više različitih rješenja.

U procesu ispunjavanja cilja mogu nastati tri slučaja:

(1) Proces traženja se završava i daje određeni rezultat. U ovom slučaju riječ je o uspješnom ispunjavanju cilja (što će biti označeno sa '[ ]');
(2) Proces traženja se završava neuspješno i ne daje rezultat, tj. cilj nije ispunjen (što će biti označeno sa '?-');
(3) Proces traženja se ne završava i ne daje nikakav rezultat.

U slučajevima (1) i (2) riječ je o završenom izračunavanju, a u slučaju (3) o nezavršenom.

Prostor izračunavanja

Za svaki cilj postoji potpuni prostor izračunavanja, koji može da se razmatra kao skup svih puteva duž kojih se može tražiti. Duž nekog od tih puteva traženje se može završiti uspješno, kod drugih neuspješno, a kod nekih traženje može da se ne završi. Potpuni prostor izračunavanja određen je ciljem i pravilima za izračunavanje.

Načini zadovoljavanja ciljeva

PROLOG ispunjava glavni cilj tako što ispunjava svaki potcilj u okviru cilja. Standardni način zadovoljavanja potciljeva realizuje se počev od prvog lijevog

potcilja. Ukoliko se ispuni prvi lijevi potcilj, PROLOG nastoji da ispuni prvi potcilj iza njega, itd. Ako u procesu ispunjavanja neki potcilj ne može biti ispunjen, PROLOG se vraća na prethodni potcilj i nastoji da ga ispuni za neke druge vrijednosti. Taj postupak se nastavlja sve dok ne budu iscrpljene sve mogućnosti.

Postupak traženja sa vraćanjem (*back tracking*) omogućava nalaženje svih rješenja iz skupa mogućih rješenja i karakterističan je za PROLOG.

Na narednom primjeru će biti objašnjeni prethodno pomenuti pojmovi.

***PRIMJER**: Zadan je slijedeći PROLOG program:*

```
vozač_djaka(aco).
vozač_djaka(bob).
vozač_djaka(X) :- dobar_vozač(X).

dobar_vozač(X) :- ne_pije(X), vozač_autobusa(X).
dobar_vozač(oto).

ne_pije(tom).
ne_pije(ivo).

vozač_autobusa(tom).
vozač_autobusa(pol).
vozač_autobusa(ivo).
```

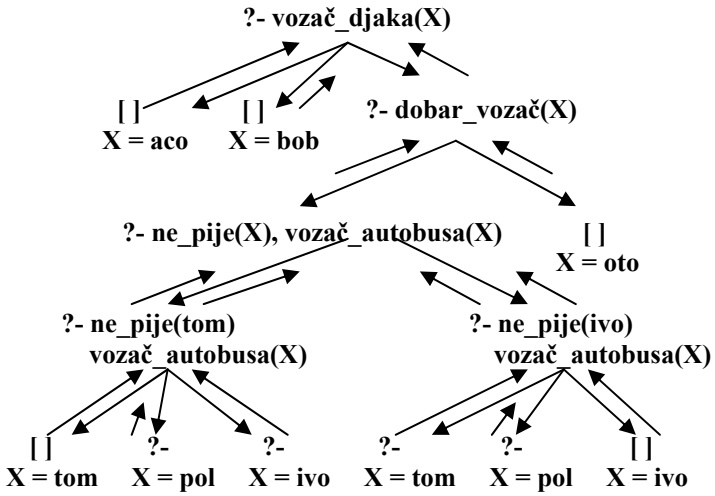
Navedeni program se sastoji od Hornovih klauzula sa glavom, koje predstavljaju činjenice i predikate. Slovom X je označena jedna promjenljiva, a ostala imena, koja ne počinju velikim slovom, su konstante. Upit:

```
?- vozac_djaka(X).
```

predstavlja Hornov disjunkt bez glave. PROLOG nastoji da ispuni ovaj cilj nalazeći konkretne vrijednosti za promjenljivu X, tako da cilj bude usaglašen sa skupom činjenica.

Stablo pretraživanja

Potpuni prostor izračunavanja može da se predstavi slijedećim stablom pretraživanja, slika 14.1.



Slika 14.1. Stablo pretraživanja.

Stablo pretraživanja je konačno, što znači da se proces nalaženja rješenja ne može neograničeno produžavati. Razlikuju se listovi označeni sa [ ], koji predstavljaju čvorove uspjeha i listovi označeni sa ?-, koji predstavljaju čvorove neuspjeha.

Iz stabla pretraživanja, prikazanog na slici 14.1, se mogu sagledati svi putevi duž kojih postoji mogućnost da zadani cilj bude ispunjen. PROLOG, na zahtjev korisnika, može da pronađe sva rješenja koja ispunjavaju zadani cilj.

Za razliku od PROLOG-a, kod proceduralnih jezika nije uvijek jednostavno naći sva rješenja.

Način pretraživanja stabla

Prilikom traženja rješenja, PROLOG nastoji da ispuni cilj usaglašavajući ga sa bazom podataka, počev od vrha ka dnu. To znači da PROLOG započinje sa traženjem rješenja počev od prvog lijevog čvora u stablu pretraživanja, a zatim nastavlja obilazak stabla pretraživanja nalazeći sva rješenja. Ako u procesu traženja naiđe na čvor uspjeha, PROLOG izdaje rješenje, u suprotnom ne obavještava da je došao do čvora neuspjeha.

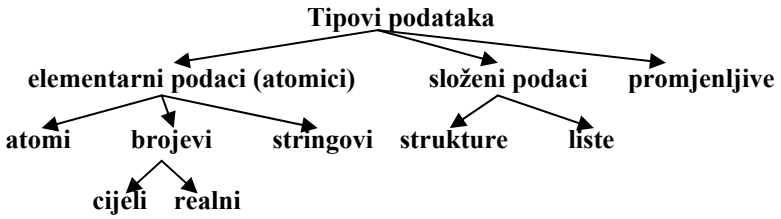
Način rada PROLOG - interpretera može se djelimično sagledati iz redoslijeda kojim izdaje rješenja. Testiranjem prethodnog programa se dobija:

```
?- vozač_djaka(X).
X = aco →;
X = bob →;
X = tom →;
X = ivo →;
X = oto →;
no
```

Proces traženja se odvija kretanjem od korjena stabla ka listovima, ali i od listova ka korjenu stabla. Kretanje od listova ka korjenu predstavlja traženje sa vraćanjem. Na slici 14.1. je strijelicama prikazan smjer traženja rješenja.

STRUKTURA PODATAKA U PROLOG-u

Tipovi podataka, koji omogućavaju pisanje PROLOG programa, prikazani su na slici 14.2.



Slika 14.2. Tipovi podataka u PROLOG-u.

Ime promjenljive

Ime promjenljive je sekvenca karaktera, cifara ili znaka '_', koja počinje velikim slovom.

Atomi

Atomi su tekstualne konstante, koje mogu sadržavati slova, cifre ili specijalne simbole (<, -, >, ...). Ako počinju velikim slovom pišu se između jednostrukih navodnika. Atom koji počinje malim slovom ne smije sadržavati specijalne znake.

Interval dozvoljenih vrijednosti za **cijele i realne brojeve** (ako postoje) zavisi od konkretne implementacije PROLOG-a.

Strukture

Strukture predstavljaju tip podataka generalne namjene, koji izražavaju vezu između **termova**. Sastoje se od **funktora** (ime strukture) i **argumenata**. Broj argumenata se naziva **arnost** strukture.

PRIMJER:

prevozno_sredstvo (automobil(reno,21,1995),francuska)
Funktor je **prevozno_sredstvo***, arnost je 2.*

Složene strukture

Za pisanje programa u PROLOG-u se koriste složene strukture u okviru kojih se mogu nalaziti promjenljive.

PRIMJER:

prevozno_sredstvo (automobil(reno,X,Y),Z)

Liste

Često se u PROLOG-u liste tretiraju kao specijalni slučajevi struktura. Rad sa listama predstavlja osnovu rješavanja mnogih problema u PROLOG-u.

Lista je struktura podataka koja predstavlja sekvencu elemenata, koji mogu biti različitog tipa (atomi, brojevi, promjenljive, liste, itd.) i pišu se u srednjim zagradama.

PRIMJERI:

[ ] *- prazna lista,*
[a] *- lista sa jednim elementom,*
[a|X] *- lista sa glavom a i repom X.*

X je promjenljiva koja predstavlja listu koja čini preostali dio liste [a|X]. Ako je dana lista [a, b, c, d], koja se pri radu programa unifikuje sa [a|X], tada se X unifikuje sa [b, c, d].

ivo,[1, 2], 12] - *lista sa tri elementa*.

Veliki broj problema, koji zahtjeva upotrebu složenijih struktura podataka, rješava se svođenjem na elementarne predikate za rad sa listama.

UNIFIKACIJA

Za razliku od proceduralnih programskih jezika, kod kojih se osnovni način izvršavanja programa zasniva na dodjeljivanju vrijednosti promjenljivim, u PROLOG-u se koristi generalni tip dodjele, tzv. unifikacija. Moguće su unifikacije između različitih struktura podataka.

(1) Unifikacija promjenljive sa promjenljivom je uvijek moguća. Ako neka od promjenljivih postane konkretizovana (instancirana) nekom drugom strukturom, takva postaje i druga promjenljiva;
(2) Unifikacija atomika ili strukture sa promjenljivom, izvršava se konkretizovanjem (instanciranjem) promjenljive tim atomikom ili strukturom;
(3) Atomik sa atomikom se unifikuje ako im se vrijednosti poklapaju;
(4) Struktura se unifikuje sa strukturom (term sa termom, lista sa listom) ako se funktori poklapaju, a argumenti se mogu unifikovati. Ako struktura sadrži podstrukture, unifikacija je rekurzivna.

Unifikacijom dvije strukture pravi se skup zamjena, preko kojih se promjenljivim komponentama dodjeljuju vrijednosti i upoređuju odgovarajuće, već konkretizovane komponente.

Ukoliko se pojavi razlika samo u jednom paru odgovarajućih komponenti dvije strukture, te dvije strukture se ne mogu unifikovati.

***PRIMJER**: Ako postoje slijedeće dvije strukture:*

prva(računar, X, tastatura, Y).

i

prva(Z, monitor, tastatura, štampač).

ove dvije strukture se mogu unifikovati konkretizovanjem (instancijacijom) promjenljivih:

X = monitor, Y = štampač i Z = računar.

Strukture:

druga(kuća, X, vrata).

i

druga(zgrada, prozor, Y).

se ne mogu unifikovati, jer komponente ***zgrada*** *i* ***kuća*** *ne mogu da se izjednače.*

PREDIKAT ODSJECANJA I TRAŽENJA RJEŠENJA

U PROLOG-u postoji mogućnost da se spriječi pretraživanje obilaskom cijelog stabla pretraživanja. U te svrhe se koristi sistemski predikat odsjecanja, koji se najčešće označava uskličnikom "!". Pomoću ovog predikata na stablu pretraživanja se odsjecaju pojedine grane i na taj način se spriječava traženje rješenja u potpunom prostoru izračunavanja. Korištenjem ovog predikata se narušava deklarativno svojstvo PROLOG jezika.

PRIMJER:

Da bi se sagledali efekti predikata odsjecanja, biće modifikovan prethodno navedeni program na slijedeći način.

```
vozač_djaka(aco).
vozač_djaka(bob).
vozač_djaka(X) :- dobar_vozač(X).

dobar_vozač(X) :- ne_pije(X), !, vozač_autobusa(X)./*
modifikovani red */
dobar_vozač(oto).

ne_pije(tom).
ne_pije(ivo).

vozač_autobusa(tom).
vozač_autobusa(pol).
vozač_autobusa(ivo).
```

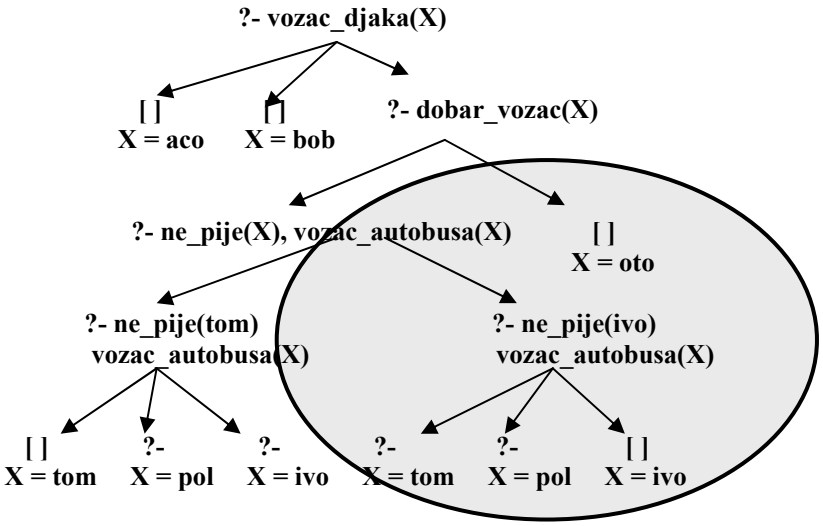
Testiranjem se dobija slijedeći skup rješenja:

```
?- vozač_djaka(X).
X = aco →;
```

```
X = bob →;
X = tom →;
no
```

Stablo pretraživanja sa odsječenim granama

Do ovog skupa rješenja se dolazi zato što su određene grane odsječene na stablu pretraživanja. Za ovaj slučaj stablo pretraživanja dobija izgled kao na slici 14.3. Uočava se da su određene grane na stablu pretraživanja odsječene i da su time odstranjeni određeni putevi za nalaženje rješenja.



