VISOKA POSLOVNA ŠKOLA
STRUKOVNIH STUDIJA
ČAČAK

SEMINARSKI RAD

Osnoveve veštačke inteligencije I

Mentor:_____
Profesor: _____

Student: _____
Br.Indeksa: _____

Sadržaj

1 Poglavlje 1 - veštačka inteligencija, istorijski razvoj i uvod

Istorijski gledano, još je Lajbnic pominjao ,,univerzalnu algebru" kojom bi se svekolika ljudska znanja (uključujući i etiku i metafiziku) obuhvatila jednog dana u jedinstvenom deduktivnom sistemu. Frege, jedan od osnivača moderne simboličke logike, je predložio notacioni sistem za mehaničko rezonovanje. Čarls Bebidž 1834. konstruiše mehaničku ,,analitičku mašinu" koja računa i štampa neke matematičke proračune, imao je nameru da napravi i mašinu za igranje šaha. Tek napretkom informatike i tehnologije 1940-tih i 1950-tih nastaju prvi rezultati koji spadaju u domen VI. McCulloch i Walter Pits još 1943. godine predlaž prvi model veštačke neuronske mreže, a 1951. godine Marvin Minsky i Dave Edmonds prave prvi elektronski računar (SNARC, sa 3000 vakuumskih cevi) zasnovan na takvoj mreži (u okviru doktorske disertacije za čiju je komisiju bilo diskutabilno da li se može takav rad svrstati u matematiku - član komisije, Jonh von Neumann, izjavio je da će biti jednog dana ako već nije - ironično, upravo je Minski teorijskim rezultatima ,,pokopao" nešto kasnije ovu oblast za narednih par decenija). Nekoliko istraživača na Dartmut koledžu 1956. g. (Dartmouth College) učestvuje u seminaru koji organizuje McCarthy na temu VI (koji je prvi predložio upravo taj naziv za tu oblast, a poznat je i kao otac LISP-a koji je bio značajan alat u VI, a i dans je u izvesnom obimu) gde su Allen Newell i Herbert Simon prezentovali ,,Logic Theorist" - prvi program za automatsko dokazivanje teorema (Bertrand Rasel je bio zadovoljan rezultatima, pogotovu jednim generisanim dokazom koji je bio kraći nego jedan naveden u ,,Principia Mathematica" - svi su ipak bili svesni da su to samo početni rezultati), gde su učestvovali i Minski, Šenon, Semjuel, Solomonov i drugi. Veštačka inteligencija beleži prve uspehe akademske prirode kao što su prvi program za igranje šaha (Claude Shannonn, 1955, poznat i kao otac savremene statističke informacione teorije - zajedno sa Alanom Tjuringom - ovo se smatra jednim od najbitnijih presudnih rezultata u istoriji VI) ili dama (Arthur Samuel, 1963), automatsko dokazivanje teorema (pomenuti ,,Logic Theorist", Simon i Newell), kao i ambiciozan pokušaj ostvarivanja opšteg sistema za rešavanje problema GPS (General Problem Solver - Newell, 1960). Sajmon i Njuel daju 1963. *pretpostavku sistema fizičkih simbola* koja je uspešno naknadno osporavana, ali je važan deo istorije: svaki sistem (ljudski ili veštački) koji se smatra inteligentnim mora da radi tako da uzima fizičke šablone (simbole,

,,physical patterns"), kombinuje ih u strukture (izraze) i rukuje njima (koristeći procese) da bi proizveo nove strukture (izraze). Suština zablude, čije su posledice razvejane tek pojavom prvih ekspertnih sistema, jeste nedostatak domenskog znanja potrebnog inteligentnim sistemima umesto isključivog oslanjanja na sintaksnu analizu.

1.1 Definicija i oblasti bavljenja VI

Definicija 1.1 *Bilo koji problem za koji ne postoji* efikasno *algoritamsko rešenje je problem veštačke Inteligencije (VI).*

Ovu definiciju i jedan dobar deo strukture ovog teksta dugujem [JL] i [GN]. Ovakva definicija daje praktičniji i bolji pogled na pojam VI od uobičajene Makartijeve (MIT) definicije da je to oblast računarstva čiji je cilj rezonovanje na računaru na način koji je sličan ljudskom. Iako ova potonja definicija daje intuitivniji i u nekom smislu precizniji opis oblasti kojima se bavi VI, ona vodi ka ozbiljnim ontološkim pitanjima i problemima: imamo sliku o ljudima kao svesnim, slobodim, umnim i racionalnim bićima, a u isto vreme ljudi su agenti u fizičkom svetu ustrojenom u naučnom smislu deterministički i materijalno, lišenom smisla (mehanicistički i partikularno, gde čestice nemaju svest). Kako to onda da u takvom svetu postoje ljudi kao bića sa svešću i namerom ? Da li je moguće um preneti iz jednog organskog bića u punom smislu u neki fizički sistem zasnovan isključivo na postojećoj ili budućoj informacionoj tehnologiji ? Ovo su samo neka pitanja koje Džon Serl vešto postavlja u [JS], povezuje ih i odgovara na njih, a na poslednje pitanje uglavnom daje negativan odgovor. Međutim, to ne znači da su oblasti VI rekle sve što imaju (daleko od toga), naprotiv - te oblasti već su ostvarile sjajne rezultate, i u mnogome pomogle kao alati i ljudima i nauci. Kognitivne nauke, ali i one koje su u vezi sa njom a nisu u direktnoj vezi sa VI i računarstvom, mogu pomoći istraživanjima u oblasti VI, ali često se dešava i obratno. Na kraju, ne postoji potpuno dobra definicija VI jer ne postoji ni potpuno dobra definicija inteligencije i pojmova u vezi nje.

Efikasnost se može jasno, pa čak i formalno definisati komplesnošću algoritma - npr. polinomijalna kompleksnost (i NP) je dobra i poželjna (u smislu efikasnosti) - prvi teorijski rezultati nastaju tek početkom 1970-tih godina (Steven Cook, Richard Karp). VI se može smatrati eksperimentalnom naukom u kojoj se eksperimenti vrše na računaru u okviru modela koji su

izraženi programima i čijim se testiranjem i dorađivanjem postižu neki modeli ljudske inteligencije (kojima se ova npr. može bolje razumeti - ne postoji realno očekivanje niti cilj da VI zameni ljudsku inteligenciju osim u nekim specifičnim oblastima ljudske delatnosti i primenama računarstva čije granice pomera VI). Pod algoritmom obično podrazumevamo uređen konačan niz precizno definisanih operacija koje mogu biti izvršene (na računaru). Ali to ne znači da će biti izvršene u nekom ,,razumnom" vremenu - postoji matematički formalizam kojim se ovo može preciznije obuhvatiti i definisati kao što su to npr. Tjuringove mašine i slični formalizmi (Alan Tjuring, inače je jedan od prvih informatičara i jedan od prvih istraživača VI na digitalnim računarima, ustanovio je prvi praktičan test programa VI u kome razdvojeni učestvuju ljudi, programi i ispitivači) . Na primer, ne postoji ,,klasičan" algoritam za igranje šaha koji bi mogao da se koristi upotrebljivo jer bi algoritmu koji bi ispitao sve moguće pozicije za svaki potez bile potrebni barem milioni godina i na najbržim postojećim računarskim sistemima.

Osnovne dve osobine oblasti kojima se bavi VI (bez osvrta na neke određene dobro definisane metode):

1. tiču se obrade simboličkih podataka (nasuprot tradicionalnoj numeričkoj obradi kao primeni računara)

2. uvek uključuju nekakav element izbora: nedeterminizam kojim se kaže da ne postoji algoritam na osnovu koga bi izabrali neku opciju u skupu mogućih za datu situaciju

Računari danas sve bolje rukuju multimedijalnim sadržajima ali je to rukovanje i njihova obrada još uvek daleko od onoga što ljudska čula i svest pružaju u opažanju i razumevanju sveta. Zato prva osobina nudi osnovu rešavanja prvog problema na koji se nailazi u VI - sakupljanje informacija. Postoje dobro ustanovljeni formalizmi i u matematici i u igrama koji čine simboličke (nenumeričke) podatke posebno značajnim. S druge strane, prepoznavanje i obrada (pattern recognition) zvučnih i vizuelnih signala predstavlja izazov za sebe, ali je posebno zanimljivo razumevanje i zaključivanje koje sledi nakon toga.

1.2 Kratak uspon i pad, zatim renesansa

Nakon početnog entuzijazma nastalog pod uticajem tehnološkog razvoja računara do početka 70-tih brzo se došlo do zaključka o pravoj težini problema VI, npr. da za automatsko prevođenje nisu dovoljni samo sintaksna analiza, rečnik i dobri algoritmi pretrage već i znanje o semantici jezika, pa i opšte znanje i iskustvo (poznat je primer o programima za automatsko prevođenje, kada se između bar dva jezika nekoliko puta ista rečenica prevede - anegdota kaže da je od engleske poslovice ,,Daleko od očiju, od srca" tako dobijen ,,nevidljivi idiot"). Takvi problemi su narušili nerealno idealnu sliku o VI i označili period njene krize, o čemu npr. piše Dreyfys 1972. i kasnije Lighthill 1973. čiji preterano kritičan izveštaj utiče na sudbine mnogih istraživačkih projekata (problem nije bio u VI već u zahtevima od tada mlade oblasti). Prvi uspešni ekspertni sistemi kao što je to bio DENDRAL i MYCIN (Edward Feigenbaum) predstavljaju početak izlaska iz te krize. Osnovu izlaska čini i posmatranje domenskog (deklarativnog) znanja inteligentnih sistema, gde su važni bili uopšteni alati kao što su frejmovi (okviri, početkom 1970-tih) Minskog kojima se to znanje formalizuje ali i praktično koristi. Minski je bio poznat i kao tvorac *mirkosvetova* kao probnih formalnih poligona za rešvanje problema VI (koje je davao svojim studentima), kakav je bio i *Svet blokova* (sistem SHRDLU koji je razvio Terry Winograd 1971. je bio veoma uspešan u rešavanju njegovih problema, ali je bio potpuno neprimenjiv za bilo kakvo uopštavanje zbog nedostatka domenskog znanja, koje je u tom slučaju bilo ,,utkano" u sintaksnu analizu tog sistema). Negde 1972. Alain Colmerauer je razvio Prolog, sledeći jezik VI (posle LISP-a) koji pored ostalih klasifikacija spada u deklarativne programske jezike i jedan je od najznačajnijih alata VI. Od 1980-tih godina nakon prvih pokušaja industrijalizacije VI (i računari 5. generacije, pored jezika) i eksplozije PC industrije počinje zreliji period razvoja VI sa akcentom na primeni postojećih teorija, novim metodama i teorijskoj potvrdi novih metoda - neki rezultati u oblasti prepoznavanja govora ili računarske vizije su tako bliži realnom svetu (naspram teorijskih mikrosvetova) i praktičnoj upotrebi. Mašinsko učenje koristi dostignuća matematičke statistike, ali i nove metode čija je primena već sada nezamenljiva. Ideja inteligentnog samostalnog entiteta ili agenta koji kontinualno funkcioniše u stvarnom svetu sa usađenom inteligencijom (situated intelligence) takođe postaje sve aktuelnija (predlog uopštenog rešenja kroz SOAR arhitekturu kao primer - Newell, Rosenbloom, John Laird, ili life-long learning, Tom M. Mitchell). Ideja deklarativnog znanja razvojem WWW-a postaje sve

aktuelnija idejom semantičkog web-a (Tim Berners-Lee, koji je ujedno i idejni tvorac web-a zasnovanog na HTTP i HTML, regulisanog W3C), gde pojam web ontologije prirodno nasleđuje okvire Marvina Minskog.

1.3 Oblasti

Inteligentnim sistemima nazivamo programske sisteme i druge praktične rezultate VI, odnosno posledicu jedne od neformalnih definicija VI (kao oblasti računarstva koja je posvećena inteligentnim sistemima): entiteti koji imaju sposobnost inteligentnog ponašanja koje srećemo kod ljudi. Međutim, ovakav pristup definisanju ima dodatnu slabost - u oblastima kakve su mašinsko učenje ili ekspertni sistemi, javlja se potreba za rešavanjem problema kojima treba prevazići neki ljudski nedostatak. Na primer, velika količina znanja kojim je teško upravljati čak i uz pomoć većeg broja ljudi - formalna definicija u uvodnom poglavlju ne ostavlja nedoumice u tom pogledu, ali ne objašnjava potrebu i način na koji ljudi žele da upravljaju znanjem. Oblasti veštačke inteligencije sa nekim podoblastima i tipovima inteligentnih sistema (neke od njih ili bar najveći deo biće objašnjen u ovom tekstu detaljnije) jesu:

- ekspertni sistemi - sistemi kojima se čuva i eksploatiše znanje na način sličan ljudskim ekspertima
- mašinsko učenje - metode klasifikacije, otkrivanja znanja (Data Mining), dobavljanje informacija (information retrieval), indukcija, prepoznavanje šablona (pattern recognition)
- igre - teorija igara i primene, šah ...
- predstavljanje znanja - jezici predstavljanja znanja, strukture
- rasuđivanje (rezonovanje) - pretraživanje, različite metode rasuđivanja (od Aristotelovih silogizama do danas) i automatsko dokazivanje teorema, formalno automatsko dokazivanje ispravnosti
- obrada prirodnog jezika - mašinsko prevođenje, razumevanje i analiza dijaloga, automatsko ispravljanje i generisanje
- agenti - multi-agentski sistemi i primene, softboti, web mining
- govor - problemi prepoznavanje, generisanje i razumevanja govora, prepoznavanje govornika i autentifikacija

- vizija - problemi interpretacije i razumevanja slika
- računska inteligencija (soft computing) - fazi logika i sistemi, neuronske mreže, genetski algoritmi, primene u automatskom odlučivanju i upravljanju
- robotika
- kognitivne nauke (multidisciplinarna oblast u kojoj se prepliću VI i psihologija, filozofija, neurologija, biologija, lingvistika, antropologija): uverenja, kreativnost, emocije, pamćenje, percepcija, priroda inteligencije i svesti, usađena sposobnost saznavanja (kognicija), i mnoge ,,kombinacije" kakva je i evolutivna psihogija (uticaj biološke strukture organizma na psihu i obratno - jedinke kao eksponenti DNK)
- edukacija - inteligentni tutorski sistemi
- inteligentni interfejsi - modeliranje korisnika, dijaloga i objašnjenja, veza sa tehnologijom
- filozofski aspekti, etičke i društvene implikacije

Naredno poglavlje ima takođe uvodni karakter, gde se pre svega ilustruje značaj pojmova problema i rešenja, znanja i njegovog reprezentovanja. Poglavlja 8, 12 i 13 (i donekle 7) izlaze izvan okvira ovog teksta, ali predstavljaju dobar nagoveštaj daljih saznanja u vezi ostalih osnovnih pojmova VI.

2 Poglavlje 2 - Predstavljanje problema

2.1 Pojam problema

Problema postajemo svesni kada želimo da nešto postignemo ali ne znamo kako da do toga dođemo, ne znamo njegovo rešenje (ili postupak, algoritam kojim bismo došli do toga). Problem uvek podrazumeva i neko rešenje ili potragu za rešenjem. Za razliku od problema u svakodnevnom životu, problemi školskog tipa su obično precizno opisani zajedno sa ponuđenim podacima neophodnim za njegovo rešavanje, pogotovu matematički problemi ili igre. U realnom svetu problem može biti opisan prirodnim jezikom (čije razumevanje u smislu interpretacije predstavlja jedan od osnovnih primera problema VI) koji sa tačke gledišta rešavanja problema ima barem četiri ozbiljna nedostatka: nekompletnost (bez konteksta lako može doći do nesporazuma u razgovoru), redundantnost, nejasnoća tj. višesmislenost i gramatička neispravnost. Potrebno je zato najpre naći formu zapisa problema tako da se ovi nedostaci izbegnu. Primer za to su zatvoreni izrazi:

$$x \in X : K(x)$$

gde se pod tim podrazumeva da za dati skup X (čime je implicitno data struktura skupa sa svojim operacijama) treba naći sve njegove elemente x za koje je ispunjen skup ograničenja $K(x)$. Ovo obično vodi ka postupku traženja prvog rešenja koje smanjuje dalji prostor rešenja koji treba pretražiti i dozvoljenim transformacijama se tako iterativno dolazi do kona'čnog zatvorenog izraza koji daje direktno rešenje. Variajante ovakvih problema mogu biti rešavanje slagalice (gde je lako navesti sve dozvoljene transoformacije od početnog stanja do završnog) ili dokazivanje jednačine gde izbor i broj transformacija uopšte nije jednostavno naći.

2.2 Rešavanje problema, uopšteni koraci

Uobičajen redosled koraka u rešavanju problema mogao bi biti:

1. Pročitaj ili upamti problem s razumevanjem
2. Izvedi neposredne zaključke o tome ako je moguće (time se može doći do nedostajućih podataka i elegantnije formulacije)
3. 'Poigraj' se sa dobijenim zaključcima i upamćenim činjenicama (veoma bitan korak ljudima)

4. Porazmisli o svemu, ostavi da stvari sazru
5. Potraži bolju formulaciju, uoči zatvoren izraz
6. Nađi delimično rešenje i vrati se na 2. korak ili nađi konačno rešenje
7. Proveri ispravnost rešenja, potraži moguće uopštenje

Postupak koji je predložio George Polya (1956) se može uporediti sa prethodnim:

1. Shvati problem (podaci, nepoznate, uslovi, crtež, itd.)
2. Napravi plan (veza podataka i nepoznatih, potproblemi i ranije rešavani problemi, drugačija formulacija, i sl.)
3. Sprovedi plan (da li su svi koraci jasni i da li se mogu potkrepiti dokazima ?)
4. Prouči dobijeno rešenje (da li je ispravno, da li se može primeniti na neke druge probleme)

Dakle, inteligentno rešavanje problema pretpostavlja stvaranje plana za njegovo rešavanja.

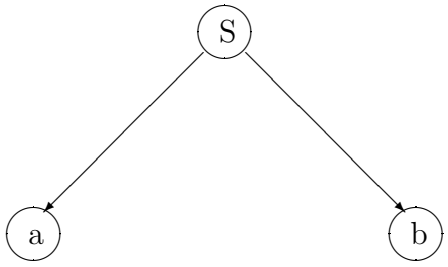
2.3 Notacije, načini reprezentovanja

Kao što je pomenuto, način zapisivanja i reprezentovanja problema je veoma bitan deo rešavanja jer pojednostavljuje i često ubrzava rešavanje. Ljudski um se u svakodnevnom životu rukovodi apstraktnim modelima što započinje u različitim slojevima od samih čula sve do psihičkih procesa. Koriste se nizovi simbola i šeme za zapis teksta, muzike ili matematičkih izraza koji su suštinski grafičkog karaktera. Matematičke notacije su polazna osnova za izgrađivanje formalizama koji su nam neophodni za proučavanje ovakvih modela.

Svi notacioni sistemi uopšteno se sastoje od simbola objekata i simbola operatora (arnost - koliko objekata napadaju) koji predstavljaju moguće akcije nad objektima. Linearne notacije predstavljaju niske ovakvih simbola. Pravilno formirane niske definisane prema redosledu objekata i operatora daju izraze koji mogu imati vrednost (primeri: infiksni, prefiksni (poljski) i postfiskni zapisi aritmetičkih izraza, izraza teorije skupova ili logičkih izraza).

Korišćenjem grafova tj. drveta kao specijalne vrste grafova koji su primer ,,dvodimenzione" notacije se ovakvi izrazi mogu takođe zapisati (čvorovi su operatori, listovi objekti, a redosledom obilaska i čitanja drveta se može dobiti linearan zapis i obratno).

Ovo nas dovodi do zapisa koji su upotrebljivi u algoritmima i programima (liste su značajne zbog toga posebno, pogotovu u nekim programskim jezicima kao što je LISP) i takođe se mogu pokazati ekvivalentnim nekim prethodnim strukturama. Liste se mogu posmatrati kao uređene trojke (S,L,R) gde je S ,,glava" ili operator, a L i R su takođe liste ili ,,rep"(leva i desna ,,sestra", redosled kao kod obrnute poljske notacije). Naravno, niz ovakvih trojki je u memoriji indeksiran i počinje sa pomenutom trojkom, dok su L i R zapravo pokazivači na članove niza, i listovi imaju objekte umesto operatora (listovi imaju ,,L=R=null", null je oznaka prazne liste). Transformacije nad ovakvim strukturama kao što su zamena podliste drugom listom ili brisanje podliste - u uobičajenoj infiksnoj notaciji se svode na zamenu ili brisanje podterma ili grane na drvetu.

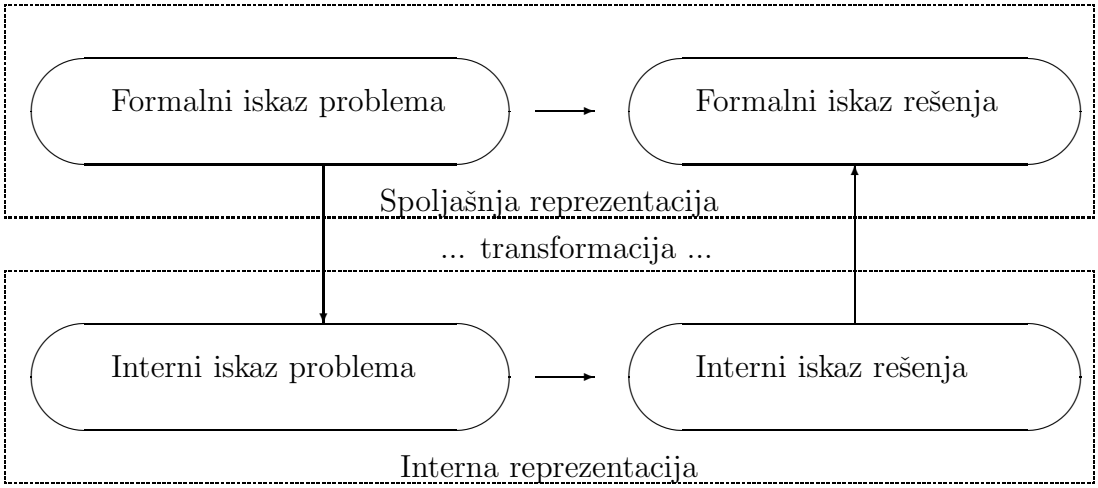


2.4 Modeli grafova u VI

Grafovi su značajan alat za reprezentaciju objekata i znanja kao dva bitna nivoa prisutna kako u matematici (npr. objekti, izrazi naspram relacija, teorema) tako i drugim oblastima. U veštačkoj inteligenciji se tako mogu lakše razmatrati problemi mašinskog dokazivanja teorema, problemi vizije i govora, automatskog rešavanja problema i razumevanja prirodnog jezika. Upotreba grafova je i u tome od značaja kako ljudima, tako i programima i rešenjima VI u smislu modela grafova. U jednom od narednih odeljaka

biće kratko navedene formalne definicije grafova i njihovih osobina, a već načeta tema reprezentacije znanja (i strukture znanja) biće dalje pojašnjena. Grafovi takođe predstavljaju i jedan od bitnih spojeva različitih formalno definisanih problema i njihovih praktičnih rešenja u VI.

2.4.1 Reprezentovanje znanja u automatskom rešavanju problema



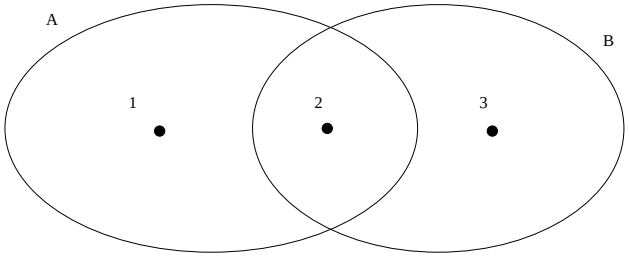
(ilustracija odnosa formalnog i internog reprezentovanja problema i rešenja)

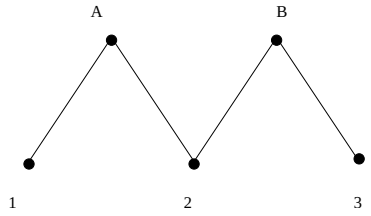
Interna reprezentacija je zapravo prostor rešenja u kome se od nekog početnog stanja nekim postupkom rešavanja stiže do rešenja (prostor rešenja je definisani podskup prostora stanja).

Relacija (time i graf kao način prikaza relacije) može imati osobine koje je čine relacijom ekvivalencije (R,S,T) ili relacijom poretka. Takvi grafovi su korisni u algoritmima za mašinsko dokazivanje gde se heuristike standardnih algoritama za rad sa grafovima svode na heuristike u dokazivanju. Graf može biti od pomoći kao vizuelno i intuitivno pomagalo čoveku i ekvivalentna struktura u programu, ili može biti od pomoći kao struktura koja opisuje postupke u rešavanju problema i odnose među objektima (noseći njihovu sintaksu i semantiku). Heuristike (kao prečice u postupku rešavanja nekog problema koje daju efikasnije algoritme) se porede npr. s internim znanjem nekog matematičara kada rešava neki problem i uopšte su veoma značajne za VI, kao i razdvajanje eksterne („sintaksnog") reprezentovanja znanja i

internog („semantičkog"). Nekoliko interesantnih primera / skica automatskog dokaza teorema u geometriji i teoriji skupova:

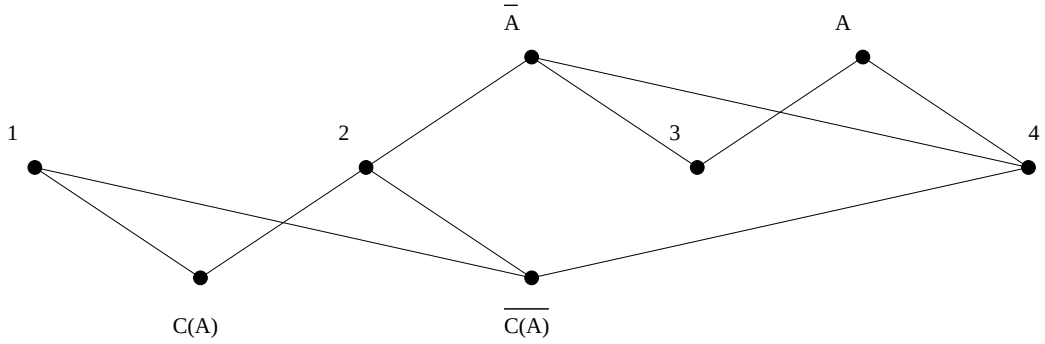
Primer 1 - polazeći od topološkog reprezentovanja skupova (Merialdo, 1979) umesto Venovih dijagrama:





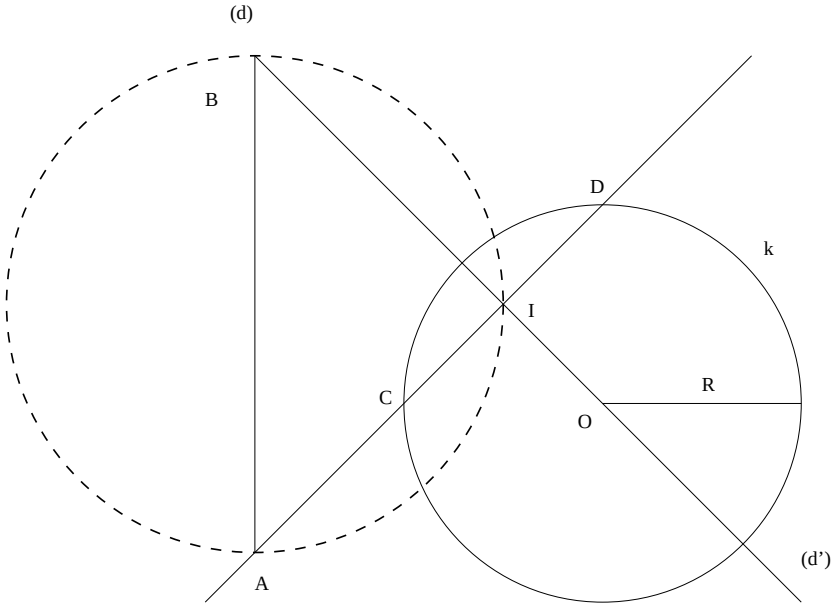
mogu se dobiti pregledniji grafovi kao pomoć u rešavanju. Primer: ako je $\overline{A}$ zatvorenje skupa A (najmanji zatvoren skup koji sadrži A) i $A^* = \overline{A} \wedge \overline{C(A)}$ njegova granica, važi:

Teorema 1 $A^* = (\overline{A} - A) \vee (\overline{C(A)} \wedge A)$



Na grafu se ,,vidi" da su ,,atomski elementi"(zapravo reprezenti klasa elemenata datih osobina) granice - atomi 2 i 4, a tvrđenje sledi pošto isto tako važi da je $\overline{A} - A =$ atom 2 i $\overline{C(A)} \wedge A =$ atom 4.

Primer 2 Traženje geometrijskog mesta tačaka ili konstrukcija lenjirom i šestarom su takođe moguća primena upotrebe grafova u VI. Primer problema: ako je dat krug k i dve tačke A i B van tog kruga konstruisati pravu p kroz A tako da su njeni preseci C i D sa k jednako udaljeni od B.



Program može kao i čovek da pođe od pretpostavke da gotovo rešenje postoji kao u dijagramu iznad, konstruišući dva bitna objekta - pravu d i tačku I koja polovi CD. Elementi ovakvog dijagrama se predstavljaju u tabeli čiji su redovi oblika: naziv objekta (d), tip (prava), stepen slobode (1), predstavnici (B, I - pripadaju d) - ovakav red se nalazi na početku rešavanja problema. Stepen slobode govori koliko je element određen (npr. prava je određena dvema različitim tačkama, B je zadata a I tek treba odrediti). Prava d' potpuno zadata tačkama O i B koincidira sa d pošto su trouglovi BCD i OCD jednakostranični, OI i BI bisektrise CD i time ista (jedinstvena) prava, što daje red u pomenutoj tabeli: d prava 0 d', B, I, O. Važi da I pripada OB. Pošto je ugao AIB prav to znači da I pripada krugu nad prečnikom

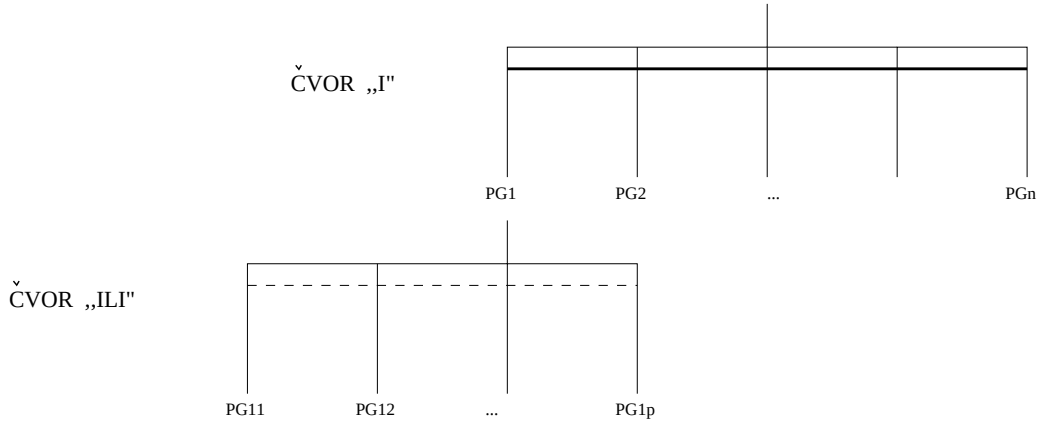
AB koji je time potpuno određen a time i I kao njegov presek sa d. Odatle sledi da je p potpuno određena tačkama I i A.

Primer 3 Primer iz teorije skupova sa preslikavanjima i kompozicijama koji je uspešno rešio program DATE (Pastre, 1978) kao i još oko 150 teorema u vezi teorije skupova, preslikavanja, kongruencija i kardinalnostu. Primer teoreme koju DATE može da dokaže:

Teorema 2 *Ako su* $f : A \to B$, $g : B \to C$, $h : C \to A$ *tri preslikavanja i ako za dva od bilo koja od tri preslikavanja* $k_1 = h \circ g \circ f$, $k_2 = f \circ h \circ g$, $k_3 = g \circ f \circ h$ *važi da su surjekcije (NA) i da je treće injekcija (1-1), onda su sva tri preslikavanja* f, g *i* h *bijekcije.*

2.4.2 Grafičko reprezentovanje znanja u automatskom rešavanju problema

O automatskom rešavanju i razvijanju grafa može se govoriti i kroz primer I - ILI drveta traženja rešenja (npr. logički iskaz se razvije i od korena ,,diskutuje" prema listovima):



Čvorovi ,,ILI" se odnose na disjunkcije a čvorovi ,,I" se odnose na konjunkcije. Svaka podgrana (PG) predstavlja podcilj u pretrazi koji se može rešavati posebnim metodama koje opet mogu proizvesti svoje podgrane (podciljeve). Tada je veoma poželjno svesti takvo drvo na jednu granu (da li zbog načina pretraživanja ili zbog samog problema nedeterminizma prisutnog u VI to je teško dostižno). Razbijanje problema na podprobleme kao i drveta na poddrveće je korisna osobina i jednog i drugog - primeri (neki detalji su u

[JL]): PRET (rešavanje trigonometrijskih problema, Grandbastien, 1974), PARI (problemi celobrojne aritmetike, Burgoin, 1978) ili automatsko dokazivanje teorema iskaznog računa (Pitrat, 1966). Primer je i upoređivanje problema optimizacije u operacionim istraživanjima gde se iskazi oblika ,,proces prethodi procesu uz potrebna vremena za izvršenje" i ,,procesi se nemogu paralelno izvršiti" rešavaju algoritimima optimizacije putanje kroz graf i bojenja grafa, redom.

2.4.3 Grafička reprezentacija i prirodni jezik

Veza sintakse i semantike jezika je presudna kod problema razumevanja prirodnog jezika u cilju automatskog prevođenja. Prvi pokušaji koji su se oslanjali samo na sintaksu i prevod reči u reč su se pokazali nedovoljnim, već je potrebno u rečniku dati nekakvo semantičko značenje na osnovu kojeg program gradi semantičku konstrukciju dela teksta, kao i dosta pragmatičnog ljudskog znanja o svetu uopšte. Kontekstno slobodne gramatke Noama Čomskog s pravilima transformacija (npr. LR1) su morale biti nadograđene gramatikama u kojima bi jezički automat u nekom trenutku analize se vraćao na prethodne nivoe obilaska drveta transformacija i razrešio neke semantičke probleme da bi nastavio analizu (rekurzivne gramatike višeg reda).

Proširene Mreže Prelaska (Augmented Transition Networks - ATN, Woods, 1975) mogu se koristiti za sintaksno-semantičku analizu i predstavljaju grafove čiji su čvorovi ili reči, ili semantičke familije ili podgrafovi (tako da je ovakva reprezentacija suštinski rekurzivna). Ono što je interesantno za njih je da jezički procesor koji ovako tekst analizira gradi na osnovu semantičkih pravila internu formu koja se zove *semantička mreža* (semantic network) i koja predstavlja rezultat obrade rečenice prirodnog jezika (u [JL] dat je primer vezan za analizu električnih kola). Grafovi su korisni i kao formalni oblik reprezentacije podataka i znanja, ali i kao intuitivan ljudski alat za rešavanje problema.

2.5 Traženje pravog reprezentovanja

Traženje pravog načina reprezentovanja problema je skoro uvek najznačijniji korak u rešavanju problema - primer problema: dva crna skakača s jedne strane i dva bela s druge na `3x3` šahovskoj tabli treba da zamene mesta u što manjem broju koraka. Kada se uoči da su pozicije skakača elementi skupa

ciklične strukture reda 8 onda se lako uoči i rešenje.

2.6 Programski jezici PROLOG i LISP

PROLOG i LISP su jedni od najznačajnijih programskih jezika bitnih za VI. Njihov značaj i primene u veštačkoj inteligenicji (pored istorijskih) su brojne. Lista kao osnovna struktura podataka u LISP-u je ujedno i način reprezentovanja znanja (sam program je takođe lista pa se npr. u nekim genetskima algoritmima koristi kao struktura koja se rekombinuje; mnogi sistemi kao što je to npr. CLIPS su inspirisani ovim jezikom, itd). Jednom usvojeno iskustvo sa ovakvom strukturom i funkcionalnom filozofijom programiranja se lako prenosi i u druge pristupe programiranju i VI. Njegova sintaksa se može vrlo jednostavno definisati

```
<S-izraz> := <atom> | <lista>
<lista> := ( <telo> )
<telo> := <nil>| <S-izraz> | <S-izraz> <telo>
<atom> := niska alfanumerika bez razmaka (standardni identifikator)
i specijalnih znakova.
```

gde je *nil* je prazna lista bez elemenata. Prvi atom liste je funkcija koja može biti ugrađena (npr. funkcija `QUOTE` koja zaustavlja evaluaciju `L` za `(QUOTE L)` ili skraćeno `'L`) a evaluacija funkcije tako zadate listom je izvršavanje LISP programa.

U ostatku teksta će se uglavnom koristiti ,,kvazi-predikatski" jezik i reprezentacija znanja koja ukazuje na predikatski račun prvog reda ili bliske forme. Ako se uzmu u obzir Hornove klauzule i rezolucija, takav način reprezentovanja znanja i jezik su najbliži PROLOG-u. PROLOG takođe koristi liste (sintaksa oblika `[`e_1`,`$\cdots$`,`e_n`]` ili `[glava|rep]`, dok se u tekstu koristi ,,." tačka umesto vertikalne crte ,,|") ali ne kao osnovnu strukturu podataka, odnosno način reprezentovanja znanja.

2.7 Grafovi

Formalna matematička definicija grafa je:

Definicija 2.1 *Struktura* $G = (X, R)$ *je* ***graf*** *gde je* X *skup čvorova ili temena grafa, a* R *je binarna relacija nad skupom* X *(*$R \subseteq X \times X$*).*

Ako je R simetrična, kaže se da graf nije orijentisan i veze između čvorova su ivice, a ako je antisimetrična (bitan je redosled temena) onda su veze između temena lukovi.

Definicija 2.2 $G' = (Y, V)$ *je* ***parcijalni graf*** *grafa* $G = (X, U)$ *akko* $Y = X$ *i* $V \subset U$*.*

G' *je* ***pod-graf*** *grafa* G *akko* $Y \subset X$ *i* $V = U - W$ *gde je*

$$W = \{\ (v, w) \mid v \in X - Y\ \vee\ w \in X - Y\}$$

(uklonjena su neka temena zajedno s lukovima).

Stepen *čvora je broj suseda tj. ukupan broj prethodnika i naslednika (ulaznih i izlaznih lukova).*

Putanja *od temena* a *do temena* b *u* G *je konačni niz temena* $c_1, ..., c_n$ *td. je* $a = c_1$ *i* $b = c_n$ *i svaki* $(c_i, c_{i-1}) \in U$*. Ako graf nije orijentisan onda je dovoljno da* $(c_i, c_{i-1}) \in U$ *ili* $(c_{i-1}, c_i) \in U$ *i onda je putanja* ***lanac*** *koji povezuje* a *i* b*.*

Ciklus *je zatvoren lanac tj.* $a = b$*.*

Ako za za svaka dva čvora grafa postoji lanac koji ih povezuje kaže se da je ***graf povezan****, a ako ih povezuje putanja (graf je orijentisan) onda je* ***jako povezan****.*

Postoji mnogi alati teorije grafova i algoritmi koji su korisni i u mnogim konkretnim primenama (npr. Warshall-ov algortiam za tranzitivno zatvorenje, problemi najkraćih puteva i drugo). Jedno od veoma korisnih proširenja pojma grafa su Petri mreže (i njeni derivati).

