

Tartu Ülikool
Arvutiteaduse instituut

TEHISINTELLEKT

Mare Koit, Tiit Roosmaa

Tartu 2011

Toimetaja Tõnu Tamme

Joonised Sven Aller

Käesoleva õpiku väljaandmist on toetanud Euroopa Sotsiaalfondi projekt „Informaatika ja infotehnoloogia magistriõppekavade arendamine Tartu Ülikooli arvutiteaduse instituudis koostöös Eesti infotehnoloogiaettevõtetega”.



Euroopa Liit
Euroopa Sotsiaalfond



Eesti tuleviku heaks

Autoriõigus: Mare Koit, Tiit Roosmaa, 2011

ISBN 978–9949–19–634–0

Tartu Ülikooli Kirjastus
www.tyk.ee

EESSÕNA

Käesolev õppevahend esitab Tartu Ülikoolis õpetatavate kursuste Tehisintellekt I ja (osaliselt) Tehisintellekt II materjali. Autorid on nimetatud kursusi õpetanud enam kui kümne aasta vältel ning püüdnud siinkohal arvesse võtta ka nende aastate jooksul üliõpilastelt saadud tagasisidet.

Õppevahendi koostamisel oleme kasutanud mitmeid maailmas tunnustatud tehisintellekti-õpikuid ja monograafiaid, eeskätt (Russell, Norvig 2003; Ginsberg 1999; Winston 1992; Rich, Knight 1991), veebimaterjale, samuti isiklike kontaktide kaudu saadud informatsiooni, sh professorite Wolfgang Wahlsteri, Ulrich Furbachi ja Karin Harbuschi (Saksamaa), Dmitri Pospelovi ja Vadim Stefanjuki (Venemaa) ning Yorick Wilksi (Inglismaa) loengutel kuuldut.

Avaldame südamlikku tänu kõigile, kes selle õppevahendi valmimisele on kaasa aidanud, eriti Sven Allerile, kes valmistas joonised, ja Tõnu Tammele, kes käsikirja hoolikalt toimetas.

Vastutus puudujääkide ja vigade eest lasub ainult autoritel.

SISUKORD

| | |
|--|-----|
| I. TEHISINTELLEKTI ARENG JA PÕHIKÜSIMUSED | 6 |
| 1. Mis on tehisintellekt? | 6 |
| 2. Tehisintellektisüsteemide põhiliigid ja ehitus | 13 |
| 3. Tehisintellekti põhiküsimused | 17 |
| 4. Tehisintellekti tehnika | 18 |
| 5. Probleemid ja probleemilahendus | 26 |
| II. TEADMUSE ESITUS | 32 |
| 6. Teadmuse mõiste. Teadmusbaasi loomine | 32 |
| 7. Produktsiooniline teadmuse esitus | 35 |
| 8. Semantilised võrgud | 43 |
| 9. Freimid | 48 |
| 10. Esimest järku predikaatarvutus teadmuse esitusel | 52 |
| 11. Mittetäielik teadmus ja mittemonotoonsed loogikad | 60 |
| 12. Hägus teadmus. Hägusad hulgad ja loogikad | 62 |
| 13. Teadmuse esituse meetodite võrdlus | 67 |
| III. PROBLEEMILAHENDUS | 71 |
| 14. Probleemilahendusmeetodite liigitus | 71 |
| III.A. Otsing | 73 |
| 15. Pimeotsing puul | 73 |
| 16. Pimeotsing graafil | 79 |
| 17. Heuristiline otsing graafil | 81 |
| 18. Heuristilise otsingu algoritme | 87 |
| 19. Otsing mänguseisude puul | 94 |
| III.B. Osandamine | 101 |
| 20. Probleemilahendus osandamise teel | 101 |
| III.C. Teoreemide automaatne tõestamine | 105 |
| 21. Herbrandi meetod | 105 |
| 21.1. Valemi teisendamine disjunktide hulgaks | 105 |
| 21.2. Disjunktide hulga H-interpretatsioon | 107 |
| 21.3. Semantilised puud | 109 |
| 21.4. Herbrandi meetod | 111 |
| 22. Resolutsioonimeetod | 113 |
| 22.1. Resolutsioonimeetod lausearvutuse jaoks | 113 |
| 22.2. Resolutsioonimeetod 1. järku predikaatarvutuse jaoks | 114 |
| 22.2.1. Asendus ja unifikaator | 115 |
| 22.2.2. Unifitseerimisalgoritm | 115 |
| 22.2.3. Resolutsioonimeetod predikaatarvutuse jaoks | 117 |
| 23. Teoreemide automaatne tõestamine resolutsioonimeetodil | 122 |
| 24. Küsimustele vastamine kui teoreemide tõestamine | 125 |
| III.D. Teisi probleemilahendusmeetodeid | 129 |
| 25. Tuletused semantilises võrgus ja freimide süsteemis | 129 |
| 26. Kitsenduste rahuldamine | 134 |

| | |
|---|-----|
| IV. PLANEERIMINE | 141 |
| 27. Probleemilahendus ja planeerimine | 141 |
| 27.1. Otsing tasandite hierarhias | 141 |
| 27.2. Metatasand tasandite hierarhias | 143 |
| 28. Probleemilahenduse planeerimine: süsteem STRIPS | 143 |
| 29. Plaani esitamine kolmnurktabelina | 151 |
| 30. Probleemilahenduse planeerimine eesmärkide koostoime korral: süsteem RSTRIPS | 153 |
| 31. Probleemilahenduse planeerimine eesmärkide koostoime korral: süsteem DCOMP | 159 |
| 32. Hierarhiline planeerimine: süsteem ABSTRIPS | 165 |
| V. TEHISINTELLEKTI TEISI KÜSIMUSI | 169 |
| 33. Suhtlus loomulikus keeles | 169 |
| 33.1. Loomuliku keele automaattöötlus | 169 |
| 33.2. Keele mudelid | 172 |
| 33.2.1. Formaalsed grammatikad | 172 |
| 33.2.2. Olekuautomaadid | 177 |
| 33.2.3. Loogikamudelid | 179 |
| 33.2.4. Tõenäosuslikud mudelid | 179 |
| 33.2.5. Keelemudelite arengusuunad | 180 |
| 33.3. Kõnetuvastus ja kõnesüntees | 181 |
| 33.4. Dialoogsüsteemid | 183 |
| 33.4.1. Juturobotid | 183 |
| 33.4.2. Praktilised dialoogsüsteemid | 185 |
| 33.4.3. Ratsionalism ja empirism | 186 |
| 33.4.4. Kõnedialoogsüsteemide näiteid | 188 |
| 33.5. Eesti keeles suhtlev arvuti | 189 |
| 33.5.1. Eesti dialoogikorpus | 190 |
| 33.5.2. Loomuliku keele töötlus dialoogsüsteemis | 191 |
| 33.6. Suhtlus arvutiga tulevikus | 192 |
| 34. Tehisnägemine | 193 |
| 35. Masinõpe | 194 |
| 35.1. Masinõppe meetodid | 194 |
| 35.2. Induktiivõpe | 195 |
| 35.2.1. Tõenäoliselt ligikaudu korrektne õpiprotseduur | 195 |
| 35.2.2. Piiratud hüpoteesiruumi eeldus | 196 |
| 35.2.3. Eelarvamus $x > c$ | 199 |
| 35.2.4. Otsustuspüü | 200 |
| 35.3. Juhendajaga ja juhendajata õpe | 202 |
| 36. Tehisintellekti programmeerimiskeeled | 203 |
| Tehisintellekti ajalugu (lühikokkuvõte) | 208 |
| Võtmeküsimused läbi aegade | 209 |
| KIRJANDUS | 210 |
| MÄRKSONAD | 212 |

I. TEHISINTELLEKTI ARENG JA PÕHIKÜSIMUSED

1. Mis on tehisintellekt?

Intellekti on defineeritud kui mõtlemisvõimet, mõistust, aru; oskust õigesti reageerida uuele olukorrale. Intelligentsus (intellekti omamine) on siis taibukus, arukus, oskus olemasolevaid teadmisi efektiivselt kasutada, probleeme lahendada, ratsionaalselt mõelda, võime mõista ümbritsevat maailma. Tehisintellekt (*artificial intelligence*) on loomuliku intellekti jäljendamine, s.t arvutisüsteemi võime täita funktsioone, mida üldiselt seostatakse inimhõimusega, nt arutleda ja õppida.

Eesti Entsüklopeedia (1998) defineerib tehisintellekti, esiteks, kui tehisaru, arvuti suutlikkust jäljendada inimese vaimset tegevust, ja teiseks, kui arvutiteaduse ja -tehnikat haru, mis uurib tehisaru omavate arvutisüsteemide loomise meetodeid ja ajuprotsesside modelleerimist elektronarvutil. Mõistet tehisintellekt kasutatakse seega kahes tähenduses – ta võib tähistada nii teatava arvutisüsteemi omadust kui ka uurimissuunda, mis tegeleb selliste arvutisüsteemide, nn tehisintellektisüsteemide väljatöötamisega. Uurimissuuna tähistamiseks kasutatakse eesti keeles ka terminit intellektitehnika või intellektika.¹

Uurimissuuna teke

Esimesed elektronarvutid loodi teatavasti üle kuuekümnepäevase tagasi (1943. a Inglismaal spetsialiseeritud arvuti Colossus salakirja dešifreerimiseks; aastatel 1943–1945 Ameerika Ühendriikides universaalne arvuti ENIAC). Juba oma loomise algusest peale hakati arvuteid kasutama mitte ainult puht-arvutuslike ülesannete lahendamisel, vaid nende abil püüti imiteerida ka muid protsesse, mis on iseloomulikud inimese loominguks, intellektuaalsele tegevusele.

Üks varasematest arvutite „mittearvutuslikest” rakendustest oli masintõlge: esimene avalik masintõlkealane eksperiment viidi läbi 1954. a Ameerika Ühendriikides, kus IBM-tüüpi arvuti tõlkis vene keelest inglise keelde 200 sõnast koosneva teksti.

1950-ndatel aastatel loodi ka esimesed programmid arvutitega mitmesuguste mängude mängimiseks (sh trips-traps-trull, domino, kabe). Koostati programme, mille järgi töötades arvutid tõestasid teoreeme, komponeerisid muusikat, luuletasid, s.t tegid kõike seda, mida traditsiooniliselt loetakse inimese intellektuaalseks tegevuseks. Neil esimestel mittearvutuslike ülesannete lahendamise programmidel oli aga üks iseloomulik piiratus: iga programm oli määratud ühe konkreetse ülesande lahendamiseks, mistõttu polnud päris õige väita, et sellise programmi järgi töötades „masin mõtleb”. Näiteks kirjutati programm vene kabe mängimiseks. Selle alusel ei suutnud arvuti mängida 100-ruudulist kabet, rääkimata juba muudest mängudest, sest iga selline programm kujutas endast jäika protseduuri, mis ei olnud võimeline kohanema ega ümber organiseeruma. Esimeste mittearvutuslike ülesannete programmeerimine tegi selgeks, et kui tahame arvutil jäljendada inimese

¹ IT terministandardi sõnastik <http://www.keeveeb.ee> (18.02.2010)

loomingulist tegevust, siis ei piisa sellest, kui koostame programme üksikute ülesannete lahendamiseks, vaid arvutil tuleb imiteerida universaalseid protseduure, nn **metaprotse-duure**, mis suudaksid genereerida konkreetseid protseduure üksikülesannete lahendamiseks. Just selle asjaolu mõistmisest hakkaski arenema uurimissuund, mis hiljem nimetati tehisintellektiks. Uue teooria esimeseks ülesandeks sai imiteerida metaprotseduure, mida inimene rakendab loominguliste ülesannete lahendamisel.

Termin *Artificial Intelligence* võeti kasutusele 1956. a, kui John McCarthy kutsel kogunes Ameerika Ühendriikides Dartmouthis grupp erinevate erialade teadlasi, keda ühendas eesmärk jäljendada arvutil inimese intellekti. Hiljem on küll arvatud, et teadusliku uurimissuuna nimetusena pole see termin sobiv, ja pakutud selle asemele muid termineid, mis aga siiski tunnustust pole leidnud.

Esimene uue teooria raames valminud programm oli 1957. a Ameerika Ühendriikides loodud GPS (General Problem Solver; autoriteks A. Newell, J.C. Shaw ja H. Simon). See programm imiteerib ühte metaprotseduuri – „labürindi läbimine”. Autorid arvasid esialgu, et nad ongi avastanud üldise metaprotseduuri, mis on kogu inim mõtlemise aluseks. Algul lahendas GPS tõepoolest mitmesuguseid erinevaid ülesandeid: tõestas 1. järku predikaat-arvutuse teoreeme, mängis mõningaid mänge. Kuid malemänguga ei tulnud ta toime. Siitpeale sai tehisintellektialaste uuringute eesmärgiks otsida uusi metaprotseduure.

Vahel arvatakse, et intellektitehnika eesmärgiks on luua seadmed, mis täielikult imiteeriks inimest („tehisinimene”). Lähem eesmärk on siiski tagasihoidlikum: välja selgitada need mehhanismid, millel põhineb inimese intellektuaalne tegevus, et kasutada neid konkreetsete teaduslik-tehniliste ülesannete lahendamisel (näiteks keskkondades, kus inimese viibimine on võimatu või ohtlik) – luua nn tehisintellektisüsteeme (TIS).

Tehisintellekti definitsioonid

Üks esimesi tehisintellekti definitsioone pärineb inglise matemaatikult A.M. Turingilt (Turing 1950). See on nn **Turingi test**: kui masin vastab eksperimendiaatori (loomulikus keeles esitatud) küsimustele nii, et pole võimalik vahet teha, kas vastajaks on masin või inimene, siis on ta testi läbinud. See ei eelda sugugi, et kõik vastused peavad olema õiged, sest ka inimene pole eksimatu. Muuhulgas võib masin püüda eksperimendiaatorit alt vedada, et testi edukalt läbida. Näiteks kui küsitakse „Kui palju on 12345×789 ?”, siis pidada pikk paus ja vastata paari numbriga võrra valesti.

Kuigi Turingi artikli ilmunisest möödunud kuue aastakümne jooksul on jõudsalt arenenud arvutustehnika ja kasvanud meie teadmised intellekti olemusest, pole seni siiski veel suudetud koostada programmi, mis tuleks toime Turingi testiga. 1990. a pani dr Hugh Loebner välja kuldmedali ja 100 000 USA dollari suuruse auhinna arvutiprogrammi eest, mis esimesena läbib Turingi testi.² 1991. aastast alates korraldatakse iga-aastasi võistlusi, kus ekspertide kogu hindab programme, mis suhtlevad inglise keeles. Siiani on saadud välja anda ainult pronksmedalit ja lohutusauhindu parimatele programmidele. Kolmel korral (2000, 2001 ja 2004) on võitnud Richard Wallace'i programm Alice³, aastatel 2005 ja 2006 aga Rollo Carpenteri programmid (vastavalt Jabberwacky ja Joan), kus kasutati

² <http://www.loebner.net/Prize/loebner-prize.html> (28.02.2011)

³ <http://www.alicebot.org/> (18.02.2010)

masinõpet. 2009. a võistlus toimus konverentsil Interspeech⁴ Brightonis, võitis programm Do-Much-More (autor David Levy)⁵. 2010. a Los Angeleses toimunud võistluse võitis programm Suzette, mille autor oli Bruce Wilcox.

Õpikutest ja monograafiatest võib leida veel mitmesuguseid tehisintellekti (intellekti-tehnika) definitsioone. Olgu järgnevas toodud mõned näited.

N. Nilsson (1971): TIS on masin, mis täidab toiminguid, milleks tavaliselt on nõutav inimese intellekt.

J. McCarthy ja P. Hayes (1969): TIS on masin, mida iseloomustavad järgmised tunnused:

- 1) välismaailma mudeli olemasolu,
- 2) oskus seda mudelit kasutades vastata mitmesugustele küsimustele,
- 3) oskus muutavas maailmas eesmärgipäraselt käituda.

Analoogilise definitsiooni annab D. Michie (1971): TIS-i iseloomustavad järgmised tunnused:

- 1) maailma mudeli olemasolu,
- 2) oskus koostada plaane,
- 3) oskus analüüsida plaanides sisalduvaid alternatiivseid võimalusi,
- 4) plaanide muudetavus,
- 5) maailma mudeli muudetavus.

P. Winston (1977): intellektitehnika on teadus kontseptsioonidest, mis võimaldavad arvutil teha seda, mis inimeselt nõuab arukust. Eesmärgiks on muuta arvutid kasulikumaks ja uurida printsiipe, millel põhineb inimese intellekt.

E. Rich (1983): intellektitehnika uurib meetodeid eksponentsiaalse keerukusega ülesannete lahendamiseks polünomiaalse ajaga.

S. Russell ja P. Norvig (Russell, Norvig 2003) süstematiseerivad tehisintellekti (-süsteemi) definitsioonid vastavalt sellele, missugust inimtegevuse aspekti modelleeritakse ja kas arvuti peaks jäljendama inimest või mitte:

- 1) süsteem, mis mõtleb nagu inimene (süü kuuluvad tehisintellekti sellised käsitlused, mis asetavad rõhu inim mõtlemise jäljendamisele, nt otsustuste tegemine, probleemi-lahendus, õppimine);
- 2) süsteem, mis tegutseb nagu inimene (nt Turingi test; pole oluline, et jäljendataks inim mõtlemist, küll aga inimekäitumist);
- 3) süsteem, mis mõtleb ratsionaalselt (nt Richi definitsioon; pole oluline jäljendada inim mõtlemist või -käitumist, küll aga „õige mõtlemise” seaduste kasutamine);
- 4) süsteem, mis tegutseb ratsionaalselt (nn ratsionaalsed agendid, kus rõhk asetatakse arukale käitumisele).

Intellektitehnikat loetakse küll arvutiteaduse ja -tehnika haruks, kuid nagu nähtub kas või eespool toodud määratlustest, on tal seoseid ka paljude muude teadustega: filosoofia, matemaatika, psühholoogia jne (vt tabel 1.1).

⁴ <http://interspeech2009.org/conference/> (18.02.2010)

⁵ <http://www.chatbots.org/chatbot/do-much-more> (18.02.2010)

Tabel 1.1. Intellektitehnika seos teiste teadustega

| <i>Teadusharu</i> | <i>Mida kasutab intellektitehnika</i> |
|-------------------|--|
| Filosoofia | Loogika, arutlusmeetodid, mõistus kui füüsikaline süsteem, õppimine keel, ratsionaalsus. |
| Matemaatika | Formaalsed esitused ja tõestused. Algoritmid, arvutamine, (mitte)lahenduvus, tõenäosus. |
| Psühholoogia | Kohanemine, taju ja motoorne juhtimine, eksperimenditehnikad (psühhofüüsika jt). |
| Majandusteadus | Ratsionaalsete otsustuste formaalne teooria. |
| Keeleteadus | Teadmiste formaalne esitamine, grammatika. |
| Neuroteadus | Vaimse tegevuse füüsikalised alused. |
| Juhtimisteooria | Püsisüsteemid, stabiilsus, lihtsa optimaalse agendi kavandamine. |

Kaks lähenemisviisi intellekti modelleerimisele

Ajalooliselt on kõrvuti arenenud kaks lähenemisviisi intellekti modelleerimisele – neuroküberneetika ja „musta kasti” küberneetika.

Neuroküberneetika aluseks on eeldus, et ainus mõtlemisvõimeline objekt on inimaju. Seepärast tuleb tehisintellekti saamiseks jäljendada inimaju ehitust.

Neuroküberneetika püüabki modelleerida inimaju struktuuri, nii tarkvaraliselt kui ka riistvaraliselt. Neuronite matemaatiline modelleerimine sai alguse 1943. a W. McCullochi ja W. Pittsi töodes. 1958. a defineeris F. Rosenblatt tajuri (*perceptron*) mõiste. Tajur on inimaju hierarhilise struktuuri lihtsustatud mudel, lihtne tehisnärvivõrk, mis suudab õppida ja mustreid ära tunda. Hiljem on võetud kasutusele ka tunduvalt keerulisemaid tehisnärvivõrke. J. Hopfield lõi 1980. aastate lõpul tehisnärvivõrkude teooria ja näitas ühtlasi, kuidas tehisnärvivõrk suudab lahendada tuntud rändkaupmehe probleemi (vt ka näide 5.3) konstantse ajaga, samal ajal kui tavaliste algoritmide ajavajadus on eksponentsiaalne. 1981. a alustati Jaapanis nn 5. põlvkonna arvuti projekti, kus seati eesmärgiks luua neuron-arvuti, milles tehisintellekt oleks saavutatud inimese närvisüsteemi modelleerimise teel. Kahjuks ei õnnestunud seda projekti planeeritud tähtajaks (1992) ellu viia, kuid ühe olulise tulemusena töötati siiski välja paralleelsümboltöötuse tehnoloogia.

„Musta kasti” küberneetika väidab, vastupidi, et pole tähtis, kuidas funktsioneerib tehisintellekt – peaaasi, et ta reageeriks sisendile nii nagu inimaju. Eesmärk on töötada välja algoritme selliste ülesannete lahendamiseks, mis inimeselt nõuavad mõtlemisvõimet, arukust, loominguilisust. Varased tehisintellektisüsteemid põhinesid loogikal ja probleemide lahendamiseks viisid läbi sümboltöötlust. 1960. aastatel lõi J. McCarthy sümboltöötuseks programmeerimiskeele Lisp. Väga paljud intellektitehnika meetodid esindavad just „musta kasti” küberneetikat, sh heuristiline programmeerimine, matemaatilise loogika

meetodid (sh teoreemide automaatse tõestamise meetodid), samuti toimivad sel viisil tänapäeva tööstuslikud ekspertsüsteemid.

Räägitakse ka tugevast (*strong*) ja nõrgast (*weak*) tehisintellektist. Tugeva tehisintellekti all mõistetakse loogistel alustel põhinevat tehisintellekti ning nõrga tehisintellekti all tehismärgivõrke, geneetilisi algoritme ja evolutsioonilisi meetodeid kasutavat tehisintellekti. Praeguseks on selgunud, et kumbki lähenemisviis ei ole täiuslik ja parima tulemuse võiks anda nende kombinatsioon.

Intellektitehnika eesmärgid ja suunad

Tehisintellekt on suhteline mõiste: mitu inimpõlve tagasi oleks isegi tänapäeva lihtne taskuarvuti tunnistanud intelligentseks. Liati pole kindel, kas intellekt tema täies mahus üldse on modelleeritav. Uuringutel on praegu kaks põhieesmärki:

- 1) inimese loominguilise tegevuse toetamine ja võimendamine (nt teoreemide tõestamine, peamurdmisülesannete lahendamine, mängude mängimine, diagnostika- jms probleeme lahendavate ekspertsüsteemide loomine),
- 2) inimese asendamine rasketes või eluohtlikes tingimustes (intelligentsete robotite loomine).

Algul oli intellektitehnika eeskätt teoreetiline distsipliin, nüüdseks aga on olemas juba palju praktilisi rakendusi, sh ekspertsüsteemid ja robotid.

Olgugi et siiani pole leitud täiuslikke lahendusi nt sellistele tehisintellekti probleemidele nagu automaatne tõlkimine ühest loomulikust keelest teise, kõne mõistmine, tehismärgimine jms, saab ainevaldkonda kitsendades ometi luua kasulikke lahendusi. Näiteks rakendatakse loomuliku keele süsteeme andmebaaside kasutajaliidestest, kus sisend on kitsendatud lihtlausetega. Samuti on kättesaadavad kõnelejust sõltumatud kõnetuvastussüsteemid (enamasti küll inglise keele jaoks). Neid kasutatakse koos ekspertsüsteemidega telefonikõnekeskustes, kus nad suudavad edukalt asendada inimametnikke ja täita klientide tellimusi (olgugi et sellised süsteemid ei läbiks Turingi testi).

Ekspertsüsteeme on kombineeritud andmebaasidega, eesmärgiga tuvastada andmetes mustreid, nagu seda teeb inimene, ja automaatsete otsuste tegemise süsteemidega andmekaevanduse (*data mining*) abil. Üks tähtis rakendus on lennujaama turvasüsteem, mis kasutab nägude tuvastust.

Intellektitehnikas võib eristada kolme suunda:

- 1) teoreetiline – intellekti olemuse uurimine ja seletamine (sellega tegelevad eeskätt psühholoogid);
- 2) praktiline – kommertsproduktide loomine (nt ekspertsüsteemid);
- 3) arvutiteaduslik – intellekti modelleerimine arvutil, selleks vajalike programmeerimiskeelte ja -tehnikate väljatöötamine.

Intellektitehnika üht peamist ülesannet näevad paljud uurijad selles, et muuta arvutid kasulikumaks ja kasutajate jaoks mugavamaks.

Rahvusvaheline koostöö

Intellektitehnika on kujunenud laialdaseks uurimisvaldkonnaks, millega tegeldakse paljudes kohtades üle terve maailma. Korraldatakse konverentse ja ilmub suur hulk mitmesuguseid teadusajakirju.

Intellektitehnika valdkonnas toimub mitmeid pikaajaliste traditsioonidega rahvusvahelisi konverentse. Tähtsaim neist on rahvusvaheline tehisintellekti konverents IJCAI⁶ (International Joint Conference on Artificial Intelligence), mis leiab aset igal paaritul aastal:

1. 1969 USA (Washington)
2. 1971 Inglismaa (London)
3. 1973 USA (Stanford)
4. 1975 Gruusia (Thbilisi)
5. 1977 USA (Cambridge)
6. 1979 Jaapan (Tokyo)
7. 1981 Kanada (Vancouver)
8. 1983 Saksamaa (Karlsruhe)
9. 1985 USA (Los Angeles)
10. 1987 Itaalia (Milano)
11. 1989 USA (Detroit)
12. 1991 Austraalia (Sydney)
13. 1993 Prantsusmaa (Chambery)
14. 1995 Kanada (Montreal)
15. 1997 Jaapan (Nagoya)
16. 1999 Rootsi (Stockholm)
17. 2001 USA (Seattle)
18. 2003 Mehhiko (Acapulco)
19. 2005 Suurbritannia (Edinburgh)
20. 2007 India (Hyderabad)
21. 2009 USA (Pasadena)
22. 2011 Hispaania (Barcelona)⁷

Igal paaris aastal toimub Euroopa tehisintellekti konverents ECAI (European Conference on Artificial Intelligence):

1. 1974 Suurbritannia (Brighton)
2. 1976 Suurbritannia (Edinburgh)
3. 1978 Saksamaa (Hamburg)
4. 1980 Holland (Amsterdam)
5. 1982 Prantsusmaa (Orsay)
6. 1984 Itaalia (Pisa)
7. 1986 Suurbritannia (Brighton)
8. 1988 Saksamaa (München)
9. 1990 Rootsi (Stockholm)
10. 1992 Austria (Viin)
11. 1994 Holland (Amsterdam)
12. 1996 Ungari (Budapest)
13. 1998 Suurbritannia (Brighton)

⁶ <http://ijcai.org/> (18.02.2010)

⁷ <http://ijcai-11.iia.csic.es/> (18.02.2010)

14. 2000 Saksamaa (Berliin)
15. 2002 Prantsusmaa (Lyon)
16. 2004 Hispaania (Valencia)
17. 2006 Itaalia (Riva del Garda)
18. 2008 Kreeka (Pátra)
19. 2010 Portugal (Lissabon)
20. 2012 Prantsusmaa (Montpellier)⁸

Korraldatakse ka piirkondlikke konverentse, nt Skandinaaviamaade tehisintellekti konverents SCAI (Scandinavian Conference on Artificial Intelligence):

1. 1987 Norra (Tromsø)
2. 1989 Soome (Tampere)
3. 1991 Rootsi Stockholm,
4. 1993 Taani (Århus)
5. 1995 Norra (Trondheim)
6. 1997 Soome (Helsingi)
7. 2001 Taani (Odense)
8. 2003 Norra (Bergen)
9. 2006 Soome (Espoo, toimus koos Soome tehisintellekti konverentsiga STeP-2006)
10. 2008 Rootsi (Stockholm)
11. 2011 Norra (Trondheim)⁹

Euroopas koordineerib tehisintellektialaseid uuringuid 1982. a loodud ECCAI (the European Coordinating Committee for Artificial Intelligence)¹⁰.

Tähtsamad keskused tehisintellekti alal on aegade jooksul olnud Ameerika Ühendriikides (Massachusettsi Tehnoloogiainstituut¹¹, Stanfordini Uurimisinstituut¹², Carnegie Melloni Ülikool¹³), Jaapanis (kus arendatakse silmapaistval tasemel robotikat¹⁴) ja Saksamaal (uurimiskeskus¹⁵ Saarbrückenis, Kaiserslauternis ja Bremenis).

Eestis tegeldakse tehisintellektiga põhiliselt Tallinna Tehnikaülikoolis ja Tartu Ülikoolis. TTÜ küberneetika instituudi tarkvaraosakonnas (mis loodi 1978. a akad Enn Tõugu juhtimisel, praegune juhataja on Ahto Kalja) uuritakse programmide sünteesi, ontoloogiale orienteeritud programmeerimist, modelleerimisülesannete lahendamist tehisintellektisüsteemide abil jpm; juhtimisüsteemide osakonnas (juh Jüri Vain) kontseptuaalset modelleerimist jms; foneetika ja kõnetehnoloogia laboris (juh Einar Meister) eestikeelse kõne tuvastamist ja tehiskõne sünteesi. TTÜ informaatika instituudis (dir prof Rein Kuusik) tegeldakse teadmussüsteemide arendamisega. Tartu Ülikoolis tegutseb keeletehnoloogia uurimisrühm (selle rajas akad Haldur Õim, rühma liikmeid on nii matemaatika-informaatika- kui ka filosoofiateaduskonnast), kus käsitletakse eesti keele automaattöötamise probleeme, sh suhtlust arvutiga (TIS-iga) eesti keeles. Eesti keele arvutitoe väljatöötamist toetab ka riiklik programm „Eesti keeletehnoloogia” (2011–2017); sellele eelnes riiklik

⁸ <http://www.lirmm.fr/ecai2012/> (18.12.2010)

⁹ <http://events.idi.ntnu.no/scai2011/> (18.12.2010)

¹⁰ <http://www.eccai.org/> (18.02.2010)

¹¹ <http://www.csail.mit.edu/index.php> (18.02.2010)

¹² <http://www.ai.sri.com/> (18.02.2010)

¹³ <http://www.cmu.edu/> (18.02.2010)

¹⁴ <http://transit-port.net/Lists/Robotics.Org.in.Japan.html> (18.02.2010)

¹⁵ <http://www2.dfki.de/web/> (18.02.2010)

programm „Eesti keele keeletehnoloogiline tugi” (2006–2010). TÜ arvutiteaduse instituudis tegeldakse intelligentsete õpisüsteemide loomisega (Rein Prank). Varem on tegeldud kujundite tuvastamisega (Mati Tombak).

2. Tehisintellektisüsteemide põhiliigid ja ehitus

TIS-i all mõistetakse enamasti arvutiprogrammi, mis modelleerib intellektuaalset tegevust. (Vahel ka riist- ja tarkvara sümbioosi, nt robotit koos sisseehitatud arvuti ja programmiga.)

Põhiliike on kaks:

- 1) ekspertsüsteem (tööd maailma esimese ekspertsüsteemi Dendral kallal algasid Stanfordini ülikoolis juba 1965. a),
- 2) intelligentne robot (esimesed loodi aastal 1970).

Vahel loetakse eraldi TIS-ideks ka näit.

- loomuliku keele mõistmise süsteemid,
- kõnetuvastajad ja -süntesaatorid,
- visuaalse info töötlemise süsteemid,
- intelligentsed programmeerimissüsteemid,
- intelligentsed õpisüsteemid.

Ideaalis peaksid sellised süsteemid allsüsteemidena kuuluma ka ekspertsüsteemi ja (intelligentse) roboti koosseisu.

Praegu eksisteerivad TIS-id suudavad lahendada enamasti kitsa ainevaldkonna probleeme.

Ekspertsüsteemid

Briti arvutiühingus on heaks kiidetud järgmine ekspertsüsteemi definitsioon: *ekspertsüsteem on süsteem, mis ühendab arvuti võimalused ja eksperdi teadmised sellises vormis, et süsteem suudab anda mõistlikku nõu või realiseerida antud ülesande mõistliku lahenduse.*

Ekspertsüsteem on seega TIS, mis abistab mingi ainevaldkonna spetsialisti tema ülesannete lahendamisel (näit arsti haiguste diagnoosimisel, arheoloogi leidude dateerimisel, geoloogi maavarade leiukohtade prognoosimisel, keemikut keeruliste ühendite struktuuri määramisel jne).

Tänu sellele, et ekspertsüsteemi loomisel hangitakse teadmisi vastava ainevaldkonna parimatelt asjatundjatelt (seda teevad nn teadmusesenerid), suudab ka keskmise kvalifikatsiooniga spetsialist ekspertsüsteemi abil töötada paremini kui ilma sellise süsteemita, rääkimata juba madala kvalifikatsiooniga töötajast.

Mõned valdkonnad on olnud ekspertsüsteemide loomisel eriti populaarsed, sh:

- meditsiiniline diagnostika (nt ekspertsüsteem MYCIN, mille loomine algas Stanfordini ülikoolis 1972, autorid E. Shortliffe, B. Buchanan jt),
- tehniline diagnostika,
- geoloogiline diagnostika,
- tootmise planeerimine,

- ainete keemilise struktuuri määramine (nt eespool juba nimetatud Dendral, autorid E. Feigenbaum jt),
- teadmusbasiside konstrueerimine,
- ekspertüsteemide vahendid (nn ekspertüsteemide „kestade” ehk tühjade ekspertüsteemide loomine). Siin on teadmised selgelt eraldatud tuletusmootorist, see võimaldab luua kiiresti uusi ekspertüsteeme. Kolm olulist komponenti on: reeglid (produktioonireeglid, mis modelleerivad inimese probleemilahendust), kest (teadmus ja tuletusreeglid on lahutatud) ja teadmus (kui ekspertiisi võti).

Enne kui asuda looma ekspertüsteemi, tuleb vastata mitmele küsimusele:

1. kas probleemi saab efektiivselt lahendada tavalise programmeerimisega? Kui jah, siis ei ole vaja luua ekspertüsteemi. Näiteks kui on vaja diagnoosida mingit seadmestikku ja kõik sümptomid on ette teada, siis piisab lihtsast tabelist või otsustuspuust. Ekspertüsteemid sobivad paremini sellistes situatsioonides, kus efektiivsed algoritmid puuduvad (nn halvasti struktureeritud probleemide puhul).
2. Kas valdkond on sidus või mitte? Näit. peavalu diagnoosimisel on vaja nii meditsiinilisi kui ka neurokeemia, molekulaarse biofüüsika jpm teadmisi.
3. Kas ekspertüsteemi järele on vajadus ja soov?
4. Kas leidub inimeksperite, kes on valmis ekspertüsteemi loojatega koostööd tegema?
5. Kas ekspert suudab seletada oma teadmisi arusaadavalt?
6. Kas probleemilahenduseks vajalik teadmus on põhiliselt heuristiline ja ebaselge?

Ekspertüsteeme luuakse tavaliselt siis, kui probleemilahendusel on vaja käsitleda ebakindlust. See võib tekkida nii sisendandmetes kui ka teadmusbasisis endas. Paljud inimteadmised on heuristilised, s.t nad võivad töötada korrektselt mitte alati, vaid ainult osa ajast. Samuti võivad sisendandmed olla mittekorrektset, mittetäielikud, mittekooskõlalised ja muidu vigased. Algoritmilised lahendused ei ole suutelised neid situatsioone käsitlema, kuna algoritm peab tagama probleemi lahenduse lõpliku sammude järjendi tulemusel.

Sõltuvalt sisendandmetest võib ekspertüsteem anda kas korrektse või ebakorrektse vastuse või üldse mitte vastata. Siiski ei tegutse hea ekspertüsteem halvemini kui parim probleemilahendaja – inimekspert –, kuid võib teha seda paremini.

Robotid

Robotite arengus eristatakse kolme põlvkonda. *Intelligentne robot* kuulub kolmandasse põlvkonda.

Esimese põlvkonna moodustavad tööstusrobotid, nt programmjuhtimisega tööpingid. Sellise roboti välismaailm peab olema rangelt fikseeritud, ta töötab täpselt programmi järgi, mis on koostatud konkreetse ülesande lahendamiseks.

Teise põlvkonda kuuluvad anduritega robotid, mis erinevalt tööstusrobotitest suudavad vastu võtta ja analüüsida teatavat infot välismaailmast. Nende välismaailm ei pea seetõttu olema täiesti fikseeritud, objektid selles võivad teatavates piirides oma asukohti muuta ning robot leiab need tänu oma anduritele. Selline robot tegutseb küll jäiga programmi alusel nagu tööstusrobotidki, kuid suudab sellesse ise paigutada mõningate parameetrite väärtusi.

Kolmanda põlvkonna robotid ehk intelligentsed robotid võtavad andurite abil vastu infot välismaailmast (nii on tagatud nende nägemine, kuulmine, kompimine jm meeled), nad on autonoomsed, mobiilsed ja suudavad ka ise muuta keskkonda.

Esimesed intelligentsed robotid loodi 1970. aastate algul. Tuntuks on saanud STRIPS (Richard Fikes ja Nils Nilsson, 1971) tänu oma planeerimissüsteemile (seda käsitleme ka 4. peatükis) ja SHRDLU (Terry Winograd, 1971) kui üks esimesi loomulikku (inglise) keelt mõistvaid roboteid.

STRIPS (Stanford Research Institute Problem Solver) oli ratastel liikuv robot, mis tegutses üksteisest vaheseintega eraldatud ja ustega ühendatud ruumides, kust ta suutis korralduste kohaselt kaste leida ja neid ümber paigutada, oma tegevust eelnevalt planeerides.¹⁶ Robot oli varustatud telekaamera ja puudutuste detektoriga ning tal oli raadioside arvutiga.

SHRDLU (loodud Massachusettsi Tehnoloogiainstituudis) modelleeris roboti kätt nn „kuubikute maailmas”, kus paiknesid erineva värvusega risttahukad, püramiidid, silindrid jm geomeetrilised kehad. Vastavalt inglise keeles antud korraldustele leidis see käsi objekte ja paigutas neid ümber, demonstreerides sellega korraldustest arusaamist. Vajadusel suutis robot esitada täpsustavaid küsimusi.¹⁷

Intelligentseid roboteid saab kasutada sellistes tingimustes, kus inimese viibimine on võimatu või ohtlik, nt teisel planeedil, tuumakoldes või metallurgiatehases, kus tootmisprotsess kulgeb tervist kahjustavates tingimustes. Sellise roboti kavandamisel tuleb välja töötada roboti maailma mudel ning ette näha roboti tegutsemine ja otsuste vastuvõtmine ka ebatäieliku informatsiooni tingimustes.

Robootikas arendatakse praegu kahte suunda: tööstuslikku ja teaduslikku. On loodud inimkujulisi roboteid, mis suudavad käia, joosta, ronida (Asimo¹⁸), väljendada näoilmega mitmesuguseid emotsioone (Kismet), mänguloomalaadseid roboteid (Aibo¹⁹, Furby²⁰), mis suudavad täita loomulikus keeles antud käsklusi, jpm.

Tehisintellektisüsteemi ehitus

TIS-i tuumaks on teadmusbaas, mis võimaldab süsteemil probleemide lahendamisel valida efektiivseid meetodeid ja teha arukaid otsuseid. Teadmusbaas sisaldab nii faktiandmeid kui ka üldistatud teadmisi: tuletustehnikaid, algoritme, programmimoduleid. Väliskeskonnaga suhtlemise tagab kasutajaliides. Suhtlemine võib toimuda loomulikus keeles (ekspertsüsteemi puhul on see kindel nõue; loomulike keele asemel kasutatakse ka lihtsustatud keelt või erialažargooni). Põhilised ehitusplokid on interpreteerija (keskkonnast saabuvate stiimulite mõistmiseks ja probleemide püstitamiseks), lahendaja (probleemide lahendamiseks), õppimisplokk (mille ülesandeks on uusi teadmisi hankida ja probleemi-lahendusel kasutatud teadmisi üldistada) ning selgitusplokk (mille abil suudab nt ekspert-

¹⁶ http://www.rci.rutgers.edu/~cfs/472_html/Planning/STRIPS_472.html (18.02.2010)

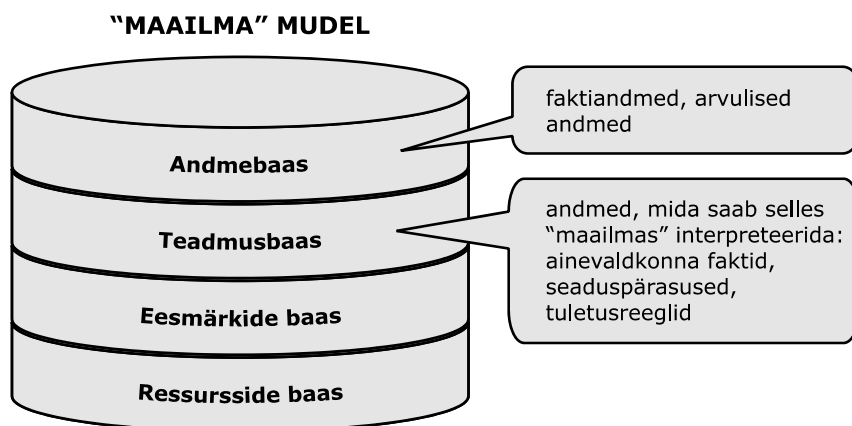
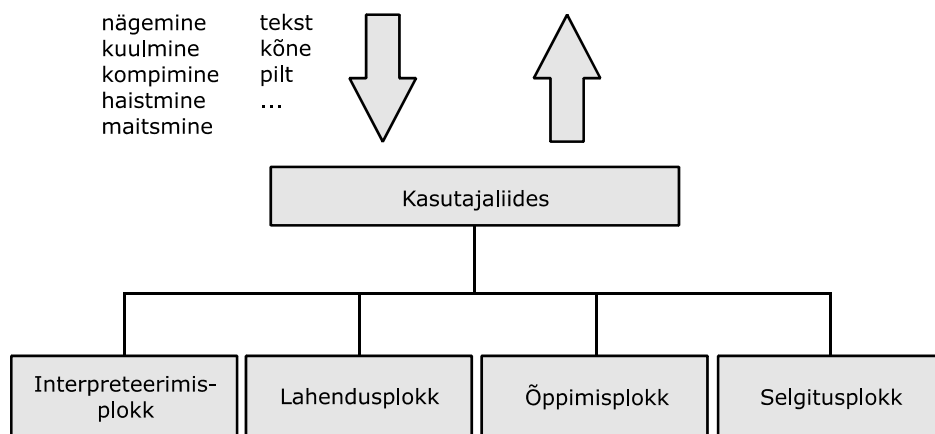
¹⁷ <http://hci.stanford.edu/~winograd/shrdlu/> (18.02.2010)

¹⁸ <http://world.honda.com/ASIMO/> (18.02.2010)

¹⁹ <http://support.sony-europe.com/aibo/index.asp> (18.02.2010)

²⁰ <http://www.virtualpet.com/vp/farm/furby/furby.htm> (18.02.2010)

süsteem kasutajale selgitada ja põhjendada probleemilahenduse samme). Ekspertsüsteemi skemaatiline ehitus on näidatud joonisel 2.1.



Joonis 2.1. Ekspertsüsteemi ehitus

Käesoleval ajal projekteeritakse enamasti selliseid TIS-e, mille „maailmaks” on üks kitsas ainevaldkond. Kuid samas on ette võetud ka suurejoonelisi projekte, nt Cyc²¹, mida alustati juba 1984. a Douglas Lenati juhtimisel: luuakse ülisuurt teadmusbaasi koos tuletusmootoriga. Aastal 2000 alustati Chris McKinstry juhtimisel projekti GAC (*Generic Artificial Consciousness* ehk üldine tehisteadvus), millesse loodeti kaasata suur hulk veebikasutajaid, et nende kaasabil koguda 2010. aastaks miljard nn arupikslit ehk elementaarset fakti (*mindpixel*) ja ehitada üles tohtu suur tehishärvivõrk. Pärast McKinstry surma 2006. a jaanuaris on see projekt seiskunud.

²¹ <http://www.cyc.com/> (18.02.2010)

3. Tehisintellekti põhiküsimused

Tehisintellekti põhiküsimusteks nimetatakse põhilisi uurimisülesandeid, mille lahendamiseks intellekti modelleerimisel aegade jooksul on tegeldud (ja siia maani tegeldakse).

1) **Teadmiste (e teadmuse) esitus** (*knowledge representation*). Uuritakse, missuguseid teadmisi modelleeritavast ainevaldkonnast esitada teadmusbaasis ja missugust formalismi selleks kasutada, kuidas efektiivsel viisil leida teadmusbaasist konkreetse probleemi lahendamiseks vajalikke teadmisi, kuidas kontrollida, et teadmised pole omavahel vastuolus, kuidas kõrvaldada vananenud teadmisi jms. 1969. a püstitasid J. McCarthy ja P. Hayes nn raamide (e piiride) probleemi (*frame problem*): kuidas piiritleda fakte, mis ei muutu, ja neid, mis muutuvad, kui sooritatakse tegevusi. Nt kui sinine pink värvitakse punaseks ja tõstetakse aeda, siis kuidas kirjeldada teadmust, et kahe järjestikuse tegevuse tulemusel asub aias punane (aga mitte sinine) pink.

On välja töötatud ja kasutatud mitmeid erinevaid teadmuse esituse meetodeid. Näiteks esitatakse teadmust:

- kui-siis reeglina (e produktsioonidena),
- loogikavalemitena,
- graafina, mille tippudeks on ainevaldkonna mõisted, kaarteks aga nende vahelised seosed (sh semantilised võrgud, freimid, kontseptuaalsete sõltuvuste graafid, skriptid jms).

Teadmuse esitust käsitleme lähemalt 2. peatükis.