Slika 14.3. Stablo pretraživanja sa odsječenim granama.

KRATKI PREGLED OSNOVNIH OPERATORA I PREDIKATA

, Označava 'AND', određuje konjunkciju ciljeva;

; Označava 'OR', određuje disjunkciju ciljeva. Izvršava se slijeva nadesno i da bi uspio mora biti ispunjen cilj bar sa jedne strane operatora ';';

! Operator odsjecanja. Cilj koji uspjeva kada se do njega dođe pri izvršavanju "nadolje", a propada kada se do njega dođe pri traženju sa vraćanjem. U tom slučaju propada i cio predikat u okviru koga se našao operator odsjecanja;

true Cilj koji uvijek uspjeva;
false Cilj koji uvijek propada. Pogodan je kada se želi traženje sa vraćajem;
repeat Kada se naiđe na njega uspjeva i pri traženju sa vraćanjem. Realizovan je tako da omogući formiranje ciklusa pri izvršavanju programa. Moguće ga je realizovati na slijedeći način:

```
repeat.
repeat :- repeat.
```

not(P) Ako P uspije, onda not(P) propada, a ako P propadne tada not(P) uspjeva. Može se realizovati na slijedeći način:

```
not(P) :- P , ! , fail.
not(P).
```

read(P) Omogućava unošenje podataka.
write(P) Omogućava izdavanje podataka.
nl Prelazi u novi red prilikom izdavanja podataka.

PROLOG I NJEGOVA SINTAKSA PRI IZGRADNJI ES

PROLOG je programski jezik kreiran sa posebnom namjenom da se uz njegovu pomoć dobijaju odgovori na upite iz baze znanja, koja je sastavljena od činjenica i pravila. U PROLOG-u su ugrađene rutine olančavanja unaprijed i olančavanja unazad, tako da se između pravila pisanih u PROLOG-u i pravila pohranjenih u bazi znanja uspostavlja obostrano jednoznačna korespodencija.

PROLOG i njegova sintaksa zadovoljavaju slijedeće osobine:

(a) Cjelokupno pravilo napisano u PROLOG-u se naziva **klauzula**. Postoje dva tipa klauzula:

- klauzula - pravilo,
- klauzula - činjenica.

PRIMJER *klauzule - pravila:*

```
temperatura(da):-write('POVIŠENA'),read(POVIŠENA),POVIŠENA=da.
```

Write *i* read *dio pravila omogućavaju da se sazna vrijednost promjenljive* POVIŠENA*, tako što se zahtjeva od korisnika da se izjasni da li je* temperatura *povišena ili nije. Ukoliko je korisnikov odgovor* "da", *dio pravila* POVIŠENA=da *će biti tačan iskaz, u protivnom netačan.*

PRIMJER *klauzule - činjenice:*

vršnjaci(X,Y) :- rođen(X,G),rođen(Y,G).
rođen(ivo,1975).
rođen(ana,1972).
rođen(aco,1975).

(b) Znak :- se čita kao IF (ako).

(c) Dio pravila koji se nalazi sa lijeve strane znaka :- se naziva **glava klauzule** i ekvivalentan je THEN dijelu pravila u bazi znanja ekspertnog sistema.

(d) Desno od znaka :- se nalazi tzv. **tijelo klauzule** i odgovara IF dijelu produkcionih pravila, kojima se znanje predstavlja u bazi. Tijelo je sačinjeno od više dijelova, razdvojenih zarezima. Svaki dio se naziva **ciljem**. Zarezi imaju značenje logičke konjunkcije.

(e) Na kraju cijele klauzule je obavezna tačka.

(f) Riječ ispred zagrade je **predikat** i uvijek se piše malim početnim slovom. U navedenim primjerima to su: temperatura, write, read, vršnjaci i rođen. Termin 'predikat' se često koristi u smislu procedura.

Procedura

Procedura je dio programa (niz klauzula) koji definiše predikat određenog imena i arnosti. Iako se definicija predikata najčešće posmatra deklarativno, ona je i proceduralna (redoslijed klauzula i ciljeva je bitan). Zbog toga termin 'predikat' može stvoriti zabunu, te je bolje koristiti termine 'procedura', 'ugrađena procedura' i slično.

(g) Riječi u zagradama su **argumenti** predikata ispred zagrade.

Argumenti mogu biti:
- promjenljive,
- vrijednosti.

Promjenljive se pišu velikim početnim slovom, a vrijednosti malim.

Postavljanje determinističkih pitanja o činjenicama

Programski jezik PROLOG pruža mogućnost postavljanja pitanja o objektima i relacijama posmatranog domena. Pitanja o činjenicama pohranjenim u bazi su najjednostavnija, a sintaksa im je slijedeća:

?- klauzula

Suočen sa zadatkom da odgovori na pitanje, računar daje rješenje sa **yes** (potvrdan odgovor) ili **no** (odrečan odgovor).

***PRIMJER**: Za podatke iz prethodnog primjera. na pitanje:*

?- vršnjaci(ivo,ana)

računar će odgovoriti sa **no***, dok će odgovor na pitanje:*

?- vršnjaci(ivo,aco)

biti potvrdan **(yes)**.

Potrebno je napomenuti da se odgovori daju na osnovu podataka pohranjenih u bazi, odnosno negativan odgovor treba da se tumači u smislu da se na osnovu postojećih podataka ono što je pitanjem zahtjevano ne može zaključiti, a analogno to važi i za potvrdan odgovor. To bi značilo da se odgovor **yes** ne poistovjećuje sa **tačno**, a odgovor **no** sa **netačno**. Do sada navođena pitanja su bila deterministička: svako pitanje je imalo tačno jedan odgovor. Takav je slučaj sa konvencionalnim i funkcionalnim programskim jezicima (Pascal, FORTRAN, LISP i dr.).

Postavljanje nedeterminističkih pitanja o činjenicama

Budući da se PROLOG zasniva na teoriji relacija, a ne funkcija, u PROLOG-u su moguća i nedeterministička pitanja, u smislu da postoji više od jednog ispravnog odgovora na pitanje.

***PRIMJER**: Na pitanje:*

?- rođen(X,1975)

računar daje dva odgovora, na osnovu baze iz prethodnog primjera, i to:

X = ivo
X = aco

***PRIMJER** programa na programskom jeziku PROLOG.*

Dana je baza znanja o starim automobilima. Odštampati spisak svih automobila iz baze koji su proizvedeni poslije 1980. godine. Nakon toga odštampati za sve automobile karakteristike njihovih brzina.

```
/*****************************************************/
/*                  Stari automobili                 */
/*****************************************************/
auto(škoda,1987,120).
auto(opel,1982,180).
auto(fićo,1964,100).
auto(porše,1981,200).
auto(lada,1988,140).

prvi:- auto(A,X,Y),
  ifthenelse(X > 1980,write(A),write('staro-gvoždje')),
  nl,fail.

drugi :- auto(A,X,Y),
         case([Y ≤ 120 → štampa(A,'spor'),
               Y ≤ 160 → štampa(A,'prosjecno brz')|
                         štampa(A,'brz')]),
         nl, fail.
štampa(A,X) :- write(A),
               write(' je '),
               write(X),
               write(' auto.').

/*****************************************************/
/*                  Test primjeri                    */
/*****************************************************/

?- prvi.                    ?- drugi.
škoda                       škoda je spor auto.
opel                        opel je brz auto.
staro-gvoždje               fićo je spor auto.
porše                       porše je brz auto.
lada                        lada je prosječno brz auto.

no                          no
```

14.3.3. Jezici inženjeringa znanja

Programski jezici na najvišem nivou, koji su isključivo namjenjeni izgradnji ES, nazivaju se jezici inženjeringa znanja. Ovi jezici pružaju posebne olakšice u radu, ali su za predstavljanje i manipulisanje znanjem manje fleksibilni od programskih jezika opšte namjene.

Jezici inženjeringa znanja se sastoje od jezika za izgradnju ES, kojima je pridodano široko razvojno okruženje. Ukoliko je jeziku inženjeringa znanja, da bi prerastao u ES, potrebno dodati samo određeno domensko znanje, tada je riječ o **jeziku ogoljenog tipa**. Takvi jezici se nazivaju **skeletima ES**.

PRIMJER:

Korištenjem jezika inženjeringa znanja KAS, koji je namjenjen klasifikaciji i dijagnostici, i dodavanjem geološkog znanja (kao domenskog znanja), izgrađen je ES PROSPECTOR.

Skeleti ES, pored već pripremljene strukture, pružaju niz olakšica za brzi i jednostavni razvoj ES, ali im nedostaje opštost i fleksibilnost u rukovanju problemima koji ne spadaju u unaprijed odabranu klasu. Jezici inženjeringa znanja opšte namjene pružaju bolju kontrolu pristupa podacima i bolje pretraživanje od jezika ogoljenog tipa, ali im je upotreba složenija.

14.4. RAZVOJ EKSPERTNOG SISTEMA

14.4.1. Mehanizam logičkog zaključivanja

U narednom izlaganju biće razmotren način funkcionisanja mehanizma za zaključivanje, koji obavlja proces logičkog rasuđivanja nad **bazom znanja**, predstavljenom u obliku produkcionih pravila, i **bazom činjenica** koja opisuje stanje sistema.

Da bi mehanizam za zaključivanje mogao da započne postupak logičkog rasuđivanja, potrebno je prvo postaviti cilj rasuđivanja i inicijalizirati sve činjenice potrebne za testiranje uslova, odnosno izvršavanje odgovarajućih akcija.

Inicijalizacija se vrši uz pomoć vrijednosti, koje se nalaze u bazi činjenica, ili se može raditi **interaktivno**, s tim što tada mehanizam zaključivanja postavlja pitanja korisniku tokom postupka korištenja ekspertnog sistema. Za efikasnu realizaciju dijaloga sa korisnikom, potrebno je da mehanizam zaključivanja ponudi korisniku i skup legalnih odgovora, koji su dozvoljeni za inicijalizaciju pojedinih činjenica, i na taj način vodi korisnika u pravcu da daje odgovore koje sistem razumije.

Mehanizam zaključivanja treba, takođe, da sadrži i takozvana meta znanja, koja ustvari predstavljaju pravila, meta pravila, koja upravljaju postupkom logičkog rasuđivanja u smislu njegove optimizacije. Ova meta pravila se, prije svega, odnose na grupisanje, odnosno utvrđivanje pravila i način njihovog pretraživanja, da bi se obezbjedilo minimalno moguće vrijeme pretraživanja pravila.

Obzirom na postavljeni cilj logičkog rasuđivanja, odnosno cilj konsultacije koja se očekuje od ekspertnog sistema, moguća su dva opšta pristupa:

- direktno rasuđivanje (sistem sa olančavanjem unaprijed),
- inverzno rasuđivanje (sistem sa olančavanjem unazad).

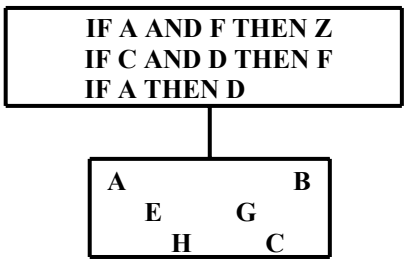
14.4.2. Direktno rasuđivanje

Direktno rasuđivanje, ponekad nazvano "sistem sa olančavanjem unaprijed", se bazira na ideji da se **svako pravilo ispituje počev od uslova ili premise**. Akcija (zaključak) se ne izvršava sve dok odgovarajući uslov ne postane tačan. Ukoliko uslov datog pravila nije tačan, njegova akcija se trenutno ne izvršava, već se prelazi na ispitivanje slijedećeg pravila. Napušteno pravilo se ispituje ponovo u slijedećem prolazu kroz pravila.