2.7.1 Petri-mreže

Osnovnu postavku Petri mreža u svojoj doktorskoj disertaciji dao je Carl Adam Petri, čija se formalna definicija odnosi na *standardne* ili obične Petri

mreže kao najrasprostanjeniji dijalekat (vrsta). Postoje i mnoga proširenja, primene i posledice ovog alata. Petri mreža kao struktura se oslanja na pojam multi-skupa (skup u kome je dozvoljeno ,,ponavljanje" elementa - multiset, bag - formalno par (S, f) gde je $f : S \to \mathbb{N}$ preslikavanje koje slika element osnovnog skupa S u broj ponavljanja - u suštini dovoljno je f kao multiskup ako se S podrazumeva), broj ponavljanja elementa x multiskupa B, $x \in B$, se označava i sa $\#(x, B)$ (njegova kardinalnost).

Definicija 2.3 *Petri mreža je petorka $C = (P, T, B, F, \mu)$, gde je:*

- $P = \{p_1, ..., p_n\}$ *neprazan skup mesta,*
- $T = \{t_1, ..., t_m\}$ *neprazan skup prelaza td.* $P \cap T = \emptyset$*,*
- $F : T \to \mathbb{N}^P$*,* ulazna funkcija *preslikava prelaz u multiskup ulaznih mesta,*
- $B : T \to \mathbb{N}^P$*,* izlazna funkcija *preslikava prelaz u multiskup izlaznih mesta,*
- $\mu : P \to \mathbb{N}$ *je* funkcija markiranja *koja dodeljuje nenegativan ceo broj mestu, ali može predstavljena i kao n-dimenzionalni* vektor markiranja $\mu = (\mu_{p_1}, ..., \mu_{p_n})$, $n = |P|$ *gde je μ_i broj tokena u mestu p_i.*

Prelaz $t_i \in T$ može biti *upaljen* ako je:

$$(\forall p_i \in P)\mu_{p_i} \geq \#(p_i, F(t_j))$$

Paljenjem prelaza $t_j \in T$ dolazi do promene vektora markiranja μ u novi vektor μ^* takav da je:

$$(\forall p_i \in P)\mu^*_{p_i} = \mu_{p_i} - \#(p_i, F(t_j)) + \#(p_i, B(t_j))$$

Nizom paljenja prelaza se definiše izvršavanje Petri mreže.

Graf Petri mreže $G = (V, A)$ je takav da skup čvorova $V = \{v_1, ..., v_s\}$ koga čine dva disjunktna skupa $V = P \cup T$, $P \cap T = \emptyset$ (skup *mesta* i skup *prelaza*), i $A = \{a_1, ..., a_r\}$ skup lukova gde vredi:

$$(\forall a_i \in A)a_i = (v_j, v_k) \Rightarrow (v_j \in P \wedge v_k \in T) \vee (v_j \in T \wedge v_k \in P)$$

Grafička reprezentacija mesta je obično krug ili elipsa (sa nekom oznakom tokena označavanja), a prelaz pravougaonikom ili vertikalnom crtom. Tako se graf sastoji pre svega iz dva tipa lukova:

- ulaznih (od mesta ka prelazu - važi ako je $F(t_j, p_i) > 0$, ako je vrednost veća od 1 upisuje se iznad luka)
- izlaznih (od prelaza ka mestu - važi ako je $B(t_j, p_i) > 0$, ako je vrednost veća od 1 upisuje se iznad luka)

Moguće su mnoge primene i primeri ovakvih struktura: modeli i formalna verifikacija distribuiranih sistema (multi-agentskih sistema, primera radi), komunikacionih protokola, upavljanje projektima i planiranje, modeli multi-procesorskih sistema, itd.

3 Formalni sistemi - deklarativno znanje i zaključivanje

Formalno predstavljanje znanja je neophodan korak u reprezentovanju znanja i izgrađivanju osnovnih struktura u programu pa i u VI. Formalni sistemi su vrsta apstraktnih struktura kojima se mogu strogo matematički zasnovati formalni jezici, matematička logika ili druge strukture i osnovne matematičke oblasti koje su neophodne kao osnovni primeri formalnog reprezentovanja znanja i zaključivanja o njemu - za strogo zasnivanje neophodno bi bilo definisati pojmove kao što su: niz, nizovi simbola (niske - ϵ je prazna reč dužine 0, Σ^n je skup svih niski dužine n nad alfabetom Σ, $\Sigma^* = \bigcup_{i \in N} \Sigma^i$), jezik kao podskup svih niski datog alfabeta čiji su elementi rečenice, formalna (kontekstno slobodna) gramatika kao struktura $G = (V, T, P, S)$ (gde su V neterminalni simboli, T terminalni, P skup produkcija tj. relacija među rečenicama kojima se zadaju pravila izvođenja (koraka transformacije) rečenica, S je početni neterminalni simbol) i jezik $L(G)$ njome definisan, itd.

3.1 Definicija formalnih sistema

Definicija 3.1 *Formalni sistem (FS) je uređena petorka* (Σ, G, A, P, T) *gde je:*

1. Σ *konačni alfabet (čiji su elementi terminalni simboli jezika formalnog sistema)*

2. G *formalna gramatika - kao način strogog definisanja pravila formiranja ispravnih rečenica (wff - well formed formulas) odnosno formula FS*

3. A *skup rečenica koje predstavljaju aksiome - formule FS koje imaju posebnu ulogu u FS.*

4. P *konačan skup pravila izvođenja (ili dedukcija, zaključivanja) rečenica (ispravnih u sistemu) u obliku relacija rečenica:*

$$U_1, U_2, ...U_p \rightarrow W_1, W_2, ...W_n$$

čime se označava izvođenje iz reči U_i $(1 \leq i \leq p)$ *u reči* W_j $(1 \leq j \leq n)$

5. T *skup teorema - formula FS koje se mogu izvesti iz aksioma, uključujući i aksiome*

Dokaz *je konačan niz reči* $M_1, ..., M_r$ *čiji su članovi ili aksiome ili reči izvedene iz prethodnih članova tog niza prema pravilima izvođenja (4).*

Teorema t *je reč (formula) za koju postoji dokaz tako da je* $M_r \equiv t$ *i piše se* $\vdash t$*. Aksiome su teoreme po definiciji. Dok se za nisku može u konačnom broju koraka odrediti da li je ispravna rečenica, za pitanje da li je formula teorema to ne mora biti tako.*

Važi: $T \subseteq L(G) \subseteq \Sigma^*$. Kao što postoje neterminalni simboli kod formalnih gramatika koji nisu deo alfabeta ali učestvuju u produkcijama (svojevrsne promenljive, konačno izvedena rečenica ih ne sadrži), tako se i u aksiomama i pravilima izvođenja mogu koristiti gde zamenjuju bilo koju ispravnu rečenicu FS (praktično se mogu shvatiti i interpretirati kao sheme aksioma i pravila - npr. jedna aksioma sa takvim simbolom predstavlja zapis prebrojivo mnogo aksioma, koliko ima i formula FS). Pravila koja sadrže takve promenljive zovu se prepravljanja (re-writing - odnose se na deo rečenice leve strane pravila), inače su zovu produkcijama. Pretpostavka je da je broj aksioma i rečenica rekurzivno prebrojiv (postoji pravilo, algoritam po kome se može doći do svakog u konačnom broju koraka).

Pored ovih apstraktnih struktura, značajan je i pojam konceptualizacije kao modela univerzalne algebre, odnosno trojke (Δ, F, R) gde je Δ skup elemenata domena, F skup funkcija (elementi su $f : \Delta^n \to \Delta$, različitih arnosti n), R skup relacija konceptualizacije (elementi su $\rho \subseteq \Delta^m$, različtih arnosti m). Uz predikatski račun prvog reda (PR1) kao odgovarajući jezik konceptualizacije dobijamo sintaksni nivo *deklarativnog znanja* koji određuje alfabet sa tri klase: simbolima konstanti domena, konstanti funkcija i konstanti relacija, a uz interpretaciju (preslikavanje ovakvih elemenata jezika u odgovarajuće elemente konceptualizacije tj. modela) dobija se *deklarativna semantika*, veza između sintakse (jezika) i semantike (konceptualizacije). Znanje formalizovano ovakvim strukturama se naziva *deklarativnim znanjem.* Značaj pojma konceptualizacije je i taj da ne mora da zavisi od izbora jezika, tako da umesto PR1 to može da bude jezik binarne tabele, semantičke mreže, okvira (koji se uglavnom mogu svesti na PR1, proceduralni deo okvira se jedino ne uklapa) ili neki drugi.

Pomenuti formalni sistemi su osnova za definisanje pojma formalnog matematičkog dokaza, gde se obično podrazumeva Hilbertov sistem dedukcije koji posmatra logiku sa čisto sintaksnog aspekta, dok teorija modela (univerzalna algebra + logika) teži semantičkom pogledu. Definicija FS potiče iz knjige [JL], o formalnim jezicima se može saznati više iz [HU], dok se o deklarativnom znanju i zaključivanju može saznati više iz [GN]. Slede primeri i pojašnjenja.

3.2 Iskazni račun i predikatski račun prvog reda

Tako je iskazni račun jedan od najpoznatijih primera formalnih sistema (klasičan oblik matematičke logike u užem smislu, kao i Bulova algegbra, dok se u širem smislu podrazumeva i teorija modela, teorija skupova i teorija izračunljivosti), i mnogo više od toga - prethodi definiciji predikatskog računa prvog reda (PR1), koji je osnova mnogih praktičnih inteligentnih sistema i osnovni primer matematičkog jezika i zaključivanja kao modela ljudskog razmišljanja - PR1 se može formalizovati (u smislu prethodno definisanih FS) i praktično koristiti kao reprezentacija znanja, ali i kao metod dedukcije (zaključivanja o znanju i njegovim posledicama):

- alfabet: {p,q,r,s...,$\neg$,$\wedge$,$\vee$,$\Rightarrow$,(,)}
- ako su w, w_1 i w_2 pravilne rečenice onda su to i:

  slovo alfabeta, (w),
  $\neg w$,
  $w1 \Rightarrow w2$,
  $w1 \wedge w2$,
  $w1 \vee w2$

- šema aksioma (koje uključuju i $\vee$ iako se može sistem definisati potpuno bez disjunkcije koja se onda naknadno definiše: $p \vee q \equiv \neg(\neg p \wedge \neg q)$, i time se nešto smanji broj aksioma, ali to ne znači da je onda dedukcija efikasnija):

$$p \Rightarrow (q \Rightarrow p) \tag{1}$$
$$(p \Rightarrow (q \Rightarrow r)) \Rightarrow ((p \Rightarrow q) \Rightarrow (p \Rightarrow r)) \tag{2}$$
$$p \wedge q \Rightarrow p \tag{3}$$
$$p \wedge q \Rightarrow q \tag{4}$$

$$p \Rightarrow (q \Rightarrow (p \wedge q)) \quad (5)$$
$$p \Rightarrow p \vee q \quad (6)$$
$$q \Rightarrow q \vee q \quad (7)$$
$$(p \Rightarrow q) \Rightarrow ((r \Rightarrow q) \Rightarrow (p \vee r \Rightarrow q)) \quad (8)$$
$$(p \Rightarrow q) \Rightarrow ((p \Rightarrow \neg q) \Rightarrow \neg p) \quad (9)$$
$$p \Rightarrow (\neg p \Rightarrow q) \quad (10)$$
$$p \vee \neg p \quad (11)$$

(umesto (2) može $(p \Rightarrow q) \Rightarrow ((p \Rightarrow (q \Rightarrow r)) \Rightarrow (p \Rightarrow r))$,
i umesto (9) i (10) može $\neg\neg p \Rightarrow p$)

- *modus ponens* je dovoljan kao jedino pravilo izvođenja:
  w_1, $w_1 \Rightarrow w_2 \quad \rightarrow \quad w_2$

  (mada se mogu koristiti i druga kao što su to npr. *modus tolens*: $\neg w_2$, $w_1 \Rightarrow w_2 \rightarrow \neg w_1$, $\wedge$-eliminacija: $w_1 \wedge w_2 \rightarrow w_1$, w_2, $\wedge$-uvođenje: w_1, $w_2 \rightarrow w_1 \wedge w_2$, itd.)

Kod iskaznog računa preslikavanje τ reči u izraz sa funkcijama kao interpretacijama logičkih operatora nad skupom B - takvo preslikavanje u izraz koji zavisi samo od slova u njemu je interpretacija (bez vrednosti promenljivih), a za niz koknretnih vrednosti slova u B se kaže da je valuacija promenljivih. Ako se tako definiše semantika reči iskaznog računa nad skupom $B = \{\top, \bot\}$ (Bulova algebra), onda su validne reči (tautologije) one čija je istinitosna vrednost uvek $\top$ (ili istinite) bez obzira na vredost promenljivih i interpretaciju (i pokazuje se da je svaka tautologija teorema iskaznog računa, Emil Post, 1921).

Važne osobine ovog formalnog sistema su (ili nekog drugog formalnost sistema prvog reda): nekontradiktornost (konzistentnost), kompletnost (svaka validna reč ili njena negacija su teoreme sistema), odlučive (uvek postoji postupak kojim se u konačno mnogo koraka za bilo koju reč utvrđuje da li jeste ili nije teorema).

Sledeći važan primer formalnog sistema je predikatski račun prvog reda (PR1) gde se uvode i pojmovi predikata odnosno relacije (određene arnosti), univerzalni kvantifikator $\forall$, promenljive i konstante (kvantifikator $\exists$ se definiše sa $(\exists x)P \equiv$

$\neg(\forall x)\neg P)$.

Dodatne aksiome pored aksioma iskaznog računa:

$$(\forall x)P(x) \Rightarrow P(u) \quad \text{(aksiom partikularizacije)},$$
$$((\forall x)(w_1 \Rightarrow w_2)) \Rightarrow (w_1 \Rightarrow (\forall x)w_2), \text{ x nije slobodna u } w_1.$$

Dodatna pravila izvođenja:

$$\text{generalizacija:} \quad w \Rightarrow (\forall x)w, \text{ gde je } x \text{ slobodna u } w).$$

Definicija interpretacije je takođe proširena uz pojmove apstraktne strukture, konceptualizacije (koja uz PR1 daje model deklarativnog znanja), kao trojke (Δ, F, R) gde je Δ domen - skup iz koga interpretacije mogu uzimati vrednosti, F je skup funkcija, R je skup relacija tako da su ti objekti slike (piše se npr. $I(\rho) = \rho^I$) funkcionalnih i relacionih konstanti kojima su građeni termi (izrazi nad konstantama, promenljivama i funkcijama, npr. formalna aritmetika) i atomski iskazi redom (kojima se ,,proširuju" formule iskaznog računa). Formule mogu biti i kvantifikovane - stroga definicija je rekurzivnog karaktera. Tada se može definisati: $\models_I [V]\phi$ ili reč ϕ je zadovoljena akko postoji interpretacija I i valucija V td. je ϕ istinita. Interpretacija I je model reči (formule) ako je zadovoljena za svaku valuaciju. Ako je reč ϕ zadovoljena bez obzira na interpretaciju onda je tautologija ($\models \phi$). Formula ϕ može se izvesti koristeći se i formulama nekog skupa formula $\mathcal{T}$ (hipoteza, npr. baza podataka u PROLOG programu, ,,baza znanja") kao da su aksiome što se zapisuje kao $\mathcal{T} \vdash \phi$. Ako je formula ϕ zadovoljena za svaku interpretaciju za koju je zadovoljen i skup hipoteza $\mathcal{T}$ onda se kaže da je logička posledica ili implikacija tog skupa formula i piše se $\mathcal{T} \models \phi$. Skup $\mathcal{T}$ je teorija ako je zatvoren logičkom implikacijom (ne postoji teorema izvan njega koja proizilazi iz tog skupa) i može kao deo formalnog sistema isto biti konzistentan, kompletan ili odlučiv. Teorija je konačno aksiomatizabilna ako postoji konačna baza (podskup reči) iz kojih se mogu izvesti svi elementi $\mathcal{T}$. Teorija je nekonzistentna ako ne postoji interpretacija i valuacija tako da je svaki element zadovoljen. Takođe, može se pokazati $\mathcal{T} \vdash \phi \equiv \mathcal{T} \models \phi$ za datu PR1 teoriju $\mathcal{T}$. Za datu teoriju (ili sistem) i njene dve interpretacije I, J se kaže da su elementarno ekvivalentne ($I \equiv J$) akko važi $\models_I \phi \equiv \models_J \phi$ za proizvoljnu teoremu ϕ.

Za PR1 kao formalni sistem se pokazuje da jeste nekontradiktoran, kompletan (Gedelova teorema kompletnosti kojom se praktično pokazuje da je u PR1 zadoljovost ekvivalentna konzistentnosti, odnosno semantička vrednost formule ekvivalentna je sintaksnoj - ovo je povezano i sa osobinom kompaktnosti: po teoremi kopmaktnosti svaki nekonzistentan skup formula u PR1 ima *konačan* nekonzistentan podskup tj. skup je konzistentan ako je takav i svaki njegov konačan podskup - ovu lepu osobinu nema, recimo, PR2 gde se kvantifikuju i predikati pored promenljivih) ali da nije odlučiv (Church-ova teorema: postoje neodlučivi formalni sistemi, Gedelova teorema nekompletnosti), kao ni teorija grupa, prstena i polja (što je Tarski pokazao - dok su npr. projekivna geometrija i teorija zatvorenih realnih polja odlučive). Formalna aritmetika (Peano zasnovao oslanjajući se na PR1) nije kompletna (Gedelov dokaz aritmetizacijom). Značajna ograničenja formalnih sistema pokazuje i teorema Tarskog - postoje formalni sistemi u kojima za svaku interpretaciju postoji valjana reč za koju ne postoji dokaz.
Sledeće teoreme su praktično veoma korisne:

Teorema 3 ***(Teorema Dedukcije)***
Ako je $\Delta \bigcup \{\phi\} \vdash \psi$ *onda je* $\Delta \vdash (\phi \Rightarrow \psi)$.

Teorema 4 ***(Pravilo T)***
Ako je $\Delta \vdash \phi_1, ..., \Delta \vdash \phi_n$ *i* $\{\phi_1, ..., \phi_n\} \vdash \phi$ *tada je* $\Delta \vdash \phi$.

Teorema 5 ***(Teorema kontrapozicije)***
$\Delta \bigcup \{\phi\} \vdash \neg\psi$ *akko* $\Delta \bigcup \{\psi\} \vdash \neg\phi$.

Teorema 6 ***(Teorema odbacivanja)***
Ako je $\Delta \bigcup \{\phi\}$ *nekonzistentna tada je* $\Delta \vdash \neg\phi$.

Teorema 7 ***(Teorema generalizacije)***
Ako je $\Delta \vdash \phi$ *i* ν *je promenljiva koja se pojavljuje kao slobodna u* Δ *onda* $\Delta \vdash (\forall\nu)\phi$.

Ovim teoremama se npr. može skratiti formalan dokaz ako se koriste kao svojevrsna heuristika (kao i dodatnim pravilim zaključivanja). Postoje i mnoge alternativne logike i njihovi formalni sistemi sa svojim osobinama i domenima primene - npr. intuicionistička (naglašava matematički konstruktivizam pre nego pojam istine, npr. u PR1 problem egzistencije stepena iracionalnih brojeva koji je racionalan: $\sqrt{2}^{\sqrt{2}^{\sqrt{2}}} = 2 \in \mathbb{Q}$ iako o osnovi $\sqrt{2}^{\sqrt{2}}$ ne moramo da znamo da li je takva - kod intuicionističke logike to nije dokaz), modalna, temporalna, itd.

3.3 Zaključivanje

Automatsko dokazivanje teorema s obzirom na sve prethodno može da bude veoma težak problem. Neki metod zaključivanja tj. dokazivanja teorema je *ispravan* ako je svaki zaključak dobijen postupkom tog metoda iz njegove baze znanja logička posledica te baze (*kompletan* ako važi i obratno) u smislu logičke implikacije i zaključivanja u PR1. Postoje klase formalnih sistema i metodi koji su u tome uspešni, a jedan od poznatijih je algoritam rezolucije (na kome se bazira interpretacija PROLOG programa).

Procedura zaključivanja predstavlja izbor narednog koraka zaključivanja kao što je to npr. Markovljeva funkcija `next` koja slika skup rečenica baze znanja (kojima su zadate polazne pretpostavke i izvedene posledice) u naredni, izvedeni skup rečenica baze znanja. Može da zavisi od prethodnih zaključaka (istorije) makar implicitno zbog same prirode procedure. Ako se baza znanja u svakom koraku izvođenja uvećava tj. ako je svaki naredni korak nadskup prethodnog onda je procedura zaključivanja *inkrementalna*.

4 Rezolucija

Rezolucija je primer metode zaključivanja koja se može efikasno automatizovati, i u određenim slučajevima se pokazuje da je to ispravna i kompletna procedura zaključivanja.

4.1 Klauzalna forma

Rezolucija se primenjuje nad jednim pojednostavljenim oblikom izraza PR1 čiji su osnovni elementi klauzule. Klauzule se sastoje od literala koji su zapravo atomski predikati (pozitivni literali) ili njihove negacije (negativni literali), a klauzula je disjunkcija literala. Od posebnog značaja su Hornove klauzule koje sadrže najviše jedan pozitivan literal. Klauzalna forma je konjunkcija klauzula. Skica algoritma za pretvaranje iskaza PR1 u klauzalnu formu je (oblik PRENEX algoritma za normalnu formu iskaza):

1. izbacivanje implikacija:
   $\phi \Rightarrow \psi$ se zamenjuje sa $\neg\phi \vee \psi$
   $\phi \Leftarrow \psi$ se zamenjuje sa $\phi \vee \neg\psi$
   $\phi \Leftrightarrow \psi$ se zamenjuje sa $(\neg\phi \vee \psi) \wedge (\phi \vee \neg\psi)$

2. ulazak negacije:
   $\neg\neg\phi$ se zamenjuje sa ϕ
   $\neg(\phi \wedge \psi)$ se zamenjuje sa $\neg\phi \vee \neg\psi$
   $\neg(\phi \vee \psi)$ se zamenjuje sa $\neg\phi \wedge \neg\psi$
   $\neg\forall\nu\phi$ se zamenjuje sa $\exists\nu\neg\phi$
   $\neg\exists\nu\phi$ zamenjuje se sa $\forall\nu\neg\phi$

3. standardizovanje promenljivih - za svaki kvantifikator posebna promenljiva:

   npr. $(\forall x P(x)) \vee (\exists x P(x))$ zamenjuje se sa $(\forall x P(x)) \vee (\exists y P(y))$

4. eliminacija kvantifikatora - eliminacija egzistencijalnog kvantifikatora, skolemizacija:

   svaka formula koja nije pod dejstvom univerzalnog kvantifikatora oblika $(\exists x)P(x)$ se zamenjuje formulom $P(C)$ gde je C (Skolemova) konstanta koja se ne javlja ni u jednoj drugoj formuli.

Svaka formula prethodnog oblika koja je i pod dejstvom univerzalnog kvantifikatora se zamenjuje formulom u kojoj je promenljiva pod dejstvom egzistencijalnog kvantifikatora zamenjena (Skolemovom) funkcijom (argumenti su promenljive pod dejstvom univerzalnog kvantifikatora) koja se ne javlja ni u jednom drugoj formuli.
Npr. $\forall x \forall y P(x, y, F(x, y))$ umesto $\forall x \forall y \exists z P(x, y, z)$.

5. eliminacija kvantifikatora - eliminacija univerzalnog kvantifikatora:
   pošto drugih kvantifikatora nema, nema ni zabune ako se uklone svi kvantifikatori (slično generalizaciji).

6. svođenje na disjukntivnu normalnu formu:
   $\phi \vee (\psi \wedge \chi)$ se zamenjuje sa $(\phi \vee \psi) \wedge (\phi \vee \chi)$

7. zapis klauzalne forme:
   npr. umesto $P \wedge (Q \vee R)$ piše se: $\{P\}, \{Q, R\}$

8. standardizacija promenljivih:
   zamene se promenljive td. se ni jedna promenljiva ne javlja u više klauzula od jedne.

4.2 Unifikacija

Unifikacija je postupak u kojem se dva izraza izjednačavaju (ukoliko je to moguće) zamenama promenljivih odgovarajućim termovima. Više takvih zamena („vezivanja") promenljivih $x_1, ..., x_n$ termovima $t_1, ..., t_n$ je supstitucija $\sigma = \{x_1/t_1, ..., x_n/t_n\}$ pod uslovom da se ni jedna od navedenih promenljivih ne javlja ni u jednom od termova. Supstitucija primenjena na neku formulu predstavlja jednu instancu te formule. Ako supstitucija σ nema nijednu promenljivu koju ima supstitucija τ onda je τ različita od σ. Kompozicija dveju takvih supstitucija $\sigma\tau$ (zapisuje se postfiksno, kao i primena supstitucije na izraz) se dobija tako što se najpre primene zamene iz τ na σ a onda se dobijenom dodaju zamene iz τ. Supstitucija σ je opštija ili jednako opštija od τ ako $(\exists\delta)\sigma\delta = \tau$. Najopštiji unifikator (nou) γ izraza ϕ i ψ je opštiji od bilo koje druge supstitucije koja σ koja izjednačava ta dva izraza ($\phi\sigma = \psi\sigma$) tj. $(\exists\delta)\phi\gamma\delta = \phi\sigma = \psi\sigma$. Jedinstven je do na imenovanje promenljivih.

Rekurzivni algoritam za traženje nou za dva izraza je (može se uopštiti):

```
Nou(x,y)
  if x=y ==> Return()
  if Var(x) ==> Return(Nouvar(x,y))
  if Var(y) ==> Return(Nouvar(y,x))
  if Const(x) or Const(y) ==> Return(FALSE)
  if Not(Length(x)==Length(y)) ==> Return(FALSE)
  i=0, g=[]
  loop
    if i==Length(x) ==> Return(g)
    s=Nou(Part(x,i),Part(y,i))
    if s==FALSE ==> Return(FALSE)
    g=Compose(g,s)
    x=Substitute(x,g)
    y=Substitute(y,g)
    i=i+1
  end loop
end Nou
Nouvar(x,y)
  if Includes(x,y) ==> Return(FALSE)
  Return([x/y])
end Nouvar
```

Objašnjenje, ukratko: ,,već implementiran predikat" `Var` tj. funkcija je istinita ako je argument promenljiva, `Cons` ako je argument konstanta (uključujući i funkcijsku konstantu tj. ime funkcije - npr. `Part(F(A,B,C),0) == F`, `Part(F(A,B,C),1) == A`, itd. a važi `Const(F)=TRUE`), `Compose` spaja dve liste, `Substitute` primenjuje na izraz listu zamena (supstituciju).

Φ' je faktor Φ ako $(\exists\Psi \subseteq \Phi)(\exists\gamma)\gamma = nou(\Psi)$ td. $\Phi' = \Phi\gamma$.

4.3 Princip rezolucije

Slično modus ponensu - ako se primeni na jednostavan slučaj prikazan klauzulama sa prostim literalima izgleda ovako:

$$\frac{\{R,P\},\ \{Q,\ \neg\ P\}}{\{R,Q\}}$$

Horizontalnom crtom je razdvojena *rezolventa* (izvedena klauzula) od polaznih klauzula, slično zapisu pravila u PR1. Pozitivne (bez ¬) i negativne instance literala (sa ¬) `P` koje se javljaju u polaznim klauzulama se ,,potiru". U opštem slučaju, literali sadrže terme sa promenljivama i tada je neophodan algoritam unifikacije da bi se primenilo pravilo rezolucije:

$$\frac{\begin{array}{ll}\Phi, & \phi \in \Phi' \\ \Psi, & \neg\psi \in \Psi'\end{array}}{(\Phi - \{\phi\}) \bigcup (\Psi - \{\neg\phi\})\gamma, \quad \text{td. je } \gamma = nou(\phi, \psi)}$$

Ako je rezolventa (zaključak principom rezolucije) prazna klauzula, to znači da je u pitanju kontradikcija među pretpostavkama tj. postoji kontradikcija u bazi znanja.

Dedukcija (zaključak) Φ rezolucijom na osnovu baze (znanja) Δ je niz klauzula čiji je element Φ i čiji je svaki član dobijen primenom principa rezolucije ili na klauzulu iz Δ ili na nekog prethodnog člana niza. Kada se prikazuje niz koraka zaključivanja dodaje se Δ na kraju ako pripada bazi ili redni broj koraka na osnovu kojih se zaključuje, ili Γ ako je u pitanju negirani cilj (ako je cilj pokazati ispravnost upita zadatog klauzulom ili literalom onda se njegova negacija ,,privremeno ubaci" u bazu da bi se došlo do kontradikcije - odbacivanje rezolucijom, sistem je nezadovoljiv).

Algoritam kojim se realizuje automatska dedukcija rezolucijom se svodi na građenje stabla zaključivanja (po nivoima, počevši od baze kao polaznog nivoa, ,,resolution trace", npr. dva pokazivača (jedan ,,sporije", jedan ,,brže") prolaze kroz sve rezolvente uključujući i novonastale) sve do ispunjenja uslova. Uslov je obično ili prazna klauzula kojom se traži odgovor ISTINA / NEISTINA na postavljen cilj (zadatu klauzulu), ili se traže vrednosti promenljivih (,,fill-in-the-blank") koje zadovoljavaju postavljeni cilj gde se onda koristi pomoćni predikat `Ans(`$X_1, ...$`)` onolike arnosti koliko nepoznatih učestvuje u upitu. Primer - upit glasi `P(z,Jon)`:

```
1. {F(Art,Jon)}          Δ
2. {F(Bob,Kim)}          Δ
3. {¬F(x,y),P(x,y)}      Δ
4. {¬P(z,Jon),Ans(z)}    Γ
```

5.	{P(Art,Jon)}	1, 3
6.	{P(Bob,Kim)}	2, 3
7.	{¬F(w,Jon),Ans(w)}	3, 4

8.	{Ans(Art)}	4, 5
9.	{Ans(Art)}	1, 7

Pokazuje se da je princip rezolucije ispravan i kompletan metod dedukcije (koristeći Erbranove teoreme, Erbranov svet konstantnih terma ...).

4.4 Rezolucija i jednakost

Programsko priključenje (procedural attachment) je korisno proširenje postupka rezolucije (kao i bilo koje druge dedukcione procedure) - predikat (literal) ili funkicja se evaluira tako što se izvrši program tj. kod koji vraća njegovu vrednost. Time se može smanjiti broj koraka dedukcije, ali to može biti i problem jer u takvim slučajevima princip rezolucija nemora biti dovoljno moćan pa se mora pribeći ipak doslednom aksiomatskom definisanju. Primer je relacija jednakosti koja ima podrazumevane osobine, recimo klasična rekurzivna definicja faktorijala:

$fact(0) = 1,\ fact(k) = k * fact(k-1))$

Rezolucija nije dovoljna za takvu definiciju, već se ili mora preformulisati tako da su svi termi bez promenljivih na prvom nivou literala u kojima se javljaju:

$Fact(0) = 1$
$k - 1 = j \wedge Fact(j) = m \wedge k * m = n \Rightarrow Fact(k) = n$

ili se aksiomatizuje jednakost a onda i aksiome supstitucije terma termima za svaku relaciju i funkciju:

$\forall x\ x = x$
$\forall x \forall y\ x = y \Rightarrow y = x$
$\forall x \forall y \forall z\ x = y \wedge y = z \Rightarrow x = z$
$\forall k \forall j \forall m\ k = j \wedge Fact(j) = m \Rightarrow Fact(k) = m$

$\forall k \forall j \forall m \forall n \; j = m \wedge k * m = n \Rightarrow k * j = n$

4.5 Strategije rezolucije

Drvo rezolucije lako može da ekplozivno naraste i time postupak dedukcije postaje neefikasan. U ovom poglavlju se razmatraju varijante kao strategije i heuristike kojima se to može izbeći. Osnovna osobina svih ovih strategija je upotreba Hornovih klauzula. Može se pokazati da ako se baza znanja sastoji samo od Hornovih klauzula, da je svaka od ovih strategija ispravan i kompletan metod dedukcije.

4.5.1 Strategije brisanja

Jedan način poboljšanja rezolucije je brisanje nepotrebnih klauzula iz baze u određenim slučajevima.

Eliminacija čistih literala:
Literal je čist ako se nigde u bazi znanja ne pojavljuje nijednja njegova komplementarna instanca. Klauzule koje ga sadrže su beskorisna za odbacivanje rezolucijom i zato se mogu brisati iz baze. Dovoljno je jednom primeniti ovo pravilo na početku procesa rezolucije.

Eliminacija tautologija:
Tautologija je klauzula koja sadrži komplementarne literale. Pokazuje se da zadovoljivost baze znanja ne zavisi uopšte od takvih klauzula, prema tome mogu biti brisane. Unifikacija se ne koristi, za razliku od prethodnog, da bi se došlo do tautologija, i ovo pravilo može biti upotrebljeno nakon svakog dedukcionog koraka.

Eliminacija podklauzula:
Klauzula Φ je podklauzula („subsumption") klauzule Ψ akko postoji supstitucija σ td. $\Phi_\sigma \subseteq \Psi$. Podklauzule se mogu brisati i ovo pravilo kao i prethodno se može primenjivati nakon svakog koraka dedukcije.

4.5.2 Jedinična rezolucija

Jedinična rezolventa je ona kojoj je bar jedan roditelj jedinična klauzula, tj. sa samo jednim literalom (singlton). Jedinična rezolucija je ona u kojoj su sve rezovlente jedinične. Jedinično odbacivanje je ono koje je dostignuto jediničnom dedukcijom.

4.5.3 Ulazna rezolucija

Ulazna rezolventa je ona kojoj je bar jedan roditelj element baze znanja. Ulazna rezolucija je ona u kojoj su sve rezovlente ulazne. Ulazno odbacivanje je ono koje je dostignuto ulaznom dedukcijom.

4.5.4 Linearna rezolucija

Linearna rezolucija (ancestry-filtered) je vid uopštenja ulazne rezolucije. Linearna rezolventa ima bar jednog roditelja koji je ili u bazi znanja ili je predak svog drugog roditelja. Linearna rezolucija počinje gornjom klauzulom (iz baze znanja), i svaki sledeći korak sledi iz poslednje rezolvente (bliski roditelj) i klauzule koja je u bazi znanja ili predak prvog / bliskog roditelja (daleki roditelj).

4.5.5 Rezolucija skupom podrške

Ako odbacime sve rezolvente isključivo nad klauzulama iz skupa znanja koji je zadovoljiv pokazuje se da to ne utiče na odbacivanje rezolucijom. Podskup Γ skupa Δ (baze znanja) td. je $\Delta - \Gamma$ zadovoljiv zove se skupom podrške za Δ. Rezolvetna skupom podrške ima uvek jednog roditelja iz Γ ili je potomak od Γ. Dedukcija skupom podrške se sastoji od rezolventi skupom podrške.

Ako je baza zadovoljiva onda su negirane klauzule cilja upravo skup podrške. Dokazi dobijeni ovom metodom polaze od cilja unatrag i obično su ,,čitkiji" od drugih.

4.5.6 Uređena rezolucija

Ova strategija je veoma restriktivna ali i veoma efikasna. Klauzule se tretiraju kao uređeni nizovi literala i rezolvente mogu biti samo nad prvim literalima u klauzuli.

4.5.7 Usmerena rezolucija

Ovo je vid uređene rezolucije u kojem se klauzule razvrstavaju u dve grupe Hornovih klazula: prednje (pozitivni literal je na kraju) i zadnje (pozitivni literal je na početku). Tako onda imamo dve vrste rezolventi i rezolucija: unapred (u kojem učestvuju prednje) i unazad (u kojem učestvuju zadnje). Za neke upite je efikasnije koristiti jednu podstrategiju od druge. Sam problem biranja podstrategije je NP-kompletan.

4.5.8 Sekvencijalno zadovoljenje uslova

Ovo je strategija koja se koristi za ciljeve gde se traže vrednosti i gde su upiti oblika npr.:

$P \wedge Q \wedge R$

... gde se traže vrednosti promenljivih za koje je zadovoljen. Sam redosled formula u konjunkciji upita je bitan u odnosu na broj konstantnih literala po svakom konjunktu u bazi znanja. Pokazuje se da je optimalan redosled određen td. se pretraživanje procena koštanja minimizuje kao i samo koštanje redosleda tj. broj dedukcionih koraka potrebnih da bi se došlo do cilja.

5 Zaključivanje sa nesigurnim uverenjima i drugi načini zaključivanja

5.1 Nemonotono zaključivanje

U ovom poglavlju se razmatraju metodi dedukcije u kojima dodavanje formule skupu pretpostavki utiče na zaključak. Kod logičkog zakljičivanja u PR1 to nije bio slučaj i zato se zove monotonim. Nemonotono zaključivanje može zavisiti i od celog skupa pretpostavki a ne od njegovog podskupa, ili od formula koje ne pripradaju skupu pretpostavki. Ovakvo prošerenje zaključivanja može biti od značaja za sistem koji npr. treba da se prilagodi nepotpunoj bazi znanja.

Skup formula Δ se može zatvoriti logičkom implikacijom $\tau(\Delta)$ ali to nemora dati kompletnu teoriju. Najjednostavniji metod kompletiranja je pretpostavka zatvorenog sveta (PZS, ,,closed-world assumption"). Jednostavno, ako se za konstantni literal ne može izvesti da pripada teoriji niti njegova negacija, onda se njegova negacija dodaje u skup uverenja Δ_{pu} - pretpostavljena uverenja, pored Δ skupa ispravnih aksioma teorije, $\Delta \cup \Delta_{pu}$ je onda dopunjena teorija. $\phi \in PZS(\Delta)$ akko $(\Delta \bigcup \Delta_{pu}) \models \phi$. Pokazuje se da ako je Δ konzistentna baza i sastoji od Hornovih klauzula onda je i $PZS(\Delta)$ konzistentna. Uz PZS se obično koristi i pretpostavka jedinstvenih imena (PJI, ,,unique names assumption") koja primenjuje princip PZS na jednakost $(\Delta \nvDash (t_1 = t_2) \Rightarrow \{t_1 \neq t_2\} \in \Delta_{pu})$, kao i pretpostavka zatvorenja domena (PZD, ,,domain closure assumption"), kojom se praktično svaki kvantifikator može zameniti konačnim disjunkcijama i konjunkcijama. PZD je zadat aksiomom $(\{(\forall x)x = t_1 \vee \cdots x = t_n\} \in \Delta_{pu})$, gde su t_i konstante objekata jezika, pod uslovom da nema funkcijskih konstanti u jeziku (inače postoji beskonačan broj termova nad konstantama koje bi trebalo staviti u ovakvu aksiomu ili ih kvantifikovati). PZD prevazilazi ograničenje da su jedino one konstante objekata koje se javljaju u bazi Δ moguće.