2) **Probleemilahendus** (e lahendi tuletamine, *problem solving*). On olemas nii universaalseid kui ka teadmuse esituse formalismist tingitud, spetsiifilisi probleemilahendusmeetodeid. Universaalseks meetodiks on otsing olekute ruumis, mis võimaldab näiteks leida, kuidas malemängus jõuab valgete malenditega mängija sellisesse lõppolekusse, kus ta on võitnud, s.t kuidas kõrvõimalikke lubatavaid malelaua seisest esitavas ruumis jõuda algolekust lõppolekusse, kasutades lubatavaid käike. Semantilises võrgus (graafis, mille tippudeks on ainevaldkonna mõisted ning kaarteks nende vahelised semantilised seosed) saab teha tuletusi, kasutades soo-liigi seost: nt kõik elusolendi omadused on ka koeral.

Probleemilahendusmeetoditele on pühendatud 3. peatükk.

3) **Planeerimine** (*planning*) on probleemilahenduseks vajalike tegevuste järjendi analüüsimine enne nende täitmist, selleks et valida välja parim järjend. Enamiku mittetriviaalsete probleemide lahendamisel tuleb omavahel kombineerida probleemilahenduse strateegiaid ja teadmuse esituse mehhanisme. Sageli on kasulik jaotada probleem väiksemateks osaprobleemideks ja lahendada need eraldi. Raskete probleemide lahendusplaanide konstrueerimiseks kasutatakse mitmesuguseid tehnikaid, nt eesmärkide magasin, mittelineaarne ja hierarhiline planeerimine jm.

Planeerimist käsitleme 4. peatükis.

4) **Suhtlemine** (*interaction*) on infovahetus tehisintellektisüsteemi ja väliskeskkonna vahel. See võib toimuda mitmel viisil, nt võib intelligentne robot temperatuuri- ja lõhnanandurite näitude alusel lülitada sisse teatava seadme või jooksvalt olukorda analüüsides reguleerida tänavaliiklust. See eeldab, et oskame modelleerida inimese meeli (nägemist,

kuulmist jne), aga ka meelte kaudu saabuva info mõistmist (nt tänavapildist või inimesest arusaamist).

5) **Õppimine** (*learning*) on uute teadmiste omandamine. Tehisintellektisüsteemides kasutatakse mitmesuguseid masinõppe tehnikaid, nt parameetrite korrastamine, induktiivõpe, õppimine analoogia alusel jpt.

Suhtlemist ja õppimist tehisintellektisüsteemides vaatleme 5. peatükis.

Tehisintellektialaste uurimuste põhivaldkonnad kaasajal

Intellektitehnika on kujunenud laiahaardeliseks teadusalaks. Ometi võib väita, et põhi küsimused on endiselt aktuaalsed. Otsitakse uusi ja järjest paremaid meetodeid teadmuse esituseks ning teadmusbaaside loomiseks ja kasutamiseks, täiustatakse klassikalisi probleemilahendusalgoritme ja luuakse uusi (nt nn pehmed arvutused ebakindlastest teadmistest usaldusväärsete järelduste tegemiseks), koostatakse rakenduspakette intelligentsete süsteemide tööstuslikuks väljatöötamiseks, luuakse uusi programmeerimiskeeli ja uusi arvutiarhitektuure (nt paralleelarvutid), uuritakse ja rakendatakse masinõppe tehnikaid, arendatakse roboteid, tegeldakse kujundite automaatse tuvastamise ja pildi-analüüsiga, loomuliku keele mõistmisega.

Seda kinnitab ka tehisintellekti konverentside temaatika. Nii näiteks käsitleti rahvusvahelisel konverentsil IJCAI-2009²² selliseid teemasid nagu teadmiste esitamine ja arutamine, planeerimine, otsing, masinõpe, loomuliku keele automaattöötlus, intelligentsete agentide süsteemid, robotika jpt. Mitmed ideed on intellektitehnikasse tulnud mängude mängimise ja peamurdmisülesannete lahendamise analüüsimisest. On loodud koguni rahvusvaheline arvutimängude assotsiatsioon (*International Computer Games Association*, ICGA, mis korraldab programmide võistlusi males, kabes jt mängudes, samuti iga-aastasi teaduslikke konverentse. Kommertsrakendused soodustavad arvutimänguprogrammide arendamist.

4. Tehisintellekti tehnika

Üks tulemus tehisintellekti viiekümneaastasest arengust on tõdemus, et *intellekt nõuab teadmisi*. Tehisintellekti tehnikat võib määratleda kui meetodit, mis kasutab teadmisi, kusjuures

- teadmised on *üldistatavad*: pole tarvis esitada teadmisi eraldi iga üksiksituatsiooni jaoks, vaid situatsioonid, millel on ühiseid omadusi, grupeeritakse. Kui teadmistel poleks seda omadust, siis oleks nende esitamiseks ja töötlemiseks vaja palju rohkem mälu ja aega.
- Inimesed, kes peavad teadmisi hankima, *mõistavad* neid. Ehkki mõnedel juhtudel võib andmeid hankida automaatselt (näiteks lugeda mõõteriistalt), on palju tehisintellekti valdkondi, kus teadmisi peavad hankima inimesed, kes teevad seda nendes terminites, millest nad aru saavad.

²² <http://ijcai-09.org/> (18.02.2010)

- Teadmisi saab lihtsalt *modifitseerida*, et vajaduse korral parandada vigu ja kajastada muudatusi meie maailmakäsituses.
- Teadmisi võib kasutada inimese *piiratud võimete laiendamiseks*.

Tehisintellekti ülesandeid saab lahendada ka tehisintellekti tehnikaid kasutamata (tõsi, need lahendused ei ole eriti head). Teiselt poolt, tehisintellekti tehnikatega saab lahendada ka muid, mitte ainult tehisintellekti probleeme. Mis siis iseloomustab tehisintellekti tehnikat? Et vastata sellele küsimusele, vaatame kahte näidet: 1) trips-traps-trulli mängimine ja 2) käsitsi kirjutatud trükitähe äratundmine. Mõlema ülesande puhul esitame kolm algoritmi, millest ainult viimane esindab tehisintellekti tehnikat (Rich, Knight 1991).

Trips-traps-trull

Toome siin kolm programmi selle mängu mängimiseks. Iga järgmine programm on üldisem ja paindlikum kui eelmine ning viimane esindab seda, mida võib nimetada tehisintellekti tehnikaks.

Programm 1

Andmestruktuurid

Mängulaua seisu esitame kui vektorit, mille komponendid vastavad laua positsioonidele järgmiselt:

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |

Iga komponendi väärtus on kas 0 (tühi ruut), 1 (ruudus on X) või 2 (ruudus on O).

Eeldame, et igale mänguseisule on eelnevalt leitud parim võimalik järgnev seis ja kõikvõimalikud mänguseisud on järjestatud nii, et seisu esitav vektor määrab parima jätku järjekorranumbri selles järjendis.

Algoritm

Et teha käiku, toimi järgmiselt.

1. Vaata lauda esitavat vektorit kui 3-ndarvu. Teisenda see 10-ndarvuks.
2. Leitud 10-ndarvu kasuta viidana järjekorranumbrile seisude järjendis, leia vastava järjekorranumbriga vektor. (Igast seisust on seega üksainus jätk.)
3. Sammul 2 valitud vektor näitab laua seisu pärast käiku, mis tuleb teha. Sea mängulauale vastavusse see vektor (s.t soorita see käik).

Kommentaariid

See algoritm on ajaliselte efektiivne ja teoreetiliselte mängib ta optimaalset trips-traps-trulli.

Puudused:

- Vajab palju mälu, et säilitada kõikvõimalike seisude järjestatud hulka (mis määrab korrektse käigu igast mänguseisust lähtudes).
- Inimene peab eelnevalt tegema palju tööd, et seda järjestust koostada (seejuures on eksimisvõimalus suur).

- Kui me soovime laiendada mängu n -mõõtmeliseks, siis võib kõikvõimalike lauaseisude järjestatud hulga salvestamine vajada nii palju mälu, et see ületab meie arvuti mälu mahu. (Näiteks $n=3$ korral oleks vaja salvestada 3^{27} mängupositsiooni, milleks kulub ligikaudu 10^{13} baiti = 10^{10} kilobaiti = 10^7 megabaiti.)

Selles algoritmis realiseeritud tehnika ei rahulda tehisintellekti tehnika nõudeid.

Programm 2

Andmestruktuurid

Mängulauda kujutame vektorina nagu eelmiseski variandis, kuid vektori komponentide väärtused olgu: 2 – tühi ruut, 3 – X, 5 – O.

Algoritm

Kasutatakse kolme protseduuri:

Leia2 – püüab leida mängulaua reas tühja ruudu (2). Esmalt katsub mängida tsentrisse. Kui see pole võimalik, siis katsub mängida ükskõik millisesse mitteturkmissesse ruutu.

Võiduvõimalus(p) – tagastab 0, kui mängija p (kas X või O) ei saa järgmisel käigul võita. Vastasel korral tagastab selle ruudu numbrit, mis sisaldab võitva käigu.

„Meie” võib olla kas X või O. Mängija X teeb käigud nr 1,3,5,7,9; O teeb käigud 2,4,6,8.

See protseduur kutsutakse välja algul selleks, et otsustada, kas „meie” saame võita ning kui vastus on jaatav, siis kuidas. Kui „meie” võime võita, siis teeme selle käigu. Kui „meie” ei või võita, siis kutsume selle protseduuri välja selleks, et otsustada, kas vastane võib järgmisel käigul võita. Kui jah, siis peame seda blokeerima. Protseuur kontrollib iga rida, veergu ja diagonaali (kokku 8). Testida saab sel teel, et korrutatakse ruutudes olevad arvud. Kui korrutis on 18 (=3·3·2), siis saab X võita, kui korrutis on 50 (=5·5·2), siis saab O võita. Pärast võitva rea (veeru, diagonaali) leidmist kontrollitakse selle iga elementi (ruutu), et määrata, milline neist on tühi.

Mine(n) – teeb käigu ruutu n .

Algoritmil on iga võimaliku käigu jaoks sisseehitatud strateegia. Mängija X teeb paarituarvulised ja mängija O paarisarvulised käigud. Iga käigu jaoks on eraldi strateegia:

1 – Mine(1) (ülemisse vasakusse nurka; s.t mängija X alustab ülemisest vasakust nurgast).

2 – Kui laud[5] (s.o mängulaua positsioon 5) on tühi, siis Mine(5), muidu Mine(1).

3 – Kui laud[9] on tühi, siis Mine(9), muidu Mine(3).

4 – Kui Võiduvõimalus(X) ei ole 0, siis Mine(Võiduvõimalus(X)) (s.t blokeeri vastase võit), muidu Mine(Leia2)).

5 – Kui Võiduvõimalus(X) ei ole 0, siis Mine(Võiduvõimalus(X)) (s.t võida), muidu kui Võiduvõimalus(O) ei ole 0, siis Mine(Võiduvõimalus(O)) (s.t blokeeri võit), muidu kui laud[7] on tühi, siis Mine(7), muidu Mine(3). (Katse teha „kahvlit”.)

6 – Kui Võiduvõimalus(O) ei ole 0, siis Mine(Võiduvõimalus(O)), muidu kui Võiduvõimalus(X) ei ole 0, siis Mine(Võiduvõimalus(X)), muidu Mine(Leia2).

7, 8, 9 – Kui Võiduvõimalus(meie) ei ole 0, siis Mine(Võiduvõimalus(meie)), muidu kui Võiduvõimalus(vastane) ei ole 0, siis Mine(Võiduvõimalus(vastane)), muidu Mine ükskõik kuhu.

Kui arvuti alustab selle algoritmi järgi, siis tekib valikuvõimalus alles 4. käigul (eeldusel, et mõlemad osalejad mängivad selle strateegia järgi). Aga inimene ei tarvitse mängida täpselt nii (ja arvuti ei tarvitse alustada mängu).

Kommentaariid

See programm pole nii efektiivne ajas kui eelmine, kuna ta kontrollib enne käigu tegemist erinevaid tingimusi. Kuid ta on efektiivsem mälu mõttes. Samuti on tema strateegiat lihtsam mõista ja soovi korral muuta. Üldise strateegia määrab aga ikkagi programmeerija. Samuti pole võimalik hõlpsalt üldistada selle programmi teadmisi teistele valdkondadele, näiteks 3-dimensionaalsele trips-traps-trullile.

Programm 2'

Sama mis 2, kuid mängulaua ruudud on teisiti nummerdatud.

| | | |
|---|---|---|
| 8 | 3 | 4 |
| 1 | 5 | 9 |
| 6 | 7 | 2 |

Selline numeratsioon moodustab maagilise ruudu (summa 15). Tänu sellele saab lihtsustada võiduvõimaluse kontrolli. Säilitame kummagi mängija jaoks nimestikku nendest ruutudest, millesse ta on juba käinud, et märkida muudatust laual pärast käigu tegemist. Kontrollimaks, kas mängija saab võita, vaatleme iga ruutude paari, mis sellel mängijal (nimestikus) on. Arvutame vahe: 15 – (nende kahe ruudu summa). Kui see vahe on mittepositiivne või ta on suurem kui 9, siis ei ole esialgsed kaks ruutu kollineaarsed ja neid võib ignoreerida. Kui see vahe kuulub lõiku [1,9], siis kontrollime, kas kolmas ruut vastavas reas (veerus, diagonaalil) on tühi. Kui jah, siis leidub käik, mis viib võidule. Kuna kumbki mängija ei saa omada nimestikus rohkem kui 4 ruutu, siis tuleb kontrollida vähem kui 8 ruutu, s.t vähem ruute kui eelmises programmis.

Seega näeme, et teadmuse esituse valik võib mõjutada ülesande lahendamise efektiivsust.

Siit tekib võrdlus: kuidas lahendavad ülesannet inimesed ja kuidas seda teevad arvutid? Miks inimese jaoks on rea kontrollimine lihtsam kui järjekorranumbri väljaarvutamine (mis arvutis on efektiivsem)? Praegu pole veel piisavalt teadmisi selle kohta, kuidas toimub infotöötlus inimese peas, kuid arvatakse, et inimesed on *paralleelprotsessorid* ja suudavad vaadata üheaegselt mängulaua erinevaid osi, mida traditsioonilised arvutid ei suuda. Uurides, kuidas inimesed lahendavad ülesandeid, jõuame arusaamisele, kuidas seda peaksid tegema arvutid. Nagu eespool mainitud, on selleks kaks lähenemisviisi: kas püüda täpselt jäljendada inimest (nagu seda teeb neuroküberneetika) või mitte (nagu seda teeb „musta kasti” küberneetika).

Programm 3

Andmestruktuurid

Iga lauaseisuga seome

- 1) 9-komponendilise vektori (nagu varemgi),
- 2) nimestiku lauaseisudest, mis võivad tekkida järgmise käigu tulemusel,
- 3) arvu, mis esitab hinnangut, kui tõenäoliselt see lauaseis viib „meie” võiduni.

Algoritm

Otsustamaks järgmist käiku, vaata kõikvõimalikke lauaseise, mis võivad tekkida iga võimaliku käigu tulemusel. Otsusta, milline seis on parim (nii, nagu kirjeldatakse allpool), tee käik, mis viib sellesse seisu, ja võta selle (parima) käigu hinnang jooksva seisu hinnanguks.

Otsustamaks, milline lauaseisude hulgast on parim, tee iga seisu jaoks järgmist:

1. Kontrolli, kas see on võit. Kui jah, siis anna talle väga kõrge hinnang.
2. Vastasel korral vaata kõiki võimalikke vastase järgmisi käike. Kontrolli, milline neist on halvim „meie” jaoks (kutsudes rekursiivselt välja sedasama protseduuri). Eelda, et vastane teeb selle käigu, mida „meie” peame halvimaks (ja mida vastane peab parimaks). Võta halvima käigu hinnang vaadeldava seisu hinnanguks.
3. Parim seis on üks (suvaline) neist, millel on kõrgeim hinnang.

See on nn. minimaks-protseduur: ta sooritab otsingut kõikvõimalike mänguseisude puul, eeldades, et igal tasemel „meie” püüame maksimeerida oma võidu tõenäosust, aga vastane püüab seda minimeerida. Detailselt vt seda protseduuri edaspidi (p 19, otsing mänguseisude puul).

Kommentaarisid

See programm nõuab palju rohkem aega kui teised, kuna ta peab enne järjekordse käigu tegemist sooritama otsingu puul, mis esitab kõikvõimalikke käikude järjendeid. Aga ta on teistest oluliselt parem seetõttu, et teda saab laiendada keerulisematele mängudele kui see, mille jaoks eespool vaadeldud programmide (1, 2 ja 2') ammendava läbivaatamise lähenemisviis pole võimalik. Samuti saab teda täiendada, kasutades teadmisi konkreetse mängu kohta. Näiteks selle asemel, et vaadata kõiki võimalikke järgmisi käike, võib vaadata käikude mingit alamhulka, mis määratakse mingi lihtsa algoritmiga, ja selle asemel, et vaadelda käikude järjendeid kuni ühe mängija võiduni, võib otsida mingi fikseeritud sügavuseni ja arvutada mängulaua seisu väärtuse, kasutades teatavaid staatilisi kriteeriume.

Programm 3 on näide tehisintellekti tehnikast. Väga väikeste ülesannete jaoks on ta vähem efektiivne kui otsesed meetodid. Kuid teda saab kasutada sellistes situatsioonides, kus otsesed meetodid on kõlbmatud.

Kujutuvastus

Teine probleem, mida käsitleme, on käekirja tuvastamine. Vaatleme käsitsi kirjutatud suurtähe äratundmise probleemi ning esitame jälle programmide seeria, millest iga järgmine kasutab rohkem teadmisi ja on paremini laiendatav mittetriviaalsetele probleemidele kui eelmine.

Esmalt tuleb otsustada, kuidas esitada sisendit. Eeldame, et sullepea abil produtseeritud analoogsignaali on esitatud bitimaatriksina, kus elementideks on kas 1 (vastavas ruudus esineb tint) või 0 (tinti ei esine). Tähe äratundmise programmi väljundiks olgu üks 26 ladina tähestiku suurtähest, millele sisend kõige enam vastab.

Programm 1

Andmestruktuurid

SisendKujund – maatriks (nagu eespool kirjeldatud).

TuntudKujundid – tabel, milles iga tähe jaoks on üks ühik, nn võti. Võti arvutatakse järgmiselt: luuakse bitimaatriks, mis esitab tähte. Maatriksist võetakse kolm rida: üks alumisest veerandist, üks keskelt ja üks ülemisest veerandist. Valitud read ühendatakse üheks bitijärjendiks, mis saabki vastava tähe võtmeks. Veendutakse, et kahel tähel pole sama võtit. (Kui on, siis modifitseeritakse võtit loovat funktsiooni nii, et ta kasutaks teisi, erinevaid ridu.)

Algoritm

Et klassifitseerida sisendmaatriksit kui tähte, tee järgmist:

1. Võta sisendmaatriksi bitid samadest ridadest, mida kasutati tuntud kujundite tabeli moodustamisel, ja moodusta nende abil võti.
2. Tabelis selle võtme kohal olev täht ongi sisendsümbol.

Kommentaariid

See algoritm töötab hästi fikseeritud hulga kujundite korral, mis vastavad täpselt sisendile. Aga mitteideaalses maailmas tekivad järgmised probleemid.

- Sisendtähed peaksid vastama salvestatud kujunditele täpselt – muidu võidakse neid valesti klassifitseerida. Iga pisike plekk sisendkujundis moonutab võtit ja ühtlasi ka vastust.
- Alati, kui tuvastatavate sümboleid hulka on vaja lisada uusi sümboleid, võib olla vaja muuta kogu andmebaasi, sest uuel sümbolil võib olla sama võti mis mõnel olemasoleval.
- Kui sümboleid arv on suur, siis võib juhtuda, et ükskõik, kuidas me ka ei valiks bitimaatriksi ridu, on mitmel kujundil ikka sama võti.

Seega puudub meie esimesel programmil paindlikkus, mida ülesanne nõuab.

Programm 2

Andmestruktuurid

SisendKujund – maatriks nagu eelmises.

TuntudKujundid – N-mõõtmeliste vektorite hulk, iga tuvastatava tähe jaoks üks vektor. Vektorid moodustatakse nii, et jagatakse tähele vastav bitimaatriks N piirkonnaks ja igas piirkonnas esinevate 1-de arv võetakse maatriksi järjekordse komponendi väärtuseks. N peab olema piisavalt suur, et kõik kujundid oleksid esitatud erinevate vektoritega.

Algoritm

Et klassifitseerida sisendmaatriksit, tee järgmist:

1. Teisenda maatriks N-mõõtmeliseks vektoriks, nagu eespool kirjeldatud.
2. Arvuta N-mõõtmelises ruumis sisendvektori kaugus kõikide kujundite vektoritest.
3. Liigita sisend kui kujund, mis on talle kõige lähemal.

Kommentaariid

See programm on parem kui eelmine: ta ei ole nii tundlik andmebaasi võimalike muutuste suhtes ja – mis tähtsam – suudab korrektselt klassifitseerida ka selliseid sisendeid, mis mõnevõrra erinevad kujundeist, millele nad eeldatavasti vastavad. See on oluline, sest käsitsi kirjutatud tähed pole kunagi identsed. Samas on see programm aeglasem kui esimene.

Veel puudusi:

- Ei erista tähtsaid ja tähtsusetuid erinevusi, kui need on ühesuurused (joonis 4.1).



Joonis 4.1. Tähtis ja tähtsusetu erinevus

- Ei käsitle efektiivselt sama tähe variante, millel on erineva suhtelise suurusega komponendid (joonis 4.2).



Joonis 4.2. Sama tähe kaks varianti

- Suudab käsitleda sama tähe variante ainult siis, kui iga iseärasuste kombinatsiooni jaoks on andmebaasis olemas oma kujund. (Nt „I” kriipsukestega ülal ja all või ilma nendeta.)

Programm 3

Andmestruktuurid

SisendKujund – sama mis eelmistes.

TuntudKujundid – tähtede kirjelduste hulk, kus igaüks on esindatud kui binaarsete tunnuste konjunktsioon. Näiteks täht G võib olla esitatud järgmiselt (vt joonis 4.3):

Kaar(a,b) & Kohal(a,b) & Lõik(b,c) & Vasakul(c,b) & (NIL \vee (Lõik(b,d) & All(d,b))) & (NIL \vee (Lõik(a,e) & All(e,a)))



Joonis 4.3. Täht G

See kirjeldus ütleb, et G sisaldab ühe kaare ja ühe lõigu, mis esineb kaare alumises otsas ja suundub vasakule. Samuti *võib* tal olla (aga ei pruugi – NIL) teine lõik, mis tõuseb üles kaare ülemisest tipust. Tal *võib* olla veel üks lõik, mis suundub kaare alumisest otsast alla.

Algoritm

Klassifitseerimaks sisendvektorit, tee järgmist:

1. Leia bitimaatriksis kaarte ja lõikude esinemised ja konstrueeri sisendtähe komponentide nimekiri. (Seega Kaar, Vasakul jne tuleb defineerida maatriksi termineis.)
2. Võrdle seda kirjeldust iga salvestatud kujundiga ja leia selline, mis kõige paremini vastab.

Kommentaarid

See programm on oluliselt paindlikum kui eelmised. Ta suudab käsitleda tähtedes suuruse suhtelisi erinevusi. Ta lubab tähtedes erinevaid tunnuste kombinatsioone. Selle programmi jaoks on lihtsam lisada andmebaasi uusi tähti. Võib-olla on selleks vaja modifitseerida kasutatavat tunnuste kombinatsiooni, kuid sellised modifikatsioonid on inimesele arusaadavamad, kuna kirjeldused antakse nüüd loomulikumates terminites (inimesed mõtlevad tähtedest just sellisel viisil).

Järeldused

Me käsitlesime kahte programmide seeriat. Kummaski seerias esindab viimane programm seda, mida me mõtleme tehisintellekti tehnika all. Need kaks programmi on aeglasemad kui eelmised, kuid nad illustreerivad kolme tähtsat tehisintellekti tehnikat: otsing (*search*),

objektide struktuuri iseloomustavate teadmiste kasutamine ning abstraktsioon, eristamaks tähtsaid ja tähtsusetuid iseärasusi.

Raskete probleemide lahendamisel on neid tehnikaid rakendavatel programmidel eelised: nad on vähem tundlikud ega jookse sisendi väikestest häiretest ummikusse. Inimesed suudavad kergesti mõista teadmisi, mida programm kasutab. Sellised tehnikad toimivad ka suurte probleemide korral, kus otsesed meetodid pole rakendatavad.

5. Probleemid ja probleemilahendus

Probleemilahendus (*problem solving*) on sellise operatsioonide või toimingute järjendi määramine, mis võib viia soovitud sihini.²³

Probleemid, millega tehisintellekt tegeleb, on näiteks teoreemide automaatne tõestamine, meditsiiniline ja tehniline diagnostika (ekspertsüsteemides), loomuliku keele mõistmine, samuti peamurdmisesannete lahendamine, mängude mängimine jpm.

Probleemilahendusel on vaja täita kolm etappi:

1. Probleemi kirjeldamine.
2. Probleemi analüüs.
3. Tehnika valik ja selle rakendamine probleemi lahendamisel.

Probleemi kirjeldamine

Sageli kujutab probleemilahendus endast protsessi, mille puhul alustatakse mingist algolekust ja sooritatakse otsing probleemiruumis e olekute ruumis (*state space*), et leida soovitud sihti. Edukas probleemilahendus eeldab algoleku, antud sihile vastava aktsepteeritava tulemuse ning probleemiruumi määratlevate elementide või tegevuste teadmist.

Seega on vaja:

- a) defineerida olekute ruum,
- b) määrata alg- ja lõppolek,
- c) kirjeldada tegevused, mille sooritamise tulemusel saab liikuda probleemiruumis ühest olekust teise. Sageli esitatakse tegevused kui-siis reeglina e produktsioonidena.

Näide 5.1. Malemäng.

Malelaua võib esitada kui 8×8 maatriksi. Võimalikke lauaseise on ligikaudu 10^{120} , need moodustavad olekute ruumi. Algolek on malemängus kindlaksmääratud lauaseis, kus kõik mustad ja valged malendid on oma ettenähtud kohtadel. Lõppolekuid on rohkem kui üks, nt kui üks mängijatest ei saa teha käiku ja tema kuningas on tules. Tegevused määravad lubatavad käigud. Näiteks võib esitada tegevuse reeglina ($i=1, \dots, 8$):

kui

valge ettur ruudus $(i,2)$ ja ruut $(i,3)$ on tühi ja ruut $(i,4)$ on tühi,

²³ IT terministandardi sõnastik <http://www.keeleeveeb.ee/> (18.02.2010)

siis

tõsta ettur ruudust (i,2) ruutu (i,4).

Olekute ruumis algolekust lõppolekusse viiva tee leidmiseks kasutatakse otsingut, rakendades ühest olekust teise liikumiseks võimalikke tegevusi.

Näide 5.2. Veekannuülesanne.

Antud on kaks kannu, mahuga vastavalt 4 l ja 3 l. Kanne võib täita veega, samuti valada vett ühest kannust teise või maha. Kuidas saavutada, et suuremas kannus oleks täpselt 2 l vett?

Sellegi probleemi võime püstitada kui otsinguprobleemi olekute ruumis, Olekuid tähistame (x,y) , kus $x=0,1,2,3,4$; $y=0,1,2,3$. Algolek on $(0,0)$ ja lõppolekud $(2,y)$.

Reeglid (produksioonid) vastavad lubatavatele tegevustele (tabel 5.1).

Tabel 5.1. Reeglid veekannuülesandes

| Jrk nr | Eeltingimus | Reegel | Kirjeldus |
|--------|---------------------|---------------------------------|--|
| 1 | $x < 4$ | $(x,y) \rightarrow (4,y)$ | Täida 4-liitrine kann |
| 2 | $y < 3$ | $(x,y) \rightarrow (x,3)$ | Täida 3-liitrine kann |
| 3 | $x > 0$ | $(x,y) \rightarrow (0,y)$ | Vala kogu vesi 4-liitrisest kannust maha |
| 4 | $y > 0$ | $(x,y) \rightarrow (x,0)$ | Vala kogu vesi 3-liitrisest kannust maha |
| 5 | $x+y \geq 4, y > 0$ | $(x,y) \rightarrow (4,y-(4-x))$ | Vala vett 3-liitrisest kannust 4-liitrisesse, kuni see on täis |
| 6 | $x+y \geq 3, x > 0$ | $(x,y) \rightarrow (x-(3-y),3)$ | Vala vett 4-liitrisest kannust 3-liitrisesse, kuni see on täis |
| 7 | $x+y \leq 4, y > 0$ | $(x,y) \rightarrow (x+y,0)$ | Vala kogu vesi 3-liitrisest kannust 4-liitrisesse |
| 8 | $x+y \leq 3, x > 0$ | $(x,y) \rightarrow (0,x+y)$ | Vala kogu vesi 4-liitrisest kannust 3-liitrisesse |

Enamus praegu kasutatavatest ekspertüsteemidest on reeglipõhised ja lahendavad probleeme, sooritades *otsingut olekute ruumis*.

Probleemi analüüs

Otsing on väga üldine meetod, mida saab rakendada paljudele ülesannetele. Aga ta hõlmab ka spetsiifilisemaid tehnikaid, millest igatüki on efektiivne teatava ülesannete klassi jaoks. Et valida sobivat tehnikat, tuleb probleemi analüüsida.

1) Kas probleemi saab osadeks lahutada?

Selle tunnuse kohaselt saab probleemid jaotada kahte klassi: (sõltumatuteks) osadeks lahutatavad (*decomposable*) ja osadeks mittelahutatavad (*nondecomposable*) probleemid.

2) Kas lahendussamme saab ignoreerida, parandada?

Selle tunnuse kohaselt jagunevad probleemid kolmeks.

- Probleemid, kus lahendussammud on ignoreeritavad (*ignorable*). Nt teoreemi tõestamisel võime mittevajaliku lemma tõestamist kogu tõestuse seisukohalt lihtsalt ignoreerida.
- Probleemid, kus lahendussammud on parandatavad (*recoverable*). Nt 8-mängus, mida mängitakse 3x3 mängulaual ja kus eesmärgiks on paigutada ruutudes paiknevad numbrid 1 kuni 8 ettenähtud järjekorda, kasutades ainult numbrini nihutamist kõrvalasuvasse tühja ruutu, saab käike tagasi võtta.
- Probleemid, kus lahendussammud ei ole ignoreeritavad ega parandatavad (*unrecoverable*). Selline on nt malemäng.

3) Kas lahend on prognoositav?

Selle tunnuse kohaselt jagunevad probleemid kaheks:

- kindla tulemusega probleemid (*certain-outcome*), nt 8-mäng, kus on vaja saavutada kindel lõppseis mängulaual,
- ebamäärase tulemusega probleemid (*uncertain-outcome*), nt bridž.

4) Kas on vaja leida parim lahend?

See tunnus võimaldab probleemid jaotada kaheks:

- mingi tee (*any-path*) probleemid, kus piisab, kui leitakse ükskõik missugune tee algolekust lõppolekusse (nt teoreemi tõestamine),
- parima tee (*best-path*) probleemid (nt rändkaupmehe ülesanne, kus on vaja leida lühim tee kõikide linnade läbimiseks, vt näide 5.3).

5) Kas lahendamine nõuab suhtlemist inimesega?

Selle tunnuse kohaselt jagunevad probleemid jälle kaheks:

- isoleeritud (*solitary*) probleemid, kus inimese sekkumine ei ole vajalik (nt teoreemi automaatne tõestamine),
- konversatsioonilised (*conversational*) probleemid, kus on vajalik suhtlus inimesega, nt ekspertsüsteemides lahendatavad probleemid.

Tehnika valik ja selle rakendamine probleemi lahendamiseks

Kui probleem on defineeritud otsinguprobleemina olekute ruumis, siis üldine tehnika probleemi lahendamiseks on otsing. Otsingu juhtimiseks võib rakendada erinevaid juhtimisstrateegiaid (*control strategy*). Vajadus juhtimise järele tekib siis, kui järjekordsest olekust on mitu võimalikku jätku.

Juhtimisstrateegiaid esitatakse kaks olulist nõuet:

- mobiilsus,
- süstemaatilisus.

Kui me näiteks veekannu ülesandes proovime rakendada sellist juhtimisstrateegiat, mis valib tabelist 5.1 alati kõige esimese sobiva reegli, siis jäämegi suuremat kannu veega täitma ja seejärel vett maha valama – see strateegia ei ole mobiilne, s.t ei vii meid lõppolekusse. Sama ülesande lahendamisel võime valida reegleid täiesti juhuslikult ja niiviisi küll varem või hiljem jõuda lõppolekusse, kuid selline strateegia ei ole süstemaatiline.

Kaks lihtsat näidet süstemaatiliste juhtimisstrateegiate kohta on **laiutiotsing** (*breadth-first search*) ja **süvitsiotsing** e sügavutiotsing (*depth-first search*).

Joonistel 5.1 ja 5.2 on esitatud vastavalt laiuti- ja süvitsiotsingu algoritmid. Eeldatakse, et olekute ruum on esitatav puuna. Mõlemad algoritmid kasutavad nimestikku (L), kuhu järjest paigutatakse läbimisele tulevaid tippe.

- 1) Paiguta algolekule vastav tipp nimestikku L. Kui algolek on ühtlasi lõppolek, siis on lahend leitud, lõpeta.
- 2) Kui L on tühi, siis lahendit ei leidu, lõpeta.
- 3) Eemalda esimene tipp (n) nimestikust L.
- 4) Leia tipu n kõik vahetud järglased. Kui tipul n pole järglasi, siis mine 2.
- 5) Paiguta tipu n kõik vahetud järglased nimestiku L **lõppu**.
- 6) Kui mõni tipu n järglastest vastab lõppolekule, siis on lahend leitud, lõpeta. Muidu mine 2.

Joonis 5.1. *Laiutiotsing* olekute puul

- 1) Paiguta algolekule vastav tipp nimestikku L. Kui algolek on ühtlasi lõppolek, siis on lahend leitud, lõpeta.
- 2) Kui L on tühi, siis lahendit ei leidu, lõpeta.
- 3) Eemalda esimene tipp (n) nimestikust L.
- 4) Leia tipu n kõik vahetud järglased. Kui tipul n pole järglasi, siis mine 2.
- 5) Paiguta tipu n kõik vahetud järglased nimestiku L **algusesse**.
- 6) Kui mõni tipu n järglastest vastab lõppolekule, siis on lahend leitud, lõpeta. Muidu mine 2.

Joonis 5.2. *Süvitsiotsing* olekute puul

Laiuti- ja süvitsiotsing kujutavad endast mõlemad nn **pimedat otsingut** e pimeotsingut (*blind search*).

Näide 5.3. Rändkaupmehe probleem (*The Traveling Salesman Problem*). Rändkaupmees peab läbima n linna, külastades igaühte üksainus kord. Iga linnade paar on ühendatud maanteega, mille pikkus on teada. Leida lühim teekond, mis algaks ja lõpeks samas linnas.

Rändkaupmehe probleem on tähtis kas või seetõttu, et see on klassikaline ülesanne, millega tuleb kokku puutuda nt telekommunikatsioonisüsteemides signaalide suunamisel.

Lihtne mobiilne juhtimisstrateegia selle probleemi lahendamiseks oleks vaadata läbi kõikvõimalikud teekonnad ja valida neist lühim. Aga n linna puhul on $n!$ teekonda. Ühe teekonna kontrollimine võtab aja, mis on võrdeline linnade arvuga n . Seega kogu kontrollimiseks kuluv aeg on võrdeline arvuga $n!$ (nt 10 linna puhul on vaja kontrollida $10! = 3\,628\,800$ teekonda).

Parem juhtimisstrateegia oleks nn **harude ja tõkete meetod** (*branch and bound*): genereerides täielikke teekondi, säilitada kogu aeg ainult lühim. Kui parajasti genereeritava teekonna pikkus juba ületab siiani genereeritud lühimat, siis katkestada selle genereerimine. See tehnika garanteerib lühima teekonna leidmise ja on efektiivsem kui eelmine, kuid ometi on selgi puhul ajavajadus eksponentsiaalne.

Raske ülesande lahendamisel on sageli vaja leida kompromiss juhtimisstrateegia mobiilsuse ja süstemaatilisuse vahel ning konstrueerida selline juhtimisstrateegia, mis ei leia tingimata parimat lahendit, kuid enamasti leiab väga hea lahendi. Selleks tuuakse sisse **heuristiku** mõiste ja sooritatakse nn **heuristilist otsingut** (*heuristic search*). Heuristiline otsing on kogemusel ja arvamustel põhinev probleemilahendus, mida kasutatakse otsingu tõhustamiseks, **taotlede vastuvõetavaid tulemusi, kuid garanteerimata edu**.²⁴

Heuristikuid on kahte liiki – üldised ja ainevallaspetsiifilised. Üks üldine heuristik on nt *lähima naabri algoritm*. Rändkaupmehe ülesande puhul võib lähima naabri algoritmi sõnastada järgmiselt.

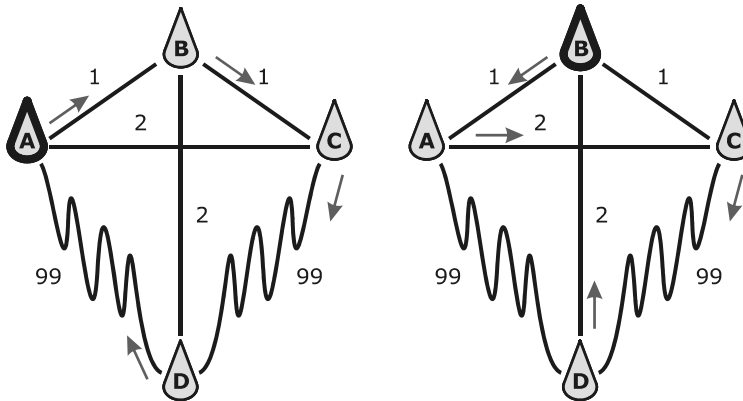
- 1) Vali suvaliselt lähtelinn.
- 2) Vaata läbi kõik linnad, mida pole veel külastatud, ning vali neist lähim.
- 3) Korda sammu 2, kuni kõik linnad on läbitud.

Algoritmi ajahinnang n linna korral on $(n-1)+(n-2)+\dots+(n-(n-2)) = ((n-1)+(n-(n-2))) \times (n-2)/2 = (n+1)(n-2)/2 \approx n^2$.

Heuristiline otsing võimaldab lahendada eksponentsiaalse keerukusega ülesandeid. Ehkki heuristikute abil saadud lahendid võivad mõnikord olla halvad (vt joonis 5.3), esinevad halvimal juhul praktikas siiski harva. Sageli polegi vaja parimat lahendit, vaid piisab lihtsalt heast lahendist. Ei taotle ju inimesedki alati ilmtingimata parimat.

²⁴ IT terministandardi sõnastik <http://www.keeleeveeb.ee/> (18.02.2010)

| | A | B | C | D |
|---|----|---|----|----|
| A | | 1 | 2 | 99 |
| B | 1 | | 1 | 2 |
| C | 2 | 1 | | 99 |
| D | 99 | 2 | 99 | |



Joonis 5.3. Rändkaupmehe probleem: tabelis on antud linnade A, B, C ja D vahelised kaugused. Lähima naabri algoritmi rakendamise tulemus sõltub lähtelinna valikust. Nt alustades linnast A või B saame erineva pikkusega teekonnad

Suure hulga probleemilahendusel kasulikke heuristikuid võib leida G. Pólya raamatust²⁵ (Pólya 2001).

²⁵ Vt ka <http://www.math.utah.edu/~pa/math/polya.html> (18.02.2010)

II. TEADMUSE ESITUS

6. Teadmuse mõiste. Teadmusbbaasi loomine

Seni ei ole veel üldtunnustatud arusaama selle kohta, kuidas täpselt defineerida teadmust (ehk teadmisi, *knowledge*) ja mille poolest teadmised erinevad andmetest (*data*). Sageli kasutatakse neid mõisteid sünonüümidena. Samas on tehtud katseid formaalselt määratleda teadmuse mõistet, näiteks nimetatakse teadmuseks andmeid, millel on modelleeritavas ainevaldkonnas kindel tähendus, või teatud viisil üldistatud andmeid (andmeid andmete kohta ehk metaandmeid).

Andmed on faktid, mis iseloomustavad ainevaldkonna objekte, protsesse ning nähtusi ja nende omadusi: mõõtmis- ja vaatlustulemused, andmed infokandjatel (tabelid, protokollid, teatmikud), andmemudelid (andmestruktuurid) diagrammide, graafikute, funktsioonide kujul, andmed arvutis andmekirjelduskeeles, andmebaasid.

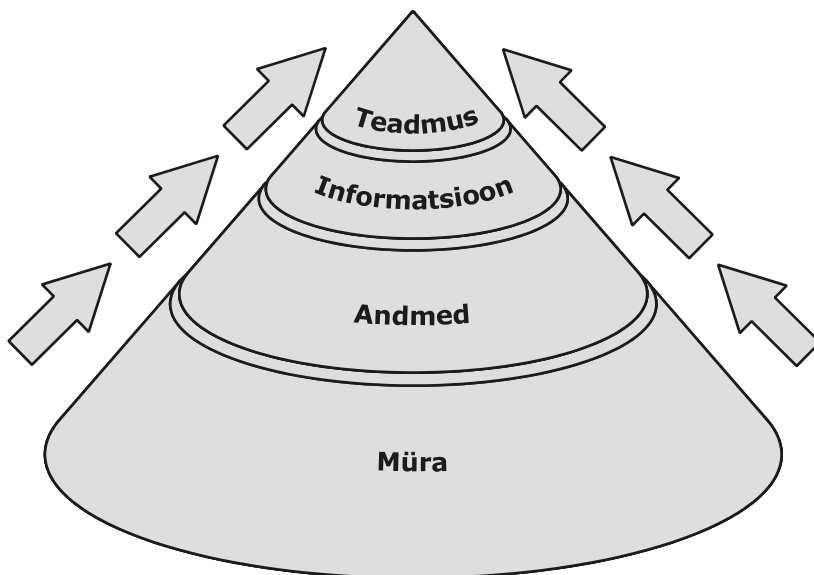
Teadmus on ainevaldkonna seaduspärasused (põhimõtted, seosed, seadused), mis on saadud praktilise tegevuse ja ametialase kogemuse tulemusel ning võimaldavad spetsialistidel püstitada ja lahendada selle ainevaldkonna ülesandeid. **Teadmus on hästi struktureeritud andmed** ehk **andmed andmete kohta** ehk **metaandmed** (Gavrilova ja Choroshevsky 2000): teadmised inimese mälus kui mõtlemise tulemus, teadmised materiaalsel kandjatel (õpikud, metoodilised materjalid), teadmuseväli (ainevaldkonna põhiliste objektide, nende atribuutide ja neid ühendavate seaduspärasuste kirjeldus), teadmised, mida kirjeldatakse teadmuse esituse formaalses keeles (nt produktsioonilised keeled, semantilised võrgud, freimid jne), teadmusbbaas.

Tehakse vahet pind- ja süvateadmuse vahel. **Pindteadmus** on teadmised ilmutatud, nähtavatest seostest ainevalla üksikute sündmuste ja faktide vahel (nt *kui vajutada uksekella nuppu, siis kostab helin; kui pea valutab, siis tuleb võtta aspiriini*).

Süvateadmus seevastu on abstraktsioonid, analoogiad, skeemid, mis väljendavad ainevallas toimuvate protsesside struktuuri ja olemust (nt *uksekella ja juhtmete elektriline skeem; füsioloogide ja arstide teadmised peavalu põhjustest ja liikidest ning ravimise meetoditest*). Need teadmised seletavad nähtusi ja neid võib kasutada ainevaldkonna objektide funktsioneerimise ennustamiseks.

Praegused ekspertsüsteemid töötavad põhiliselt pindteadmusega, sest süvateadmuse väljaselgitamiseks ja töötlemiseks pole veel loodud universaalseid meetodeid.

Teadmust võib mõista ka kui teatava hierarhia osa (Giarratano ja Riley 2004). Kõige alumisel tasemel (joonis 6.1) on müra – ebahuvitavad üksused (mis samas võivad varjata andmeid). Järgmine tase on (toor)andmed – üksused, mis võivad (kuid ei tarvitse) pakkuda huvi. Veel kõrgemal tasemel on informatsioon – töödeldud andmed. Seejärel tulevad teadmised – tähtis informatsioon, mida tuleb säilitada, et seda saaks edaspidi kasutada.



Joonis 6.1. Teadmuse püramiid

Järgnev näide (Giarratano ja Riley 2004) illustreerib teadmuse püramiidi. Olgu antud numbrijärjend

137585766832525156430015.

Kui meil pole teadmisi, siis võime seda järjendit pidada lihtsalt müräks. Kui aga võib oletada, et see järjend tähendab midagi, siis on need hoopis andmed. Andmetest saab informatsiooni. Näiteks rakendame järgmist algoritmi.

- Rühmitada numbrid 2-kaupa.
- Ignoreerida 2-kohalisi arve, mis on väiksemad kui 32.
- Tõlgendada allesjäänud 2-kohalisi arve kui ASCII-koode ja leida vastavad sümbolid.

Saame informatsiooni: KULD 438+

Sellele informatsioonile rakendame teadmisi, mis võivad olla antud näiteks järgmise reeglina:

kui KULD < 500 ja hind kasvab (+), **siis** osta kulda.

Kuna 438 < 500, siis saame teha järelduse (tuletada uue teadmise), et tuleb osta kulda.

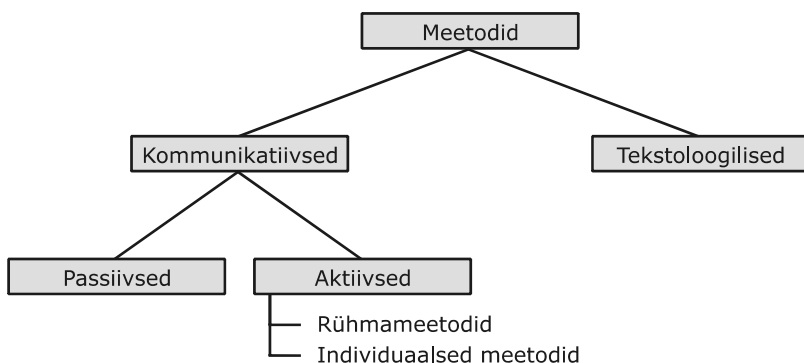
Teadmuspõhise tehisintellektisüsteemi väljatöötamine on pikk protsess, mis koosneb mitmest etapist. Seda protsessi viib läbi töökollektiiv, kuhu kuuluvad (Gavrilova ja Choroshevsky 2000)

- teadmusinsener – isik, kes hangib teadmust ainevalla asjatundjatelt ja teistest teadmusallikatest ning organiseerib selle teadmusbaasina,
- ainevaldkonna asjatundjad – eksperdid,
- programmeerijad,
- kasutajad.