Izvršavanjem akcija jednog pravila se u stvari mijenja stanje uslova nekog drugog pravila, na takav način što taj uslov može postati tačan. Ovo je jedino moguće ukoliko su činjenice u datom uslovu posljedica akcija iz nekih drugih pravila. Tokom ispitivanja uslova, moguće je naići na uslov za koji je nemoguće odrediti njegovu logičku vrijednost, jer u bazi stanja sistema nije moguće naći vrijednosti za činjenice koje učestvuju u formulaciji uslova. Tada, obično, **mehanizam zaključivanja postavlja pitanja korisniku**, nudeći mu kao mogućnost skup dozvoljenih odgovora, ili poziva druge programe koji će izračunati vrijednosti zahtjevanih činjenica.

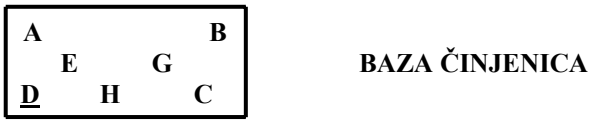
Ilustracija metode direktnog rasuđivanja

Metoda direktnog rasuđivanja se može ilustrovati koristeći slijedeći skup produkcionih pravila i njemu asociranu inicijalnu bazu činjenica (stanja), slika 14.4. Podatak da se recimo A nalazi u bazi činjenica označava da uslov, gdje se A pojavljuje, asocira logičku vrijednost TAČAN.



Slika 14.4. Skup produkcionih pravila i baza činjenica.

Obzirom na dani skup pravila, kao i početno stanje baze činjenica, prvim prolazom kroz bazu pravila, odnosno ispitivanjem logičkog stanja odgovarajućih uslova, biće generisano slijedeće stanje baze činjenica:



gdje je D nova činjenica koja se pojavila u bazi činjenica, a zatim:

i na kraju:

Ukoliko je Z bio cilj zaključivanja, navedeni sistem pravila obezbjeđuje, zajedno sa inicijalnim stanjem baze činjenica, zaključak da Z ima tačnu vrijednost. U postupku zaključivanja, takođe, se može ustanoviti da i D i F imaju tačnu vrijednost.

Nedostatak direktnog rasuđivanja

Nedostatak direktnog rasuđivanja je da se u principu moraju ispitivati sva pravila, pa i ona koja nisu relevantna za postavljeni cilj zaključivanja. U slučaju

skupa sa velikim brojem pravila ovo može da znači veliki gubitak vremena na neproduktivna zaključivanja. U takvim situacijama, metoda zaključivanja koja se naziva inverzno rasuđivanje može da bude mnogo pogodnija.

Treba imati u vidu da je **direktno rasuđivanje pogodno** ukoliko se žele saznati svi mogući zaključci koji se mogu izvući na osnovu date baze činjenica i baze pravila zaključivanja, a ne samo krajnji zaključak, odnosno cilj konsultacije.

14.4.3. Inverzno rasuđivanje

Inverzno rasuđivanje, koje se, takođe, naziva "sistem sa olančavanjem unazad" ili "ciljno rasuđivanje", se bazira na ideji da se **pravila razmatraju počev od akcije**, **odnosno zaključka**. U ovom slučaju se počinje od pravila koja kao svoj zaključak sadrže cilj zaključivanja.

Kod inverznog rasuđivanja se skup pravila dijeli u podgrupe koje je potrebno ispitivati da bi se došlo do željenog cilja zaključivanja, čime se značajno sužava prostor pretraživanja i u kraćem vremenu dolazi do cilja zaključivanja. Isto kao i u slučaju direktnog rasuđivanja, tokom ispitivanja uslova, moguće je naići na uslov za koji je nemoguće odrediti njegovu logičku vrijednost, jer u bazi stanja sistema ne postoje vrijednosti za činjenice koje učestvuju u formiranju uslova. U tom slučaju mehanizam zaključivanja **postavlja pitanja korisniku**, nudeći mu kao mogućnost skup dozvoljenih odgovora, ili pak poziva druge programe koji će izračunati vrijednosti zahtjevanih činjenica.

Ilustracija metode inverznog rasuđivanja

Pretpostavka je da je kao u prethodnom primjeru cilj zaključivanja nalaženje istinite vrijednosti Z.

U prvom koraku metoda inverznog rasuđivanja pretražuje bazu činjenica da utvrdi da li postoji Z. Kad ustanovi da Z ne postoji, pronalazi pravilo u kome se Z pojavljuje kao zaključak. Ukoliko u bazi pravila postoji više ovakvih pravila, onda se uz pomoć odgovarajućeg meta pravila odlučuje koje će se koristiti. U danom primjeru, pravilo koje zaključuje Z je:

IF A AND F THEN Z.

Prema tome, mehanizam zaključivanja ispituje bazu činjenica za postojanje A i F, i rasuđuje da kao novi podcilj mora da postoji F.

U drugom koraku se sprovodi zaključivanje za novo postavljeni podcilj F i pronalazi pravilo:

IF C AND D THEN F,

na osnovu čega mehanizam zaključivanja mora da postavi novi podcilj D, koji je moguće zaključiti u trećem koraku na osnovu pravila:

IF A THEN D.

Po zaključivanju D, moguće je u slijedećem koraku zaključiti F i u **zadnjem koraku** glavni cilj Z.

Može se zaključiti da metoda inverznog rasuđivanja formira hijerarhiju podciljeva, u čijem se korjenu nalazi glavni cilj zaključivanja.

14.4.4. Sistemi sa olančavanjem unazad

U narednom tekstu će biti objašnjene faze izgradnje jednostavnih ES, koji se najčešće realizuju kao sistemi sa olančavanjem unazad (*Backward Chaining Systems* - BCS). BCS se najčešće koriste u slučajevima kada je potrebno da se odredi vrijednost određenog cilja.

Kao primjer će poslužiti ES za odabir video rekordera. Da bi se izgradio BCS, potrebno je, najčešće iterativno, proći kroz četiri faze:

1. Definisanje problema;
2. Izgradnja početnog skupa pravila;
3. Poboljšanje izgrađenog sistema;
4. Prilagođavanje načina zaključivanja i kontrola sistema.

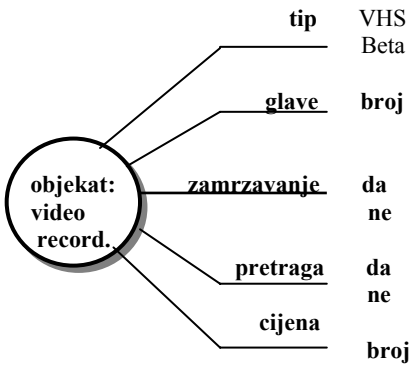
Definisanje problema

U ovoj, veoma bitnoj fazi izgradnje, se vrši proučavanje problema, sa kojim će se budući ES suočavati. U razmatranom slučaju to je pružanje pomoći kupcima video rekordera. Kupac postavlja prodavcu mnoštvo pitanja. Da se pojednostavi primjer, pretpostavka je da se odluka donosi samo na osnovu pet osnovnih karakteristika:

1. Tip (VHS ili Beta);
2. Cijena;

3. Broj glava;
4. Da li omogućavaju zamrzavanje slike;
5. Da li imaju mogućnost pretraživanja.

Nakon ovoga razmatranja potrebno je konstruisati Dijagram objekata (u razmatranom slučaju to je samo video rekorder).



Slika 14.5. Dijagram objekata za primjer kupovine video rekordera.

Nakon definisanja objekta, potrebno je informisati se o načinu na koji ekspert dolazi do rješenja. U razmatranom primjeru se može pitati prodavac kaja pitanja kupci najčešće postavljaju, osim toga mogu se pogledati prospekti i slično.

Kada se definišu bitne karakteristike za odlučivanje, mora se donijeti odluka na koji način će biti opisani i dodjeliti im imena. Osim toga, potrebno je definisati cilj, a to je upućivanje preporuke kupcu.

Definisanje početnog skupa pravila

Prvo se definiše pravilo koje treba da neposredno dodjeljuje vrijednost cilju.

PRIMJER: *Video rekorder VCX_1000 je VHS, sa četiri glave, omogućava zamrzavanje slike i pretragu, a cijena je relativno niska.*

Pravilo 1:

```
If tip=VHS       AND
   Glave=4       AND
   Zamrzava=da   AND
   Pretraga=da   AND
   Cijena=niska
THEN  preporuka=VCX_1000.
```

Ako sistem posjeduje samo pravila koja neposredno dodjeljuju vrijednost cilju, PREPORUKA za razmatrani sistem ima samo jedan nivo. Takav sistem će funkcionisati tako što se kupci pitaju za potrebe, a nakon toga se aktivira pravilo, koje se slaže sa tim potrebama. Podrazumjeva se da je potrebno definisati onoliko pravila ovog tipa, koliko ima različitih video rekordera. Ovakav sistem ne sadrži mnogo znanja. Osim toga, ne može uvijek riješiti problem, te je neophodno uvesti apstrakciju u sistem.

Apstrakcija u ovim sistemima se predstavlja unutrašnjim pravilom koje praktično vrši zaključivanje na osnovu ulaza korisnika. U razmatranom primjeru potrebno je uvesti apstrakciju koja bi omogućila kupcima da unesu vrijednost za atribut glave, obzirom da prosječni kupac ne zna koliko je potrebno glava da bi njegov video rekorder davao jasnu sliku, a vrlo često ne znaju da su video rekorderu uopšte potrebne glave. Zbog toga se mora utvrditi na koji način prodavac zaključuje o tome koliko je potrebno glava. Prodavac odlučuje o tome na osnovu toga da li kupac zahtjeva veoma kvalitetnu projekciju ili ne. Ako je odgovor potvrdan, tada je neophodno da video ima najmanje četiri glave, pa se uvodi novo pravilo:

Pravilo 2:

```
IF kvalitet_slike = važan
THEN glave =>4.
```

Na ovaj način je uveden drugi nivo u sistem. Pravilo broj dva će dodjeljivati vrijednost cilju glave, nižem po hijerarhiji. Sada će sistem raditi na slijedeći način: prvo će se ispitivati istinitosna vrijednost premisa koje se nalaze u prvom pravilu, kada sistem pokuša da utvrdi vrijednost za tip tada će upitati kupca za tip (VHS ili Beta). Ako je odgovor VHS, tada će sistem preći na ispitivanje druge premise

pravila 1. Da bi utvrdio tu istinitost potrebno je da od kupca dobije odgovor o tome kakav kvalitet slike želi i ispita pravilo 2, da bi pronašao vrijednost za atribut glave. Ukoliko je druga premisa tačna, prelazi se na slijedeću i tako dalje. Ako su sve premise istinite, tada se cilju PREPORUKA dodjeljuje vrijednost VCX_1000, pravilo 1. U slučaju da je bar jedna od premisa neistinita, prelazi se na ispitivanje istinitosti premisa ostalih pravila.

Poboljšanje izgrađenog sistema

Pod poboljšanjem se podrazumjeva:

1. Provjera da li sistem dodjeljuje vrijednost cilju (promjenljivoj PREPORUKA);
2. Dodavanje tekstualnih poruka, koje treba da omoguće komunikaciju između ES i korisnika;
3. Osposobljavanje sistema za rješavanje situacija kada treba više preporuka dodjeliti cilju;
4. Poboljšavanje pravila;
5. Omogućavanje iterativnog korištenja sistema.

Dodavanje pravila da bi se osiguralo dodjeljivanje vrijednosti cilju predstavlja prvo poboljšanje. Prilikom kreiranja ES se teži da se sistem osposobi da za bilo koje ulazne parametre može da da savjet. Međutim, u nekim slučajevima to nije moguće. U razmatranom primjeru se pretpostavlja da kupac zahtjeva visoki kvalitet, ali je spreman da izdvoji malu sumu novca. U tom slučaju ni u jednom od pravila neće biti istinite sve premise (pod pretpostavkom da takav video rekorder ne postoji) i PREPORUKA neće dobiti vrijednost. Nepostojanje vrijednosti cilja, odnosno ne davanje preporuke, je nedopustivo i neophodno je sistem dopuniti pravilom koje će se izvršiti u ovom slučaju. Većina sistema se dopunjava slijedećim ili sličnim pravilom:

```
IF  preporuka  = UNKNOWN
THEN  DISPLAY "Nažalost ne možemo dati preporuku".
              Preporuka = Not_Available.
```

Ovo pravilo prvo provjerava da li postoji vrijednost za promenljivu PREPORUKA. Da bi to provjerio potrebno je prethodno pokušati sa izvršavanjem svih pravila sistema koja dodjeljuju vrijednost preporuci. Ukoliko je preporuka

ostala bez dodjeljene vrijednosti, tada će se izvršiti THEN grana sistema, koja uglavnom izdaje poruku da nije aktivirano nijedno pravilo, odnosno da ne postoji preporuka i vrijednost preporuke postavlja na `Not_Available`.