Baza se takođe može kompletirati u odnosu na svoje predikate tako da se pretpostavlja da zadate činjenice u bazi definišu sve zadovoljive vrednosti predikata. Može se pokazati da je ovo ekvivalentno postupku PZS uz neke pretpostavke. Suštinu čini $COMP[\Delta; P]$ kompletiranje predikata P u bazi Δ koje daje proširenje baze tako da P važi samo za one vrednosti za koje je

P istinit u bazi, za koje baza Δ to ,,dozvoljava". Npr. ako je $\Delta = \{P(A)\}$ onda važi $P(A) \Leftrightarrow ((\forall x)x = A \Rightarrow P(x))$ a formula $(\forall x)P(x) \Rightarrow x = A$ daje potreban uslov da bude zadovoljeno isključivo jedino $P(A)$. U tom slučaju je $COMP[\Delta; P] \equiv (\Delta \wedge ((\forall x)P(x) \Rightarrow x = A)) \equiv ((\forall x)P(x) \Leftrightarrow x = A)$ (može se odmah koristiti i ekvivalencija umesto implikacije). Ako je $\Delta = \{P(A), P(B)\}$ onda važi $COMP[\Delta; P] \equiv \Delta \wedge ((\forall x)P(x) \Rightarrow x = A \vee x = B)$. Kompletiranje predikata odgovara PZS u odnosu na predikat gde se PZS primenjuje samo u odnosu na zadati skup predikata (ako je to skup svih predikata u bazi onda se poklapa sa PZS, npr. iz $\Delta = \{(\forall x)Q(x) \Rightarrow P(x), Q(A), R(B) \vee P(B)\}$ se dobija $\neg R(B)$ i $\neg P(B)$ u opštem slučaju, a u odnosu na predikat P se dobija samo $\neg P(B)$ što posle dovodi do zaključka $R(B)$) - tu se javlja problem nekonzistentnosti iako se koristi baza Hornovih klauzula u odnosu na predikat (npr. ako je $\Delta = \{P(A) \vee Q, P(B) \vee \neg Q\}$ onda se u odnosu na P dobija i $P(A)$ i $P(B)$, što je nekonzistentno sa Δ). Zato se kompletiranje radi samo sa predikatima *usamljenim* u bazi - skup klauzula je usamljen u P akko svaka klauzula sa pozitivnim pojavljivanjem (instancom) P ima najviše jedno takvo pojavljivanje. Usamljene klauzule u odnosu na predikat jesu Hornove, ali obratno ne važi. Postoji postupak *paralelnog kompletiranja* usamljenih klauzula za skup predikata u bazi, za koji se može pokazati da čuva konzistentnost, i u kojem se pazi da ne dođe do cirkularnih definicija (predikati $\Pi = \{P_1, \cdots, P_n\}$ su *uređeni*: za svaku $(\forall x)E_i \Rightarrow P_i(x)$ disjunkciju klauzula iz baze za P_i, E_i da sadrži nijedan iz $\{P_i, \cdots, P_n\}$ niti negativne instance iz $\{P_1, \cdots, P_{i-1}\}$) i gde se kompletiranje skupa predikata dobija kao konjunkcija kompletiranja pojedinih predikata. U opštem slučaju kompletiranje je $COMP[\Delta; P] \equiv_{def} \Delta \wedge ((\forall x)P(x) \Leftrightarrow \bigvee E_i)$ gde su E_i leve strane implikacija klauzula u normalnoj formi u bazi $(\forall x)E_i \Rightarrow P(x)$ koje se mogu grupisati disjunkcijom. Normalna forma klauzula je oblika $\forall x(\exists y(x = t) \wedge Q_1 \wedge \cdots \wedge Q_m) \Rightarrow P(x)$ gde se pod $x = t$ podrazumeva $x_1 = t_1 \wedge \cdots x_n = t_n$, t_i su termi, x promenljive koje se ne javljaju u t_i a Q_i literali u kojima se ne javlja P.

Ovo se može uopštiti minimalnim modelom, konstrukcijom u PR2 (kvantifikuju se predikati) td. kompletiranje predikata ,,radi" i za formule oblika $P(A) \vee P(B)$ koje nisu usamljene u bazi (cirkumskripcija): $CIRC[\Delta, P] \equiv \Delta \wedge ((\forall P^*)(\Delta(P^*) \wedge P^* \leq P) \Rightarrow P \leq P^*)$ gde je $A \leq B \equiv_{def} ((\forall x)A(x) \Rightarrow B(x))$ a x može biti i n-torka promenljivih.

5.2 Taksonomijske hijerarhije i pretpostavljeno zaključivanje (default reasoning)

Često je potrebno predstaviti bazu znanja u obliku šeme odnosa među objektima, kao što je to npr. ,,Noj(x) ⇒ Ptica(x)", odnosno upotrebom relacije ,,JESTE" koja je parcijalno uređena i tranzitivna (Noj JESTE Ptica). Mogu postojati izuzeci u ovakvom odnosu nasleđivanja koji se opisuju pravilima prekidanja nasleđivanja (inheritance cancellation rules). Svaki objekat može imati neke opisane osobine koje su date skupom rečenica osobina Δ_P, a prethodne rečenice o odnosima i izuzecima odnosa daju Δ_H - taksonomijsku hijerarhiju. Dobro je takva pravila napisati dovoljno uopšteno - npr. ako je data rečenica u Δ_P: `Stvar(x)` $\wedge$ $\neg$ `Ab1(x)` $\Rightarrow$ $\neg$ `leti(x)` gde je opisana osobina letenja stvari, onda je pravilo izuzetka u Δ_H: `Ptica(x)` $\Rightarrow$ `Ab1(x)`, gde je `Ab1` predikat koji se vezuje za određeni tip izuzetka, abnormalnosti. Da bi primer bio kompletan, u Δ_H se mogu uvrstiti onda:

```
Stvar(Tviti)
Ptica(x) ⇒ Stvar(x)
Ptica(x) ⇒ Ab1(x)
Noj(x) ⇒ Ptica(x)
Noj(x) ⇒ Ab2(x)
Leteći-Noj(x) ⇒ Noj(x)
Leteći-Noj(x) ⇒ Ab3(x)
```

što se može prikazati i grafom, dok se u Δ_P mogu uvrstiti rečenice:

```
Stvar(x) ∧ ¬ Ab1(x) ⇒ ¬ leti(x)
Ptica(x) ∧ ¬ Ab2(x) ⇒ leti(x)
Noj(x) ∧ ¬ Ab3(x) ⇒ ¬ leti(x)
Leteći-Noj(x) ⇒ leti(x)
```

Kompletiranjem (paralelnim) predikata u Δ_H se dobijaju rečenice:

```
1.  Stvar(x) ⇒ Ptica(x) ∨ x=Tviti
2.  Ptica(x) ⇒ Noj(x)
3.  Noj(x) ⇒ Leteći-Noj(x)
4.  ¬ Leteći-Noj(x)
5.  Ab1(x) ⇒ Ptica(x)
```

6. `Ab2(x)` $\Rightarrow$ `Noj(x)`
7. `Ab3(x)` $\Rightarrow$ `Leteći-Noj(x)`

Iz toga se može zaključiti da Tviti ne leti jer je stvar, ali ako se izmeni tvrdnja i pretpostavi da je Tviti ptica onda se kompletirane formule o stvarima i pticama menjaju (`Stvar(x)` $\Rightarrow$ `Ptica(x)`, `Ptica(x)` $\Rightarrow$ `Noj(x)` $\vee$ `x=Tviti`) i može zaključiti da Tviti leti. Tako se sistem vremenom menja u toku samog učenja činjenica. Ovaj proces delimičnog kompletiranja u bazi naziva se razdvojenim kompletiranjem (delimited completion). Može biti korisno, opet primera radi, zaključiti da sve ptice lete osim onih za koje se ekplicitno tvrdi da ne lete. Nemonotono zaključivanje može biti i posledica nestandardnih pravila zaključivanja, pretpostavljenih (prototipnih) pravila (default rules) i pretpostavljenih teorija: $\alpha(x) : \beta(x) \rightarrow \gamma(x)$. Proširenje $\varepsilon(\Delta, D)$ baze Δ skupom pretpostsavljenih pravila D sadrži $\gamma(X_0)$ ako postoji instanca X_0 za x td. $\alpha(X_0)$ sledi iz $\varepsilon(\Delta, D)$ i $\beta(X_0)$ je konzistentna sa $\varepsilon(\Delta, D)$. Npr. $ptica(x) : leti(x) \rightarrow leti(x)$ (ovo ujedno primer *normalnih* pravila kod kojih je $\beta = \gamma$), ili PZS u odnos na predikat P:

$$\frac{:\neg P(x)}{\neg P(x)}$$

Problem sa univerzalno kvantifikovanim rečenicama sa implikacijom i izuzecima kao kod taksonomijskih hijerarhija je poznat kao *problem kvalifikacije* (Lifschitz, 1986). Zato je zgodno koristiti proceduru zaključivanja sa privremenim pretpostavkama odnosno pretpostavljenim rasuđivanjem.

5.3 Indukcija

Veoma važna osobina zaključivanja je i uopštavanje zaključivanja. Bazu znanja delimo na bazu uverenja Δ nad kojom se rade uopštavanja i pozadinsku teoriju Γ td. $\neg(\Gamma \models \Delta)$. Tada je ϕ induktivni zaključak ($\Gamma \bigcup \Delta \triangleleft \phi$) akko:

1. hipoteza je konzistenta sa pozadinskom teorijom:
   $\neg(\Gamma \bigcup \Delta \models \neg\phi)$

2. hipoteza objašnjava bazu:
   $\Gamma \bigcup \{\phi\} \models \Delta$

Indukcija je pomak od pojedinog ka opštem pod nekim uslovima, način zaključivanja opravdan induktivnom hipotezom (IH, poznat je primer Peanovog

modela prirodnih brojeva) - na primer, ako važi $P(A)$ onda važi $P(x)$, pod uslovom da nije $\neg P(x)$ (ne postoji negativan primer). Indukcija je povezana je i sa problemima (mašinskog) učenja i verovatnosnog zaključivanja. Primer može biti i problem klasifikacije činjenica prema nekim atributima i kriterijumima (kao u sistemu i algoritmu ID3 gde se generiše pravilo klasifikacija ulaznih činjenica). Primer postupka induktivnog zaključivanja je i problem *formacije koncepta*. Definiše se formalno četvorka (P, N, C, Λ) kao problem formacije koncepta gde je P skup pozitivnih instanci koncepta (potvrđuju ga), N skup negativnih, C skup svih koncepata koji se koriste da bi se definisao prostor hipoteza problema formacije (koncept daje istinitosnu vrednost za datu instancu, pozitivnu ili negativnu; konceptualni bias - PR1 zaključi moraju pripadati definisanom rečniku) i Λ je logički bias (zaključi moraju biti određene forme zadate jezikom Λ, formalnom teorijom kao vidom ugradjenog znanja) i uvodi se pojam prihvatljve relacije (ako zadovoljava biase, definisana nad C u jeziku Λ). Prihvatljiva relacija je karakteristična ako zadovoljena za sve iz P, diskriminanta ako ne zadovoljava nijednu iz N i dopustljiva je ako zadovoljava oba uslova. Skup svih dopustljivih relacija je skup verzija V, a graf verzija u kojem su orijentisani lukovi relacijom *opštosti* - čvor p je *manje opšti* od čvora q ako je $p \subseteq q$ (ispravni podskup relacije kao skupa, ili više specifičan). Skup V je dobro formiran ako za svaki lanac u grafu postoji minimalni i maksimalni element, S skup (specifična granica) minimalnih a G (generalna, uopštena granica) maksimalnih elemenata. Tada važi:

Teorema 8 *Za (P, N, C, Λ) sa dobro formiranim V i S, G skupovima tada $r \in V$ akko je ograničena elementima iz S i G.*

Postoji postupak *eliminacije kandidata* kojim se za svaku (pozitivnu i negativnu simetrično) instancu tj. uneti podatak prepravljaju skupovi G i S (umesto celog prostora V) td. je pokrivena nova činjenica. Algoritam dovodi do $S = G$ tj. ostaje samo jedna instanca u V. Prethodna teorema (kao i sam postupak i njegove osobine) garantuje rešenje i to u konačnom broju koraka.

Zavisno od prirode problema neki put je moguće uticati na izbor naredne instance i tražiti informacije o njenoj klasifikaciji - moguće je vršiti *eksperimente*. Ovo nudi mogućnost dodatnog poboljšavanja postupka. Osnovni tip poboljšanja je npr. izbor instance koja će prepoloviti prostor verzija, ali često samo traženje takve instance može da bude zahtevno samo po sebi. Ako se definiše proizvod prostora verzija, a time i faktorizacija, moguće je dobiti bolje rezultate i varijante algoritma. Pored formacije koncepta nezavisne od domena postoje i sistemi kao što su to npr. Meta-DENDRAL ili ID3 koji su ,,model-driven" tj.

koji su manje ili više zavisni od domena jer pretpostavljaju da su svi podaci na raspologanju na samom početku, dok su ostali inkrementalni (data-driven). Više o ovome i o nemonotonom zaključivanju se može naći u [GN].

5.4 Verovatnosno zaključivanje

Potrebno je povezati pojam iskaza sa pojmom slučajne promenljive tako što će svaki iskaz imati distribuciju slučajne promenljive sa dve vrednosti {1-p,p}. Tako atom P (događaj) je istinit sa verovatnoćom p, a $\neg P$ sa verovatnoćom $1-p$. Sa dva konstantna atoma možemo formirati raspodelu višedimenzionalne slučajne promenljive (za složene događaje $\{P,Q\}$, $\{P,\neg Q\}$, $\{\neg P,Q\}$, $\{\neg P,\neg Q\}$) i njihove verovatnoće:

$p(P \wedge Q) = p_1$
$p(P \wedge \neg Q) = p_2$
$p(\neg P \wedge Q) = p_3$
$p(\neg P \wedge \neg Q) = p_4$

tada su verovatnoće događaja odnosno konstantnih atoma P i Q verovatnoće marginalnih raspodela takve višedimenzionalne slučajne promenljive za $\{P,Q\}$:

$p(P) = p_1 + p_2 = \sum_i p(P|Q = Q_i)$
$p(Q) = p_1 + p_3 = \sum_i p(Q|P = P_i)$

Najčešće nisu date složene verovatnoće i bez njihove distribucije je teško o njima znati dovoljno na osnovu distribucije marginalnih promenljivih. Tako se Bajesovo pravilo može upotrebiti slično modus ponensu: ako je $p(Q|P)$ uslovna verovatnoća događaja Q ako je P ispunjeno. To je deo slučajeva za koje je P ispunjeno kada je i Q ispunjeno: $p(Q|P) = \frac{p_1}{p_1+p_2} = \frac{p(P,Q)}{p(P)}$, $p(P,Q) = p(P \wedge Q)$. Obrnuto, $p(P|Q) = p(P,Q)/p(Q)$ i odatle sledi:

$$p(Q|P) = \frac{p(P|Q)p(Q)}{p(P)}$$

Dakle, Bajesovo pravilo nudi mogućnost da se zaključi nešto i o uzroku na osnovu posledice.

$$p(\neg Q|P) = \frac{p(P|\neg Q)p(\neg Q)}{p(P)}$$

$$\frac{p(Q|P)}{p(\neg Q|P)} = \frac{p(P|Q)p(Q)}{p(P|\neg Q)p(\neg Q)}$$

$$O(E) =_{def} \frac{p(E)}{p(\neg E)} = \frac{p(E)}{1-p(E)}$$

(,,izgledi za E") pa ako je (faktor dovoljnosti) $\lambda =_{def} \frac{p(P|Q)}{p(P|\neg Q)}$ i (faktor potrebnosti) $\overline{\lambda} =_{def} \frac{p(\neg P|Q)}{p(\neg P|\neg Q)}$ onda je:

$$O(Q|P) = \lambda O(Q),\ O(Q|\neg P) = \overline{\lambda} O(Q)$$

Postoji povezanost vrednosti λ i $\overline{\lambda}$:

$$\overline{\lambda} = \frac{1 - \lambda p(P|\neg Q)}{1 - p(P|\neg Q)}$$

ali su obe neophodne da bi se našla uslovna verovatnoća za Q ako je P ili $\neg P$ posebno. Pošto je je $0 < p(P|\neg Q) < 1$, ako je $\lambda < 1$ onda je $\overline{\lambda} > 1$ i obratno, kao i $\lambda = 1$ akko $\overline{\lambda} = 1$. O tome treba voditi računa prilikom građenja baze znanja. Često se koriste logaritmi ovih koeficijenata $l = \log \lambda$ koji se nazivaju *indeksi dovoljnosti* (što je veći to je i $p(Q|P)$ veće) i $\overline{l} = \log \overline{\lambda}$ indeks potrebnosti (što je manji to je i $p(Q|\neg P)$ manje). Takođe, važi veza između $p(Q)$ i $O(Q)$:

$$p(Q) = O(Q)/(O(Q) + 1)$$

Na osnovu ovoga svega, ako je poznato $p(Q)$ i ako se pretpostavi P ili $\neg P$ onda se može izračunati uslovna verovatnoća za Q. Ekspertni sistemi (rule-based) koriste bazu znanja u kojima se nalaze i pravila oblika $P \rightarrow Q$ da Q može slediti iz P. U PR1 to znači da se može zaključiti Q uz to pravilo ako je P istinito, ali u verovatnosnom zaljučivanju to nije tako, ili bar nije jednostavno doći do verovatnoće $p(Q)$ uz $p(P \Rightarrow Q)$ pored $p(P)$, ali ako uz pravilo se veže i njegovo λ i $\overline{\lambda}$ onda je to moguće. A ako se sa P' izrazi nesigurnost u pretpostavku P (tj. $\neg P$) i sa $p(P|P')$ verovatnoća da je P onda se može pretpostaviti da je $p(Q|P, P') = p(Q|P)$ i $p(Q|\neg P, P') = p(Q|\neg P)$ (P i P' su zavisne u tom smislu) i važi:

$$p(Q|P') = p(Q, P|P') + p(Q, \neg P|P') = p(Q|P, P')p(P|P') + p(Q|\neg P, P')p(\neg P|P')$$

gde je onda $p(Q|P')$ linearna interpolacija verovatnoće između kranjih vrednosti da je P tačno ili nije znajući verovatnoću da je P. Zanimljivo, ako je

$p(P|P') = p(P)$ onda je $p(Q|P') = p(Q)$ - gubi se informacija o uticaju P' na Q. Slično prethodnom, ako su $\{P_1, ..., P_n\}$ hipoteze koje su uslovno nezavisne (jaka pretpostavka, može se samo opravdati samo do izvesne mere, aproksimativno), onda se verovatnoća zaključka Q može izračunati, kao i da se uslove verovatnoće za P_i nekakvim uverenjima P_i'. Tada uz pomenutu pretpostavku i pretpostavku da su obzervacije P_i' nezavisne od P_j osim odgovarajuće ,,svoje" P_i, i da Q ne zavisi dodatno od P_i', važi:

$$p(Q|P_2', P_1') = p(Q|P_2, P_1')p(P_2|P_2') + p(Q|\neg P_2, P_1')p(\neg P_2|P_2')$$

gde je $O(Q|P_2, P_1') = \lambda_2 O(Q|P_1')$ i $O(Q|\neg P_2, P_1') = \overline{\lambda_2} O(Q|P_1')$. Tu se naslućuje iterativni postupak u kome se koristi prethodno izračunato $O(Q|P_1')$ gde se za svako P_i vezuje odgovarajući par λ_i i $\overline{\lambda_i}$.

Tako se mogu graditi ***mreže zaključivanja*** (inference networks) - npr. ako su P_1, P_2, P_3, P_4 uslovno nezavisne, A zavisi od P_1, P_2 i B zavisi od P_3, P_4 onda su i A i B uslovno nezavisne i zaključak Q_f koji sledi iz A, B zavisi od njih. Mnogi ekspertni sistemi ih koriste. Zaključivanje unapred (forwad-chaining) propagiranjem pravila nad činjenicama sve do zaključka nalazi verovatnoću zaključka u mreži. Zaključivanje unazad (backward-chaining - sličan mehanizam, ,,forward-propagation" i ,,back-propagation", postoji kod nekih klasa neuronskih mreža kao što je perceptron, gde se takoreći menjaju koeficijenti pravila na osnovu početnih pretpostavki, izračunatog i zadatog zaključka) npr. analizira drvo mreže zaključivanja tražeći početnu pretpostavku koja najviše utiče na zaključak - onda se interaktivno unosi verovatnoća takvih pretpostavki ako je potrebno dok se ne potvrdi uticaj na zaključak. Problem je ako neki od međuzaključaka zavisi od nekih drugih međuzaključaka iako se pretpostavlja da su nezavisni iz bilo kog razloga. To se rešava obično dodatnim ad hoc mehanizmima i podešavanjima. Ako imamo pravilo oblika $P_1 \wedge ... \wedge P_n \rightarrow Q$ onda treba najpre izaračunati zavisnu verovatnoću za $P = P_1 \wedge ... \wedge P_n$, iskaz koji nije atom - npr. neki ekspertni sistemi koriste $p(P) = \min_i[p(P_i)]$ ili $p(P_1 \vee ... \vee P_n) = max_i[p(P_i)]$ iako bi uz pretpostavku da su P_i nezavisne verovatnoća konjunkcije bila manja od navedenog minumuma, ali u kranjem slučaju gde sve verovatnoće imaju vrednosti 0 ili 1 i jedno i drugo se svodi na Bulovu algebru (koja se poklapa sa fuzzy teorijom skupova u ovakvom specijalnom slučaju - Zadeh, 1965-1975 - svakom elementu i podskupu dodeljena je funkcija koja meri pripadnost skupu, što bi moglo da se tumači kao verovatnoća, ali to onda nije fuzzy

teorija u opštem slučaju).

5.5 Jedno formalno zasnivanje verovatnosne logike

Formalno zasnivanje verovatnosne logike se vezuje za formalno zasnivanje slučajnih promenljivih i verovatnoće. Ako je rečenica ϕ u svetu $\mathcal{W}_1$ istinita, a u svetu $\mathcal{W}_2$ nije, pošto ne znamo u kojem od ta dva sveta zaista jeste („u stvarnom svetu može biti samo u jednom od ta dva") to se izražava verovatnoćom p da pripada $\mathcal{W}_1$, odnosno $1-p$ da pripada $\mathcal{W}_2$. Ako imamo više reči onda ima i više kombinacija svetova u kojima su tačne, ali npr. za $\phi_1 \wedge \phi_2$ ne uzimaju se u obzir svetovi gde je ϕ_1 tačno kao i $\phi_1 \wedge \phi_2$, a ϕ_2 nije. Za skup rečenica Γ je moguće napraviti ***semantičko drvo*** i tako odrediti moguće svetove - svaka rečenica može biti tačna ili ne (pozitivni ili negativni literal je tačan), na svakom nivou po jedna iz Γ, i od korena (prve rečenice) do lista (zadnje rečenice) postoje putevi koji daju konzistentne skupove (kombinacije), ostali se odbacuju (može se prikazati tabelarno - ima ih praktično $2^{2^{broj\ slova}}$ a ne $2^{|\Gamma|}$, gde su slova iskazne promenljive, odnosno osobine za koje se vezuju elementarni događaji). Verovatnoća rečenica je onda zbir verovatnoća svetova u kojima je tačna. Neka ima K nepraznih skupova u $\mathcal{K} = \{\mathcal{W}_i\}$ mogućih svetova za L rečenica iz Γ, i ako su nabrojani, neka je onda $\mathbf{P}$ kolona dimenzije K veorvatnoća $[p_i]^T$ vezanih za određeni skup svetova $\mathcal{W}_i$. Neka su rečenice ϕ_j u Γ nabrojane, L-dimenzioni vektori $\mathbf{V}_1, ..., \mathbf{V}_K$ odgovaraju konzistentnim valuacijama rečenica u Γ td. i-tom skupu svetova $\mathcal{W}_i$ odgovara $\mathbf{V}_i = [v_{ji}]^T$ gde je:

$$v_{ji} = \begin{cases} 1, & \phi_j \text{ tačna u } \mathcal{W}_i \\ 0, & \phi_j \text{ netačna u } \mathcal{W}_i \end{cases}$$

Neka je onda $L \times K$ matrica $\mathbf{V} = [\mathbf{V}_1, ..., \mathbf{V}_K]$. Ako je L-dimenziona kolona $\Pi = [\pi_j]^T$ verovatnoća rečenica ϕ_j iz Γ onda je:

$$\Pi = \mathbf{VP}$$

uz uslov $\sum_i p_i = 1$ za $0 \leq p_i \leq 1$ (*). Ako je Δ skup uverenja (belief) - rečenica sa njihovim poznatim π_i verovatnoćama, verovatnosna derivacija (probabilistic entailment) rečenice ϕ iz Δ se može svesti na problem rešavanja sistema linearnih nejednačina gde je $\Gamma = \Delta \bigcup \{\phi\}$ i V je dobijeno npr. semantičkim drvetom - ova metoda se može proširiti Skolemizacijom i na

PR1. Γ se može proširiti sa $\top$ i V sa jednim redom (npr. prvi red) za tu formulu da bi se dodao i uslov (*). Ako se ϕ doda kao poslednji red $\varphi = [v_{\phi i}]$ u $\mathbf{V}$ tada je:

$$\Pi = \begin{bmatrix} 1 \\ \vdots \\ \pi_j \\ \vdots \\ \pi_\phi \end{bmatrix} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ v_{11} & v_{12} & \cdots & v_{1K} \\ \vdots & & \ddots & \\ v_{L1} & v_{L2} & \cdots & v_{LK} \\ v_{\phi 1} & v_{\phi 2} & \cdots & v_{\phi K} \end{bmatrix} \mathbf{P}$$

Ako se sa $\mathbf{V}'$ označi $\mathbf{V}$ bez poslednjeg reda (za ϕ) i sa Π' kolona bez zadnjeg člana π_ϕ u Π, tada se najpre rešavanjem po $\mathbf{P}$ sistema $\Pi' = \mathbf{V}'\mathbf{P}'$ dobija $\pi_\phi = p(\phi) = \varphi\mathbf{P}$. Sistem najčešće ima mnogo rešenja i zato je interesantno naći interval u kome se kreće rešenje.

Primer: $\Delta = \{(\exists y)P(y), (\forall x)P(x) \Rightarrow Q(x)\}, \varphi = (\exists z)Q(z)$ (npr. prvi nivo semantičkog drveta ima dve grane: $P(A)$ i $\neg P(y)$, sledeći je naredna formula u Δ i njena negacija, i treći je φ i njena negacija - za svaki list onda imamo kolona nula i jedinica za formule i negacije po nivoima). Tada se može pokazati da je:

$$p((\exists y)P(y)) + p((\forall x)P(x) \Rightarrow Q(x)) - 1 \leq p((\exists z)Q(z)) \leq 1$$

Ovo se može dobiti tako što se 2. i 3. jednačina za V' saberu i od toga se oduzme prva odakle se dobije da je $p_1 = \pi_1 + \pi_2 - 1$ i na osnovu toga $p_3 = \pi_1 + \pi_2 - 1 + p_2 + p_4$ (3 jednačine, pošto je zadnja ona kojom se računa verovatnoća φ, i 4 nepoznatih p_i). Ovaj diedar kada se preseče jediničnom kockom $[0,1]^L$ u prostoru vektora Π kao slika vektora $P \in [0,1]^K$ je konveksna oblast (kao slika konveksne oblasti linearnim preslikavanjem V) i predstavlja oblast gde su verovatnoće π_j ***konzistentne*** i u opštem slučaju je poliedar omeđen hiperravnima u L-dimenzionom prostoru (i ima K temena). Ovo se može i geometrijski shvatiti i pokazati: jedinične vektore $\mathbf{V}$ (kao operator) slika u temena te oblasti koja su zapravo kolone matrice $\mathbf{V}$. Dakle, treba u sistemu ovakvog verovatnosnog zaključivanja voditi o tome računa (npr. ako neka verovatnoća nije u toj oblasti onda se menjanjem ostalih parametara dovede u tu oblast). Kanonizacijom $\mathbf{V}$ i drugim standardnim postupcima se može optimizovati rešavanje ovakvih sistema. Ovo, naravno, ima smisla samo ako je matrica $\mathbf{V}$ dovoljno mala (pa se mogu primeniti metode linearnog programiranja), što u praksi nije čest slučaj i zato se koriste aproksimativne

metode. Često se koriste intervali verovatnoća, kako je pomenuto, u smislu gornjih i donjih granica verovatnoća umesto jedinstvenih vrednosti.

5.6 Znanja i uverenja

Već je pomenuta razlika između znanja u smislu baze znanja u vezi PR1 i uverenja u smislu verovatnosnog zaključivanja (derivacije). Npr. inteligentni agent sa svojom konceptualizacijom tako raspolaže uverenjima pre nego znanjem, jer uvek mora da zadrži mogućnost da neka informacija ili zaključak nisu tačni. Š aviše, nije dovoljno pretpostaviti da agent veruje logičkom zatvorenju svoje baze znanja već se najčešće koristi i prikladnije je da agent veruje onim rečenicama koje može da zaključi u zadatom vremenskom roku uz zadatu proceduru zaključivanja.

5.6.1 Iskazna logika uverenja

(Sentential Logic of Belief)

Konstruiše se proširenje PR1 na sledeći način - ispravne reči su:

1. PR1 ispravne reči
2. Atomi uverenja: ako je ϕ obična PR1 zatvorena formula i α je konstantni term, onda je $\mathbf{B}(\alpha, \phi)$ ispravna reč ($\mathbf{B}_\alpha(\phi)$)
3. Ako su ψ i ϕ ispravne reči onda su iskazne formule nad njima takođe ispravne reči

Agent a je određen svojim skupom pravila zaključivanja ρ_a i skupom uverenja δ_a. Tada je teorija T_a zatvorenje δ_a svojim pravilima zaključivanja ($P \in T_a$ akko $\delta_a \vdash_a P$). Zaključivanje $\vdash_{a,b}$ sa ugnježdenim verovanjem se zasniva na modelu u b zaključivanja u a. Skolemova konstanta se alocira i obeležava simbolom `Sk`. Problem nastaje kada ta Skolemova konstanta ide i unutar modalnog kvantifikatora - agent („vernik") na osnovu svojih uverenja i zaključivanja u svom modelu može izabrati odgovarajuću konstantu koja uopšte nemora biti ista. Zbog toga se uvodi pomoćni operator koji se obeležava sa $\bullet$ („metak") i piše se ispred (zavisne) Skolemove konstante pod kvantifikatorom - npr. $\exists Q(x) \wedge B(A, P(x))$ postaje $Q(Sk) \wedge B(A, P(\bullet Sk))$. Obični kvantifikatori mogu da šetaju unutar i izvan modalnog kvantifikatora $\mathbf{B}$. Slično, supstitucija

podizraza logički ekvivalentnom podizrazom u formuli ispod modalnog kvantifikatora nije dozvoljena kao što je to moguće kod iskaznih operatora. Može se uvesti dodatno skup javnih konstanti C koje specijalno imaju istu vrednost među svim agentima po definiciji.

Koristi se pravilo zaključivanja priključenja šeme („schema attachment", slično rezoluciji - koriste se klauzule, ali ne transformišu se formule ispod modalnih kvantifikatora):

Iz:

$\mathbf{B}(\alpha, \phi_1) \vee \psi_n$
$\mathbf{B}(\alpha, \phi_2) \vee \psi_2$
$\vdots$
$\mathbf{B}(\alpha, \phi_n) \vee \psi_n$
$\neg\mathbf{B}(\alpha, \phi_{n+1}) \vee \psi_{n+1}$
$\phi_1 \wedge \cdots \wedge \phi_n \vdash_\alpha \phi_{n+1}$

Sledi:

$\psi_1 \vee \cdots \vee \psi_{n+1}$

Jedna definicija modalnog kvantifikatora *znanja* **K** glasi $K_\alpha(\phi) \equiv B_\alpha(\phi) \wedge \phi$ (nemože neko znati nešto što nije tačno). Da bi se formalnije definisala semantika modalnih kvantifikatora potrebno je uvesti pojam *logike mogućih svetova*. Svetovi w_0, w_1, ..., w_i mogu biti apstraktne alternative kao konceptualizacije znanja (u smislu ranije pomenutih konceptuazicija koje mogu biti brojevi ili neki drugi objekti, manje bitno), ... i relacija dostupnosti (accessibility) sveta w_j iz sveta w_i za agenta α: $k(\alpha, w_i, w_j)$, gde interpretacija za datu konceptualizaciju može imati slike u bilo kojem od svetova. U jednom svetu neki agent može znati neku činjenicu, a u drugom ne. Semantikom mogućih svetova se objašnjavaju konstrukcije iskazne logike verovanja. Kaže se da atom znanja $\mathbf{K}(\alpha, \phi)$ ima istinitu vrednost za svet w_i akko je ϕ istinita za svaki svet dostupan iz w_i, a time se namerava značenje da agent α *zna* formulu ϕ. To se može dalje rekurzivno primenjivati. Šeme aksioma i pravila zaključivanja za rad sa **K** (formalna sintaksna definicija):

- (A1) $K_\alpha(\phi) \wedge K_\alpha(\phi \Rightarrow \psi)) \Rightarrow K_\alpha(\psi)$

ili ekvivalentno: $K_\alpha(\phi \Rightarrow \psi) \Rightarrow K_\alpha(\phi) \Rightarrow K_\alpha(\psi)$
(aksioma distribucije)

- (A2) $K_\alpha(\phi) \Rightarrow \phi$
(aksioma znanja, dostupnost je refleksivna)
- (A3) $K_\alpha(\phi) \Rightarrow K_\alpha(K_\alpha(\phi))$
(pozitivna introspekcija - dostupnost je tranzitivna relacija)
- (A4) $\neg K_\alpha(\phi) \Rightarrow K_\alpha(\neg K_\alpha(\phi))$
(negativna introspekcija, dostupnost je euklidska tj.
$k(\alpha, w_1, w_2) \wedge k(\alpha, w_1, w_3) \Rightarrow k(\alpha, w_2, w_3)$)
- pravilo (P1) (epistemična obaveznost, ,,epistemic necessity"):
ako $\vdash \phi$ onda je $K_\alpha(\phi)$
- pravilo (P2):
ako $\phi \vdash \psi$ i $K_\alpha(\phi)$ onda je $K_\alpha(\psi)$
- pravilo ekvivalentno prethodnom (P2'):
ako $\phi \vdash \psi$ onda je $K_\alpha(\phi) \Rightarrow K_\alpha(\psi)$

Iz (P2) npr. se može zaključiti distributivnost (K) nad konjunkcijom. Pripadnost aksioma jw međusobno uslovljene osobinama dostupnosti (RST). Dodatne osobine verovanja:

- (A5) $\neg \mathbf{B}_\alpha(F)$
- (A6) $\mathbf{B}_\alpha(\phi) \Rightarrow \mathbf{B}_\alpha(\mathbf{B}_\alpha(\phi))$
- (A7) $\mathbf{B}_\alpha(\phi) \Rightarrow \mathbf{K}_\alpha(\mathbf{B}_\alpha(\phi))$
- (A8) $\mathbf{B}_\alpha(\mathbf{B}_\alpha(\phi)) \Rightarrow \mathbf{B}_\alpha(\phi)$
- (A9) $\mathbf{B}_{\alpha_1}(\mathbf{B}_{\alpha_2}(\phi)) \Rightarrow \mathbf{B}_{\alpha_1}(\phi)$

Koriste se grupe agenata i dodatni modalni operatori (pomoćni): za konačnu grupu G $\mathbf{IK}(G, \phi)$ znači da grupa G ima implicitno znanje o ϕ akko postoji skup $\{\phi_i\}$ td. $\{\phi_i\} \vdash \phi$ i za svako ϕ_i postoji agent $A_k \in G$ td. $\mathbf{K}(A_k, \phi_i)$. Dalje, neki agent iz G zna ϕ:

$$\mathbf{SK}(G, \phi) \equiv \bigvee_{A_i \in G} \mathbf{K}(A_i, \phi)$$

Svaki agent zna:

$$\mathbf{EK}(G,\phi) \equiv \bigwedge_{A_i \in G} \mathbf{K}(A_i,\phi)$$

$$\mathbf{EK}^{k+1}(G,\phi) \equiv \mathbf{EK}(\mathbf{EK}^{k}(G,\phi))$$

Opšte znanje u grupi:

$$\mathbf{CK}(G,\phi) \equiv \phi \wedge \mathbf{EK}(G,\phi) \wedge \mathbf{EK}^2(G,\phi) \cdots$$

Iako je beskonačna konjunkcija, **CK** se može koristiti slično kao i **K** u aksiomama, kao i pravilo zaključivanja: ako $\vdash \phi$ onda $\mathbf{CK}(\phi)$ (G se podrazumeva).

5.7 Meta-znanje i meta-zaključivanje

Potrebno je neki put znati kako se došlo rešenja i obrazložiti to na prihvatljiv način a ne samo naći rešenje. Prethodno poglavlje i opisana modalna logika se ovde tretiraju kao polazni domen nad kojim se gradi formalna konceptualizacija i PR1 rečnik kojim se opisuje formalno zaključivanje nad ovakvim domenom. Kao posledica toga mogu se konstruisati agenti koji su u stanju da zaključuju o verovanjima i zaključivanju drugih agenata (ili se može dodatno optimizovati njihov proces zaključivanja). Introspekcija je osobina agenta da samog sebe objašnjava tj. svoje zaključke i verovanja.

Meta-jezik kojim će ovaj koncept biti opisan je PR1 (formalnim jezikom se opisuje sličan formalni jezik). Koriste se znaci navoda da bi se time naznačilo da je reč o simbolu kao meta-objektu - npr. Kratko("Osoba") može značiti da je simbol Osoba kratak (do 5 karaktera) a Visoka(Osoba) može biti rečenica jezika kojom se utvrđuje (c)injenica o osobi a "Visoka(Osoba)"je izraz kojim je to zapisano (mada se izraz ne tretira kao niska karaktera već ima PR1 strukturu). Koristi se klauzalna forma (kao liste literala ...), a kao mehanizam zaključivanja rezolucija (formalno se zapisuju meta-jezikom rečenice kojima se definišu predikati (pretpostavljaju se predikati `Objconst`, `Variable`, `Funconst`, `Relconst`, ,,." je operator konkatenacije elementa i liste) `Costant`, `Term`, `Termlist` (uz implicitnu definiciju člana liste `Member`), `Funexpr`, `Atom`, `Literal`, `Clause`, `Database`, npr.:

$\forall$`x Constant(x)` $\Leftrightarrow$ `Objconst(x)` $\vee$ `Funconst(x)` $\vee$ `Relconst(x).`
$\forall$`x Term(x)` $\Leftrightarrow$ `Objconst(x)` $\vee$ `Variable(x)` $\vee$ `Funexpr(x)`

```
∀l Termlist(l) ⇔ (∀x Member(x,l) ⇒ Term(x))
∀f ∀l Funexpr(f.l) ⇔ (Funconst(f) ∧ Termlist(l))
∀f ∀l Atom(r.l) ⇔ (Relconst(x) ∧ Termlist(l))
∀x Literal(x) ⇔ (Atom(x) ∨ (∃z x="¬<z>"∧ Atom(z)))
∀c Clause(c) ⇔ (∀x Member(x,c) ⇒ Literal(x))
(∀d Database(d) ⇔ (∀x Member(x,d) ⇒ Clause(x))
```

Za potrebe unifikacije i rezolucije definišu se `Subst`, `Extend`, `Combine`, `Mgu` (Nou, „Most General Unifier"), `Resolvent`, `Among`, `Append`, `Delete`), npr.:

```
∀x Subst(x,[ ])=x
∀x∀s Constant(x) ⇒ Subst(x,s)=x
∀x∀z∀s Variable(x) ⇒ Subst(x,(x/z).s)=z
∀x∀y∀z∀s Variable(x) ∧ y ≠ z ⇒ Subst(x,(y/z).s)=Subst(x,s)
∀x∀l∀s Subst(x.l,s)=Subst(x,s).Subst(l,s)
∀x∀z Extend([ ],x,z)=[x/z]
∀u∀v∀x∀z∀s Extend((u/v).s,x,z)=(u/Subst(v,[x/z])).Extend(s,x,z)
∀s Combine(s,[ ])=s
∀s∀t∀x∀z Combine(s,(x/z).t)=Combine(Extend(s,x,z),t)

∀x Mgu(x,x,[ ]) ∀x∀y Variable(x) ∧ ¬Among(x,y) ⇒ Mgu(x,y,[x/y])

∀x∀y ¬Variable(x) ∧ Variable(y) ∧ ¬Among(y,x) ⇒ Mgu(x,y,[y/x])

∀x∀y∀l∀m∀s∀t Mgu(x,y,s) ∧ Mgu(Subst(l,s),Subst(m,s),t)
⇒ Mgu(x.l,y.m,Combine(s,t))

∀x∀y∀s Mgu(x,y,s) ⇔ Resolvent(x.l,"¬<y>".m,Subst(Append(l,m),s))

∀c∀d∀x∀y∀s (Member(x,c) ∧ Member("¬<y>",d) ∧ Mgu(x,y,s)) ⇔
Resolvent(c,d,Subst(Append(Delete(x,c),Delete("¬<y>",d)),s))
```

Procedura zaključivanja opisuje se Markovljevom funkcijom koja slika bazu u bazu naslednika (`Concs(c,d)` daje sve rezolvente klauzule i baze, i ako se koriste samo Hornove klauzule i upiti samo kao konjunkcije pozitivnih literala onda važi `Next(d)=Append(Concs(Car(d),d),Cdr(d))`):

```
∀d Step(d,1)=d
```

∀d∀n n>1⇒Step(d,n)=Next(Step(d,n-1))

Definiše se dalje `Derivable(d,r)` (da li r proizilazi iz d po rezoluciji), `Provable` je onda dokaz odbacivanjem (refutation) tako da se zaključi [ ] (prazna klauzula). Konačno, ako `Data` mapira agenta u listu rečenica (njegovu bazu), predikat `Bel` ima značenje modalnog kvanitifikatora verovanja:

∀d∀r Derivable(d,r) ⇔ Member(r,d) ∨
(∃p∃q Derivable(d,p) ∧ Derivable(d,q) ∧ Resolvent(p,q,r))

∀d∀p Derivable(d,p) ⇔ (∃n Member(p,Step(d,n)))
∀d∀p Provable(d,p) ⇔ Derivable(Append(Clauses("¬<p>"),d),[ ])

∀a∀p Bel(a,p) ⇔ Provable(Data(a),p)

Osnovna prednost ovakvog pristupa formalnog definisanja metaznanja je mogućnost odgovora na pitanja o procesu zaključivanja - meta-zaključivanje.