Tähtsamad etapid on järgmised.

- 1) Probleemi valik (siia kuuluvad ka vastava ainevalla ekspertide leidmine, lahendusmeetodite määramine, kasu ja kahju hindamine, tööplaani koostamine).
- 2) Prototüüpsüsteemi väljatöötamine, mille võib omakorda jaotada mitmeks alaetapiks.
 - Probleemi konkretiseerimine: määratakse ressursid, teadmusallikad, leitakse loodava süsteemi olemasolevad analoogid, püstitatakse eesmärgid, määratletakse lahendatavate ülesannete klassid.
 - Ainevaldkonna teadmiste hankimine.
 - Hangitud teadmiste struktureerimine: täpsustatakse terminoloogia, põhimõisted, seosed nende vahel, määratakse sisend- ja väljundinfo struktuur, otsuste tegemise strateegia.
 - Formaliseerimine (teadmiste esitamine eelnevalt kindlaksmääratud teadmuse esituse keeles).
 - Realiseerimine – programmeerimine kas traditsioonilises või spetsialiseeritud programmeerimiskeeles või ekspertsüsteemi „kesta” (tühja ekspertsüsteemi) sisustamine antud ainevalla teadmistega.
 - Testimine.
- 3) Eeltööstusliku süsteemi väljaarendamine. Esmalt luuakse demonstratsiooniprototüüp, siis toimiv prototüüp, seejärel tööstuslik süsteem ja lõpuks kommertssüsteem.
- 4) Hindamine.
 - Kasutajad hindavad süsteemi arusaadavuse, läbipaistvuse, liidese mugavuse seisukohalt.
 - Kutsutud ekspert annab hinnangu süsteemi vastuvõetud otsustele ja seletustele.
 - Töökollektiiv ise hindab realisatsiooni efektiivsust, tööviljakust, reaktsiooniga, disaini, ainevaldkonna katvust, teadmusbaasi mittevastuolulisust, ummikseisude hulka, tõrkekindlust.
- 5) Süsteemi ühendamine teiste samas ainevaldkonnas kasutatavate programmidega, inimeste väljaõpetamine tööks uue süsteemiga.
- 6) Toetamine – ümberprogrammeerimine süsteemi kiiruse ja tõrkekindluse tõstmiseks.

Tehisintellektisüsteemi väljatöötamise oluline osa on *teadmuse hankimine*. Selleks leidub mitmesuguseid meetodeid (vt joonis 6.2). Kommunikatiivsed meetodid eeldavad suhtlemist ekspertidega, tekstoloogilised meetodid põhinevad varem olemasolevate tekstide (õpikute, monograafiate, dokumentide) läbitöötamisel. Kommunikatiivsed meetodid võivad omakorda olla kas passiivsed, kus teadmusinsener lihtsalt jälgib eksperdi tegevust (palub eksperti „valjusti mõelda”, vaatab pealt tema tegevust, kuulab loengut), või aktiivsed, kus eeldatakse koostööd. Viimasel juhul võib omakorda eristada rühma-meetodeid (ajurünnak, ümarlaud, rollimängud) ja individuaalseid meetodeid (anketeerimine, dialoog ekspordiga, intervjuu).



Joonis 6.2. Teadmuse hankimise meetodid

Teadmuse esitus (knowledge representation) on teadmuse kodeerimise ja teadmusbaasis salvestamise protsess või tulemus.²⁶ Kasutusel on kümneid erinevaid teadmuse esituse formalisme: kui-siis reeglid, semantilised võrgud, freimid, skriptid, kontseptuaalsed graafid, predikaatloogika, tehisnärvivõrgud jne. On loodud ka spetsiaalseid teadmuse esituse keeli, nt FRL (1978) ja KL-ONE (1985).

Teadmuse esituse rolli tehisintellektis on nähtud selles, et taandada aruka tegutsemise probleemid otsinguprobleemideks, vt joonis 6.3 (Ginsberg 1993).

| "Tavalise" probleemi lahendamine | Intellektuaalse probleemi lahendamine |
|--|---|
| 1. mõtelda välja algoritm 2. valida programmeerimiskeel 3. esitada algoritm selles keeles 4. käivitada programm, et lahendada probleemi | 1. määrata, milliseid teadmisi on vaja 2. valida teadmuse esituse keel 3. esitada teadmus selles keeles 4. teha järeldusi teadmistest (sooritada otsing), et lahendada probleemi |

Joonis 6.3. Probleemi lahendamine arvutil

7. Produktsiooniline teadmuse esitus

1943. a defineeris Emil Post (1897–1954) produktsiooni $A \Rightarrow B$ kui reegli, mis asendab sisendsõnes osasõne A osasõnega B.

Hiljem kasutati produktsioone algoritmi mõiste täpsustamiseks. **Markovi normaalalgoritm** (1954) on defineeritud kui produktsioonide järjend, kus sisendsõnele rakendatakse esmalt kõige esimest sobivat produktsiooni (s.t asendatakse produktsiooni vasaku poole esimene esinemine parema poolega), seejärel saadud sõnele jälle kõige esimest sobivat jne. Algoritmi rakendamine lõpeb, kui ei leidu enam ühtegi sobivat produktsiooni.

²⁶ IT terministandardi sõnastik <http://www.keeleeveeb.ee/> (18.02.2010)

Produksiooni mõistet kasutavad mõned programmeerimiskeeled, nt Refal²⁷ ja PROLOG.

Tehisintellektisüsteemides hakati produktsioone teadmuse esituseks kasutama 1970. aastatel. Produktsioonid e kui-siis reeglid („kui A, siis B”) määravad, mida saab millest järeldada. Produktsiooni vasak pool A võib olla nt tingimus ja parem pool B nt tegevus, mis sooritatakse, kui tingimus on täidetud. Praegused ekspertsüsteemid on enamasti reeglipõhised.

On kahte liiki reeglipõhiseid süsteeme, sõltuvalt sellest, kas järeldamine toimub päripidi või tagurpidi (*forward/backward-chaining*). Esimesed lähtuvad teadaolevatest faktidest ja rakendavad reegleid päripidi (seda nim ka tingimusest lähtuvaks e otsetuletuseks). Teised lähtuvad mingist hüpoteesist ja püüavad seda tõestada, rakendades reegleid tagurpidi (seda nim ka tegevusest lähtuvaks e pöördtuletuseks). Näited 7.1 ja 7.2 illustreerivad otse- ja pöördtuletust.

Näide 7.1. Teadaolevad faktid: {A, F}.

Produktsioonid:

1. $A \& B \& C \rightarrow D$
2. $D \& F \rightarrow G$
3. $A \& J \rightarrow G$
4. $B \rightarrow C$
5. $F \rightarrow B$
6. $L \rightarrow J$
7. $G \rightarrow H$

Kas kehtib H?

- Otsetuletus: $F \Rightarrow B \Rightarrow C \text{ (A, B)} \Rightarrow D \text{ (F)} \Rightarrow G \Rightarrow H$.
- Pöördtuletus: $H \Rightarrow G \Rightarrow D \Rightarrow B, C \text{ (A)}$. (B kehtib, sest F kehtib; C kehtib, sest B kehtib.)

Näide 7.2. Olgu ekspertsüsteemis faktid:

„inimene on aktiivne”,

„inimene armastab päikest”

ja produktsioonid:

R1. Kui „inimene puhkab suvel” ja „inimene on aktiivne”, siis „inimene sõidab mägedesse”.

R2. Kui „inimene armastab päikest”, siis „inimene puhkab suvel”.

Küsimus: kuidas veeta puhkus?

- **Otsetuletus** (faktidest lähtudes jõuda nõuandeni, mille ekspertsüsteem väljastab):

I tsükkel

R1 ei saa rakendada,

R2 rakendub, andmebaasi lisandub fakt „inimene puhkab suvel”.

²⁷ <http://www.refal.net/> (18.02.2010)

II tsükkel

R1 rakendub, aktiveerub eesmärk „*inimene sõidab mägedesse*” (see on nõuanne, mille ekspertsüsteem kasutajale väljastab).

• **Pöördtuletus** (kinnitada valitud eesmärki olemasolevate reeglite ja faktide alusel):
Eesmärk – „inimene sõidab mägedesse”.

I tsükkel

R1 ei rakendu (pole eeldust „inimene puhkab suvel”). Uus eesmärk on „inimene puhkab suvel”.

Reegel R2 rakendub, kinnitades seda eesmärki.

II tsükkel

R1 rakendub – kinnitab püstitatud eesmärki („inimene sõidab mägedesse”).

Reegli põhise tehisintellektisüsteemi teadmusbaasi koosseisu kuulub ka tuletusmehhanism (reeglite interpretaator), millel on kaks funktsiooni:

1. leida olemasolevaid fakte töömälust (andmebaasist) ja reegleid teadmusbaasist ning võimaluse korral lisada töömälusse uusi fakte,
2. määrata reeglite läbivaatamise ja rakendamise järjekord.

Tuletusmehhanism töötab **tsüklliselt**.

1. Vaadata läbi kõik reeglid, et leida sellised, mille eeldused langevad ühte antud momendil teadaolevate faktidega töömälust.
2. Valida reegel, rakendada seda reeglit, tema järelendus kanda töömällu.
3. Korrata sama tsükli.

Ühes tsükli saab rakendada üksainus reegel. Kui leidub mitu faktidega sobivat reeglit, siis valitakse mingi kindla kriteeriumi alusel üks, mida selles tsükli rakendatakse.

Enamasti koosneb tuletusmehhanism kahest komponendist:

1. *tuletuskomponent* realiseerib tuletust,
2. *juhtimiskomponent* juhib seda protsessi.

Tuletuskomponent rakendab reeglit *modus ponens*:

kui on teada, et A on tõene **ja** leidub reegel kujul $A \rightarrow B$, **siis** on B samuti tõene.

Juhtimiskomponent määrab reeglite rakendamise järjekorra. Juhtimiskomponendil on neli funktsiooni.

1. Kõrvutamine – reeglit kõrvutatakse olemasolevate faktidega.
2. Valik – kui korraga on rakendatavad mitu reeglit, siis valitakse nende hulgast üks (sobivaim). See on nn **konflikti lahendamine**.
3. Rakendamine – kui kõrvutamisel osutus reegel ühtelangevaks mingite faktidega töömälust, siis reegel rakendub.
4. Tegevus – muudetakse töömälu: rakendunud reegli järeldused lisatakse töömällu.

Tuletuse juhtimise strateegiad määravad reeglite rakendamise järjekorra, valikuprotseduur määrab otsingu suuna ja otsingu realiseerimise viisi. Valikuprotseduurid kuuluvad tavaliselt tuletusmehhanismi koosseisu.

Strateegia valikul tuleb määrata:

1. milline punkt olekute ruumis võtta alguspunktiks (s.t kas kasutada otse- või pöördtuletust),
2. kuidas tõsta otsingu efektiivsust.

Mõned juhtimisstrateegiad:

- 1) raamatuhunniku printsiip: kõige sagedamini kasutatav tootmine on kõige kasulikum, seega peab olema kõige kättesaadavam („raamatuhunnikus” pealmine);
- 2) pikima tingimuse printsiip: erijuhtude reeglid on tähtsamad kui üldised, sest arvestavad rohkem informatsiooni;
- 3) metaproduktioonide printsiip: produktioonid klassifitseeritakse ja iga klassi esindab üks nn metaproduktioon;
- 4) klassitahvli printsiip: produktioonid suhtlevad vahemälu („klassitahvli”) kaudu, jättes sinna üksikute teateid ja kustutades läbiloetud teateid;
- 5) prioriteetse valiku printsiip: produktioonidele omistatakse eelnevad kaalud ja mitme võimaluse olemasolu korral valitakse suurima kaaluga produktioon;
- 6) juhtimine nimede kaudu: produktioonidele omistatakse eelnevalt nimed ja produktioon valitakse nime järgi.

Näide 7.3. Puuris on ahv. Puuris asub kast ja puuri keskel laes ripub kimp banaane. Puuri põrandal seisest ei ulatu ahv banaanideni, kuid ulatub nendeni, kui seisab puuri keskel kasti otsas. Ahv oskab puuris liikuda, kasti nihutada, kastile ronida, banaane haarata. Mida peab ahv tegema selleks, et kätte saada banaane?

Selle probleemi võime püstitada kui otsinguprobleemi olekute ruumis. Oletame, et puuri põrand (ja lagi) on tükeldatud ruudukesteks a, b, c jne (c vastab puuri keskpunktile, joonis 7.1) ning ahv saab mööda põrandat liikuda ruudust ruutu. Siis on iga olek esitatav kui vektor (w, x, y, z) ,

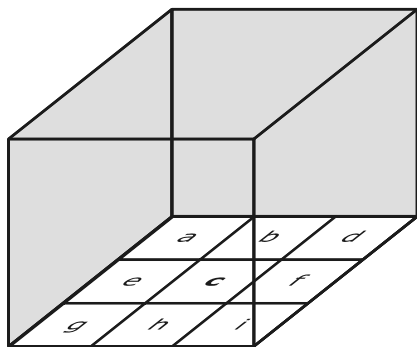
kus

w – ahvi asukoht puuris (a, b, c jne),

x – kas ahv on kasti otsas (0/1),

y – kasti asukoht (a, b, c jne),

z – kas ahvil on banaanid käes (0/1).



Joonis 7.1. Puuri põrand

Algolek on $(a, 0, b, 0)$ ja lõppolek $(c, 1, c, 1)$. Ahvi võimalikud tegevused saame esitada produktsioonidena (tabel 7.1).

Tabel 7.1. Ahvi tegevused

| <i>Tegevus</i> | <i>Produktsioon</i> |
|----------------|---|
| Minna(w, u) | $(w, 0, y, z) \rightarrow (u, 0, y, z)$ |
| Nihutada(y,v) | $(y, 0, y, z) \rightarrow (v, 0, v, z)$ |
| Ronida(w) | $(w, 0, w, z) \rightarrow (w, 1, w, z)$ |
| Haarata | $(c, 1, c, 0) \rightarrow (c, 1, c, 1)$ |

Probleemi lahendamine seisneb sellise produktsioonide järjendi leidmises, mis võimaldavad algolekust jõuda lõppolekusse. Selle järjendi leidmiseks võib sooritada näiteks laiutiotsingu olekute ruumis.

Näide 7.4. Kolm misjonäri ja kolm kannibali tahavad pääseda jõe vasakult kaldalt paremale. Neil on paat, mis mahutab ainult kaks isikut. Kui kannibalide arv kaldal ületab misjonäride arvu, siis süüakse viimased ära. Kuidas korraldada vedusid, et kõik pääseksid õnnelikult üle jõe?

Sellegi probleemi võime püstitada kui otsinguprobleemi olekute ruumis ja teadmuse esitada produktsioonidena. Olgu olek esitatud kui vektor (M_v, K_v, P) , kus

M_v – misjonäride arv vasakul kaldal,

K_v – kannibalide arv vasakul kaldal,

P – kas paat asub vasakul kaldal (1/0).

Algolek on siis $(3,3,1)$ ja lõppolek $(0,0,0)$.

Produktsioonid esitame üldistatud kujul, andes lisatingimused, millal produktsioon rakendub. Toome sisse lisatähistuse: M_p, K_p – vastavalt kas misjonäride või kannibalide arv paadis.

Vasakult kaldalt paremale sõitmine:

$\forall (M_p, K_p)$, kus $1 \leq M_p + K_p \leq 2$:

$(M_v, K_v, 1) \rightarrow (M_v - M_p, K_v - K_p, 0)$,

kui

$(M_v - M_p = 0) \vee (M_v - M_p \geq K_v - K_p)$ ja $(3 - (M_v - M_p) = 0) \vee (3 - (M_v - M_p) \geq 3 - (K_v - K_p))$

Paremalt kaldalt vasakule sõitmine:

$\forall (M_p, K_p)$, kus $1 \leq M_p + K_p \leq 2$:

$(M_v, K_v, 0) \rightarrow (M_v + M_p, K_v + K_p, 1)$,

kui

$(M_v + M_p = 0) \vee (M_v + M_p \geq K_v + K_p)$ ja $(3 - (M_v + M_p) = 0) \vee (3 - (M_v + M_p) \geq 3 - (K_v + K_p))$.

Probleemi lahendamiseks, s.t algolekust lõppolekusse viiva tegevuste järjendi leidmiseks, tuleb sooritada otsing olekute ruumis.

Näide 7.5. Produktsiooni kuju ekspertsüsteemis MYCIN illustreerib järgmine reegel:

```
($AND (SAME CNTEXT COLOR GRAMNEG)
      (SAME CNTEXT MORPH BAR)
      (SAME CNTEXT ANAER))
CONCLUDE CNTEXT IDENT BACT HIGH (0.6)
```

s.t „kui

mikroorganismi värving on gramm-negatiivne,
mikroorganismi morfoloogia on kangikujuline,
mikroorganism on anaeroobne,

siis

on alust arvata (tõenäosusega 0,6), et see mikroorganism on bakteriaalse iseloomuga”.

Produktsioonide tüübid

Produktsioonide tüüpide klassifitseerimise aluseks võib võtta skeemi (Pospelov 1989):

välismaailm W,
arutlev süsteem R,
teadmusbaas K.

Siis saame järgmised produktsioonide tüübid:

1) $A_W \rightarrow B_R$

– kui välismaailmast saabub info, siis sellest infost tulenevad muudatused arutlevas süsteemis.

Näide: **kui** sajab, **siis** istu toas.

2) $A_W \rightarrow B_K$

Näide (komandör luurajale): **kui** näed midagi huvitavat, **siis** edasta see sidemehele (K – sidemees).

3) $A_K \rightarrow B_W$

Sellist tüüpi reeglit rakendatakse näiteks siis, kui teadmusbaasis avastatakse vastuoluline info: antakse teade teadmusbaasist välismaailma.

4) $A_R \rightarrow B_K$

– arutleva süsteemi poolt saadud fakt salvestatakse teadmusbaasis.

5) $A_K \rightarrow B_R$

– teadmusbaasist valitakse arutlemiseks vajalik info ja antakse see töötlemiseks arutlevasse süsteemi.

6) $A_W \rightarrow B_W$

– vaadeldavast situatsioonist välismaailmas tuleneb süsteemi tegevus („hetkeline vastukaja”).

7) $A_R \rightarrow B_W$

– „enne mõtelda, siis teha”.

8) $A_R \rightarrow B_R$

– need on arutleva süsteemi sisereeglid, mis kirjeldavad arutlusprotsessi vahesamme; ei mõju vahetult teadmusbaasi sisule ega välismaailma seisundile; kirjeldavad mitmesammuliste arutlusprotsesside üksiksamme.

9) $A_K \rightarrow B_K$

– kirjeldavad teadmiste teisendamise protseduure teadmusbaasis: üldistamist, uute teadmiste saamist vanadest loogilise tuletuse teel, teadmistevaheliste seaduspärasuste kindlakstegemist üksikfaktide töötlemise alusel.

Produktsioone võib interpreteerida ka teisiti. Nt

- A_W – kasutaja küsimus, B_K – süsteemi vastus,
- A_K – kasutaja vastus, B_W – süsteemi küsimus.

Produktsiooni üldkuju võib olla ka keerulisem:

(i); Q; P; $A \rightarrow B$; N,

kus i on produktsiooni nimi, Q – rakendussfäär, P – eeltingimus, $A \rightarrow B$ – tuum, N – järeltingimus.

Näide: **kui** x on hobune, **siis** x ei lenda.

Q: tavalises maailmas on see reegel tõene, kuid kreeka mütoloogias (hobune – pegasus) või vene folklooris (hobune – küürselg sälg) väär.

Q esineb produktsioonides üksnes suurte mittehomoogeensete andme- ja teadmusbaaside korral; N – robotites (siis on tuuma kuju $A_R \rightarrow B_W$).

Tuumad võib omakorda liigitada:

– determineeritud (**kui** kehtib A, **siis** tingimata teha/kehtib B)

a) ühesed,

b) alternatiivsed (**kui** A, **siis** kaaluga p_1 teha/kehtib B_1 , kaaluga p_2 teha/kehtib B_2 , ..., kaaluga p_n teha/kehtib B_n).

– mittedetermineeritud (**kui** kehtib A, **siis** tõenäosusega p teha/kehtib B).

Produktsioonilisel teadmuse esitusel on rida eeliseid teiste teadmuse esituse meetodite ees:

1. loomulikkus: produktsioonide rakendamine on sarnane inimese tunnetuslike protsessidega probleemilahendusel. Lihtne kui-siis esitus võimaldab hõlpsasti selgitada ekspertidele teadmuse struktuuri, mida nendelt ekspertsüsteemi loomisel välja meelitatakse.
2. Selgitusvõime: lihtne on luua selgitusmoodulit, sest reegli eeldused kirjeldavad täpselt, mida on vaja selleks, et aktiveerida reeglit. Kui pidada meeles, milliseid reegleid rakendati, siis saab selgitusmoodul esitada kasutajale arutlusahela.
3. Modulaarsus (mistõttu on lihtne teadmusbaasi täiendada).
4. Protseduraalsus (s.t saab hõlpsasti realiseerida algoritme).
5. Paralleelsus, produktsioonide rakendamise asünkroonsus (see asjaolu leiab kasutust paralleelarvutites).

Puudused:

1. suure teadmusbaasi mittevasturääkivust on raske kontrollida,
2. suurt teadmusbaasi on raske juhtida (valida reeglit paljude üheaegselt sobivate hulgast).

Produktsiooniliste TIS-ide näiteid:

- palju ekspertsüsteeme, nt. DENDRAL, MYCIN;
- adaptiivne produktsioonide süsteem pokkeri mängimiseks (1970);
- HEARSAY – kõnetuvastusega süsteem, kasutati malemängimiseks (1976).
- Programmeerimiskeel OPS5 (1981) põhineb produktsioonidel, põhineb otsetuletusel (kasutati nt arvuti VAX-11 konfiguratsiooni määramiseks).

Produktsioonide süsteem sobib teadmuse esituseks, kui

- 1) ainevaldkond on hajus, on palju fakte (nt kliiniline meditsiin),
- 2) protsesse saab esitada kui sõltumatute tegevuste hulka,
- 3) teadmisi saab lihtsalt eraldada nende kasutamise viisist (nt klassifikatoorne taksonoomia bioloogias).

Harjutusülesanded

1. Antud on nn. generatiivne grammatika (Chomsky 1956)

$G = (\mathbf{N}, \mathbf{T}, \mathbf{P}, \langle \text{lause} \rangle)$,

kus

- mitteterminaalide hulk $\mathbf{N} = \{ \langle \text{lause} \rangle, \langle \text{subjektifraas} \rangle, \langle \text{pärisnimi} \rangle, \langle \text{omadussõna} \rangle, \langle \text{verb} \rangle \}$,
- terminaalide hulk $\mathbf{T} = \{ \text{Mari}, \text{Peeter}, \text{suur}, \text{tubli}, \text{jookseb}, \text{õpib} \}$,
- produktsioonide e ümberkirjutusreeglite hulk $\mathbf{P} =$

{<lause> → <subjektifraas> <verb>,
<subjektifraas> → <omadussõna> <subjektifraas>,
<subjektifraas> → <pärisnimi>,
<pärisnimi> → *Mari*,
<pärisnimi> → *Peeter*,
<omadussõna> → *suur*,
<omadussõna> → *tubli*,
<verb> → *jookseb*,
<verb> → *õpib*}.

Grammatika lähtesümbol on <lause>.

Lause genereerimine algab grammatika lähtesümbolist ja seisneb produktsioonide rakendamises (produktsiooni vasaku poole esinemine asendatakse parema poole esinemisega), kuni saadakse mingi terminaalide järjend – lause.

Kõikvõimalike lausete hulka, mida saab genereerida produktsioonide rakendamisel, nimetatakse grammatikaga genereeritavas keeleks.

Leida antud grammatikaga genereeritav keel.

2. Isa ja kaks poega peavad pääsema jõe vasakult kaldalt paremale. Neil on paat, mis suudab korraka kanda kas isa või kahte poega. Kuidas korraldada vedusid?

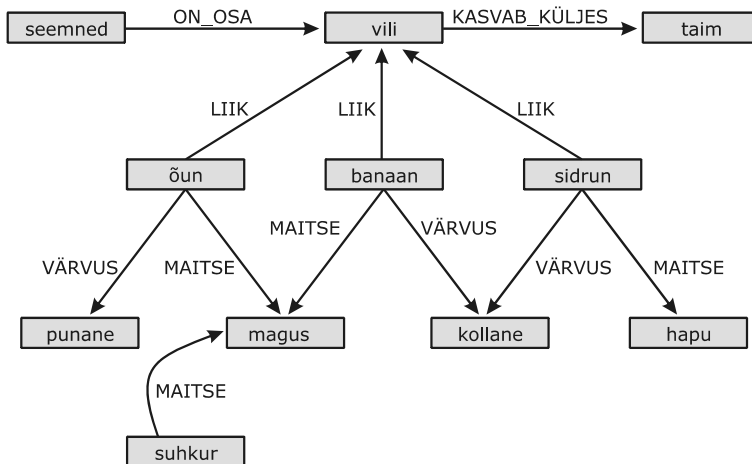
Esitada teadmus produktsioonidena ja lahendada ülesanne, kasutades laiutiotsingut.

8. Semantilised võrgud

Semantilisi võrke kasutatakse nn propositsioonilise info esitamiseks. Propositsioon (väide, väitlause) saab olla kas tõene või väär. Väited esitavad fakte, olles seega deklaratiivse teadmuse vorm.

Matemaatilises mõttes on semantiline võrk suunatud graaf, mille tipud ja kaared on märgendatud. Tippudele vastavad modelleeritava ainevaldkonna objektid, kaarte aga nendevahelised semantilised (e tähenduslikud) seosed (joonis 8.1). Seostel on semantilises võrgus väga oluline osa, sest nad moodustavad teadmuse esituse alusstruktuuri ja võimaldavad teha järeldusi ilmutatult antud faktidest.

Tehisintellektis kasutas semantilisi võrke esimesena 1960. aastatel M. Ross Quillian loomuliku keele mõistmise süsteemis. Iga analüüsitavas lauses esinev sõna aktiveerib semantilises võrgus vastava tipu ja temast lähtuvad kaared. Aktiveeritud osa võrgust – tipud ja neid ühendavad kaared – esitavad lause tähendust. See on analoogiline inimese peas toimivate protsessidega: kuigi me teame tuhandeid sõnu, mõtleme me lauset lugedes ainult konkreetsetele sõnadele. Aluseks on idee, et omavahel seotud mõisted asuvad inimese mälus lähestikku. Semantilisi võrke nimetatakse vahel ka assotsiatiivseteks võrkudeks, sest tipud on üksteisega assotsieeritud (ehk seotud).



Joonis 8.1. Semantiline võrk

Semantilise võrgu tippudele võivad vastata füüsilised objektid, mõisted või situatsioonid.

Seosed võivad olla mitut erinevat tüüpi. Kaks üldkasutatavat **hulgateoreetilist** seost on IS-A (ISA, *is an instance of*, „on näide”, „on üks”) ja A-KIND-OF (AKO, *a kind of*, „on liik”). Seos IS-A tähendab teatava klassi liikmeks (teatava hulga elemendiks) olemist, nt *Jüri on inimene* (Jüri on inimeste hulga element). Seost AKO kasutatakse klasside seostamisel, nt *inimene on elusolend* (inimeste hulk on elusolendite hulga alamhulk). Kasutades neid kahte seost, võime teha semantilises võrgus järeldusi, nt järeldada, et Jüri on elusolend. Veel üks üldine seos on HAS-A („omab”), mida kasutatakse objekti ja selle osa vahelise seose tähistamiseks, nt *auto HAS-A mootor* (aga *Audi IS-A auto*).

Ühte klassi kuuluvatel elementidel on ühised atribuudid ja igal atribuudil on väärtused. Atribuudi ja väärtuse kombinatsioon on omadus. Nt pallil on atribuudid suurus, kaal, kuju, värvus. Atribuudi kuju väärtus on ümmargune. Seega palli omadus on ümmargune kuju. Õhupall on pall, seega on õhupallil kõik palli omadused. Ühe tipu karakteristikute ülekandmist tema alluvale nimetatakse **pärimiseks** (*inheritance*). Klassi kõik liikmed pärivad oma ülemklassi kõik omadused. Lisaks sellele võib igal liikmel muidugi olla eriomadusi. Pärimine on teadmuse esitamisel kasulik, sest ühiseid karakteristikuid pole vaja dubleerida.

Lisaks juba nimetatud hulgateoreetilistele seostele kasutatakse semantilistes võrkudes veel lingvistilisi, loogilisi ja kvantifikatsioonilisi seoseid.

Lingvistilised seosed leiavad kasutust eeskätt loomuliku keele mõistmise süsteemides. Lingvistilised seosed on nt sõnaliik, käane, arv jms (neid nimetatakse grammatilisteks seosteks). Üks lingvistiliste seoste alaliik on veel nn süvakäanete väljendamiseks mõeldud seosed. Süvakäande mõiste võttis esimesena kasutusele Ch.J. Fillmore. Oma 1968. a ilmunud artiklis „The case for case” defineeris ta süvakäanded agent (e tahtliku tegevuse tegija), objekt, koht jmt. Näiteks lauses *Peeter loeb toas raamatut* on Peeter lugemise agent, tuba on lugemise koht ja raamat – objekt.

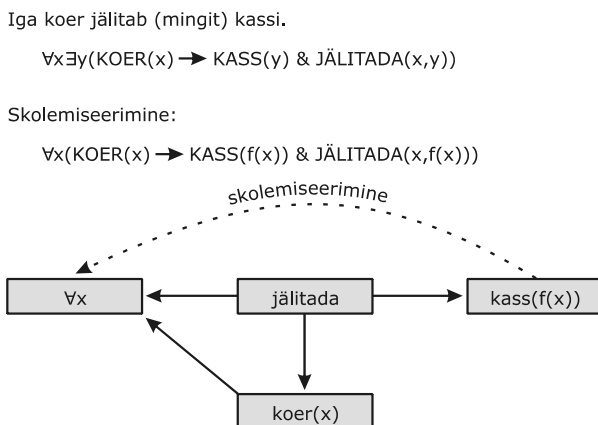
Loogiliste seoste esitamisel semantilistes võrkudes puudub ühtne standard. Konjunktsioon esitatakse ilmutamata kujul, s.t eeldatakse, et kõik võrgus esitatud faktid kehtivad ühe-

aegsel. Teiste loogiliste seoste (eitus, disjunktsioon, implikatsioon jne) esitamiseks tuuakse tavaliselt sisse täiendavad tipud, mis märgendatakse vastava seose nimega, ja täiendavad kaared, mis seovad selle tipuga fakte, mille vahel leiab aset vastav loogiline seos.

Kvantifikatsioonilisteks seosteks võivad olla objektide sellised karakteristikud nagu kaal, pikkus jms, millel on arvulised väärtused. Üheks eriliigiks on loogilised kvantorid. Olemasolukvantor esitatakse ilmutamata kujul, s.t eeldatakse, et kõik võrgus esitatud objektid eksisteerivad. Üldisuskvantori esitamiseks on kasutatud analoogilist võtet nagu loogiliste seoste puhul – tuuakse sisse uus tipp, mille märgendiks on üldisuskvantori tähis, ning sellest tipust lähtuv kaar viitab üldistatavale väitele (joonis 8.2).

Teatud ainevaldkondades kasutatakse eri liiki semantilisi võrke, sh

- stsenaariumid (kus esitatakse põhjuse-tagajärje seoseid),
- klassifikatsioonilised võrgud (esitatakse sisaldumisseoseid),
- arvutusmodelid (esitatakse seoseid argument-väärtus).



Joonis 8.2. Üldisuskvantori kasutamine semantilises võrgus

Joonisel 8.3 on toodud näide programmeerimissüsteemis PRIZ kasutatud arvutusmodelist (Tõugu 1984).

Täisarv w – 100%

osa p – $c\%$

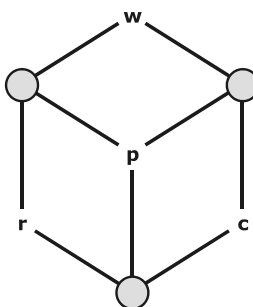
$r = w - p$

Seosed:

1) $w = p + r$

2) $\frac{p}{w} = \frac{c}{100}$

3) $\frac{p}{r} = \frac{c}{100 - c}$



Joonis 8.3. Arvutusmodel protsendi leidmiseks

Roger Schank on kasutanud oma loomuliku keele mõistmise süsteemis (1973) selliseid semantiliste võrkude eriliike nagu kontseptuaalsete sõltuvuste graafid (*conceptual dependency graph*) ja jaotatud semantilised võrgud (*partitioned semantic network*).

Kontseptuaalsete sõltuvuste graafis kasutatakse piiratud hulka seoseid, mille abil väljendatakse tegevusi (nt PTRANS – objekti füüsilise asukoha muutumine, INGEST – objekti enda sisse viimine agendi poolt jne). Nt lauses *Peeter sööb saia* on seos INGEST tegija Peeter ja objekti sai vahel. Sama seos kehtib lauses *Peeter joob vett* tegija Peeter ja objekti vesi vahel.

Jaotatud semantilises võrgus on võrk tükeldatud nn ruumideks, mis koosnevad teatavast hulgast tippudest ja neid ühendavatest kaartest, ning iga sellist ruumi vaadeldakse kui (üldistatud) tippu. See võimaldab hõlpsamini kui tavalises võrgus käsitleda loogilisi seoseid ja kvantoreid.

Semantilisi võrke on kasutatud eelkõige mitmesugustes loomuliku keele töötlemise süsteemides, aga ka mõningates ekspertsüsteemides:

- 1) SIR (1964, B. Raphael) – oli esimene loomulikku (inglise) keelt mõistev küsimusvastussüsteem;
- 2) SCHOLAR (1970, J.L. Carbonell) – õpetamissüsteem (Lõuna-Ameerika geograafia kohta);
- 3) PROSPECTOR (1981) – ekspertsüsteem maavarade leiukohtade selgitamiseks;
- 4) TORUS (TORonto Understanding System) – andmebaaside juhtimissüsteem;
- 5) CASNET – ekspertsüsteem glaukoomi diagnoosimiseks, kasutab stsenaariume;
- 6) PRIZ, NUT – programmeerimissüsteemid, mis kasutavad arvutusmudeleid;
- 7) SNePS²⁸ (1979, Stuart C. Shapiro) – semantiliste võrkude töötlemise süsteem, mis esitab loogika osahulka semantilise võrguna. Tipud vastavad kas konstantidele, muutujatele või propositsioonidele. Kaared vastavad binaarsetele relatsioonidele või esitavad tippude omadusi.

Semantilisi võrke on lihtne esitada programmeerimiskeeles PROLOG. Faktid esitatakse kui predikaadid, nt

ema(anne, peeter). *Anne on Peetri ema*

ja reeglid kujul

p:- p1, p2, ..., pN.

Siin on p nn eesmärk, mis on rahuldatud parajasti siis, kui kõik tema alameesmärgid pk (k=1,..., N) on rahuldatud, nt

lapsevanem(X,Y) :- isa(X,Y).

²⁸ <http://www.cse.buffalo.edu/sneps/> (18.02.2010)

Programm koosneb faktidest ja reeglitest. Programmi täitmisel on olulised tegevused tagurdamine (*backtracking*) ja kujuvõrdlus (*pattern matching*)²⁹. Programm leiab kasutaja sisestatud predikaadi (nn päringu) tõeväärtuse. Interpretaator selgitab esmalt, kas see predikaat esineb andmebaasis, ning jaataval juhul väljastab vastuse jah. Eitaval juhul võetakse päring eesmärgiks ja interpretaator püüab rahuldada selle alameesmärke, sooritades süvitsiotsingul kujuvõrdlust, s.t võrreldes teadaolevaid fakte alameesmärkidega. Juhtimisstruktuuriks on Markovi algoritm, s.t otsing on määratud reeglite sisestamise järjekorraga. See erineb reeglipõhisest süsteemist, kus tavaliselt reeglite sisestamise järjekord ei oma tähtsust.

Kuigi semantilised võrgud on teadmuse esituse kasulikud, on neil siiski piirangud, millest olulisem on nimede standardi puudumine. Nt kasutatakse seoseid IS-A ja AKO nii üldiste kui ka individuaalsete seoste tähistamiseks: kui semantiline võrk sisaldab tippu „tool”, siis see võib tähistada nii konkreetset tooli, kõikide toolide klassi kui ka mõistet „tool”. Seetõttu on raske kontrollida, kas kõik objektid ja seosed on võrgus esitatud järjekindlal ja kooskõlalisel viisil. Praeguseks on see probleem siiski lahendatud tänu märgenduskeelele XML (eXtensible Markup Language) ja ontoloogiatele, mistõttu pole veeb enam lihtsalt andmete hulk, vaid on muutumas semantiliseks veebiks.

Teine semantiliste võrkude probleem on otsingul läbitavate tippude hulk, eriti sel juhul, kui vastus küsimusele on eitav (tuleb läbida kogu võrk). Varem vaadeldud rändkaupmehe probleemi puhul selgus, et n tipu korral, kui iga tipp on seotud iga ülejäänud tipuga, on võrgus $(n-1)!$ seost. Iga probleem ei nõua küll sellist seotuse astet, kuid ometi on olemas kombinatoorse plahvatuse oht.

Nagu eespool märgitud, käsitati semantilisi võrke esialgu kui inimese assotsiatiivse mälu mudelit, kus tipul on seosed teiste tippudega ja info otsimine toimub tippude aktiveerituse levitamise teel. Ometi peavad inimese mälus eksisteerima ka teistsugused mehhanismid. Inimajus on umbes 10^{11} neuronit ja 10^{15} seost. Kui kogu teadmus oleks esitatud ainult semantilise võrgu abil, siis võtaks küsimustele eitavate vastuste leidmine väga palju aega, sest kõik seosed oleksid ju kaasatud.

Nagu eelnevalt selgus, on loogiliste seoste ja kvantorite esitamine semantilises võrgus keeruline. Probleemaatiline on ka heuristilise info kasutamine efektiivseks otsimiseks võrgus. Heuristik on teatavasti rusikareegel, mis võib aidata leida lahendit, kuid see pole garanteeritud sellisel viisil, nagu algoritm tagab probleemi lahendamise. Heuristikud on tehisintellektis väga tähtsad, sest tüüpilised probleemid on nii rasked, et algoritmiline lahend kas ei eksisteeri või on praktikas ebaefektiivne. Ainus standardne semantilise võrku sisseehitatud juhtimisstrateegia on pärimine, kuid mitte kõigil probleemidel ei tarvitse olla selline struktuur.

Pärimisprobleemide paremaks lahendamiseks semantilistes võrkudes on pakutud mitmeid lähenemisi. Nt lisatakse tippudele protseduurid, mis käivitatakse siis, kui tipp muutub aktiivseks. Ometi on niiviisi saavutatud paranemine väike ning sedagi semantiliste võrkude loomuliku väljendusrikkuse arvelt.

²⁹ **Kujuvõrdlus** — kujundi identifitseerimine ta võrdlemise teel mingi etteantud kujundite kogumiga ning lähima valimise teel mingi etteantud kriteeriumi alusel. IT terministandardi sõnastik <http://www.keelevaab.ee> (18.02.2010)

Harjutusülesanded

1. Koostada semantiline võrk (kuni 5 tippu), mis esitab ametialaseid vm suhteid inimeste vahel (valdkonna võib vabalt valida).
2. (Giarratano, Riley 2004:111) Koostada semantiline võrk, mis kirjeldab arvuteid: mikroarvuti, suurarvuti, superarvuti, arvutisüsteem, spetsialiseeritud arvuti, üldotstarbeline arvuti, arvuti kiibil, ühe protsessoriga arvuti, mitme protsessoriga arvuti, konkreetset arvutitüübid. Kasutada seoseid AKO ja IS-A.

9. Freimid

Freim e kaader (*frame*) on andmekeskne teadmuse esitus, mis seostab objekti mingi tunnusomaduste kogumiga.³⁰

Freimi mõiste aluseks on hüpotees, et inimene säilitab oma mälus üldistatud teadmisi – suurt hulka stereotüüpe, mida ta tunneb kas oma isiklikust kogemusest või mõnest muust allikast. Näiteks kujutame ette, mida tähendavad „sünnipäevapidu”, „jalgpallimäng”, „teatriskäik”, „perekonnatüli”, „vestlus ülemusega” jne. Iga selline üldistatud teadmine ongi freim. Inimene püüab kõikjal näha tüüpilist, et osata käituda uues situatsioonis. Inimese mälus arvatakse olevat tuhandeid freime.

Freimid võttis teadmuse esituseks kasutusele 1975. a USA psühholoog Marvin Minsky oma tehisnägemise süsteemis. Psühholoogilised alused loodi aga juba 1930. aastatel.

Freimi üldkuju on antud joonisel 9.1 ja üks konkreetne freim joonisel 9.2. Freim on analoogiline kirjega kõrgtaseme keeles nagu C. Kirje väljadele ja väärtustele vastavad freimi slotid (*slot* – pilu, lõhe) ja nende sisud, täitjad (ingl k *filler*).

| FREIMI NIMI |
|------------------------------|
| 1. sloti nimi: 1. sloti sisu |
| 2. sloti nimi: 2. sloti sisu |
| ... |
| n. sloti nimi: n. sloti sisu |

Joonis 9.1. Freimi üldkuju

| INIMENE |
|--|
| AKO: ELUSOLEND Kasv: 30 kuni 220 cm Mass: 1 kuni 200 kg Rahvus: STRING (Vaikimisi - Eestlane) Nimi: STRING |

Joonis 9.2. Freim INIMENE

³⁰ IT terministandardi sõnastik <http://www.keeleveeb.ee/> (18.02.2010)

Sloti sisu võib olla mitmesugune, nt:

- 1) väärtus või väärtuste hulk: 1–200 kg,
- 2) väärtuse tüüp: STRING,
- 3) vaikimisi (*default*) võetav väärtus (e vaikeväärtus), nt Vaikimisi – Eestlane,
- 4) protseduuri nimi, mis kutsutakse välja sloti täitmisel (sealjuures on võimalik dialoog kasutajaga),
- 5) teise freimi nimi, nt ELUSOLEND.

Mõned freimipõhised programmeerimisvahendid nagu KEE (Knowledge Engineering Environment) võimaldavad salvestada slottides reegleid, graafikat, kommentaare, silumisinfot, küsimusi kasutajale, hüpoteese situatsiooni kohta, teisi freime.

Slottide vaikimisiväärtused esitavad ootusi, mis inimestel on teatud situatsioonis oma eelneva kogemuse põhjal. Kui inimene tuvastab uue situatsiooni, siis ta ei alusta tühjalt kohalt, vaid modifitseerib lähimat freimi, mis aitab tal käsitleda seda situatsiooni.

Kuna slottide täitjatena võivad esineda teised freimid, siis moodustub freimide hierarhiline süsteem, kus olulisel kohal on semantiliste võrkude teooriast ülevõetud *omaduste pärimine*, tänu seostele IS-A ja AKO.

Kui freimis on slotid sisustatud konkreetsete andmetega, siis seda nimetatakse freimi *eksemplariks*. Freimi, mida konkretiseeritakse, nimetatakse sellisel juhul prototüüpfreimiks e näidisfreimiks. Joonisel 9.3 on toodud prototüüpfreimi INIMENE eksemplar.

| Inimene127 |
|---|
| IS-A: INIMENE Kasv: 175 cm Mass: 70 kg Rahvus: Venelane Nimi: Maria |

Joonis 9.3. Freimi INIMENE eksemplar

Freimide hierarhia tipus on kõige üldisemad freimid. Freimid modelleerivad reaalse maailma objekte, kasutades enamiku atribuutide jaoks üldist teadmust ja erijuhtude jaoks spetsiifilist teadmust. Näiteks me teame, et üldiselt linnud lendavad. Aga teatud liiki linnud nagu pingviin või jaanalind ei lenda. Need on spetsiifilisemad lindude liigid ja nende karakteristikud asuvad freimide hierarhias madalamal kui selliste tüüpiliste lindude omad nagu vares või papagoi.

Joonisel 9.4–9.6 on toodud veel mõned freimide näited.

```
(IRA
(AKO($VALUE(PERSON)))
(MEETINGWHEN($PREFER(DURING AFTERNOON)(ON FRIDAY)))
(MEETINGWHERE($DEFAULT(B-206)))
(PA_MEETINGWHEN($DEFAULT(AT 3 PM)(AT 10 AM)$PREFER(ON THURSDAY)))
```

Joonis 9.4. Freim keeles FRL (R. Bruce Roberts ja Ira P. Goldstein 1977), mis kirjeldab isikut nimega IRA

```
(DATE
(MONTH - (JANUARY..DECEMBER))
(DAY - (INTEGER(1..31)))
(YEAR - (INTEGER(TOFILL ASSUME 1999)))
(WEEKDAY - (SUNDAY..SATURDAY)
(WHENFILLED FIND DAY OF DATE)))
```

```
(CURRENTDATE
ISA DATE
(MONTH - FEBRUARY)
(DAY - 29)
(YEAR - 2008)
(WEEKDAY - FRIDAY))
```

Joonis 9.5. Kaks freimi – protüüpfreim DATE ja eksemplarfreim CURRENTDATE lennupiletite tellimise süsteemist GUS (1977)

HÕIVAMINE

```
SUP=SOCACTION
AGENT=REQUIRE(PERSON)
SOURCE=REQUIRE(PERSON)
OBJECT=REQUIRE(PHOBJ)
CONSEQUENCE=POSSESS
  PACIENT=AGENT from SELF
  OBJECT=OBJECT from SELF
```

VARGUS

```
SUP=HÕIVAMINE
SETTING=NOT AGREEMENT
  PACIENT=SOURCE from SELF
  OBJECT=OBJECT from SELF
GOAL=OWN
  PACIENT=AGENT from SELF
  OBJECT=OBJECT from SELF
```

Joonis 9.6. Kaks freimi Tartu Ülikoolis 1980. aastatel loodud teksti mõistmise süsteemist. Märksõnad aktiveerivad freime, mistõttu suudab süsteem tuvastada tekstis kirjeldatud sündmuse, nt järgnevas tekstis varguse (märksõnad kursiivis): „Jüri *sõitis* Moskva rongiga Tartust Jõgevale. Rongist väljudes *unustas* ta kupeesse portfelli rahaga. Samas kupees sõitev Jaan *märkas* võõrast portfelli. Ta *võttis* portfelligist raha ja *ostis* selle eest endale Tallinnas käekella.”