Dodavanje teksta, koji obezbjeđuje komunikaciju, vrši se na slijedeći način. Da bi ES imao smisla i funkcionisao kako treba, potrebno je da korisnik unese ulazne parametre, odnosno podatke na osnovu kojih će se davati preporuka. Kada sistem prilikom izvršavanja naiđe na atribut sa nepoznatom vrijednošću, on izdaje pitanje "Koja je vrijednost atributa x?". Pitanja ovog tipa su prilično nejasna korisnicima, ukoliko korisnici nisu ujedno i tvorci ES, pa zbog toga može doći do pogrešnog unošenja ulaznih podataka, a samim tim i loših preporuka. Zbog toga treba sistem osposobiti da postavlja pitanja koja će mu omogućiti dobijanje pravilnih podataka od korisnika. Ova pitanja mogu za razmatrani primjer izgledati:

```
ASK tip: "Koji tip videorekordera vam je potreban?"
CHOICE tip: VHS,Beta
ASK kvalitet_slike: "Koliko vam je važan kvalitet slike?"
CHOICE kvatilet_slike: važan,nevažan
ASK zamrzava: "Da li je neophodno da video ima mogućnost
zamrzavanja slike?"
CHOICE zamrzava: da,ne
ASK pretraga: "Da li je neophodna sposobnost pretraživanja?"
CHOICE pretraga: da,ne
ASK cijena: "Koliku cijenu ste spremni da platite?"
CHOICE cijena: niska,srednja,visoka
```

Ovakva pitanja omogućavaju znatno lakšu (prije svega jasniju) komunikaciju sistema sa korisnicima. Ovde je neophodno naznačiti i to da nije postavljeno pitanje u vezi sa brojem glava. Ranije je uvedeno pravilo koje atributu glava dodeljuje vrijednost na osnovu toga koliko je bitan kvalitet slike.

Davanje višestrukih preporuka se može vršiti na slijedeći način. U realnim problemima rijetke su situacije kada je moguće ili poželjno da preporuka ima samo jednu vrijednost. U razmatranom primjeru prodavac će najčešće dati par mogućnosti koje odgovaraju potrebama kupaca, a ne samo jednu. Da bi se omogućio višestruki odgovor mora se za promjenljivu preporuka obezbjediti više memorijskih lokacija, na kojima će se čuvati sve vrijednosti preporuka koje odgovaraju ulaznim parametrima. To se postiže na sledeći način:

```
MULTIVALUED: preporuka
```

Sistem će sačuvati sve vrijednosti, koje se tokom izvršavanja mogu dodijeliti promjenljivoj PREPORUKA, i na kraju će sve ove vrijednosti biti dane kao preporuke. Ako se pažljivije posmatra razgovor između prodavca i kupca može se uočiti da prodavac uvijek daje više preporuka, ali na određeni način i pravi razliku između njih (npr. Ovaj video u potpunosti zadovoljava vaše zahtjeve, ali je cijena nešto viša i sl.). Da bi izgrađivani sistem mogao da daje slične odgovore, treba pravilima dodati faktore sigurnosti, te će **pravilo 1** izgledati :

```
If  type=VHS       AND
    Glave=4        AND
    Zamrzava=da    AND
    Pretraga=da    AND
    Cijena=niska
THEN  preporuka=VCX_1000    (CF=90),
```

što znači da VCX_1000 sa 90% sigurnosti zadovoljava potrebe kupca, što će sigurno poboljšati kvalitet preporuka koje daje sistem. Pomoću faktora sigurnosti se može korisnicima omogućiti da daju preciznije odgovore i na taj način poboljšaju kvalitet dobijenih odgovora. Tako, na primjer, se može omogućiti kupcu da umjesto na pitanje “Koliko vam je važan kvalitet slike?” ne odgovore sa "važan" ili "nevažan", nego koliki je stepen važnosti kvaliteta slike, npr:

```
”Koliko vam je važan kvalitet slike?”
    važan (CF=80)       nevažan
```

Poboljšanje pravila se vrši da bi se poboljšala komunikacija između korisnika i ES i da bi se poboljšala efikasnost sistema. Poboljšanje komunikacije se može postići, kao što je već pokazano, uvođenjem apstrakcije (umjesto koliko ima glava, kupci žele da sistem pita u vezi sa željenim kvalitetom slike). Takođe, moguće je uvođenje pravila koja olakšavaju korisnicima korištenje sistema i povećavaju preciznost ulaznih parametara, samim tim i kvalitet odgovora.

Za razmatrani primjer se može primjetiti da na pitanje koliku su cijenu kupci spremni da plate, kupci mogu da se odluče između: niske, srednje i visoke. Postavlja se pitanje koja je cijena visoka, koja je njena donja granica. Ta vrijednost varira i može se doći u situaciju da iako je kupac unijeo da je spreman da plati srednju cijenu, za preporuku dobije video sa, po njegovom, mišljenju visokom cijenom. Da ne bi dolazili u takve situacije, mogu se u sistem uvesti slijedeće promjene: dati korisniku da unese graničnu cijenu koju je on spreman da plati i

uvesti nekoliko pravila koja će unijetu cijenu svrstati u niske, srednje ili visoke cijene, a na osnovu pripadnosti intervalima koje je definisala osoba koja je izgrađivala ES, npr:

```
IF max < 300
THEN cijena = niska;

IF max >= 300 AND
   max <= 600
THEN cijena = srednja;

IF max > 600
THEN cijena = visoka.
```

U ES vrlo često se može pojaviti da različita pravila imaju nekoliko istih premisa, pa je moguće skratiti pisanje ovakvih pravila uvođenjem novog pravila koje će ispitivati iste premise, a koje će se pozivati u trenucima kada je potrebno odrediti istinitosnu vrednost tih premisa. Za razmatrani primjer se može predpostaviti da se za VCX_100, RECORD_MATE i SUPER_VIEWER ispituje da li je vrijednost atributa "zamrzava" i "pretraga" jednaka "da". Da bi se izbjeglo pisanje ovih premisa na tri mjesta, može se umjesto te dvije premise ispitivati da li je vrijednost promjenljive "efekti" jednaka "da". Promenljiva "efekti" dobijaće vrijednost u novom pravilu koje će izgledati:

```
IF zamrzava=da AND
   Pretraga=da
THEN efekti=da.
```

To će obezbjediti aktiviranje pravila samo u slučaju da su oba ogovora potvrdna.

Skraćenje koda se može postići i u slučajevima kada se u nekoliko pravila ispituje vrijednost istih atributa. Za razmatrani primer je očigledno da se u nekoliko pravila ispituju vrijednosti atributa: tip, kvalitet_slike, zamrzava, pretraga i cijena. U ovakvim situacijama poželjno je uvesti "*database rule"*, pravilo koje će se povezati sa bazom podataka u kojoj će se nalaziti određne vrijednosti atributa i preporuke. Za razmatrani primer pravilo će izgledati ovako:

```
If tip=[       ]     AND
   Glave=[       ]   AND
    Zamrzava=[    ]   AND
```

```
   Pretraga=[    ]   AND
   Cijena=[    ]
THEN  preporuka=[    ].
```

Izgled baze podataka je slijedeći:

Tip	Glave	Zamrzava	Pretraga	Cijena	Video
VHS	2	da	da	niska	VCX_1000
VHS	4	da	da	srednja	Record_Mate_99

Prilagođavanje načina zaključivanja i kontrole sistema

Ugrađene BCS tehnike zaključivanja, koje su ugrađene u mnogim alatima za kreiranje malih ES, su najčešće dovoljni za pravilno funkcionisanje sistema. Međutim, u nekim situacijama neophodno je napraviti male promjene. Alati ne dozvoljavaju velike promene koje se tiču tehnike zaključivanja, ali moguće je napraviti mala podešavanja i to koristeći:

- utvrđivanje redoslijeda ispitivanja istinitosne vrijednosti pravila i utvrđivanje praga istinitosti.

Uređivanje redoslijeda izvršavanja pravila

U složenijim sistemima postoji mogućnost dodjeljivanja prioriteta pravilima, odnosno vezivanje posebnih vrijednosti za pravila koja će određivati kojim redoslijedom će se ona aktivirati. Tako će sistem umjesto da ispituje pravila redom od početka prema kraju, prvo ispitivati ona sa najvišim prioritetom pa onda ona sa nižim.

Takođe, mogu se prvo pisati pravila za koja se vjeruje da postoji najveća vjerovatnoća aktiviranja ili se može urediti ispitivanje istinitosti premisa tako da se kritične postave na početak. Na primer, zna se da većina kupaca želi VHS video rekorder i srednji kvalitet slike, ali postoji velika raznolikost u tome koju cijenu su spremni da plate. Prvo će se ispitivati da li je cijena visoka, srednja ili niska, tako da se mogu na samom početku odbaciti pravila koja nemaju cijenu koja odgovara kupcu.

Uvođenje praga istinitosti

Prag istinitosti se definiše na nivou sistema, koji daju više preporuka, a ima funkciju da za preporuke daje samo one koje imaju zadovoljavajući stepen istinitosti.

U razmatranom primjeru se može uzeti da su kupci zainteresovani samo za one videorekordere koji sa 80% zadovoljavaju njihove potrebe. Na taj način se dobija manji broj preporuka, ali će sve biti zadovoljavajuće, jer su nebitne one preporuke koje sa samo npr. 10% zadovoljavaju potrebe kupaca. Veoma je bitno dobro odrediti ovaj koeficijent da ne bi došlo do odbacivanja dobrih preporuka. Koeficijent omogućava bolju efikasnost sistema jer prilikom ispitivanja nekog pravila, čim istinitost nekog pravila padne ispod zahtjevanog nivoa, prelazi se na slijedeće pravilo i na taj način se spriječava ispitivanje velikog broja pravila, čije bi se vrijednosti ispitivale kroz BCS.

14.4.5. Sistemi sa olančavanjem unaprijed

Sistemi sa olančavanjem unaprijed (*Forward Chaining Systems* - FCS), za razliku od sistema sa olančavanjem unazad, koji idu od cilja i kreću se unazad da bi se pronašle činjenice na osnovu kojih će biti moguće odrediti njegovu vrijednost, polaze od činjenica i kreću se ka cilju aktivirajući sva pravila za čije su aktiviranje ispunjeni uslovi zadani njihovim premisama. BCS su pogodni za kreiranje ES čiji je skup rješenja unapred određen i za njih je karakteristično da programer ne mora mnogo da brine o toku kojim će se vršiti zaključivanje, dok FCS omogućavaju da se kreiraju sistemi kod kojih skup rješenja nije unaprijed određen, ali za kreiranje ovakvih sistema neophodno je da programer usmjerava sistem da bi se došlo do ispravnih rješenja.

Da bi se ilustrovao FCS biće korišten OPS5 (*Official Production Systems 5*). Ovaj sistem vrši zaključivanje na slijedeći način (slika 14.6):

Prvo se pretraže sve činjenice koje se nalaze u radnoj zoni memorije, zatim se pronalaze pravila čije su premise istinite. Slijedeći korak je rangiranje premisa da bi se utvrdilo koje će se pravilo prvo izvršiti i ono se izvršava. Nakon toga ponovo se ide na prvi korak (utvrđivanje skupa pravila koja se mogu izvršiti), itd. Kruženje se nastavlja sve dok se ne dođe u situacija da ne postoji nijedno pravilo koje može da se aktivira.

Postoji nekoliko načina na koje se određuje koje će se pravilo izvršiti (tom prilikom se osmatraju samo pravila za čije su se aktiviranje stekli uslovi):

- pravila koja se tiču činjenica koje su posljednje nastale imaju viši prioritet,
- kompleksnija pravila imaju viši prioritet od jednostavnijih,
- pravilima se moraju dodjeliti prioriteti i u tom slučaju izvršava se pravilo najvišeg prioriteta.

Slika 14.6. Sistemi sa olančavanjem unaprijed.

Faze kreiranja FCS :

1. Definisanje problema;
2. Pisanje koda koji opisuje činjenice;
3. Pisanje početnog skupa pravila;
4. Definisanje načina zaustavljanja sistema;
5. Kontrola redoslijeda izvršavanja pravila;
6. Dalji razvoj sistema.

Definisanje problema

U ovom slučaju će se uzeti za primjer problem raspoređivanja novinskih članaka (slika 14.7.). Radi se o dnevnim novinama malog grada. Pokušaće se da se

napravi sistem koji će glavnog urednika osloboditi ovog dijela posla. Da se to uspije mora se detaljno ispitati urednika na koji način on određuje koji će se članci naći na naslovnoj strani. Ispostavlja se da on to radi na osnovu brojnih faktora, a to su:

- tip priče (da li obrađuje lokalnu ili nacionalnu temu ili je u pitanju neka senzacija),
- tema, i
- da li je članak dio feljtona.

Bitno je naglasiti da je broj članaka na naslovnoj strani ograničen.