Neophodno je u ranije opisani mehanizam dodati nekoliko izmena da bi radio na takav način - predikati koji potrđuju promenljive i konstante definišu se programskim priključenjem (npr. `Variable("v")` vraća da v jeste promenljiva ako jeste i briše klauzulu iz baze jer ne može da učestvuje u rezoluciji dalje, inače je netačan literal i briše se iz klauzule). Takođe, modifikuje se algoritam unifikacije da bi se upoređivali i izrazi pod navodnicima i pošto je ravnopravan zapis u obliku liste sa izrazom pod navodnicima - npr. za `"P(A,B)"` i `["P",x,"B"]` dobija se lista vezivanja `[x/"A"]`.

Osnovno (baselevel) i meta (metalevel) zaključivanje su mononivoovska (monolevel) jer koriste isključivo rečenice jednog tipa. Dvonivoosko zaključivanje (bilevel) sadrži rečenice oba tipa. Rečenice nemogu da budu i jednog i drugog tipa niti meta-meta tipa. Ako se pretpostavi da baza Ω može da se razdvoji na `data(Ω,1)` osnovni i `data(Ω,2)` meta nivo, i ako se dva nivoa posebno obrađuju onda je `next(Ω)=append(next(data(Ω),2)),next(data(Ω,1)))`.

Međutim, u realnosti je potrebna veza između dva nivoa jer meta-nivo utiče na osnovni. Ako Markovljeva funkcija `next` vraća bazu prema pravilima u `data(Ω,2)` ako ih ima i nikada ne vraća bazu koja nije po pravilima u `data(Ω,2)` onda je ***introspektivno verna*** u bazi Ω. Ne postoje procedure

zaključivanja koje su introspektivno verne nad svim bazama (teorema - dokaz sledi iz primera koji sadrži kontradikciju u metabazi). Metabaza je *introspektivno kompletna* ako propisuje svaki korak zaključivanja osnovne baze (svako pravilo zaključivanja nad osnovnom bazom potiče od metabaze). Ako se pri svakom koraku rezolucijom iz metabaze dobija pravilo u kojem se zaključi nad osnovnom bazom i dobijeni zaključak osnovnog tipa pridruži bazi, ovakvo zaključivanje dvonivoovska baze se zove *kompulsivna introspekcija*. Pokazuje se da je kompulsivna introspekcija introspektivno verna za svaku konzistentnu i introspektivno kompletnu bazu.

Proces u kome se prekida trenutni način zaključivanja, u kome se zaključuje o zaključivanju i utiče (menja mehanizam) na zaključivanje zove se ***refleksija***. Proces zaključivanja na jednom ili više nivoa višenivoovkse baze (uzima se najniži nivo k u obzir) može da promeni bazu (`data(next(Ω),k)` $\neq$ `data(Ω,k)`). Proces zaključivanja je refleksivan akko obuhvata više nivoa. Interesantno je kada postoji veza između različtih nivoa. Npr. *Unverzalno pravljenje podciljeva* (universal subgoaling) se definiše tako da ako u ranijoj definiciji `next` funkcije imamo 5 ili manje zaključaka ostaje kako jeste, inače se primenjuje refleksija (preveliki broj ili nedostatak zaključaka okidaju refleksiju u kojoj se npr. postavlja negacija cilja i dodaje prikladan skup rečenica Θ) `next(Ω)=reflect(Ω)=[Next("Δ")` $\neq$ `d, Ans(d)].Θ` - ovaj skup Θ npr. definiše i primeni `Add` i `Order` kojim se ,,prerasporede" klauzule po broju zaključaka td. manja ide ispred. Postoji efikasnija varijanta je da se Θ doda metabazi i da se koriste dužine klauzula (broj literala) umesto broja zaključaka.

Kompulsivna refleksija - za svaki korak se konsultuje metabaza i pri tome se, za razliku od kompulsivne introspekcije, zaključuje i o metabazi:

```
newmeta(Δ)=[Next("Δ") ≠ d, Ans(d)]

Ans("Δ") ∉ Ω →
  next(Ω)=append(concs(car(data(Ω,2)),data(Ω,2)),data(Ω,1))

Ans("Δ") ∈ Ω →
  next(Ω)=append(data(Ω,2)-answers(data(Ω,2)), newmeta(Δ), Δ)
```

Traži se zaključak o `next` funkciji i kada se nađe odgovor na osnovu pravila metabaze, baza se čisti od `Ans` literala i dodaje se odgovor osnovnoj bazi i time je spremna za sledeći korak.

6 Stanje i akcije

6.1 Stanja

Pojam stanja je osnovni pojam u konceptualizaciji fizičkog sveta. Stanje, ili situacija, je snimak sveta u datom trenutku. U različitim trenucima svet može biti u različitim stanjima. Ova ideja je lepše ilustrovana u kontekstu mikrokosmosa kakav je poznati primer *Sveta blokova*. Posmatrajmo varijaciju ovog sveta u kojem postoje samo tri kutije. Svaka kutija može biti negde: na tabli ili na vrhu, iznad samo jedne kutije. Različita stanja odnose se na različite konfiguracije kutija(blokova). Ilustrujmo primer:

Označimo sa `S1` i `S2` objekte prostora pretrage. *Oznaka stanja* (state designator) će označavati stanja tih objekata. Da bismo označili da je objekat stanje koristimo konstantu unarne relacije (unary relation constant) `State(S1)`.

Najjednostavniji način da se opiše stanje je da se koriste jednostavnije funkcije ili relacije za svaku vrstu informacija vezane za stanje.
Primer:

```
On(B,A,S1)
Clear(C,S1)
Clear(B,S1)
Table(A,S1)
```

itd.

Postoji alternativa ovom pristupu u kojem predstavljamo svojstva koja zavise od stanja kao funkcije koje ne zavise od stanja. Te funkcije preslikavaju objekte u skupove stanja u kojima ti objekti imaju asocirana obeležja zavisnih stanja. Na primer, koristimo `On(A,B)` kao konstantu binarne funkcije (binary function constant) koja označava skup stanja u kojima je blok A na bloku B. Ovaj term, `On(A,B)`, nazivamo *deskriptor stanja* (state descriptor), a skup stanja koje određuje nazivamo *fluentom*. Da bismo sada zapisali rečenice

koje zavise od stanja koristimo konstantu binarne relacije T (binary relation constant),

```
T(On(C,A),S1)
```

koja se interpretira kao ,,Tačno je da je C na A u stanju S1".

Pošto deskriptori stanja označavaju skupove stanja, možemo govoriti o kompoziciji deskriptora stanja koji predstavljaju komplement, uniju i presek tih skupova. Možemo pisati takve kompozicije sa običnim skupovnim o -peratorima, ali obično koristimo simbole identične logičkim operatorima da naglasimo da deskriptori predstavljaju osobine stanja. Koristimo operator negacije za komplement, konjukciju i disjunkciju za $\cap$ i $\cup$ i implikaciju da bismo izrazili da je uzrok podskup posledice. Primer:

$\forall$p $\forall$s T($\neg$p,s) $\Leftrightarrow$ $\neg$T(p,s)
$\forall$p $\forall$q $\forall$s T(p$\wedge$q,s) $\Leftrightarrow$ (T(p,s) $\wedge$ T(q,s))
$\forall$p $\forall$q $\forall$s T(p$\vee$q,s) $\Leftrightarrow$ (T(p,s) $\vee$ T(q,s)
$\forall$p $\forall$q $\forall$s T(p$\Rightarrow$q,s) $\Leftrightarrow$ (T(p,s) $\Rightarrow$ T(q,s))

Neke činjenice su tačne u svim stanjima iako sadrže funkcije ili relacije zavisnih stanja. Takve činjenice nazivamo *ograničenja stanja* (state constraints). Evo samo jednog primera:

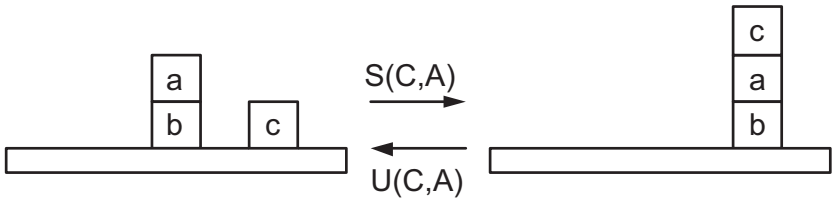
$\forall$x $\forall$s T(Table(x),s) $\Leftrightarrow$ $\neg\exists$y T(On(x,y),s)

tj. "Objekat je na tabli akko nije na nekom drugom objektu".

6.2 Akcije

Svet egzistira u jednom stanju dok akcija menja to stanje u novo stanje. U našem prostoru pretrage akcije konceptualizujemo kao i stanja kao objekte. Na primer, akcija `M(a,b,c)` pomera blok `a` od bloka `b` do bloka `c` i sl.

Operator je funkcija između objekata i akcija koja preslikava grupu objekata u zajednički način manipulisanja tim objektima. Npr., operator preslikava 3 objekta (blokove, kutije, A, B i C) koja su uključena u akciju *move* (M). Slično, akcija `unstack(U)` skida blok sa vrha i smešta ga na tablu, a akcija *stack* (S) stavlja jedan blok na drugi.



`S(C,A)` - stavlja blok `C` na blok `A`, `U(C,A)` - skida blok `C` sa bloka `A` i smešta ga na tablu. Akcije u dometu operatora se često nazivaju instancama tog operatora.

Uopšte, da bismo opisali operatore i akcije prvo im dajemo imena kao što smo već i učinili. Ovakvom notacijom imenujemo svaku akciju, a te terme (npr. `M(C,A,B)`) nazivamo *oznake akcija* (action designator). Činjenicu da je neki operator akcija možemo izraziti koristeći unarnu relaciju `Action(M(C,A,B))` čije su vrednosti T (true) ili F (false) u zavisnosti od toga da li je term koji je argument relacije akcija ili ne.

Efekte akcija možemo koncipirati u obliku funkcije:

$$\texttt{do:}\quad \texttt{A} \times \texttt{S} \rightarrow \texttt{S}$$

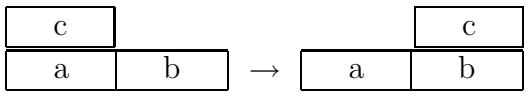
koja preslikava par (akcija, stanje) u novo stanje. Na primer: `do(M(C,A,B),S15)` - gde je rezultat stanje nakon delovanja akcije `M` u stanju `S15`.
Operator U možemo opisati na sl. način

$$\texttt{T(On(x,y),s)} \wedge \texttt{T(Clear(x),s)} \Rightarrow$$
$$\texttt{T(Table(x),Do(U(x,y),s)} \wedge$$
$$\texttt{T(Clear(y),Do(U(x,y),s)}$$

6.3 Problem okvira

Opisi operatora nisu kompletni. Oni opisuju činjenice koje postaju istinite (tačne) kao rezultat izvršavanja instanci svakog operatora i indirektno opisuju činjenice koje postaju netačne. Ponekad, oni ne označavaju ništa vezano

za činjenice koje su pre toga bile istinite i ostaju istinite posle toga, ili označavaju nešto vezano za činjenice koje su bile netačne pre i koje ostaju netačne posle.
U našem primeru, razmotrimo sl.: blok `b` je u nekom stanju na tabli i on je na tabli posle premeštanja bloka `c` sa bloka `a` na blok `b`.



Problem koji se karakteriše aspektima stanja koja se ne menjaju akcijama naziva se *problem okvira*. Naziv potiče iz analogije sa svetom animacija. Animatori često prvo nacrtaju okvir, pozadinu. Zatim to ostave i u narednim sličicama, a u prvom planu postavljaju akcije. Problem okvira razlikuje pozadinu, nemenjanu akcijama, od prvog plana u kojem akcije dovode do izmena. Jedan od načina da se to uradi je da se pišu *aksiome okvira* (frame axioms) koje ukazuju na osobine koje ostaju nepromenjene delovanjem akcija. Npr., razmotrimo sl. aksiomu okvira za U operator:

`T(Clear(u),s)` $\Rightarrow$ `T(Clear(u), Do(U(x,y), s))`

- aksioma izražava da je blok *clear* posle akcije `U`, ako je *clear* pre te akcije, i sl. Izostavljamo navođenje ostalih aksioma okvira za `U` operator, kao i za opertore `S`, `M` i `Noop`. Napomenimo samo da je broj aksioma okvira proporcionalan proizvodu broja relacija i broja operacija. U svetu realne složenosti postoji veliki broj relacija i operacija pa je i broj aksioma okvira velik.

6.4 Redosled akcija

Složene akcije dobijamo kao kompoziciju jednostavnih akcija uz pretpostavku da se one odvijaju u određenom redosledu, bez preklapanja. Dakle, ključ uspeha pri analizi složenih akcija je redosled akcija.
Blok akcija je konačan niz akcija. Pošto ne postoji granica broja akcija u bloku, možemo formirati neograničeno mnogo takvih objekata iz svakog nepraznog skupa takvih akcija. Rezultat svake od akcija u nizu je stanje na koje deluje sledeća akcija. Blok akcija označavamo na sledeći način (kao listu), npr.:
`[U(C,A),S(B,C),S(A,B)]`.

Kada govorimo o rezultatima izvršavanja bloka akcije proširujemo `Do` funkciju. Rezultujuće stanje posle izvršavanja praznog bloka u stanju `s` je `s`. Rezultat izvršenja nepraznog bloka sa početnom akcijom `a` i ostatka (repa) bloka `l` u stanju `s` je stanje dobijeno izvršavanjem bloka `l` u stanju koje je rezultat izvršavanja `a` u `s`, tj.

```
Do([ ],s) = s
Do(a.l,s) = Do(l,Do(a,s))
```

Primetimo da je predstavljanju ove kompozicije preslikavanja redosled akcija obrnut od redosleda elemenata u listi, tj. `Do([a,b],s) = Do(b,Do(a,s))`. Isto tako, opisujemo i efekte bloka akcija u značenjima osobina tih stanja. Sledeće rečenice koriste relaciju `T` u izražavanju prethodnih definicija u ovom alternativnom obliku.

```
T(p,s) ⇔ T(p,Do([ ],s))
T(p,Do(l,Do(a,s))) ⇔ T(p,Do(a.l,s))
```

Ovde je važno da se naglasi da možemo redukovati pitanja vezana za efekte blokova akcija u pitanja vezana za efekte akcija sadržanih u tim blokovima. Razmotrimo problem opisivanja beskonačnog niza akcija. Naravno, ne može se koristiti beskonačna lista,pa se kao rešenje koriste različiti koncepti redosleda akcija. *Sekvencijalna procedura* je funkcija pozitivnih celih brojeva koja se slika u beskonačan niz akcija $f : N \rightarrow A$.

Opisujući sekvencijalnu proceduru, mi implicitno karakterišemo niz akcija koji se na nju odnosi. Primer - rečenice koje definišu sekvencijalnu proceduru koja diktira tri akcije nakon kojih sledi niz `Noop` akcija:

```
F(1)=U(C,A)
F(2)=S(B,C)
F(3)=S(A,B)
n ≥ 3 ⇒ F(n)=Noop
```

6.5 Uslovljenost

Često želimo razgovarati o akcijama koje su izvršene samo pod određenim uslovima. Diskutovaćemo o tri pristupa formalizovanja uslovljavanja: uslovne akcije, produkcioni sistemi i Markovljeve procedure.

Uslovna akcija se sastoji od uslova i dve akcije. Ako je uslov zadovoljen primenjuje se prva akcija, a ako nije druga akcija.

Uslovne akcije označavamo *uslovnim izrazima* (upitima): `If(`ϕ`,`α`,`β`)`, gde je `If` ternarna funkcija konstante, ϕ deskriptor (opisivač) stanja, a α i β akcije.

Produkciona pravila su parovi koji se sastoje od uslova i akcije. Produkcioni sistem je konačan skup produkcionih pravila.
Izvršavanje produkcionog sistema u početnom stanju može uključiti mnogo koraka. U svakom koraku, izvršena akcija je deo akcije prvog produkcionog pravila u nizu uslova koji su zadovoljeni. Izvršavanje se završava akko ne postoji produkciono pravilo čiji je uslov zadovoljen.

Produkciona pravila su oblika $\phi \rightarrow \alpha$, pri čemu je ϕ deskriptor stanja, a α oznaka akcija. Da bismo formalizovali efekte izvršavanja produkcionog sistema definišemo relaciju ozanačenu sa Dictates, koja za dati produkcioni sistem, stanje i akciju važi akko produkcioni sistem diktira specifičnu akciju u datom stanju:

`T(p,s)` $\Rightarrow$ `Dictates((p`$\rightarrow$`a).l,s,a)`
$\neg$`T(p,s)` $\wedge$ `Dictates(l,s,b)` $\Rightarrow$ `Dictates((p`$\rightarrow$`a).l,s,b)`

Ako sistem ne diktira neku akciju za dato stanje, rezultat je njegovo postojeće stanje. Inače, rezultat izvršavanja produkcionog sistema u datom stanju je stanje dobijeno izvršavanjem diktirane akcije:

($\neg\exists$`a Dictates(p,s,a))` $\Rightarrow$ `Do(p,s)=s`
`Dictates(p,s,a)` $\Rightarrow$ `Do(p,s)=Do(p,Do(a,s))`

Navedena definicija relacije `Dictates` zahteva da se uvek uzme prvo pravilo

iz liste koje ispunjava uslove tako da je svaka dvosmislenost eliminisana. Neki produkcioni sistemi koriste različite strategije za razrešavanje konfliktne rezolucije. Jedna lepa posledica ove politike razrešavanja konfliktne rezolucije (postupanja u konfliktnim situacijama) jeste da možemo koristiti redosled pravila.

Markovljeva procedura f je funkcija $f : S \rightarrow A$. Produkcioni sistem je samo specifičan deo Markovljeve procedure. *Markovljev program* opisuje Markovljevu proceduru formalnim programskim jezikom. Koristimo predikatski jezik kao opisni jezik pa se tako Markovljev program sasatoji od konstante funkcije (function constant) π koja opisuje proceduru i skupa $\triangle$ rečenica predikatskog jezika koje opisuju tu proceduru.

Markovljev program je *kompletan* ako i samo ako opisuje samo jednu akciju u svakom stanju. Naravno, Markovljev program nije uvek kompletan. Čak i ako procedura opisana programom diktira jednu akciju za svako stanje, taj opis može biti nekompletan. U nekim slučajevima možemo imati opis koji ograničava skup akcija, u drugim slučajevima program može specificirati akcije za neka stanja dok za druga ne i takvi *parcijalni programi* su karakteristični za VI (veštačku inteligenciju). Primer pravila koje je uključeno u Markovljevu proceduru za parcijalni program `P1`:

`T(Clear(C),s)` $\wedge$ `T(On(C,z),s)` $\Rightarrow$ `P1(s)=U(C,z)` koje odgovara zapisu pravila produkcionog sistema: `Clear(C)` $\wedge$ `On(C,z)` $\rightarrow$ `U(C,z)`, itd.

Markovljev program je *lokalni* ako i samo ako svaka rečenica sadrži najviše jedan term oznake stanja, ili višestruka pojavljivanja tog terma, i ako postoji oznaka stanja je promenljiva stanja kvantifikovana univerzalnim kvantifikatorom. Prisustvo deskriptora stanja neće narušiti ovu definiciju. Značaj ove osobine je u tome da je onda akcija određena isključivo osobinama stanja.

Kada je Markovljev program lokalni, jednostavno se konvertuje u produkcioni sistem: formira se lista rečenica, iz svake rečenice izostavljamo ime procedure (kao i T relacije) i sve promenljive stanja. Konačno menjamo $\Rightarrow$ u $\rightarrow$. Ipak ne može svaki Markovljev program biti napisan na ovakav način. Problem nastaje kada program sadrži rečenice koje ne završavaju vrednostima koje su definisane za tu proceduru, kada zaključak nije pozitivan i kada se pojavljuju višestruki različiti termi stanja.

Uz problemu okvira i ranije pominjan problem kvalifikacije (Lifschitz, 1986) javlja se i alternativni metod *nepoznavanja hronologije* (chronological ignorance, Shoham, 1986).

Klasični modeli stanje-akcija, opisani u ovom poglavlju mogu biti generalizovani u tri pravca. Prvo - potrebno je direktno shvatanje hronologije (obično formulacije stanja ne pominju hronologiju eksplicitno, mada je očigledna implicitna veza), drugo - pretpostavkom konceptualizacije koja uključuje hronologiju možmo posmatrati kontinuitet akcija, i treće - od značaja su i simultane akcije koje se javljaju pored akcija u rečenicama.

Pored ovih smernica, u veštačkoj inteligencijiji u oblasti problema shvatanja hronologije neka rešenja uključuju argument vremena u same relacije, dok neka druga pribegavaju modalnoj temporalnoj logici.

7 Planiranje

Sposobnost planiranja unapred je ključni aspekt inteligentnog ponašanja. Znanjem posledica preduzetih akcija i korišćenjem tog znanja mi stižemo do cilja izbegavajući opasnosti i dobro ekonomišući resursima. U planiranju započinjemo sa skupom željenih osobina i pokušavamo smisliti plan kako do cilja doći. U ovom poglavlju prvo ćemo razmotriti ulaz procesa planiranja, zatim njegov izlaz, a potom metode za planiranja bloka akcija i uslova planiranja.

7.1 Početno stanje

Početno stanje u problemu planiranja je stanje u kojem izvršilac očekuje početak akcije. Oznaka početnog stanja je naše ime za ovo stanje i mi ga koristimo za pisanje rečenica vezanih za početno stanje. Npr.:

```
T(Clear(C),S1)
```

-u početnom stanju `S1` na bloku `C` nema ništa , itd. dok se u potpunosti ne opiše početno stanje.

7.2 Ciljevi

Uopšteno govoreći, cilj može biti svako dostignuto stanje. U nekim planiranjima postoji samo jedno ciljno stanje. Ove mogućnosti predstavljaju koncepte ciljeva kao unarnu relaciju stanja. Kažemo da je stanje *ciljno stanje* ako i samo ako zadovoljava tu relaciju. U opisivanju ciljeva koristimo konstantu relacije `Goal` koja označava cilj relacije. Npr.:

`T(On(A,B),t)` $\wedge$ `T(On(B,C),t)` $\Leftrightarrow$ `Goal(t)`

7.3 Akcije

Skup oznaka akcija u planiranju problema uključuje term za svaku primitivnu ili složenu akciju koja konvertuje početno stanje u ciljno. Iako postoji konačno mnogo oznaka primitivnih akcija, može postojati beskonačno mnogo složenih akcija u tom skupu. Kada je ovo slučaj, ne možemo uzeti ovakav skup kao

argument za naše planiranje , i zato, umesto toga specifiramo izračunljivu metanivoovsku relaciju koja je istinita za svaki term u skupu i samo ti termi su u skupu. Razlog za uključivanje ove informacije kao ulaza planiranja je ograničavanje planova koji ih proizvode kako bi mogli biti upotrebljeni od strane izvršioca kojeg imamo na umu. Npr. neprikladno je dopustiti deskriptoru stanja `Color(A,Blue)` da bude upotrebljen u uslovnim akcijama ako znamo da izvršilac ne može odrediti boju blokova.

Uzimajući u obzir elemente skupa oznaka akcija u planiranju problema, imamo prikladne deskriptore akcija i aksiome kojima su zadati. One uključuju opise operatora i aksiome okvira za primitivne akcije, obične definicije za složene akcije kao što su blokovi akcija i uslovne akcije, i ograničenja stanja koja moraju biti istinita u svakom stanju, tj. veze koje se ne menjaju ni za jednu akciju.
Primer aksioma:

$$T(On(x,y),s) \land T(Clear(x),s) \Rightarrow T(Table(x),Do(U(x,y),s))) \land T(Clear(y),Do(U(x,y),s)))$$

$$T(Table(x),s) \land T(Clear(x),s) \land T(Clear(y),s) \land x \neq y \Rightarrow T(On(x,y),Do(S(x,y),s))$$

Aksiome za blokove akcija:

$$T(p,s) \Rightarrow T(p,Do([\,],s))$$
$$T(p,Do(l,Do(a,s))) \Rightarrow T(p,Do(a.l,s))$$

Aksiome ograničenja stanja:

$$T(Table(x),s) \Leftrightarrow \neg\exists y\ T(On(x,y),s)$$
$$T(Clear(y),s) \Leftrightarrow \neg\exists x\ T(On(x,y),s)$$
$$T(On(x,y),s) \land y \neq z \Rightarrow \neg T(On(x,z),s)$$

Aksiome uslovnih akcija:

$$T(p,s) \land T(q,Do(a,s)) \Rightarrow T(q,Do(If(p,a,b),s))$$
$$\neg T(p,s) \land T(q,Do(b,s)) \Rightarrow T(q,Do(If(p,a,b),s))$$

Neke karakteristike stanja nisu obuhvaćene ovim operatorima kao što su sledećim aksiomama okvira koje uključuju i `Noop` akciju.

```
T(Table(u),s) ⇒ T(Table(u),Do(U(x,y),s))
T(Clear(u),s) ⇒ T(Clear(u),Do(U(x,y),s))
T(On(u,v),s) ∧ u ≠x ⇒ T(On(u,v),Do(U(x,y),s))
T(Table(u),s) ∧ u ≠x ⇒ T(Table(u),Do(S(x,y),s))
T(Clear(u),s) ∧ u ≠y ⇒ T(Table(u),Do(S(x,y),s))
T(On(u,v),s) ⇒ T(On(u,v),Do(S(x,y),s))
T(p,s) ⇒ T(p,Do(Noop,s))
```

... itd.

7.4 Planovi

Problem planiranja se sastoji od oznake početnog stanja σ, oznake ciljne relacije ρ, skupa oznaka akcija Γ , baze podataka Ω koja uključuje rečenice koje opisuju početno stanje, ciljnu relaciju i upotrebljive akcije.
Oznaka akcije γ je plan za planiranje problema ove vrste ako i samo ako zadovoljava sledeće uslove:

1. Oznaka akcije mora biti elemenat skupa oznaka akcija, tj. $\gamma \in \Gamma$.

2. Za Ω mora biti dokazano γ postiže zadovoljavajuće stanje ρ kada je izvršena u stanju σ:

   $\Omega \models (\rho(Do(\gamma, \sigma)))$

Npr., razmotrimo situaciju u kojoj je S1 oznaka početnog stanja i Goal ime ciljne relacije. Pretpostavimo da Γ uključuje imena za sve obične primitivne akcije Sveta blokova i stoga sve konačne rečenice. Deskriptor početnog stanja tvrdi da je blok `C` na bloku `A` i blokovi `A` i `B` da su na tabli. Deskriptor cilja tvrdi da stanje zadovoljava ciljeve ako i samo ako je blok `A` na bloku `B` i blok `B` je na na bloku `C`.

Term `[U(C,A),S(B,C),S(A,B)]` je plan za rešavanje ovog problema. On je, jasno, elemenat Γ, a koristeći informacije iz Ω dokazujemo da ovaj plan radi, tj. `Goal(Do([U(C,A),S(B,C),S(A,B)],S1))`.

7.5 Grinov metod

Grinov metod je procedura planiranja zasnovana na rezoluciji. Ovaj metod uzima kao argumente : term koji označava početno stanje, konstantu unarne relacije koja označava ciljnu relaciju, predikat zadovoljen planovima izvršavanja i samo njima , i bazu podataka vezanu za početno stanje, relaciju cilja i raspoložive operacije.

Osnova ovog metoda je popunjavanje praznine rezolucije dobijene kao sporedni efekat korektnog plana prilikom dokaza njegovog postojanja. Dati su term početnog stanja σ i konstante ciljne relacije ρ . Pokušavamo da dobijemo tvrđenje postojanja plana(*plan-existence*):

$\exists\nu\ \rho$(Do(ν,σ))

Koristimo predikate koji se izvršavaju da bismo označili svaki odgovor dobijen ovim procesom. Ako pronađemo odgovor koji zadovoljava ovaj predikat dobijamo taj term kao odgovor na problem planiranja. Inače, nastavljamo nabrajanjem rešenja.

Kako je Grinov metod zasnovan na rezoluciji moguće je dokazati neke jače karakteristike vezane za njegove sposobnosti planiranja. Metod je siguran u smislu da produkuje samo korektne planove. On je i kompletan i to garantuje produkovanje korektnih planova kad god oni postoje. Ne postoje ograničenja vezana za tip uključenih planova.
Na žalost, Grinov metod, kao i sve procedure planiranja, mogu biti ekstremno neefikasne.

7.6 Blokovi akcija

Jednostavna primena Grinovog metoda je u poznatom primeru određivanja niza akcija u svetu blokova. Primer:

Uzmimo da je `S1` početno stanje u kojem je blok `A` na bloku `B` i blok `B` na bloku `C`.

```
T(Clear(A),S1)
T(On(A,B),S1)
T(On(B,C),S1)
T(Table(C),S1)
```

Definišimo cilj sa

```
T(Table(a),t) ⇔ Goal(t)
```

Planiranje procesa započinjemo sa (plan-existence) tvrđenjem postojanja plana, konvertujući ga u klauzalnu formu, i dodajući literal odgovora dobijamo:

```
1.{¬Goal(Do(a,S1)), Ans(a)}
2.{¬T(Table(A), Do(a,S1)), Ans(a)}
3.{¬T(On(A,y),S1), ≠T(Clear(A),S1), Ans(U(A,y))}
4.{¬T(Clear(A),S1), Ans (U(A,B) )}
5.{Ans(U(A,B))}
```

Ans(a) je ,,answer literal", tj. literal sa odgovorom.

7.7 Uslovni planovi

Kada informacija nedostaje u toku planiranja, ponekad je nemoguće planirati blok akcija koji garantuje postizanje cilja. Na sreću možmo rešiti probleme ove vrste pomoću uslovnih akcija.
Npr. koristeći Grinov metod za generisanje uslovnog plana, razmatramo problem planiranja u kojem znamo da na bloku a nema ništa u početnom stanju i ne znamo ništa više. Tako imamo,

```
T(Clear(A),S1)
```

Cilj nam je da blok `A` bude natabli, tj.

```
T(Table(A),t) ⇔ Goal(t)
```

Problem ne možemo ograničiti u smislu da blok `A` može biti na tabli ili na tabli mogu biti blok `B` ili blok `C`. Stoga ne postoji jedna akcija koja garantuje rešenje problema, ali možemo napisati uslovni program za rešavanje problema . Npr.,

```
1.  {¬Goal(Do(a,S1)), Ans(a)}
2.  {¬T( Table(A), Do(a,S1)), Ans(a)}
```

3. {¬T(p,S1), ¬T(Table(A),Do(a,S1)), Ans(If(p,a,b))}
4. {¬T(p,S1), ¬T(On(A,y),S1), ¬T(Clear(A),S1), Ans(If(p,U(A,y),b))}
5. {¬T(p,S1), ¬T(On(A,y),S1), Ans(If(p,U(A,y),b))}
6. {¬T(On(A,y),S1), Ans(If(On(A,y),U(A,y),b))}
7. {T(p,S1), ¬T(Table(A),Do(b,S1)), Ans(If(p,a,b))}
8. {T(p,S1), ¬T(Table(A),S1), Ans(If(p,a,Noop))}
9. {T(p,S1), T(On(A,K),S1), Ans(If(p,a,Noop))}
10. {T(On(A,K),S1), Ans(If(On(A,K),a,Noop))}
11. {Ans(If(On(A,K),a,Noop)), Ans(If(On(A,K),U(A,K),b)}
12. {Ans(If(On(A,K),U(A,K),Noop))}

K je Skolemova konstanta za promenljivu iza egzistencijalnog kvantifikatora poteklu od od ograničenja za stanje. Generalizacijom K se dobija željeni plan.

7.8 Smer planiranja

Jedan od načina za povećanje efikasnosti planiranja je smer planiranja (planning direction). U nekom slučaju je bolje da planiramo unapred počev od početnog stanja, u drugim slučajevima bolje je da idemo unazad, od cilja, dok je u nekim najbolje korišćenje oba metoda.

U planiranju zasnovanom na rezoluciji možemo uticati na smer korišćenjem restrikcione strategije modifikovanog skupa podrške u kojoj slabimo pretpostavku da je komplement tog skupa zadovoljiv. Ako uzmemo rečenice dobijene iz negacije (plan-existence statement) postojanja plana kao skupa podrške, rezultat je planiranje unazad. Ako uzmemo rečenice koje opisuju početno stanje našeg skupa dobijamo planiranje unapred. A ako posmatramo uniju ovih skupova dobijamo treću varijantu - primenu oba metoda.

Svi proizvodi prethodnih primera su instance planiranja unazad. U svakom od njih, počinjemo negacijom cilja, redukujemo cilj na podciljeve, i tako dok ne dobijemo uslove početnog stanja.

Jedan od problema korišćenja (set-of-support) strategije podržanog skupa i njegove implementacije unapred jeste taj da on ne mora biti kompletan. Npr. razmotrimo problem planiranja u kojem ne postoje informacije vezane za početno stanje i u kojem ne postoji akcija koja zadovoljava cilj u svakom stanju. Koristeći u ovoj situaciji planiranje unazad možemo dobiti plan koji nećemo moći dedukovati unapred. U mnogim slučajevima i jedno i drugo planiranje su jednako kompetentni.

Sa druge strane, postoje situacije u kojima je nepraktično primeniti planiranje unazad. Na primer, problem pobeđivanja u igranju šaha. Možemo, idući unazad, počev od pozicije koja je donela pobedu, da odredimo svaki naš sledeći potez. Problem će biti velik broj mogućnosti. Alternativa je uzeti u obzir nekoliko koraka unapred, zamena cilja pobede ciljem čiju vrednost menjamo evaluacionom funkcijom stanja. Izbor načina planiranja pravimo vodeći se efikasnošću. Ako broj mogućnosti koje se koriste u smeru unapred premašuje broj onih koje se koriste u smeru unazad, tada koristimo planiranje unazad. Ako faktor grananja u smeru unazad premašuje isti u direkciji unapred, bolje je koristiti planiranje unapred.

7.9 Odsecanje nedostižnošću

Jedan razlog izračunljive slabosti u planiranju unazad je rad na rečenicama koje opisuju nedostignuta stanja. Npr. nemoguće je stanje u kojem ja A na B, a B je Clear (tj. na B nije postavljen nijedan drugi blok). Stoga, takva tvrđenja odsecamo.
Jedan način da detektujemo takve slučajeve je da postavimo rezolucioni potproces koji testira validnost rečenica (klauzula). Ako ovaj test pokaže da je rečenica validna (njena negacija nije nekonzistentna sa ograničenjima stanja), ta rečenica se izuzima iz daljeg razmatranja. Ovakva strategija brisanja se ponekad zove *odsecanje nedostižnošću*.

Baza podataka za testiranje logičnosti sastoji se od (1) aksioma ograničenja u problemu planiranja, i (2) rečenica dobijenih iz negiranog tvrđenja koje je pitanju. Za mnoge baze znanja može se pokazati da je rezolucioni postupak garantovano određen uz pokazivanje konzistentnosti ili nekonzistentnosti. Postoje i slučajevi za koje je nemoguće znati da li će se desiti problem ako originalni rezolucioni proces čeka kompletiranje ovog rezolucionog potprocesa. Jedan od načina za rešavanje ove situacije je ograničavanje utrošenog vremena u kontrolisanju logičnosti. Drugi pristup je isprepletena provera konzistentnosti procesa sa procesom planiranja.

7.10 Poravnavanje stanja (usaglašavanje)

Tokom procesa planiranja možemo se sresti sa situacijom u kojoj postoji nekoliko uslova koji moraju biti zadovoljeni u jednom stanju. Kada koristimo operator (operator description axiom) koji zadovoljava jedan od ovih uslova,

dolazimo do potproblema u kojem preduslovi operatora moraju biti u jednom stanju, a ostali uslovi se moraju nalaziti u sledećim stanjima.
Npr.,

```
1.  {¬T(On(A,B),Do(a,S1)), ¬T(Table(B), Do(a,S1))}
2.  {¬T(On(A,B),Do(U(B,y),S1)), ¬T(On(B,y), S1),

  ¬T(Clear(B),S1)}
```

Prva rečenica izražava da je cilj da je `A` na `B` i `B` na tabli. Posle korišćenja operatora opisa aksioma za `U`, redukujemo jedan od uslova u stanju označnom sa `Do(a,S1)`, dobijamo na kraju rečenicu koja uključuje dva uslova u stanju `S1` i preostali uslov u `Do(a,S1)`. Obično je taj podcilj nedostižan, ali mi ne možemo koristiti nedostižna odsecanja, sve dok stanja ne budu poravnata (usaglašena). Sada imamo mogućnost izbora: da li da redukujemo uslove u stanju `S1` ili da redukujemo preostale uslove u `Do(U(B,y),S1)`.

Poravnanje stanja (state alignment) je strategija restrikcije (redukcije) koja isključuje bilo kakvu rezoluciju nad literalom koji sadrži term stanja σ kada postoji drugi literal u istoj rečenici koji sadrži (state term) term stanja `Do(`α, σ`)`. Naša je namera da izbegnemo redukcije uslova u jednom stanju dok postoji još uslova u sledećim stanjima koje treba redukovati.

Kada koristimo odsecanje nedostižnošću, poravnanje stanja može biti vođeno suštinskim poboljšanjima u planiranju efikasnosti. Poravnanjem uslova u rečenici u jednom stanju ponekad srećemo protivrečne rečenice na koje inače ne bismo nailazili. Kao rezultat, možemo eliminisati takve rečenice i poštedeti se daljeg posla oko njih. Npr. možemo odseći drugu rečenicu u prethodnom primeru, zato što je nemoguće dostići stanje u kojem na B nema ništa i u kojem je A na B. Ovaj problem se ne bi desio da smo primenili odsecanje nedostižnošću u prvoj rečenici.

Jasno je da upotreba poravnanja stanja može narušiti kompletnost Grinovog metoda kada se upotrebljava na proizvoljnom skupu aksioma. Ponekad, ako su sve te aksiome napisane u formi opisa operatora ili aksioma okvira to se neće desiti.

7.11 Ukidanje aksioma okvira

Pokazuje se da je često korisno ukinuti rezoluciju u kojoj se primenjuje aksioma okvira kao ograničenje za promenljivu akcije tj. gde se zaključuje iz literala koji je uslov za term stanja oblika `Do(`ν`,`σ`)` gde je ν promenljiva. Ovo je pored pomenutog odsecanja takođe način da se poveća efikasnost planiranja.