Freime võib liigitada ka nende kasutuse järgi. Situatsioonifreim (nt töörežiim, avarii) esitab teadmuse selle kohta, mida on oodata antud situatsioonis. Tegevusfreim esitab protseduraalset teadmust. Tegevusfreimis kirjeldavad slotid tegevusi, mida täita antud situatsioonis. Slottide sisuks on protseduurid tegevuste täitmiseks, nt eemaldada konveieri-rihma defektne osa. Situatsiooni- ja tegevusfreimide kombinatsiooni saab kasutada põhjuse ja tagajärje suhete kirjeldamiseks nn kausaalfreimis. Struktuurifreime kasutatakse objektide ja mõistete tähistamiseks (nt laen, võlakiri). Rollifreimid on nt autojuht, kassiir, klient. Freimid-stsenariumid on nt sünnipäevapidu, restoranikülastus.

Freimipõhised ekspertsüsteemid on väga kasulikud põhjusliku teadmuse esituseks, sest nende info on organiseeritud põhjuse ja tagajärje abil. Reeglipõhised süsteemid seevastu tuginevad organiseerimata teadmusele, mis ei ole põhjuslik.

On loodud palju freimide süsteeme. Üks mõjukamaid süsteeme, mis esmakordselt demonstreeris freimide võimet loominguiliselt avastada matemaatilisi mõisteid oli Douglas B. Lenati AM (Automated Mathematician) 1976. a. Süsteem tegi oletusi huvitavate uute mõistete kohta ja uuris neid. Ta tuli toime mõnede teoreemide täiesti uute matemaatiliste tõestustega.

Freimid võeti algselt kasutusele stereotüüpsete teadmiste esitamiseks. Stereotüübil on hästi defineeritud tunnused, nii et paljudel slottidel on vaikimisiväärtused. Stereotüüpide head näited on matemaatilised mõisted. Freimi paradigma on intuiitiivselt huvitav, sest ta organiseerib teadmuse viisil, mida on lihtsam mõista kui loogikapõhise või reeglipõhise süsteemi oma.

Freimide süsteemi kui teadmuse esitamise mudeli põhieelised:

- väljendab inimese mälu organiseerituse kontseptuaalset alust,
- paindlikkus,
- ülevaatlikkus.

Freimipõhiseid programmeerimiskeeli: KRL (D. Bobrow ja T. Winograd, 1977), FRL (1977), KL-ONE (R. Brachman 1979, 1985), hübriidne KRYPTON (R. Brachman jt 1983), NIKL (T.R. Kaczmarek jt 1986).

Põhilised raskused freimide kasutamisel on seotud sellega, et freimid lubavad slottide piiramatut teisendamist ja tühistamist. Seda võib illustreerida järgneva näitega. Joonisel 9.7 on toodud elevanti freim.

| |
|----------------|
| Nimi: elevant |
| AKO: imetaja |
| Värvus: hall |
| Jalgu: 4 |
| Keha: silinder |

Joonis 9.7. Elevanti freim

Kui nüüd oletame, et eksisteerib 3-jalgne elevant, siis see kirjeldus tema kohta ei kehti, kuna sätestab täpselt 4 jalga. Olukorda ei paranda sloti *Jalgu* muutmine, lubamaks 3, 2, 1 või 0 jalga, sest teiste slottidega võivad tekkida samasugused probleemid (nt värvus ei tarvitse olla

hall, vaid valge). Alternatiivne võimalus oleks vaadelda freimi kui tüüpilise elevandi kirjeldust. See aga toob pärimise tõttu kaasa uued probleemid. Elevant on imetaja ja meie freimide süsteem ütleb, et ta on tüüpiline imetaja. Aga nii see paraku ei pruugi olla.

Enamik freimisüsteeme võimaldavad defineerida muudetavaid slotte. Kui aga iga slotti võib muuta, siis omadusi, mida freim pärib, saab muuta või tühistada hierarhia ükskõik millisel tasemel. Iga freim kehtestab oma reeglid. Sellises tõkestamata süsteemis pole mitte miski kindel, seega on võimatu mõistlikult koostada keerulisemaid objekte, nt kolmejalgne elevant. Sedasama tüüpi probleemid tekivad pärimisega semantilistes võrkudes. Kui tipu omadusi võib muuta, siis pole miski kindel.

Lisaks tavapärastele hierarhilistele freimisüsteemidele on teadmuse esituseks kasutatud ka **skripte** (R. Schank ja R. Abelson 1977). Skript (*script*) on ajaliselt järjestatud situatsiooni-freimide hulk (stsenaarium). Näiteks restoranikülastuse skript koosneb sellistest stseenidest nagu saabumine, tellimine, söömine ja lahkumine (Rich ja Knight 1991:286). Skripte on kasutatud loomuliku keele tekstide mõistmisel (vrd hüpersündmuste tuvastamine TÜ süsteemis).

Harjutusülesanded

1. Koostada prototüüpfreimid (esitusviisi ja detailsuse võib vabalt valida):

KODULOOM

KOER

KUTSIKAS

ja ühe kindla kutsika eksemplarfreim.

2. (Giarratano, Riley 2004:111) Koostada freimide süsteem selle õppehoone kohta, kus toimub teie õppetöö. Vaadelda tööruume, loenguruume, arvatiklasse jne. Loodav süsteem peab sisaldama täidetud slottidega näiteid iga vastavat tüüpi freimi jaoks.

3. (Giarratano, Riley 2004:111) Koostada tegevusfreimide süsteem, mis näitab, kuidas toimida arvutisüsteemi riistvara rikke korral. Vaadelda ketta riket, voluvarustuse, keskprotsessori ja mälu probleeme.

10. Esimest järku predikaatarvutus teadmuse esitusel

Teadmuse võib esitada ka loogika abil. Loogikapõhist teadmuse esitust kasutati juba 1950. aastatel loodud programmides, mis modelleerisid inimese intellektuaalse tegevuse mõningaid külgi.

Varaseim formaalne loogika oli 4. saj. e.m.a. Aristoteelse loogika, mis põhineb süllogismidel. Klassikaline näide süllogismi kohta:

1. eeldus: Iga inimene on ekslik.

2. eeldus: Turing on inimene.

Järeldus: Turing on ekslik.

Formaalne loogika tegeleb väidete vormiga (s.t süntaksiga), arutluse täpsete reeglite uurimisega, mitte aga sisuga (s.t semantikaga). Seda on eriti tähtis silmas pidada ekspert-süsteemide korral: teadmus tuleb eraldada arutlusest.

Aristoteelse süllogismid olid loogika aluseks kuni aastani 1847, kui inglise matemaatik George Boole avaldas esimese teose sümbolloogika kohta. Boole loobus Aristoteelse subjekti eksistentsi nõudest (mille kohaselt näiteks ainult siis, kui eksisteerib vähemalt üks näkineiu, saab väita, et kõik näkineiuud ujuvad hästi), et võimaldada arutelusid ka tühjade klasside üle. Boole defineeris ka aksioomide hulga, mis koosnevad objekte ja klasse esitavatest sümbolitest ja algebralistest tehetest nende sümbolitega manipuleerimiseks.

Lausearvutus (*propositional logic*) on sümbolloogika, mis manipuleerib väidetega (propositsioonidega) – jutustavate lausetega, mis saavad olla kas tõesed või väärad (nt *kolmnurgal on kolm nurka*). Lausest saab moodustada keerulisemaid lauseid, kasutades loogilisi tehteid (eitus, konjunktsioon, disjunktsioon, implikatsioon jt). Lausearvutusel on piirangud, eeskätt see, et ta suudab käsitleda ainult täislauseid ega uuri lausete sisemist struktuuri. Lausearvutuses pole näiteks võimalik kontrollida eespool toodud süllogismi kehtivust. Seda suudab aga teha 1. järku predikaatarvutus (IJPA), mis on ka programmeerimiskeele PROLOG aluseks.

1. järku predikaatarvutuse formaalne ülesehitus³¹

Defineerime järgmised mõisted.

Tähestik. Koosnegu IJPA tähestik järgmistest sümbolitest:

1. eraldajad () ,
2. loogiliste tehete märgid \neg (eitus), $\&$ (konjunktsioon), \vee (disjunktsioon), \rightarrow (implikatsioon)
3. kvantorid \forall (üldisuskvantor) ja \exists (olemasolukvantor)
4. (indiviid)muutujate sümbolid x_k ($k \geq 1$)
5. n-kohaliste funktsioonide sümbolid f_k^n ($k \geq 1, n \geq 0$), seejuures sümboleid f_k^0 nimetame konstantideks
6. n-kohaliste predikaatide sümbolid p_k^n ($k \geq 1, n \geq 1$).

Edaspidi kasutame ka järgmisi tähistusi: x_k asemel u, v, w, x, y, z, \dots ; f_k^0 asemel a, b, c, \dots ; f_k^n ($n \neq 0$) asemel f, g, h, \dots ; p_k^n asemel P, Q, R, \dots .

Tähestiku sümbolitest võib konstrueerida mitmesuguseid avaldisi. Vaatleme terme, elementaarvalemeid ja valemeid.

Term:

- a) iga indiviidmuutuja ja iga konstant on term.
- b) Kui t_1, \dots, t_n on termid ($n \geq 1$), siis ka $f_k^n(t_1, \dots, t_n)$ on term.
- c) Avaldis on term parajasti siis, kui ta on seda a) või b) põhjal.

³¹ Vt ka (Chang, Lee 1973), (Koit 1989).

Elementaarvalem ehk atomaarne valem ehk aatom:

kui p^n_k on predikaatsümbol ja t_1, \dots, t_n on termid, siis $p^n_k(t_1, \dots, t_n)$ on elementaarvalem.

Valem:

a) iga elementaarvalem on valem.

b) Kui F ja G on valemid ning x on indiviidmuutuja, siis $(\neg F)$, $(F \& G)$, $(F \vee G)$, $(F \rightarrow G)$, $((\forall x) F)$, $((\exists x) F)$ on valemid.

c) Avaldis on valem parajasti siis, kui ta on seda a) või b) põhjal.

Valemites $((\forall x) F)$ ja $((\exists x) F)$ nimetatakse valemite F vastavalt üldisus- ja olemasolukvantori mõjupiirkonnaks. Muutujat x nimetatakse kvantoriga seotud muutujaks. Valemeid üles kirjutades jätame edaspidi osa sulge ära, lugedes kvantoreid koos eitusega kõrgeima prioriteediga teheteks. (Teiste tehete prioriteedid: $\& \vee \rightarrow$.)

Kinnine valem. Öeldakse, et valem on kinnine, kui kõik temas esinevad muutujad on seotud. Meid huvitavad edaspidi just kinnised valemid.

Valemi interpretatsioon. Et anda valemile sisu, interpreteeritakse teda kui teatavat väidet vaadeldava indiviidide piirkonna kohta. Selleks tuleb fikseerida indiviidide piirkond (indiviidmuutujate väärtuste piirkond) ning määrata valemis esinevate konstantide, predikaat- ja funktsionaalsümbolite tähendused.

Valemi interpretatsiooniks nimetatakse paari, mis koosneb mittetühjast hulgast E (nn interpretatsiooni kandjast) ja kujutusest, mis seab vastavusse

- igale predikaatsümbolile p^n_k mingi n -kohalise relatsiooni hulgal E ,
- igale funktsionaalsümbolile f^n_k mingi n -kohalise funktsiooni hulgal E ,
- igale konstandile f^0_k mingi elemendi hulgal E .

Indiviidmuutujad omandavad väärtusi hulgast E .

Antud interpretatsiooni korral saab iga kinnine elementaarvalem $p^n_k(t_1, \dots, t_n)$ tõeväärtuse „tõene” (1) või „väär” (0): kui hulga E elemendid, mis vastavad termidele t_1, \dots, t_n , kuuluvad relatsiooni, mis on määratud selle interpretatsiooniga, siis loetakse elementaarvalemit tõeseks, vastasel korral vääraks.

Kvantoreid mittesisaldava mitteelementaarse valemi tõeväärtus arvutatakse tema koostisosade väärtustest nii, nagu on näidatud tabelis 10.1 (F ja G on valemid).

Tabel 10.1. Tõeväärtustabelid

| F | G | $\neg F$ | $F \& G$ | $F \vee G$ | $F \rightarrow G$ |
|---|---|----------|----------|------------|-------------------|
| 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 | 1 | 1 |

Valem $\forall x F$ on tõene, kui x iga väärtuse korral hulgast E on valem F tõene; vastasel korral on valem $\forall x F$ väär. Valem $\exists x F$ on tõene, kui leidub selline x väärtus hulgast E , et valem F on tõene; vastasel korral on $\exists x F$ väär. Lõpliku piirkonna E korral võib selliste valemite tõeväärtused määrata tõeväärtustabelist.

Näide 10.1. Vaatleme valemite $\forall x P(x)$ ja $\exists x \neg P(x)$ sellist interpretatsiooni, kus $E = \{1, 2\}$ ning predikaatsümbolile P vastab relatsioon

| | |
|-------|------|
| P(1) | P(2) |
| tõene | väär |

Selles interpretatsioonis on esimene valem väär ja teine tõene.

Kehtestatav (rahuldav) valem. Valem F nimetatakse kehtestatavaks, kui leidub selline interpretatsioon, milles F on tõene. Kui valem F on tõene interpretatsioonis I , siis öeldakse, et I on F mudel ehk I rahuldab valem F . Näites 10.1 vaadeldud interpretatsioon on valem $\exists x \neg P(x)$ mudel.

Kui valem F on väär interpretatsioonis I , siis öeldakse, et I ei rahulda valem F . Näites 10.1 vaadeldud interpretatsioon ei rahulda valem $\forall x P(x)$.

Samaselt tõene (samaselt väär) valem. Kui mingi valem on tõene kõikvõimalikes interpretatsioonides, siis nimetatakse teda samaselt tõeseks. Näiteks valem $P(a) \rightarrow (P(a) \vee P(b))$ on samaselt tõene.

Valemit nimetatakse samaselt vääraks (ehk mittekehtestatavaks), kui ta on väär kõikvõimalikes interpretatsioonides.

Kui valem F on samaselt tõene, siis on tema eituse $\neg F$ samaselt väär.

Viimati defineeritud mõisted võib üle kanda ka valemite hulgale. Seejuures eeldatakse, et kõik valemid hulgas on seotud konjunktsiooni märgiga.

Kui valemite hulk $S = \{F_1, \dots, F_n\}$ on tõene interpretatsioonis I , siis öeldakse, et I on S mudel ehk I rahuldab hulka S .

Öeldakse, et valemite hulk $S = \{F_1, \dots, F_n\}$ on *vasturääkiv* ehk mittekehtestatav, kui konjunktsioon $F_1 \& \dots \& F_n$ on samaselt väär.

Öeldakse, et valemite hulk $S = \{F_1, \dots, F_n\}$ on *kehtestatav*, kui konjunktsioon $F_1 \& \dots \& F_n$ on kehtestatav, s.t leidub interpretatsioon, milles iga valem F_i ($i=1, \dots, n$) on tõene.

Öeldakse, et valemite hulk $S = \{F_1, \dots, F_n\}$ on tõene (väär) interpretatsioonis I , kui konjunktsioon $F_1 \& \dots \& F_n$ on tõene (väär) interpretatsioonis I .

Valemi (valemite hulga) *loogiline järelduvus* valemite hulgast. Valem G järeljub loogiliselt valemite hulgast $S = \{F_1, \dots, F_n\}$, kui iga interpretatsioon, mis rahuldab valemite hulka S , rahuldab ka valem G .

Näide 10.2 Vaatleme valem

$$F_1 = \forall x (P(x) \rightarrow Q(x)),$$

$F_2 = P(a)$.

Näitame, et valem $Q(a)$ järelneb loogiliselt valemitest F_1 ja F_2 . Vaatleme suvalist interpretatsiooni I , mis rahuldab valemit $F_1 \& F_2$. Selles interpretatsioonis peavad mõlemad valemid F_1 ja F_2 olema tõesed. Oletame vastuväiteliselt, et valem $Q(a)$ on interpretatsioonis I väär. Siis peab ka valem F_1 olema selles interpretatsioonis väär. See aga on vastuolus eeldusega. Järelikult on valem $Q(a)$ tõene igas interpretatsioonis, mis rahuldab valemit $F_1 \& F_2$, s.t $Q(a)$ järelneb loogiliselt valemitest F_1 ja F_2 .

Tõestame nüüd kaks edaspidise käsitluse jaoks olulist tulemust.

Tulemus 10.1. Olgu antud valemid F_1, \dots, F_n ja valem G . Valem G järelneb loogiliselt valemitest F_1, \dots, F_n parajasti siis, kui valem $F_1 \& \dots \& F_n \rightarrow G$ on samaselt tõene.

Tõestus (\Rightarrow) Järeldugu valem G loogiliselt valemitest F_1, \dots, F_n . Olgu I suvaline interpretatsioon. Kui kõik valemid F_1, \dots, F_n on tõesed selles interpretatsioonis, siis definitsiooni kohaselt on seda ka G . Järelikult on ka valem $F_1 \& \dots \& F_n \rightarrow G$ tõene. Kui aga kõik valemid F_1, \dots, F_n pole tõesed selles interpretatsioonis (vähemalt üks on väär), siis on valem $F_1 \& \dots \& F_n \rightarrow G$ ometi tõene selles interpretatsioonis. Järelikult on ta tõene igas interpretatsioonis, s.t samaselt tõene.

(\Leftarrow) Olgu valem $F_1 \& \dots \& F_n \rightarrow G$ samaselt tõene. Vaatleme tema suvalist interpretatsiooni I . Kui valem $F_1 \& \dots \& F_n$ on selles interpretatsioonis tõene, siis peab seda olema ka valem G . Järelikult valem G järelneb loogiliselt valemitest F_1, \dots, F_n .

Tulemus 10.2. Olgu antud valemid F_1, \dots, F_n ja valem G . Valem G järelneb loogiliselt valemitest F_1, \dots, F_n parajasti siis, kui valem $F_1 \& \dots \& F_n \& \neg G$ on samaselt väär.

Tõestus. Tähistame valemite A ja B loogilist samaväärsust $A=B$. Kuna $\neg(F_1 \& \dots \& F_n \rightarrow G) = \neg(\neg(F_1 \& \dots \& F_n) \vee G) = F_1 \& \dots \& F_n \& \neg G$, siis tulemuse 10.1 kohaselt järelneb valem G loogiliselt valemitest F_1, \dots, F_n parajasti siis, kui valem $F_1 \& \dots \& F_n \& \neg G$ on samaselt väär.

Kui valem G järelneb loogiliselt valemitest F_1, \dots, F_n , siis nimetatakse valemit $F_1 \& \dots \& F_n \rightarrow G$ *teoreemiks*. Valemite F_1, \dots, F_n nimetatakse teoreemi *eeldusteks* ja valemit G teoreemi *väiteks*.

Näide 10.3 Näitest 10.2: kuna valem $Q(a)$ järelneb loogiliselt valemite hulgast $\{\forall x (P(x) \rightarrow Q(x)), P(a)\}$, siis valem $\forall x (P(x) \rightarrow Q(x)) \& P(a) \rightarrow Q(a)$ on teoreem.

Ülesannet, mis seisneb selle asjaolu tõestamises, et teatav valem järelneb loogiliselt antud valemite hulgast, nimetatakse *teoreemi tõestamise probleemiks* (TTP).

Tulemuse 10.1 kohaselt võib selle asjaolu tõestamiseks, et valem G järelneb loogiliselt valemitest F_1, \dots, F_n , tõestada, et valem $F_1 \& \dots \& F_n \rightarrow G$ on samaselt tõene.

Teatavasti ei eksisteeri algoritmi, mis suvalise IJPA valemiga korral selgitaks, kas see valem on samaselt tõene, s.t IJPA on mittelahenduv. Eksisteerib aga algoritm, mis iga samaselt tõe valemiga korral teeb kindlaks, et see valem on samaselt tõene, s.t IJPA on poolalahenduv.

Praktikas osutub mugavamaks kindlaks teha mitte seda, kas valem on samaselt tõene, vaid hoopis kontrollida, kas ta on samaselt väär. Seepärast vaatleme teoreemi $F_1 \& \dots \& F_n \rightarrow G$ tõestamiseks valemite $\neg(F_1 \& \dots \& F_n \rightarrow G)$ ehk $F_1 \& \dots \& F_n \& \neg G$, s.t lähtevalemi eitust, ja tõestame, et ta on samaselt väär ehk, nagu öeldakse, *kummutame (refute)* valemi. Selleks on vaja näidata, et ei eksisteeri ühtegi interpretatsiooni, mille korral valemid F_1, \dots, F_n ja $\neg G$ oleksid korraga tõesed.

Seoses predikaatarvutuse poollahenduvusega on tõestus (kummutamine) edukas vaid siis, kui valem G järeldeb loogiliselt valemitest F_1, F_2, \dots, F_n . Vastasel korral võib kummutamisprotseduur töötada lõpmatuseni (või anda negatiivse vastuse).

Predikaatarvutuse valemi võimalike interpretatsioonide hulk pole lõplik, sest juba interpretatsiooni kandjat võib valida piiramatul arvul erinevatel viisidel. Seetõttu ei saa asjaolu, kas valem on samaselt väär, üldiselt välja selgitada valemi tõeväärtuse kontrollimise teel kõikvõimalikes interpretatsioonides.

1965. a esitas USA loogik J. Alan Robinson teoreemide automaatse tõestamise meetodi – resolutsioonimeetodi, mida hakati kasutama TIS-ides. Selle meetodiga tutvume 3. peatükis.

Teadmuse esitus

1JPA rakendamisel teadmuse esituse meetodina tuleb kõik teadmised ainevaldkonna kohta esitada valemitega, mis lisatakse 1JPA aksioomidele uute aksioomidena. Lahendatav probleem püstitatakse kui TTP.

Teadmuse esitusel tuleb kõigepealt valida konstandid (mis määravad ainevaldkonna objektid), funktsionaal- ja predikaatsümbolid ning nende interpretatsioonid. Seejuures kerkib aga rida küsimusi.

Küsimus 1: predikaatide valik. Predikaatide valikul tuleb lähtuda tulevastest probleemidest, mida tehisintellektisüsteem lahendada peab hakkama.

Vaatame näiteks lauset *Igaüks, kes on lõpetanud gümnaasiumi Eesti Vabariigis 2000. aastal või hiljem, oskab lugeda kolmes keeles.*

Predikaatide valik 1:

$P(x)$ – x on lõpetanud gümnaasiumi Eesti Vabariigis 2000. a või hiljem,

$Q(x)$ – x oskab lugeda kolmes keeles.

Siis saame vaadeldava lause esitada valemiga

$\forall x (P(x) \rightarrow Q(x))$.

TIS, kus kasutatakse selliseid predikaate, ei suudaks vastata küsimusele „Missuguseid keeli oskab lugeda see, kes on lõpetanud keskkooli Eesti Vabariigis 2000. a või hiljem?”

Predikaatide valik 2:

$P(x)$ – x on lõpetanud gümnaasiumi Eesti Vabariigis 2000. a või hiljem,

$Q1(x)$ – x oskab lugeda eesti keeles,

$Q2(x)$ – x oskab lugeda vene keeles,

$Q3(x)$ – x oskab lugeda inglise keeles,

$Q4(x)$ – x oskab lugeda saksa keeles,

$Q5(x)$ – x oskab lugeda prantsuse keeles.

Vaadeldav lause esitub valemiga „ $x(P(x) \rightarrow (Q1(x) \& Q3(x) \& (Q2(x) \vee Q4(x) \vee Q5(x))))$ ”.

Predikaatide valik 3:

$P(x)$ – x on lõpetanud gümnaasiumi Eesti Vabariigis 2000. a või hiljem,

$R(x,y)$ – x oskab lugeda keeles y,

$E(x,y)$ – $x=y$.

Vaadeldav lause esitub valemiga „ $x(P(x) \rightarrow \exists y \exists z \exists w (R(x,y) \& R(x,z) \& R(x,w) \& \neg E(y,z) \& \neg E(y,w) \& \neg E(z,w)))$ ”

Küsimus 2: kas välistav või mittevälistav „või”?

Näiteks kuidas esitada väidet *Patsiendile A määrata kas ravim X või ravim Y?* Võib arvata, et siin sobiks välistav „või”, aga disjunktsioon vastab mittevälistavale „või”-le: $p \vee q$ tähendab „kas p või q või mõlemad”.

Küsimus 3: implikatsiooni kasutamine. Kuigi $p \rightarrow q$ on tõlgendatav „kui p, siis q”, ei vasta implikatsioonitehe intuiitiivses mõttes põhjuse ja tagajärje seosele, sest väärist eeldusest järeldub mis tahes. Näiteks on tõene lause *kui 1=2, siis saabub aastal 2050 maailma lõpp*.

Lisaks lausearvutusele ja IJPAle on teadmuse esitusel kasutatud mitmesuguseid muid loogikaid:

- 2. (ja kõrgemat) järku predikaatarvutust, kus ühed predikaadid võivad olla teiste argumentideks;
- mitmevalentseid loogikaid, kus on rohkem kui kaks tõeväärtust (näit. 3-valentses loogikas „väär”, „määramata”, „tõene”);
- tõenäosusloogikat, kus määratakse lausete tõesuse ja vääruse tõenäosus;
- modaalne loogika, mis võimaldab esitada nn modaalsusi, nagu „võimalik” ja „paratamatu”;
- uskumuste (*belief*) loogika, kus saab esitada väiteid, nagu „A usub, et B usub, et p” (on rakendatud näiteks kasutaja modelleerimisel);
- veendumuste loogika (kus esinevad operaatorid *on_veendunud*, *kahtleb* jms);
- tunnete loogika (kus esinevad operaatorid *tajub*, *kardab*, *muretsseb*, *loodab*, *rõõmustab*, *kahetseb* jms);
- aja-, ruumi-, tegevuste jms nn pseudofüüsilised loogikad.

Harjutusülesanded

1. Tuua sisse sobivad predikaadid ja esitada IJPA valemitega järgmised väited.

- Kõik koerad on imetajad.
- Ükski koer ei ole elevant.
- Mõnes programmis on vead.
- Üheski minu programmis pole vigu.
- Kõigis sinu programmides on vead.

2. Koostada tõeväärtustabelid ja leida iga järgmise väite korral, kas on tegu samaselt tõese, samaselt väära või kehtestatava valemiga.

- Kui ma läbin selle kursuse ja saan hinde A, siis ma läbin selle kursuse või saan hinde A.
- Kui ma läbin selle kursuse, siis ma saan hinde A ja ma läbin selle kursuse ega saa hinnet A.
- $((A \ \& \ \neg B \rightarrow (C \ \& \ \neg C)) \rightarrow (A \rightarrow B))$
- $(A \rightarrow B) \ \& \ (\neg B \vee C) \ \& \ (A \ \& \ \neg C)$

3. Valida predikaadid ja esitada IJPA valemitega väited:

a) iga ratsionaalarv on reaalarv;

b) leidub arv, mis on algarv;

c) iga täisarvu jaoks leidub parajasti üks täisarv, mis vahetult järgneb talle.

4. Antud on interpretatsioon: kandja $E = \{1, 2\}$, kujutus:

a -1, b - 2, $f(1) = 2$, $f(2) = 1$; $P(1,1)$ - tõene, $P(1,2)$ - tõene, $P(2,1)$ - väär, $P(2,2)$ - väär.

Leida järgmiste valemite tõeväärtused selles interpretatsioonis:

$P(a, f(a)) \ \& \ P(b, f(b))$

$\forall x \ \exists y \ P(y, x)$

$\forall x \ \forall y (P(x, y) \rightarrow P(f(x), f(y)))$

5. Olgu predikaadid $C(x)$ – x on vanakraamikaupmees, $H(x)$ – x on aus inimene. Tõlkida eesti keelde järgmised valemid:

$\exists x \ C(x)$

$\exists x \ H(x)$

$\forall x (C(x) \rightarrow \neg H(x))$

$\exists x (C(x) \ \& \ H(x))$

$\exists x (H(x) \rightarrow C(x))$

6. Saarel elab kaks suguharu: tõesuud, kes alati tõtt räägivad, ja luiskajad, kes alati valetavad. Rändur kohtas üht pärismaalast ja kuulnud selle olevat tõesuude suguharust, palkas ta oma teenriks. Nad läksid koos edasi ja nägid eemal teist pärismaalast. Rändur saatis oma uue teenri küsima, millisesse suguharusse too pärismaalane kuulub. Naastes ütles teener, et teine pärismaalane oli öelnud enda olevat tõesuude suguharust. Millisest suguharust oli ränduri teener? (Esitada teadmus lausearvutuse valemitega.)

7. Kirjeldada loogilisi tingimusi, mis väljendavad triikrauda sisselülitava kontakti, seda ülekuumenemise eest kaitsva kontakti (termoregulaatori) ja küttespiraali oleku vahekorda, s.t selgitada välja, millised kolm olekut – triikrauda kontakti sisse- või väljalülitatus, termoregulaatori sisse- või väljalülitatus ja küttespiraali üle- või alakuumenemine – on omavahel kooskõlas ja millised mitte.

8. Esitada teadmus predikaatarvutuse valemitega:

a. inimesed, kes armastavad kasse, ei armasta koeri.

b. kui meil on juustu, siis ma teen võileibu.

c. mõningaid eestikeelseid lauseid ei saa esitada predikaatarvutuse valemitega.

11. Mittetäielik teadmus ja mittemonotoonsed loogikad

Klassikaline loogika (IJPA) lähtub eeldusest, et temas määratud aksioomide hulk on täielik ja varem tehtud järeldused jäävad kehtima uute faktide lisamisel – see on nn *monotoonsus*. Harilik inim mõtlemine on aga mittemonotoonne. Mittemonotoonsus tähendab, et uue teadmise lisamine võib blokeerida mingid varasemad järeldused ning järeldatavate faktide hulk võib seega kahaneda.

Mittemonotoonse mõtlemise formaliseerimiseks on mitmeid viise. Vaatame siin kahte – vaikimisiloorikat ja piiramise tehnikat.

Vaikimisiloorika (*default logic*, Reiter 1980) laiendab 1. järku predikaatarvutust tuletusreeglitega (nn vaikimisireeglitega, *defaults*) kujul

$$\frac{A : B}{C},$$

s.t kui A kehtib ja B pole meie teadmistega vastuolus (ei saa tuletada B eitust), siis võime järeldada, et kehtib C (vrd vaikimisiväärtustega slotid freimides). Seega ei vaadelda lauseid kui vankumatuid tõdesid, vaid nad võivad osutada vääradeks, kui me saame olukorra kohta uut informatsiooni.

J. McCarthy klassikaline näide illustreerib mittemonotoonset arutlust: kui teame, et linnud, v.a pingviinid, on lennuvõimelised ja Tweety on linn, siis võib selle teadmuse esitada järgmisel kujul.

$$\frac{\text{Lind}(x) : \text{Lennuvõimeline}(x)}{\text{Lennuvõimeline}(x)}$$

$$\text{Pingviin}(x) \rightarrow \text{Lind}(x)$$

$$\text{Pingviin}(x) \rightarrow \neg \text{Lennuvõimeline}(x)$$

$$\text{Lind}(\text{Tweety})$$

Kasutades vaikimisireegleid, võime teha järelduse: $\text{Lennuvõimeline}(\text{Tweety})$.

Kui aga lisandub uus teadmine $\text{Pingviin}(\text{Tweety})$, siis blokeeritakse esimene järeldus ja tehakse uus: $\neg \text{Lennuvõimeline}(\text{Tweety})$. Seega järelduste hulk ei kasva uute faktide teadasaamisel monotoonselt.

Praktikas kasutatakse sageli vaikimisiloorikate erijuhte: normaalset ja poolnormaalset vaikimisiloorikat.

Normaalses vaikimisiloorikas on kõik vaikimisireeglid kujul

$$\frac{A : B}{B}$$

Poolnormaalses vaikimisiloorikas on reeglitel kuju

$$\frac{A : B \& C}{B}$$

Vaikimisilooikat kasutades tuleb aga arvestada, et tehtud järeldused võivad sattuda omavahel vastuolla. Vaatame järgmist näidet (konflikti kohta, Tammet 1987).

Olgu teadmusbaasis vaikimisireeglid

1. $\frac{\text{Täiskasvanu}(x) : \text{OmabTööd}(x)}{\text{OmabTööd}(x)}$
2. $\frac{\text{Väljalangenu}(x) : \text{Täiskasvanu}(x)}{\text{Täiskasvanu}(x)}$
3. $\frac{\text{Väljalangenu}(x) : \neg\text{OmabTööd}(x)}{\neg\text{OmabTööd}(x)}$

ja fakt

Väljalangenu(Jaan).

Vaatame, kuidas TIS vastab küsimusele „Kas Jaan omab tööd?” Näeme, et vastus sõltub reeglite rakendamise järjekorrast.

- Tuletus 1:

reeglist 2 järeldub Täiskasvanu(Jaan),
reeglist 3 järeldub $\neg\text{OmabTööd}(Jaan)$.

- Tuletus 2:

reeglist 2 järeldub Täiskasvanu(Jaan),
reeglist 1 järeldub OmabTööd(Jaan).

Konflikti lahendamiseks võib kasutada järgmisi võtteid.

1) Asendada reegel 1 uue reeglga

1' $\frac{\text{Täiskasvanu}(x) : \text{OmabTööd}(x) \ \& \ \neg\text{Väljalangenu}(x)}{\text{OmabTööd}(x)}$

2) Vaikimisireeglid osaliselt järjestada: nõuda, et reeglit 3 tuleb rakendada enne kui reeglit 1.

Vaikimisilooika puuduseks on asjaolu, et teoreemide (s.t võimalike järelduste) hulk pole rekursiivselt loetletav (s.t ei leidu algoritmi, mis põhimõtteliselt alati suudaks tõestada tegelikult tõestatavad teoreemid).

Mingit ainevaldkonda täielikult kirjeldada on väga keeruline. Võib küll loetleda tõsikä kindlad teadmised, kuid võimatu on loetleda ebaõigeid teadmisi. Seetõttu on mugav teadmusbaasis määrata eranditult õiged teadmised, aga kõik, mis pole määratud (mida ei saa olemasolevatest teadmistest tuletada), lugeda ebaõigeiks. See on nn *suletud maailma eeldus* (*closed world assumption*, Reiter 1978). 1980. a võttis J. McCarthy kasutusele **piiramise** (ehk ümbritsemise, *circumscription*) **tehnik**a, eesmärgiga rakendada minimaalseid mudeleid mittetäieliku teooria jaoks. McCarthy on esitanud terve rea nn piiramisreegleid suletud maailma eelduse rakendamiseks. Vaatame siin predikaadi piiramist.

Piiramisreegel on reegel määramaks, millistel objektidel on teatav omadus P. Näiteks eespool vaadeldud misjonäride ja kannibalide probleemi puhul eeldasime vaikimisi, et paat ei leki, misjonäre ega kannibale ei tule juurde jms. Seega lähtusime suletud maailma eeldusest: maailm on täielikult kirjeldatud ning see, mille kohta pole midagi teada, ei kehti (nt kui pole öeldud, et paat lekib, siis ta ei leki). See vastab teatud omadustega objektide arvu *minimeerimisele*: kui objektilt pole ilmutatult nõutud omadust P, siis objektil *ei ole* seda omadust.

Vt näiteks „plokide maailma”, kus kehtib valem (Tamme jt 2002:283).

$\text{Plokk}(a) \vee \text{Plokk}(b)$.

Aksioom

$\forall x (\text{Plokk}(x) \rightarrow (x=a \vee x=b))$

suleb plokide maailma (mis tähendab, et ainsad plokid on a ja b).

Üldiselt, olgu P – predikaatsümbol ja A[P] – valem, mis sisaldab P. Reegli predikaadi P piiramiseks valemis A[P] mingi predikaadiga Z (P „minimeerimiseks”) võib esitada kujul:

$(\forall [P] \ \& \ \forall [Z] \ \& \ \forall x (Z(x) \rightarrow P(x))) \rightarrow \forall x (P(x) \rightarrow Z(x))$.

Selle reegli kohaselt on omadusega P need ja ainult need objektid, mis rahuldavad valemit $\forall [P]$. Reegel väljendab suletud maailma eeldust: omadus P on parajasti nendel objektidel, mille kohta on seda ilmutatult väidetud.

Vaatame veel ühte näidet. Olgu teadmusbaasis ainult fakt P(a).

Piiramine annab, et $\forall x (P(x) \equiv (x=a))$.

Kui b on objekt, mis pole a, siis järeldame, et $\neg P(b)$.

Kui aga **lisandub uus fakt** P(b), siis annab piiramine tulemuse:

$\forall x (P(x) \equiv (x=a) \vee (x=b))$.

Predikaadi laiendamise minimeerimisel on üks tähtis rakendus: ta esitab ideed, et asjad on tavaliselt sellised, nagu oodatud. Seda ideed saab formaliseerida, minimeerides predikaati, mis väljendab olukorra ebanormaalsust. Iga teadaolevale faktile lisatakse literaal $\neg Eba(\dots)$, mis sätestab, et see fakt kehtib ainult normaalses (ehk mitte-ebanormaalses) situatsioonis. Selle predikaadi laiendamise minimeerimine võimaldab arutleda vaikimisi-eelduse üle, et asjad on nii, nagu oodatud (s.t nad ei ole ebanormaalsed), ja see eeldus tehakse üksnes siis, kui võimalik (s.t ebanormaalsus on tõene ainult siis, kui see on kooskõlas faktidega). See võimaldab kasutusele võtta vaikimisireeglid, nt $\forall x (Lind(x) \ \& \ \neg Eba(x) \rightarrow Lendab(x))$ (iga mitte-ebanormaalne lind lendab).

12. Hägus teadmus. Hägusad hulgad ja loogikad

Kui vana on vana inimene: vanusega üle 50, 60 või 90 aasta? Kui pikk on pikk mees: pikkusega üle 180, 188 või 200 cm? Kui väidame, et *Jüri on pikk*, siis väljendame sellega

nn hāgusat (*fuzzy*) teadmust, sest igāuhe ettekujutus sellest, missugusest pikkusest alates saame kōnelda pikast mehest, vōib olla pisut erinev. Vaevalt peab keegi meist 170 cm pikkust meest pikaks ja vaevalt vaidleb keegi vastu, et 200 cm pikkune on pikk mees, kuid nende pikkuste vahel asub n.ō „hall ala”, kus arvamused vōivad lahku minna.

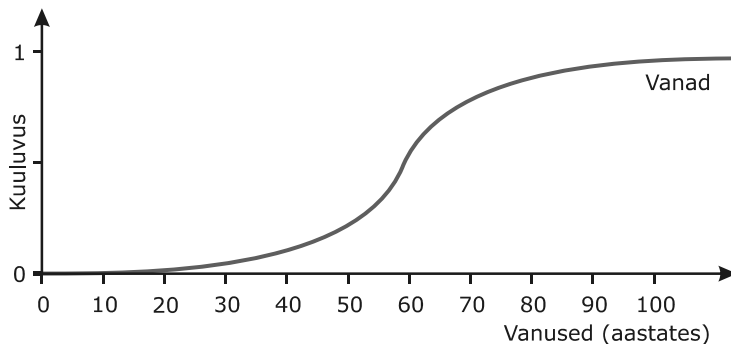
Hāgusa teadmuse esitusele pani aluse Lofti A. Zadeh 1965. a, tuues sisse **hāgusa hulga** (*fuzzy set*) mõiste, eesmärgiga formaliseerida loomuliku keele (ebatāpseid) vāiteid. Zadeh vōttis kasutusele nn lingvistilised muutujad, mille vāartusteks on hāgusad hulgad. Nāiteks *pikkus* on lingvistiline muutuja, mille vāartusteks vōivad olla *lūhike*, *keskmine*, *pikk*, *ūlipikk* jms.

Hāgus hulk erineb tavalisest hulgast selle poolest, et iga objekti puhul nāidatakse, millisel määral (0 kuni 1) kuulub objekt hāgusasse hulka (0 – kindlasti ei kuulu; 1 – kindlasti kuulub).

Olgu X nn tāielik (ehk universaalne) hulk, mis hōlmab kogu ainevaldkonna. Hulga X hāgus alamhulk A defineeritakse *kuuluvusfunktsiooni* (*membership function*) μ_A kaudu:

$$\mu_A: X \rightarrow [0,1].$$

Nāiteks kui universaalseks hulgaks on inimeste vanuste hulk, siis vōib kuuluvusfunktsiooni abil defīneerida hāgusa hulga Vanad, nagu seda on tehtud joonisel 12.1.



Joonis 12.1. Hāgusa hulga Vanad kuuluvusfunktsioon

Kuuluvusfunktsioon on (tavalise) hulga karakteristliku funktsiooni ūldistus.

Universaalne hulk X ei tarvitse olla pidev. Vanuste (aastates) puhul vōib selliseks hulgaks olla ka nt

$$X = \{0, 10, 20, \dots, 100\}.$$

Siis vōib hāgusa hulga Vanad defīneerida jārgmiselt:

$$\text{Vanad} = \{0/0 + 0/10 + 0/20 + 0/30 + 0,1/40 + 0,4/50 + 0,8/60 + 1/70 + 1/80 + 1/90 + 1/100\}.$$

Sūmbol / eraldab kuuluvusmāāra ja elemendi vāartust ning sūmbol + elemente. Tavaliselt jāetakse kirjutamata need elemendid, mille kuuluvusmāār on 0, seega vōib sellesama hāgusa hulga esitada ka kujul

Vanad = $\{0,1/40 + 0,4/50 + 0,8/60 + 1/70 + 1/80 + 1/90 + 1/100\}$.

Sõltuvalt rakendusest võib TIS-i loomisel kuuluvusfunktsiooni väärtused määrata kas üheainsa inimese või inimeste grupi arvamuste põhjal. Ekspertsüsteemis koostatakse kuuluvusfunktsioon vastavalt eksperdi (ekspertide grupi) arvamusele, keda selles süsteemis modelleeritakse. Sellised otsustused ei ole lihtsalt jah või ei. Kuigi jaatava ja eitava otsuse jaoks saaks kehtestada läviväärtused, on küsitav, kui jäik peaks olema läve väärtus. Nt kas isikule antakse pangalaenu, kui tema kuusissetulek on 999,99 eurot, aga nõutav lävi on 1000 eurot?

Tehted hägusate hulkadega

Hägusatele hulkadele saab üle kanda kõik hulgateoreetilised tehted, kasutades kuuluvusfunktsiooni:

hägusa hulga A *täiend*: $\neg A$, kus $\mu_{\neg A}(x) = 1 - \mu_A(x)$;

hägusate hulkade A ja B *ühend*: $A \vee B$, kus $\mu_{A \vee B}(x) = \mu_A(x) \vee \mu_B(x) = \max(\mu_A(x), \mu_B(x))$;

ühisosa: $A \wedge B$, kus $\mu_{A \wedge B}(x) = \mu_A(x) \wedge \mu_B(x) = \min(\mu_A(x), \mu_B(x))$, $\forall x \in X$.

Kaks samal universaalsel hulgal defineeritud hägusat hulka on võrdsed, kui nende kuuluvusfunktsioonid on võrdsed.

Võib defineerida ka hägusad suhted (*fuzzy relation*) hulkade vahel (nt „palju suurem kui”, „peaaegu võrdne” jms).

Olgu X ja Y universaalsed hulgad. **Hägusaks suhteks** R hulkade X ja Y vahel nimetatakse nende hulkade Cartesiusse korrutise $X \times Y$ hägusat alamhulka R ($R \subseteq X \times Y$), kus kuuluvusfunktsioon

$$\mu_R : X \times Y \rightarrow [0, 1]$$

näitab suhte tugevust X ja Y vahel.

Näide 12.1. Hägus suhe „samasugune”.

Olgu

$X = \{\text{hobune, eesel}\}$,

$Y = \{\text{muul, lehm}\}$.

Hägus suhe „samasugune”:

$R = \text{Samasugune} = \{0,8/(\text{hobune, muul}) + 0,4/(\text{hobune, lehm}) + 0,9/(\text{eesel, muul}) + 0,5/(\text{eesel, lehm})\}$.

Selle võib esitada ka maatriksina:

| | | |
|--------|--|------|
| | muul | lehm |
| hobune | $\begin{bmatrix} 0,8 & 0,4 \\ 0,9 & 0,5 \end{bmatrix}$ | |
| eesel | | |

Võib defineerida ka hägusate suhete kompositsiooni. Olgu hägusad suhted

$$R \subseteq X \times Y \text{ ja } S \subseteq Y \times Z.$$

Hägusate suhete kompositsiooniks $R \circ S$ nimetatakse hägusat suhet ($\subseteq X \times Z$), kus

$$\mu_{R \circ S}(x,z) = \bigvee_{y \in Y} \{ \mu_R(x,y) \wedge \mu_S(y,z) \} \quad (x \in X, z \in Z).$$

Näide 12.2. Hägusate suhete kompositsioon.