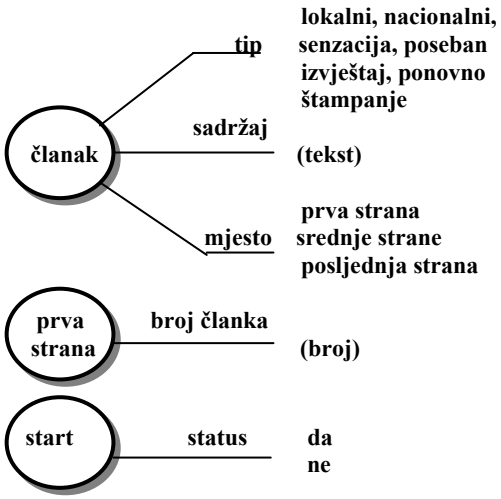
Iz ovoga slijedi da je potrebno definisati dva objekta i to: objekat koji se tiče članka i objekat koji će sadržavati podatke o tome koliko je članaka već raspoređeno na naslovnu stranu. Takođe, treba definisati i objekat "start", čija će se funkcija kasnije objasniti.

Pisanje koda koji sadrži činjenice

Osnovni pokretač FCS su podaci. Sve dok se podaci ne uvedu u sistem nemoguće je pokrenuti sistem. Snabdjevanje sistema podacima se može izvesti na različite načine, npr. sistem može dobijati potrebne podatke iz baze podataka, zatim putem ugrađenih senzora ili se podaci mogu dobijati njihovim unošenjem od strane korisnika. Da bi se aktivirao sistem i unijeli potrebni podaci, mora se napisati **aktivaciono previlo** (*sturtup rule*). Ovo pravilo se aktivira odmah nakon ulaska u sistem i zahtjeva neophodne podatke. Za razmatrani primjer ovo pravilo može izgledati ovako:

```
Rule Startup
IF start-status=da
THEN  pitanje_u_vezi_sa_tipom_priče
      pitanje_u_vezi_sa_sadržajem
      napravi_novi_objekat_članak
      ponovno_aktiviranje_početnog_pravila.
```

Kada se aktivira ovo pravilo, sistem uzima podatke o tipu i sadržaju članka i ponovo aktivira ovo pravilo da bi se omogućilo unošenje novog članaka.



Slika 14.7. Raspoređivanja novinskih članaka.

Pisanje početnog skupa pravila

Da bi se sistem mogao snabdjeti pravilima, neophodno je da se dodatno izvrši informisanje o načinu na koji urednik određuje da li će se neki članak naći na naslovnoj strani ili ne. Tokom razgovora dobijene su sledeće informacije:

- samo četiri članka se mogu naći na naslovnoj strani,
- specijalni izvještaji uvijek se postavljaju na naslovnu stranu,
- članci sa lokalnim temama imaju veću važnost od onih sa nacionalnim, sem ako tema nije od velikog značaja za građane tog grada (to su teme koje se tiču radničkih sindikata, s obzirom da najveći dio stanivnika tog grada radi u fabrici),
- senzacije se uvijek postavljaju na naslovnu stranu da bi privukle pažnju.

Na osnovu informacija koje su do sada dobijene, može se kreirati ovakav sistem:

```
LITERALIZE  članak
    Tip
    Sadržaj
    Mjesto
LITERALIZE   Naslovna_strana
    Broj_članaka
Rule Spec_izvještaj
```

```
IF članak-tip=specijalni_izvještaj THEN članak-mjesto=prva_strana
Rule Radnički
IF članak-tip=nacionalni AND članak-radnički THEN članak-
                                                    mjesto=prva_strana
Rule senzacija
IF članak-tip=senzacija THEN članak-mjesto=prva_strana
Rule  Lokalni
IF članak-tip=lokalni THEN članak—mjesto=prva_strana
LITERALIZE Start
    Status
                Rule Startup
IF start-status=da
THEN  pitanje_u_vezi_sa_tipom_priče
            pitanje_u_vezi_sa_sadržajem
            postavi_novi_članak
     ponovno_aktiviranje_početnog_pravila.
```

Definisanje načina zaustavljanja sistema

Ranije je objašnjeno da će se sistem zaustaviti u trenutku kada nijedno pravilo ne može da se aktivira. Iako je to najbolji način da se zaustavi sistem, nekada to nije moguće. U razmatranom primjeru, ukoliko se ne doda pravilo koje će zaustaviti unos podataka o novim člancima, sistem će imati mogućnost da neograničeni broj članaka postavi na naslovnu stranu, što nije moguće obzirom da je prostor naslovne strane ograničen. Uvođenjem ograničenja koje se odnosi na maksimalni broj članaka koji se mogu naći na naslovnoj strani, ujedno će se obezbjediti i način zaustavljanja sistema.

```
 Rule Brojač
IF članak-mjesto=prva_strana THEN increment prva_strana-brojčl.
Rule Stop
IF prva_strana-brojčl=4 THEN STOP.
```

Dodavanjem ovih pravila omogućeno je da se svaki put, kada je neki članak postavljen na naslovnu stranu, brojač broja članaka (koji je atribut objekta naslovna strana) poveća za jedan i da se u trenutku kada se četiri članka postave na naslovnu stranu sistem zaustavlja pomoću naredbe STOP. Na taj način korisnik će dobijati odgovore sve do trenutka u kome naslovna strana postaje popunjena i tada se sistem zaustavlja.

Kontrola redoslijeda izvršavanja pravila

U jednostavnim sistemima kao što je razmatrani primjer, posebna kontrola nije neophodna jer će sistem uzeti podatke, neka pravila će se izvršiti i sistem će

stati. Međutim, prilikom kreiranja ES vrlo često postoji potreba da se neka pravila izvršavaju poslije nekih drugih, ili samo pod određenim okolnostima, itd. Jedan od načina da se reguliše redoslijed izvršavanja pravila je definisanje okolnosti pod kojima je dozvoljeno aktiviranje pojedinih pravila. Tako se postiže da se neka pravila ne uzimaju u razmatranje sve dok se za to ne postignu uslovi. Ovakva pravila nisu rezervisana samo za FCS već se koriste i u BCS, s tom razlikom što u BCS ova pravila suže za povećanje efikasnosti i njihovo prisustvo nije neophodno, dok za FCS ova pravila mogu biti presudna za donošenje pravilnog zaključka.

U razmatranom primjeru problem bi se javio u slučaju da se prva četiri unijeta članka odnose na lokalna pitanja, a peti je senzacija. U tom slučaju, zbog popunjenosti prve strane, peti članak ne bi bio postavljen na naslovnu stranu iako urednik ne bi tako postupio. Da bi se spriječile ovakve situacije, uvode se neke promjene u sistem. Uvodi se novi objekat "Senzacije_unijete", koji će imati jedan atribut `status`. `Status` će imati vrijednost “ne” sve dok se ne unesu sve senzacije, a nakon toga će se promeniti u “da”. Takođe, u sva pravila koja postavljaju članke na naslovnu stranu dodaje se jedna premisa koja će ispitivati status unijetih senzacija, i blokiraće izvršavanje svih pravila sve dok sve senzacije ne budu unijete.

Problem redoslijeda aktiviranja pravila se može riješiti i vezivanjem broja za svako pravilo. Ovaj broj označava prioritet aktiviranja pravila. U takvim sistemima prvo se aktiviraju pravila koja imaju najviše prioritete. Sada se postavlja pitanje kako odrediti koja će pravila imati više prioritete. Prvo treba podjeliti skup mogućih ulaza na standardne ulaze i izuzetke. Izuzeci su oni ulazi za koje se odstupa od uobičajene procedure dobijanja savjeta. Kada se to odredi, viši prioriteti se dodjeljuju onim pravilima koja se tiču izuzetaka, da bi se obezbedilo da se njihovo preispitivanje i aktiviranje, ako su uslovi zadovoljeni, obavi prije nego što se na njih primjeni standardna procedura. Vrlo često je potrebno potpuno zabraniti izvršavanje standardne procedure, pa se u sva pravila nižeg prioriteta dodaju premise koje ispituju da li se izvršilo neko od pravila višeg prioriteta, ako jeste pravilo se ne aktivira, a u suprotnom se aktivira.

Ove zabrane se uvode samo u slučajevima kada je primjena različitih procedura isključiva, odnosno kada jedna isključuje mogućnost izvršavanja druge. U razmatranom primjeru izuzeci su članci koji se bave nacionalnom temom, ali su od važnosti i za grad u kome izlaze novine. Ovi članci bi po tome što su nacionalni trebali biti raspoređeni na srednje strane, međutim urednik ih stavlja na naslovnu, zbog njihove važnosti za grad. Zbog toga će se pravilu koje postavlja ovakve

članke dati viši prioritet od drugih, a u pravilo koje nacionalne članke postavlja na srednje strane dodati premisa koja ispituje da li je članak već postavljen na naslovnu stranu. Ovo je potrebno da bi se time zabranilo mijenjanje položaja članka, kada se dođe do pravila koje bi ga postavilo na srednje strane.

Dalji razvoj sistema

U FCS se mogu primjenjivati slična poboljšanja kao i kod BCS, s tim što kod uvođenja apstrakcija i dodavanja novih pravila treba biti vrlo oprezan. Svako dodavanje pravila zahtjeva detaljno preispitivanje na koji način to pravilo može uticati na tok zaključivanja. Apstrakcije se u BCS automatski koriste zahvaljujući načinu na koji se vrši zaključivanje. Međutim, u FCS se mora voditi računa da se apstraktna pravila moraju povezati sa postojećim pravilima da bi se omogućilo njihovo aktiviranje u pravo vrijeme.

Kod FCS je moguće koristiti i mrežni algoritam (*The Rete Algorithm*), koji u velikim sistemima znatno povećava efikasnist. On funkcioniše na taj način što povezuje pravila koja imaju neke zajedničke premise u mreže. Ove mreže omogućavaju da se prilikom pravljenja skupa pravila, koja mogu da se aktiviraju, provjeravaju iste premise samo jednom i ukoliko premisa ima netačnu vrijednost iz razmatranja se izbacuje grupa pravila. Ovako se postiže dvostruka ušteda vremena izvršavanja, jer se čim je premisa neistinita odbacuje grupa pravila, i na taj način se izbjegava provjeravanje ostalih premisa nekog pravila.

14.5. PRIMJER EKSPERTNOG SISTEMA

Opšte o Jess skeletnom sistemu (shell)

Java Expert System Shell (Jess) pripada grupi softverskih alata za razvoj ekspertnih sistema, odnosno jezika inženjeringa znanja opšte namjene. Jess je skeletni sistem (*shell*) ES i u potpunosti je napisan u Java programskom jeziku. Razvijen je u kompaniji *Distributed Computing System (DCS) - Sandia National Laboratories Livermore, CA, USA*. Veoma je brzo stekao popularnost, tako da se u mnogim radovima iz oblasti vještačke inteligencije može sresti kao primjer za razvoj Java aplikacija baziranih na znanju.

Kako autori Jess-a navode, Jess je kopija jezgra razvojnog alata za ES CLIPS, koji je počeo da poprima i prihvata uticaj Jave. Jess omogućava da aplikacije

(*apleti*), pisane u Javi, dobiju mogućnost da "odlučuju". Treba napomenuti da Jess ne duplira CLIPS u bukvalnom smislu, već samo njegovu suštinu.

Jess je kompatibilan sa svim verzijama Jave, počinjući od verzije 1.0.2., a najkompatibilniji je sa Javom 1.1, iako prilikom kompajliranja se može vidjeti upozorenje o nedozvoljenim metodama (*deprecated methods*). Međutim, to je cijena kompatibilnosti.

Jess je projekat u razvoju – svakim danom mu se dodaju nove opcije. Ima svoju sopstvenu semantiku, koja dosta podsjeća ma programski jezik LISP. Kao i svaki *shell* ES i Jess koristi istu logiku i principe. Osnovu svakog programa predstavljaju činjenice i pravila. Skup pravila gradi mehanizam zaključivanja ES, a upravo taj dio daje programu sposobnost odlučivanja. Skup činjenica gradi bazu znanja, bez koje napisana pravila ne bi imala nikakvu funkciju.

Opis problema

Potrebno je napraviti ES u Jess-u koji bi pružao podršku kod izbora *hard* diska. Kao ulaz, program bi prikupljao odgovore korisnika na postavljena pitanja. Kada se sakupi dovoljno informacija, program prikazuje prijedlog *hard* diska, koji odgovara danim zahtjevima. Podaci, koje daje ekspert o *hard* diskovima se čuvaju u posebnoj datoteci i prilikom svakog startovanja programa ti podaci se učitavaju u program kao činjenice. Radi se o običnoj tekstualnoj datoteci, tako da je moguće u svakom trenutku vršiti izmjene, kako bi se program prilagodio trenutnom stanju na tržištu *hard* diskova.