7.12 Ciljna regresija

Interesantno je pomenuti da svi operatori koji se opisuju u našim primerima imaju prilično jednostavnu formu. Efekti svakog operatora su karakterisani jednom rečenicom (ne računajući aksiome okvira i ograničenja stanja). Rečenica je u svakom slučaju implikacija u kojoj je premisa uslov o stanju koje je neophodno da bi operator imao efekte koji se javljaju u zaključku. Ishod svega ovog je to da kada imamo opise operatora ove vrste, možemo koristiti veoma jednostavnu ali moćnu strategiju planiranja poznatu kao *ciljna regresija* (goal regression).

Prvo prevedemo naš operator u ekvivalentan, ali jednostavniji oblik. Svaki primer operatora se karakteriše skupom preduslova, skupom pozitivnih i skupom negativnih efekata. Preduslovi `Pre(a)` akcije `a` su uslovi koji moraju biti tačni da bi akcija `a` imala željene efekte. Pozitivni efekti `Add(a)` su efekti koji postaju tačni posle izvršavanja akcije. Negativni efekti `Del(a)` su uslovi koji postaju netačni.

Npr., razmotrimo kako možemo drugačije zapisati opis operatora `U`. Posmatrajući opis operatora primećujemo da uslovi u primeru oblika `U(x,y)` uključuju deskriptore stanja `Clear(x)` i `On(x,y)`. Pozitivni efekti uključuju `Table(x)` i `Clear(y)`. Postoji samo jedan negativan efekat, `On(x,y)`.

```
Pre(U(x,y)) = {On(x,y),Clear(x)}
Add(U(x,y)) = {Table(x),Clear(y)}
Del(U(x,y)) = {On(x,y)}
```

U ovoj formulaciji definišemo *ciljni skup* (goal set), koji predstavlja skup skupova stanja, tako da je svako stanje koje je u preseku ovih skupova zadovoljeno. Npr. sledeći ciljni skup opisuje skup stanja u kojima su blokovi

`A` i `B` na tabli: {`Table(A), Table(B)`}

Osnovni korak u ciljnoj regresiji je redukcija jednog cilja na podcilj na osnovu opisa akcija. Redukcija mora imati osobine da izvršavanje opisane akcije u stanju u kojem je podcilj zadovoljen dovodi do stanja u kojem je cilj zadovoljen. Shodno prethodnoj definiciji vidimo da se podcilj `Reg(q,a)`, koji proizilazi iz *regresije* za `q` *kroz* akciju a, sastoji od preduslova za a zajedno sa članovima u `q` među kojima nisu pozitivni efekti za `a`. Da bi se akcija primenila ne sme biti preklapanja negativnih efekata akcije i uslova u cilju.

`(q` $\cap$ `Del(a)) = {}` $\Rightarrow$ `Reg(q,a) = Pre(a)` $\cup$ `(q-Add(a))`

Npr. regresirajući ovaj ciljni skup kroz akciju `U(A,B)` dolazimo do sledećeg ciljnog skupa. Nijedan od originalnih ciljeva ne predstavlja negativan efekat ove akcije, tako da ova definicija važi.

Podciljni skup se sastoji iz preduslova za `U(A,B)` zajedno sa onim ciljem koji nije sadržan u pozitivnim efektima akcija.

{`Clear(A), On(A,B), Table(B)`}

Dalje, definišimo ternarnu relaciju `Plan`, koja je tačna za dati ciljni skup, stanje, i niz akcija akko je stanje koje je rezultat izvršavanja niza akcija u datom stanju u ciljnom skupu.

`Plan(q,s,l)` $\Leftrightarrow$ `T(q,Do(l,s)`

Konačno možemo iskoristiti definiciju regresije da damo uslove unutar kojih je niz akcija plan. Prazan niz je plan za ciljni skup `q` u stanju `s` ako `s` zadovaljava elemente u `q`. Niz `a.l` je plan za ciljni skup `q` ako:

(1) `a` je akcija pozitivnih efekata sa nekim elementom iz `q` i
(2) `l` je plan koji postiže ciljni skup dobijen pomoću regresiranjem `q` kroz `a`.

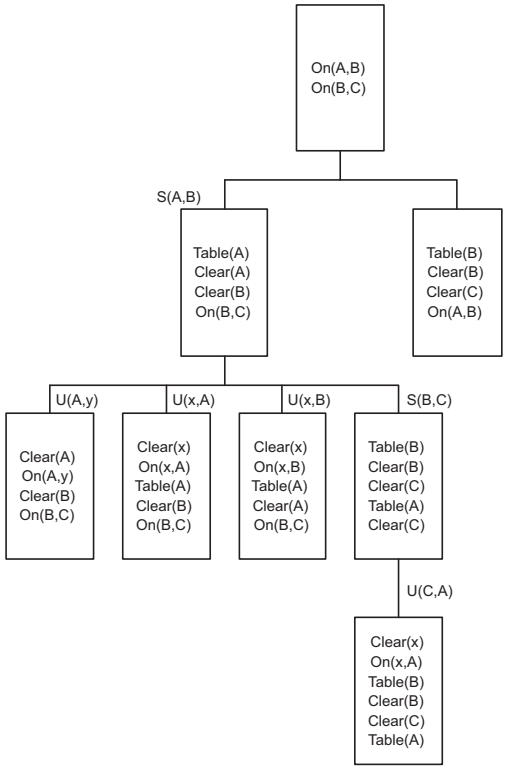
`T(q,s)` $\Rightarrow$ `Plan(q,s,{})`

`(q` $\cap$ `Add(a))` $\neq$ {} $\wedge$ `Plan(Reg(q,a),s,l)` $\Rightarrow$ `Plan(q,s,a.l)`

Ako je inicijalno stanje (deskriptor) σ, cilj Ψ, ciljna regresija je onda

pronalaženje γ td. važi `Plan(`Ψ`,`σ`,`γ`)`.

Kao primer ciljne regresije posmatrajmo sledeći primer: U početnom stanju blok `C` je na bloku `A` i blokovi `A` i `B` su na tabli. Cilj je doći u stanje u kojem je `A` na `B` i `B` na `C`. Postoje dve akcije sa pozitivnim efektima koje uključuju elemente našeg cilja. Akcija `S(A,B)` izvršava akciju `On(A,B)` i akcija `S(B,C)` akciju `On(B,C)`. Skupovi podciljeva koji se dobiju iz regresije cilja kroz ove dve akcije su pokazani ispod cilja i relevantne akcije su indukovane pomoću oznaka na granama.



Podcilj desno može biti napušten. On zahteva da `B` bude ,,čist", tj. da na `B` nema ništa i da `A` bude na `B`. Vidimo da je to nemoguće.

Podcilj levo ima četiri moguća podcilja. Krajnji levi je nemoguć. Promenljiva y ne može biti `A` pošto blok ne može biti na vrhu. Isto tako y ne može biti `B` pošto `B` mora biti ,,čist" i ne može biti `C` jer je `B` na `C`. Slično, drugi i treći podcilj su nekonzistentni (nesaglasni) i mogu biti odsečeni.
Poslednji podcilj se odnosi na akciju `S(B,C)`, i ovaj cilj je saglasan. U stvari, ovaj cilj ima podcilj koji smo videli i on je tačan u početnom stanju ako x

zamenimo sa `C`. U toj tački možemo naći korektan plan čitajući akcije sa stabla u obrnutom poretku (unazad). Prvo, skinemo `C` sa `A`, zatim stavimo `B` na `C`, i na kraju, stavimo `A` na `B`.

Iako se ciljna regresija veoma razlikuje od predhodnih strategija planiranja, čak i male analize pokazuju da su slične. U stvari, ciljna regresija je ekvivalentna Grinovom metodu kada se on koristi u konjukciji sa poravnanjem stanja i aksiomama okvira.

7.13 Razlike stanja

Iako je u nekoj restrikcionoj strategiji moguće eliminisati sve pretrage, ovakav ishod je prilično nemoguć. Ostaje nam problem odlučivanja kojim redom izvršavati odluke koje dozvoljava ta strategija. Jedan uobičajeni način pravljenja ovog izbora je korišćenje mere nesličnosti između stanja.
Funkcija razlikovanja stanja (state-difference function) je binarna funkcija stanja koja vraća broj koji odgovara stepenu sličnosti između stanja. Veća vrednost funkcije označava da se stanja više razlikuju. Ako je vrednost funkcije nula, stanja su identična. Razmotrimo funkciju razlikovanja stanja u primeru Sveta blokova. Ukupna vrednost je suma razlike lokacija i razlika koje se odnose na to da li je na bloku smešten neki drugi blok (clearness difference-razlika popunjenosti). Razlika lokacije ukazuje na to da li je blok smešten na različitom mestu u dva različita stanja. Ako su dva različita bloka u dva različita stanja na istom bloku, vrednost razlike lokacije je 1. Razlika popunjenosti za blok je 1 ako se dva stanja ralikuju po tome da li je jedan blok u jednom stanju popunjen, a u drugom ne.

Redosled stanja (state ordering) je rezoluciona strategija u kojoj se redosled rezolucije (odluke) na klauzulama poravnanja stanja određuje pomoću funkcije razlikovanja stanja.

Da bi funkcija razlikovanja stanja bila u potpunosti iskorištena za poboljšanje efikasnosti planiranja mora postojati korelacija između funkcije razlike i planiranja da bi se jedno stanje konvertovalo u drugo. U ekstremnim slučajevima, kada je funkcija stanja monotona i kada je u pitanju teškoća planiranja možemo koristiti hillclimbing. Kada funkcija razlikovanja stanja nije monotona moramo se osloniti na strategiju sa (backup) podrškom kao, na primer, sa procedurom „najbolji prvi".

Iako smo govorili samo o teškoći planiranja uobičajeno je da saberemo razlike stanja i da izračunamo troškove plana da bi došli do komplikovanijih pravila redosleda. Koristeći ovo kombinovano merenje u proceduri ,,najbolji prvi" dolazimo do procedure A^*. Možemo eliminisati neefikasnost proširujući pojam razlike stanja na skupove stanja. Ovo dobijamo na osnovu veličine preseka između dva skupa stanja.

8 Arhitektura inteligentnih agenata

Agenti su formalizmi slični konačnim automatima i Tjuringovim mašinama, ali imaju i dodatne osobine. Ovde se kratko razmatraju vrste agenata koji deluju usamljeni u svetu, iako je u oblasti veštačke inteligencije čest slučaj da ih je više (različitih vrsta) i da interaguju međusobno.

8.1 Tropistični agenti

Tropizam je tendencija biljaka i životinja da (odgovaraju) reaguju na spoljašnje uticaje. Tako je i aktivnost ove klase agenata vezana isključivo za spoljašnji uticaj. U ovom poglavlju govorimo o agentima sa unutrašnjim stanjima (memorijom), ali za sada ćemo ignorisati tu mogućnost.

Različiti agenti će opaziti (reagovati) na različita spoljašnja stanja, pa recimo, u našem primeru, neki registruju boju blokova, neki njihovu težinu i sl. Karakterišući senzorne sposobnosti agenata delimo skup S spoljašnjih stanja u skup T nepovezanih podskupova. Uvodeći funkciju *see* koja preslikava stanje iz S u particiju kojoj pripada, povezujemo stanja iz S sa odgovarajućim particijama. Funkciju ove vrste nazivamo *senzorna funkcija* (sensory function).

$$see : S \rightarrow T$$

Slično senzornim sposobnostima, različiti agenti mogu imati i sposobnosti pravljenja različitih efekata (effectory capabilities). Neki agenti mogu crtati blokove, ali ih ne mogu pokretati, drugi ih mogu pokretati ali ne mogu menjati njihovu boju i sl. Karakterišući efekte ovih akcija definišemo funkciju *do* koja preslikava svaku akciju i stanje u stanje koje nastaje posle primene date akcije u prvobitnom stanju. Funkciju ove vrste nazivamo *sposobnost delovanja* (effectory function).

$$do : A \times S \rightarrow S$$

Posmatrajući aktivnost agenata definišemo funkciju *action* koja preslikava particiju kojoj stanje pripada u akciju.

$$action : T \rightarrow A$$

Konačno, definišemo tropistične agente kao šestorku (S,T,A,see,do,action)

- S - skup stanja spoljnog sveta
- T - skup particija od S, služe da bi se opisao tok rada, ,,algoritam"
- A - skup akcija
- $see : S \rightarrow T$
- $do : A \times S \rightarrow S$
- $action : T \rightarrow A$

Možemo ovako sumirati delovanje tropističnih agenata: u svakom ciklusu agentovo okruženje je u nekom stanju s ; agent posmatra particiju t koja se odnosi na senzornu funkciju *see(s)*; koristi *action* da nađe akciju a koja je dodeljena particiji t; na kraju izvršava akciju ,čime produkuje stanje $do(a, s)$; ciklus se ponavlja.

Jednostavan primer jeste *Svet lavirinta* gde imamo 3×3 povezana kvadratića, kolica i zlato. Cilj je da se u kolica natovari zlato. Postoji 90 mogućih stanja : kolica mogu biti u 9 polja i za svaku takvu mogućnost zlato se može naći u nekom od 9 polja ili u kolicima (razlikuje se situacija kada su zlato i kolica u istom polju i kada je zlato u kolicima). Sa naše tačke gledišta, moguće je iz bilo kog stanja dostići svako od stanja. Za razliku od toga, inteligentni agent sa senzorima na kolicima može reći svoju lokaciju, ali kada je u pitanju zlato može samo reći da li je zlato u kamionu, u nekom polju ili negde drugde. Ova senzorna ograničenja dele skup od 90 stanja u 27 podskupova. Stanje u svakom podskupu se slaže sa pozicijom kolica. Ono se, takođe, slaže sa pozicijom zlata u odnosu na kolica, ali se ne slaže sa tačnom pozicijom zlata kada je ono locirano u različitom polju u odnosu na kolica.

Pored senzorne ograničenosti, ovi agenti imaju i ograničeno delovanje (ograničene efekte). U našem primeru oni mogu pomeriti kolica od polja do polja i mogu manipulisati zlatom kad god je ono u kolicima ili na nekom polju.

Efekte koje proizvode agenti možemo grupisati u sedam akcija

- agent može pomeriti kolica gore, dole, levo i desno (4)
- može da stavi zlato u kolica
- može da premesti zlato

- može da ne proizvodi nikakav efekat (ne radi ništa)

Razmotrimo problem dizajniranja akcija za agente sa ovim ograničenjima. Pretpostavimo da su u početnom stanju kolica u gornjem levom polju lavirinta. Cilj nam je da dođemo do zlata koje je u donjem desnom polju.

1. ako su kolica na izlazu i ako je zlato u istom polju, agent ne radi ništa
2. ako su kolica na izlazu i ako je zlato u kolicima, agent pomera zlato
3. ako su kolica na bilo kom polju i ako je zlato na istom polju, agent stavlja zlato u kolica
4. ako kolica nisu na izlazu i ako je zlato u kolicima, agent pomera kolica ka izlazu
5. inače, agent pomera kolica kroz lavirint dok zlato ne bude pronađeno i to tako što ga pomera prvo kroz prvu vrstu, pa se spušta u krajnje desno polje druge vrste i na kraju iz prvog polja druge vrste prelazi u prvo polje treće vrste

8.2 Histeretični agenti

Agent može biti u nekom od stanja iz skupa unutrašnjih stanja I. Pretpostavimo da agent može dostići bilo koje stanje iz bilo kog drugog stanja te ovde nije potrebno particionisanje skupa I u podskupove (particije) ili definisanje senzorne funkcije. Isto tako pretpostavimo da agent može transformisati I u neki od elemenata istog u jednom koraku.

Važna razlika između tropističnih i histeretičnih agenata je to da funkcija akcije za histeretičnog agenta uzima u obzir interna stanja kao i opažanja u diktirajućim akcijama.

$$action : I \times T \rightarrow A$$

Kod histeretičnih agenata takođe postoji memorija koja pokreće funkciju koja preslikava unutrašnje i posmatrano stanje u sledeće untrašnje stanje.

$$internal : I \times T \rightarrow I$$

Histeretični agenti se definišu kao osmorka

$$(I, S, T, A, see, do, internal, action)$$

gde su :

- I skup unutrašnjih stanja
- S skup spoljašnjih stanja
- A skup akcija
- *see* funkcija iz S u T
- *do* funkcija iz $A \times S$ u S
- *internal* funkcija iz $I \times T$ u I
- *action* funkcija iz $I \times T$ u A

Kada je u pitanju naš primer, agenti sa senzornim ograničenjima dele skup od 90 stanja u tri podskupa. Prvi podskup sadrži 9 stanja u kojima je zlato u kolicima. Drugi se odnosi na 9 stanja u kojima su zlato i kolica u istom polju, ali da pri tom zlato nije u kolicima. I treći podskup sadrži 72 stanja u kojima je zlato u drugim poljima i pri to nije u kolicima. Funkcijom *see* preslikavamo svako stanje u odgovarajuću particiju (podskup). Sada definišemo skup unutrašnjih stanja koja ćemo predstaviti brojevima od 1 do 9 i koja označavaju polja u lavirintu(umesto podataka koji odgovaraju vrsti i koloni). I u ovom slučaju će situacije u kojima su kolica na izlazu a zlato negde dalje biti nemoguće.

8.3 Agenti nivoa znanja

Problem je što za ciljeve veštačke inteligencije nije potrebno predstavljati problem uvek na visokom nivou, tj. sa mnogo detalja (npr. nije potrebno znati svako kolo računara da bismo znali kako računar radi). Želimo samo dizajn u kojem su fizički elementi predstavljeni apstraktno.

U ovom poglavlju ispitujemo koncept agenata koje nazivamo *nivoi znanja* u kojem se sav višak detalja eliminiše. U ovoj apstrakciji unutrašnja stanja agenta sadrže samo baze podataka rečenica predikatskog računa, i agentove

mentalne akcije koje predstavljaju zaključke koji su dobijeni iz tih baza podataka. Funkcija akcije ,*action*, za agenta nivoa znanja, preslikava bazu podataka $\triangle$ i particiju skupa stanja T u akciju koja će biti izvršena od strane agenta .

$$action : D \times T \rightarrow A$$

Funkcija osvežavanja baze podataka *database* preslikava bazu podataka $\triangle$ i particiju stanja T u novu internu bazu podataka.

$$database : D \times T \rightarrow D$$

Agenta nivoa znanja predstavljamo osmorkom. Skup D ovde predstavlja proizvoljan skup predikatskog računa baze podataka, S je skup spoljašnih stanja, T je skup particija od S, A je skup akcija, *see* je funkcija iz S u T, *do* je funkcija iz $A \times S$ u S, *database* je funkcija iz $D \times T$ u D i *action* je funkcija iz $D \times T$ u D.

$$(D, S, T, A, see, do, database, action)$$

Odavde vidimo da je svaki agent nivoa znanja ujedno i histeretični agent. Celobrojne vrednosti kojima smo označili unutrašnja stanja u prethodnom poglavlju ovde zamenjujemo rečenicama predikatskog računa baze podataka. U našem primeru lavirinta imenujmo 9 polja simbolima AA, AB, AC, BA, BB, BC, CA, CB i CC. Imamo tri moguće particije stanja IC ("in the cart", u kolicima), SC ("in the same cell", u istom polju) i EW ("elsewhere", drugde). Uzmimo relacioni simbol $Cart$ koji označava unarnu relaciju koja sadrži polje na kojem su kolica, i simbol *Gold* koji označava unarnu relaciju koja sadrži onu particiju stanja koja se odnosi na lokaciju na kojoj je zlato. Umesto da startujemo sa 1 kao početnim stanjem mi startujemo sa sl. jednočlanim skupom :

$$Cart(AA)$$

Pošto je unutrašnje stanje promenjeno, treba da redefinišemo agentovu funkciju akcije tako da ona uzima u obzir bazu podataka umesto brojeva.
Takođe treba da definišemo funkciju baze podataka koja preslikava bazu podataka i particiju stanja u baze podataka koje odgovaraju celim brojevima u unutrašnjem stanju kod prethodnih agenata.

Za ove agente je karakteristično ekstremno ograničenje sposobnosti. Iako je njihovo ponašanje različito shodno položaju zlata ono predstavlja fiksiranu

pretragu u nalaženju zlata i sledi fiksiranu putanju do izlaza ako je zlato pronađeno. Modifikacija ove vrste nije moguća bez definisanja potpuno nove funkcije za agenta. Ako želimo modifikovati fizičkog agenta i kao i njegove funkcije implementirane u hardver, izmena će biti velika. Alternativa je definisanje fleksibilnijih agenata koji će biti programirani pomoću izmena rečenica u agentovoj bazi podataka. Ilustrujmo, kratko, primerom. Potreban nam je rečnik te koristimo simbole R, L, U i D za akcije desno, levo, gore i dole. Simbolima I i O označimo akcije stavljanja zlata u kolica i izlaska van lavirinta (in, out), te simbol N za null akciju. Sa $Must$ označimo akciju koju želimo da preduzme agent u datom trenutku.

$$Cart(AA) \wedge Gold(IC) \Rightarrow Must = R$$

$$Cart(AA) \wedge Gold(SC) \Rightarrow Must = I$$

$$Cart(AA) \wedge Gold(EW) \Rightarrow Must = R$$

$$\vdots$$

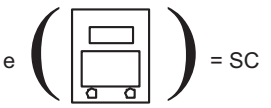
$$Cart(CC) \wedge Gold(IC) \Rightarrow Must = O$$

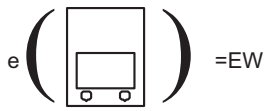
$$Cart(CC) \wedge Gold(SC) \Rightarrow Must = N$$

Pretpostavimo da početno stanje sadrži rečenicu koja opisuje lokaciju kolica u početnom stanju.

$$Cart(AA)$$

Definišimo, zatim, pomoćnu funkciju e. Levo ćemo navesti imena particija a desno akcija.





e(left) = L
e(right) = R
e(up) = U
e(down) = D
e(in) = I
e(out) = O
e(noop) = N

Kada baza podataka $\triangle$ sadrži rečenicu $Cart(\sigma)$ i

$$Cart(\sigma) \wedge Gold(e(t)) \Rightarrow Must = e(a)$$

tada agent izvršava akciju a .

$$action(\triangle, t) = a$$

Funkcija baze podataka diktira novu bazu koja sadrži sve rečenice stare baze osim one koja opisuje lokaciju kolica koja se kroz funkciju *next* prepravlja u novu lokaciju.

$$database(\triangle, t) = (\triangle - Cart(\sigma)) \cup Cart(next(\triangle, t))$$

Primećujemo da ovaj agent izvršava opisanu proceduru u svojoj početnoj bazi, pa zaključujemo da možemo izmeniti proceduru menjajući bazu podataka. Iako je oblik rečenica u opisu nešto stroži, možemo definisati i jednako moćne agente koji će biti mnogo fleksibilniji, a to ćemo videti u sl. poglavlju.

8.4 Agenti znanja u koracima

Agenti opisani u prethodnom poglavlju nisu monotoni: rečenice mogu biti izmenjene ili dodavane u bazu. Razlog za ovo je to što naš koncept relacija zavisnosti od stanja ne obuhvata stanje - na primer lokacija kolica. Svaka baza opisuje samo jedno stanje; posle svake akcije stanje je promenjeno i opis mora biti promenjen tako da se odnosi na stanje koje nastaje posle dejstva akcije.

Ovo razmatranje nameće pitanje da li je moguće dizajnirati monotone agente u kojima su nove rečenice dodane u unutrašnju bazu ali nisu premeštane (dodavanje da, premeštanje-uklanjanje ne). Ovo je zaista moguće, ali je potrebno napraviti neke izmene.

Prvo je potrebno primeniti koncept zasnovan na stanjima. Koristićemo relaciju T za opisivanje karakteristika individualnih stanja. Zatim treba da konvertujemo relacioni simbol, kao što je $Cart$, u funkcijski simbol; koristimo unarni funkcijski simbol Ext koji označava funkciju koja preslikava svaki

pozitivan ceo broj u spoljašnje stanje ciklusa agentove operacije koji odgovara tom celom broju. Primetimo da Ext preslikava ceo broj u spoljašnje stanje a ne u pariciju stanja. Sa ovim "rečnikom"možemo opisati početno stanje u primeru Lavirinta. Naravno ovaj opis neće biti kompletan jer ne uzima u obzir položaj zlata.

$$T(Cart(AA), Ext(1))$$

Ovaj "rečnik"možemo koristiti i za opis agentovih procedura prema prethodnom slučaju. U ovom slučaju koristimo promenljivu n koja označava redosled kroz ciklus agentove operacije i imamo konvertovan objekat konstante $Must$ u funkcijsku konstantu.

$$T(Cart(AA), Ext(n)) \ \wedge \ T(Gold(IC), Ext(n)) \ \Rightarrow Must(n) = R$$

$$T(Cart(AA), Ext(n)) \ \wedge \ T(Gold(SC), Ext(n)) \ \Rightarrow Must(n) = I$$

$$T(Cart(AA), Ext(n)) \ \wedge \ T(Gold(EW), Ext(n)) \ \Rightarrow Must(n) = R$$

$$\vdots$$

$$T(Cart(CC), Ext(n)) \ \wedge \ T(Gold(IC), Ext(n)) \ \Rightarrow Must(n) = O$$

$$T(Cart(CC), Ext(n)) \ \wedge \ T(Gold(SC), Ext(n)) \ \Rightarrow Must(n) = N$$

Na žalost ove izmene same po sebi nisu dovoljne da dozvole čisto monotono ponašanje. Agent ipak treba da zna koji ciklus se izvršava da bi koristio informaciju zabeleženu u bazi podataka. On ne može čuvati informacije vezane za tekući ciklus u svojoj bazi, pošto se informacije menjaju posle svake akcije. Alternativa je definisati novu vrstu agenta nivoa znanja u kojem unutrašnje stanje uključuje brojač isto kao i njegova baza rečenica. *Agenti nivoa znanja u koracima* su osmorka

$$(D, S, T, A, see, do, database, action)$$

gde su:

- D -skup baza podataka predikatskog računa
- S -skup spoljašnjih stanja
- T -skup particija od S
- A -skup akcija

- *see* -funkcija iz S u T
- *do* -je funkcija iz $A \times S$ u S
- *database* -je funkcija $D \times N \times T$ u D
- *action* -je funkcija iz $D \times N \times T$n u A

Primetimo da je jedina razlika između agenata nivoa znanja u koracima i običnih agenata nivoa znanja zavisnost baze podataka i funkcija akcija od agentovog rednog broja ciklusa. Redni broj ciklusa čuva se van baze podataka.

Sasvim je jednostavno modifikovati akcije i bazu funkcija za programibilne agente u prethodnom poglavlju tako da zadovoljavaju definiciju i uslove željenog ponašanja. Oni treba da budu malo komplikovaniji da bi upravljali promenljivima u bazi, a inače su identični.
Za cilj analize, često je korisno karakterisati kako se unutrašnje stanje, spoljašnje stanje, posmatranje i akcije u agentima nivoa znanja u koracima menjaju s obzirom na redni broj ciklusa. Funkcija $int_{\triangle,s}$ preslikava ceo broj n u unutrašnje stanje koje je rezultat n-tog ciklusa agenta nivoa znanja u koracima sa početnom bazom podataka $\triangle$ i početnm spoljašnjim stanjem s. Funkcija $ext_{\triangle,s}$ preslikava ceo broj u spoljašnje stanje koje je rezultat aktivnosti u n-tom ciklusu. Funkcija $obs_{\triangle,s}$ preslikava ceo broj n u skup stanja posmatran od strane agenta u n-tom ciklusu. Funkcija $act_{\triangle,s}$ preslikava ceo broj n u akciju koju uzima agent u n-tom ciklusu.

Pogledajmo šta se dešava sa početnim vrednostma. Unutrašnje stanje prvog ciklusa agentovih opercija je agentova početna baza podataka i spoljašnje stanje prvog cikusa je početno spoljašnje stanje. Prvo agentovo opažanje se odnosi na primenu funkcije *see* u početnom spoljašnjem stanju, i agentova prva akcija je određena njegovom početnom bazom označenom brojem 1 i agentovim početnim opažanjem.

$$int_{\triangle,s}(1) = \triangle$$

$$ext_{\triangle,s}(1) = s$$

$$obs_{\triangle,s}(1) = see(s)$$

$$act_{\triangle,s}(1) = action(\triangle, 1, see(s))$$

Definicije za ove funkcije slede posle prvog ciklusa. Unutrašnje stanje u svakom ciklusu je rezultat delovanja agentove funkcije memorije u prethodnom unutrašnjem stanju, prethodnom (rednom) broju ciklusa, i agentovog posmatranja prethodnog spoljašnjeg stanja. Spoljašnje stanje je rezultat izvršavanja akcije označene u prethodnom ciklusu prethodnim spoljašnjim stanjem. Agentovo opažanje (posmatranje) je particija stanja koja sadrži spoljašnje stanje. Akcija koja će biti izvršena je određena primenom funkcije *action* u tekućem unutrašnjem stanju, tekućem broju ciklusa, i agentovim opažanjem tekućeg spoljašnjeg stanja.

$$int_{\triangle,s}(n) = database(int_{\triangle,s}(n-1), n-1, obs_{\triangle,s}(n-1)$$

$$ext_{\triangle,s}(n) = do(act_{\triangle,s}(n-1), ext_{\triangle,s}(n-1)$$

$$obs_{\triangle,s}(n) = see(ext_{\triangle,s}(n))$$

$$act_{\triangle,s}(n) = action(int_{\triangle,s}(n), n, obs_{\triangle,s}(n))$$

Agent nivoa znanja sa početnom bazom $\triangle$ i početnim spoljašnjim stanjem postoji ako i samo ako njegova baza podataka postoji u svakom ciklusu. Agent nivoa znanja pamti bazu podataka (*database retentive*) ako i samo ako njegova baza u svakom ciklusu posle prvog logički implicira bazu prethodnog cikusa.

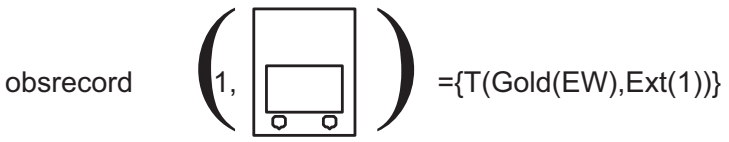
$$int_{\triangle,s}(n) \models int_{\triangle,s}(n-1)$$

Jednostavniji tip (*database retentive*) agenata koji pamte bazu podataka je onaj kod kojeg su sve rečenice iz $int_{\triangle,s}(n-1)$ sadržane u $int_{\triangle,s}(n)$

8.5 Agenti s namerom

Posmatrajući agente opisane u prethodnom poglavlju, interesantno je primetiti da pod uobičajenom interpretacijom simbola u rečniku ovog agenta baza podataka svakog ciklusa korektno opisuje svoje spoljašnje okruženje. Pošto se agent pomeri desno u početnom stanju kolica su u polju AB kao što je specificirano u bazi podataka tog ciklusa. Ako bi permutovali baze podataka sistematski i modifikovali bazu podataka agenta kao i funkcije akcije, agent bi rešio problem podjednako dobro, ali rečenice u bazi podataka će biti netačne pod uobičajenom interpretacijom. Sa druge strane, analizirajući nivo znanja agenta mi obično želimo da pričamo o ponašanju agenta uzimajući u obzir neke interpretacije ili parcijalne interpretacije za rečenice u bazi podataka. Uopšte, ne možemo očekivati od agenta da se povinuje našoj interpretaciji

za sve simbole u njegovom rečniku. Ipak, interesantno je pogledati agentove osobine ako pretpostavimo da se slažu sa nama u pogledu nekih simbola u njegovom rečniku. Sledeće veze su posebno korisne.
Funkcija *obsrecord* preslikava pozitivne cele brojeve n i particiju stanja T u skup rečenica tvrdeć da je spoljašnje stanje u ciklusu n član particije T. U prethodnom primeru posmatranje prvog ciklusa i particije stanja u kome se zlato nalazi na nekom drugom mestu je baza podataka koja se sastoji od jednostruke rečenice $T(Gold(EW), Ext(1))$.



Da bismo kodirali naredbe u bazi podataka agenta potreban nam je rečnik koji opisuje akcije koje bi agent *trebalo da* radi. Funkcija *mustrecord* preslikava pozitivan ceo broj n i akciju a u skup rečenica tvrdeći da bi agent trebalo da izvrši akciju a u ciklusu n. Npr. možemo kodirati činjenicu da se agent kreće desno u svom prvom ciklusu:

$$mustrecord(1, right) = Must(1) = R$$

Funkcija *mustnotrecord* preslikava pozitivan ceo broj n i particiju stanja T u skup rečenica tvrdeći da bi agent trebalo da izbegava akciju a u ciklusu n. Npr.,možemo kodirati činjenicu da se agent ne pomera desno u svom prvom ciklusu kao što je prikazano :

$$mustnotrecord(1, right) = Must(1) \neq R$$

Funkcija *actrecord* preslikava pozitivan ceo broj n i akciju a u skup rečenica tvrdeći da agent u stvari izvršava akciju a u ciklusu n. Možemo npr.,kodirati činjenicu da se agent pomera desno u svom prvom ciklusu rečenicom :

$$act(1) = R$$

$$actrecord(1, right) = Act(1) = R$$

Kao ostale aspekte operacija agenta, pogodno je da se napravi koncept funkcija koji definiše zapise (record) za opažanje i akcije agenta. Definišemo $obsrec_{\triangle,s}$ koja preslikava broj ciklusa u zapis posmatranja za n-ti ciklus

aktivnosti agenta nivoa znanja sa početnom bazom podataka $\triangle$ i početnim spoljašnjim stanjem s. Funkcija $actrec_{\triangle,s}$ preslikava broj ciklusa u odgovarajući zapis akcije. Koristeći terminologiju iz zadnjeg odeljka možemo definisati

$$obsrec_{\triangle,s} = obsrecord(n, obs_{\triangle,s}(n))$$

$$actrec_{\triangle,s} = actrecord(n, act_{\triangle,s}(n))$$

Kažemo da je agent (*observation retentive*) pamti opažanja ako i samo ako zapisuje svoja zapažanja u svakom ciklusu u svoju bazu, tj. u svakom ciklusu, posle prvog, agentova baza logički povlači zapis opažanja prethodnog ciklusa.

$$int_{\triangle,s}(n) \models obsrec_{\triangle,s}(n-1)$$

Agent čuva (pamti) akcije (*action retentive*) ako i samo ako je zapis njegovih akcija u svakom ciklusu u njegovoj bazi, tj. ako u svakom ciklusu, posle prvog, agentova baza logički povlači zapis akcije prethodnog ciklusa.

$$int_{\triangle,s}(n) \models actrec_{\triangle,s}(n-1)$$

Kažemo da baza podataka $\triangle$ *zapisuje* akciju a u ciklusu n agentove operacije ($P(\triangle, n, a)$) ako i samo ako $\triangle$ logački povlači da akcija mora biti izvršena u n-tom koraku.

$$\triangle \models mustrecord(n, a)$$

Koristeći ovu notaciju možemo definisati šta se podrazumeva pod zabranjenom akcijom. Kažemo da $\triangle zabranjuje$ akciju a u n-tom ciklusu agentovih operacija ($F(\triangle, n, a)$, ako i samo ako *triangle* logički povlači da akcija a ne sme biti izvršena u koraku n.

$$\triangle \models mustnotrecord(n, a)$$

Agent nivoa znanja je *lokalno* ,,veran" (locally faithful) ako i samo ako svaki ciklus njegovih operacija zadovoljava sl.uslove:

1. Agent izvršava svaku akciju koja je zapisana pomoću agentove baze podataka i njegovih opažanja u tekućem stanju.

$$P(int_{\triangle,s}(n) \cup obsrec_{\triangle,s}(n), n, a) \Rightarrow act_{\triangle,s}(n) = a$$

2. Agent izbegava (poništava) svaku akciju koja je zabranjena pomoću njegove baze podataka i njegovog opažanja u tekućem stanju.

$$F(int_{\triangle,s}(n) \cup obsrec_{\triangle,s}(n), n, a) \Rightarrow act_{\triangle,s}(n) \neq a$$

Kod nekih agenata nivoa znanja ovi su uslovi redukovani. Npr. pretpostavimo da agentova baza ima aksiome koje tvrde da postoji samo jedna zapisana akcija za svaki ciklus i pretpostavimo da, takođe, postoje aksiome koje tvrde nejednakost agentovih različitih akcija. Tada, ako baza odredi (zapiše) akcije za svaki ciklus, agent zabranjuje sve ostale akcije; i ako baza zabrani sve akcije osim jedne, neophodno je odrediti (zapisati) remaining akcije. Sa druge strane, ne možemo izostaviti uslove vezane za zabranjene akcije. Zato postoji baza za zabranjene akcije u kojoj nisu zapisane ostale akcije i mi želimo biti sigurni da agent neće selektovati zabranjenu akciju. Slično, ne možemo ništa raditi bez zapisanih uslova pa zato postoje baze koje zapisuju neke akcije koje ne zabranjuju druge akcije i mi ne želimo da agent izvršava samo nezabranjenu akciju kada postoje neke druge (određene) zapisane akcije.

Teorema 9 *Saglasnost je neophodan uslov za lokalnu vernost.*

Istorijski zapis (history record) za partikularne korake agentove operacije je skup opažanja i akcija koje se zapisuju za svaki korak i za sve prethodne korake. Funkcija *histrec* preslikava broj u odgovarajući istorijski zapis.

$$histrec_{\triangle,s}(n) =$$

$$= \begin{cases} \{\ \} & n = 0 \\ histrec_{\triangle,s}(n-1) \cup obsrec_{\triangle,s}(n) \cup actrec_{\triangle,s}(n) & \text{inače} \end{cases}$$

Primetimo da prethodne informacije vezane za istoriju u agentovoj bazi često dopuštaju agentu izvođenje zaključaka koji inače ne bi bili mogući. Npr., posle uočavanja da zlato nije u ćeliji AA i posle pomeranja u ćeliju AB, agent može zaključiti da zlato nije locirano u AA, iako on nije dugo razmatrao tu činjenicu. Agent koji razmišlja je *globalno veran* (globally faithful) akko on postupa u skladu sa njegovom početnom bazom, njegovom istorijom i tekućim opažanjem, tj.

1. agent izvršava svaku akciju koja je odeđena njegovom početnom bazom, istorijom i opažanjem u tekućem stanju

$$P(\triangle \cup histrec_{\triangle,s}(n-1) \cup obsrec_{\triangle,s}(n), n, a) \Rightarrow act_{\triangle,s}(n) = a$$

2. agent izbegava svaku akciju koja je zabranjena njegovom početnom bazom, istorijom i opažanjem u tekućem stanju

$$F(\triangle \cup histrec_{\triangle,s}(n-1) \cup obsrec_{\triangle,s}(n), n, a) \Rightarrow act_{\triangle,s}(n) \neq a$$

Teorema 10 *Pamćenje baze podataka, opažanja i akcija i lokalna vernost impliciraju globalnu vernost.*

8.6 Promišljeni agenti

U ovom poglavlju definišemo klasu nešto specifičnijih agenata nivoa znanja od globalno vernih. Ključna ideja u definisanju agenata ove klase je korišćenje metode automatskog zaključivanja kao što su rezolucija u proizvođenju rečenice koja ukazuje na traženu akciju u svakom ciklusu. Agent ove vrste je *promišljen* u tome što razmišlja u svakom ciklusu o spoljašnjoj akciji koju treba izvršiti. Ako je u ciklusu n moguće dokazati *mustrec(n,a)* za tekuću bazu i zapis opažanja koristeći rezoluciju ili drugu proceduru zaključivanja, tada agent izvršava akciju a.

$action(\triangle, n, t) = a$
whenever $\triangle \cup obsrecord(n, t) \models mustrecord(n, a)$

Agentova baza je ažurirana usled opažanja i akcija u ciklusu.