Olgu universaalsed hulgad $X=Y=Z=\{1,2,3,4\}$

ning hägusad hulgad

$$F \subseteq X = \text{VÄIKESED} = \{1/1 + 0,6/2 + 0,1/3\},$$

$$G \subseteq Y = \text{SUURED} = \{0,1/2 + 0,6/3 + 1/4\},$$

$$\neg F = \text{MITTEVÄIKESED} = \{0,4/2 + 0,9/3 + 1/4\}$$

$$H = \text{VÄGASUURED} = \{0,5/3 + 1/4\}.$$

Esitame hägusa suhte $R \subseteq X \times Y$ (ligikaudne väljend „kui x on väike, siis y on suur”) järgmise maatriksina:

$$\begin{bmatrix} 0 & 0,1 & 0,6 & 1 \\ 0 & 0,1 & 0,6 & 0,6 \\ 0 & 0,1 & 0,1 & 0,1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

ja hägusa suhte $S \subseteq Y \times Z$ („kui y on mitteväike, siis z on väga suur”) maatriksina:

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0,4 & 0,4 \\ 0 & 0 & 0,5 & 0,9 \\ 0 & 0 & 0,5 & 1 \end{bmatrix}.$$

Siis hägusate suhete R ja S kompositsioon on esitatav järgmisel viisil (meenutab maatriksite korrutamist, kuid korrutamise asemel on miinimumi ja liitmise asemel maksimumi leidmine):

$$\begin{bmatrix} 0 & 0,1 & 0,6 & 1 \\ 0 & 0,1 & 0,6 & 0,6 \\ 0 & 0,1 & 0,1 & 0,1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \circ \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0,4 & 0,4 \\ 0 & 0 & 0,5 & 0,9 \\ 0 & 0 & 0,5 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0,5 & 1 \\ 0 & 0 & 0,5 & 0,6 \\ 0 & 0 & 0,1 & 0,1 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

Hägusloogikad

Klassikaline loogika piirub kahe tõeväärtusega (tõene ja väär). Seda on lihtne modelleerida ja järeldused sellises süsteemis on täpsed. Hägusloogikaid aga kasutatakse

„terve mõistuse” arutluste formaliseerimiseks, kus eeldustest, mis ei tarvitse olla päris täpsed, tehakse järeldusi, mis võivad samuti olla ebatäpsed.

Hägasloogikas võib kasutada hägasaid predikaate (nt Vana, Pikk) ja hägasaid kvantoreid (nt Enamus, Paljud).

Kasutatakse mitmesuguseid erinevaid hägasloogikaid, kuid need võib jaotada kahte põhiklassi: 1) tõeväärtused lõigul $[0,1]$ pole hägasad, 2) tõeväärtused on hägasad (nt „peaaegu tõene”, „pigem tõene kui väär” jne).

Loogilised tehted defineeritakse järgmisel viisil.

Olgu A ja B hägasad väited ning $\tau(A)$ ja $\tau(B)$ nende hägasad tõeväärtused.

Eitus: $\tau(\neg A) = 1 - \tau(A)$.

Konjunktsioon: $\tau(A \cap B) = \tau(A) \wedge \tau(B) = \min(\tau(A), \tau(B))$.

Disjunktsioon $\tau(A \cup B) = \tau(A) \vee \tau(B) = \max(\tau(A), \tau(B))$.

Implikatsioon: $\tau(A \rightarrow B) = 1 \wedge (1 - \tau(A) + \tau(B)) = \min(1, 1 - \tau(A) + \tau(B))$.

Hägasat kvantorit Q ja hägasat predikaati F sisaldava väite $Qy F(y)$ tõeväärtus määratakse järgmiselt.

- Hägas kvantor Q on defineeritud kuulumisfunktsiooniga $\mu_Q : [0, 1] \rightarrow [0, 1]$.
- Olgu $Y = \{y_1, \dots, y_n\}$ objektide hulk ja olgu sellel hulgal defineeritud hägas predikaat, kusjuures tõeväärtused $\tau(F(y_i)) = \mu_F(y_i) \quad \forall i=1, \dots, n$.
- Väite $Qy F(y)$ tõeväärtuse $\tau(Qy F(y))$ määramiseks tehakse järgmist:

1. arvutatakse $r = (1/n) \sum_{i=1}^n \mu_F(y_i)$,
2. tõeväärtuseks võetakse arvu $r \in [0,1]$ kuulumise määr hägasasse hulka Q:

$$\tau(Qy F(y)) = \mu_Q(r).$$

Näide 12.3. Hägusa väite tõeväärtus.

Olgu universaalne hulk

$Y = \text{sõbrad} = \{A, B, C\}$

ja sellel defineeritud hägas predikaat

$F = \text{TõelisedSõbrad} = \{0,1/A + 0,6/B + 0,8/C\}$.

Olgu Q hägas kvantor Enamus, mis on defineeritud kuulumisfunktsiooniga:

$$\mu_Q(r) = \begin{cases} 1, & \text{kui } r \geq 0,8 \\ 2r - 0,6, & \text{kui } 0,3 < r < 0,8 \\ 0, & \text{kui } r \leq 0,3 \end{cases}$$

Hägusa väite $Qy F(y)$ (*enamus sõpru on tõelised sõbrad*) tõeväärtuse määrame järgmiselt:

1. $r = 1/3 \cdot (0,1 + 0,6 + 0,8) = 0,5$
2. $\mu_Q(0,5) = 2 \cdot 0,5 - 0,6 = 0,4$

Seega

$$\tau(\text{QyF}(y)) = \mu_Q(r) = 0,4.$$

Hägasloogika rakendusi (Giarratano, Riley 2004)

- Robotite juhtimine (Hirota, Fuji Electric, Toshiba, Omron).
- Temperatuurikõikumiste ärahoidmine õhukonditsioneerisüsteemis (Mitsubishi, Sharp).
- Bussisõiduplaanide koostamise optimeerimine (Toshiba, Nippon-System, Keihan-Express).
- Maavärinate ennustamine (Jaapani seismoloogiasinstituut).
- Käekirja tuvastamine taskuarvutis (Sony).
- Kõnetuvastus (Hitachi).
- Tuumareaktori ohutuse kontroll (Hitachi, Bernard, Nuclear Fuel Div.).

Harjutusülesanded

1. Olgu antud universaalne hulk (neidude pikkuste hulk)

$$X = \{1.50, 1.60, 1.70, 1.80, 1.90\}.$$

Defineerida selle hulga hägasad alamhulgad PikadNeiud ja LühikesedNeiud.

Leida

$$\neg\text{PikadNeiud}$$

$$\text{PikadNeiud} \cup \text{LühikesedNeiud}$$

$$\text{PikadNeiud} \cap \text{LühikesedNeiud}$$

2. Antud on hägasad hulgad

$$A = \{0, 1/1 + 0, 2/2 + 0, 3/3\}$$

ja

$$B = \{0, 2/1 + 0, 3/2 + 0, 4/3\}.$$

- a. Kas hulgad A ja B on võrdsed? Põhjendage.
- b. Leidke mõlema hulga täiendid.
- c. Leidke hulkade A ja B ühend.
- d. Leidke hulkade A ja B ühisosa.

3. Olgu universaalne hulk U – seda kursust kuulavate üliõpilaste hulk või (omal valikul) mingi alamhulk (nimed). Defineerida selle hulga kolm hägasat alamhulka: Pikad, Keskmised, Lühikesed ning seejärel (kasutades tehteid hägasate hulkadega) Mittepikad, MittepikadJaKeskmised, LühikesedVõiKeskmised.

4. Olgu antud universaalne hulk $U = \{1, 2, 3, 4\}$ ja selle kaks hägasat alamhulka:

1) $V = \{1/1 + 0, 8/2 + 0, 6/3\}$ – väikeste arvude hulk

2) $S = \{0, 6/3 + 0, 8/4 + 1/4\}$ – suurte arvude hulk.

Kasutades tehteid hägasate hulkadega, defineerida: mitteväikesed arvud, mittersuured arvud; arvud, mis on mitteväikesed ja mittersuured; arvud, mis on väikesed või suured; arvud, mis on väikesed ja suured.

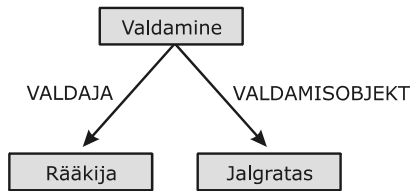
13. Teadmuse esituse meetodite võrdlus

Nagu eelnevast näha, kasutatakse tehisintellektis mitmesuguseid erinevaid teadmuse esituse meetodeid, mille hulgast sobiva valik sõltub nii probleemivaldkonnast kui ka loodava TIS-i eesmärkidest. Joonis 13.1 illustreerib ühe ja sama teadmuse esitust erinevatel meetodidel.

a) IJPA valem

$$\exists x \exists y (\text{Valdamine}(x) \ \& \ \text{Valdaja}(\text{Rääkija}, x) \ \& \ \text{Valdamisobjekt}(y, x) \ \& \ \text{Jalgratas}(y))$$

b) Semantiline võrk



c) Freim



Joonis 13.1. Fakti „Mul on jalgratas” esitus kolmel erineval viisil

Igal meetodil on oma eelised, aga ka puudused, täiesti ideaalset teadmuse esituse formalismi pole suudetud välja töötada (ja kas seda ongi vaja?).

On sõnastatud rida nõudeid, mida hea teadmuse esituse meetod peaks rahuldama nii kasutaja, teadmusbaasi kui ka tehisintellektisüsteemi kui terviku seisukohalt (Ohsuga 1989).

I. Inimese seisukohalt:

- 1) deklaratiiivne,
- 2) võimaldab tuletusi,
- 3) saab tõlkida loomulikku keelde,
- 4) saab tõlkida arvutikeelde.

II. Teadmusbaasi seisukohalt: võimaldab

- 5) kirjeldada teadmusbaasi struktuuri,
- 6) dünaamiliselt töödelda teadmusbaasi,
- 7) kontrollida teadmusbaasi mittevasturääkivust.

III. Tehisintellektisüsteemi juhtimise seisukohalt: võimaldab esitada

- 8) erinevaid ainevaldkondi ja esitust juhtida,
- 9) mitmekihilisi teadmisi,
- 10) hägusaid teadmisi.

Kõige paremini vastavad nendele nõuetele loogikal põhinevad meetodid. Põhiline vajaka jäämine on piisavalt arendatud andmestruktuuri puudumine.

Teadmuse esitus on jätkuvalt tehisintellekti üks põhiküsimusi, mida näitab kas või see, et rahvusvahelistel konverentsidel on teadmuse esituse meetodid alati tähelepanu all. Teadmuse esituse aktuaalsed uurimisküsimused on

- ontoloogiad,
- mittemonotoonsed loogikad,
- teadmusbaasi juhtimine, kooskõlalatus (*consistency*) ja täielikkus.

Ontoloogia on metateadmus, mis kirjeldab kõike seda, mis probleemivaldkonna kohta on teada. Ideaalsel juhul tuleks ontoloogiast kirjeldada formaalselt, et teadmuse mitteköölalisuse ja mittetäielikkuse väljaselgitamine oleks võimalikult lihtne. Ontoloogia peab olema konstrueeritud enne ekspertsüsteemi rakendamist, sest muidu tuleb reegleid alati uuesti kontrollida, kui valdkonna kohta on teada saadud rohkem infot. See aga tõstab kulutusi, arendamiseks vajaminevat aega ja vigu. Ontoloogiate konstrueerimiseks on olemas arvukalt vaba- ja kommertstarkvara.

Eraldi on uuritud mitmesuguseid nn üldisi ontoloogiaid.

- Olemid (*entity*)

Enamus teadmuse esituse skeeme lähtuvad eeldusest, et maailm koosneb teatavatest üksustest (olemitest), millel on unikaalne ja muudetamatu identsus, eksisteerimise aeg ja omavahelised seosed. Näiteks uuris P.J. Hayes 1985. a materiaalseste substantside ontoloogiast, kus üksusi (nt 1 liiter vett, 10 grammi soola) saab liita, mille tulemusel tekib uus üksus.

- Aeg
- Ruum
- Põhjuslikkus
- Agendid

Käsitletakse intelligentseid (tarkvara)agente, millel on uskumused, soovid, eesmärgid ja kavatsused ning mis on võimelised sooritama tegevusi (James F. Allen 1981).

- Eksistents

Käsitletakse erinevaid eksisteerimise viise: füüsiline, abstraktne (nt number 7), mitteeksistents (nt ärajäänud reis), võimatu eksistents (ruutjuur -1-st reaalarvude valdkonnas).

Teadmusbaasi kooskõlalikus ja täielikkus

Kooskõlalikus saavutamiseks tuleb

1. Kasutada tõehoolde (*truth maintenance*) mehhanisme: korraldatakse seoste võrk, mis esitab loogilisi suhteid faktide vahel. Kui avastatakse vasturääkivus, siis määratakse selle mehhanismiga vasturääkivuse allikas ja leitakse faktid, mis tuleb üle kontrollida. Tõehoolde vahendid võimaldavad TIS-il hallata informatsiooni faktide vahel loogiliste sõltuvuste kohta (nt fakt C on tuletatud faktidest A ja B), mida hiljem saab kasutada (mittemonotoonseid järeldusi sisaldava) teadmusbaasi esialgse seisundi taastamisel pärast uute faktide lisamist (kui teatavad varem tehtud järeldused enam ei kehti).
2. Kasutada loogikaid, mis ilmutatult keelavad implikatsiooni paradoksid. Näiteks **relevantsusloogikad** piiravad tõestamise mõistet: eeldusest A järeldub B, kui eksisteerib B selline tõestus, mis tegelikult kasutab mingil sammul eeldust A. Relevantsusloogikad pakuvad huvi nii arvutuslikult kui ka teadmuse ja uskumuse formaalse mudelina.

Reaalsed teadmusbasisid on mittetäielikud, sest teadmus ainevaldkonna kohta on puudulik (teatavate objektide kohta pole veel midagi teada) või nõrgalt tõestatud. Tuletusprotseduur võib olla mittetäielik ka kokkuvõiu kaalutlustel. Mittetäielikku teadmust saab käsitleda, kasutades predikaatarvutust ja selle mitte-monotoonseid laiendusi.

Teadmusbasiside loomise kitsaskoht on **teadmuse hankimine**. Olemasolevad teadmuse hankimise vahendid toetavad mitmesuguseid teadmuse hankimise mehhanisme, sh reeglite õppimine toorandmetest ja teadmuse ekstraheerimine loomulikus keeles kirjutatud tekstidest.

Hindamisvahendid aitavad kasutajal süstemaatilisel viisil hinnata teadmusbasisis olevaid teadmisi.

Dokumenteerimisvahendid kirjeldavad teadmusbasisi sisu loomulikus keeles.

Juhtimisvahendid on teadmusbasisi arendamiseks, vanade teadmiste unustamiseks, resümeeerimiseks jms.

III. PROBLEEMILAHENDUS

14. Probleemilahendusmeetodite liigitus

Probleemilahendus (problem solving) on kõrvuti teadmuse esitusega tehisintellekti üks põhiküsimusi. Mitmel juhul määrab kasutatav teadmuse esituse meetod ka võimalikud probleemilahendusmeetodid, kuid sõltumata teadmuse esitusest on paljude probleemilahendusmeetodite aluseks esitus olekute ruumina. Probleemilahendus kujutab endast siis **otsingut** olekute ruumis: liikumist algolekust lõppolekusse, rakendades reegleid. Nagu nägime eespool, on üks hea viis reeglite esitamiseks produktsioonide süsteem.

Iga otsinguprotsessi saab vaadata kui liikumist suunatud graafis, mille tipud vastavad olekutele ning kaared üleminekutele olekute vahel. Tavaliselt ei konstrueerita ülesande lahendamisel mitte kogu otsimisgraafi, vaid ehitatakse välja ainult need osad, mida otsustatakse läbida.

On terve rida üldisi küsimusi, mis otsingul tuleb lahendada.

1) Otsingu suund: kas päripidi (lähtudes algolekust ja liikudes lõppoleku poole) või tagurpidi (lähtudes lõppolekust ja liikudes algoleku poole).

Siin on kaks võimalust:

a) otsetuletus: alustada puu/graafi läbimist/konstrueerimist juurtipust, mis vastab lähteolekule. Rakendada kõiki produktsioone, mille vasak pool vastab lähteolekule. Jätkata analoogilisel viisil, kuni saavutatakse lõppolek.

b) Pöördtuletus: alustada puu/graafi konstrueerimist lõppolekust, millele rakendada kõiki selliseid produktsioone, mille parem pool vastab lõppolekule. Jätkata analoogilisel viisil, kuni saavutatakse algolek.

Võimalik on ka korraga kahesuunaline otsing.

See, kumb suund valida, sõltub vähemalt kolmest asjaolust:

a) mida on rohkem: kas võimalikke alg- või lõppolekuid? Parem on alustada vähemast arvust. Nt kodust võõrasse kohta minekut on kergem planeerida lõppolekust lähtudes, sest mida lähemale kodule, seda rohkem on tuttavaid kohti (mida saab samastada kodusolekuga).

b) Kummas suunas on *hargnemistegur*³² suurem? Tuleks valida see suund, kus hargnemistegur on väiksem. Nt teoreemi tõestamine „tagant ette”.

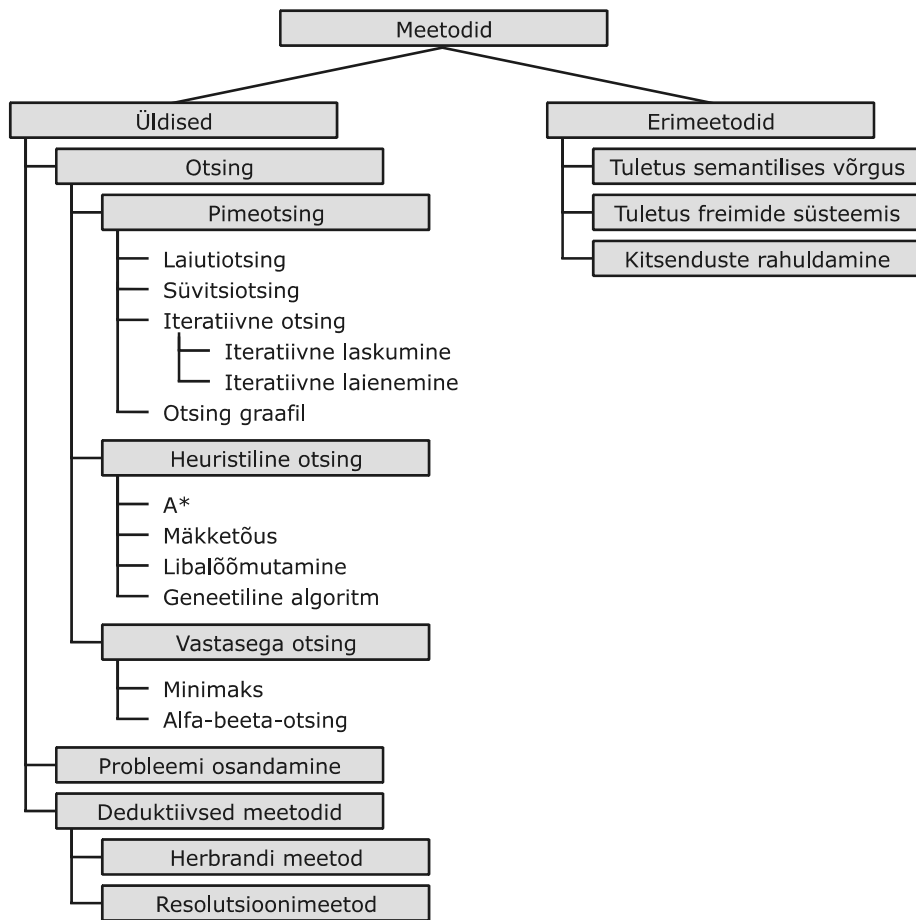
c) Kas programm peab suhtlema kasutajaga? Kui jah, siis on soovitatav valida see tuletussuund, mis on inimesele loomulikum. Ekspertsüsteem peab suutma oma otsuseid kasutajale põhjendada, nt vastata küsimusele „Miks tuleks teha patsiendiga test A?”.

2) Otsinguprotsessi topoloogia: kas esitada olekute ruum graafina või puuna?

³² Puu hargnemistegur – maksimaalne ühest tipust väljuvate kaarte arv.

Kui otsustada puu kasuks, siis võib olla vaja üht ja sedasama olekut esitada mitu korda ja seega teha liigseid jõupingutusi otsingu ühesugusel jätkamisel. Puul otsingu saab asendada graafil otsinguga, aga siis tuleb võtta kasutusele täiendav „arvepidamine”: kui mingi tipu läbimisel ehk avamisel (s.t tema vahetute alluvate genereerimisel) genereeritakse uus tipp (läbitava tipu vahetu alluv), siis kontrollida, kas ta juba esineb graafis:

- kui ei, siis lisada uus tipp (koos uue kaarega);
- kui jah, siis suunata uus kaar äsja läbitud tipust varem olemasolevasse tippu.



Joonis 14.1. Probleemilahendusmeetodite liigitus

3) Kuidas esitada olekuid (graafi tippe)? Võib kasutada kas täielikke või osalisi olekukirjeldusi.

a) Täielike olekukirjelduste kasutamise puhul tuleb igas olekus näidata ka tegevused, mida teha, kui käiku on vaja tühistada (otsingul sellesse olekusse tagasi pöörduda).

b) Kirjeldada täielikult ainult lähteolek. Igas muus tipus säilitada ainult muudatused, mida tuleb teha eelmise olekuga võrreldes. Kui otsingul on vaja sellesse tippu tagasi pöörduda, siis täieliku olekukirjelduse saamiseks läbida tee jooksvast olekust tagasi sellesse olekusse ja „korjata kokku” muudatused, mis tuleb tühistada.

4) Kuidas kergesti leida sobiv reegel, mida rakendada? Siin on mitmeid võtteid.

a) Märksõnastamine: sama vasaku poolega produktsioonidele omistada sama märksõna. Eelduseks on, et reeglid ei sisalda muutujaid.

b) Kui reeglid siiski sisaldavad muutujaid, siis reegli sobivuse kontrollimiseks paigutada jooksvas olekus kehtivaid fakte produktsioonides muutujate asemele, et otsustada, kas reegel on rakendatav või ei.

c) Kui produktsioonide vasakud pooled vastavad ainult ligikaudu jooksvale olekule, siis on sobiva reegli valik raskendatud. Selline olukord esineb näiteks kõnetuvastuses, kus häälikute varieeruvus on suur.

5) Kas kasutada heuristikuid või mitte?

Heuristiku eesmärk on juhtida otsingut kasulikus suunas. Paljude ülesannete puhul on aga heuristilise funktsiooni väärtuste arvutamise hind kõrgem kui otsingu enda hind ja seetõttu on vaja leida kompromiss. Heuristilise otsingu korral püütakse avatud tippe olekute graafis järjestada nii, et otsinguprotsess jätkuks perspektiivikamates suundades.

Probleemilahendusmeetodid võib liigitada üldisteks ja erimeetoditeks. Joonis 14.1 esitab kokkuvõtlikult meetodid, mida käsitleme selles peatükis.

III.A. Otsing

15. Pimeotsing puul

Termin *pimeotsing* (*blind search*) tähistab otsingumeetodeid, mis kasutavad ainult probleemi kirjelduses leiduvat infot ning millele pole kättesaadav täiendav info. Kõik, mis nad suudavad teha, on genereerida iga tipu järglasi ja tunda ära, kas tegu on lõppolekule vastava tipuga või mitte. Seevastu otsingut, mis suudab kindlaks teha, et üks olek on „perspektiivikam” kui teine, nimetatakse informeeritud otsinguks e *heuristiliseks otsinguks*. Otsingumeetodeid eristab üksteisest see, millises *järjekorras* tippe läbitakse.

Jaotises 5 tutvusime kahe pimeotsingu algoritmiga: need on laiuti- ja süvitsiotsing (joonised 5.1 ja 5.2). Hindame nüüd nende mälu- ja ajavajadust. Mäluvajadus on võrdeline nimestikus L paiknevate (läbiuurimiseks väljavalitud, üheaegselt säilitatavate) tippude arvuga, ajavajadus on võrdeline läbiuuritavate tippude arvuga.

Eeldame, et otsing toimub lõplikul puul, *hargnemisteguriga* **b**, kus on üksainus lõppolekule vastav tipp **g** sügavusel **d**.

Laiutiotsing

Laiutiotsing on täielik, s.t kui juurele kõige lähem (kõige väiksema sügavusega) lõpptipp asub sügavusel d , siis laiutiotsing avastab selle, olles enne läbinud kõrgemal asuvad tipud (eeldusel, et hargnemistegur b on lõplik). Muidugi ei tarvitse vähima sügavusega lõpptipp olla parim.

Mäluvajadus on võrdeline nimestikus L olevate tippude arvuga, s.t eksponentsiaalne d suhtes: $O(b^d)$.

Ajavajadus on võrdeline läbiuuritavate tippude arvuga (vt tabel 15.1).

Tabel 15.1. Läbiuuritavate tippude arv laiutiotsingul sügavustel 0 kuni $d-1$

| Sügavus | Tippude arv |
|---------|--------------|
| 0 | 1 (puu juur) |
| 1 | b |
| 2 | $b \cdot b$ |
| ... | ... |
| $d-1$ | b^{d-1} |

Kokku kuni sügavuseni $d-1$ kaasa arvatud on läbiuuritavate tippude arv $1 + b + b \cdot b + \dots + b^{d-1} = (b^d - 1)/(b - 1)$. Sügavusel d on parimal juhul vaja läbida üksainus tipp, halvimal juhul aga kõik tipud, s.t b^d , seega keskmiselt $(1 + b^d)/2$ tippu. Parima ja halvima juhu keskmine sobib hinnanguks, sest otsingupuud on sümmeetriline. Seega **keskmine ajavajadus** on $(b^{d-1})/(b-1) + (1 + b^d)/2 \approx O(b^d)$ – eksponentsiaalne d suhtes.

Binaarse puu korral ($b=2$) on ajavajadus $3/2 \cdot 2^d$.

Mälu on tõsisem probleem kui aeg. Näiteks eeldame, et 1 sekundis genereeritakse 10 000 tippu ja iga tipu hoidmiseks mälu kulub 1000 baiti. Kui puu hargnemistegur on $b=10$, siis sügavusel 8 kuluks kõikide tippude läbimiseks aega 31 tundi (mis tähtsa probleemi korral ei ole liiga palju) ning mälu 1 terabait (aga põhimälu ei ole tavaliselt nii mahukas). Siiski pole ka aeg vähetähtis. Näiteks kui lahend asub sügavusel 12, siis laiutiotsinguga kulub aega 35 aastat.

Üldiselt kehtib järeldus, et eksponentsiaalse keerukusega probleeme ei saa lahendada pimeotsinguga.

Süvitsiotsing

Mäluvajadus on võrdeline nimestikus L olevate tippude arvuga. Igal sügavusel 1 kuni $d-1$ tuleb säilitada $b-1$ tippu, sügavusel d aga b tippu, seega kokku $d(b-1) + 1 \approx O(d)$, s.t mäluvajadus on lineaarne d suhtes.

Ajavajadus on võrdeline läbiuuritavate tippude arvuga.

Parimal juhul tuleb uurida igal sügavusel (0 kuni d) üksainus tipp (esimene), s.t kokku d+1 tippu.

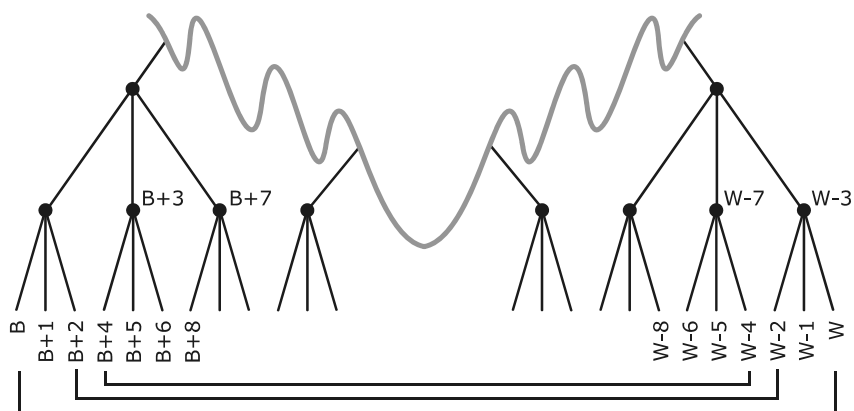
Halvimal juhul tuleb igal sügavusel läbi uurida kõik tipud (tabel 15.2), kokku $1+b+ \dots + b^d = (b^{d+1}-1)/(b-1)$ tippu.

Tabel 15.2. Läbiuuritavate tippude arv süvitsiotsingul sügavustel 0 kuni d halvimal juhul

| Sügavus | Tippude arv |
|---------|--------------|
| 0 | 1 (puu juur) |
| 1 | b |
| 2 | b·b |
| ... | ... |
| d-1 | b^{d-1} |
| d | b^d |

Parima ja halvima juhu keskmine:

$((d+1)+((b^{d+1}-1)/(b-1))) / 2 = (bd+b-d-1+b^{d+1}-1) / (2(b-1)) \approx O(b^d)$, s.t eksponentsiaalne. Parima ja halvima juhu keskmine sobib hinnanguks, sest otsingupuud on sümmeetriline (vt näide joonisel 15.1).



b = 3

B - parimal juhul kontrollitavate tippude arv

W - halvimal juhul kontrollitavate tippude arv

Otsingupuud sümmeetrilisuse tõttu võib leida keskmise $\frac{B + W}{2}$

Joonis 15.1. Süvitsiotsingu ajavajadus puul, hargnemisteguriga 3

Binaarse puu korral on süvitsiotsingu ajavajadus 2^d .

Seega on laiutiotsing kallim kui süvitsiotsing nii mälu kui ka aja mõttes (ajavajaduse suhte kohta vt tabel 15.3).

Tabel 15.3. Laiuti- ja süvitsiotsingu ajavajaduste suhe sõltuvalt puu hargnemistegurist

| Hargnemistegur b | Ajavajadus laiutiotsingul / ajavajadus süvitsiotsingul |
|--------------------|---|
| 2 | 1,5 |
| 3 | 1,3 |
| 5 | 1,2 |
| 10 | 1,1 |
| 25 | 1,04 |
| 100 | 1,01 |

Süvitsiotsingul võivad aga tekkida raskused, mida laiutiotsing suudab vältida:

- kui otsingupuus on mitu lõpptippu ja tuleb leida selline, mis asub kõige väiksemal sügavusel, siis süvitsiotsing ei tarvitse seda leida, kuid laiutiotsing leiab.
- Süvitsiotsingul on oht, et otsing võib suunduda valesse harru. Halvimal juhul genereerib süvitsiotsing kõik $O(b^m)$ tippu, kus m on otsingupuus maksimaalne sügavus (m võib olla palju suurem kui madalaima lõppolekule vastava tipu sügavus d ja võib olla lõpmatu puu korral lõpmatu). Selle ärahoidmiseks võib kehtestada teatava piirsügavuse, mille ammendamisel jätkub süvitsiotsing eelmiselt sügavuselt.

Süvitsiotsingu üks tuntud variant, nn **tagurdamine** (*backtracking*), kasutab vähem mälu kui tavaline süvitsiotsing. Tagurdamise puhul genereeritakse tipu läbimisel korraga üks-ainus järglane. Igas osaliselt avatud tipus peetakse meeles, millist järglast genereerida järgmisena. Nii on vaja $O(bm)$ asemel ainult $O(m)$ mälu (m on puu maksimaalne sügavus). Tagurdamine kasutab veel teistki mälu kokkuhoiu võtet: järglase genereerimiseks modifitseerib jooksva oleku kirjeldust. Seega on vaja meeles pidada ühtainust olekukirjeldust ja $O(m)$ tegevust. Et see toimiks, peame suutma tühistada kõik modifikatsioonid, kui me pöördume tagasi järgmist järglast genereerima. Need tehnikad on edukad eeskätt pikkade olekukirjeldustega probleemide puhul.

Tekib küsimus: kas leidub selline otsingumeetod, millel oleks süvitsiotsingu mäluvajadus (lineaarne) ja laiutiotsingu täitmisomadused (leiaks vähima kaarte arvuga tee lõpptipuni)? Ühe niisuguse meetodi töötas välja R. Korf 1985. a ja nimetas selle *iteratiivseks laskumiseks* (*iterative deepening*). Teise meetodi, nn *iteratiivse laienemise* (*iterative broadening*) töötasid välja M. Ginsberg ja W. Harvey 1990. a.

Iteratiivne laskumine on süvitsiotsing, mis kasutab kunstlikku limiiti sügavusele ja suurendab seda limiiti järk-järgult, kuni lahend on leitud. Iteratiivne laienemine on samuti süvitsiotsing, mis aga kasutab kunstlikku limiiti laiusele ja suurendab seda limiiti järk-järgult, kuni lahend on leitud. Eeldame siin endiselt, et otsing toimub puul, hargnemis-

teguriga b , kus on üksainus lõppolekule vastav tipp sügavusel d . Mõlemad algoritmid kasutavad avatud tippude nimestikku, mida tähistame L .

Iteratiivne laskumine

1. Olgu $c=1$ (jooksev limiit sügavusele).
2. Olgu L lähtetippude nimestik.
3. Olgu n esimene tipp nimestikus L . Kui L on tühi, siis $c:=c+1$, mine 2.
4. Kui n on lõpptipp, siis lahend leitud, lõpeta.
5. Vastupidisel juhul kustuta n nimestikust L . Kui tipu n sügavus on väiksem kui c , lisa nimestiku L algusesse n kõik vahetud alluvad. Mine 3.

Võrdlus laiutiotsinguga:

- **sarnane** laiutiotsinguga: kui sügavamale enam minna ei saa (sammul 5) ning osutub, et L on tühi (sammul 3), siis suurendatakse sügavuse limiiti 1 võrra ja alustatakse uuesti lähtetipust.
- **Erinev** laiutiotsingust: selle asemel, et säilitada vahepealsetel sügavustel kõiki tippe (nagu laiutiotsingul), genereeritakse need tipud uuesti parajasti sel hetkel, kui neid on vaja kontrollida.

Võrdlus süvitsiotsinguga:

- **sarnane** süvitsiotsinguga: iga iteratsioon on süvitsiotsing:
 - algul otsitakse puul kunstliku sügavusega 1, nii et ei kontrollita ühtki tippu allpool seda sügavust.
 - Kui lõpptippu sügavusel 1 ei leita, siis pöörduakse tagasi ja otsitakse sügavustel 2, 3 jne. Iga selline „iteratiivne otsing” toimub süvitsi.
- **Erinev** süvitsiotsingust: kui süvitsiotsing suundub valesse harru, siis ta ei avasta väiksemal sügavusel olevat lõpptippu (või töötab lõpmatuseni lõpmatu otsingupuul korral). Iteratiivne süvendamine aga garanteerib, et lõpptipp leitakse.

Mäluvajadus. Kuna iteratiivne laskumine on süvitsiotsing, siis on mäluvajadus fikseeritud b korral d suhtes lineaarne, s.t $O(d)$.

Hindame nüüd iteratiivse laskumise ajavajadust. Viimane, edukas iteratsioon (sügavusel $c=d$) peab kontrollima

$$T_1 = ((d+1+(b^{d+1}-1)/(b-1)) / 2) \text{ tippu (s.o süvitsiotsingu ajavajadus).}$$

Iga eelmine, j -s iteratsioon, mis ebaõnnestus (sügavustel $j=1, 2, \dots, d-1$), pidi kontrollima $1+b+\dots+b^j = (b^{j+1}-1)/(b-1)$ tippu. Ebaõnnestunud iteratsioonid kokku pidid kontrollima

$$T_2 = \sum_{j=1}^{d-1} (b^{j+1}-1)/(b-1) \text{ tippu.}$$

Seega on summaarne ajavajadus $T_1 + T_2$ eksponentsiaalne: $O(b^d)$.

Tabelis 15.4 on toodud iteratiivse laskumise ja süvitsiotsingu võrdlus: mida suurem on puu hargnemistegur, seda enam läheneb iteratiivne laskumine ajavajaduse mõttes tavalisele süvitsiotsingule.

Tabel 15.4. Iteratiivse laskumise ja süvitsiotsingu ajavajaduste suhe sõltuvalt puu hargnemistegurist

| Hargnemistegur b | Ajavajadus iter. laskumisel/ ajavajadus süvitsiotsingul |
|--------------------|--|
| 2 | 3 |
| 3 | 2 |
| 5 | 1,5 |
| 10 | 1,2 |
| 25 | 1,08 |
| 100 | 1,02 |

Iteratiivne laienemine kehtestab kunstliku limiidi laiusele ja suurendab seda 1 kaupa seni, kuni lõpuks leiab lahendi. Algul otsitakse laiusega 2 (süvitsi!), siis (algusest peale) laiusega 3 (süvitsi!) jne. Limiit c tähendab seda, et järjekordse tipu puhul kontrollitakse ülimalt c tema vahetut alluvat, enne kui see tipp tunnistatakse ebasobivaks ning jätkatakse järgmise tipuga.

1. Olgu $c=2$ (jooksev limiit laiusele).
2. Olgu L lähtetippude nimestik.
3. Olgu n esimene tipp nimestikus L . Kui L on tühi, siis $c:=c+1$, mine 2.
4. Kui n on lõpptipp, siis lahend leitud, lõpeta.
5. Vastupidisel juhul kustuta n nimestikust L . Lisa nimestiku L algusesse c esimest tipu n vahetut alluvat. Mine 3.

Iteratiivse laienemise mäluvajadus on süvitsiotsingu mäluvajadus (lineaarne).

Ajavajadus on eksponentsiaalne: sooritatakse ülimalt $b-1$ iteratiivset laienemist ($c=2, c=3, \dots, c=b$ korral) ning igal (süvitsi-)otsingul on ajavajadus $O(c^d)$. Kokku tuleb avada $2^d+3^d+\dots+b^d$ tippu. Suure b korral on see ligikaudu b^{d+1}/d (s.t b/d korda halvem kui tavalisel süvitsiotsingul) ja suure d korral ligikaudu b^d .

Iteratiivne laienemine sobib, kui otsingupuus leidub mitu lõpptippu ja tavalise süvitsiotsingu korral on võimalus otsingu varasel etapil teha saatuslik viga, suundudes puu valesse harru.

Harjutusülesanded

1. Otsing toimub lõplikul puul, mis ei sisalda ühtki lõppolekule vastavat tippu. Kuidas käitub iteratiivne laskumine sellisel juhul?
2. Otsing toimub lõplikul puul, kus on üksainus lõppolekule vastav tipp. Kuidas käitub iteratiivne laienemine sellisel juhul?

16. Pimeotsing graafil

Otsingu võib muuta keeruliseks asjaolu, et üht ja sedasama olekut tuleb läbida korduvalt. On probleeme, kus sellist komplikatsiooni ei esine, sest olekute ruum on puu ja igasse olekusse viibki üksainus tee. Sellise probleemi näiteks on 8 lipu paigutamine ükshaaval algselt tühja malelaua vasakpoolseimasse parajasti vabasse veergu nii, et ükski lipp ei tulistaks ühtegi teist lippu. Mõningate probleemide puhul on aga korduvad olekud vältimatud.

Süvitsiotsingu korral säilitatakse mälus need tipud, mis asuvad juurtipust algaval ja jooksvasse tippu suunduval teel. Võrreldes neid jooksva tipuga, saab algoritm avastada tsükleid. Puul otsingu algoritmi saab modifitseerida, võttes kasutusele veel ühe nimestiku, milles säilitatakse kõiki läbitud tippe (s.t tippe, mille vahetud alluvad on juba genereeritud). Nimetame seda nimestikuks SULETUD ning nimestikku, milles säilitatakse läbimiseks väljavalitud, kuid veel läbimata tippe, nimestikuks AVATUD (eespool vaadeldud algoritmides tähistasime seda L). Kui tipp, mis kuulub parajasti läbimisele, juba asub nimestikus SULETUD, siis jäetakse ta kõrvale. Nii saame graafil otsingu algoritmi GRAPHSEARCH (Nilsson 1982).

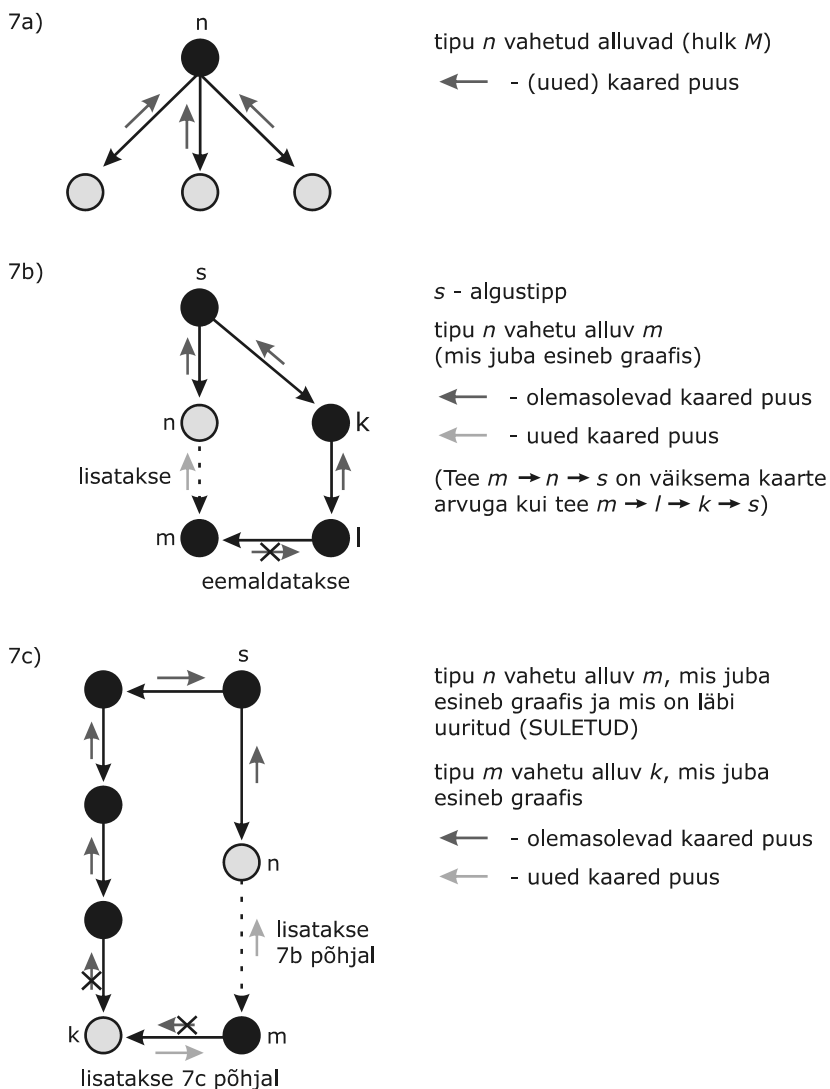
Algoritm GRAPHSEARCH on efektiivsem puul otsingu algoritmidest selliste probleemide puhul, kus esineb palju korduvaid olekuid.

GRAPHSEARCH

1. Konstrueeri otsimisgraaf G (tippudeks on seisundid), mis koosneb vaid algolekule vastavast tipust s . Paiguta s nimestikku AVATUD.
2. Loo nimestik SULETUD, mis on algul tühi.
3. Kui AVATUD on tühi, siis ebaõnn. Lõpp.
4. Eemalda esimene tipp nimestikust AVATUD ja paiguta ta nimestikku SULETUD. Tähistada teda n .
5. Kui n on lõppolekule vastav tipp, siis edu. Lõpp: lahenduse saamiseks liigu viitu mööda tipust n tipu s . (Viidad määratakse sammul 7.)
6. Läbi tipp n , genereerides kõik tema vahetud alluvad (tähistame seda hulka M). Paiguta graafisse G tipu n vahetute alluvatena need tipud hulgast M , mis seal varem ei esine.
7. a) Paiguta viit tipule n nendelt hulga M elementidelt, mis varem ei esinenud graafis G . Lisa need elemendid nimestikku AVATUD. b) Kui M element oli juba AVATUD või SULETUD, siis otsusta, kas tuleb ümber orienteerida viit n -le. c) Iga M elemendi jaoks, mis on juba SULETUD, otsusta iga tema vahetu alluva puhul graafis G , kas on vaja ümber orienteerida tema viit.
8. Järjesta ümber nimestik AVATUD.
9. Mine 3.

Kommentaariid

Sammul 7 toimub viitade konstrueerimine (joonis 16.1). Viitad orienteeritakse nii, et lõpuks saaks neid mööda liikuda tagasi lõppolekule vastavast tipust algustippu s.



Joonis 16.1. Viitade paigutamine graafis

7a) Konstrueeritakse viitad läbitava tipu n uutelt vahetatelt alluvatelt tipu n .

7b) Kui mõni genereeritud alluv juba esineb graafis, s.t kas nimestikus AVATUD või SULETUD (viimasesse saab ta sattuda kuidagi ainult läbi nimestiku AVATUD), siis võidakse mõni olemasolev viit ümber paigutada. Ümberpaigutamise mõte seisneb selles, et

võib-olla eksisteerib tänu uuele seosele lühem tee tipust n algtippu s . Seega püüab GRAPHSEARCH leida ühtlasi kõige otsema tee algtipust s lahendini.

7c) Kui mõni tipu n genereeritud vahetutest alluvatest (m) on juba olemas nimestikus SULETUD, siis see tähendab, et ta esineb graafis ja on ise juba ka läbitud (s.t tema vahetud alluvad on leitud ja viidad nendelt temale paigutatud). Nüüd kontrollitakse, kas m vahetult järglaselt (k) tipu n kaudu poleks algtippu s minna otsem kui siiani eksisteerinud viitu mööda. Kui jah, siis paigutatakse viit ümber nii, et ta suunaks tipule m , s.t otsingupuus saab tipu k vahetuks eellaseks hoopis m . (Varem ei olnud tee tipust k tipu m kaudu tipu s võimalik, sest puudus seos m ja n vahel.)

Sammul 8 võidakse järjestamine läbi viia erinevate kriteeriumide alusel, näiteks:

- kui tipud järjestatakse sügavuse kahanemise järjekorras, siis on see süvitsotsing.
- Kui tipud järjestatakse sügavuse kasvamise järjekorras, siis on see laiutiotsing.

17. Heuristiline otsing graafil

Otsingu tõhustamiseks võib kasutada ainevaldkonna kohta käivat, nn *heuristilist* infot. Enamikus praktilistes ülesannetes püütakse minimeerida eesmärgile viiva tee hinna ja selle tee leidmiseks vajaliku otsingu hinna mingit kombinatsiooni. Meid huvitavad sellised otsingumeetodid, mis minimeerivad selle kombinatsiooni keskmiselt kõigi oodatavate probleemide jaoks. Kui ühe otsingumeetodi puhul on keskmine hind väiksem kui teise puhul, siis öeldakse, et esimesel otsingumeetodil on suurem *heuristiline jõud* kui teisel.