Analiza rješenja

Jess-ovom semantikom je napisan skup pravila koji predstavlja mehanizam zaključivanja ES. U posebnoj datoteci se čuvaju podaci o hard diskovima, koji predstavljaju bazu znanja. Slijedi listing programa u Jess-u.

Jess, the Java Expert System Shell
Copyright (C) 1998 E. J. Friedman Hill and the Sandia Corporation
Jess Version 5.1. 4/24/2000

```
;;;==============================================================
;;; Program za izbor hard diska
;;;
;;; Ovaj program pokušava da utvrdi vrstu hard diska koji je
```

```
;;; odgovarajući, prema danim odgovorima na postavljena pitanja
;;;=================================================================
; Definicija šablona. Na ovaj način se definiše nesortirana
; činjenica

(deftemplate cvor
       (slot rb-cvor)
       (slot proizvodjac)
          (slot tip)
          (slot da-grana)
          (slot ne-grana)
       (slot interfejs)
       (slot brzina)
          (slot kapacitet)
          (slot klasa)
          (slot pitanje)
       (slot odgovor))

; Inicijalizacija. Prilikom startovanja programa postavi
; tekuci-cvor na početak (korjen).
(defrule inicijalizuj
   (not (cvor (rb-cvor korjen)))
   =>
   (load-facts "diskovi.dat")
   (assert (tekuci-cvor korjen)))

; Ako je vrijednost polja "tip" jednaka "odluka", onda postavi
; pitanje koje je definisano u tekućem čvoru.
(defrule postavi-pitanje
   ?cvor <- (tekuci-cvor ?rb-cvor)
   (cvor (rb-cvor ?rb-cvor)
         (tip odluka)
         (pitanje ?pitanje))
   (not (odgovor ?))
   =>
   (printout t ?pitanje " (da ili ne) ")
   (assert (odgovor (read))))
; Ako je odgovor razlicit od "da" ili ""ne", ponisti odgovor.
(defrule pogresan-odgovor
   ?odgovor <- (odgovor ~da&~ne)
   =>
   (retract ?odgovor))

; Ako je odgovor "da", postavi tekuci-cvor na čvor koji pokazuje da
; grana.
(defrule idi-na-da-granu
   ?cvor <- (tekuci-cvor ?rb-cvor)
   (cvor (rb-cvor ?rb-cvor)
```

```
         (tip odluka)
         (da-grana ?da-grana))
   ?odgovor <- (odgovor da)
   =>
   (retract ?cvor ?odgovor)
   (assert (tekuci-cvor ?da-grana)))

; Ako je odgovor "ne", postavi tekuci-cvor na čvor koji pokazuje ne
; grana.
(defrule idi-na-ne-granu
   ?cvor <- (tekuci-cvor ?rb-cvor)
   (cvor (rb-cvor ?rb-cvor)
         (tip odluka)
         (ne-grana ?ne-grana))
   ?odgovor <- (odgovor ne)
   =>
   (retract ?cvor ?odgovor)
   (assert (tekuci-cvor ?ne-grana)))

; Ako je tip "odgovor", prikazi vrijednosti tekućeg čvora
; i pitaj da li ce se postupak ponoviti. Upiši odgovor.
; Eksplicitni poziv pravila "Pokusajte-ponovo".
(defrule provjeri-da-li-je-odgovor-zadovoljavajuci
   ?cvor <- (tekuci-cvor ?rb-cvor)
   (cvor (rb-cvor ?rb-cvor) (tip odgovor) (odgovor ?vrijednost1)
(proizvodjac ?vrijednost2)
    (interfejs ?vrijednost3) (brzina ?vrijednost4) (kapacitet
?vrijednost5) (klasa ?vrijednost6))
   (not (odgovor ?))
   =>
   (printout t crlf)
   (printout t "Odgovarajuci hard disk za vas je:"?vrijednost1 crlf)
   (printout t "Proizvodjac:                    "?vrijednost2 crlf)
   (printout t "Interfejs:                      "?vrijednost3 crlf)
   (printout t "Brzina:                         "?vrijednost4 crlf)
   (printout t "Kapacitet:                      "?vrijednost5 crlf)
   (printout t "Klasa:                       "?vrijednost6 crlf crlf)
   (printout t "Da li zelite da pokusate ponovo? ")
   (assert (odgovor (read)))
   (assert (Pokusajte-ponovo))
   (retract ?cvor))

; Ako je pozvano ovo pravilo, potvrdi poziv.
(defrule Pokusajte-ponovo
   (Pokusajte-ponovo)
   (not (pitanje ?))
   =>
   (assert (Pokusajte-ponovo)))
```

```
; Ako je pozvano pravilo Pokusajte-ponovo i ako je odgovor "da"
; postavi tekuci-cvor na korjen.
(defrule jos-jednom
   ?temp <- (Pokusajte-ponovo)
   ?odgovor <- (odgovor da)
   =>
   (printout t crlf crlf crlf)
   (retract ?temp ?odgovor)
   (assert (tekuci-cvor korjen)))

; Ako je pozvano pravilo Pokusajte-ponovo i ako je odgovor "ne"
; onda kraj rada.
(defrule necu-vise
   ?temp <- (Pokusajte-ponovo)
   ?odgovor <- (odgovor ne)
   =>
   (printout t "Kraj rada...")
   (retract ?temp ?odgovor))

(reset)
(run)
```

Uz svako pravilo je napisan komentar koji treba da bude dovoljan za razumjevanje značenja pravila, a osim toga i sam naziv pravila govori dosta o njegovoj funkciji.

Baza znanja se čuva u slijedećoj posebnoj datoteci.

```
(cvor (rb-cvor korjen) (tip odluka) (pitanje "Da li zelite da kupite
hard disk?") (da-grana cvor1) (ne-grana cvor2) (odgovor nil))

(cvor (rb-cvor cvor1) (tip odluka) (pitanje "Da li imate dovoljno
novca za najskuplji model?") (da-grana cvor3) (ne-grana cvor4)
(odgovor nil))
(cvor (rb-cvor cvor3) (tip odgovor) (pitanje nil) (da-grana nil)
(ne-grana nil) (odgovor "Atlas X")
      (proizvodjac Maxtor) (interfejs SCSI) (brzina "10 000 RPM")
(kapacitet "40 Gb") (klasa Visoka))

(cvor (rb-cvor cvor4) (tip odluka) (pitanje "Da li vam je bitna
brzina?") (da-grana cvor5) (ne-grana cvor6) (odgovor nil))

(cvor (rb-cvor cvor5) (tip odluka) (pitanje "Da li vam je potreban
veci kapacitet?") (da-grana cvor7) (ne-grana cvor8) (odgovor nil))

(cvor (rb-cvor cvor6) (tip odluka) (pitanje "Da li vam je potreban
veci kapacitet?") (da-grana cvor9) (ne-grana cvor10) (odgovor nil))
```

```
(cvor (rb-cvor cvor7) (tip odgovor) (pitanje nil) (da-grana nil)
(ne-grana nil) (odgovor "Fireball AS")

      (proizvodjac Quantum) (interfejs IDE) (brzina "7200 RPM")
(kapacitet "40 Gb") (klasa Srednja))

(cvor (rb-cvor cvor8) (tip odgovor) (pitanje nil) (da-grana nil)
(ne-grana nil) (odgovor "Fireball AS")
      (proizvodjac Quantum) (interfejs IDE) (brzina "7200 RPM")
(kapacitet "20 Gb") (klasa Srednja))

(cvor (rb-cvor cvor9) (tip odgovor) (pitanje nil) (da-grana nil)
(ne-grana nil) (odgovor "VL 40")
      (proizvodjac Maxtor) (interfejs IDE) (brzina "5600 RPM")
(kapacitet "40 Gb") (klasa Ekonomska))

(cvor (rb-cvor cvor10) (tip odgovor) (pitanje nil) (da-grana nil)
(ne-grana nil) (odgovor "VL 30")
      (proizvodjac Maxtor) (interfejs IDE) (brzina "5600 RPM")
(kapacitet "20 Gb") (klasa Ekonomska))
```

Za Jess, kao i za druge shell-ove ES, treba naglasiti da se program ne izvršava sekvencijalno. Rečeno je da svaki program sadrži skup pravila, koji predstavlja njegovu osnovu zaključivanja. Svako pravilo bi se moglo uporediti sa "If then else" strukturom i u kovencionalnim programskim jezicima. Da bi se neko pravilo aktiviralo, potrebno je da bude zadovoljen uslov na lijevoj strani pravila (if dio). Tada se izvršava desni dio pravila (then dio).

Da bi se bolje objasnio ovaj mehanizam, biće prikazan jedan primjer iz ovog programa. Na primjer, ako korisnik odgovori sa "da" na pitanje "Da li želite da kupite hard disk?" (pogledati listing datoteke, čvor korjen), tada je lijeva strana pravila "idi-na-da-granu" zadovoljena, i zato se ovo pravilo aktivira. Kao što se vidi u ovom pravilu, prilikom izvršavanja desne strane pravila, `tekuci-cvor` se postavlja na čvor koji se nalazi u polju "da-grana", a to je u ovom slučaju "`cvor1`". Na sličan način funkcionišu i ostala pravila.

Grafički interfejs

Grafički interfejs je napisan u Javi. Pri tome je korištena Jess-ova klasa *"Main.class"* koja u sebi sadrži *"embedded"* Jess. To znači da je u Javin kod inkorporiran Jess-ov pogon zaključivanja *(Rete)*, kao i *parser* koji je zadužen za čitanje Jess-ove sintakse. Naravno, tu je i mehanizam koji je zadužen za izuzetke i greške.

U datoteci "DiskPanel.java" sadržana je osnova programa koji koristi Jess-ovu *"main"* klasu. Implementiran je jedan *Panel* na koji su postavljena dva dugmeta (Da i Ne), kao i komponenta *TextArea.* Listing sa komentarima prikazan je u nastavku.

```
package jess;

import java.awt.*;
import jess.awt.TextReader;
import jess.awt.TextAreaWriter;
import java.awt.event.*;
import java.util.*;
import java.io.*;

/**
 * Konzolni graficki interfejs.
 * Ova klasa predstavlja Panel koji kao ulaz prihvata kliktaje misem
 * korisnika na dugmad "Da"
 * i "Ne". Rezultat obradjen u Rete pogonu prikazuje na ekranu (u
 * TextArea).
 * Koristi TextReader i TextAreaWriter klase da pretvori tekstualne
 * podatke u ulazno\izlazne tokove.
 * Disk i DiskAplet klase prikazuju instance ove klase **/

public class DiskPanel extends Panel implements Serializable
{
  // Promjenljiva za predstavljanje izlaza
  private TextAreaWriter m_taw;

  // Promjenljiva za hvatanje ulaza
  private TextReader m_in;

  // Pogon ekspertnog sistema
  Rete m_rete;

  /**
   * Postavi panel; prikaci Rete objekat za
   * ulazne i izlazne komponente
   */

  public DiskPanel(Rete r)
  {
    m_rete = r;

    // Podesavanje elemenata grafickog interfejsa
```

```
final TextArea ta = new TextArea(10,40);
ta.setEditable(false);
Button bDa = new Button("Da");
Button bNe = new Button("Ne");
Panel p = new Panel();
p.setLayout(new GridLayout(1,2));

// Definisi  I/O tokove
m_taw = new TextAreaWriter(ta);
m_in = new TextReader(false);

// Postavi elemente grafickog interfejsa
setLayout(new BorderLayout());
add("Center", ta);
p.add(bDa);
p.add(bNe);
add("South", p);

// Funkcija koja hvata dogadjaje dugmeta "Da"
bDa.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent ae)
    {
      synchronized (ta)
        {
          try
            {
           m_taw.write("da");
              m_taw.write('\n');
              m_taw.flush();
            }

          catch (IOException ioe)
            {
            }
        }
           m_in.appendText("da" + "\n");
    }
});

// Funkcija koja hvata dogadjaje dugmeta "Ne"
bNe.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent ae)
    {
    synchronized (ta)
        {
          try
            {
           m_taw.write("ne");
```

```
                    m_taw.write('\n');
                    m_taw.flush();
                  }
                catch (IOException ioe)
                  {
                  }
              }
                 m_in.appendText("ne" + "\n");
          }
      });

      // Podesi Rete objekat (ulazne i izlazne rutere)
      PrintWriter pw = new PrintWriter(m_taw, true);
      r.addInputRouter("t", m_in, true);
      r.addOutputRouter("t", pw);
      r.addInputRouter("WSTDIN", m_in, true);
      r.addOutputRouter("WSTDOUT", pw);
      r.addOutputRouter("WSTDERR", pw);
  }

}
```

U datotekama “Disk.java” i “DiskAplet.java” korištene su instance klase “DiskPanel”. Klasa disk predstavlja implementaciju aplikacije, a klasa DiskAplet predstavlja implementaciju apleta. Slijede njihovi listinzi.