$database(\triangle, n, t) = \triangle \cup obsrecord(n, t) \cup actrecord(n, a)$
whenever $\triangle \cup obsrecord(n, t) \models mustrecord(n, a)$

```
Procedure     CD(DB)
Begin         CYCLE := 1
Tag           OBS  := OBSERVE(CYCLE)
              DB := APPEND([T(OBS,Ext(CYCLE)=k,DB)
                         ACT :=  FIND(k,Must(CYCLE)=k,DB)
                         EXECUTE(ACT)
                         DB := APPEND([Act(CYCLE)=ACT],DB)
                         CYCLE := CYCLE+1
              GOTO Tag
End
```

Program `CD` uzima početnu bazu kao argument i manipuliše sa 4 promenljive: `CYCLE` - broj tekućeg satnja, `OBS` - opisivač stanja, `DB` - čuva početnu bazu, sva opažanja i zapise akcija i `ACT` - je ime akcije koja će biti izvršena. Agentova senzorna sposobnost je implementirana u potprogramu `OBSERVE` (argument joj je redni broj ciklusa i kada je izvršena u stanju s vraća kao vrednost $obsrecord(n, see(s))$). Agentov "rečnik efikasnosti" je implementiran u primitivnoj potprogramu `EXECUTE` (argument joj je oznaka akcije i kada je pozvana izvršava odgovarajuću akciju).

Kod definiše jednostavnu slobodno-izlaznu petlju. U svakom trenutku petlje, agent prolazi kroz jednostavan ciklus njegove istorije. Prvo, okolina je posmatrana i u bazu je upisana karakteristična rečenica. Tada agent izvodi zaključak na bazi dok dedukuje akciju za izvođenje. To povlači akciju i ažuriranje baze i broja ciklusa. Tada se ciklus ponavlja.
Iz ove definicije lako se vidi da je promišljeni agent pamti opažanja, pamti akcije i pamti bazu podataka.

Stoga imamo:

Teorema 11 *Svaki promišljeni agent sa valjanim i kompletnim dokazivačem teoreme je globalno tačan (veran).*

9 Klasične metode rešavanja problema

Mogu se izdvojiti četiri osnovna načina rešavanja problema:

1. Primena eksplicitno zadate formule koja nalazi rešenje
2. Upotreba rekurzivne definicije
3. Upotreba algoritma koji konvergira ka rešenju
4. Upotreba određenih procesa npr. pokušaja i greške sa nabrajanjem slučajeva

Vidimo da je najbolje kada možemo primeniti prvi način za nalaženje rešenja. U tom slučaju, složenost je merena naporom da se izračuna gotova formula koja uključuje samo konačan broj simbola, pa je tako bez obzira šta su ulazni parametri, složenost O(1).

Primeri algoritama (uglavnom polinomijalne složenosti):

Primer 1. Izračunavanje sume prvih n prirodnih brojeva:

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

izračunavanje sume kvadrata prvih n prirodnih brojeva:

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$$

Ovi primeri se lako rešavaju pomoću date eksplicitne formule (složenost je konstantna tj. $O(1)$ i odnose se na prvi način pronalaženja rešenja).

Primer 2. Fibonačijevi brojevi (Leonard de Pise, Bonaccij-jev sin 1540 .)

f *Rekurzivna definicija*

$F(n) = F(n-1) + F(n-2)$, $F(2) = F(1) = 1$

Za n=30 npr. potrebno je 832040 izračunavanja tj. rekurzivno računanje

je $O(F(n))$, što je veoma skupo.

Iterativno rešenje glasi :

```
Fibonacci(n)
  i:=2; u:=1; v:=1;
  repeat while i!=n
    i:=i+1; w:=u; u:=u+v; v:=w;
  return u;
end Fibonacci
```

Složenost je $O(n)$ - potrebno je izvršiti n-2 koraka sabiranja.

Eksplicitno rešenje rekurentne relacije se traži u obliku $F(n) = r^n$ preko karekteristične jednačine. Eksplicitna formula glasi :

$$F(n) = \frac{[(1+\sqrt{5})^n - (1-\sqrt{5})^n]}{2^n\sqrt{5}}$$

Problem sa ovom formulom je taj da je rezultat prilično veliki broj s pokretnim zarezom i samim tim s velikom greškom a traži se ceo broj tako da je to nepraktično u ovom slučaju.

Primer 3. Sortiranje poređenjem. Skup od n brojeva može biti uređen na $n!$ načina (permutacije). Stablo pretrage za proizvoljni algoritam će imati 2^t listova sa t poređenja. Kako je $2^t > n!$, prema Stirlingovoj formuli koja tvrdi da je za veliko n $n!$ reda $n^{\frac{n+1}{2}}$ pa je $t = O(n \log n)$. Klasičan ,,bubble sort" algoritam je $O(n^2)$ što je lošije od teoretskog $O(n \log n)$.

Bolji algoritam možemo dobiti ,,takmičenjem" parova brojeva pri čemu redukujemo broj poredjenja. Dubina stabla je tada $\log_2 n$. U svakom koraku polovimo broj elemenata (uz upotrebu ,,heap" struktura) koji se porede, i u najgorem slučaju, ukupan broj potrebnih poređenja je $n \cdot broj$ potrebnih pretraga stabla od n elemenata, tj. $O(n \log n)$ što je najbolje moguće rešenje. Sortiranje je primer netrivijalnog rešenja za koje možemo dati polinomijalni algoritam.

Primer 4. Pronalaženje najkraćeg puta između tačaka mreže (grafa). Krajnje je nepraktično nabrajati sve moguće putanje i tražiti najkraću - za početak uzeti samo elementarne putanje (bez samopresecanja, tj.

svaki čvor se pojavljuje najviše jednom). Može se krenuti od početne tačke ka susedima pa onda dalje iterativno - za svaki stupanj se tako zna putanja minimalne dužine (bez potrebe za rekurzijom tj. backtracking-om). Time se čvorovi grafa dele u dve particije: S^* - gde se za sve čvorove zna minimalno rastojanje od čvora 1, S - gde to još nije poznato. Na početku $S^* = \{1\}$. Ako je 1 polazna tačka, a n završna tačka putanje koja se traži u grafu $(\{1, ..., n\}, U)$ td. $U \subseteq \{1, ..., n\}^2$, $D^*(i)$ =udaljenost od 1 do čvora $i, i \in S^*$, $D(i)$ =najkraće rastojanje od 1 do $i, i \in S$ na datum stupnju, $L(i, j)$ = pozitivno rastojanje za luk (i,j). Tada na svakom stupnju važi:

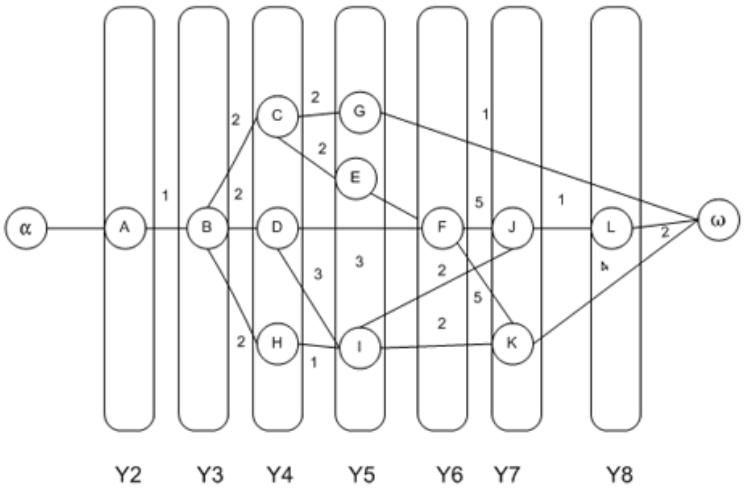
ako $i \in S^*$ onda $D(i) = D^*(i)$
inače $i \in S$ i važi $D(i) = min[D(k) + L(k, i)], k \in S^*, (k, i) \in U$
Na početku je $S^* = \{1\}$, $D^*(1) = 0$, $D(i) = \infty, i \neq 1$.
Tada, ako je $D(j) = \min_{i \in S}[D(i)]$ za neko $j \in S$ onda je $D(j) = D^*(j)$ najkraći put od 1 do j (dokaz po konstrukciji puta iz dva dela, od 1 do prve tačke u S i ostatak do j). Algoritam koji sledi iz ovoga:

```
D(1):=0; S:={2,3,...,n};
do ∀i ∈ S: if (1,i) ∈ U then D(i):=L(1,i) else D(i):= ∞;
while S ≠ ∅ repeat choose j ∈ S such that D(j) = min_{i∈S} D(i);
  S:=S-{j};
  do ∀i ∈ S and (j,i) ∈ U
    D(i):= min[D(i), D(j) + L(j,i)];
  end repeat;
end
```

Algoritam su prvi dali Mur 1957. i Dijkstra 1959., varijacije algoritma su dali Dantzig 1960. i Whiting i Hiller 1960. Složenost algoritma se može oceniti sa $\sum_{k=1}^{n}(n-k)$ tj. $O(n^2)$. Ako se pretpostavi da je graf orijentisan postupak je isti, za razliku od sledećeg primera:

Primer 5. Problem redosleda poslova / zadataka (task-ordering problem). Ako uređeni graf predstavlja redosled procesa sa vremenima potrebnim za njihovo obavljanje onda je cilj naći putanju sa maksimumom zbira vremena tj. vreme za koje je minimalno potrebno da se obavi ceo proces (od tačke α to tačke ω npr.). Pod uslovom da nema ciklusa u grafu, ideja je slična prethodnom $D(i) = \max_{j \in J}[D(j) + L(j, i)]$ se

traži iterativno, ali je moguće naći redosled po kome tražiti čvorove da bi se dobila najbolja konvergencija. Uvek postoji polazni čvor tj. čvor bez ulaznih lukova ako je graf uređen. Ako nema ciklusa onda se može pokazati da postoji bijekcija ν iz skupa tačaka grafa X u $\{1, ..., n\}$ za svaku putanju od α do ω td. $(\forall(x_1, x_2) \in U)\nu(x_1) < \nu(x_2)$. Tako se čvorovi klasifikuju u ,,slojeve" Y_k (ako se iz grafa najpre izbaci početni čvor zajedno sa lukovima koji ga povezuju s narednim onda naredni predstavljaju nove početne čvorove u novom ,,sloju"):



Ovakav graf sadrži potrebne informacije:
α-početak procesa, ω-kraj procesa, A, B, C ... - su zadaci, a brojevi predstavljaju vreme u satima potrebno za izvršenje zadataka. Usmerenje grafa govori o mogućem redosledu izvršavanja zadataka. Problem se svodi na nalaženje puta od α do ω za koje je suma vrednosti grana (težina grana) maksimalna (u suprotnom neki od poslova ne bi bili završeni u datom vremenu).

Prethodni algoritam koji se odnosio na pronalaženje minimalnog puta može biti modifikovan za rešavanje ovog problema, uzimajući u obzir odsustvo ciklusa i prolazak kroz granu samo jednom. Proces započinje zato što važi sledeće svojstvo:

Svaki konačan graf bez ciklusa ima najmanje jedan čvor koji nema predhodnika (izvor).

Teorema 12 *Konačan usmeren graf je bez ciklusa ako i samo ako postoji bijekcija v skupa čvorova X u interval $\overline{1.n}$, gde je n ukupan broj čvorova u X takav da*

$$\forall x(x_1, x_2), (x_1, x_2) \in U : v(x_1) < v(x_2)$$

Ovaj problem rešavamo reorganizacijom grafa. Čvorove grupišemo u slojeve tako da ne sadrže čvor koji ima predhodnika (ne postoji grana koja spaja dva čvora unutar jednog sloja).
Ovaj metod je poznat kao metod potencijala-Bernard Roy 1960.
Algoritam:

```
p:=1; T:=X; K;=0;
while T ≠ ∅ repeat ( Y_k je skup polaznih) izvora u T
  do (∀i)i ∈ Y_k
     ν(i) := p ; p:=p+1;
    do (∀j)j ∈ X, (i,j) ∈ U
        izbaci (i,j) iz U;
  T:= T − Y_k ; k:=k+1;
end repeat;
```

Nakon ovakve konstrukcije važi da ako pomenutom bijekcijom prenumerišemo graf td. $(i, j) \in U \Rightarrow i < j$. Algoritam je onda jednostavan:

```
D*(1) :=0;  D*(2) :=L(1,2);
repeat for j, 3<j<n:  D*(j) := max_{(i,j)∈U}[D*(i) + L(i,j)];
end repeat;
```

Algoritam je takođe složenosti $O(n^2)$.

Faze koje se nameću u rešavanju problema tj. primera u ovom poglavlju:

1. Analiza problema koja podrazumeva stavljanje u oblik koji je lakši za razumevanje
2. Proučavanje problema izraženog u ovom jednostavnijem prostoru, pomoću metoda koje su vođene jednostavnim idejama proizašlim iz primera
3. Generalizacija
4. Konstrukcija i dokaz algoritma za rešavanje

Primer 8 Topološki problem Ojlervoih ciklusa
(ponatiji kao problem Kenigzberških mostova, nad rekom Pregel u gradu Königsberg) Treba pronaći ako je moguće putanju koja počinje u nekoj tački grafa i prolazi kroz svaki luk grafa tačno jednom i vraća se na početak. Preduslov je da je graf povezan. Potreban uslov je takođe i da je stepen svakog čvora paran (da bi za svaki dolazak u teme postojao i različit odlazak), a to je ujedno i dovoljan uslov.

Dokaz se izvodi matematičkom indukcijom:
(IH) Povezani graf sa manje od m čvorova u kojem svi čvorovi imaju paran stepen sadrži Ojlerov put i umemo da ga nađemo.
Posmatrajmo $G(V, E)$ sa m grana. Neka je P neki ciklus u G i G' graf dobijen uklanjanjem grana koje čine P iz grafa G. Stepeni svih čvorova u G' su parni jer broj uklonjenih grana susednih bilo kom čvoru je paran (kada uđemo u čvor, moramo i izaći iz njega). **(IH)** se ne može primeniti na G' jer on ne mora biti povezan. Izdvojimo komponente povezanosti $G'_1, G'_2 ... G'_k$i (IH) primenimo na njih. Izdvojene zatvorene Ojlerove cikluse označimo sa $P'_1, P'_2 ... P'_k$ i sada je samo potrebno da ih objedinimo u jedan povezani graf. Ovaj problem je dosta sličan Hamiltonovom problemu, gde je to problem pronalaženja ciklusa kroz čvorove, ne kroz grane (Hamiltonov ciklus je specijalan slučaj Ojlerovih gde se prolazi samo jednom kroz svaki čvor). Međutim, taj problem pripada drugoj klasi problema.

Opis postupka: prilikom svakog prolaska kroz neki luk on se izbacuje (označava). Ako se pretpostavi da ne može da se napravi takva putanja ali se završava u V, ostaje nekoliko povezanih podgrafova G_i koji se

sustiču sa putanjom P u čvorovima g_i. Ako se polazi od čvora V, konstruiše se putanja P od V do g_1, Ojlerov ciklus E_1 u grafu G_1 koji počinje i završava se u g_1, a onda dalje od g_1 do g_2 itd. sve dok se ne iscrpe svi podgrafovi, onda povratak u V. Petlja na izolovanom čvoru je takođe Ojlerov ciklus (Euler, 1736. je dao prvi dokaz).

```
Euler(w);
  u:=w; P:=null;
  repeat while exists {(u,v) je prvi luk iz u koji nije označen}
    označi (u,v);
    P:=P ⋃ {(u,v)};
    u:=v;
  until u=w;
  return P;
end Euler;;
```

Sada je lako zapisati algoritam:

```
P:=Euler(S);
repeat while postoje  neoznačeni lukovi u G;
  [H je čvor koji i na putanji P i na neoznačenom luku]
  spoj putanju Euler(H) sa P;
end repeat;
```

Algoritam je složenosti $O(|U|)$, dok je algoritam provere potrebnog i dovoljnog uslova $O(n)$.

9.1 Algoritmi za koje su poznata polinomijalna rešenja

Daleko je manji broj klase algoritama za koje je poznato da su polinomijalne složenosti u odnosu na ostale. Za mnoge od njih je tek u zadnjih 20-30 godina otkriveno polinomijalno rešenje i dokazano da jesu optimalno takvi. To barem daje opravdanje za dalja istraživanja u oblasti VI. Još neki primeri sa njihovim složenostima su:

- Pronalaženje reči u tekstu od n reči - $nO(n)$
- Konstrukcija drveta najmanje cene za graf sa m lukova - $O(m \log m)$

- npr. vodovod za zadate moguće putanje (Kruskal 1956, Prim 1957, Tarjan 1977)

- Pronalaženje najkraće putanje između zadatih čvorova sa n čvorova i m lukova - $O(mn)$ (Dijkstra 1959, Dantzig 1960, Floyd 1962, Ford 1965)

- Povezane komponente - $O(n^2)$ - pronaći maksimalno povezane podgrafove datog grafa n čvorova (Tremaux 1882, Tarjan 1972)

- Tranzitivno zatvorenje grafa (relacije) - $O(n^2)$

- Maksimalno poređenje - $O(n^{5/2})$ - cilj je naći podksup svih lukova td. se ne sustiču ni u jednom temenu (teorema Claude Berge-a daje osnovu, Jack Edmonds je našao polinomijalno rešenje)

- Maksimalni protok - $O(n^3)$ - ako su lukovi označeni protokom određenog tipa, naći maksilan između dva čvora (Ford, Fulkerson 1950, Gondran, Minoux 1978)

- Testiranje planarnosti grafa - O(n) - da li je moguće prikazati graf u ravni bez presecanja lukova (Kuratowski 1930 $O(n^6)$, Hopcroft, Tarjan $O(n \log n)$, $O(n)$ 1970-1974)

- Linearno programiranje - rešiti u $\mathbb{R}^n$: $Ax \leq b$, $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, a može se i dodati uslova da neka realna funkcija cx ima minimalnu vrednost; koristi se u operativnim istraživanjima (Dantzig-ov algoritam, simpleks metoda, zbog konveksnosti prostora pretraživanja konvergira ka optimumu tek kada se prođe kroz određen broj tačaka što je $O(n^m)$ ali u praksi više kao $O(n^3)$; L. G. Kachian 1979. je našao algoritam polinomijalne složenosti koji je kao posledicu imao mnoge algoritme numeričke analize zadate tačnosti - iste složenosti)

9.2 Klasifikacija problema prema složenosti

Složenost procedure definišemo kao gornju granicu broja elementarnih operacija potrebnih za rešavanje, izraženih u funkciji veličine ulaznih podataka. Složenost problema je složenost najbolje poznate procedure za njegovo rešavanje.
Nameću se dva pitanja:

- Do kojih granica se može poboljšati neki algoritam?
- Da li složenost može sugerisati grupisanje u klase?

Navedimo klasifikaciju problema shodno složenosti:

1. polinomijalni algoritmi (sastoje se od svih problema čiji su algoritmi poznati i složenost im je polinomijalna funkcija veličine ulaznih podataka)
2. suštinski eksponencijalni algoritmi (složenost ove klase je najmanje reda f^n, gde je f ili konstanta ili polinom od n
3. problemi koji nisu ni polinomijalni ni eksponencijalni (za ovu klasu važi da se za njih ne zna nijedan algoritam polinomijalne složenosti)

Navedimo samo neke od problema za koje su poznati algoritmi polinomijalne složenosti: sortiranje skupa od n brojeva $O(n \log n)$, pronalaženje Ojlerovog ciklusa $O(n)$, konstrukcija minimalno povezanog stabla $O(m \log n)$, najkraći put između čvorova grafa od n čvorova i m grana $O(mn)$ itd.

Ako je n mera veličine podataka (dužina niza cifara npr. u operaciji sabiranja) i ako je broj koraka algoritama oblika $ax + b$ onda je algoritam složenosti $O(n)$, linearan. Linearni algoritmi su obično najbolji po pitanju složenosti. Osnovne klase algoritama prema složenosti su:

- **klasa P: polinomijalni algoritmi**
  Klasa ,,dobrih" problema, za koje postoji poznat algoritam složenosti $O(n^r)$ gde r ne zavisi od n.
- **klasa E: suštinski eksponencijalni algoritmi**
  algoritam čija je složenost barem $O(f^n)$ - stvar može biti i gora, jer f ne mora biti samo konstanta ili polinom, već takođe eksponencijalna funkcija ($2^{2^{\cdots^n}}$)

- **klasa III: problemi koji nisu ni P ni E**
  Ništa u njihovoj formulaciji ne ukazuje da su suštinski eksponencijalni, niti je nađen polinomijalan algoritam za njihovo rešavanje. Primeri:
  - Ceolobrojne (Difoantove) jednačine
  - Traženje ciklusa koji prolazi kroz svaki čvor datog grafa samo jednom (Hamiltonov problem, nasuprot Ojlerovom gde je uslov da ciklus prolazi kroz svaki luk)
  - postojanje skupa logičkih vrednosti koje zadovoljavaju logički izraz (Cook, 1971)
  - Problem optimizacije putovanja putujećeg trgovca (u narednom poglavlju)
  - Problem biranja fajlova u nestruktuiranoj bazi radi pretrage (traženja nekog elementa) po najmanjoj ceni
  - optimalno pakovanje po najmanjoj ceni
  - dijagnoza / troubleshooting (npr. kod ekspertnih sistema)
  - itd.

9.3 klasa NP: nedeterministički polinomijalni problemi

Tjuringova deterministička mašina (DTM kojom se npr. formalno zasniva pojam algoritma i može se pokazati da je ekvivalentan sličnim formalizmima kao što su rekurzivne funkcije) je automat sa stanjem, programom i domenom koji vrši u svakom koraku neku operaciju nad domenom koja zavisi od stanja i promeni stanje. Pored te mašine postoji NDTM (nedeterministička Tjuringova mašina) koja se u odnosu na DTM razlikuje u samo jednoj dodatnoj instrukciji *choice[S]*. Ova instrukcija kreira onoliko kopija mašine koja je izvršava koliko ima elemanata u S i dalje nastavlja izvršanje paralelno sve dok jedna od kopija ne izvrši komandu *stop*. Ovako nešto je veoma korisno za probleme koji se rešavaju nabrajanjem tj. ispitivanjem svih mogućnosti pokušavanjem i greškom („trial and error") sve do rešenja.

Primer 1 - izvodljivost logičkog iskaza sa promenljivama q_1, ..., q_n - iskaz je izvodljiv ako postoje vrednosti promeljivih tako je njegova vrednost $\top$. Skica algoritma za iskaz $E(q_1, ..., q_n)$:

```
repeat for i,  1 ≤ i ≤ n:
   q_i  := choice[T,F];
end repeat;
if E(q_1, ..., q_n) then IZVODLJIV; else NIJE;
stop;
```

Tako nastaje 2^n kopija.

Primer 2: bojenje mape pomoću 3 boje - oblasti su R_1, ..., R_n od kojih su svake dve susedne različite b oje:

```
boja( R_1 ):=c1;
repeat for i,  1 ≤ i ≤ n:
  if R_1 šusedna" R_i then nadji boju za R_i
end repeat;
repeat for k,  1 ≤ k ≤ n: while postoje regioni koji nisu obojeni
  if R_k obojiva samo bojom j then boja( R_k ) := j;
  else R_min je region sa najmanjim indeksom
            za kojeg su boje j1 i j2 raspoložive;
      boja( R_min ) := choice[j1,j2];
end repeat;
stop;
```

Svaka kopija je DTM, ako je pri tom svaka klase P onda je problem NP.

$$P \subset NP$$

Neki put je potrebna veoma mala promena parametra algoritma da bi postao P umesto NP (npr. ako se u drugom primeru koriste dve umesto tri boje). U prvom primeru ako je iskaz dat u normalizovanoj formi i ako se algoritam drugačije napiše može se dobiti algoritam koji je polinomijalan u odnosu na broj elemanata normalizovane forme. Pored pomenutih, primeri su i:

- Pokrivanje datog skupa - za datu familiju $\mathfrak{F}$ podskupova E_i skupa E naći podfamiliju $\mathfrak{G}$ td.
$$\bigcup_{\mathfrak{G}} E_i = \bigcup_{\mathfrak{F}} E_i$$
- Particija skupa: slično prethodnom, td. su E_i proizvoljni skupovi iz $\mathfrak{G}$ i $E_j \bigcap E_k = \emptyset$ za svako $j \neq k$ (disjunktni)

- Pronalaženje klike od k čvorova neuređenog grafa: klika je skup čvorova sa lukom za svaki par, k-klika je klika sa k čvorova.
- Egzistencija Hamiltonovog ciklusa za neuređen graf
- ,,knapsack": naći $x_i \in \{0, 1\}$ za date cele brojeve a_i, b td.

$$\sum_{i=1}^{n} a_i x_i = b$$

  ili uopšte, rešavanje Diofantove jednačine

- Binarno particioniranje skupa $S = \{y_1, ..., y_n\}$ celih brojeva na dva podskupa S_1 i S_2 td.

$$\sum_{i \in S_1} y_i = \sum_{i \in S_2} y_i$$

- Problem putujućeg trgovca za uređeni graf i cenu manju od zadate je problem Hamiltonovog ciklusa sa cenom manjom od zadate:

```
v_1 :=1; // početni čvor
nv := 1; // broj obiđenih čvorova
cena := 0; S:={2,...,n};
repeat while S ≠ ∅
   v_{nv+1} := naslednik(od  v_nv  u S);
  nv := nv + 1; S := S - { v_{nv+1} };
  cena := cena + cena(luk( v_nv, v_{nv+1} ));
end repeat;
if nv=n and and cena ≤ b then USPEH stop else NEUSPEH;
```

  Ali sam problem optimalnog Hamiltonovog ciklusa nije NP i zapravo je problem komplentaran ovom jer se traži da li postoji ciklus cene veće od zadate, što se može znati tek kada sve kopije mašina vrate USPEH ili postoji neka sa vrednošću NEUSPEH. Za problem komplementaran problemu P klase je jasno da je takođe P, ali za NP to nemora da važi.

Definicija 9.1 *Problem Q je svodljiv na problem R ako za rešenje s problema R postoji polinomijalno izračunljiva funkcija g td. je g(s) rešenje problema Q.*

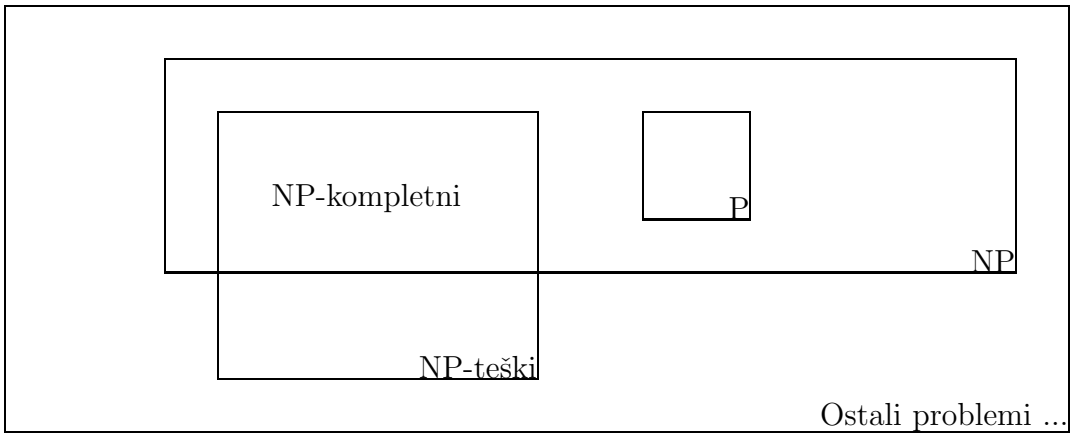
Piše se $Q \rightarrow R$. Znači da ako može da se reši R onda može i Q.

Definicija 9.2 *Ako je $Q \rightarrow R$ i $R \rightarrow Q$ onda su Q i R ekvivalnentni.*

Definicija 9.3 *Problem je NP-težak akko bilo koji NP problem može biti sveden na njega.*

NP-težak ne povlači da jeste NP.

Teorema 13 ***Fundamentalna teorema*** *(Cook)*
SAT problem (zadovoljivost logičkog izraza) je NP-težak.



Definicija 9.4 *Problem je NP-kompletan akko je NP-težak i ako $\in$ NP.*

Mnogi klasični problemi su NP-teški i NP-kompletni, štaviše, td. teško da je $NP = P$. Svi NP-kompletni problemi su ekvivalentni u tom smislu da ako jedan može da se reši polinomijalno onda mogu i ostali. Ali ne znamo da li su polinomijalni i da li postoji deterministično rešenje za neki od njih.

Svodljivost je tranzitivna, i uz dokaze gde se izodljivost logičkog izraza svodi neki problem (konjunktivna normalna forma, KNF) može se dobiti npr. ovakvo drvo svodljivosti nekih NP-kompletnih problema:

- SAT (← ukazuje na relaciju svodljivosti) ←
  - k-klika ←
    - postoji Hamiltonov ciklus ← Hamiltonov ciklus
    - pokrivanje ← ,,knapsack"←
      - binarno particioniranje
      - ...
  - 3-SAT (sa tačno 3 literala po klauzuli) ← bojenje ← particioniranje
  - izvodljivost sistema celobrojnih nejednačina

Na kraju, malo je problema za koje se može reći da su dobri, većina ostalih zato se i tiče oblasti VI. Sam problem prepoznavanja nekog od klasičnih (poznatih) dobrih problema i njihovo prevođenje u oblik koji je upotrebljiv za direktnu primenu poznatog algoritma je sam po sebi nepolinomijalan problem !

10 Rešavanje problema propagiranjem i nabrajanjem

Ovo poglavlje se bavi detaljnije metodama nabrajanja pomentim na početku prethodnog.

Definicija 10.1 *Kombinatorni problem je problem oblika: naći element $x \in X$ td. važi skup uslova $K(x)$ gde je X konačan i diskretan (postoji konačna separacija između svake dva para tačaka tj. skup je diskontinualan).*

Uopšteni postupak za rešavanje jeste:

1. izaberi prvu $x_0 \in X$ koja nije razmatrana
2. proveri $K(x_0)$
3. ako nije zadovoljen neki uslov, pređi na 1
4. x_0 je jedno rešenje, pređi na 1 ako su potrebna sva rešenja

Primer: Pronalaženje izlaska iz lavirinta (x_0 je trenutno polje, isprobavaju sve dostupne putanje redom, ako se iscrpu rešenja back-tracking na neko ranije gde nije).

10.1 Gradijent metode

Metode poznate i kao ,,hill-climbing" (gradijent je varijacija, gradus = korak), za svaki problem se nalazi funkcija čiji se ekstrem traži tako što se za svaku iteraciju približava ekstremu. Metoda ima svoja ograničenja, zahteva pre svega da slika oblasti u kojoj se rešava bude konveksna (inače ne garantuje rešenje, ne pamti prethodne korake i ne snalazi se sa ,,preprekama" tj. ne razlikuje globalne od lokalnih ekstrema, postoje modifikacije i primeri numeričkih algoritama). Jedan primer je simpleks algoritam i neke njegove varijante. *Prednosti ovog metoda:* smanjuje se broj čvorova koje treba obići, efikasnost je ista kao kod pretrage stabla u dubinu.

Nedostaci ovog metoda: lažni vrhovi u kojima će doći do povratka, a i rešenje ne mora biti pronađeno iako postoji; neophodna je heuristika pretrage.

Generalna strategija za prethodni problem u okviru ove metode je da se pronađe uz neke mofikacije globalni optimum na osnovu lokalnih optimuma te ovo može biti dobro čak i ako prostor pretrage nije povezan.

10.2 Linearno programiranje

Problemi oblika $\mathbf{A}\ \mathbf{x} \leq \mathbf{b}$, naći $Z = \max(\mathbf{c}\ \mathbf{x})$, $x \in \mathbb{R}^n$. Simpleks algoritam, [JL]:

1. Presek više nejednačina tog oblika daje poliedar (ograničen ili ne)

2. Svaka tačka poliedra je linearna kombinacija svojih temena (ekstremnih tačaka), koje su presek m hiperravni dobijenih iz m jednačina - temena onda ima $\binom{n}{m}$ u $\mathbb{R}^n$

3. Optimum nije unutar poliedra jer se može konstruisati tačka na površini linearnim kombinacijama (preformuliše se sistem dodatnim pomoćnim promenljivama za nejednačine po kojima se onda isto traži maksimum, fiksiraju se dostignuti maksimumi, gradijent je koeficijent u pivotiranoj matrici takvog sistema) počevši od trivijalnog rešenja za $\mathbf{x} = 0$ td. je Z uvećano. Ako problem nije degenerisan $rang(A) = m$ onda je moguće ivicom površine stići tako do najbližeg temena

4. Baza od $\mathbf{A}$ je podksup pljosni $[1, \cdots, m]$ kojima odgovara kvadratna matrica $\mathbf{B}$ ranga n dobijena od $\mathbf{A}$. Onda je optimum dostignut samo kao rešenje baze. $\mathbf{A}$ se particioniše u dve podmatrice $\mathbf{A}$ i $\mathbf{N}$ td. je $\mathbf{B} \cdot \mathbf{x}_B + \mathbf{N} \cdot \mathbf{x}_N = \mathbf{b}$, bazno rešenje definisano sa $\mathbf{x}_N = 0$ se nalazi rešavanjem (Kramer) $\mathbf{B} \cdot \mathbf{x}_B = \mathbf{b}$.

5. Algoritam pretražuje samo ostvarive baze $\mathbf{x}_B \geq 0$ šetajući od temena do temena (jedna promenljiva sa vrednošću 0 se zamenjuje drugom da bi se dobila vrednost različita od 0 ako je moguće), vođen gradijent metodom kao izborom. Broj temena je konačan pa algoritam konvergira.

10.3 Gradijent metoda u teoriji grafova

Ako je $G = (X, U)$ označen graf težinama (svakom luku dodeljena), naći parcijalno drvo $A = (X, V^*)$ minimalne ukupne težine (drvo nema ciklusa i za n temena ima $n - 1$ luk). Kruskal je predložio metodu gradijenta (ovaj algoritam se zove još i ,,pohlepni"):

```
uredi lukove po težini;
V:= ∅;
while ima lukova koji nisu razmotreni
```

```
  if prvi od ovih ne formira ciklus sa V then dodaj ga u V;
end;
```

10.4 Heurističko pretraživanje

Jedna uopštena klasa problema se može rešavati na sledeći način:

1. Izaberi neku akciju iz skupa mogućih
2. Izvrši je promenivši trenutno stanje
3. Evaluiraj novo stanje
4. Odbaci neželjena stanja
5. Ako je dostignut cilj - kraj, inače ponovo 1.

Simpleks algoritam je primer heurstičke gradijent metode. U koracima 1, 3 i 5 se koristi obično funkcija kojom se evaluira stanje. Tako se može prethodni Kruskalov algoritam modifikovati i poboljšati: uvek se bira luk najmanje težine među raspoloživim, nema potrebe sortirati lukove, i ako se naiđe na ciklus u V nakon dodavanja se obriše luk najveće težine u V.
Kod ovakvih algoritama se obično vode dve liste čvorova - jedna za već obiđene čvorove (zatvorene) i one koje su raspoloživi za pretragu (otvorene, npr. početni).

10.5 A^* algoritam

Poboljšanje gradijent metode u smislu navedenih nedostataka (Hart, Nilsson, Raphael, 1968-1972). Ako je S_0 početno stanje i S situacija u nizu kao posledica akcija algoritma, cilj je dostići određeno stanje. Funkcija $f(S)$ ima numeričku vrednost u ovom algoritmu td. $f(S) = g(S) + h(S)$ gde je $g(S)$ poznata cena u postizanju stanja S ($c(S_1, S_2, a)$ je cena od stanja S_1 do S_2 akcijom a), a $h(S)$ heuristička funkcija koja procenjuje cenu dostizanja rešenja iz S najboljim (optimalno) mogućim redosledom akcija. Ako je $h^*(S)$ minimum cene za proizvoljan put od situacije S do rešenja, onda je $h(S)$ donja granica: $0 < h(S) \leq h^*(S)$ (obično se koristi najbolja procena h jer se retko može izračunati baš h^*). Ako je $g^*(S)$ najmanja cena dostizanja situacije S od početne S_0 onda je $f^*(S) = g^*(S) + h^*(S)$ rešenje najmanje

cene koje prolazi kroz S, a $f(S)$ je njegova procena i donja granica. A^* algoritam:

Korak 0: konstruiši graf $R = (E, A)$ (lukovi su akcije),
$f(S_0) \leftarrow O$,
$O \leftarrow S_0$ (particija otvorena za pretraživanje),
$C \leftarrow \emptyset$ (particija zatvorena za pretraživanje)
dok je $O \neq \emptyset$ **ponavljaj**
Korak 1: izaberi $S \in O$ td. je $f(S)$ minimum
$O \leftarrow O - S$, $C \leftarrow C + S$
ako je S rešenje stani, **inače**:
Korak 2: razvij S konstruišući stanja S_i, $0 \leq i \leq n_S$ redom iz akcija $\in S$
Korak 3: **za svako ovo** i računaj $f(S_i)_{novo} = g(S_i)+h(S_i)$ td. $h(S_i) \leq h^*(S_i)$
Korak 4: u O stavi sve S_i koji nisu ni u O ni u C, a za one koji jesu:
$f(S_i)_{novo} \leftarrow min[f(S_i)_{staro}, f(S_i)_{novo}]$
ako je S_i u C **onda** $C \leftarrow C - S_i$, $O \leftarrow O + S_i$
opet

Može se pokazati da algoritam uvek nalazi rešenje ako postoji konačan niz akcija koji vodi od početnog stanja ka rešenju:

Teorema 14 *Algoritam A^* se završava ako postoji konačan niz akcija od početne situacije S_0 do rešenja.*

Trivijalna posledica: ako je skup akcija konačan kao i broj mogućih situacija, onda A^* na osnovu ovoga konvergira. Dokaz (reductio ad absurdum): graf nije konačan, A^* ne konvergira što je moguće samo akko O raste beskonačno, što znači da je vrednost $f(S)$ neograničena. Ako je $v^*(S)$ broj akcija od S_0 do S (ne mora uopšte da se poklapa sa A^* putanjom), i onda na osnovu definicije g^* važi: $g(S) \geq g^*(S) > kv^*(S)$, gde je k neka konstanta. Pošto je $h(S)$ nenegativna, važi: $g(S) \geq G(S) > kv^*(S)$, pa ako je v^* neograničena onda je i $f(S)$ (A^* ne konvergira). Može se ipak pokazati da tokom izvršavanja A^* uvek postoji situacija S_2 takva da je $f(S_2) \leq f^*(S_0)$: ako postoji konačan niz akcija od S_0 do cilja, neka je S_2 (mora postojati ako A^* nije konvergirao) prva situacija td. je $S_2 \in O$. Pošto je $f(S_2) = g(S_2)+h(S_2)$ i po definiciji situacije pre S_2 su u C, važi $g(S_2) = g^*(S_2)$. Po konstrukciji A^* važi $h(S_2) \leq h^*(S_2)$, i odatle konačno: $f(S_2) \leq g^*(S_2) + h^*(S_2) = f^*(S_2)$. Ako je niz akcija optimalan, onda je $f(S_2) \leq f^*(S_0)$, što protivreči pretpostavci da je f

neograničena.

Najveća mana algoritma je njegovo ponašanje kada nema rešenja. Pretraže se sva stanja da bi se zaključilo da nema rešenja - primer: slagalica 3×3, $g(S)$ = broj koraka od S_0 do S, $h(S)$ = broj elemenata koji nisu na pravom mestu. Varijanta algoritma kada su težine sortirane je algoritam za pronalaženje najkraće putanje kroz graf (Moore-Dijkstra, 1957), kao i uopštenje (koje je predložio Pohl, 1970): $f(S) = (1-\alpha)g(S) + \alpha h(S)$, td. je onda $\alpha = \frac{1}{2}$ zapravo A^*-algoritam, $\alpha = 1$ gradijent metoda ($\alpha = 0$,,British Museum"), mada se može vrednost α menjati i dinamički.