Heuristilist infot saab kasutada algoritmis GRAPHSEARCH nimestikus AVATUD olevate tippude järjestamiseks 8. sammul, et otsing jätkuks perspektiivikamatest tippudest.

Kuidas aga leida perspektiivikamat tippu? Üks viis on kasutada selleks nn hinnangu-funktsiooni f , mis omistab igale tipule n reaalarvulise väärtuse $f(n)$.

Hinnangufunktsioon e hindefunktsioon (*evaluation function*) on funktsioon, mis määrab probleemiruumi vaheolekute väärtuse lahenduse otsingu ajal.³³

Tipu perspektiivikuse hindamiseks on erinevaid võimalusi:

- määrata, missuguse tõenäosusega asub tipp parimal teel,
- mõõta tipu oletatavat kaugust lõpptipust,
- omistada olekule teatav arv punkte (seda kasutatakse lauamängudes ja peamurdmis-ülesannetes).

Olgu $f(n)$ tee koguhind algtipust lõpptippu, eeldusel, et see tee läbib tipu n . Eeldame siin, et mida väiksem $f(n)$, seda suurem on tõenäosus, et n asub optimaalsel teel.

³³ IT terministandardi sõnastik <http://www.keeveeb.ee/> (18.02.2010)

Näide 17.1. 8-mäng.

Olgu hinnangufunktsioon

$$f(n) = d(n) + W(n),$$

kus $d(n)$ – tipu n sügavus, $W(n)$ – valem kohal asuvate numbrite arv.

Kui valida selline hinnangufunktsioon, mis alahindab tipu perspektiivikust, siis võidakse saada lahendav tee, mille hind ületab minimaalse (kaotus tee hinnas). Kui aga valida selline hinnangufunktsioon, mis ülehindab tipu perspektiivikust, siis võidakse avada liiga palju tippe (kaotus ajas).

Definierime hinnangufunktsiooni nii, et ta hindaks 1) minimaalse hinnaga (odavaima) tee hinda algtipust s tipuni n ja 2) odavaima tee hinda tipust n lõpptippu.

Olgu $k(n_i, n_j)$ odavaima tee tegelik hind tipust n_i tippu n_j ;

$$h^*(n) = \min_{\{t_i\}} k(n, t_i),$$

kus t_i on lõpptippud.

Teed tipust n lõpptippu, mille hind on $h^*(n)$, nimetatakse *optimaalseks teeks*.

Toome sisse uue funktsiooni $g^*(n) = k(s, n)$ kõigi selliste tippude n jaoks, mis on saavutatavad algtipust s .

Definierime funktsiooni f^* kui kahe liidetava summa:

$$f^*(n) = g^*(n) + h^*(n),$$

kus $g^*(n)$ on optimaalse tee hind algtipust s tippu n ja $h^*(n)$ – optimaalse tee hind tipust n lõpptippu.

Soovime, et hinnangufunktsioon f oleks f^* hinnang. Sellepärast kasutame hinnangufunktsiooni kujul $f(n) = g(n) + h(n)$, kus g on g^* hinnang ja h on h^* hinnang. Funktsiooni h nimetatakse *heuristiliseks* funktsiooniks. Eeldame, et mida väiksem $f(n)$, seda tõenäolisem on, et n asub optimaalsel teel algtipust lõpptippu.

Def. Algoritmi GRAPHSEARCH, mis kasutab tippude järjestamiseks sammul 8 hinnangufunktsiooni $f(n) = g(n) + h(n)$, nimetatakse algoritmiks A .

Laiutiotsingu algoritm on A , sest siin $h=0$ (heuristilist infot pole) ja $g=d$ (tipu sügavus).

Def. Algoritmi A , mille korral h on h^* alumine tõke, nimetatakse algoritmiks A^* (loe: A tärn).

Laiutiotsingu algoritm on A^* , sest $h=0$ on h^* alumine tõke.

Def. Ütleme, et otsingualgoritm on *lubatav* (ingl *k admissible*), kui ta lõpetab töö suvalise graafi korral, kus leidub tee algtipust lõpptippu, leides optimaalse tee.

Kui võtta hinnangufunktsioonis $h=0$, siis saadud algoritm on lubatav, kuid tekitab laiutiotsingu (mis tavaliselt pole efektiivne).

Meie eesmärk on nüüd näidata, et A^* on lubatav algoritm (vt ka Nilsson 1980).

Tulemus 1. Algoritm A^* lõpetab alati töö lõplike graafide korral.

Algoritm GRAPHSEARCH peatub (kui üldse) kas sammul 3 või 5. Igal tsükli kordamisel eemaldatakse üks tipp nimestikust AVATUD ja lisatakse sinna lõplik arv *uusi* (läbitava tipu) vahetuid alluvaid. Lõpliku graafi korral me ammandame varem või hiljem uute alluvate hulga ja kui algoritm ei peatu sammul 5, siis ta peatub sammul 3, olles ammandanud nimestiku AVATUD.

Tulemus 2. Suvalisel momendil enne A^* töö lõpetamist on nimestikus AVATUD selline tipp n' , mis asub optimaalsel teel algtipust s lõpptippu ja mille korral $f(n') \leq f^*(s)$.

Vaatame suvalist momenti, kus A^* pole veel jõudnud lõpptipu läbimiseni. Olgu nimestikus AVATUD 1. kohal tipp n' . Tema jaoks $f(n')=g(n')+h(n')$. (Võib eeldada, et A^* on avastanud odavaima tee $s \Rightarrow n'$, sest igal sammul valitakse avamiseks vähima f väärtusega tipp.) Seega $f(n')=g^*(n')+h(n') \leq g^*(n')+h^*(n')=f^*(n')$.

Optimaalse tee iga tipu n jaoks $f^*(n)=f^*(s)$, sest $f(n)=f^*(n)=g^*(n)+h^*(n)$. Kuna vaadeldav tipp n' asub optimaalsel teel ja $f(n') \leq f^*(n)$, siis järelikult $f(n') \leq f^*(s)$.

Tulemus 3. Kui tee algtipust s lõpptippu eksisteerib, siis A^* lõpetab töö (ka lõpmatute graafide korral).

Oletame vastuväiteliselt, et A^* ei peatu kunagi.

Töö ei lõpe, kuni lisatakse pidevalt uusi tippe nimestikku AVATUD. Näitame, et isegi vähim f väärtus nimestikus AVATUD paiknevate tippude jaoks kasvab kuitahes suureks.

Olgu $d^*(n)$ – lühima tee pikkus vaadeldavas (ilmutamata antud) graafis algtipust s tippu n , mis on genereeritud (s.t avatud) algoritmi A^* poolt. Olgu minimaalne kaare hind e (>0). Siis $g^*(n) \geq d^*(n)e$ ($g^*(n)$ on optimaalse tee hind $s \Rightarrow n$). Kuna $g(n) \geq g^*(n)$, siis $g(n) \geq d^*(n)e$. Eeldusel $h(n) \geq 0$ saame $f(n) \geq g(n) \geq d^*(n)e$. Ka iga tipu n jaoks nimestikust AVATUD on $f(n)$ seega vähemalt $d^*(n)e$. Kuigi A^* valib avamiseks sellise tipu n nimestikust AVATUD, mille korral $f^*(n)$ on vähim, on selle valitud tipu jaoks $d^*(n)$ lõpuks kuitahes suur (kuna tee on pikk).

Tulemuse 2 kohaselt saame: kui A^* pole veel lõpetanud, siis leidub nimestikus AVATUD 1. kohal tipp n' , mille puhul $f(n') \leq f^*(s)$. Seega hetkel läbimiseks (nimestikku SULETUD paigutamiseks) valitava tipu n' jaoks on $f(n')$ ülalt tõkestatud ega saa lõpmatult kasvada. Järelikult ei saa tippude sulgemise protsess kesta lõpmatuseni, vaid peab jõudma lõpptipu (t) sulgemiseni. Seega pole võimalik, et A^* ei lõpeta tööd, olgugi et otsingugraaf on lõpmatu.

Järeldus. Suvaline tipp n nimestikust AVATUD, mille jaoks $f(n) < f^*(s)$, valitakse algoritmi A^* poolt läbimiseks varem või hiljem.

Tulemus 4. Algoritm A^* on lubatav (s.t kui eksisteerib tee algtipust s lõpptippu, siis A^* lõpetab töö, olles leidnud optimaalse tee).

Algoritmi lubatavuse definitsiooni kohaselt on vaja näidata, et kui tee $s \Rightarrow t$ eksisteerib, siis A^* lõpetab töö, olles leidnud odavaima (optimaalse) tee $s \Rightarrow t$.

1. A^* võib peatuda kas sammul 3 (nimestik AVATUD tühi, ebaõnn) või 5 (leidis nimestiku AVATUD 1. kohal lõpptipu).

Peatus sammul 3 pole võimalik. Miks? Tulemuse 2 kohaselt: igal nimestikust AVATUD valimise momendil, mis pole lõppmoment, on 1. kohal tipp n' , mis asub optimaalsel teel. Järelikult avastab A^* optimaalse tee lähtetipust s kuni eelviimase tipuni. Eelviimase tipu vahetu alluv on lõpptipp. See genereeritakse parajasti siis, kui eelviimast tippu hakatakse sulgema.

2. Kas A^* leiab optimaalse tee $s \Rightarrow t$?

Oletame vastuväiteliselt, et A^* peatus, leidnud tee $s \Rightarrow t$, mis pole optimaalne, s.t $f(t) = g(t) > f^*(s)$. Kuid tulemuse 2 kohaselt saame, et ka eelviimasel sammul leidub nimestikus AVATUD selline tipp n' , mis a) asub optimaalsel teel ja b) $f(n') \leq f^*(s)$. Järelikult valib A^* viimasel sammul avamiseks selle ega vali mõnda teist.

Järeldus. Laiutiotsing leiab lühima tee.

Tulemus 5. Suvalise tipu n jaoks, mille algoritm A^* valib läbimiseks, kehtib $f(n) \leq f^*(s)$.

Kui n on lõpptipp ($n=t$), siis tulemuse 4 põhjal $f(t) = f^*(s)$.

Kui n ei ole lõpptipp, siis tulemuse 2 põhjal leidub nimestikus AVATUD tipp n' , mille korral $f(n') \leq f^*(s)$. Kui A^* valib avamiseks n' , siis on tõestus sellega lõppenud. Kui A^* valib avamiseks mingi muu tipu n nimestikust AVATUD, siis $f(n) < f(n')$. Järelikult tõepoolest $f(n) \leq f^*(s)$.

Algoritmi A^ variantide võrdlus*

Def. Vaatame algoritmi A^* kahte varianti A_1 ja A_2 , kus hinnangufunktsioonid on vastavalt $f_1(n) = g_1(n) + h_1(n)$ ja $f_2(n) = g_2(n) + h_2(n)$ ning h_1 ja h_2 olgu h^* alumised tõkked. Ütleme, et A_2 on *informeeritum* (ehk: rohkem informeeritud) kui A_1 , kui kõigi mittelõpptippude n korral $h_2(n) > h_1(n)$.

Intuitiivselt on arusaadav, et mida suurem on $h(n)$ (mida lähemal väärtusele $h^*(n)$), seda täpsem on heuristiline info.

Näide 17.3. 8-mäng.

Vaatame kahte algoritmi.

A_1 : $f(n) = d(n)$ (laiutiotsing),

A_2 : $f(n) = d(n) + W(n)$, kus $W(n)$ – vael kohal asuvate numbrite arv, mis on mängu lõpuni jäävate sammude arvu alumine tõke.

Algoritm A_2 on informeeritum kui A_1 .

Intuitiivselt on mõistetav, et informeeritum algoritm peab läbima vähem tippe. Samas võib informeeritum algoritm teha raskemaid arvutusi ja seetõttu võib-olla polegi efektiivsem.

Tõestame, et vähem informeeritud algoritm läbib **vähemalt needsamad tipud** (ilmutamata antud) otsingugraafis, mida läbib rohkem informeeritud algoritm.

Olgu A_1 ja A_2 algoritmid A^* .

Tulemus 6. Kui A_1 ja A_2 on A^* kaks varianti ja A_2 on informeeritum kui A_1 , siis otsingu-protsesside lõpuks mõlemas graafis, kus on leitud tee algtipust s lõpptippu, on iga algoritmi A_2 poolt läbitud tipp ühtlasi läbitud ka A_1 poolt (s.t A_1 läbib vähemalt needsamad tipud mis A_2).

Tõestame induktiivselt tipu sügavuse d järgi.

$d=0$.

Sügavusel 0 asub ainult alg Tipp. Selle läbivad mõlemad. (Kui see on lõpptipp, siis ta lihtsalt tõstetakse nimestikku SULETUD ja töö lõpeb. Kui see pole lõpptipp, siis genereeritakse tema vahetud alluvad.)

Oletame, et väide kehtib sügavuse $d=k$ korral ($k=1, 2, \dots$) ja tõestame, et kui sügavusel $k+1$ on tipp n , mille läbib A^*_2 , siis läbib selle tipu ka A^*_1 .

Oletame vastuväiteliselt, et A^*_1 ei läbi tipu n sügavusel $k+1$. Kuna induktiivse eelduse kohaselt A^*_1 (ja A^*_2) läbis tipu n vahetu ülemuse, siis peab algoritmi A^*_1 töö lõppedes n olema tema nimestikus AVATUD.

Kuna A^*_1 on lubatav, siis leidis ta optimaalse tee $s \Rightarrow t$. Sellel teel ei asu n . Järelikult pole õige (järelendus tulemusest 3), et $f_1(n) < f^*(s)$. Seega $f_1(n) \geq f^*(s)$ ehk $g_1(n) + h_1(n) \geq f^*(s)$ ehk $h_1(n) \geq f^*(s) - g_1(n)$. Aga $g_1(n) \leq g_2(n)$, järelikult $h_1(n) \geq f^*(s) - g_2(n)$.

Kuna A^*_2 avas tipu n , siis tulemuse 5 kohaselt $f_2(n) \leq f^*(s)$ ehk $g_2(n) + h_2(n) \leq f^*(s)$ ehk $h_2(n) \leq f^*(s) - g_2(n)$.

Seega saame: $h_1(n) \geq h_2(n) -$ vastuolu eeldusega, et A^*_2 on informeeritum kui A^*_1 .

Monotoonne piirang

Algoritm GRAPHSEARCH

- kontrollib, kas tippu pole juba varem genereeritud (sammud 6,7),
- korrigeerib otsingugraafi, et leida optimaalset teed (samm 7).

Meie järgmine eesmärk on piirata h nii, et A^* ei peaks otsingugraafi korrigeerima.

Def. Öeldakse, et heuristiline funktsioon h rahuldab **monotoonset piirangut**, kui iga tippude paari n_i, n_j korral, kus n_j on n_i vahetu alluv, kehtib $h(n_i) - h(n_j) \leq c(n_i, n_j)$, kusjuures $h(t) = 0$. (Siin $c(n_i, n_j)$ – kaare (n_i, n_j) hind, t – lõpptipp.)

Näide 17.4. 8-mäng.

Heuristiline funktsioon $h(n) = W(n)$ rahuldab monotoonset piirangut.

Tulemus 7. Kui kehtib monotoonne piirang, siis algoritm A^* avastab optimaalse tee suvalise tipuni n , mille ta on valinud läbimiseks, ja $g(n) = g^*(n)$.

Olgu n suvaline tipp, mille A^* valis avamiseks. Kui $n=s$ (alg Tipp), siis $s \Rightarrow s$ ongi optimaalne tee. Kui s valitakse alles esmakordseks läbimiseks, siis on tõestus sellega lõppenud. Kui s valiti korduvalt, siis järelikult paiknes ta juba varem nimestikus SULETUD.

Oletame, et n pole s . Olgu optimaalne tee alg tipust tippu n

$$s = n_0, n_1, \dots, n_m, n_{m+1}, \dots, n_k = n.$$

Olgu n_m viimane neist tippudest, mis asub nimestikus SULETUD sel ajal, kui A^* valib läbimiseks n . Tipp n ise ei saa olla SULETUD, sest ta valitakse parajasti läbimiseks (s.t asub nimestikus AVATUD esimesel kohal). Järelikult tipp n_{m+1} (kui n_m vahetu alluv) on sel hetkel samuti nimestikus AVATUD.

Monotoonse piirangu tõttu kehtib iga i ($i=0, 1, \dots, k-1$) korral sellel optimaalsel teel: $h(n_i) \leq h(n_{i+1}) + c(n_i, n_{i+1})$ ehk $g^*(n_i) + h(n_i) \leq g^*(n_i) + h(n_{i+1}) + c(n_i, n_{i+1})$. Kuna n_i ja n_{i+1} asuvad optimaalsel teel, siis $g^*(n_i) + h(n_i) \leq g^*(n_{i+1}) - c(n_i, n_{i+1}) + h(n_{i+1}) + c(n_i, n_{i+1})$ ehk $g^*(n_i) + h(n_i) \leq g^*(n_{i+1}) + h(n_{i+1})$.

Transitiivsuse tõttu $g^*(n_{m+1})+h(n_{m+1}) \leq g^*(n_k)+h(n_k)$ ehk $f(n_{m+1}) \leq g^*(n_k)+h(n_k)$.

Sel ajal, kui A^* valib läbimiseks tipu n , eelistades teda teistele tippudele nimestikus AVATUD, peab kehtima $g(n) \leq g^*(n)$ (sest muidu oleks $f(n) > f(n_{m+1})$). Kuna aga iga tipu p korral otsingupuus $g(p) \geq g^*(p)$, siis järelikult $g(n) = g^*(n)$.

Tulemus 8. Kui kehtib monotoonne piirang, siis funktsiooni f väärtused A^* poolt läbitud tippude järjendis ei kahane.

Vaatame tippe n_i ja n_{i+1} . Tipp n_{i+1} ei saa asuda n_i läbimise ajal nimestikus SULETUD, sest ta pole veel läbitud. Kui n_{i+1} asub tipu n_i läbimise ajal nimestikus AVATUD, siis peab kehtima $f(n_i) \leq f(n_{i+1})$, sest vastasel korral valiks A^* läbimiseks n_{i+1} .

Kui n_{i+1} ei asu nimestikus AVATUD, vaid alles genereeritakse, siis paigutatakse ta nimestikku AVATUD kui n_i vahetu järglane.

Sel momendil, kui n_{i+1} valitakse läbimiseks, kehtib tulemuse 7 põhjal $g(n_{i+1})=g^*(n_{i+1})$.

Seega $f(n_{i+1})=g(n_{i+1})+h(n_{i+1})= g^*(n_{i+1})+h(n_{i+1})=g^*(n_i)+c(n_i,n_{i+1})+ h(n_{i+1}) \geq g(n_i)+h(n_i)=f(n_i)$, seega $f(n_{i+1}) \geq f(n_i)$.

Kui monotoonne piirang ei kehti, siis on võimalik, et järjekordseks läbimiseks valitava tipu f väärtus on väiksem kui eelmistel. Seda asjaolu saab kasutada A^* efektiivsuse parandamiseks. Tulemus 5 ütleb, et iga tipu n jaoks, mida A^* valib läbimiseks, kehtib $f(n) \leq f^*(s)$. Võtame kasutusele globaalse muutuja F , mille väärtus igal hetkel olgu A^* poolt selle hetkeni läbitud tippude f väärtuste maksimum. Kehtib $F \leq f^*(s)$. Kui nimestikus AVATUD asub tipp n , mille korral $f(n) > F$, siis kunagi ta läbitakse. Selliseid tippe võib olla ka mitu. Nende hulgast võib teha valiku hoopis g väärtuse järgi (valida sellise, mille g on vähim). Eesmärk on suurendada tõenäosust, et kõige esimene avastatud tee mingi lõpptipuni ongi optimaalne (ja sellega vähendada viitade ümberorienteerimise vajadust).

Heuristiline jõud

Algoritmi heuristilist jõudu mõjutavad kolm faktorit:

- tee hind,
- tee leidmisel avatavate tippude arv,
- h arvutamise raskus.

Kuidas suurendada algoritmi heuristilist jõudu?

1) Kasutada mingit funktsiooni h , mis pole h^* alumine tõke (sellisel juhul muidugi pole algoritm lubatav).

Näide 17.4. 8-mäng.

Funktsioon $h(n)=W(n)$ on funktsiooni $h^*(n)$ alumine tõke, kuid ta ei hinda positsiooni eriti hästi (lõppolekusse viivate sammude arvu mõttes). Parema hinnangu annaks funktsioon $h(n)=P(n) -$ summa kõikide numbrite kaugustest lõppolekuni, ignoreerides segavaid numbreid. Veel parem hinnang on $h(n)=P(n)+3 \cdot S(n)$, kus $S(n)$ saadakse, kontrollides järjest mittetsentraalseid ruute ning iga numbrist eest, mille järel ei seisa õige, liidetakse 2, ja keskel asuva numbrist eest liidetakse 1. (See funktsioon ei ole h^* alumine tõke.)

2) Heuristilise funktsiooni arvutamise raskus.

Parim funktsioon oleks $h(n) = h^*(n)$, sest see tagaks avatavate tippude minimaalse arvu (aga seda võib olla raske arvutada).

Funktsioonide g ja h suhtelist osakaalu hinnangufunktsioonis võib varieerida, kasutades funktsiooni $f=g+w \cdot h$, kus $w > 0$:

- w väga suured väärtused omistavad suure tähtsuse heuristilisele komponendile,
- w väga väikesed väärtused annavad otsingule laiutiotsingu iseloomu.

Eksperimentaalselt on selgitatud, et otsingu efektiivsus kasvab, kui lubada w muutumist pöördvõrdeliselt tipu sügavusega otsingupuus:

- väikeste sügavuste korral suunab otsingut põhiliselt heuristiline komponent,
- sügavuse kasvades läheneb otsing laiutiotsingule, kindlustamaks, et mingi tee lõppolekuni leitakse.

Paljudes ülesannetes ongi vaja leida lihtsalt *mingi* tee lõppolekuni. Tagamaks, et tee leitakse, on vaja lülitada funktsiooni f koosseisu liidetav g (juhuks, kui h pole täpne hinnang).

Kui alati avatakse ainult minimaalse h väärtusega tippe, siis võiks otsinguprotsess kulgeda vales suunas. Funktsiooni g kaasamine lisab otsinguprotsessile laiutiotsingu komponendi ja järelikult tagab, et (ilmutamata antud) graafi mõni osa ei jää kogu aeg vaatluse alt välja.

18. Heuristilise otsingu algoritme

Algoritmi A^* rakendamisel tekivad raskused seoses sellega, et paljude probleemide puhul on tippude arv eksponentsiaalne lahendustee pikkuse suhtes. Võib näidata, et

$$|h(n) - h^*(n)| \leq O(\log h^*(n)),$$

kus $h^*(n)$ on tee tegelik hind tipust n lõpptippu. Peaaegu kõigi praktikas kasutatavate heuristikute korral on viga vähemalt võrdeline tee hinnaga ning selle tulemusel tekivad avatavate tippude arvu eksponentsiaalne kasv ületab arvutite võimalused. Seetõttu ei otsitagi alati optimaalset lahendit. Igal juhul võimaldab hea heuristiku kasutamine erakordset kokkuhoidu, võrreldes pimeotsinguga. Arvutusaeg pole siiski A^* põhiline puudus. Kuna ta säilitab mälu kõik genereeritud tipud (nagu seda teeb GRAPHSEARCH), ammendab A^* mälu enamasti ammu enne ajalimiidi ületamist. Seetõttu ei ole A^* suurte probleemide lahendamiseks praktikas eriti sobiv. Mõned teised algoritmid suudavad ületada mälu probleemi, ohverdades optimaalsuse või täielikkuse. Üks selline algoritm on IDA^* , mis kasutab iteratiivse laskumise ideed.

*IDA**: iteratiivne laskumine A^* (*iterative deepening A^** , algoritm IDA^* , R. Korf, 1985)

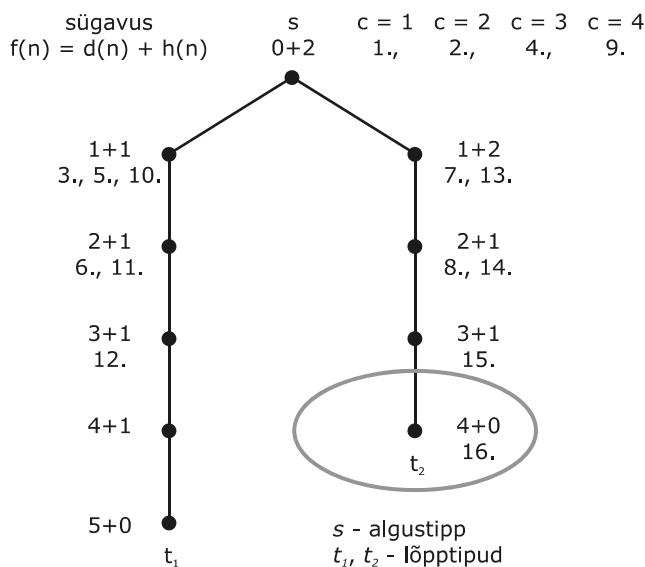
Algoritmi A^* mäluvajadus on eksponentsiaalne lõpptipu sügavuse suhtes (kuna $h=0$ korral taandub A^* laiutiotsingule). Mäluprobleemi võib lahendada iteratiivse laskumisega (nagu pimeotsingu korral). Nüüd aga ei lõigata ära selliseid tippe, mis asuvad allpool (järjest suurendatavat) sügavuse limiiti, vaid need tipud, millest *lähim lõpptipp* asub sügavamal kui sügavuse limiit. Nagu tavalisel iteratiivsel laskumisel, nii ka siin suurendatakse sügavuse limiiti järk-järgult seni, kuni lõpuks leitakse lahend.

Siin kasutatakse hinnangufunktsiooni $f(n)=g(n)+h(n)$, kus $g(n)$ – tippu n sügavus e tee pikkus algtipust tippu n ; $h(n)$ – hinnang tipust n lõpptipuni jääva tee pikkusele. See on tavaline iteratiivne laskumine (vt p.15), kus aga lisaks kasutatakse hinnangufunktsiooni, et määrata tippe, mis sügavuse limiidi tõttu ära lõigatakse. Üksikiteratsioonid kujutavad endast tavalist süvitsiotsingut (mitte A^*). Hinnangufunktsiooni kasutatakse tippude äralõikamiseks, mitte aga tippude läbimise järjekorra määramiseks nagu algoritmis A^* . Kui iga iteratsioon toimuks algoritmiga A^* , siis oleks mäluvajadus eksponentsiaalne.

1. Olgu $c=1$ jooksev sügavuse limiit.
2. Olgu L probleemi algolekute nimestik. Olgu $c'=\infty$ (sügavuse limiit järgmisel iteratsioonil).
3. Olgu n esimene tipp nimestikus L . Kui L on tühi ja $c'=\infty$, siis lahendit ei leidu, lõpeta. Kui L on tühi ja $c' \neq \infty$, siis võta $c=c'$ ja mine 2.
4. Kui n on lõppolek, siis lahend leitud, lõpeta.
5. Vastasel korral eemalda tipp n nimestikust L . Tippu n iga vahetu alluva n' jaoks, kus $f(n') \leq c$, lisa n' nimestiku L algusesse. Vastasel korral võta $c'=\min(c', f(n'))$. Mine 3.

Nagu algoritm A^* , nii leiab ka IDA* optimaalse lahendi, kui c' on lubatav heuristik (s.t $c' \leq h^*(n)$). Oma süvitsiotsingu tehnika tõttu on IDA* väga efektiivne mälu mõttes. IDA* oli esimene heuristilise otsingu algoritm, mis leidis mõistliku aja ja mäluga optimaalsed lahendid 15-mängu jaoks.

Joonisel 18.1 on toodud näide algoritmi IDA* rakendamise kohta.



Joonis 18.1. Algoritm IDA* leiab optimaalse lahendi

Vaatame siin veel nn selektiivgenereerimise algoritmi ja tema mõningaid modifikatsioone.

Selektiivgenereerimine e Geneereeri ja testi (generate and test)

1. Geneereeri võimalik lahend (algtipust lähtuv tee).

2. Testi: kas see on tee lõpptipuni.
3. Kui jah, siis lahend leitud, lõpeta. Kui ei, siis mine 1.

Süsteemaatilisel kujul on selektiivgenereerimine lihtsalt probleemiruumi täielik läbivaatus (süvitsotsingu iseloomuga).

Selektiivgenereerimine võib lahendite genereerimisel tegutseda ka juhuslikult, aga siis pole garanteeritud, et ta lahendi leiab. Sellisel kujul tuntakse teda *Briti muuseumi algoritmina*.³⁴

Nende äärmuste vahel on otsing süsteemaatiline, kuid mõningaid teid ei vaadelda, kuna need on perspektiivitud. Perspektiivikuse hindamine toimub hinnangufunktsiooni abil.

Näide 18.1. Antud on neli kuupi, mille iga tahk on mingit värvi (võimalikke värve on neli). Paigutada kuubid üksteise peale nii, et saadud torni igal neljal küljel oleksid esindatud kõik värvid.

Selle lihtsa ülesande lahendamiseks on Briti muuseumi algoritm täiesti arvestatav tehnika. Inimene lahendab selle ülesande tavaliselt niiviisi, et proovib süsteemaatilist läbi erinevaid võimalusi. Kui rakendada heuristikuid, saadakse lahend kiiremini. Näiteks kui tundub, et klotsidel on punaseid tahke kõige rohkem, siis mitme punase tahuga klotsi torni paigutades peaks püüdma asetada punase tahu kas alla või peale.

Mäkketõus (hill climbing) on selektiivgenereerimise „parandatud” variant, milles genereerija kasutab tagasisidet testijalt, valimaks otsingusuunda. Oleku hindamiseks kasutatakse hinnangufunktsiooni ning paremaks loetakse seda olekut, mille hinnang on suurem (piltlikult öeldes, asub „kõrgemal”, mäetipule lähemal).

1. Kontrolli algolekut. Kui see on ühtlasi lõppolek, siis on lahend leitud, lõpeta. Vastasel korral võta algolek jooksvaks olekuks ja jätkka.
2. Korda kuni hetkeni, kus lahend on leitud või ei leidu enam ühtki operaatorit, mida jooksvas olekus saaks rakendada:
 - (a) vali operaator, mida pole jooksvas olekus veel rakendatud ning rakenda seda, genereerimaks uut olekut.
 - (b) Testi uut olekut:
 - (i) kui see on lõppolek, siis lahend leitud, lõpeta.
 - (ii) Kui see pole lõppolek, aga on jooksvast olekust parem, siis võta ta jooksvaks olekuks.
 - (iii) Kui see olek pole jooksvast olekust parem, siis korda tsüklit.

Näide 18.2. Vaatame uuesti nelja kuubi ülesannet.

Olgu hinnangufunktsioon määratud kui erinevate värvide arvu summa kõigil neljal küljel. (Lõppolekus $4+4+4+4=16$.)

Lähteolekuks olgu kuupide suvaline paigutus tornis. Ainus operaator olgu järgmine: valida kuup ja pöörata seda 90^0 mingis suunas. Rakendame mäkketõusu algoritmi: kui uus olek on parem kui eelmine, siis võtame ta jooksvaks olekuks. Vastupidisel juhul aga pöördume tagasi eelmisse olekusse ja proovime teisiti.

³⁴ Üks selle nimetuse seletus on järgmine: kui panna kari ahve kirjutusmasinate taha, siis juhuslikult klahve vajutades võiksid nad varem või hiljem produtseerida kõik Briti muuseumis säilitatavad Shakespeare'i teosed.

Näide 18.3. Antud on seitsmest järjestikusest ruudust koosnev mängulaud ning kolm valget ja kolm musta nuppu, mis mängu alguses paiknevad juhuslikus järjekorras. Mängus saab teha järgmisi käike:

1) paigutada nupp kõrvalasuvasse tühja ruutu;

2) hüpata nupuga üle ühest või kahest naaberruudust, jõudmaks tühja ruutu.

Eesmärk on paigutada kõik valged nupud kõikidest mustadest nuppudest vasakule; tühja ruudu asukoht ei oma tähtsust.

Hinnangufunktsiooniks võib siin võtta mustadest vasakul asuvate valgete nuppude arvu.

Mäkketõusu algoritmi rakendamisel võivad aga tekkida raskused, mis on seotud lokaalsete maksimumidega:

- *mäejalam* (*foothill*) – olek, mis on parem igast oma naabrist, kuid ei ole parem mõnest muust (kaugemast) olekust.
- *plato* (*plateau*) – olek, mille kõik vahetud järglased on sama hinnaga.
- *mäeseljak*³⁵ (*ridge*) – olukord, kus soovitavas suunas pole võimalik liikuda.

Nende olukordade käsitlemiseks on mitmeid võtteid:

- tagasipöördumine mingisse varasemasse olekusse ja uue suuna valik (sobib mäejalami puhul),
- pikk hüpe mingis ühes suunas (sobib platoo puhul),
- mitme operaatori järjest rakendamine, enne kui kontrollida uut olekut (sobib mäeseljaku puhul).

Mäkketõusu algoritmi mitmed modifikatsioonid lubavad ka kõrvalekaldumisi parimalt suunalt, võimaldades paremini toime tulla lokaalsete maksimumidega. Vaatame siin 1) libalõõmutamist ja 2) geneetilist algoritmi.

Libalõõmutamine (simulated annealing), S. Kirkpatrick 1983.

Libalõõmutamise eesmärk on toime tulla lokaalsete maksimumidega, mis on mäkketõusu põhiraskus. Kasutatakse mäkketõusu tehnikat, kuid vahetevahel tehakse samm mitte selles suunas, kus hind on maksimaalne, vaid mingis teises suunas („allamäge“). Aja möödudes vähendatakse allamäge tehtava sammu tõenäosust teatava määra võrra.

Algoritmi nimetus tuleneb analoogiast metallivalu tehnikaga³⁶. Kui sula metall valatakse vormi, siis ta jahtub aeglaselt, kuni saavutab tahke oleku ja säilitab oma kuju. Et muuta saadud detaili võimalikult vastupidavaks, tuleb temperatuuri alandada vähehaaval. Kui protsess on lõppenud ja metall on tahkunud, siis öeldakse, et ta on lõõmutatud.

Füüsikalised substantsid liiguvad tavaliselt kõrgemalt energiatasemelt madalamale. Aga teatavatel juhtudel on võimalik ka vastupidine üleminek. Sellise vastupidise ülemineku tõenäosus on $p = e^{-\Delta E/(kT)}$, kus ΔE – energiataseme muutus, T – temperatuur, k – Boltzmanni konstant. Sellest valemist järeldub:

1. suure allamäge liikumise tõenäosus (ΔE suur) on väiksem kui väiksemal;
2. suurel T väärtusel on allamäge liikumise tõenäosus suurem kui väiksemal.

³⁵ Mäeseljak (geogr) – suhteliselt kitsas järsunõlvaline piklik kõrgendik.

³⁶ Lõõmutama – (metalle) kuumutades ja aeglaselt jahutades termiliselt töötlemata.

Allamäge liikumise tõenäosus alaneb sedamööda, kuidas temperatuur langeb. Järelikut tehakse selliseid käike rohkem protsessi alguses, kui temperatuur on kõrge. Allamägekäigud on lubatavad kogu aeg. Suured allamäge-liikumised võivad esineda protsessi alguses, edaspidi muutuvad nad väiksemaks, kuni lõpuks protsess tardub.

Väärtused, millel süsteemi jahutatakse, võetakse *lõõmutamisgraafikult* (*annealing schedule*). Füüsikaline lõõmutamisprotsess on lõõmutamisgraafiku suhtes väga tundlik. Kui jahtumine toimub liiga ruttu, siis kujunevad detailis stabiilsed alad, kus säilib kõrge energianivoo. Teisiti öeldes, saavutatakse lokaalne, mitte aga globaalne maksimum. Kui jahtumine toimub aeglasemalt, siis on globaalse maksimumi saavutamise tõenäosus suurem. Kui aga lõõmutamisgraafik on liiga aeglane, siis kulub ülearu palju aega.

Seda protsessi modelleeritaksegi libalõõmutamise algoritmis.

1. Kontrolli algolekut. Kui see on ühtlasi lõppolek, siis lahend leitud, lõpeta. Vastasel korral võta algsolek jooksvaks olekuks ja jätk.
2. Võta SENI-PARIM väärtuseks jooksev olek.
3. Võta T (temperatuuri) algväärtus lõõmutamisgraafikult.
4. Korda kuni hetkeni, kus lahend on leitud või enam pole reegleid, mida jooksvas olekus saaks rakendada.
 - (a) Vali reegel, mida pole jooksvas olekus veel rakendatud, ja rakenda seda, genereerimaks uut olekut.
 - (b) Kontrolli uut olekut. Arvuta $\Delta E = (\text{jooksva oleku väärtus}) - (\text{uue oleku väärtus})$.
 - Kui uus seisund on lõppolek, siis lahend leitud, lõpeta.
 - Kui uus seisund pole lõppolek, kuid on parem kui jooksev olek, siis võta ta jooksvaks olekukks. Võta SENI-PARIM väärtuseks jooksev olek.
 - Kui uus olek ei ole parem kui jooksev olek, siis muuda ta jooksvaks olekuks tõenäosusega $p' = e^{-\Delta E/T}$. Sellel sammul kasutatakse tavaliselt juhuslike arvude generaatorit, genereerimaks arvu $[0,1]$. Kui see arv on väiksem kui p' , siis tee see käik. Vastasel korral ära tee midagi.
 - (c) Muuda T vastavalt lõõmutamisgraafikule.
5. Tagasta SENI-PARIM kui vastus.

Libalõõmutamise erinevused mäkketõusust:

1. kasutatakse lõõmutamisgraafikut;
2. lubatakse käike halvematesse olekutesse;
3. lisaks jooksvale olekule võetakse kasutusele ka seni-parim. See on olek, millesse tagasi pöörduda, kui satutakse ummikusse.

Libalõõmutamise ja lõõmutamise võrdlus

- Algoritm ei näita ΔE energia, vaid hinnangufunktsiooni väärtuse muutumist.
- T ei ole temperatuur, vaid on kunstlik suurus.
- Tõenäosuse arvutamiseks kasutatakse muudetud valemit $p' = e^{-\Delta E/T}$.
- Lõõmutamisgraafik libalõõmutamisel sisaldab järgmisi suursi:
 - T algväärtus (kasutatakse sammul 3).
 - Kriteerium, mille alusel otsustatakse, kas T tuleb alandada (sammul 4c).
 - Suurus, mille võrra T alandatakse (sammul 4c).
 - Võib olla veel 4. komponent – millal lõpeta.

Libalõõmutamise algoritmi on palju kasutatud selliste ülesannete lahendamiseks, kus võimalike käikude arv mingist seisundist on väga suur. Kui T läheneb nullile, siis tõenäosus, et aktsepteeritakse käiku halvemasse seisundisse, läheneb samuti nullile ning libalõõmutamine läheneb tavalisele mäkketõusule.

Geneetiline algoritm (genetic algorithm), John Holland, 1960–1970.

Analoogiliselt libalõõmutamisega on siin tegu mäkketõusuga, kus on teatava tõenäosusega lubatud allamäge liikumised. Algul genereeritakse juhuslikult olekute hulk, nn indiviidide populatsioon. Iga järgmise oleku (lapse) genereerimisel arvestatakse *kahte* võimalikku eelnevat (lapsevanemad). Järgmise „põlvkonna” genereerimisel võetakse arvesse igale indiviidile omistatud *kõlblikkuse hinnangut (fitness)*; kõrgema hinnanguga indiviidid valitakse lapsevanemateks suurema tõenäosusega.

funktsioon GENEETILINE-ALGORITM (populatsioon, kõlblikkusfunktsioon) tagastab indiviidi

sisendid: populatsioon (indiviidide hulk)
 kõlblikkusfunktsioon (funktsioon, mis mõõdab indiviidi sobivust)

korda

uus_populatsioon := tühihulk

korda i = 1 **kuni** populatsiooni_suurus

x := JUHUSLIK-VALIK(populatsioon, kõlblikkusfunktsioon) /„ema”/

y := JUHUSLIK-VALIK(populatsioon, kõlblikkusfunktsioon) /„isa”/

laps := REPRODUTSEERI(x,y)

kui (väike juhuslik tõenäosus) siis laps := MUTEERI(laps)

lisa laps uude_populatsiooni

populatsioon := uus_populatsioon

kuni mingi indiviid on piisavalt sobiv või on kulutatud piisavalt aega tagasta parim indiviid populatsioonis, vastavalt kõlblikkusfunktsioonile

- *Reprodutseerimine*

funktsioon REPRODUTSEERI(x,y) tagastab indiviidi

sisendid: x,y (indiviidid-lapsevanemad)

n := pikkus(x)

c := juhuslik arv 1 kuni n

tagasta ÜHENDA(ALAMSÕNE(x,1,c), ALAMSÕNE(y,c+1,n))

- (Indiviidi) *muteerimine* seisneb selles, et indiviidile (s.o olekule) rakendatakse juhuslikult ühte võimalikest reeglitest.

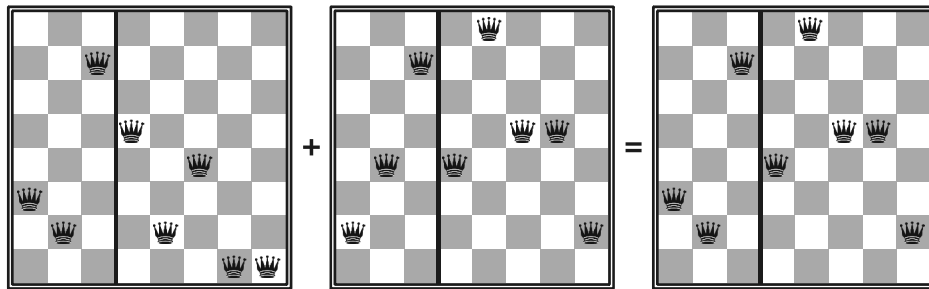
Näide 18.4. Paigutada malelauale 8 lippu nii, et ükski neist ei tulistaks ühtegi teist.

Siin on populatsiooniks juhuslikult genereeritud olekute (lauaseisude) hulk. Iga olekut saab esitada kui numbrijärjendit, kus i-s number näitab, mitmendas malelaua reas paikneb i-ndas veerus asuv lipp (i=1,...,8).

Oleku kõlblikkus on selliste lipupaaride arv, kus kumbki lipp ei tulista teist (minimaalselt 0, maksimaalselt $8 \cdot 7/2 = 28$), lõppolekus 28.

Muteerimine tähendab olekus juhuslikult ühe lipu valimist ja temaga juhuslikult ühe võimaliku (s.t tavalise lipu-)käigu tegemist.

Kokku leidub sellel ülesandel 92 lahendit. (Kui sümmeetriat mitte arvestada, siis on lahendite arv 12.) Reprodutseerimise näide on toodud joonisel 18.2: „isa” ja „ema” reprodutseerivad „lapse”.



Joonis 18.2. Reprodutseerimine kaheksa lipu ülesandes ($n=8, c=3$):
 $327|52411+247|48552=32748552$

Geneetilist algoritmi on rakendatud ka näiteks keeletehnoloogias. Stanfordi ülikoolis tehti 2003. a katse koostada automaatselt inglise keele formaalset grammatikat geneetilise algoritmi abil. Aluseks võeti lausete korpus, kus olid märgendatud sõnaliigid. Grammatika esitati kui lõplik mitteterminaalide ja eelterminaalide sõne (s.o kontekstivaba grammatika nn Chomsky normaalkujul). Nt sõne

SABABCBCDAE

vastab grammatikale (Chomsky normaalkujul):

$S \rightarrow A B$

$A \rightarrow B C$

$B \rightarrow C D$

$C \rightarrow A E$

Kõlblikkuse arvutamisel arvestatakse 1) grammatikaga korrektselt analüüsitud lausete suhtarvu, 2) grammatikat esitava sõne pikkust (s.t grammatikareeglite arvu).

Eksperimendid näitasid, et nii saadi grammatikad, millel on kõrge täpsus ja saak³⁷, kuigi nad erinevad inimeste koostatud grammatikatest.

Inimese koostatud grammatika sisaldas näiteks reeglid (esitatud nn Backus-Nauri valemitega):

$S \rightarrow NP VP \mid N VP \mid NP V \mid N V$ /* lause koosneb kas nimisõna fraasist ja verbifraasist või nimisõnast ja sellele järgnevast verbifraasist või nimisõna fraasist ja sellele järgnevast verbist või nimisõnast ja verbist */

³⁷ Täpsus (*precision*) – leitud õigete analüüside osakaal kõikide leitud analüüside hulgas; saak (*recall*) – leitud õigete analüüside osakaal kõikide õigete (inimese poolt tehtud) analüüside hulgas.

NP \rightarrow N N | T N /* nimisõnafraas koosneb kas kahest nimisõnast või artiklist ja nimisõnast */

VP \rightarrow V NP | V N /* verbifraas koosneb verbist ja sellele järgnevast nimisõnafraasist või verbist ja nimisõnast */

Geneetiline algoritm produtseeris aga hoopis erineva grammatika (siin 6 on uus mitte-termiaal):

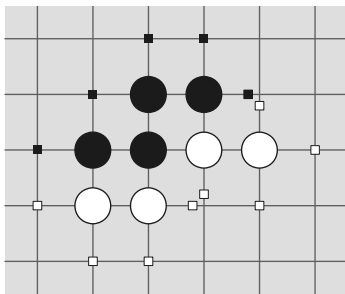
S \rightarrow 6 V | S 6 | N V | S N

6 \rightarrow T N | N N

19. Otsing mänguseisude puul

Võtame vaatluse alla kahe isiku mängud, milles vastased teevad kordamööda käike ja mõlemal on täpne informatsioon mängu kohta. Sellised mängud on näiteks trips-traps-trull (mida käsitlesime p. 4), kabe, male, aga ka näiteks go³⁸ ja Othello³⁹ (ehk Reversi).

Mäng go sai alguse Hiinas 4000 a tagasi. Sealt levis ta 8. sajandil Jaapanisse ja 19. sajandil Euroopasse. Seda mängu mängitakse 19x19 mängulaual. Ühel mängijal on mustad ja teisel valged nupud, kummalgi võib olla kuni 361 (=19x19) nuppu. Nuppe asetatakse mängulaua joonte ristumiskohtadesse, eesmärgiks on piirata oma nupukettidega (reas, veerus, diagonaalis) võimalikult palju joonte ristumiskohti, vastase nuppe ja nende kette (joonis 19.1). Võitja on see, kellel õnnestub piirata suurem arv vastase nuppe.