Disk.java:

```
package jess;

import java.io.*;
import java.awt.*;
import java.awt.event.*;

/**
*********************************************************************
**
 * Graficka konzola zasnovana na jess-ovoj klasi Console koja je
 * prilagodjena specijalno za ovaj primjer ali i za druge primjere
 * gde su odgovori korisnika iskljucivo Da i Ne.

*********************************************************************
** */

public class Disk extends Frame implements Serializable
```

```
{
  DiskPanel m_panel;
  Rete m_rete;

  // Konstruktor koji samo postavlja naslov frame-a i alocira novi
Rete objekat
  public Disk(String title)
  {
    this(title, new Rete());
  }

  // Kreiraj konzolu koristeci vec postojeci Rete objekat

  public Disk(String title, Rete r)
  {
    super(title);
    m_rete = r;
    m_panel = new DiskPanel(r);

    add("Center",m_panel);
    Button b;
    add("South", b = new Button("Kraj rada i izlaz iz programa"));
    b.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent ae) {
        dispose();
        System.exit(0);
      }
    });
    validate();
    setSize(500,300);
    show();
  }

  /**
   * Proslijedi niz argumenata prema objektu jess.Main povezanom sa
   * ovom konzolom i pozovi metodu Main.execute().
   */

  public void execute(String [] argv)
  {
    Main m = new Main();
    m.initialize(argv, m_rete);
    m_panel.setFocus();
    while (true)
      m.execute(true);
  }
```

```
  // Main funkcija
    public static void main(String[] argv)
  {
    new Disk("Jess Konzola - Izbor hard diska").execute(argv);
  }
}
```

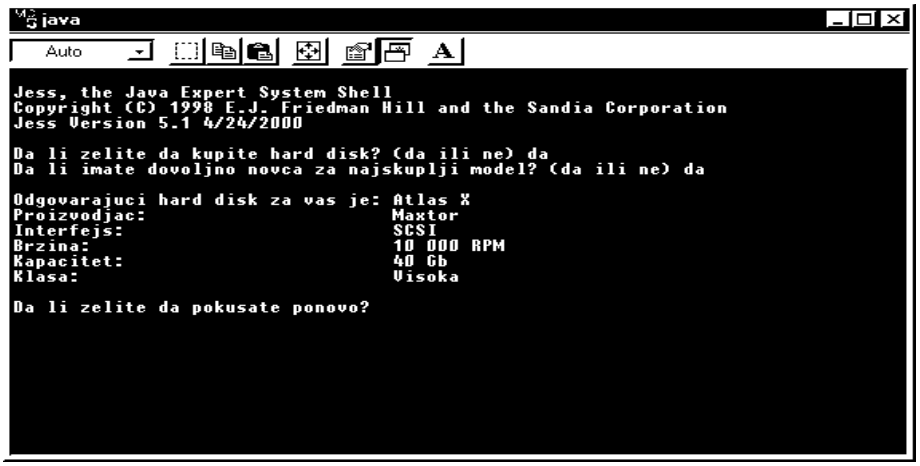
Diskaplet.java:

```
package jess;

import java.awt.*;
import java.applet.*;
import java.util.*;
import java.io.*;


/**
 *
 *    Aplet koji koristi DiskPanel. Moze da posluzi kao osnova bilo
 *    kojeg programa napisanog u jess-u koji kao ulaz prihvata
 *    iskljucivo odgovore Da i Ne korisnika.
 *
 *    Parametri apleta:
    •  INPUT: ako je zadan ovaj parametar, Rete pogonu se
       proslijedjuje datoteka
 *    na koji on ukazuje i procesira se u batch modu.
 */

public class DiskAplet extends Applet implements Runnable,
Serializable
{
  // Disk panel
  private DiskPanel m_panel;
  // Pogon
  private Rete m_rete;
  // Thread u kojem se vrsi parsiranje
  private Thread m_thread;
  // Main objekat koji upravlja Rete pogonom
  private Main m_main;

  // inicijalizacija
  public void init()
  {
    setLayout(new BorderLayout());
    m_rete = new Rete(this);
```

```
    m_panel = new DiskPanel(m_rete);
    add("Center", m_panel);
    add("South", new Label());

    String [] argv = new String[] {};
    // Obradi parametre apleta
    String appParam = getParameter("INPUT");
    if (appParam != null)
      argv = new String[] {appParam};

    m_main = new Main();
    m_main.initialize(argv, m_rete);
  }

  public synchronized void run()
  {
    do
      {
        try
          {
            m_panel.setFocus();
            while (true)
              m_main.execute(true);
          }
        catch (Throwable t)
          {
            m_thread = null;
          }
      }
    while (m_thread != null);
  }

  public void start()
  {
    if (m_thread == null)
      {
        m_thread = new Thread(this);
        m_thread.start();
      }
  }

  public void stop()
  {
    m_thread = null;
  }

}
```

Korisničko uputstvo

Program se može startovati ili sa Jess-ove konzole preko *batch* komande ili kao zasebna aplikacija, sa sopstvenim grafičkim interfejsom, preko datoteke "disk.bat" (što je mnogo lakše). Korisnik odgovara na postavljena pitanja sa "da" ili "ne". Na kraju, kao rezultat njegovih odgovora prikazuje se odgovarajući *hard* disk.

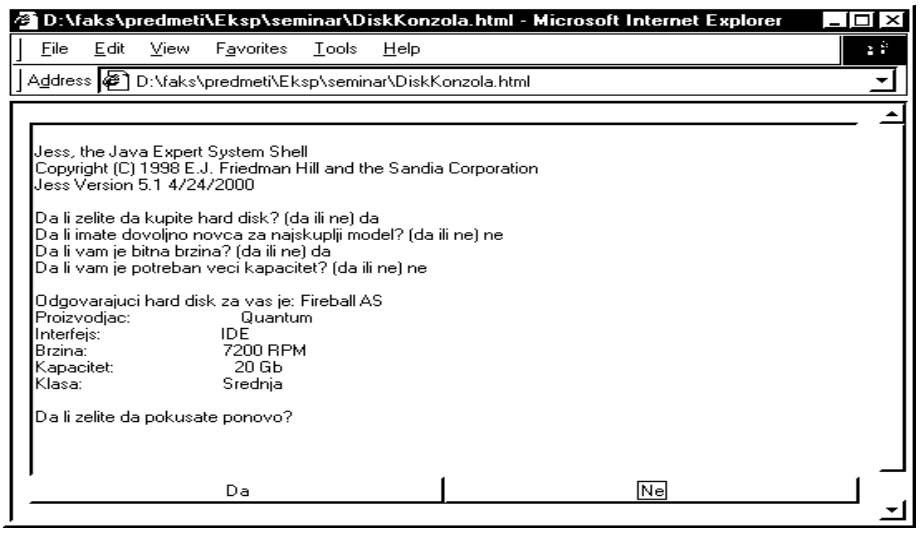


Slika 14.8. Primjer ekrana kada se program izvršava u Jess-ovoj konzoli.

Prvo pitanje koje se postavlja korisniku je: "Da li želite da kupite *hard* disk?". U zavisnosti od odgovora korisnika program izvršava odgovarajuće akcije. Ako je odgovor "ne", program prekida sa radom. Ako je odgovor "da", postavlja se slijedeće pitanje: "Da li imate dovoljno novca za najskuplji model?". Ako je opet odgovor "da", prikazuje se najskuplji i najbolji *hard* disk. Ako je odgovor "ne", postavlja se slijedeće pitanje i tako redom. Kada se prikaže na ekranu *hard* disk, postavlja se pitanje korisniku da li želi da pokuša ponovo. Ako je odgovor potvrdan, postupak se ponavlja još jednom.

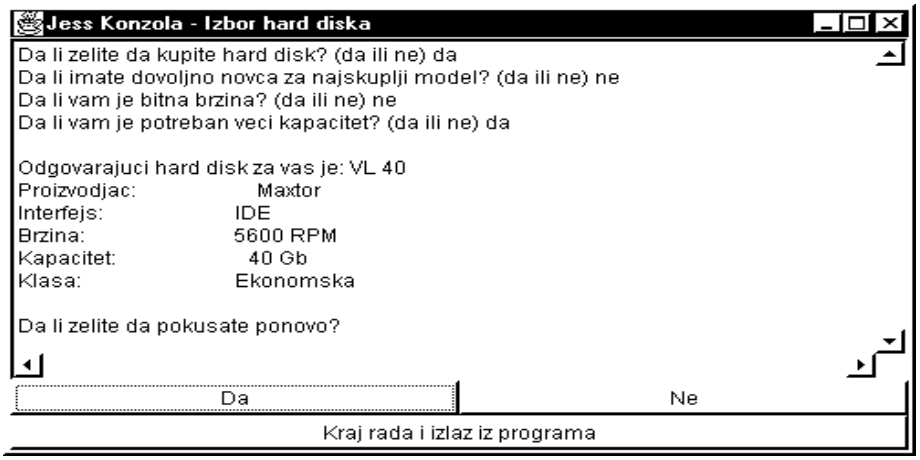
Grafički interfejs pruža malo udobniji rad. Naime, korisnik ne mora da unosi odgovore preko tastature, već svoje odgovore unosi pritiskom mišem na odgovarajuću dugmad.

Program se može izvršavati i kao aplet u bilo kojem *Internet browser*-u, koji ima ugrađenu *Java Virtual Machine*, pokretanjem datoteke "*DiskKonzola.html*".



Slika 14.9. Primjer izvršavanja programa kao zasebne aplikacije.

Ovaj jednostavni program predstavlja samo jedan primjer, čija je namjera da bliže objasni način funkcionisanja Jess-a, kao i da prikaže neke od mogućnosti ovog alata.



Slika 14.10. Primjer izvršavanja programa kao *aplet*-a.

LITERATURA

1. Aiello, N., Bock, C., Nii, H. P. and White, W., "AGE reference manual", *Knowledge Systems Laboratory,* Memo HPP-81-24, 1981.
2. Anand, S. S., "Designing a Kernel for Data Mining", *IEEE Expert*, 1997, pp. 65-74.
3. Battle, D., Vose, M. D., "Isomorphisms of genetic algorithms", *Artificial Intelligence 60*, 1993.
4 Berkovic, I., "Elementi veštačke inteligencije kroz primere i zadatke", *Tehnički fakultet "M. Pupin"*, Zrenjanin, 1999.
5. Buchanan, B. G. and Shortliffe, E. H., "Rule-Based Expert Systems: The MYCIN Experiments oJ the Stanford Heuristic Programming Project", *Addison-Wesley*, Reading, MA, 1984.
6. Bojić, D., Velašević, D., Misić V., "Zbirka zadataka iz ekspertnih sistema", *ETF*, Beograd, 1996.
7. Bradshaw, J. M. (Ed.), "Software Agents", *MIT Press,* Cambridge, MS, 1997.
8. Colmerauer, A., Kanoui, H., Van Caneghem, M., "Prolog, basis theoriques ad developpment actual", *Technique at Science Informatiques*, 1983.
9. Čubrilo, M., "Matematička logika za ekspertne sisteme", *Informator*, Zagreb, 1989.
10. Davis, R. and Lenat, D., "Knowledge-Based Systems in Artificial Intelligence: AM and TEIRESIAS", *McGraw-Hill*, New York, 1982.
11. DENDRAL, ES., *Stanford University*, 1965.
12. De Jong , K. A., "Genetic Algorithms are NOT Function Optimizers", *Foundation of Genetic Algorithms 2*, Whitley Darrel, Morgan Kaufmann (Ed.), *Publishers*, San Mateo, California, pp. 5-19, 1992.
13. Devedžic, V., "Ekspertni sistemi za rad u realnom vremenu", *Institut "M. Pupin"*, Beograd, 1994.
14. Devedzic, V., "Understanding Ontological Engineering", *Communications of the ACM*, Vol. 45, No. 4, 2002, pp. 136-144.
15. Devedzic, V., "Software Patterns", in: Chang, S., K. (ed.), "Handbook of Software Engineering and Knowledge Engineering Vol.2 – Emerging Technologies", *World Scientific Publishing Co.,* Singapore, 2002, pp. 645-671.
16. Devedzic, V., "Software Project Management", in: Chang, S., K. (ed.), "Handbook of Software Engineering and Knowledge Engineering Vol.2 – Emerging Technologies", *World Scientific Publishing Co.,* Singapore, 2002, pp. 419-446.