10.6 Implicitno nabrajanje propagiranjem uslova

Ovo se može objasniti kroz primer problema rasporeda dama na šahovskom polju $N \times N$ td. nijedna figura ne napada drugu (i da je u u svakom redu po jedna). Ako se pretpostavi da je $X = \prod_{i=1}^{N} Y_i$ gde je $Y_i = \{1, \cdots, N\}$ nekakav skup izbora, onda se ovaj problem može prikazati kao traženje $x \in X$ td. je ispunjeno $K(x)$ i uopštiti ako se posmatra svaki element iz x kao neki izbor datih mogućnosti. Rešavanje ovog problema pretraživanjem redom po drvetu mogu'ćnosti izbora iz reda u red (u dubinu) je onda algoritam implicitnog nabrajanja (uopštenog kombinatornog karaktera). Postoje dva poboljšanja ovakvog algoritma za problem dama koja se svode na poboljšavanje provere uslova i izbora obilaska drveta pretrage. Pomenutim metodom nabrajanja se mogu pamtiti izbori u nekom vektoru brojača kojima se pravi backtracking i proverava uslov $K(x)$ ali se ne postavlja zauzetost polja na osnovu izbora za naredne izbore; zato, prvo poboljšanje je da se umesto vektora brojača koristi vektor vektora odnosno matrica ,,zauzetosti" polja (elementi su indeksi izbora ili 0 ako je polje ostalo slobodno), i treće, da se na kraju pravi izbor redosleda redova (najpre oni koji imaju manje slobodnih polja).

Primer - verzija 1 (posmatramo tablu 4x4):

Neka y_k označava koju smo poziciju izabrali za damu, a L_k je vektor u koji zapisujemo zabranjene pozicije. Na početku su sve pozicije u vektoru dozvoljene (slobodne) pa je $L_1 = L_2 = L_3 = L_4 = (0, 0, 0, 0)$ Ako izaberemo y_1=1:

$L_1 = (1, 1, 1, 1)$
$L_2 = (1, 1, 0, 1)$
$L_3 = (1, 0, 1, 1)$
$L_4 = (1, 0, 0, 1)$

Znači ako smo izabrali da je dama na prvoj poziciji, onda su sva polja u prvoj vrsti, prvoj koloni i po dijagonali zabranjena. Sledeće polje koje biramo neka bude na poziciji druge vrste i treće kolone, te pošto je to drugi izbor po redu obeležimo ga sa 2. Zabranjena su sl. polja

$L_2 = (1, 1, 2, 2)$
$L_3 = (1, 2, 1, 1)$
$L_4 = (1, 0, 2, 1)$

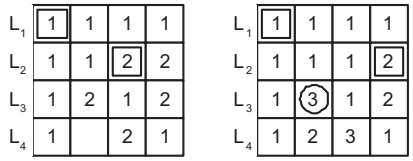
Vidimo da nam u trećoj vrsti ne ostaje nijedno dozvoljeno polje pa vršimo bektrek.
Izaberimo u drugoj vrsti četvro polje

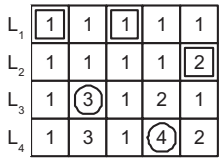
$L_2 = (1, 1, 2, 2)$
$L_3 = (1, 0, 1, 1)$
$L_4 = (1, 0, 0, 1)$

Ovo nam omogućava izbor u trećoj i četvrtoj vrsti.

Problem osam kraljica - verzija 2:

Osnovna razlika izmedju prve i druge verzije je da je shema izvođenja testiranja promenjena tako što se posle svakog učinjenog izbora uzima u obzir ono što taj izbor povlači. Sva polja koja su slobodna postaju zabranjena i zapisana u vektor slobodnih zajedno sa brojem izbora (biramo slobodno polje koje nam omogućava više nezabranjenih polja u sledećem koraku).





Primer je i dinamičko programiranje (opštije od linearnog programiranja - ako se funkcija dekomponuje na kompoziciju funkcija od kojih je prva monotono nerastuća onda se problem svodi na potproblem optimizacije druge funkcije, primeri nad diskretnim skupovima):

10.7 Dinamičko programiranje

Dinamičko programiranje je metod za rešavanje problema optimizacije. Osnovna ideja je traženje optimalnog rešenja problema upotrebom optimalnih rešenja potproblema. Rešavanje se obično odvija u koracima ili fazama i svakom takvom koraku postoji određen broj mogućnih izbora: ako dva različita niza odluka daju rešenje, samo će bolje od njh biti čuvano. Obično se zadaje funkcija čija se ekstremna vrednost traži pod nekim dodatnim uslovima - npr. ako je data tabela cene proizvodnje 0-4 artikla (kolone) u toku tri meseca za svaki mesec posebno (vrste), cilj je proizvesti (bar) 4 artikla za 3 meseca po što manjoj ceni. Za svaki broj artikala u trećem mesecu se nalazi optimalan (minimalan) zbir preostalog broja artikala u prva dva meseca ukupno (potproblem), i na kraju ukupno optimalno rešenje tj. zbir (minumum). Optimalno rešenje zavisi od izbora potproblema kao zbira ali ne i od pojedinih njegovih vrednosti. Linearno programiranje je takođe primer, a i mnogi drugi problemi se mogu rešavati ovom metodom (npr. knapsack, brojni primeri iz operacionih istraživanja, ekonomije, itd.).

Definicija 10.2 *Funkcija f tri realne promenljive u, v, w može se dekomponovati ako*

1. postoje funkcije g, h tako da f može biti napisana kao kompozicija:

$$\forall(u,v,w) f(u,v,w) = g(u,h(u,w))$$

2. funkcija g je monotono neopadajuća

Za dinamičko programiranje značajna je sledeća teorema (koja se može uopštiti na proizvoljan broj promenljivih i potfunkcija):

Teorema 15 *Za svaku realnu funkciju realnih promenljivih u, v, w takvu da ju je moguće dekomponovati* $f(u, v, w) = g(u, h(v, w))$ *važi:*

$$opt_{u,v,w}[f(u, v, w)] = opt_u[g(u, opt_{v,w}[h(v, w)])]$$

Značaj dinamičkog programiranja ogleda se u sledećem:

Svaka podstrategija optimalne strategije mora i sama biti optimalna.

Obično se formiraju velike (memo) tabele sa svim prethodnim rezultatima koji bi mogli biti od značaja u narednim iteracijama. Problem je organizovati izračunavanje tabele na najefikasniji način i često se javlja problem kombinatorne eksplozije njenih vrednosti. Dinamičko programiranje je efikasno kada se može svesti na nekoliko manjih, ali ne sasvim malih nezavisnih potproblema. Problemi bojenja grafa i trgovačkog putnika (traveling salesman) se mogu rešiti ovom metodom, a primer je i Floyd-Warshall-ov algoritam najkraće putanje uređenog grafa (zasnovan na rekurzivnoj relaciji

$$p(i, j, k) = \min(p(i, j, k-1), p(i, k, k-1) + p(k, j, k-1))$$

gde su $p(i, j, 0)$ zadate težine lukova) reda $O(n^3)$.

Na primer, bojenje čvorova grafa: potrebno je najpre odrediti najmanji broj različitih boja potrebnih za bojanje mape tako da države koje se graniče ne budu obojene istom bojom (hromatski broj grafa je minimalan broj boja potreban da bi se graf obojio - specijalno, za mape na površi postoji kombinatoran dokaz da je taj broj 4, Appel i Haken 1976.), ili npr. raspored ljudi na sastanku uz pravila ko sme pored koga da sedi.

Označimo sa $\gamma(G)$ broj boja, sa S skup čvorova grafa, U skup grana (i,j), i sa [1,q] skup prvih q brojeva koji predstavljaju boje. Zadatak glasi:

Pronaći funkciju boje: $S \longrightarrow [1, q]$ *tako da za sve* $(i, j) \in U$, $boja(i) \neq boja(j)$ *i q je minimalno.*

Možemo postupiti kao i kod problema osam dama: obojimo prvi čvor i on dalje nameće pravila igre dok ne obojimo celu mapu. Međutim, postoji razlika u načinu optimizacije - u ovom slučaju se zaustavljamo pre što nego iscrpimo sve mogućnosti, tj. čim smo u mogućnosti da pokažemo da ne postoji bolje rešenje.

Polazimo od čvora sa najvećim stepenom (koji ima najviše neobojenih susednih čvorova, efikasnosti radi) i bojimo njemu susedne čvorove (analogno opet problemu rasporeda dama na tabli) uz ograničenja i backtracking - rezultat je ili obojen graf ili činjenica da se nemože obojiti zadatim brojem boja (,,pohlepni" algoritam). Drugi korak je *traženje optimalnog rešenja*. Gornje ograničenje može biti (u najgorem) broj čvorova. Iterativno se smanjuje broj boja dok se ne utvrdi minimalan broj, ali efikasnije je poći od donje granice i uvećavati broj boja. Minimalan broj boja potreban za bojenje date mape je donekle dualan problemu kardinalnosti najveće klike (klika je potpuno povezan podgraf, traženje maksimalne klike može biti sam po sebi zahtevan problem) grafa koji predstavlja mapu koji daje praktično donje ograničenje potrebnog broja boja. Čak i klika koja nije maksimalna ali dovoljno velika nudi već neko ograničenje.

Na ovo se oslanja naredni primer problema putujućeg trgovca (najčuveniji problem operacionih istraživanja): trgovački putnik mora da obiđe n različitih gradova. Znajući rastojanja između gradova treba da pronađemo put kojim trgovački putnik treba da putuje kako bi ponudio svoje proizvode prolazeći kroz svaki grad samo jednom i praveći tako minimalan put.

Grafovski, problem glasi: za zadati težinski graf G (svakom luku je dodeljen broj) ustanoviti da li u G postoji Hamiltonov put manji od zadate cene a zatim i najmanje moguće cene (problem je NP-kompletan, redukcija na pokrivač grana). Poseban (lakši) slučaj su Euklidski grafovi (rastojanja poštuju nejednakost trougla, Hamiltonov ciklus kome se izbaci čvor onda nemože biti duži), ali u opštem slučaju rastojanja mogu biti čak i negativna i asimetrična. Problem se rešava najpre traženjem proizvoljnog rešenja kao gornje granice (npr. Christofides, 1975), a zatim se traži optimalno rešenje - načelan postupak sličan prethodnom. Grubom silom (kombinatorno) problem je reda $O(n!)$, dok se uz upotrebnu pomenutih metoda optimizacije dostiže $O(2^n)$.

10.8 GPS - General Problem Solver

GPS - uopšteni rešavač problema - ime je čuvenog programa kojeg su razvili Newel, Shaw i Simon 1950. a unapredili su ga Ernst i Newel 1957. Zamišljen je kao uopšteni sistem VI za rešavanje problema, od simboličke integracije, slagalica do PR1 dokazivanja i gramatičke analize, i to jednim uopštenim postupkom u kome se inicijalni objekti i ciljevi, operatori koji omogućavaju transformacije nad objektom, i traženje razlike i operatora koji je smanjuje između dva data objekta. Svaki problem stavljen u GPS ima oblik : inicijalni objekat, ciljni objekat, skup operatora. GPS je u boljoj poziciji od ljudskog rešavaoca zato što postoji lista operacija upotrebljivih za taj problem, dok se čovek svega mora setiti ili čak izmisliti u toku rešavanja. GPS ima tri cilja:

1. A(O,x) primeniti operator O na objekat x

2. F(O,Δ,x,y) pronaći operator O koji će redukovati neke razlike iz Δ izmedju x i y

3. T(x,y) transformacija objekta x u y

Svi problemi imaju na početku cilj: T(objekat(početni), objekat(završni))

Polazeći od početnog objekta, koristeći opisane metode, GPS traži jedan ili više podciljeva dok ili ciljni objekat ne bude dostignut ili dok ne dođemo do objekta koji je napušten a koji još može biti razvijen. Primer za ovakovo rešavanje je dao JohnMcCourty 1963. - ,,majmun i banane". Na podu sobe su majmun i kutija, a grana banana je zakačena za plafon. Majmun može doći do banana samo ako kutija stoji ispod banana i ako se majmun na nju popne. Problem je trivijalan za čoveka što ne važi za opisivanje ponašanja GPS-a u rešavanju zahteva. Rešenje zahteva niz koraka. Vrednost ovog primera je da pokaže u potpunosti detalje kako svakodnevno rezonovanje može zahtevati niz elementarnih akcija. Cilj se dostiže mehanizmom koji je ekstremno jednostavan i lak za implementaciju. Pretraga je vođena pomoću koncepta razlike između objekata prostora pretrage, pri čemu potreba redukovanja neželjenih razlika generiše novi cilj na prirodan način.

U GPS pokušajima izbori se prave pomoću tabele konekcija za redukovanje razlika izmedju objekata.
Primer operacija za ovaj problem:

```
O1    veranje :uslov    loc(M)=loc(C)
               akcija    loc(M)=on(C)
O2    setnja  :uslov    x je lokacija
               akcija    loc(M)=x
O3    pokret:  uslov    x je lokacija
                        loc(M)!=on(C)
                        loc(M)=loc(C)
               akcija    loc(M)=x
                        loc(C)=x
O4             uslov     loc(C)=ispod(B)
                        loc(M)=na(C)
               akcija    sadrzaj(rukeM)=B
```

Zaključak: GPS pokazuje da je moguće imati jedan uniforman sistem za rešavanje različitih, netrivijalnih problema, zasnovan na strategiji identifikovanja razlika izmedju objekata i redukovanju tih razlika. Ovu ideju je dalje razradjivao Slagle 1963. i Quirilan 1969.

Međutim, sistem nije upotrebljiv za iole složenije probleme i često je dovoljno teško formulisati problem i predstaviti ga u GPS-u.

Postoje i druga okruženja za uopšteno rešanje problema, npr. ALICE (A Language for Intelligent Combinatorial Exploration).

11 Programi - igre, psihologija rešavanja problema

11.1 Drvo pretraživanja (drvo ispravnih poteza)

U igri u kojoj učestvuje više igrača svaki učesnik se trudi da predvidi moguće poteze ostalih do izvesne dubine i da izabere najpovoljniji potez koji bi ga doveo do pobede. Dobar primer je šah. Ako se kao skupovi izbora u algoritmu nabrajanja izaberu mogući potezi u igri, veoma lako dolazi do kombinatorne eksplozije (otvaranje počine sa izborom od oko 20 mogućih poteza, što do središnjice narasta do reda 40-50 mogućih poteza - prosečna partija traje oko 40 poteza a već je 40^{12} za pretraživanje dubine 12 poteza reda veličine broja sekundi starosti sunčevog sistema). Veliki izazov zato predstavlja igrati, ali i napraviti poboljšanja algoritma kojima se ublažava eksponencijalnu eksploziju algoritama koji igraju ovakve i neke druge igre u realnom vremenu. Osnovni metod je ograničavanje drveta pretraživanja ili po širini ili po dubini. Po širini se mogu odbacivati grane npr. ako se pouzdano oceni da ne donose bitnu prednost ili su potpuno nebitne. Po dubini se može skratiti pretraga ako se zna da preko zadate dubine nije u realnom vremenu. Potrebno je voditi računa o tome da neki put presecanje ne pokaže veoma loš ili dobar ishod u sledećem koraku pa se proverava da li je pretraga donela ,,stabilne" rezultate. Uopšte, algoritmi pretraživanja koji prate stablo pretrage do zadate dubine (depth-first i varijanta koja prvo bira po nekom kriterijumu najbolju granu: best-first, uz to da se može povećavati iterativno dubina i drvo pretrage) i oni koji ga pretražuju nivo po nivo (breadth-first) nisu ipak dovoljno efikasni za mnoge probleme, ali nude mogućnost postupnog pretraživanja i poboljšavanja pretrage ,,dokle god ima vremena". Stanje u drvetu pretrage može biti pozicija u igri, ali može npr. biti i čvor I - ILI drveta.

11.2 Evaluacija pozicije

Osnovni način da se utvrdi koliko je stanje u drvetu pretrage dobro za igrača je ocena funkcijom evaluacije pozicije $F(S)$ (gde je S stanje npr. na šahovskoj tabli). Ako je njena vrednost negativno onda je protivnik u prednosti (i obratno), i što je veća vrednost to je prednost izraženija. To je, naravno, vrednost koja ne može biti apsoltno tačna već je posledica praktičnih vrednosti i osobina igre, pozicija i vrsta figura kao i drugih bitnih elemenata (parametri koji je određuju osim argumenta se takođe mogu dinamički

menjati). Primer:

$$F(s) = aB + bR + cM + dC + eP + fA = \sum_i w_i f_i(s)$$

gde su R, M, C, P, A mere koje se odnose na karakteristike u opadajućem redosledu važenja:

- R - relativna sigurnost kralja
- M - mobilnost figure; broj polja koje svaka figura mozže napustiti
- C - kontrola centra
- P - struktura pijuna - pozitivan doprinos dolazi od zaštićenih pijuna, pijuna koji napreduju i isto iz neprijateljskih, izolovanih i nezaštićenih pijuna
- A - mera relativne mogućnosti napada

Sve ovo zahteva veliko strpljenje i istrajnost u određivanju odgovarajućih koeficijenata koji se čak i u toku igre mogu promeniti. Ovaj oblik funkcije može dati i žrtvovane poteze - to su potezi za koje će vrednost za B biti negativna, iako će težina cele sume biti pozitivna.

11.3 MINIMAX izbor i algoritam, alfa-beta algoritam

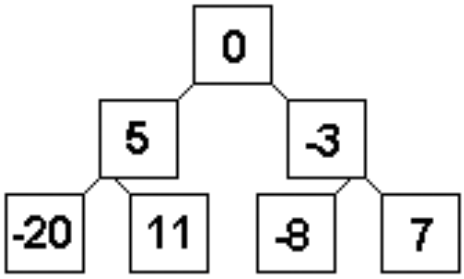
Ako se tako pretpostavi da oba igrača koriste istu funkciju evaluacije (Neumann, Morgenstern) onda je jednom bitno da maksimizuje njenu vrednost a drugom da je minimuzuje u toku pretrage. Ako je na prvom i ostalim koracima (nivoima drveta) pretrage cilj naći maksimalnu vrednost, onda se za pretragu zadate dubine traži maksimalna ili minimalna vrednost funkcije listova koja se propagira na pozicije iznad sve do polazne pozicije gde se konačno bira najveća (uz pretpostavku da protivnik isto radi pa se zato u neparnim koracim traži minimalna vrednost).

Ovo je MINIMAX algoritam (primer - šah):

```
MINIMAX(tabla)
```

```
dubina ← 1; strana ← 1;
E(1) ← potezi(tabla,strana); eval(1) ← −∞;
repeat while E ≥ 1
   repeat while E(dubina) ≠ ∅
      potez(dubina) ← sa_liste(E(dubina));
      tabla ← uradi(tabla,potez(dubina));
      if dubina ≠ maksimalna onda
         dubina ← dubina+1;
         strana ← -strana;
         E(dubina) ← potezi(tabla,strana);
      else
         eval(dubina) ← MAX[eval(dubina)*strana,valuacija(tabla)];
         tabla ← vrati(tabla,potez(dubina-1);
      end if;
   end repeat; [sledi backtracking]
   if dubina = 1 then MINIMAX ← eval(1) end if;
   strana ← -strana;
   dubina ← dubina-1;
   potez(dubina) ← sa_liste(E(dubina));
   tabla ← vrati(tabla,potez(dubina));
   eval(dubina) ← MAX(eval(dubina),-eval(dubina+1));
end repeat;
end MINIMAX;
```

Ukratko: `potezi` generiše listu poteza koje igrač može (na tom nivou) da napravi, `sa_liste` skida vrednost sa liste kao steka, `uradi` menja tablu prema potezu a `vrati` suprotno.



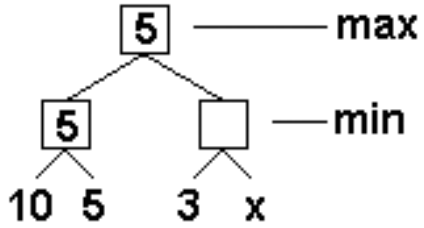
$\max(\min(-20, 11), \min(-8, 7)) = \max(-20, 8) = 8$

Mnogo je čitljiviji (i poznatiji) rekurzivni oblik ovog algoritma, gde se pretpostavlja da prvi igrač vuče `MAX` poteze a drugi `MIN` poteze (što je određivala promenljiva `strana` u prethodnom algoritmu), i `u` je čvor pretrage za koji se traži MINIMAX ocena pozicije (trenutno stanje):

```
minimax(u)
   if jeste_list(u) then
        valuacija(u);
   else
        if jeste_max_potez(u) then
            za_naredne_poteze(v1,...,vn);
            return max(minimax(v1),...,minimax(vn));
        else
            za_naredne_poteze(v1,...,vn);
            return min(minimax(v1),...,minimax(vn));
        end if;
   end if;
end minimax;
```

11.4 α-β kresanje (odsecanje)

Postoji poboljšanje ovog algoritma tzv. **alfa-beta algoritam** (McCarthy, 1961). Pod određenim uslovima se mogu ,,kresati" (prune) čitave grane drveta pretrage. Ako je s buduća minimax vrednost početne pozicije, α vrednost podgrane ispod nje (dobijena punom pretragom do zadate dubine), y buduća vrednost naredne podgrane koja ima svoju podgranu s vrednošću z (takođe dobijene punom pretragom) tako da je $z \leq \alpha$ onda se može odustati od dalje pretrage ,,ispod" y jer je (po pretpostavci i minimax osobinama) $y \leq z \leq \alpha \leq s$ - alfa kresanje. Isto tako se i za protivničke poteze radi beta kresanje. Takođe, ako se na većoj dubini iste parnosti naiđe na izračunatu poziciju $z \leq \alpha$ može se analogno prema njoj primeniti alfa kresanje.



$$\max(\min(10,5),\min(3,x)) = \max(5,x) = 5, \quad x \geq 3 \veebar x < 3$$

Primer rekurzivnog alfa-beta algoritma uz čuvanje dubinskog kresanja:

```
alphabeta(u)
   if jeste_max_potez(u) then
      return evalmax(u,-infty,+infty);
   else
      return evalmin(u,-infty,+infty);
   end if;
end alphabeta;

evalmax(u,alpha,beta)
   z := alpha;
   if jeste_list(u) then
      return valuacija(u);
   else
      loop naredni_potez(v) do
         val := evalmin(v,z,beta);
         z := max(z,val);
         if z >= beta then exit loop;  // beta-kresanje
      end loop;
   end if;
   return z;
end evalmax;

evalmin(u,alpha,beta)
   z := beta;
   if jeste_list(u) then
      return valuacija(u);
   else
      loop naredni_potez(v) do
         val := evalmax(v,alfa,z);
         z := min(z,val);
         if z <= alpha then exit loop;  // alfa-kresanje
      end loop;
   end if;
   return z;
end evalmin;
```

Osnovni nedostaci ovih algoritama su: nedostatak sagledavanja opšte strategije igranja (modifikacije sa otvaranjem i završnicama ne rešavaju problem suštinski), kao i problem ,,horizonta" (program ne vidi pozicije van dubine pretraživanja, i uvek se ,,trudi" da izbegne neugodne pozicije koje su u dometu pretraživanja).

11.5 Psihološka izučavanja rešavanja problema i igranja

Računari danas raspolažu velikom količinom memorije i brzo procesiraju podatke i mogu efikasnije to da rade od mozga u domenu nekih specifičnih primena ali i dalje je to daleko od ,,hardvera" sa nekih 100 milijardi neurona (od koji svaki nosi sigurno više od jednog bita). Mozak koristi kratkotrajnu i dugotrajnu memoriju - pokazano je da ta kratkotrajna memorija može da sadrži 7 ± 2 osnovna objekta pamćenja (,,operandi" instrukcije). Međutim sama obrada velike količine podataka je daleko složenija od prostog izvršavanja programa ili algoritma. Psiholozi (Alfred Binet, 1894 - Otto Seiz, 1920) su davno uspostavili model ljudskog razmišljanja i rešavanja problema kao linearnog niza osnovnih operacija (kao deo psihičkih procesa koji se odvijaju u čoveku). Tako je odavno poznato da procesi razmišljanja i zaključivanja zavise od memorije, ali i percepcije. Kratkotrajna memorija je upravo vezana direktno (brže, mnogi se procesi odvijaju paralelno, ,,nesvesno") za čula - spoljni svet, dok komunikacija kratkotrajne i trajnije memorije traži dodatno vreme. Igranje šaha i igranje uopšte su poslužili kao dobar model za ovakva proučavanja, a VI danas nudi sredstva kojima se mogu ovakvi modeli proučavati.

11.6 Teorija igara

Teorija igara se definiše kao disciplina koja se bavi pronalaženjem optimalnog rešenja u uslovima konflikta (konflikt je skup uslova kojim se ograničava optimalno rešenje problema - u smislu teorije igara to su različiti interesi igrača). Formalno, *igra* je struktura:

$$\Gamma = (K_d, \{S_K\}_{K \in K_d}, S, K_i, \{\prec_K\}_{K \in K_i})$$

gde su komponente:

- K_d skup koalicija dejstva: ako je skup igrača $\mathcal{K}$, onda $K_d \subseteq P(\mathcal{K})$ čine skupovi igrača koji mogu uticati na rešenje konflikta

- S_K je *skup strategija koalicije* K, gde svaka koalicija dejstva traži neka rešenja putem izbora nekog elementa iz skupa svih rešenja koja su joj dostupna - kada svaka koalicija izabere strategiju, implicitno je definisan ishod konflikta
- $S \subseteq \prod_{K \in K_d} S_K$ je *skup situacija*: sve mogući tokovi konflikta, definisani kada svaka koalicija izabere svoju strategiju, zovu se situacije
- K_i *skup koalicija interesa*: $K_i \subseteq P(\mathcal{K})$ čine skupovi igrača koji su zainteresovani za ishod rešenja
- $\prec_K$ je binarna *relacija prednosti* nad situacijama kojom svaka koalicija interesa daje neku prednost jedne situacije u odnosu na druge. Ovo se često realizuje *funkcijom dobitka* H_K koja se definiše nad skupom situacija u neki uređeni skup (brojeva)

Podrazumeva se da su ovi skupovi konačne kardinalnosti. Igre se u odnosu na ovu strukturu dalje mogu klasifikovati - interesantne su igre $|K_i| > 1$ (za $|K_i| = 1$ se svodi na klasičan ekstremalni problem), $|K_d| > 1$ su *strateške* igre, $K_i = K_d$ su *beskoalicione*, gde se onda posmatra jednostavnija struktura $\Gamma = (I, \{S_i\}_{i \in I}, S, \{H_i\}_{i \in I})$ gde je I skup igrača, antagonistička igra ima $|I| = 2$ i $(\forall s \in S) H_1(s) = -H_2(s)$. Dalje, pozicione igre (kakav je i šah), podrazumevaju:

- postoji jedno unapred definisano stanje igre - početna pozicija
- od početne pozicije, potezi se igraju naizmenično, a među njima moe biti i slučajnih
- moguće je da igračima nisu poznate sve informacije u vezi partije
- funkcija dobitka definiše isplatu na kraju igre za svakog od igrača

Drvo igre je konačan graf pozicija igre koji ne sadrži cikle, koren je početna pozicija. Kada kažemo da neki igrač igra po nekoj strategiji, podrazumevamo da taj igrač u svakoj poziciji u kojoj on odlučuje o nastavku igre ima definisan sledeći potez. U praksi, svaki igrač zaista razmišlja pomalo unapred. To razmišljanje ide u pravcu "ako se desi ova pozicija, igraću ovako". Bez umanjenja opštosti, možemo pretpostaviti da je svaki od igrača već pre početka partije izabrao strategiju. To je u neku ruku i tačno, jer ako pratimo pozicije unazad od kraja partije, dobijamo niz pozicija koji ako se raščlani

na podnizove poteza pojedinih igrača i čita unazad, upravo definiše jednu strategiju.

Navešću još nekoliko osnovnih osobina i klasa igara. Pozicione igre podrazumevaju sledeće dodatne komponente:

- konačno drvo igre Γ sa korenom A
- razbijanje skupa svih nezavršnih pozicija na $n+1$ skup: skup S_0 pozicija iz kojih se igra slučajan potez, i skupovi pozicija iz kojih igra svaki igrač $(i = \overline{1,n})$ $S_1, ..., S_n$
- za svaku od pozicija iz S_0, raspodela verovatnoće na skupu njenih alternativa
- razbijanje svakog od skupova S_i na informacione skupove S_i^j takve, da ako pozicije X i Y pripadaju istom informacionom skupu, tada:
  - broj alternativa pozicije X jednak je broju alternativa pozicije Y
  - ako je $X \neq Y$, tada X ne može slediti Y i Y ne može slediti X
- funkcija dobitka koja svakoj završnoj poziciji dodeljuje jedan n-dimenzionalni vektor

Čista strategija ne uzima u obzir potez protivničkog igrača (što praktično ima jedino smisla u igrama gde protivnik ,,skriva poteze" - u *igrama sa nepotpunom informacijom*), optimalna je ona čija je funkcija dobitka veća od svake druge strategije - skup čistih strategija $\mathcal{S}$ svih n igrača je *ravnotežna situacija* (rešenje igre) ako je svaka strategija iz tog skupa optimalna za igrača i $(i = \overline{1,n})$, skup dobitaka definisanih ravnotežnom situacijom zove se ravnotežna tačka. Skup dostižnih pozicija bilo koje partije neke igre čini prostor stanja igre. Nojman-Morgenšternova teorema tvrdi da svaka igra konačnog broja igrača sa potpunom informacijom ($|S_i^j| = 1$) ima ravnotežnu situaciju. Igra je sa potpunom memorijom za nekog od igrača ako on tokom igre ne zaboravlja poteze koje je do tog trenutka igrao, kao i poteze svih svojih protivnika. Nedeterministička igra uvek podrazumeva neke faktore koji ne zavise samo od kvaliteta igrača i njegove strategije jer je skup slučajnih poteza S_0 neprazan (iskustvo igrača postaje manje bitno). Potez M koji poziciju A transformie u poziciju B je *konverzija* ako nijedna pozicija iz drveta igre koje opisuje igru od početka do pozicije A ne može da se pojavi u drvetu

igre koje opisuje nastavak igre iz pozicije B do kraja (potez u šahu lovcem je konverzija, kao i uzimanje figure) - ovo je važno za osobinu *konvergentnosti* igre. Pozicije u koje je moguće doći na više od jednog načina ,,kretanja"kroz drvo igre, zovu se *transpozicije*.

Programi koji igraju deterministične antagonistične igre sa potpunom informacijom i memorijom, u Šenonovom smislu se klasifikuju u tri tipa:

- tip A: bira uvek najbolji potez, što podrazumeva potpunu pretragu drveta igre (čija složenost najčešće nije zanemariva)
- tip B: koristi heuristike i pretragu pozicija koje najviše obećavaju - poseduje *statičko znanje*
- tip C: koristi iskustvo da bi učio tokom igranja i poboljšavao performanse - poseduje *dinamičko znanje*

Za njih je značajan (kanonski) globalan plan igre:

- sve dok se pozicija prepoznaje u bazi otvaranja:
  igraj poteze iz baze otvaranja
- sve dok se na tabli ne prepozna pozicija iz baze završnica:
  igraj poteze pretraivanjem i evaluacijom pozicija
- sve dok igra nije gotova:
  igraj poteze iz baze završnica

Za antagonističke igre sa potpunom informacijom, postoje tri definicije rešene igre:

- veoma slabo rešena igra - određena je teoretska vrednost početne pozicije (u šahu, recimo, to ne znači praktično puno)
- slabo rešena igra - pronađena je optimalna strategija iz početne pozicije
- jako rešena igra - pronađena je optimalna strategija iz svih dopuštenih pozicija (u nekom razumnom vremenu, naravno)

Oblast teorije igara u okvirima VI je dostigla mnoge primene (ne samo u računarskim igrama), a više o svemu tome u [MD].

12 Ekspertni sistemi

Vremenom i razvojem računarska obrada podataka se sve više okreće ka simboličkim podacima a sve manje ka numeričkim. Klasa programa koji se bave pomaganjem u donošenju odluka u dobro definisanim oblastima znanja su ekspertni sistemi.

12.1 MYCIN - primer

Ekspertni sistemi su računarski programi izgrađeni modelom rešavanja problema pomoću ljudskih eksperata. Jedan od njih je MYCIN (medicinska dijagnoza), tj. njegovo proširenje MYCIN-TEIRESIAS. MYCIN je shvaćen kao interaktivan sistem koji može pomoći doktorima u davanju dijagnoza. Organizovan je u vidu dijaloga između korisnika i samog sistema. Sistem postavlja pitanja za koja su mu potrebni odgovori i vrši zaključivanje na osnovu produkcionih pravila. Osnovni izvor znanja ovog ES je oko 400 produkcionih pravila oblika:

$$P_1 \; and \; P_2 \; and ... \; then \; a_1 \; and \; a_2 ... \; and \; a_j$$

P_i su pravila, a a_i akcije. Predikati su oblika četvorke :

$$< predikat >< objekat >< atribut >< vrednost >$$

Težinski koeficijenti sa vrednostima između 0 i 1 su dati za svaku akciju. Oni izražavaju stepen pouzdanosti. Svaka činjenica ima svoju meru verovatnoće, broj između -1 i 1. $V(F) = V(R) \cdot \min[V(P_j)]$, gde je P_j j-ta premisa pravila R, F je dedukovano iz pravila R. Moguće je izvoditi zaključak iz dva ili više pravila, pri čemu, ako su u pitanju npr. dva pravila, verovatnoć tog zaključka se izvodi na osnovu Bajesove formule $V = V_1 + V_2 - V_1V_2$. Sve $|V| < 0,2$ izbacujemo iz baze znanja. Proces zaključivanja se temelji na i/ili stablu. Za bilo koji cilj sistem razmatra sva pravila čiji se zaključci odnose na cilj; leva strana takvog pravila („I" čvor) je evaluirana i daje koeficijent verovatnoće V koji je jednak po apsolutnoj vrednosti premisi minimalne vrednosti. Ako je $|V| \geq 0,2$ desna strana pravila se konstruiše i ima koeficijent verovatnoće $|V|$. Ako je $|V| < 0,2$ pravilo ne daje informaciju ali ni ne poriče pretpostavke.

Ako neke premise ne mogu biti evaluirane u nekom dostignutom stanju bivaju ostavljene kao unutrašnji podciljevi („ILI" čvorovi). Tek kada se

iscrpe sve mogućnosti traže se nove informacije u obliku pitanja (pitanja se generalizuju kada god je to moguće). Sistem pamti sve informacije koje je dobio i koristio i to da bi:

1. izbegao traženje informacija koje je već koristio ili koje ne može dobiti
2. da bi pokazao ekspertu kako su dobijeni neki zaključci
3. da bi znao šta pitati eksperta kada neke informacije nedostaju

Sistem ima pristup ličnoj prezentaciji znanja i poseduje određenu količinu znanja o tome kroz profile i metapravila:

- Profili su dodani svakom od (u ovom primeru, 24) predikata i govore sistemu koji atributi kontrolišu predikate. Ovo ubrzava izračunavanje tako što pokazuje da li je poseban atribut uključen više puta u različitim premisama kroz različita pravila i tako se postiže brza i gruba preliminarna evaluacija premisa.

- Metapravila su simbolički podaci slične strukture kao pravila koja omogućavaju sistemu napredovanje u pretrazi. Metapravila izražavaju strategiju kojom se izbegava iscrpljivanje nabrajanjem. To su pravila o pravilima. Označena su brojevima i pozivaju se kada su svi podciljevi razmotreni. Mogu izmeniti redosled razmatranja pravila, a isto tako mogu i redukovati broj pravila, te na taj način upravljati pretragom i odsecanjem stabla pretrage.

MYCIN je u stanju i da nudi objašnjenje za svoje odgovore pored samih odluka. Ekspertni sistem TEIRESIAS je zamišljen bio najpre kao prošerenje prethodnog i omogućava sistemu da organizuje telo znanja uz pomoć eksperta, ali i da upravlja znanjem i o njemu zaključuje. Komunikacija se vrši prirodnim jezikom kao između dva eksperta. Njegov model pravila sastoji se iz:

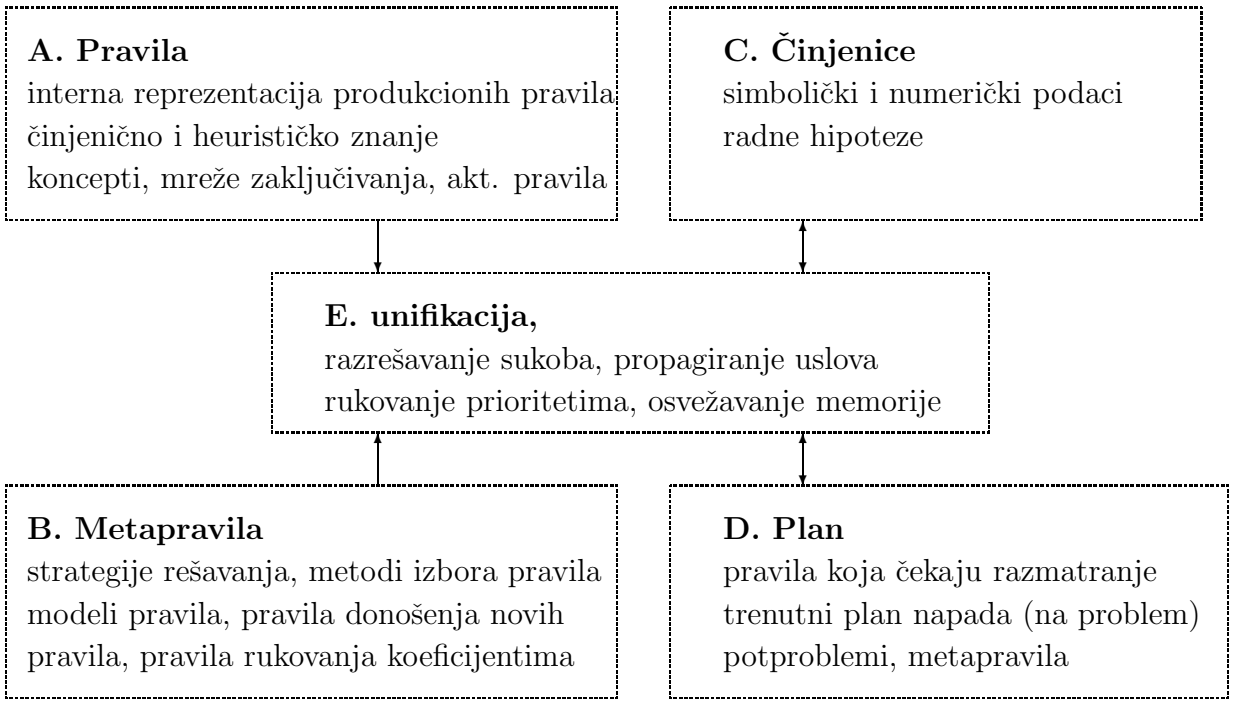
1) liste primera (jednostavan podskup relevantnih pravila)

2) opis karakterističnog člana podskupa
(premise i zaključci karakterizovani atributima koji se često ponavljaju)

3) korelacija između atributa (statistička analiza)

4) dve liste pravila: osnovnih i specifičnih

12.2 Produkcioni sistemi



Produkcioni sistemi se sastoje od produkcionih pravila:

$$LS \rightarrow DS$$

pri čemu leva strana (LS) opisuje situaciju koja je zadovoljena (uslov) - ,,if deo", a desna strana (DS) predstavlja akciju (posledica) - ,,then deo". Pravila su slična teoremama, silogizmima. Osnovni princip zaključivanja je *modus ponens*, ali nije jedini. Dok je rad sa produkcionim pravilima nekako suprotna paradigma u odnosu na proceduralne jezike, s druge strane su formalno ekvivalentni. Prvi su iskorišćeni za definisanje formalnih jezika i gramatika (E. Post, 1936 i Čomski, 1963). Iz skupa pravila koji čine bazu

znanja izdvaja se radni prostor i interpretator. Radni prostor u svakom trenutku sadrži sve činjenice koje je program dedukovao do tog trenutka; u početku sadrži samo tvrđenja problema koji će biti rešavan. Ovaj prostor igra ulogu kratkotrajne memorije koja čuva jednostavna tvrđenja da bi formirala deo statičkog opisa prostora pretrage, dok dugotrajna memorija predstavlja dinamičko znanje koje sadrži operatore transformacije u obliku produkcionih pravila. Pravila mogu uključivati promenljive čije su vrednosti dodeljene pomoću interpretatora u svakom trenutku izvršavanja pravila pa ga tako čine saglasnim sa poznatim činjenicama pomoću semiunifikacije ili filtriranja. Unifikacija je mehanizam koji čini mogućim rad ovakvog sistema tako što pronalazi skup supstitucija koji dve logičke formule čini identičnim (ako to jesu).