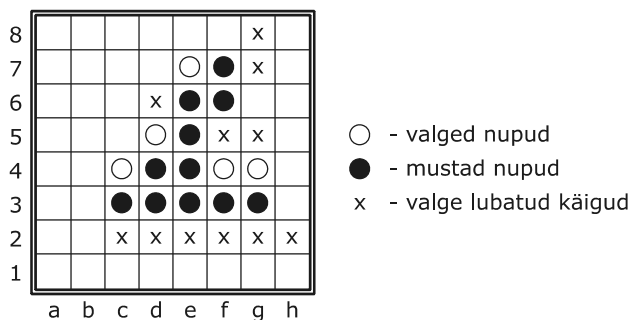
Joonis 19.1. Go mängulaud

Othellot mängitakse kabelaual. Kõik nupud on ühelt poolt valged ja teiselt poolt mustad. Algseisus asuvad kaks musta nuppu positsioonidel e4 ja d5 ning kaks valget nuppu positsioonidel d4 ja e5. Must alustab mängu. Oma nupu saab paigutada vahetult vastase nupu kõrvale esimesse vabasse positsiooni (joonis 19.2). Käigu tulemusel muutuvad vastavas reas, veerus või diagonaalis olevad vastase nupud „oma” nuppudeks, s.t nad pööratakse ümber (joonis 19.3). Võitja on see, kellel on mängu lõpuks rohkem nuppe.

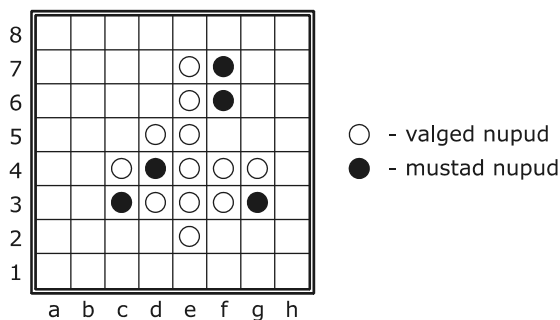
³⁸ <http://www.usgo.org/resources/whatisgo.asp> (18.02.2010)

³⁹ <http://home5.swipnet.se/~w-50714/othello/default.htm> (18.02.2010). Mängu nimi on võetud W. Shakespeare'i draamast ja viitab konfliktile kahe peategelase vahel.

Mängude mängimine on tehisintellekti jaoks hea valdkond, sest mängude näol on tegu struktureeritud probleemidega, kus on lihtne mõõta edu ja ebaedu. Juba 19. sajandil kavatses esimese programmjuhtimisega arvuti projekti autor Charles Babbage programmeerida oma nn analüütilise arvuti⁴⁰ malemängu ja trips-traps-trulli mängimiseks. 1912. a ilmunud artiklis esitas Ernst Zermelo minimax algoritmi. 1950. a avaldas Claude Shannon artikli malemänguprogrammis kasutatavate algoritmide kohta. Alan Turing koostas 1951. a malemänguprogrammi, mille arvutil katsetamiseni ta siiski ei jõudnud. Esimene märkimisväärne kabemängu mängimise programm õnnestus luua 1960. aastate algul Arthur Samuelil. 1970. a alustati Põhja-Ameerika maleprogrammide tšempionaate. 1974. a peeti esimene ülemaailmne arvutimale tšempionaat Stockholmis. 1997. a võitis maleprogramm Deep Blue tolleagese maailmameistri Garri Kasparovi.



Joonis 19.2. Othello mängulaud



Joonis 19.3. Othello: seis pärast valge käiku joonisel 19.2 kujutatud seisust ruutu e2

Võiks arvata, et programm saab mängu mängida, sooritades täieliku otsingu mänguseisude puul lähtesest kuni võiduni. Kahjuks pole see aga isegi päris lihtsate mängude puhul võimalik. Malemängus on keskmine hargnemistegur 35 ja keskmiselt teeb kumbki mängija 50 käiku. Seega sisaldab puu 35^{100} tippu. Üldiseks probleemilahendusmeetodiks võib olla genereerimine ja testimine. Üks äärmus on genereerida täielik lahend – tee lähtetipust kuni leheni – ja seejärel testida, kas jõuti võitvasse tippu. Teine äärmus on genereerida korraga ainult jooksva seisuga vahetud alluvad, mida kohe testida, et nende hulgast

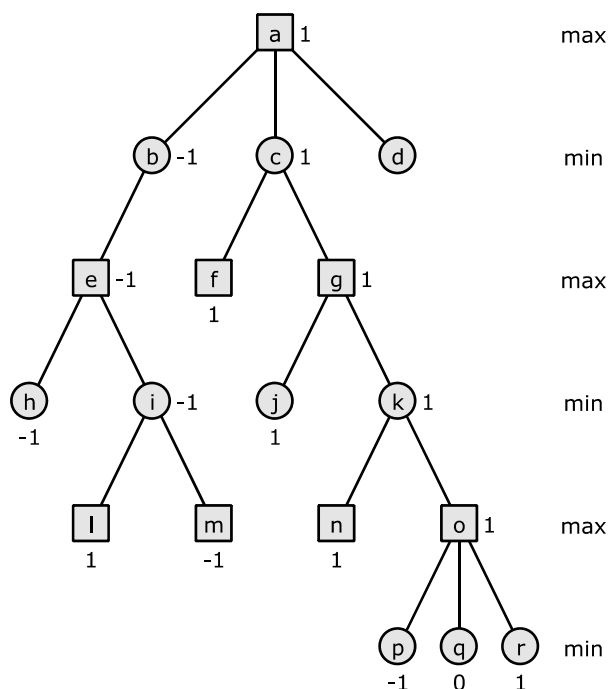
⁴⁰ Arvuti jäi valmis ehitamata.

valida parim. Otsingut saab parandada, lisades heuristilist infot nii genereerijasse kui ka testijasse.

Minimax-protseduurid (kasutatakse ka terminit *minmax*) püüavad minimeerida käigust tuleneda võivat maksimaalset kahju.

Ühte minimax-algoritmi vaatlesime põgusalt p. 4. Algoritmi idee on genereerida jooksvast positsioonist lähtudes võimalike järglaste hulk kuni puu lehtedeni (terminaalsete tippudeni), nendele positsioonidele rakendada staatilist hinnangufunktsiooni ja, liikudes puus tase-haaval kõrgemale tagasi lähte positsiooni, arvutada selle hinnang. Siin me eeldame, et staatiline hinnangufunktsioon omistab suured väärtused headele positsioonidele, nii et meie eesmärk on maksimeerida järgmise positsiooni hinnangut.

Ühe võimaliku mängu puu on toodud joonisel 19.4. Eeldame, et terminaalsetel tippudel on väärtused 1 (meie võit), -1 (meie kaotus) või 0 (viik). Igas tipus, kus meie oleme käigul, püüame teha parima käigu, s.t liikuda tippu, mille väärtus on võimalikult suur. Seevastu kui käigul on vastane, siis tema püüab valida sellise käigu, mis on parim tema jaoks, kuid halvim meie jaoks, s.t võimalikult väikese hinnangufunktsiooni väärtusega (joonis 19.4). Seetõttu nimetataksegi mängijaid minimeerijaks ja maksimeerijaks ning meetodit minimax-meetodiks.



Joonis 19.4. Mängupuu tippude hindamine

Minimax 1

Hindamaks tippu n mängupuul, tee järgmist.

1. Konstrueeri puu, alates tipust n kuni terminaalse tippudeni (lehtedeni).
2. Omista väärtused puu lehtedele (1 – maksimeerija võit, -1 – minimeerija võit).
3. Vali veel hindamata tipp, mille kõigil vahetutel järglastel on juba väärtused. Kui hindamata tippe ei ole, siis tagasta tipule n omistatud väärtus.
4. Kui valitud tipus on käigul minimeerija, siis võta tipu väärtuseks minimaalne tema vahetute järglaste väärtustest. Kui valitud tipus on käigul maksimeerija, siis võta tipu väärtuseks maksimaalne tema vahetute järglaste väärtustest. Mine 3.

Nii algoritmi mälu- kui ka ajavajadus on puu sügavuse suhtes eksponentsiaalne, sest kogu puu tuleb kuni lehtedeni valmis ehitada, enne kui saab hakata omistama väärtusi vahepealsetele tippudele.

Seda algoritmi saab parandada, rakendades süvitsiotsingut (Minimax 2). Algul ehitatakse valmis (süvitsi) üks puu haru kuni lehtedeni, omistades vahepealsetele tippudele väärtusi $-\infty$ või $+\infty$. Siis hakatakse lehtedest alates ülispoole tõusma ja vahepealsete tippude väärtusi muutma.

Minimax 2

Hindamaks tippu n mängupuul, tee järgmist.

1. Olgu $L = \{n\}$ - läbimata tippude nimestik.
2. Olgu x esimene tipp nimestikus L . Kui $x=n$ ja talle on omistatud väärtus, siis tagasta see väärtus.
3. Kui x väärtus on v_x , siis olgu p tipu x vahetu eellane ning v_p tema jooksev väärtus. Kui p on minimeerija tipp, siis võta $v_p = \min(v_p, v_x)$. Kui p on maksimeerija tipp, siis võta $v_p = \max(v_p, v_x)$. Eemalda x nimestikust L ja mine 2.
4. Kui tipule x pole omistatud väärtust ja ta on terminaalne tipp, siis omista talle väärtus 1 (kui see on maksimeerija võit), -1 (kui see on minimeerija võit) või 0 (kui see on viik). Jäta x nimestikku L (et käsitleda tema vahetut eellast) ja mine 2.
5. Kui tipule x pole omistatud väärtust ja ta pole lõpptipp, siis olgu⁴¹ $v_x = -\infty$ (kui x on maksimeerija tipp) või $v_x = +\infty$ (kui x on minimeerija tipp). Lisa x vahetud järglased nimestiku L algusesse ja mine 2.

Selle algoritmi mäluvajadus on lineaarne, ajavajadus aga eksponentsiaalne nagu eelmiselgi algoritmil. Näiteks malemängus ei tuleks selle algoritmi kasutamine kõne alla, sest mängupuu tippude arv on liiga suur. Kuidas aga mängib malet inimene? Ta ei analüüsi teed kuni mängu lõpuni, vaid ainult teatud seisuni, kus saab hinnata, kumb mängija tõenäoliselt võidab. Seega on meil vaja hinnangufunktsiooni $e(n)$, mis omistaks hinnangud mitteterminaalsetele tippudele. Olgu $-1 \leq e(n) \leq 1$, kusjuures $e(n) = 1$, kui tipus n kindlasti võidab maksimeerija, ja $e(n) = -1$, kui tipus n kindlasti võidab minimeerija. Malemängus võib sellise hinnangufunktsiooni defineerida igale malendile teatava arvu punktide omistamise teel (nt ettur – 1, ratsu – 3, vanker – 5 jne) ja võtta $e(n) = (v-m)/(v+m)$, kus v on valgete ja m mustade punktisumma.

⁴¹ Selline väärtuste omistamine on vajalik, et esimene minimeerimine või maksimeerimine algoritmi 3. sammul omaks mõtet.

Algoritm Minimax 3 kasutab hinnangufunktsiooni $e(x)$, kus $-1 \leq e(x) \leq 1$ ning 1 tähistab maksimeerija võitu ja -1 – minimeerija võitu.

Minimax 3

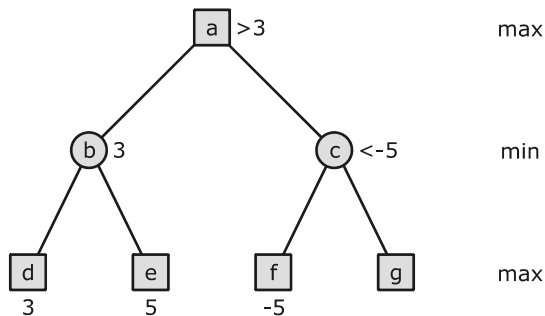
Hindamaks tippu n mängupuul, tee järgmist.

1. Olgu $L = \{n\}$.
2. Olgu x esimene tipp nimestikus L . Kui $x=n$ ja talle on omistatud väärtus, siis tagasta see väärtus.
3. Kui x väärtus on v_x , siis olgu p tipu x vahetu eellane ning v_p tema jooksev väärtus. Kui p on minimeerija tipp, siis võta $v_p = \min(v_p, v_x)$. Kui p on maksimeerija tipp, siis võta $v_p = \max(v_p, v_x)$. Eemalda x nimestikust L ja mine 2.
4. Kui tipule x pole omistatud väärtust ja ta kas on terminaalne tipp või *oled otsustanud puud temast allapoole mitte läbida*, siis arvuta tema väärtus, kasutades hinnangufunktsiooni, ja mine 2.
5. Vastasel korral võta $v_x = -\infty$ (kui x on maksimeerija tipp) või $v_x = \infty$ (kui x on minimeerija tipp). Lisa x vahetud järglased nimestiku L algusesse ja mine 2.

Sammul 4 tekib küsimus: kuidas otsustada, millist puu haru ei läbita? Lihtsaim viis oleks määrata teatud sügavus, millest sügavamale otsing ei suundu. On aga võimalik, et mõni haru on perspektiivikam, mida näitab asjaolu, et hinnangufunktsiooni $e(n)$ väärtused kasvavad seal järsult. Kui näiteks malemängus oleme otsustanud otsida kuni sügavuseni 7, aga just 7. sammul ilmneb, et vastane ähvardab meid, siis me peaksime vaatama sügavamale kui 7. Vastupidi, kui hinnangufunktsiooni väärtus üleminekul järgmisel sügavusel asuvasse tippu peaaegu ei muutu, siis see piirkond on stabiilsem ja seda poleks tarvis sügavamale uurida ega selleks asjatult energiat kulutada.

Süvitsiotsingu protseduuride (nagu Minimax 2 ja Minimax 3) hea omadus on see, et nende efektiivsust saab parandada harude ja tõkete tehnikate rakendamise teel. Seda tehnikat kirjeldasime rändkaupmehe probleemi puhul (p. 5): vaja on ainult meeles pidada seni leitud lühima tee pikkus. Niipea kui leitakse uus, lühem tee, võetakse see uueks lühimaks. Sedasama ideed saab rakendada ka kahe isiku mängudele. Selleks on vaja modifitseerida otsinguprotseduuri, et käsitleda nii maksimeerijat kui ka minimeerijat, ning muuta ka harude ja tõkete strateegiat nii, et ta käsitleks kahte tõket – ühte kummagi mängija jaoks. Moditseeritud strateegiat nimetatakse *alfa-beeta-kärpimiseks* (*alpha-beta pruning*). See nõuab kahe läviväärtuse käsitlemist, millest üks esitab alumist tõket väärtusele, mida saab omistada maksimeerivale tipule (α), ja teine esitab ülemist tõket väärtusele, mida saab omistada minimeerivale tipule (β).

Näide joonisel 19.5 illustreerib läve α käsitlemist. Pärast tipu f läbiuurimist me teame, et vastane saavutab tipus c kas väärtuse -5 või sellest väiksema väärtuse (kuna vastane on minimeerija). Samuti teame, et meie (maksimeerija) saavutame väärtuse 3 või sellest suurema, kui liigume tippu b . Iga muu käik tekitaks väärtuse, mis on väiksem kui 3 , ja seetõttu võime selliseid käike ignoreerida. Kontrollinud ainult tippu f , oleme kindlad, et liikumine tippu c on halvem (väiksem või võrdne -5), olenemata tipu g hinnangust.



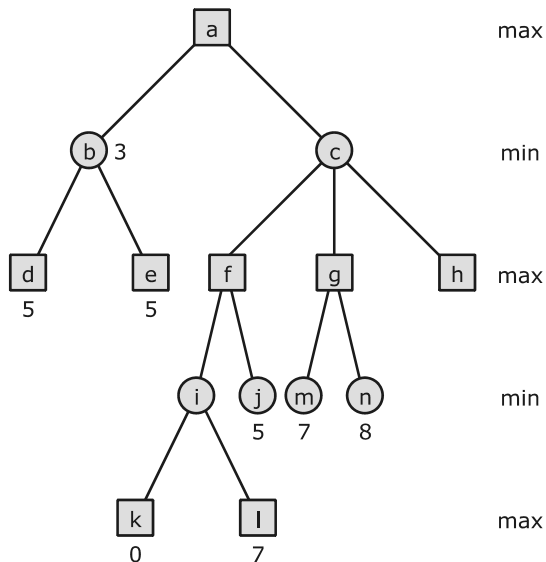
Joonis 19.5. Alfa-kärbe

Lävede α ja β koos kasutamist selgitab näide joonisel 19.6.

Läbides alampuud, mille juur on tipus b, avastame, et tipu a väärtus on vähemalt 3. Kui α väärtus antakse alla kuni tipuni f, siis see võimaldab meil vältida tipu l uurimist, sest pärast seda, kui k on kontrollitud, näeme, et i maksimaalne garanteeritud skoor on 0, mis tähendab, et f minimaalne garanteeritud skoor on 0. See aga on väiksem kui α väärtus 3, nii et teisi tipust i lähtuvaid harusid ei ole vaja vaadata. Maksimeerija juba teab, et tuleb vältida käiku tippu c ja siis tippu i, sest pärast seda käiku ei ole tulemusskoor parem kui 0, aga liikudes hoopis tippu b, saab saavutada skoori 3.

Nüüd vaatame, kuidas kasutada β väärtust. Pärast seda, kui tippude i ja j läbimine on ära lõigatud, on tipul f väärtus 5 ($= \max(0, 5)$). See väärtus saab β väärtuseks tipus c. Ta näitab, et c garanteeritud väärtus on ülimalt 5. Nüüd peame läbima g. Esmalt kontrollime tippu m – tema väärtus on 7 ja see antakse tagasi tippu g kui selle tipu ajutine väärtus. Nüüd võrreldakse 7 ja β ($=5$). 7 on suurem kui β ja tipus c on käigul minimeerija. Seega ei vali ta tippu g, mis tekitaks skoori vähemalt 7, kuna on olemas alternatiivne käik tippu f, mis viib skoorini 5. Järelikult ei ole teisi tipust g lähtuvaid harusid vaja läbi uurida.

Sellest näitest näeme, et maksimeerivatel tasemetel võime käigu varakult elimineerida, kui selgub, et tema väärtus on *väiksem* kui jooksev lävi, kuna aga minimeerivatel tasemetel võime katkestada otsingu niipea, kui avastame väärtuse, mis on *suurem* kui jooksev lävi. Kui maksimeerija elimineerib võimaliku käigu, siis see tähendab tegelikult otsingu äralõikamist minimeerival tasemel. Vaatame veel kord näidet joonisel 19.4, Niipea kui me tegime kindlaks, et c on halb käik tipust a, ei saa me läbida tippu g ega teisi harusid minimeerival tasemel allpool tippu c. Seega kasutatakse lävesid α ja β niiviisi, et otsing minimeerival tasemel peatatakse, kui leitakse väärtus, mis on väiksem kui α , ning otsing maksimeerival tasemel peatatakse, kui leitakse väärtus, mis on suurem kui β . Otsingu äralõikamine maksimeerival tasemel, kui leitakse suur väärtus, võib tunduda ebaloomulikuna, aga kui meenutada, et maksimeerival tasemel läheme me konkreetsesse tippu ainult sel juhul, kui minimeerija valib ühe taseme võrra ülalpool selle tipu, siis on arusaadav, et sellisel käitumisel on siiski mõtet.



Joonis 19.6. Alfa-kärbe ja beeta-kärbe

Olles illustreerinud alfa-beeta-kärpimist näidetega, võime nüüd uurida, kuidas tuleks modifitseerida minimax-protseduuri, et seda tehnikat esitada. Märgime, et maksimeerivatel tasemetel kasutatakse ainult läve β , et määrata, kas otsingupuud saab kärpida, ja minimeerivatel tasemetel kasutatakse selleks ainult läve α . Aga ka maksimeerivatel tasemetel peab α olema teada, sest kui tehakse minimax-protseduuri rekursiivne väljakutse, siis luuakse minimeeriv tase, mis vajab juurdepääsu lävele α . Seega peab maksimeerivatel tasemetel olema teada ka α , et teda saaks anda edasi puu alumistele tasemetele. Sama kehtib minimeerivatel tasemetel β kohta. Iga tase peab saama mõlemad väärtused – ühe kasutamiseks samal tasemel ja teise edasiandmiseks allapoole, et seda kasutada järgmisel, alumisel tasemel.

Alfa-beeta-kärpimine (α - β -otsing, harvendamine, ingl k α - β pruning, α - β search)

Hindamaks tippu n mängupuul, tee järgmist.

1. Olgu $L = \{n\}$.
2. Olgu x esimene tipp nimestikus L . Kui $x = n$ ja talle on omistatud väärtus, siis tagasta see väärtus.
3. Kui tipule x pole omistatud väärtust, siis mine 5. Kui tipule x on omistatud väärtus v_x , siis olgu p tipu x vahetu eellane. Kõigepealt tuleb määrata, kas tipu p ja tema järglased saab puust kärpida (*prune*): kui p on minimeerija tipp, siis olgu α maksimaalne p kõigi vahetute naabrite (*sibling*) ja p eellasteks olevate minimeerija tippude kõigi vahetute naabrite jooksvatest väärtustest. (Kui selliseid väärtusi ei ole, siis võta $\alpha = -\infty$.) Kui $v_x \leq \alpha$, siis eemalda tipp p ja kõik tema järglased (*descendant*) nimestikust L (**α -kärbe**).
Analoogiliselt toimi juhul, kui p on maksimeerija tipp: olgu β minimaalne p kõigi vahetute naabrite ja p eellasteks olevate maksimeerija tippude kõigi vahetute naabrite jooksvatest väärtustest. (Kui selliseid väärtusi ei ole, siis võta $\beta = +\infty$.) Kui $v_x \geq \beta$, siis eemalda tipp p ja kõik tema järglased nimestikust L (**β -kärbe**).

4. Kui tippu p koos tema järglastega ei saa kärpida, siis olgu v_p tipu p jooksev väärtus. Kui p on minimeerija tipp, siis võta $v_p = \min(v_p, v_x)$. Kui p on maksimeerija tipp, siis võta $v_p = \max(v_p, v_x)$. Eemalda x nimestikust L ja mine 2.
5. Kui tipule x pole omistatud väärtust ja ta on kas terminaalne tipp või sa oled otsustanud puud tipust x allapoole mitte läbida, siis arvuta tipu x väärtus, kasutades hinnangu-funktsiooni, ja mine 2.
6. Vastasel korral võta $v_x = -\infty$ (kui x on maksimeerija tipp) või $x = +\infty$ (kui x on minimeerija tipp). Lisa x kõik vahetud järglased nimestiku L algusesse ja mine 2.

Alfa-beeta-kärpimise ajavajadus

Harvendamine võib taandada otsinguruumi suurus. *Halvimal* juhul on võimalik, et kärpimine ebaõnnestub – kui järjestame iga tipu vahetud järglased nii, et esimesena kontrollitakse halvimat. *Parimal* juhul kontrollitakse parimaid käike esimesena. Vaatame minimeerija vastust, mis pole temale parim. Et seda saaks kärpida, tuleb meil kontrollida nii palju otsinguruumist, et saaksime väita, et minimeerija jaoks on see käik viga – seega kontrollime selle käigu „eitust”, mis näitab, kuidas maksimeerija saab seda ära kasutada. Et nii toimida, kontrollime ühtainsat maksimeerija vastust – parimat. Siis kontrollime minimeerija kõiki valikuid, kuid ainult maksimeerija parimat vastust igähele neist, jne. Hargnemistegur minimeerija jaoks on b (puu eeldatav hargnemistegur), maksimeerija jaoks aga 1. Kui vaadata protsessi minimeerija seisukohast, siis on analüüs analoogiline, üksnes mängijate rollid on vahetatud. Tippude koguarv, mida tuleb kontrollida sügavusel d , on ligikaudu $b^{d/2} + b^{d/2} = 2b^{d/2}$ (mitte aga b^d). Seega on harvendamise puhul oluline, kuidas tipu vahetuid alluvaid järjestada. Üks võimalus on kasutada selleks hinnangufunktsiooni.

Maleprogrammides võimaldab harvendamine toime tulla 1,5-kordse teoreetiliselt minimaalse arvu tippude uurimisega.

NegaScout⁴² on harvendamise modifikatsioon, kus ei kontrollitagi tippe, mida harvendamine võib kärpida. Eelduseks on siin tippude hea järjestus.

III. B. Osandamine

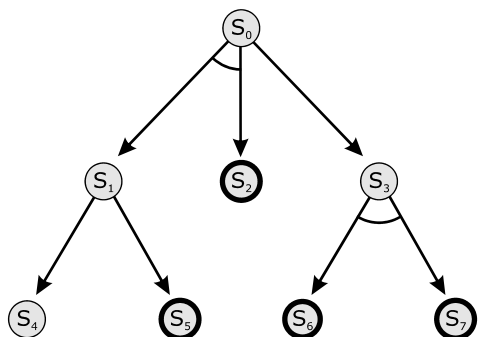
20. Probleemilahendus osandamise teel

Probleem on lahendatav osaprobleemideks redutseerimise ehk osandamise teel, kui on antud:

- 1) lähteprobleemi kirjeldus,
- 2) reeglite hulk, mis redutseerivad probleeme osaprobleemideks,
- 3) elementaarsete probleemide kirjelduste hulk.

⁴² <http://www.zib.de/reinefeld/Research/nsc.html> (18.02.2010)

Joonisel 20.1 on toodud näide probleemi osandamisest: lähteprobleemi S_0 saab redutseerida osaprobleemideks S_1 , S_2 ja S_3 . Edasi on kaks võimalust: kas lahendada osaprobleemid S_1 ja S_2 (väljuvad kaared on omavahel ühendatud) või lahendada osaprobleem S_3 . Osaprobleemi S_3 lahendamiseks tuleb omakorda lahendada mõlemad tema osaprobleemid S_6 ja S_7 . Elementaarsed probleemid on joonisel raamitud (S_2 , S_5 , S_6 ja S_7).



Joonis 20.1. Probleemi osandamine

Näide 20.1. Integraali arvutamine

Lähteprobleem: arvutada integraal $\int \frac{x^2 - 2x + 1}{x - 1} dx$.

Reeglid:

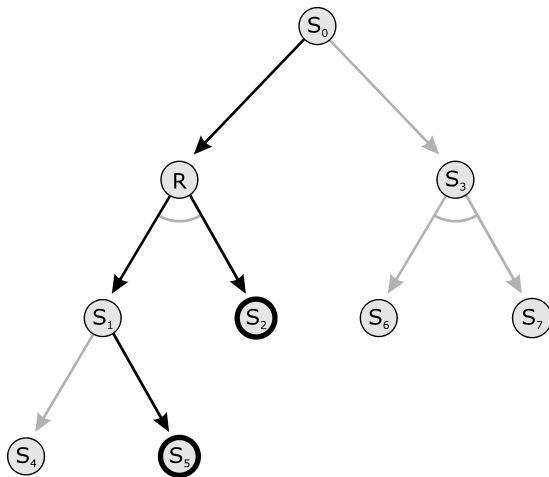
- a. ositi integreerimise reeglid,
- b. summaks lahutamise reeglid,
- c. konstandi ettetoomise reegel,
- d. algebraliste asenduste reeglid

jne.

Elementaarseteks probleemideks on tabeliintegraalid.

Probleemilahenduse võib esitada JA-VÕI graafi (*and/or graph*) või JA-VÕI puu (*and/or tree*) kujul, mille tippudele vastavad probleemid ja osaprobleemid. Graafis on kaht liiki tippe: JA-tipud, millele vastava probleemi lahendamiseks tuleb lahendada *kõik* vahetud osaprobleemid, milleks see probleem on taandatud (graafis on sellisest tipust väljuvad kaared ühendatud joonekesega), ning VÕI-tipud, millele vastava probleemi lahendamiseks tuleb lahendada *vähemalt üks* selle probleemi vahetu osaprobleem. Joonisel 20.2 on toodud JA-VÕI puu näide.

Arvestades seda, et JA/VÕI graafis peab iga tipp olema ainult kas JA- või VÕI-tüüpi, pidime joonisel 20.1 esitatud graafi ümberjoonistamiseks sisse tooma uue tipu R (joonis 20.2).



Joonis 20.2. JA-VÕI graaf. Jämedate joontega on esile toodud üks lahendus

Et leida lähteprobleemi lahendit, on vaja konstrueerida (ilmutamata antud) JA/VÕI-graafi ilmutatud osagraaf, mis esitaks lähteprobleemi redutseerimist kuni elementaarsete osaprobleemideni vähemalt ühel viisil – s.o nn lahendusgraaf (selline osagraaf, mis koosneb ainult nn lahenduvatest tippudest).

Graafi tipp on *lahenduv*, kui (tabel 20.1)

- 1) ta on lõpptipp (vastab elementaarsele osaprobleemile) või
- 2) ta on mittelõpptipp, kuid JA-tipp, mille kõik vahetud alluvad on lahenduvad, või
- 3) ta on mittelõpptipp, kuid VÕI-tipp, mille vähemalt üks vahetu alluv on lahenduv.

Graafi tipp on *mittelahenduv*, kui (tabel 20.1)

- 1) ta on mittelõpptipp, millel pole järglasi (s.o. mitteelementaarne probleem, millele ei saa rakendada ühtki reeglit), või
- 2) ta on mittelõpptipp, mis on selline JA-tipp, mille vähemalt üks vahetu alluv on mitte- lahenduv, või
- 3) ta on mittelõpptipp, mis on selline VÕI-tipp, mille kõik vahetud alluvad on mitte- lahenduvad.

Tabel 20.1. Lahenduvad ja mittelahenduvad tipud

| <i>Lahenduvad tipud</i> | <i>Mittelahenduvad tipud</i> |
|---|---|
| lõpptipp (vastab elementaarsele probleemile) | mittelõpptipp, millel pole alluvaid (mitteelementaarne probleem, millele ei saa rakendada ühtegi reeglit) |
| mittelõpptipp, mis on JA-tipp, mille kõik vahetud alluvad on lahenduvad | mittelõpptipp, mis on JA-tipp, mille vähemalt üks vahetu alluv on mittelahenduv |
| mittelõpptipp, mis on VÕI-tipp, mille vähemalt üks vahetu alluv on lahenduv | mittelõpptipp, mis on VÕI-tipp, mille kõik vahetud alluvad on mittelahenduvad |

JA/VÕI graafil on rakendatavad nii laiuti- kui ka süvitsiotsing.

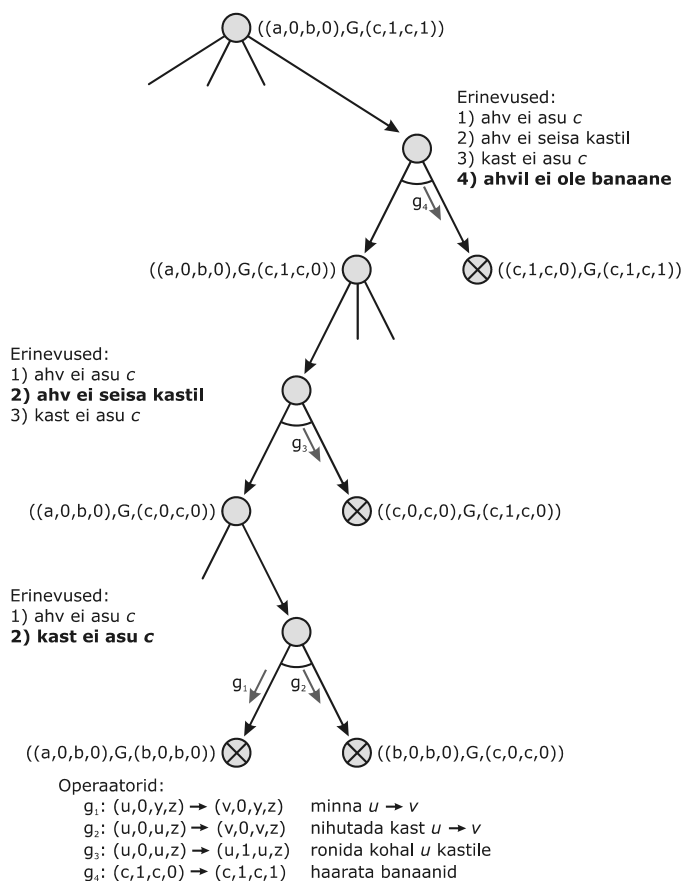
Üks meetod probleemi lahendamiseks osandamise teel on nn eesmärgi-abinõu analüüs e *vahendianalüüs (means-ends analysis)*. Meetod eeldab, et probleem on esitatud kui otsinguprobleem olekute ruumis. Leitakse erinevused jooksva oleku s_i ja lõppoleku s_f vahel. Valitakse neist välja üks erinevus, mida saab likvideerida ühe operatori rakendamise teel, s.t probleem „minna jooksvast olekust s_i lõppolekusse s_f ” redutseeritakse kaheks osaprobleemiks, millest (vähemalt) üks on lahendatav üheainsa reegli rakendamise tulemusel:

- a) „minna olekust s_i olekusse s_j ”,
- b) „minna olekust s_j olekusse s_i ”.

Teiste, mitteelementaarsete osaprobleemidega toimitakse analoogiliselt, kuni lõpuks viimasel sammul jääb järele kaks elementaarset osaprobleemi. Sellega ongi algne probleem lahendatud.

Joonisel 20.3 on ülesanne ahvist ja banaanidest lahendatud vahendianalüüsi meetodil.

Muidugi võib osutada oluliseks erinevuste käsitlemise järjekord. Tähtsaid erinevusi tuleks käsitleda varem kui vähemtähtsaid. Keeruliste probleemide lahendamiseks ei pruugi see lihtne meetod sobida, sest erinevuste permutatsioonide arv võib olla liiga suur.



Joonis 20.3 Ahv ja banaanid: probleemi osandamine

III.C. Teoreemide automaatne tõestamine

Teoreemi tõestamise probleem⁴³ püstitatakse järgmisel viisil: tõestada, et valemite hulgast F_1, \dots, F_n järeldeb loogiliselt valem G . Meenutame kahte varem tõestatud tulemust (p. 10).

Tulemus 10.1. Valem G järeldeb loogiliselt valemite hulgast $\{F_1, \dots, F_n\}$ parajasti siis, kui valem $F_1 \& \dots \& F_n \rightarrow G$ on samaselt tõene.

Tulemus 10.2. Valem G järeldeb loogiliselt valemite hulgast $\{F_1, \dots, F_n\}$ parajasti siis, kui valem $F_1 \& \dots \& F_n \& \neg G$ on samaselt väär.

Teoreemi tõestamist võib seega vaadelda kui teatava valemi $F_1 \& \dots \& F_n \& \neg G$ kummutamist (ingl k *refutation*).

21. Herbrandi meetod

21.1. Valemi teisendamine disjunktide hulgaks

Tõestusprotseduuri lihtsustamiseks teisendatakse kummutatav valem kõigepealt nn disjunktide hulgaks.

Disjunktiks (ingl k *disjunct, clause*) nimetatakse literaalide (elementaarvalemite ja/või nende eituste) disjunktsiooni.

Erilises rollis on tühidisjunkt – disjunkt, mis ei sisalda ühtegi literaali (s.o samaselt väär valem).

Teisendatav valem esitatakse kõigepealt nn prefikskujul

$$(Q_1x_1)(Q_2x_2)\dots(Q_nx_n)M, \quad (21.1)$$

kus iga Q_i ($i=1,2,\dots,n$) on kas olemasolu- või üldisuskvantor ning M on valem, mis ei sisalda kvantoreid. Seejuures $(Q_1x_1)(Q_2x_2)\dots(Q_nx_n)$ nimetatakse prefiksiks ja M maatriksiks.

Valemi teisendamine disjunktide hulgaks võib toimuda järgmiste tegevuste järjest rakendamise teel:

- 1) implikatsioonimärgid elimineerida,
- 2) eitusemärgid viia elementaarvalemiteni,
- 3) muutujad ümber nimetada, nii et iga kvantoriga oleks seotud erinimeline muutuja,
- 4) kvantorid tõsta valemi algusesse (saadakse valemi prefikskuju, *prenex normal form*, mis koosneb prefiksist ja maatriksist),
- 5) teisendada standardkujule (maatriks konjunktiivsele normaalkujule, olemasolukvantorid elimineerida, kasutades Skolemi funktsioone),
- 6) teisendada disjunktide hulgaks (prefiks ja konjunktsioonimärgid ära jätta).

Disjunktide hulka loeme kõigi temasse kuuluvate disjunktide konjunktsiooniks ja kõiki disjunktes esinevaid muutujaid loeme üldisuskvantoriga seotuks. Selle kokkuleppe

⁴³ Vt ka (Chang, Lee 1973), (Koit 1989).

kohaselt võibki (ainult üldisuskvantoreid sisaldava) prefiksi ja konjunktsiooni märgid ära jätta.

Näide 21.1. Teisendada disjunktide hulgaaks valem

$$\forall x \forall y \exists z (P(x,z) \& P(y,z) \rightarrow \exists u Q(x,y,u)).$$

Elimineerime implikatsioonimärgi (1):

$$\forall x \forall y \exists z (P(x,z) \& P(y,z) \rightarrow \exists u Q(x,y,u)) = \forall x \forall y \exists z (\neg P(x,z) \vee \neg P(y,z) \vee \exists u Q(x,y,u)).$$

Tegevused 2 ja 3 toimivad tühjalt. Tegevuse 4 tulemusel saame valem

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

Tegevus 5: elimineerime olemasolukvantorid ja asendame nendega seotud muutujad z ja u vastavate (kahekohaliste) Skolemi funktsioonidega. Saame valem standardkuju:

$$\forall x \forall y \exists z (\neg P(x,z) \vee \neg P(y,z) \vee \exists u Q(x,y,u)) = \forall x \forall y (\neg P(x,f(x,y)) \vee \neg P(y,f(x,y)) \vee Q(x,y,g(x,y))).$$

Tegevuse 6 tulemusel saame algsele valemile vastava disjunktide (üheelemendilise) hulga $\{\neg P(x,f(x,y)) \vee \neg P(y,f(x,y)) \vee Q(x,y,g(x,y))\}$.

Harjutusülesanne. Teisendada disjunktide hulkadeks järgmised valemid.

- $\neg(\forall x P(x) \rightarrow \exists y \forall z Q(y,z))$
- $\forall x \exists y \exists z ((\neg P(x,y) \& Q(x,z)) \vee R(x,y,z))$
- $\forall x (P(x) \rightarrow \exists y Q(x,y))$
- $\exists x (\forall y ((P(x) \rightarrow Q(x,y)) \rightarrow R(y)))$

Tulemus 21.1. Olgu S disjunktide hulk, mis esitab valem F standardkuju. Valem F on samaselt väär parajasti siis, kui hulk S on vasturääkiv.

Tõestus. Üldsust kitsendamata võib eeldada, et

$$F = Q_1 x_1 Q_2 x_2 \dots Q_n x_n M[x_1, x_2, \dots, x_n],$$

kus Q_i on üldsus- või olemasolukvantorid ja maatriksi M on esitatud konjunktiivsel normaalkujul.

Induktsioon \exists järgi. (Eeldame, et valem F prefiksi sisaldab m olemasolukvantorit.)

Induktsiooni baas. Olgu Q_r vasakpoolseim olemasolukvantor. Tähistame

$$F_1 = \forall x_1 \forall x_2 \dots \forall x_{r-1} Q_{r+1} x_{r+1} \dots Q_n x_n M[x_1, \dots, x_{r-1}, f(x_1, \dots, x_{r-1}), x_{r+1}, \dots, x_n].$$

Tõestame, et F on samaselt väär parajasti siis, kui F_1 on samaselt väär.

Tarvilikkus. Olgu valem F samaselt väär. Oletame vastuväiteliselt, et F_1 ei ole samaselt väär. Järelikult leidub interpretatsioon I, milles F_1 on tõene, s.t iga x_1, \dots, x_{r-1} korral leidub interpretatsioonis I selline element $f(x_1, \dots, x_{r-1})$, et valem $Q_{r+1} x_{r+1} \dots Q_n x_n M[x_1, \dots, x_{r-1}, f(x_1, \dots, x_{r-1}), x_{r+1}, \dots, x_n]$ on tõene. Siis peab aga ka valem F olema selles interpretatsioonis tõene \Rightarrow vastuolu.

Piisavus. Olgu valem F_1 samaselt väär. Oletame vastuväiteliselt, et F ei ole samaselt väär. Siis leidub interpretatsioon I (kandjaga D), milles F on tõene, s.t iga x_1, \dots, x_{r-1} korral leidub element x_r nii, et valem $Q_{r+1} x_{r+1} \dots Q_n x_n M[x_1, \dots, x_{r-1}, x_r, x_{r+1}, \dots, x_n]$ on tõene. Laiendame interpretatsiooni I: lisame (r-1)-kohalise funktsiooni f, mis igale argumentide väärtuste komplektile (x_1, \dots, x_{r-1}) kandjast D seab vastavusse väärtuse x_r . Saadud uut interpretatsiooni tähistame I'. Interpretatsioonis I' on iga x_1, \dots, x_{r-1} korral valem $Q_{r+1} x_{r+1} \dots Q_n x_n M[x_1, \dots, x_{r-1}, f(x_1, \dots, x_{r-1}), x_{r+1}, \dots, x_n]$ tõene. Järelikult on interpretatsioonis I' ka valem $F_1 = \forall x_1 \forall x_2 \dots \forall x_{r-1} Q_{r+1} x_{r+1} \dots Q_n x_n M[x_1, \dots, x_{r-1}, f(x_1, \dots, x_{r-1}), x_{r+1}, \dots, x_n]$ tõene \Rightarrow vastuolu.

Induktsiooni hüpotees. Tähistame F_k valemite, mis on saadud valemist F k vasakpoolseima olemasolukvantori elimineerimisel. Oletame, et valem F on samaselt väär parajasti siis, kui F_{k-1} on samaselt väär. Järelikult on valem F samaselt väär parajasti siis, kui F_k on samaselt väär. Järelikult on valem F samaselt väär parajasti siis, kui valem F_m , mis on saadud kõigi olemasolukvantorite elimineerimisel valemist F , on samaselt väär. Aga F_m on samaselt väär parajasti siis, kui disjunktide hulk S on vasturääkiv. See tõestabki teoreemi.

21.2. Disjunktide hulga H-interpretatsioon

Tõestamiseks, et valem G järeldub loogiliselt valemite hulgast $\{F_1, \dots, F_n\}$, tuleb tulemuse 10.2 kohaselt näidata, et $F_1 \& \dots \& F_n \& \neg G$ on samaselt väär, ja selleks omakorda tulemuse 21.1 kohaselt tõestada, et vastav disjunktide hulk on vasturääkiv. Valemite hulk on vasturääkiv, kui sellesse hulka kuuluvate valemite konjunktsioon on väär kõikvõimalikes interpretatsioonides, kõikvõimalike kandjatega. Kuna kõikvõimalikke interpretatsioone kõikvõimalike kandjatega on võimatu vaadelda, siis on meie järgmiseks eesmärgiks leida üks „hea” kandja, mis esindaks kõikvõimalikke ja mida saaks hõlpsasti konstrueerida kummutatava valemi põhjal. Selline kandja leidub tõepoolest ja seda nimetatakse disjunktide hulga Herbrandi universumiks.⁴⁴

Disjunktide hulga Herbrandi universum konstrueeritakse tase-haaval.

Def. Olgu H_0 – hulga S disjunktides esinevate konstantide hulk. Kui S ei sisalda konstante, siis paigutatakse hulka H_0 suvaline konstant, nt. a .

Seejärel olgu iga $i=0, 1, \dots$ korral

$H_{i+1} = H_i \cup \{f^n(h_1, \dots, h_n) \mid f^n - \text{hulgas } S \text{ esinev } n\text{-kohaline funktsioon; } n=1, 2, \dots; h_j \in H_i, j=1, \dots, n\}$.

Hulka $H = \bigcup_{i=0}^{\infty} H_i$ nimetatakse disjunktide hulga Herbrandi universumiks ning iga hulka H_i ($i=0, 1, \dots$) Herbrandi universumi i -ndaks tasemeks.

Näide 21.2. Olgu disjunktide hulk $S = \{P(x) \vee Q(x), \neg P(a), \neg Q(b)\}$. Selle disjunktide hulga Herbrandi universum on $H = \{a, b\}$.

Näide 21.3. Olgu disjunktide hulk $S = \{R(x), P(g(x)) \vee Q(x)\}$. Selle disjunktide hulga Herbrandi universum on lõpmatu:

$H_0 = \{a\}$,

$H_1 = \{a, g(a)\}$,

$H_2 = \{a, g(a), g(g(a))\}$,

...

$H = \{a, g(a), g(g(a)), g(g(g(a))), \dots\}$.

Def. Disjunktide hulga S Herbrandi baasiks H_B nimetatakse kõikide elementaarvalemite hulka kujul $p^n(h_1, \dots, h_n)$, kus p^n sisaldub hulga S disjunktides ($n=1, 2, \dots$) ja $h_1, \dots, h_n \in H$.

Näide 21.4. Näites 21.2 toodud disjunktide hulga Herbrandi baas on

$H_B = \{P(a), P(b), Q(a), Q(b)\}$.

⁴⁴ Prantsuse matemaatiku Jacques Herbrandi (1908-1931) auks, kes töötas välja esimese teoreemide automaatse tõestamise meetodi.

Näide 21.5. Näites 21.3 toodud disjunktide hulga Herbrandi baas on lõpmatu:

$$H_B = \{P(a), Q(a), R(a), P(g(a)), Q(g(a)), R(g(a)), \dots\}.$$

Vaatleme nüüd disjunktide hulga selliseid interpretatsioone, mille kandjaks on Herbrandi universum, nn H-interpretatsioone.

Def. Disjunktide hulga H-interpretatsiooniks nimetatakse interpretatsioon I, kandjaga H, mille korral kehtivad järgmised tingimused:

1. I kujutab hulga S iga konstandi iseendaks.
2. Olgu f suvaline n-kohalise funktsiooni sümbol, mis sisaldub hulga S disjunktidest ($n=1,2,\dots$). Interpretatsioon I seab funktsiooni sümbolile f vastavusse funktsiooni $H^n \rightarrow H$, mis kujutab hulga H iga elementide kogumi (h_1, \dots, h_n) hulga H elemendiks $f(h_1, \dots, h_n)$.
3. Hulgas S esinevaid predikaatsümboleid võib väärtustada suvaliselt.