17. Devedzic, V., "Knowlede Discovery and Data Mining in Databases", in: Chang, S. K. (ed.), "Handbook of Software Engineering and Knowledge Engineering Vol. 1 - Fundamentals", *World Scientific Publishing Co.,* Singapore, 2001, pp. 615-637.
18. Devedzic, V., "Knowledge Modeling - State of the Art", *Integrated Computer-Aided Engineering,* Vol.8, No.3, 2001, pp. 257-281.
19. Devedžić, V., "Inteligentni informacioni sistemi", *DIGIT/FON*, Beograd, 2000.
20. Downing, T. E., Moss, S., Pahl-Wostl, C., "Understanding Climate Policy Using Participatory Agent-Based Cocial Simulation", "Multi-Agent-Based Simulation", *Second International Workshop, MABS 2000,* Springer, Boston, 2000, pp. 199-213.
21. Djuric, M., Devedzic, V., "I-promise - intelligent protective material selection", *Expert Systems With Applications*, Vol.23, No.3, 2002, pp. 219-227.
22. Durkin, J., "Expert Systems: design and development", *Macmillan Publishing Company*, New York, 1994.
23. Edgar, A., et al., "A Generic Service Access Network Platform", *Proceedings of the World Telecommunications Congress ISS '95,* Vol. 1, pp. 442-446.
24. Etzioni, O., Weld, D. S., "Intelligent Agents on the Internet: Fact, Fiction, and Forecast", *IEEE Expert*, 1995, pp. 44-49.
25. Erman, L. D., Hayes-Roth, F., Lesser, V. R. and Reddy, D. R., "The HearsayII speech-understanding system: Integrated knowledge to resolve Uncertainty," *Computers Surveys,*Vol. 12, pp. 213-253, 1980.
26. Evans, C., Gollancz, V., "The Mighty Micro", London, 1979.
27. Fayyad, U., Piatetsky-Shapiro, G., Smyth, P., Uthurusamy, R., "Advances in Knowledge Discovery and Data Mining", *MIT Press*, Cambridge, MA, 1996.
28. Friedman, E. J., "Jess, the Java Expert System Shell Copyright (C)" 1998.
29. Geneserth, M. R. and Ketchpel, S. P., "Software Agents", *Communications of the ACM*, 37(7), 1994.
30. Goldberg E. D., "Genetic Alghorithms in Search, Optimization & Machine Learning", *Addison-Wesley Publishing Company Inc.,* Reading, MA, 1989.
31. Grand, S., Cliff, D., "Creatures: Entertainment Software Agents with Artificial Life" (39-57), "Autonomous Agents and Multi-Agents systems", *Kluwer Academic Publishers*, Boston, 1998.
32. Grosof, B. N., "Building Comercial Agents: An IBM Research Perspective", 1997.
33. Grosof, B. N., Foulger, D., A,. "Globenet and RAISE: Intelligent Agents for Networked Newsgroups and Customer Service Support", 1995.
34. Hill and the Sandia Corporation Jess Version 5.1. 4/24/2000.

35. Holland, J. H., “Adaptation in Natural and Artificial Systems”, *The University of Michigan Press*, Ann Arbor, 1975.
36. Hotomski, P., “Sistemi veštačke inteligencije”, *Tehnički fakultet "M. Pupin",* Zrenjanin, 1995.
37. Hotomski, P., “Sistema automatičeskogo dokazateljstva teorem s rezolucijej, indukcijej i simetrijej”, *Proc. of the Conf. Algebra and Logic, Institute of mathematics*, Novi Sad, 1985., pp. 55-61.
38. Hotomski, P., Pevac I., “Matematički i programski problemi veštačke inteligencije u oblasti automatskog dokazivanja teorema”, *Naučna knjiga*, Beograd, 1991. (II izdanje).
39. Hotomski, P., “Deduktivnij podhod k automatičeskomu porožđeniju kombinatornih raspoloženi”, *Proc. of the VI Conf. on Logic and Comp. Science LIRA 92*, Novi Sad, 1992., pp. 35-42
40. Hotomski, P., “Pet godina razvoja i primene programskog sistema "DEDUC" za kreiranje rasporeda časova”, Časopis "PC u obrazovanju" br.1/2, 1996., Tehnički fakultet ¨M. Pupin¨ Zrenjanin 1996., str. 20-27.
41. Hotomski, P., “Automatsko generisanje kombinatornih rasporeda pod restriktivnim uslovima - komponenta informacionog sistema u obrazovanju”, *VI Međunarodna konferencija "Informatika u obrazovanju i nove informacione tehnologije"*, Apatin, 1996, pp. 80-83.
42. Hotomski, P., “Deduktivni prilaz optimizaciji kombinatornih rasporeda pod restriktivnim uslovima”, *XXIV Simpozijum o operacionim istraživanjima SIMOPIS 97*, Bečići, 1997., pp. 363-367.
43. Hotomski, P., “Automatsko generisanje kombinatornih rasporeda pomoću sistema DEDUC”, *Časopis Telekomunikacije br.2, 1997., Zajednica JPTT*, Beograd, pp. 29-35.
44. Huhns, M., Singh, M. (Ed.), “Readings in Agents”, *Morgan Kaufmann*, San Mateo, CA, 1998.
45. http://www.research.ibm.com/iagents/paps/rc20835.pdf
46. http://www.research.ibm.com/iagents/paps/rc20226.pdf
47. htpp:/www.iis.ns.ac.yu.htlm
48. http://iis.fon.bg.ac.yu/
49. Jennings, N. R., Sycara, K., Wooldridge, M., “A roadmap of agent research and development” (7-38), “Autonomous Agents and Multi-Agents systems”, *Kluwer Academic Publishers*, Boston, 1998.
50. Kim, W., “Introduction to object-oriented databases”, *Massachusetts Institute of Technology, Cambridge*, MA, 1992.

51. Kosko, B., "Neural Networks and Fuzzy Systems", *Prentice-Hall, Englewood Cliffs*., NJ, 1992.
52. Lindsay, R. K., Buchanan, B. G., Feigenbaum, E. A. and Lederberg, J., "Application of Artificial Intelligence for Chemistry: The DENDRAL Project", *McGraw-Hill*, New York, 1980.
53. Maes, P. (ed.), "Designing Autonomous Agents", *MIT/Elsevier Press*, London, 1994.
54. McCarthy**,** J., "LISP*", Artificial Intelligence Laboratory, Stanford University,* 1979.
55. Miller, J. P., "The Design on Intelligent Agents – A Lazered Approach", *Springer-Verlag,* NY, 1996.
56. Moukas, A., Pattie, M., "AMALTHAEA: An Evolving Multi-Agent Information Filtering and Discovery" (59-88), "Autonomous Agents and Multi-Agents Systems for the WWW", *Kluwer Academic Publishers*, Boston, 1998.
57. MYCIN., The book Rule-Based Expert Sytem: The MYCIN Experiment at the Stanford Heuristic Programming Project, 1972.
58. Nedovic, Lj., Devedzic, V., "Expert Systems in Finance - A Cross-Section of the Field", *Expert Systems With Applications*, Vol.23, No.1, 2002, pp. 49-66.
59. Parsaye, K., Chignell, M., Khoshafian, S., Wong, H., "Intelligent Databases", *AI Expert*, 1990, pp. 38-47.
60. Piatetsky-Shapiro, G., Frawley, W., "Knowledge Discovery in Databases", *MIT Press*, Cambridge, MA, 1991.
61. Poliščuk, E. J., "Prilog metodologiji razvoja sistema za podršku odlučivanju i ekspertnih sistema*", doktorska disertacija, FOI, Sveučilište u Zagrebu*, 1991.
62. Poliščuk, E. J., "Informatička organizacija ili "organizacija budućnosti"", *PRAKSA,* Beograd, 1992.
63. Poliščuk, E. J., "Razvoj sistema za podršku odlučivanju*", Publikacija Banjalučkog univerziteta,* Bihać, 1992.
64. Poliščuk, E. J., "Neki aspekti razvoja sistema za podršku odlučivanju", *III međunarodni simpozij informacijske i komunikacijske tehnologije u uredskom poslovanju '92 "OFFICE AUTOMATION",* Varaždin, 1992.
65. Poliščuk, E. J., 'Informatičko obrazovanje "inženjera budućnosti"', *V međunarodna konferencija "Informatika u obrazovanju i nove informacione tehnologje,* Novi Sad, 1995.
66. Poliščuk, E. J., "Ekspertni sistemi u mikrografiji", *JISA INFO*, Beograd, 1996.
67. Poliščuk, E. J., "Neki problemi korišćenja informacione tehnologije*", VI međunarodna konferencija "Informatika u obrazovanju i nove informacione tehnologije",* Apatin, 1996.

68. Poliščuk, E. J., "Ekspertni sistem kao element podrške odlučivanju u mikrografiji", *I naučno – stručni skup: Informacione tehnologije – sadašnjost i budućnost IT '96,* Žabljak, 1996.
69. Poliščuk, E. J., "Uvod u ekspertne sisteme", *skripta, ETF,* Podgorica, 1997.
70. Poliščuk, E. J., "O nekim problemima i pravcima razvoja ekspertnih sistema", *VII međunarodna konferencija "Informatika u obrazovanju i nove informacione tehnologije",* Novi Sad, 1997.
71. Poliščuk, E. J., "Sistemi automatskog učenja", *VIII međunarodna konferencija "Informatika u obrazovanju, kvalitet i nove informacione tehnologije",* Beograd, 1998.
72. Poliščuk, E. J., "Adaptivni sistemi mašinskog učenja", *ETF Journal of ELECTRICAL ENGINEERING, University of Montenegro,* Podgorica, 2001.
73. Poliščuk, E. J., "Metode adaptivnog mačinskog učenja", *VIII festival informatičkih dostignuća INFOFEST 2001,* Budva, 2001.
74. Poliščuk, E. J., "Baze podataka", *Informatička literatura JEP,* Podgorica, 2003.
75. Poliščuk, E. J., "Informacioni sistemi", *Informatička literatura JEP,* Podgorica, 2004.
76. Poliščuk, E. J., "The Analysis Of Experimental Results Of Machine Learning Approach", *In Proceedings of the 19th International Symposium on Information and Communication Technologies* , Sarajevo, 2003.
77. Poliščuk, E. J., "The Analysis of Experimental Results of Reinforcement Learning Systems", *The Computer Science Journal of Moldova, MATH ©*, Vol.10, No.2 (29), 2002, pp. 143-168.
78. Poliščuk, E. J., "Adaptive Machine Reinforcement Learning", *Schedae Informaticae, Jagiellonian University Krakow (since 1364) ,V*ol. 11, 2002, pp. 57-74.
79. Poliščuk, E. J., "The Analysis Of Experimental Results Of Machine Learning Approach", *Journal of Information and Organizational Sciences, IEE Inspec®, University of Zagreb (since 1669), V*ol. 27, No. 1, 2003, pp. 29-42.
80. Poliščuk, E. J., "The Machine Learning Approach: Analysis of Experimental Results", *Journal of Applied Computer Science, IEE* Inspec®, Vol 11, No.1, 2003, pp. 61–76.
81. Poliščuk E. J.: *Automatic Theorem Proving: Situation Management and Decision Making,* Emerging Technologies, Robotics and Control System, International Society for Advanced Research, Vol. 2, pp. 154-167, 2007.
82. Reynolds,D., Jagannathan, G., "Stohastic modelling of Genetic Algorithms", *Artificial Intelligence 82*, 1996.

83. Riecken, D., "Special Issue on Intelligent Agents", *Communications of the ACM, July,* 1994.
84. Ristić, Z., Balaban, N., Bošnjak, Z., "Ekspertni sistemi", *"Savremena administracija"*, Beograd, 1993.
85. Rosenschein, J. and Genesereth, M. R., "Communication and cooperation", *Knowledge Systems Laboratory*, Memo HPP-84-5, 1984.
86. Subašić, P., "Fazi logika i neuronske mreže", *Tehnička kniga,* Beograd, 1997.
87. Šaletić, Z. D., Velašević, M. D., "Rasplinuti sistemi i rasplinuti ekspertski sistemi zasnovani na pravilim", *INFO SCIENCE,* Beograd, 2000.
88. Tošić, D., Protić R., "PROLOG kroz primere", *Tehnička knjiga*, Beograd, 1990.
89. Turajlić, R. S., "Fazi sistemi i fazi upravljanje*"*, *JISA INFO,* Beograd, 1996.
90. Turing, A., "Computing Machinery and Intelligence", *Mind 49*, 1950, pp 433-460.
91. Van Melle, W., "*System Aids in Constructing Conultation Programs: EMYCIN", UMI Research Press*, Ann Arbor, MI, 1982.
92. Weizenbaum, J., "ELIZA" , MIT, 1960.
93. Zadeh, L. A., "Fuzzy sets", *Inf. & Contr,* 1965, Vol. 8, pp. 338-353.
94. Zadeh, L. A., "Outline of a new approach to the analysis of comlex systems, and decisions processes", *IEEE Trans. Syst.,Man & Cyb.,*1973, Vol.1,pp.28-44.