Pretraga koja vodi cilju može se predstaviti I-ILI drvetom, kao kod dokazivača teorema (od činjenica ka zaključku ili unazad, obrnuto). Najčešće se koristi obilazak unazad zato što činjenice nisu nezavisne i mnoge od njih neće biti relevantne za delove problema (da bi se izbegla moguća kombinatorna eksplozija). Šta je neophodno da bi se razvio jedan ekspertni sistem?

1. Potrebno je da specijalisti opišu oblast proučavanja i da to znanje formulišu pomoću pravih koncepta

2. Pravila moraju biti izražena, a veza između parametara slaba inače leve strane postaju predugačke i isprepletene

3. Zaključivanje mora biti moguće pomoću jednostavnih procedura (kao što su modus ponens i slično)

4. Sistemi ovog tipa su neprikladni za rešavanje problema čija su tvrđenja ili rešenja komplikovana (npr. problemi kombinatorne prirode) - više su orijentisani promenama oblasti proučavanja nego invarijantama

5. Baza znanja - priroda uključenih entiteta može se razlikovati od jednog do drugog sistema shodno oblasti proučavanja; to može učiniti mogućim adaptaciju nekih sistema za različite oblasti; pogodan fiksiran format je izabran za pravila i parametre koji su tipizirani tako da proces unifikacije može biti zadovoljivo restrikovan tokom primene. Dok su u nekim slučajevima premise i zaključci osnovne činjenice koje mogu biti označene i po potrebi modifikovane, mnogi sistemi u svojim pravilima uključuju i procedure kojima evaluiraju rezultat.

6. Kontrolna struktura - osnovni problem kontrolne strukture je razrešavanje sukoba (*conflict resolution*): donošenje odluke koja pravila pozvati ako postoji više kandidata koji odgovaraju. Strategija izbora je od vitalne važnosti i to ne samo zbog efikasnosti, već i zbog sposobnosti sistema da razume sopstvene akcije i poboljšava performanse.

Postoje tri familije kontrola:

1. Iscrpna pretraga
2. Izbor napravljen izračunavanjem
3. Kontrola pomoću metapravila

Ukratko, metod iscrpne pretrage je pogodan u slučajevima kada je prostor pretrage mali i kada ne postoji opasnost od kombinatorne eksplozije.
U drugom slučaju, sirova strategija bi bila primeniti prvo pravilo koje ,,sistemu dođe pod ruku". Dakle, ovde efikasnost zavisi od redosleda kojim su pravila navedena. Alternativa je evaluacija svakog kandidata, shodno nekom kriterijumu, na primer:

- važnost zaključka i akcije pravila u vreme evaluacije, radi postizanja cilja
- izbor pravila koja mogu biti unifikovana sa činjenicama koje se odnose na najvažnije
- izbor pravila koja mogu dati najspecifičnije zaključke
- izbor determinisan grafom (izvođenja)
- izbor najskorije korišćenog pravila

Kontrola pomoću metapravila zavisi od logičkog izbora, tj. u svakom trenutku pretrage ponašanje prorama zavisi od tekućeg stanja pretrage. Najvažnija karakteristika ovakve kontrole pravila je da metapravila mogu biti dodana baš kao i obična pravila, izmenjena ili obrisana, te da ona pravila koja imaju veliki broj neuspeha bivaju izbačena filtriranjem (ili ona koja izazivaju kružna izvođenja). Mnoge izmene i heuristike su pravljene da bi se ubrzali ovakvi interpreteri (npr. korišćenje samo pravila čije su premise trenutni zaključci). Znanje se može organizovati u složenije jedinice koje reprezentuju složene situacije ili objekte domena, i automatski proces izgrađivanja složenih

struktura je takođe način rešavanja prethodnog problema. Međutim, veliki broj činjenica i pravila sa promenljivama izaziva kombonatornu eksploziju i metaznanje je onda neophodno. Minski je prototip odnosno frejm video kao način da se reši ovaj problem (postoji deo znanja koji se ne menja primenom nekih pravila kojima se dolazi do rešenja, slično pozadini animacije (frame) ispred kojih su dinamički objekti - tako se može smanjiti pretraga i izbor pravila). Kao primeri ekspertnih sistema zasnovanih na produkcionim pravilima navode se DENDRAL i META-DENDRAL (strukturne organske formule), kao i pomenuti MYCIN i TEIRESIAS i mnogi drugi.

12.3 Ekspertni sistemi zasnovani na logici prvog reda

Jezik dizajniran specijalno za modelovanje ljudskog rezonovanja (zaključivanja) postoji dugo: to je jezik matematičke logike. Produkciona pravila ,,situacija - mogući zaključak" izražena u različitim formalizmima u osnovi je teorema tog subjekta koja se odnosi na tvrđenje čija interpretacija može biti tačna ili netačna. PROLOG je programski jezik čije osnovne jedinice nisu instrukcije kao u klasičnim programskim jezicima, već teoreme logike prvog reda. Ove teoreme uključuju promenljive, parametre kvantifikovane univerzalnim i egzistencijalnim kvantifikatorom, izražene Hornovim klauzulama. Primeri ekspertnih sistema napisanih u PROLOGU su PEACE (koji tiče sinteze i analize električnih kola), i MECHO (sistem za rešavanje problema u mehanici). METALOG kao predlog modifikacije PROLOGa dodatnim kontrolnim predikatima (Gallaire, Lasserre) kojima se utiče na izbor literala u klauzuli za unifikaciju ili klauzule naslednika kada ih ima više. SNARK je, takođe, primer jezika logike prvog reda.

12.4 Deklarativno-proceduralna kontroverza

Razlika između proceduralnih i deklarativnih jezika se može uporediti sa različitim paradigmama proceduralnih jezika (funkcionalno - deskriptivno, rekurzivno - iterativno, i drugi odnosi) a suština je razlika između specifikacija i procedura. Znanje, kontrola i zaključivanje su kodirani u proceduralne programe dok deklarativni imaju dostupnu bazu znanja i metapravila. Nije toliko bitna ni prisutnost redosleda proceduralnih programa (instrukcija - osnovna osobina algoritam i proceduralnosti - problemi paralelizma, Petri mreža i programskih shema se na primer ovim bave) u odnosu na deklarativne. Osnovna razlika jezika proceduralnog tipa i deklarativnih jezika u koje bi se

ubrajala i produkcionih pravila leži u samoj kontrolnoj strukturi produkcionih sistema koja je potpuno odvojena od podataka za razliku od proceduralnih programa. Proceduralna kontrolna struktura (npr. if-then) je lokalizovana i dobra je za manje količine znanja i jasne zadatke - desna strana pravila zato može biti poziv procedure, programsko priključenje. Važnost deklarativnih jezika proističe iz njihove kompletne modularnosti: korišćenje, sve komunikacije između različitih delova programa preko osnovnih činjenica, odsustvo potrebe za bilo kakvim redosledom znanja ili činjenica, odsustvo imperativnih instrukcija i mogućnosti bek-treka u procesu rešavanja i u upitima vezanim za pravila.

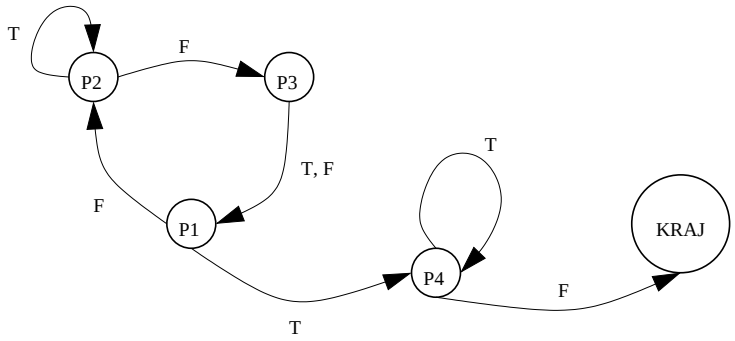
Primer Markovljevog produkcionog sistema za invertovanje niske:

$$
\begin{array}{llll}
R1 & \neq xy & \rightarrow & y \neq x \\
R2 & \neq\neq & \rightarrow & \% \\
R3 & \% \neq & \rightarrow & \% \\
R4 & \%x & \rightarrow & x\% \\
R5 & \% & \rightarrow & \\
R6 & x & \rightarrow & \%x
\end{array}
$$

gde `x` i `y` mogu zameniti bilo koju nepraznu nisku. Petri je 1962. predložio grafove kao način da se prikažu tokovi procesa, a Holt je to 1970. poboljšao tako što je uveo mesta kao čvorove obeležene krugom koji sadrže tokene (elementi koji se odnose na leve strane pravila) i tranzicije obično obeležene vertikalnom crtom kao čvorove koji se aktiviraju ako su sva mesta ulaznih grana popunjena tokenima, a nakon aktivacije tranzicije mesta izlaznih grana dobijaju nove tokene (prema pravilu, opet). Zisman 1978. predlaže modifikaciju pomenutog Markovljevog sistema (u zagradi je naredno pravilo koje se primenjuje):

$$
\begin{array}{llll}
P1 & \neq\neq & \rightarrow & (P4) \\
P2 & \neq xy & \rightarrow \neq x & (P2) \\
P3 & & \rightarrow \neq & (P1) \\
P4 & \neq & \rightarrow & (P4)
\end{array}
$$

Ovde se koriste dva tokena: T (True) i F (False) prema istinitosnoj vrednosti leve strane pravila. Ovako to izgleda prikazano grafički:



Jasno je da su konačni automati i poboljšane mreže prelaska (ATN) specijalni slučajevi Petri mreža. Ovakvom reprezentacijom se lakše uočavaju konflikti pravila (kada se više produkcionih pravila može okinuti) i prikazuju interne strukture pravila interpretatora i sistema sa problemima paralelizma. Međutim, gubi se osobina modularnosti produkcionih sistema i zato je ova reprezentacija bliža proceduralnim.

12.5 Različiti tipovi znanja i njihova reprezentacija

Cilj istraživanja u oblasti VI je kreiranje sistema koji sa jedne strane koristi veliku količinu znanja sakupljenu od strane živih eksperata, i sa druge strane dijalog koji se vodi između sistema i korisnika zasnovan na primerima njihovog ličnog rezonovanja. Posledica ovih zahteva trebao bi biti uspešan sistem za upravljanje velikom i dobro struktuiranom bazom znanja, sa naznačenim razlikama između različitih nivoa znanja, svojstvenom reprezentacijom znanja i dobro definisanim procesom za izmenu informacija između različitih izvora znanja. Možemo reći da ,,sistem zna ono što mi znamo". Sa našeg stanovišta, korišćenje takvih meta znanja je karakteristika svakodnevnog života (na primer, možemo sresti nekoga koga znamo ali da se ne možemo setiti tačno njegovog imena). Mi stalno koristimo heuristike zasnovane na metaznanju, kao ono ,,da je bilo tako, ja bih to znao", i takva pravila zavise od dva osnovna parametra: važnosti koju za nas imaju pojedinačne činjenice i nivoa naše kompetentnosti u toj oblasti.
Navedimo neke osnovne tipove znanja:

1. Osnovni elementi i objekti realnog sveta (naša percepcija fizičkog sveta)

2. Tvrđenja i definicije (odnose se na osnovne objekte)

3. Koncepti (grupisanje i generalizicaija osnovnih objekata, obično PR1 jezikom)

4. Relacije (izražavaju osnovne karakteristike osnovnih elemenata i uspostavljaju uzrozvcno posledične veze između koncepata. Moramo naglasiti da je reprezentacija znanja u ekspertnim sistemima veoma slična modelu korištenom u relacionim bazama podataka - ,,relacija-entitet", familije, frejmovi, skripte su jednostavno izražene binarne relacije.)

5. Teoreme i pravila prezapisivanja (ovo su pojedinačni slučajevi produkcionih pravila sa posebnim karakteristikama)

6. Algoritmi za rešavanje problema

7. Strategije i heuristike (postoje pravila ponašanja, urođena ili stečena, koja nam govore koje će akcije biti poduzete u datoj situaciji i shodno njima regulišemo ponašanje)

8. Meta-znanje (organizovano je u nekoliko nivoa; potrebno je znati šta je poznato, koje je pouzdano mesto u znanju, koji je stepen važnosti napada na predmet znanja u relaciji celog tela; ovo se tiče organizacije tela znanja različitih tipova, kao i kada i kako mogu biti korištena ta znanja)

12.5.1 Reprezentovanje znanja

Pomenute su mnoge reprezentacije znanja, i mogle bi se poređati od ,,zatvorenih" proceduralnih ka ,,otvorenijim", deklarativnim:

- konačni automati (Markov, 1954)
- programi (Meyer, 1978)
- skriptovi (sheme) (Schank, 1977)
- semantičke mreže (McCarthy, 1977)
- frejmovi (Minsky, 1975: u mnogo čemu sličan pojmu objekta u OOP, podrazumeva i proceduralne elemente),
  preteča danas aktuelnih (web) ontologija
- grafovi, mreže (Petri, 1962), ATN

- formalne specifikacije (Germain, 1981)
- predikatski račun (Kowalski, 1979)
- teoreme, (rewrite) pravila prezapisivanja (Huet, 1978)
- produkciona pravila (Shortliffe, 1976)
- prirodni jezik (Pitrat, 1981)

Konačni automati, programi, predikatski račun i produkciona pravila se mogu pokazati ekvivalentnim (Tjuringovoj mašini) ali je velika razlika u njihovoj praktičnoj primerni. Neki noviji formalizmi su nastali kao potreba VI: frejmovi, ili prototipovi, su kompleksne strukture podataka kojima se bavio Minski. Svaki frejm (okvir) sadrži ,,slotove" za objekte koji su uključeni u ,,statičnu situaciju" odnosno okvir kao objekat sa svojim atributima (slotovima, osobinama) koji mogu imati različite tipizirane vrednosti, sa mogućnošću da se slot poziva na drugi frejm; pored toga, sadrži informacije kako se slotovi pune (faceti, daemoni), kako se frejmovi aktiviraju kao i šta je potrebno učiniti u nekim netipičnim situacijama (slično set/get metodama). Tako se konkretne instance okvira mogu poistovetiti sa objektima, gde okviri imaju svoje atribute sa konkretnim vrednostima u konkretnim instanca (model O-A-V, objekat-atribut-vrednost).

Skript, ili (programska) shema, opisuje stereotipan scenario koji uključuje npr. svakodnevne akcije ljudi i više - bavi se dinamičkim aspektima u odnosu na frejmove.

Semantičke mreže ili grafovi, često su kolekcija frejmova i skriptova, koji opisuju elemente prostora pretrage i relacije između njih. Grafovi i dijagrami su bili korišteni u kompjuterskom dokazivanju teorema, ATN (Augmented Transition Networks) predstavlja proširenje takvog koncepta.

12.5.2 Osobine sistema produkcionih pravila

Ekspertni sistemi zasnovani na produkcionim pravilima i srodni ES su među najbrojnijim, i njihove osobine i zbog toga posebno važne. Mane sistema produkcionih pravila:

1. Teškoća dizajniranja produkcionih pravila koja se odnose na predmete znanja

2. Teškoća formulisanja pravila
3. Teškoća korištenja pravila

Prednosti:

1. Modularnost
2. Laka modifikacija
3. Čitljivost
4. Mogućnost samoobjašnjenja

Meta znanje - ako se od eksperta očekuje da ponudi sistemu znanje tj. bazu znanja, od sistema se očekuje da upravlja ovim znanjem, da ga po potrebi struktuira, generalizuje, menja i apstrahuje, da zaključuje o primeni. Znanje sistema o upravljanju na ovaj način je meta-znanje. Ovo je posebno bitno ako je baza znanja velika (slično sistemu za upravljanje bazom podataka). Kod produkcionih sistema se strategije upravljanja znanjem takođe izražavaju kao pravila, samo su izdvojena kao posebna pravila dostupna sistemu.

13 Učenje

Budućnost veštačke inteligencije leži u ovoj oblasti (mašinsko učenje). Svakako je lakše da program sam sakupi informacije tj. nauči nego da ih čovek unosi (a često je to i neophodno kao način borbe sa ogromnom količinom podataka, ili kao specifičnost problema), ali još uvek ne postoji dovoljno dobar takav program (kao primer) ili opšte rešenje. Mogu se izdvojiti pet nivoa učenja:

1. Programirano učenje (već uneto, ,,instinkti", svi kompjuterski programi su ovog nivoa)

2. Mehaničko učenje (sve moguće situacije se pamte, zajedno sa akcijama koje ih slede, ,,uslovni refleksi")

3. Statističko učenje(slično drugom nivou, samo što su sve situacije grupisane u klase dobijene prooučavanjem mnogo primera, i sistem čuva samo opise različitih klasa. Ovo predstavlja učenje bez učitelja. Za njim sledi nivo

4. Učenje pomoću učitelja(sistem može da uči i da vrši generalizaciju)

5. Nema učitelja, na ovom nivou sistem može sam kreirati nove situacije, nove hipoteze i nove koncepte

Navešćemo primer igre označavanja. Program za ovu igru je napisao A. L. Samuel, IBM-ov inženjer, između 1956 i 1967. godine. Igra se igra na šahovskoj tabli gde se pijuni kreću samo unapred, kraljevi ili napred ili nazad i to obe vrste figura samo po jedno polje.
Prvi program koji je Samjuel napisao bio je na nivou mehaničkog učenja. Program je pamtio 180000 pozicija, uzetih iz najbolje odigranih partija. Tada se, naravno, javlja problem pamćenja velike količine informacija te Samjuel koristi tri procedure:

1. Radi bržeg pristupa, situacije su sortirane shodno
   a) broju pijuna i kraljeva na svakoj strani i
   b) prema učestalosti kojom su se pojavljivale u prošloj igri

2. Situacije koje su simetrične između belih i crnih figura tretiraju se simultano (istovremeno)

3. Postojao je metod za eliminaciju (bodovanjem) onih pozicija koje su se najmanje često pojavljivale tokom prethonih igara.

Samjuel je tablu predstavio pomoću 36-to bitne reči i napravio je nekoliko verzija / poboljšanja programa. Koristio je polinomijalnu evaluacionu funkciju $F = \sum_i p_i P_i$, gde su sa p_i označeni težinski faktori , a sa P_i karakteristike (njih 38). Na osnovu upamćenih pozicija i odigranih partija, radi se vrednovanje minimax algoritmom. Korekcije težinskih faktora su vršene na sledeći način - ako je H broj poteza koji su trebali biti bolji prema evaluaciji nego pretraživanjem i L koji su se suprotno pokazali gorim, onda korelacija (ne u pravom statističkom smislu reči) $C = (L - H)/(L + H)$ vrednošću 1 pokazuje idealno poklapanje situacije sa evaluacijom, odnosno potpuno neslaganje vrednošću -1; ako se za svaki p_i proceni da li je slaganje terma $p_i P_i$ za $p_i < 0$ sa evaluacijom veće ili manje računanjem $C_i = (L_i - H_i)/(L_i + H_i)$, tj. ako je $C_i < 0$ onda pozitivan p_i treba uvećati a negativan p_i smanjiti ako je $C_i \geq 0$ i obratno za $p_i \geq 0$ (štaviše, postavlja se $p_k = 1$ za C_k sa najvećom apsolutnom vrednošću i $p_i = 1/2^n$ gde je $n \leq |c_k/c_i| \leq n + 1$, pamti se 16 karakteristika - ostalima $p_i = 0$ što se desi kada tri puta ima najmanju apsolutnu vrednost C_i i zamenjuje se karakteristikom koja je najduže „čekala"). Program je dobio modifikaciju (učenje bez učitelja) gde se npr. takmiče dve funkcije evaluacije td. ona koja izgubi biva manje promenjena, a ako se to ponovi više puta onda se radikalnije promeni („perturbed gradient" koji se kao modifikacija gradijent metode koristi za nekonveksne površi). Dalje, uvođenjem varijante alfa-beta algoritma u kojoj ako se neka ista situaciju pokaže preoptimističčno osenjenom u odnosu na ponovljenu prethodnu situaciju i novu evaluaciju ($F(S) > F'(S)$) evaluacija se upoređuje sa drugom i menja prema odgovarajućim koeficijentima korelacije (sl. C_i samo za $F(S) > F'(S)$).

Pogram je i u prethodnoj verziji posle 20-ak partija igrao veoma dobro i dugo je bio najbolji te vrste, i ovako je brzo konvergirao. Međutim, 15-tak koraka je isto tako dovoljno da promeni potpuno parametre (uči u toku igre nakon svakog koraka) jer minimaks algoritam očekuje da protivnik igra idealno najbolje, pa ako je protivnik loš i program uči i počinje da igra loše - nakon toga protivnik samo treba da zaigra dobro da bi pobedio. Da bi se to prevazišlo koristio je matrice kombinacija karakteristika (nekih 38, prema ekspertima) umesto parametara linearne kombinacije (čime se evaluacija zapravo svodi skoro na pretragu po tabelama) i šest faza igranja prema broju figura, koje je korelacijom trenirao koristeći nekih 180000 utakmica. Ovo se

u [JL] navodi kao Semjuelov primer *generalizovanog učenja*, a pored toga se opisuju: učenje planova, učenje pravila i učenje karakteristika (indukcijom).

13.1 Primer STRIPS

STRIPS je sistem čija je osnovni motiv nastanka i upotrebe upravljanje robotom sa automatizovanim učenjem. Razvijen je kao sistem koji daje sposobnost učenja i planiranja, i koji razvija i pretražuje skupove akcija koje se smatraju korisnim za familje situacija.

Elementarni operatori nad njegovim svetom zadati su kao trojke listi gde je prvi element spisak uslova pod kojim se okida operator, drugi element je lista činjenica koje se brišu okidanjem (negativna lista), a treći element je spisak činjenica koje se dodaju okidanjem operatora (pozitivna lista). Poređajmo niz operatora $O_1, \cdots, O_n$ tako da formira sledeći trougao:

$$
\begin{array}{ccccccc}
O_1 & & & & & & \\
\vdots & \ddots & & & & & \\
O_1' & \cdots & O_k' & \cdots & O_j & & \\
\vdots & & \vdots & & \vdots & \ddots & \\
O_1'' & & O_k'' & & O_j'' & \cdots & O_n
\end{array}
$$

gde se npr. O_k' u redu j trougla odnosi na činjenice dodate operatorom O_k koje važe do primene O_j. Učenje se dešava svaki put kada se pronađe niz operatora koji postiže cilj. Jednom pronađeni niz se pamti i može da se iskoristi i za neke nove probleme. Npr. ako je novi cilj prepoznat u redu j i uslovi trenutne situacije (najpre) u nekom O_i iznad O_j onda je niz $O_i, \cdots, O_j$,,makro" kojim se postiže taj novi cilj.

13.2 Učenje pravila i planova

Igranje pokera kao primer (Waterman, 1970) - vektor (šest) bitnih karakteristika je s leve strane pravila (gde mogu biti i promenljive ili oznaka da je vrednost nebitna, npr. ,,*") i s desne akcija tj. potez igrača. Za svako pravilo se vezuje skup ograničenja promenljivih s leve strane koja se mogu menjati dodavanjem novog ako ne postoji drugo primenjivo pravilo osim izabranog kojem nedostaju samo druga ograničenja. Uvek postoji i mogućnost intervencije eksperta. Uvek se podrazumeva prilikom svakog poteza praćenje pravila po

određenom redosledu sve do poslednjeg koje se uvek pretpostavlja - slučajan izbor poteza sa nebitnom vrednošću svih karakteristika - od ovog pravila se polazi. Da ne bi nekontrolisano rastao, broj pravila se ograničava i mogu se brisati ako se ograničenja nekog pravila ili samo pravilo pokažu takvim da druga pravila budu nepotrebna - novo generalizovano pravilo se dodaje ispred pogrešnog (koje je dalo pogrešnu odluku) ili se restrikuje pogrešno i dodaje ispod. Voterman je ispitao pet različitih programa: 1) slučajno igranje, 2) učenje uz pomoć učitelja, 3) automatizovano učenje bez učitelja, 4) pravila data od dobrog igrača, 5) učenje bez zadatog vektora karakteristika - program 3) se nakon perioda testiranja pokazao veoma uspešnim (evoluciona teorija igara, [MD]). Osnovna zamerka ovakvom učenju je pored nedostatka simboličkih parametara (kao i kod Semjuela) je to što zavisi od redosleda pravila i često zato dolazi do čudnogi i nestabilnog ponašanja. Program bi trebao da bude u stanju da zaključuje (generalizuje) na osnovu samo jednog primera kada god je to moguće, kao i ljudi.

Primer programa za igranje šaha koji polazi od samo dve elementarne strategije: ako je figura napadnuta - pomeri je, i ako figura napada neprijateljsku uzmi napad u obzir (Jacques Pitrat). Za ovo je bitno: razumevanje pozicije (određen potez u stablu poteza dovodi do pozicije u kojoj se neprijatelj napada ako neprijatelj ne odgovori na odgovarajući način, minimaks se primenjuje), pojednostavljenje i generalizacija (za razliku od pokera i dama u šahu postoji previše karakteristika i daleko je teže naići ponovo na istu situaciju), modifikacija i primena planova (generalizovano drveće pretraživanja se čuva u obliku analognom šemama Minskog - generalizovani potezi, polja koja mogu biti prijateljska ili neprijateljska, i drvo pretrage). Jedan način učenja planova bi mogao biti i producioni sistem kojim se utiče na strukturu i parametre pravila i planove, kao i na njihovo dodavanje ili brisanje, [PLAN].

13.3 Učenje karakteristika i koncepta, Vereov primer

Primer učenja u kome se opisuje svet predikatima (u poznatom primeru sveta blokova: iznad, pored, podržava, levo-od, ispred, itd.) i objektima kroz primere i kontraprimere (P. H. Winston, 1970). Ispravan primer se npr. može unifikacijom uporediti i zaključiti da je ispravan. Neispravan primer koji se malo razlikuje je koristan - njime se npr. korak po korak može opisati negacijama složeni objekta, ali ako se sistemu ponudi potpuno različit primer on je nemoćan. Sistem uopšte nemora da konvergira i pati od mnogih drugih

mana (ne pamti ranije primere, nema bektrekinga, itd.).

Vere (1975) daje interesantan model učenja baziran na generalizaciji kao postupku suprotnom od unifikacije. Pravo učenje jeste nalaženje rezultata indukcijom nad manjim brojem primera. Term se zamenjuje promenljivom i primenom takve supstitucije izraz postaje apstraktniji. Term se ovakvom induktivnom supstitucijom nemora uvek zameniti istom promenljivom, a supstitucija se zapisuje kao lista trojki (term, redosled pojavljivanja / promenljiva) i piše se ispred izraza. Na primer, za izraz:

`E=((napravio) Vuk Azbuka)` $\wedge$ `(prosvetitelj Vuk)` $\wedge$ `(pisac Vuk)`

... i supstituciju: `O={(Vuk,(1,3)/x); (azbuka, 1/y)}`, važi:

`OE = ((napravio) x y)` $\wedge$ `(prosvetitelj x)` $\wedge$ `(pisac x)`

Istom promenljivom se nemogu zameniti različiti objekti, i supstitucija ne može da koristi promenljive koje izraza kojeg generalizuje. Ova ograničenja (1) su neophodna ako se želi očuvanje polaznog izraza nakon deduktivne supstitucije iz generalizovanog oblika.

Definicija 13.1 *Izraz E_1 je uopšteniji od izraza E_2 akko postoji induktivna supstitucija O td. $E_1 \subseteq OE_2$ (koristi se konjunktivna normalna forma pa se porede skupovi literala) - piše se $E_1 \leq E_2$.*

Za date E_1, E_2 i O, deo OE_2 koji se javlja u E_1 zove se *vezivanje* (coupling) a ostalo je *ostatak*. Više različitih supstitucija mogu dati isto vezivanje ali različite ostatke. Koriste se *pravedne* supstitucije koje su takve da pored navedenih osobina koriste uvek istu promenljivu za isti term u svim pojavljivanjima, i da se term zamenjuje samo kada je to neophodno tj. term u ostatku se zamenjuje samo ako je već zamenjen u vezivanju. Odatle proističe da redosled pojavljivanja nije potreban za pravedne supstitucije. Generalizacija je stroga ako $E_1 \neq OE_2$ niti je E_1 dobijen prostim preimenovanjem promenljivih u E_2. Kao i u aritmetici (najveći zajednički delilac) sada se može definisati najveća zajednička generalizacija (*nzg*) E za dva data izraza E_1 i E_2 kao

$$E \leq E_1 \wedge E \leq E_2,\ \exists E' : E' < E \leq E_1 \wedge E' < E \leq E_2$$

Koncept je nzg primera (iterativno se može uopštiti na proizvoljan broj primera). Kontraprimeri prečišćavaju ali i komplikuju analizu. U najgorem

slučaju rezultat je disjunkcija svih primera i mogućih slučajeva. Algoritam koristi oblik $A \wedge \neg(B \wedge \neg C)$ umesto $(A \wedge \neg B) \vee (A \wedge B \wedge C)$. Skica Vereovog algoritma:

- ulaz: primer $u_i : \ i = \overline{1,p}$, kontra-primeri $v_j : \ j = \overline{1,q}$
- konstruiše se P1 kao *nzg*;
  - ako ih ima više, formira se ipak njihova disjunkcija,
  - *nzg* ne postoji - kontradikcija ...
- ako postoje negativni primeri, konstruiše se ispravak $N1$ kao *nzg* njihovog ostatka (primeri bez literala kontraprimera pogodnom supstitucijom), i tada je naučen koncept:

$$C1 = P1 \wedge \neg N1$$

- traži se *nzg* ostatka pozitivnih primera kao ispravak `N2` ispravka `N1`, pa naučen koncept postaje:

```
C2 = P1 ∧ ¬ (N1 ∧ ¬ N2)
```

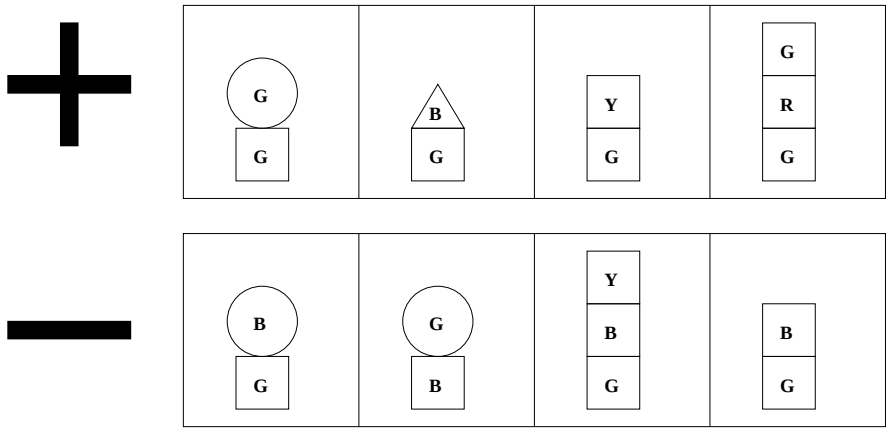
  - ako ih ima više, formira se ipak njihova disjunkcija,
  - *nzg* ne postoji - kontradikcija u ostacima, npr.:
    pozitivan primer: `(x ON y) ∧ (GREEN x)`
    negativan primer: `(x ON y) ∧ (GREEN x) ∧ (RED y)`

- ovaj postupak se ponavlja sve dok se ne dobije konačan naučeni koncept:

```
C = P1 ∧ ¬ (N1 ∧ ¬ (N2 ∧ ¬ (...  ¬ Nk))...)
```

  ili dok se ne dobije prazan *nzg* kao posledica kontradiktornosti ostataka

Primer:



u_1 = **(O1 ON O2)** $\wedge$ **(SPHERE O1)** $\wedge$ **(CUBE O2)** $\wedge$ **(GREEN O1)** $\wedge$ **(GREEN O2)**
u_2 = **(O3 ON O4)** $\wedge$ **(PYRAM O3)** $\wedge$ **(CUBE O4)** $\wedge$ **(BLUE O3)** $\wedge$ **(GREEN O4)**
u_3 = **(O5 ON O6)** $\wedge$ **(CUBE O5)** $\wedge$ **(CUBE O6)** $\wedge$ **(YELLOW O5)** $\wedge$ **(GREEN O6)**
u_4 = **(O7 ON O8)** $\wedge$ **(O9 ON O7)** $\wedge$ **(CUBE O7)** $\wedge$ **(CUBE O8)** $\wedge$ **(CUBE O9)** $\wedge$ **(RED O7)** $\wedge$ **(GREEN O8)** $\wedge$ **(GREEN O9)**
v_1 = **(Q1 ON Q2)** $\wedge$ **(SPHERE Q1)** $\wedge$ **(CUBE Q2)** $\wedge$ **(BLUE Q1)** $\wedge$ **(GREEN Q2)**
v_2 = **(Q3 ON Q4)** $\wedge$ **(SPHERE Q3)** $\wedge$ **(CUBE Q4)** $\wedge$ **(GREEN Q3)** $\wedge$ **(BLUE Q4)**
v_3 = **(Q5 ON Q6)** $\wedge$ **(Q7 ON Q5)** $\wedge$ **(CUBE Q7)** $\wedge$ **(CUBE Q5)** $\wedge$ **(CUBE Q6)** $\wedge$ **(YELLOW Q7)** $\wedge$ **(BLUE Q5)** $\wedge$ **(GREEN Q6)**
v_4 = **(Q8 ON Q9)** $\wedge$ **(CUBE Q8)** $\wedge$ **(CUBE Q9)** $\wedge$ **(BLUE Q8)** $\wedge$ **(GREEN Q9)**

Odatle sledi:

$$nzg(u_1, u_2, u_3, u_4) = \texttt{P1=(x ON y)} \wedge \texttt{(CUBE y)} \wedge \texttt{(GREEN y)}.$$

Kontra-primer $v2$ se eliminiše jer nema zelene kocke u `P1`, a ostaci $u'_i = \{v_i - O_iP1\}$, $i = 1, 3, 4$ se računaju kao:

u'_1 = (SPHERE x) $\wedge$ (BLUE x), O1 = (Q1/x,Q2/y)
u'_3 = (Q7 ON x) $\wedge$ (CUBE Q7) $\wedge$ (CUBE x) $\wedge$ (YELLOW Q7) $\wedge$ (BLUE x), O3 = (Q5/x, Q6/y)
u'_4 = (CUBE x) $\wedge$ (BLUE x), O4 = (Q8/x,Q9/y)

odakle sledi da je `N1=(BLUE x)`, i `C1=(x ON y)` $\wedge$ `(CUBE y)` $\wedge$ `(GREEN y)` $\wedge$ $\neg$ `(BLUE x)`. Među primerima, u_2 sadrži `N1`, pa ovo treba ispraviti ponovo formiranjem ostataka i pomenutim postupkom:

`C` = (x ON y) $\wedge$ (CUBE y) $\wedge$ (GREEN y) $\wedge$ $\neg$ ((BLUE x) $\wedge$ $\neg$(PYRAM x))

Knjige korišćene tokom pisanja ovog rada:

Literatura

[JL] Jean-Louis Lauriere: Problem-Solving and Artificial Intelligence

[GN] Michael R. Genesereth and Nils J. Nilsson: Logical Foundations of Artificial Intelligence

[JS] Minds, Brains and Science, John Searle, 1984.

[TB] Donald E. Knuth: The TeXbook

[PG] Predrag Janičić, Goran Nenadić: OSNOVI LaTeX-A

[HU] Hopcroft, J. & Ullman, J (1979). Introduction to Automata Theory, Languages, and Computation

[MD] Učenje u strateškim igrama, Mihailo Despotovic, 2001.

[VD] Tehnologije inteligentnih sistema, Vladan Devedžić

[PLAN] Plan Optimization by Plan Rewriting, José Luis Ambite, Craig A. Knoblock & Steven Minton

[COGN] Foundation of Cognitive Sciences, edited by Michael I. Posner, `http://books.google.com/books?id=IugxqohJA50C&printsec=frontcover&dq=Pollack+connectionism`

[www] `http://www.aaai.org`
`http://www.idsia.ch/~juergen/goedelmachine.html`
`http://www.pcug.org.au/~dakin/tspbb.htm`
`http://homepage.mac.com/mihailod/thesis/thesismain.html`
`http://sitemaker.umich.edu/soar/home`

[MW] Mathworld, Wolfram Research, `http://www.wolfram.com`

[W] `http://en.wikipedia.org/wiki/Propositional_calculus`
`http://en.wikipedia.org/wiki/Hilbert_systems`
`http://en.wikipedia.org/wiki/Category:Mathematical_logic`
`http://en.wikipedia.org/wiki/Formal_system`

http://en.wikipedia.org/wiki/Model_theory
http://en.wikipedia.org/wiki/Traveling_salesman_problem
http://en.wikipedia.org/wiki/Dynamic_programming
http://en.wikipedia.org/wiki/Multiset
http://en.wikipedia.org/wiki/List_of_algorithms
http://en.wikipedia.org/wiki/Modal_logic
http://en.wikipedia.org/wiki/Intuitionistic_logic
http://en.wikipedia.org/wiki/Heyting_algebra
http://en.wikipedia.org/wiki/Boolean_network

ARTIFICAL INTELLIGENCE ([JL], Chapter 1)
REPRESENTATION OF PROBLEMS ([JL], Chapter 2)
FORMAL SYSTEMS ([JL], Chapter 3 - umesto toga [GN], Introduction)
Declarative Knowledge ([GN], Chapter 2)
Inference ([GN], Chapter 3)
Resolution ([GN], Chapter 4)
Resolution Strategies ([GN], Chapter 5)
Nonmonotonic Reasoning ([GN], Chapter 6)
Induction ([GN], Chapter 7)
Reasoning with Uncertain Beliefs ([GN], Chapter 8)
Knowledge and Belief ([GN], Chapter 9)
Metaknowledge and Metareasoning ([GN], Chapter 10)
State and Challenge ([GN], Chapter 11)
Intelligent Agent Architecture ([GN], Chapter 12)
CLASSICAL METHODS FOR
PROBLEM-SOLVING ([JL], Chapter 4)
SOLUTION OF PROBLEMS BY
PROPAGATION AND ENUMERATION ([JL], Chapter 5)
GAME-PLAYING PROGRAMS,
PSYCHOLOGY OF PROBLEM-SOLVING ([JL], Chapter 6)
EXPERT SYSTEMS ([JL], Chapter 7)
ALICE ([JL], Chapter 8)
LEARNING ([JL], Chapter 9)

Gotovi seminarski, maturski, maturalni i diplomski radovi iz raznih oblasti, lektire , puškice, tutorijali, referati - specijalizovan tim za usluge visokokvalitetnog pisanja, istraživanja i obradu teksta za kompletan region Balkana.

Posetite nas na sajtovima ispod:

WWW.MATURSKIRADOVI.NET

WWW.SEMINARSKIRAD.ORG

WWW.MATURSKI.NET

WWW.MATURSKI.ORG

WWW.SEMINARSKIRAD.INFO

Dostupni smo Vam 24h 365 dana u godini.

Za gotove verzije rada obratiti se na mail:

maturskiradovi.net@gmail.com

061/ 11-00-105

Seminarski, diplomski, maturski radovi, prevodi na engleski i eseji…

.