Seega leidub ühel ja samal disjunktide hulgal erinevaid H-interpretatsioone. Kõik neid võib vaadelda kui disjunktide hulga Herbrandi baasi elementide (elementaarvalemite) väärtustusi (tõeväärtustega tõene või väär). Olgu $H_B = \{A_1, \dots, A_n, \dots\}$. Suvalise H-interpretatsiooni võib siis esitada kujul $\{m_1, \dots, m_n, \dots\}$, kus m_j on kas A_j või $\neg A_j$ ($j=1, 2, \dots$). Kui $m_j = A_j$, siis on elementaarvalemile A_j omistatud tõeväärtus tõene, ning kui $m_j = \neg A_j$, siis tõeväärtus väär.

Näide 21.6. Näites 21.2 vaadeldud disjunktide hulga Herbrandi baas on $H_B = \{P(a), P(b), Q(a), Q(b)\}$ (näide 21.4). H-interpretatsioonid:

$$I_1 = \{P(a), P(b), Q(a), Q(b)\}, I_2 = \{\neg P(a), P(b), Q(a), Q(b)\} \text{ jne (kokku on neid 16).}$$

Näide 21.7. Vaatleme näites 21.3 toodud disjunktide hulka. H-interpretatsiooni kandjaks on hulk $H = \{a, g(a), g(g(a)), g(g(g(a))), \dots\}$. Kujutus (definiitsiooni kohaselt):

$$a \rightarrow a,$$

$g \rightarrow g$ nii, et $g(h) \rightarrow g(h)$ iga $h \in H$ korral. Disjunktide hulga H-interpretatsioone on lõpmatu hulk:

$$I_1 = \{P(a), Q(a), R(a), P(g(a)), Q(g(a)), R(g(a)), \dots\},$$

$$I_2 = \{\neg P(a), Q(a), R(a), P(g(a)), Q(g(a)), R(g(a)), \dots\}$$

jne.

Disjunktide hulga igale (suvalisele) interpretatsioonile saab seada vastavusse teatava H-interpretatsiooni.

Def. Olgu I interpretatsioon, mille kandja on D. Interpretatsioonile I vastavaks H-interpretatsiooniks I^* nimetatakse H-interpretatsiooni, mis konstrueeritakse järgmiselt:

olgu $h_1, \dots, h_n \in H$. Vastaku termile h_i interpretatsioonis I element $d_i \in D$ ($i=1, \dots, n$; D on interpretatsiooni I kandja). Iga predikaatsümbole p^n ($n=1, 2, \dots$) korral: kui $p^n(d_1, \dots, d_n)$ omandab interpretatsioonis I väärtuse tõene (vastavalt väär), siis ka $p^n(h_1, \dots, h_n)$ omandab interpretatsioonis I^ väärtuse tõene (vastavalt väär).*

Näide 21.8. Antud on disjunktide hulk $S = \{P(x), Q(y, f(y, a))\}$. Antud on hulga S interpretatsioon I:

$D = \{1, 2\}$;

a f(1,1) f(1,2) f(2,1) f(2,2) P(1) P(2) Q(1,1) Q(1,2) Q(2,1) Q(2,2)
2 1 2 2 1 tõene väär väär tõene väär tõene

Leida vastav H-interpretatsioon I^* .

Interpretatsiooni I^* kandjaks on hulga S Herbrandi universum $H = \{a, f(a,a), f(f(a,a),a), \dots\}$. Selle disjunktide hulga Herbrandi baas:

$H_B = \{P(a), P(f(a,a), P(f(f(a,a),a))), \dots, Q(a,a), Q(a, f(a,a)), Q(f(a,a),a), \dots\}$.

Otsitava interpretatsiooni saame esitada kui hulga S Herbrandi baasi elementide väärtustuse, jälgides, et kõik elementaarvalemid omandaksid täpselt needsamad tõeväärtused mis interpretatsioonis I. Seega

$I^* = \{\neg P(a), P(f(a,a)), \neg P(f(f(a,a),a)), \dots, Q(a,a), \neg Q(a, f(a,a)), Q(f(a,a),a), \dots\}$.

Tulemus 21.2. Disjunktide hulk S on vasturääkiv parajasti siis, kui ta on väär kõigis oma H-interpretatsioonides.

Tõestus. Tarvilikkus. Olgu S vasturääkiv. Siis on temas sisalduvate disjunktide konjunktsioon väär kõikvõimalikes interpretatsioonides, sh H-interpretatsioonides.

Piisavus. Olgu hulgas S sisalduvate disjunktide konjunktsioon väär kõikides hulga S H-interpretatsioonides. Oletame vastuväiteliselt, et leidub interpretatsioon I, milles see konjunktsioon on tõene. Konstrueerime interpretatsioonile I vastava H-interpretatsiooni I^* . Elementaarvalemite tõeväärtused on interpretatsioonides I ja I^* ühesugused. Järelikult osutub, et leidub disjunktide hulga S muutujate selline väärtustus Herbrandi universumi elementidega, mille korral S kõik disjunktid on korraga tõesed. Järelikult selles H-interpretatsioonis on S disjunktide konjunktsioon tõene \Rightarrow vastuolu.

Seega piisab disjunktide hulga vasturääkivuse väljaselgitamiseks tema H-interpretatsioonide läbivaatamisest. Kui neid on lõplik hulk (nagu näites 21.6), siis on selle teoreemi alusel võimalik tõestada disjunktide hulga vasturääkivust.

21.3. Semantilised puud

Disjunktide hulga kõik H-interpretatsioonid saab esitada graafina, nn *semantilise puuna*. Disjunktide hulga vasturääkivuse tõestamist saab siis vaadelda kui tema semantilise puu konstrueerimist.

Def. Olgu S disjunktide hulk ja olgu H_B tema Herbrandi baas. Hulga S semantiliseks puuks nimetatakse puud, mille igale kaarele vastab hulga H_B elementide või nende eituste hulk nii, et:

1. suvalisest tipust N väljuvate kaarte märgendite disjunkttsioon on samaselt tõene valem;
2. tee juurtipust suvalisse tippu N ei sisalda kontraarseid paare (s.t korraga elementaarvalemite ja selle eitust).

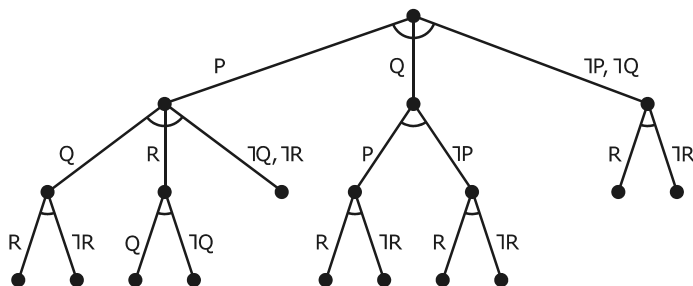
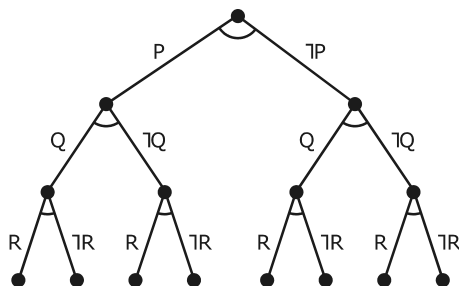
Näide 21.9. Olgu $H_B = \{P, Q, R\}$ hulga S Herbrandi baas. Joonisel 21.1 on kujutatud hulga S kaks semantilist puud.

Def. Olgu $H_B = \{A_1, \dots, A_n, \dots\}$ disjunktide hulga S semantiline puu. Semantiline puu on täielik parajasti siis, kui iga tee juurtipust suvalisse rippuvasse tippu (lehte) sisaldab H_B iga elemendi A_i ($i=1, \dots, n, \dots$) korral kas selle elemendi enda või tema eituse.

Joonisel 21.1 esitatud semantilised puud on täielikud.

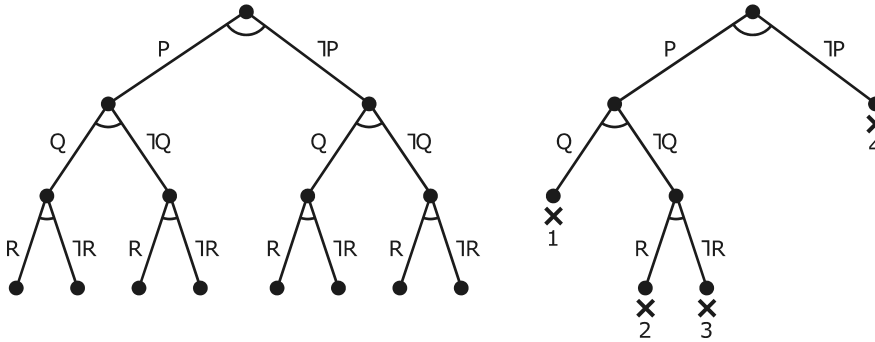
Kui hulga S Herbrandi baas on lõpmatu, siis on ka iga tema täielik semantiline puu lõpmatu. Täielik semantiline puu esitab disjunktide hulga kõikvõimalikud H -interpretatsioonid.

Disjunktide hulga S vasturääkivust saab näidata tema täieliku semantilise puu konstrueerimise kaudu: peatada puu konstrueerimine igas sellises tipus, kus osutub, et S mingi disjunkt on selles (selle hetkeni konstrueeritud osalises) H -interpretatsioonis väär (selliseid tippe nimetatakse kummutavateks tippudeks).



Joonis 21.1. Semantilised puud

Näide 21.10. Olgu $S = \{P, Q \vee R, \neg P \vee \neg Q, \neg P \vee \neg R\}$. Hulga S Herbrandi baas on $H_B = \{P, Q, R\}$. Joonisel 21.2 on vasakul näidatud hulga S täielik semantiline puu ja paremal sama puu, mille konstrueerimine on aga peatatud igas niisuguses tipus N , kus selgub, et osaline interpretatsioon ei rahulda hulga S mingit disjunkt. Tipus 1 selgub nimelt, et tõeväärtuse väär saab disjunkt $\neg P \vee \neg Q$, tipus 2 – disjunkt $\neg P \vee \neg R$, tipus 3 – disjunkt $Q \vee R$, tipus 4 – disjunkt P .



Joonis 21.2. Täielik semantiline puu

21.4. Herbrandi meetod

Teoreemide automaatse tõestamise Herbrandi meetod põhineb Herbrandi teoreemil (järgnev tulemus 21.3), mille aluseks on omakorda eespool tõestatud tulemus 21.2: disjunktide hulga vasturääkivuse tõestamiseks on tarvilik ja piisav kontrollida üksnes tema H-interpretatsioone. Kuna selliseid interpretatsioone võib aga olla lõpmatu hulk, siis tuleb neid teataval viisil süstematiseerida. Selleks saab kasutada semantilist puud.

Def. Disjunkt C fundamentaalnäiteks (*ground instance*) nimetatakse disjunkt, mis saadakse, asendades disjunktis C kõik muutujad Herbrandi universumi elementidega (samad muutujad muidugi samade elementidega).

Näide 21.11. Vaatleme näites 21.2 toodud disjunktide hulka $S = \{P(x) \vee Q(x), \neg P(a), \neg Q(b)\}$. Disjunkt $P(x) \vee Q(x)$ fundamentaalnäide on $P(a) \vee Q(a)$ ja $P(b) \vee Q(b)$.

Tulemus 21.3 (Herbrandi teoreem). Disjunktide hulk S on vasturääkiv parajasti siis, kui eksisteerib tema disjunktide fundamentaalnäidete lõplik vasturääkiv hulk S' .

Tõestus. Tarvilikkus. Olgu hulk S vasturääkiv. Hakkame konstrueerima hulga S täielikku semantilist puud. Peatame konstrueerimise igas kummutavas tipus: kui selgub, et hulga S mingi disjunkt C fundamentaalnäide C' on selles (osaliselt valmis) H-interpretatsioonis väär. Lõpuks peame saama sellise (lõpliku) täieliku semantilise puu, mille kõik lehed on kummutavad tipud. (Kui jääks mõni haru, mis ei lõpe kummutava tipuga, siis selles H-interpretatsioonis oleksid S kõik disjunktid tõesed ja S poleks vasturääkiv.) Kogume kokku kõik fundamentaalnäited kummutavatest tippudest, saame hulga $S' = \{C'_1, C'_2, \dots, C'_n\}$. See hulk on vasturääkiv, sest iga H-interpretatsiooni korral leidub mingi disjunkt C'_i , mis selles H-interpretatsioonis on väär.

Piisavus. Leidugu hulga S disjunktide fundamentaalnäidete lõplik vasturääkiv hulk S' . Seega S' disjunktide konjunktsioon on väär kõikvõimalikes interpretatsioonides, sh H-interpretatsioonides. Vaatame S' suvalist H-interpretatsiooni I' . See on osa hulga S mingi(te)st H-interpretatsiooni(de)st I . Fundamentaalnäidete hulk S' on väär interpretatsioonis I (mis on täpsem kui I'). Järelikult on S väär ka selles interpretatsioonis I . Kuna I oli suvaline H-interpretatsioon, siis järelikult on S väär igas oma H-interpretatsioonis ja tulemuse 21.2 kohaselt vasturääkiv.

Näide 21.12. Vaatleme valem $\forall xP(x) \& \exists y\neg P(y)$. Tema standardkuju võib esitada disjunktide hulga $S = \{P(x), \neg P(f(x))\}$. See hulk on vasturääkiv, sest leidub tema fundamentaalnäidete vasturääkiv hulk $\{P(f(a)), \neg P(f(a))\}$. Tulemuse 21.1 põhjal on esialgne valem samaselt väär.

Herbrandi teoreemi põhjal saab teoreemi automaatse tõestamise ülesannet konstruktiivselt lahendada: genereerime valemi standardkuju esitava disjunktide hulga S fundamentaalnäidete hulki $S_0' \subseteq S_1' \subseteq \dots$. Iga hulga S_i' saamiseks ($i=0,1,\dots$) asendame hulga S disjunktides muutujad (kõikvõimalikel viisidel) Herbrandi universumi i -nda taseme elementidega. Kontrollime, kas S_i' on vasturääkiv, kasutades selleks suvalist lausearvutuses sobivat meetodit (sest disjunktides ei sisaldu muutujaid). Kui S_i' on vasturääkiv, siis Herbrandi teoreemi põhjal on ka S vasturääkiv. Vastasel korral genereerime hulga S_{i+1}' . Protsess võib predikaatarvutuse poollahenduvuse tõttu kesta lõpmatuseni. Kui aga S on tõepoolest vasturääkiv, siis garanteerib Herbrandi teoreem, et see protseduur avastab hulga S mingi vasturääkiva fundamentaalnäidete hulga.

Herbrandi teoreem oligi esimeste arvutil realiseeritud kummutamisprotseduuride aluseks.

Näide 21.13. Olgu $S = \{P(x), \neg P(f(x))\}$. Hakkame konstrueerima fundamentaalnäidete hulki. Kuna $H_0 = \{a\}$, siis

$$S_0' = \{P(a), \neg P(f(a))\}.$$

See hulk ei ole vasturääkiv. Leiame Herbrandi universumi 1. taseme

$$H_1 = \{a, f(a)\}$$

ja fundamentaalnäidete hulga

$$S_1' = \{P(a), \neg P(f(a)), P(f(a)), \neg P(f(f(a)))\}.$$

See hulk on vasturääkiv, sest sisaldab kontraarse paari $\{\neg P(f(a)), P(f(a))\}$. Järelikult on Herbrandi teoreemi kohaselt ka hulk S vasturääkiv.

Näide 21.14. Mõned patsiendid armastavad kõiki doktoreid. Ükski patsient ei armasta posijaid. Tõestada, et ükski doktor ei ole posija.

Esitame kõigepealt teadmised 1. järku predikaatarvutuse valemitega. Toome sisse predikaadid:

$P(x)$ – x on patsient,

$D(x)$ – x on doktor,

$Q(x)$ – x on posija,

$A(x,y)$ – x armastab y -t.

Eeldused võib esitada järgmiste valemitega:

$$F_1 = \exists x (P(x) \ \& \ \forall y (D(y) \rightarrow A(x,y))),$$

$$F_2 = \forall z (P(z) \rightarrow \forall t (Q(t) \rightarrow \neg A(z,t)))$$

ja väite valemina

$$G = \forall u (D(u) \rightarrow \neg Q(u)).$$

Teoreemi tõestamise probleemi lahendamiseks tuleb tõestada, et eeldustest F_1 ja F_2 järeldub loogiliselt väide, s.t et valem $F_1 \ \& \ F_2 \rightarrow G$ on samaselt tõene ehk tema eituse $F_1 \ \& \ F_2 \ \& \ \neg G$ on samaselt väär.

Kummutatava valemi saab teisendada disjunktide hulgaks (konjunktsiooni iga liikme võib muidugi teisendada eraldi):

$$S = \{P(a), \neg D(y) \vee A(a,y), \neg P(z) \vee \neg Q(t) \vee \neg A(z,t), D(b), Q(b)\}.$$

Tõestame Herbrandi teoreemi kasutades, et saadud disjunktide hulk on vasturääkiv. Kuna $H_0 = H = \{a,b\}$,

siis

$S_0' = S' = \{P(a), \neg D(a) \vee A(a,a), \neg D(b) \vee A(a,b), \neg P(a) \vee \neg Q(a) \vee \neg A(a,a), \neg P(a) \vee \neg Q(b) \vee \neg A(a,b), \neg P(b) \vee \neg Q(a) \vee \neg A(b,a), \neg P(b) \vee \neg Q(b) \vee \neg A(b,b), D(b), Q(b)\}$.

See fundamentaalnäidete hulk on vasturääkiv. Tõepoolest: oletame vastuväiteliselt, et leidub interpretatsioon I, mis rahuldab hulka S' . Selles interpretatsioonis peavad muuhulgas elementaarvalemid $P(a)$, $D(b)$, $Q(b)$ olema tõesed. Ka $A(a,b)$ peab olema tõene, sest muidu ei saaks valem $\neg D(b) \vee A(a,b)$ olla tõene. Sel juhul aga on valem $\neg P(a) \vee \neg Q(b) \vee \neg A(a,b)$ väär, s.t interpretatsioon I ei rahulda hulka S' . See aga on vastuolus eeldusega. Järelikult peab hulk S' olema vasturääkiv ja Herbrandi teoreemi kohaselt on ka S vasturääkiv. Tulemuse 21.1 kohaselt on sellisel juhul valem $F1 \ \& \ F2 \ \& \ \neg G$ samaselt väär ja teoreemi tõestamise probleem lahendatud.

Harjutusülesanded

1. Teisendada disjunktide hulgaks valem $\forall x (\neg P(x,a) \rightarrow (\exists y (P(y,g(x)) \& \forall z (P(z,g(x)) \rightarrow P(y,z))))))$.

Leida saadud disjunktide hulga Herbrandi universum, Herbrandi baas ja üks H-interpretatsioon.

2. Antud on disjunktide hulk $S = \{P(x), Q(y, f(y,z))\}$. Antud on hulga S interpretatsioon I:

$D = \{1, 2\}$;

$f(1,1) \ f(1,2) \ f(2,1) \ f(2,2) \ P(1) \ P(2) \ Q(1,1) \ Q(1,2) \ Q(2,1) \ Q(2,2)$

1 2 2 1 tõene väär väär tõene väär tõene

Leida vastav H-interpretatsioon I^* .

22. Resolutsioonimeetod

Põhiliseks raskuseks Herbrandi teoreemil põhinevate tõestusprotseduuride kasutamisel on fundamentaalnäidete hulkade S_i' kiire kasvamine i kasvades. Näiteks vaatleme disjunktide hulka

$S = \{P(x, g(x), y, h(x, y), z, k(x, y, z)), \neg P(u, v, e(v), w, f(v, w), x)\}$.

Siin

$H_0 = \{a\}$,

$H_1 = \{a, g(a), h(a, a), k(a, a, a), e(a), f(a, a)\}$

jne.

Fundamentaalnäidete hulgas S_0' on 2 elementi, hulgas S_1' juba $6 \cdot (6 \cdot 6 + 6 \cdot 6 \cdot 6) = 1512$ elementi. Esimene vasturääkiv fundamentaalnäidete hulk on aga alles S_5' . Kuna Herbrandi universumi 5. tasemel on ligikaudu 10^{64} elementi, siis hulgas S_5' on ligikaudu 10^{256} elementi.

22.1. Resolutsioonimeetod lausearvutuse jaoks

Def. Olgu antud disjunktid C_1 ja C_2 . Sisaldagu C_1 literaali L ning C_2 selle eitust $\neg L$. Kriipsutame maha L ja $\neg L$ (nn kontraarsed literaalid, ingl k *complementary literals*) ning moodustame allesjäänud literaalide disjunktisiooni. Saadud disjunktii nimetatakse disjunktide C_1 ja C_2 *resolvendiks*.

Näide 22.1. Disjunktide $P \vee R$ ja $\neg P \vee Q$ resolvendiks on disjunkt $R \vee Q$.

Näide 22.2. Disjunktide P ja $\neg P$ resolvendiks on tühidisjunkt, mida tähistame \square .

Näide 22.3. Disjunktidel $P \vee R$ ja $P \vee Q$ ei ole resolventi.

Tulemus 22.1. Disjunktide C_1 ja C_2 resolvent järeldub loogiliselt disjunktidest C_1 ja C_2 .

Tõestus. Olgu $C_1 = L \vee C_1'$ ja $C_2 = \neg L \vee C_2'$, kus C_1' ja C_2' on literaalide disjunksioonid. Oletame, et C_1 ja C_2 on tõesed (suvalises) interpretatsioonis I. Vaja on tõestada, et nende resolvent $C_1' \vee C_2'$ on samuti tõene selles interpretatsioonis. Interpretatsioonis I on kas L või $\neg L$ väär. Oletame, et L on väär. Siis C_1 ei saa koosneda ühestainsast literaalist ja C_1' peab interpretatsioonis I olema tõene. Seega resolvent $C_1' \vee C_2'$ on samuti tõene interpretatsioonis I. Analoogiliselt võib näidata, et kui $\neg L$ on väär (ja L on tõene) interpretatsioonis I, siis C_2' peab olema selles interpretatsioonis tõene. Järelikult on $C_1' \vee C_2'$ selles interpretatsioonis tõene.

Edaspidi tõestame, et disjunktide hulk on vasturääkiv parajasti siis, kui resolventide leidmise teel võib sellest hulgast tuletada tühidisjunkt (tulemus 22.5).

Def. Tuletusreeglit, mis genereerib disjunktide hulgast selle resolvendid, nimetatakse *resolutsiooniks*.

Def. Disjunkt C *tuletuseks* disjunktide hulgast S resolutsioonimeetodil nimetatakse järjendit C_1, C_2, \dots, C_n , kus iga C_i ($i=1, \dots, n$) kas kuulub hulka S või on eelmiste resolvent ning $C=C_n$.

Def. Disjunktide hulgast S tühidisjunkt tuletust resolutsioonimeetodil nimetatakse selle hulga *kummutamiseks* (e. vasturääkivuse tõestuseks).

Näide 22.4. Olgu antud lausearvutuse disjunktide hulk $S = \{(1) \neg P \vee Q, (2) \neg Q, (3) P\}$. Disjunktide (1) ja (2) resolvendiks on (4) $\neg P$ ning disjunktide (4) ja (3) resolvendiks on tühidisjunkt \square . Seega saab hulgast S tuletada resolutsioonimeetodil tühidisjunkt ehk hulga S kummutada.

Harjutusülesanne

(L. Carroll. Alice imedemaal.) Juba langes ta pilk väikesele klaaskastile laua all: ta avas selle ja leidis sealt väga väikese koogi, millele oli marjadest kujundatud kaunis kiri: SÖÖ MIND. „Hüva, ma söön,” ütles Alice, „ja kui see muudab mind suuremaks, siis ma ulatun võtmeni; aga kui väiksemaks, siis ma mahun ukse alt läbi; niisiis pääsen ma igal juhul aeda ning pole vaja muretseda, mis juhtub!”

Olgu

- s – muudab mind suuremaks,
- u – ulatun võtmeni,
- v – muudab mind väiksemaks,
- m – mahun ukse alt läbi,
- p – pääsen aeda.

Kasutada resolutsioonimeetodit lausearvutuse jaoks, et määrata, kas p järeldub loogiliselt lausete hulgast F, mis on saadud eespool toodud katkendis esitatud informatsiooni tõlkimisel lausearvutusse, kasutades valemeid s, u, v, m, p. Eeldada, et „igal juhul” tähendab seda, et Alice ei saa muutuda samaaegselt suuremaks ja väiksemaks.

22.2. Resolutsioonimeetod 1. järku predikaatarvutuse jaoks

Asume nüüd laiendama resolutsioonimeetodit lausearvutuselt predikaatarvutusele. Resolventide leidmisel on tarvis, et kahes disjunktis leiduksid kontraarsed literaalid. Kui

disjunktid ei sisalda muutujaid, siis saab resolvendi leida niisamuti nagu lausearvutuses. Näiteks disjunktide $P(a) \vee Q(b)$ ja $\neg P(a) \vee R(a)$ resolvendiks on disjunkt $Q(b) \vee R(a)$. Kui aga disjunktid sisaldavad muutujaid, siis tuleb neile esmalt rakendada asendus, et unifitseerida teatavad literaalid.

22.2.1. Asendus ja unifikaator

Def. *Asenduskomponendiks* nimetatakse avaldist $\{t/v\}$, kus v on muutuja ja t – term, mis ei ole v (loe: „paiguta term t muutuja v asemele”).

Def. Asenduskomponentide hulka $\alpha = \{t_1/v_1, \dots, t_n/v_n\}$, kus kõik muutujad on erinevad, nimetatakse *asenduseks*.

Asenduse α rakendamise tulemust valemile C nimetatakse C *näiteks* (instance) ja tähistatakse $C\alpha$ (kõikide muutujate v_i kõik esinemised asendatakse vastavate termidega t_i ; $i=1, \dots, n$; tekkinud uued muutujad jäävad asendamata).

Näide 22.5. Asenduse $\alpha = \{g(a)/x, f(z)/y\}$ rakendamise tulemusel valemile $R(x, f(y))$ saame selle valemi näite $R(g(a), f(f(z)))$.

Def. Asenduste α ja β *kompositsiooniks* $\alpha\beta$ nimetatakse asendust, mis saadakse asenduse β rakendamisel asenduse α kõikidele termidele ja tulemusele asenduse β selliste asenduskomponentide lisamisel, mille muutujad puuduvad asenduses α .

Näide 22.6. Olgu $\alpha = \{f(x,y)/z\}$ ja $\beta = \{a/x, b/y, c/w, d/z\}$. Siis $\alpha\beta = \{f(a,b)/z, a/x, b/y, c/w\}$.

Resolventide leidmisel võib olla vaja unifitseerida kaks (või enam) avaldist, s.t leida selline asendus, mis teeks avaldised identseteks.

Def. Öeldakse, et literaalide hulk $W = \{L_1, \dots, L_m\}$ on *unifitseeritav*, kui leidub selline asendus α , et $L_i\alpha = \dots = L_m\alpha$. Asendust α nimetatakse hulga W *unifikaatoriks*.

Näide 22.7. Olgu literaalide hulk $W = \{R(f(y), x), R(f(b), f(a))\}$.

Asendus $\alpha = \{f(a)/x, b/y\}$ on hulga W unifikaator, sest $R(f(y), x)\alpha = R(f(b), f(a))$ ja $R(f(b), f(a))\alpha = R(f(b), f(a))$.

Def. Literaalide hulga $W = \{L_1, \dots, L_m\}$ *lihtsaimaks* (kõige üldisemaks) *unifikaatoriks* (*most general unifier*) nimetatakse sellist unifikaatorit α , et kui β on samuti hulga W unifikaator, siis leidub asendus γ nii, et $L_i\alpha\gamma = L_i\beta$ ($i=1, \dots, m$).

Näide 22.8. Olgu $W = \{P(a,y,z), P(x,y,z)\}$.

Hulk W on unifitseeritav, lihtsaim unifikaator on $\sigma = \{a/x\}$.

Unifikaatorid on ka näiteks $\alpha = \{a/x, b/y\}$ ja $\beta = \{a/x, b/y, c/z\}$.

22.2.2. Unifitseerimisalgoritm

Algoritmi, mis leiab unifitseeritava literaalide hulga lihtsaima unifikaatori ja teatab ebaõnnestumisest, kui hulk pole unifitseeritav, nimetatakse *unifitseerimisalgoritmiks*. Algoritm põhineb järgmisel ideel.

Vaatleme avaldise (literaale) $P(a)$ ja $P(x)$. Need ei ole identsed, vaid erinevad teineteisest selle poolest, et esimeses avaldises on konstant a , teises aga selsamal kohal muutuja x . Et ühtlustada avaldise, tuleb kõigepealt leida nende erinevus, seejärel aga püüda seda asendusega kõrvaldada. Vaadeldavas näites on erinevuste hulk $\{a, x\}$. Kuna x on muutuja, siis selle võib asendada konstandiga a , s.t rakendada asendust $\{a/x\}$ ja erinevus ongi kõrvaldatud.

Def. Ütleme, et sümbolite hulk on *alfabeetiliselt järjestatud*, kui nad paiknevad järjekorras: muutujate, funktsioonide, predikaatide sümbolid (iga grupp eraldi alfabeetilisest järjestuses) ja eituse märk.

Def. Ütleme, et avaldiste (literaalide) hulk on *leksikograafiliselt järjestatud*, kui lühemad avaldised eelnevad pikematele ning võrdse pikkusega avaldistest on eespool see, mille esimene sümbol (esimesed sümbolid) on alfabeetilisest järjestuses eespool.

Avaldiste hulga W unifitseerimisalgoritm

1. $k:=0$; $\sigma_k:=\varepsilon$ (tühiasendus).

2. Kui $W\sigma_k$ ei ole üheelemendiline hulk, siis mine 3. Vastasel korral $\sigma :=\sigma_k$, lõpeta.

3. Vaatle iga avaldist hulgas $W\sigma_k$ kui sümbolite järjendit ja eralda vasakpoolseimad osaavaldised, mis hulga $W\sigma_k$ kõigil elementidel ei ole identsed. Järjesta need osaavaldised leksikograafiliselt. Saad hulga B_k – see on hulga $W\sigma_k$ nn. erinevuste hulk. Olgu hulga B_k esimene ja teine element vastavalt v_k ja t_k . Kui v_k on muutuja, mis ei sisaldu avaldises t_k , siis

$\sigma_{k+1} := \sigma_k\{t_k/v_k\}$; $k := k+1$, mine 2. Vastasel korral ebaõnn (hulk W pole unifitseeritav), lõpeta.

Unifitseerimisalgoritm lõpetab töö igal lõplikul mittetühjal avaldiste hulgal W , sest vastupidisel juhul genereeriks ta lõplike mittetühjade hulkade lõpmatu jada $\{W\sigma_k \mid k=0,1,\dots\}$, kus iga järgmine hulk sisaldab ühe muutuja vähem kui eelmine, mis on aga võimatu, sest W sisaldab ainult lõpliku hulga erinevaid muutujaid.

Tulemus 22.2. Kui W on lõplik mittetühi unifitseeritav literaalide hulk, siis unifitseerimisalgoritm lõpetab töö 2. sammul ja viimane σ_k on hulga W lihtsaim unifikaator.

Tõestus. Olgu W unifitseeritav. Järelikult leidub tema unifikaator θ . Tõestame induktsiooniga, et $\theta = \sigma_0\lambda_0$, $\theta = \sigma_1\lambda_1, \dots$, $\theta = \sigma_k\lambda_k$, kus $\sigma_0, \sigma_1, \dots, \sigma_k$ on need asendused, mida unifitseerimisalgoritm leiab ning σ_k on hulga W lihtsaim unifikaator.

Induktsiooni baas. Olgu $\lambda_0 = \theta$. Unifitseerimisalgoritm $\sigma_0 = \varepsilon$ ning $\theta = \sigma_0\lambda_0$.

Induktsiooni hüpotees. Kehtigu iga $i=0, \dots, n$ korral $\theta = \sigma_i\lambda_i$. Samm: näitame, et $\theta = \sigma_{n+1}\lambda_{n+1}$.

Kui $W\sigma_n$ on üheelemendiline, siis unifitseerimisalgoritm lõpetab 2. sammul ja σ_n on lihtsaim unifikaator. Kui $W\sigma_n$ pole üheelemendiline, siis unifitseerimisalgoritm leiab erinevuste hulga B_n . Olgu $B_n = \{v_n, t_n, \dots\}$. Teada on, et asendus λ_n unifitseerib hulga B_n (sest $\sigma_n\lambda_n$ unifitseerib hulga W), järelikult ka $v_n\lambda_n = t_n\lambda_n$. Siin v_n peab olema muutuja ja $v_n \notin t_n$ (vastasel korral poleks W unifitseeritav). Seega moodustab unifitseerimisalgoritm asenduse $\sigma_{n+1} = \sigma_n\{t_n/v_n\}$. Tähistame $\lambda_{n+1} = \lambda_n \setminus \{t_n\lambda_n/v_n\}$ (s.o asendus, mis on saadud asendusest λ_n ühe asenduskomponendi $t_n\lambda_n/v_n$ väljajätmisel). $t_n\lambda_{n+1} = t_n(\lambda_n \setminus \{t_n\lambda_n/v_n\}) = t_n\lambda_n$ (sest $\{t_n\lambda_n/v_n\}$ toimib termile t_n kui tühiasendus, kuna $v_n \notin t_n$). Siis

$\{t_n/v_n\}\lambda_{n+1}=\{t_n\lambda_{n+1}/v_n\} \cup \lambda_{n+1} =\{t_n\lambda_n/v_n\} \cup \lambda_{n+1}=\{t_n\lambda_n/v_n\} \cup (\lambda_n \setminus \{t_n\lambda_n/v_n\}) = \lambda_n$. Seega $\theta = \sigma_n\{t_n/v_n\}\lambda_{n+1}=\sigma_{n+1}\lambda_{n+1}$. Sellega on induktsiooni samm tõestatud, mis tõestabki kogu teoreemi.

Näide 22.9. Rakendame unifitseerimisalgoritmi literaalide hulga

$$W = \{P(x, z, v), P(x, f(y), y), P(x, z, b)\}.$$

Saame

$$k=0, \sigma_0:=\varepsilon, W\sigma_0=W.$$

Erinevuste hulk

$$B_0=\{z, f(y)\}.$$

Siin $v_0=z, t_0=f(y)$.

Nüüd

$$\sigma_1 := \sigma_0\{f(y)/z\}, k=1.$$

Hulk

$W\sigma_1=\{P(x, f(y), v), P(x, f(y), y), P(x, f(y), b)\}$ ei ole üheelemendiline, erinevuste hulk on

$$B_1=\{v, y, b\}.$$

Nüüd

$$\sigma_2 := \sigma_1\{y/v\} = \{f(y)/z\}\{y/v\} = \{f(y)/z, y/v\},$$

$k=2$.

Hulk

$W\sigma_2=\{P(x, f(y), y), P(x, f(y), b)\}$ ei ole üheelemendiline, erinevuste hulk on

$$B_2=\{y, b\}.$$

Saame

$$\sigma_3 = \sigma_2\{b/y\} = \{f(y)/z, y/v\}\{b/y\} = \{f(b)/z, b/v, b/y\},$$

$k=3$.

Nüüd

$$W\sigma_3=\{P(x, f(b), b)\},$$

seega on

$$\sigma = \sigma_3 = \{f(b)/z, b/v, b/y\}$$

hulga W lihtsaim unifikaator.

Näide 22.10. Rakendame unifitseerimisalgoritmi literaalide hulga

$$W=\{Q(f(a), g(x)), Q(y, y)\}.$$

Saame

$$k=0, \sigma_0:=\varepsilon, W\sigma_0=W.$$

Erinevuste hulk

$$B_0=\{y, f(a)\},$$

seega $\sigma_1 := \sigma_0\{f(a)/y\}$.

Nüüd

$$k=1, W\sigma_1=\{Q(f(a), g(x)), Q(f(a), f(a))\}, \text{ erinevuste } B_1=\{f(a), g(x)\}.$$

Kuna $f(a)$ ei ole muutuja, siis pole hulk W unifitseeritav.

22.2.3. Resolutsioonimeetod predikaatarvutuse jaoks

Def. Olgu disjunktid C_1 ja C_2 ühiste muutujateta. Sisaldagu C_1 literaali L_1 ja C_2 literaali $-L_2$ ning leidugu hulga $\{L_1, L_2\}$ lihtsaim unifikaator σ . Disjunktide C_1 ja C_2 *binaarseks resolvendiks* nimetatakse disjunkti $(C_1-L_1)\sigma \vee (C_2-L_2)\sigma$.

Näide 22.11. Olgu antud disjunktid

$$C_1 = \neg P(v, z, v) \vee P(y, z, y),$$

$$C_2 = P(w, h(x, x), w).$$

Tähistame

$$L_1 = \neg P(v, z, v),$$

$$L_2 = P(w, h(x, x), w).$$

Hulga $\{P(v, z, v), P(w, h(x, x), w)\}$ lihtsaim unifikaator on $\sigma = \{w/v, h(x, x)/z\}$.

Seega disjunktide

C_1 ja C_2 binaarne resolvent on

$$P(y, z, y) \{w/v, h(x, x)/z\} = P(y, h(x, x), y).$$

Kehtib järgmine

Tulemus 22.3. Disjunktide C_1 ja C_2 binaarne resolvent järeldub loogiliselt disjunktide hulgast $\{C_1, C_2\}$.

Def. Kui disjunkti C kaks või enam literaali on unifitseeritavad lihtsaima unifikaatoriga σ , siis disjunkti $C\sigma$ nimetatakse disjunkti C *kleebiseks* (ingl *factor*).

Näide 22.12. Disjunkti $P(x, y) \vee P(a, f(a))$ kleebis on $P(a, f(a))$. Disjunkt $P(x) \vee Q(y) \vee P(f(x))$ pole kleebist.

Def. Disjunktide C_1 ja C_2 *resolvendiks* nimetatakse 1) C_1 ja C_2 binaarset resolventi, 2) C_1 kleebise ja C_2 binaarset resolventi, 3) C_1 ja C_2 kleebise binaarset resolventi ja 4) C_1 ja C_2 kleebiste binaarset resolventi.

Näide 22.13. Olgu

$$C_1 = R(y, g(a)) \vee R(y, g(x)) \vee P(x),$$

$C_2 = \neg R(z, g(a)) \vee \neg P(z)$. Resolvendid on $R(z, g(x)) \vee P(x) \vee \neg P(z)$ (disjunktide C_1 ja C_2 binaarne resolvent), aga ka $P(a) \vee \neg P(z)$ (C_1 kleebise ja C_2 binaarne resolvent).

Olgu antud disjunktide hulk S , mis koosneb n disjunktist. Hulga S kõik resolvendid võib leida järgmise algoritmi kohaselt.

(i) Järjestada S kõik disjunktid 1 kuni n ; $k:=1$.

(ii) Leida k -nda disjunkti kõikvõimalikud resolvendid disjunktidega $k+1$ kuni n .

(iii) $k:=k+1$. Kui $k < n$, siis minna (ii).

Näide 22.14. Olgu $S = \{Q(a), R(a), \neg Q(x) \vee \neg R(x) \vee T(a), \neg T(y)\}$. Tuletame hulgast S resolutsioonimeetodil tühidisjunkti. Resolventide leidmisel kasutame äsja toodud algoritmi. Tuletuse esitame graafina, mille tippudele vastavad hulga S disjunktid ja resolvendid (joonis 22.1).

Induktsiooni samm. Oletame, et väide kehtib iga $k(S) < n$ korral, ja näitame, et ta kehtib ka $k(S) = n$ korral.

Kui tühidisjunkt sisaldub hulgas S , siis on see tõestatud. Seega oletame, et tühidisjunkt ei sisaldu hulgas S . Olgu C selline disjunkt hulgas S , kus $C = A \vee L$ (L on literaal). Seega $S = S' \cup C$. Vaatame hulki $S_1 = S' \cup A$ ja $S_2 = S' \cup L$. Siin $k(S_2) \leq k(S_1) < k(S) = n$. Kui S_1, S_2 oleksid vasturääkivad, siis induktiivse eelduse kohaselt saaks neist tuletada tühidisjunkt. Kas nad aga on vasturääkivad?

- Näitame, et igas interpretatsioonis, mis rahuldab $C = A \vee L$, on S mingi valem väär. Olgu M disjunkt C kõigi mudelite hulk. Eelduse kohaselt on $S = S' \cup C$ vasturääkiv, järelikult ei rahulda teda ükski interpretatsioon m mudelite hulgast M . Vaatame suvalist interpretatsiooni m . Hulgas S peab leiduma valem C^* , mis interpretatsioonis m on väär. Järelikult ei rahulda see interpretatsioon hulki $S_1 = S' \cup A$ ja $S_2 = S' \cup L$.
- Igas interpretatsioonis, milles disjunkt C on väär, on valemid A ja L väärad.

Järelikult on S_1 ja S_2 vasturääkivad.

Kas nende hulkade vasturääkivusest järeldub, et hulgast S on tuletatav tühidisjunkt?

Tähistame $R^k(S)$ – disjunktide hulk, mis on saadud hulgast S k korda resolvente leides. Eelnevast on teada, et iga i ja j korral kuulub tühidisjunkt hulkadesse $R^i(S)$ ja $R^j(S)$. Rakendame nüüd i esimest sammu hulgale $S = S' \cup (A \vee L)$. Saame kas tühidisjunkt (millega väide on tõestatud) või L . Viimasel juhul rakendame j viimast sammu hulgale $S' \cup L = S_2$ ja saamegi tühidisjunkt.

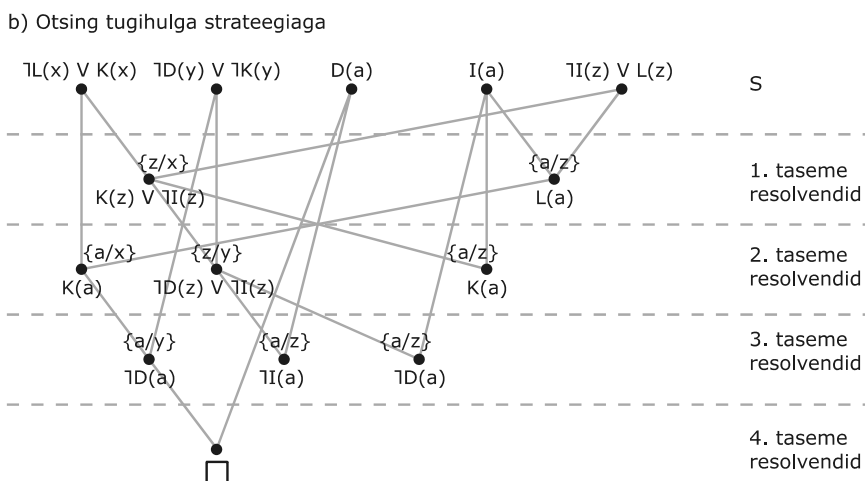
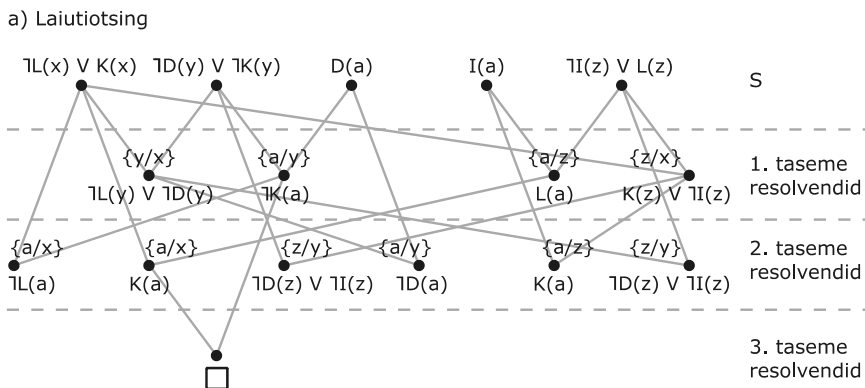
Piisavus. Oletame vastuväiteliselt, et leidub hulga S mudel m . Kuna m rahuldab suvalisi disjunkte C_1 ja C_2 hulgast S , siis ta rahuldab ka nende resolventi. Järelikult rahuldab m kõiki resolvente, mida saab tuletada hulgast S . Aga üks neist tuletatavatest resolventidest on tühidisjunkt \Rightarrow vastuolu.

Resolutsioonimeetodi täielikkus tagab selle, et kui disjunktide hulk S on vasturääkiv, siis saab temast lõplik arv kordi resolvente leides tuletada tühidisjunkt. Kui lõpliku arvu sammude järel pole osutunud võimalikuks tuletada tühidisjunkt, siis see ei tähenda veel, et S oleks rahuldav, sest võibolla õnnestuks tühidisjunkt tuletada suurema arvu sammude tulemusel.

Arusaadavalt ei osale tühidisjunkt tuletuses kõik resolvendid, mida saab hulga S disjunktidest lähtudes moodustada. Selle, millised resolvendid moodustada, määrab juhtimisstrateegia.

Mõned juhtimisstrateegiad

- Laiutiotsing: kõigepealt moodustatakse kõikvõimalikud resolvendid lähtehulga S disjunktidega (nn 1. tase), seejärel lisatakse lähtehulgale 1. taseme resolvendid ja moodustatakse nii saadud uue hulga kõik resolvendid (nn 2. tase). Uued resolvendid lisatakse jälle eelmisele disjunktide hulgale ja korratakse resolventide leidmist analoogilisel viisil, kuni lõpuks õnnestub leida tühidisjunkt (juhul kui lähtehulk on vasturääkiv).



Joonis 22.2. Vasturääkivuse tõestus resolutsioonimeetodil: a) laiutiotsinguga, b) tugihulga strateegiaga

Laiutiotsing on täielik. Juhtimisstrateegia täielikkus tähendab seda, et tema rakendamine viib protseduurini, mis lõpuks leiab kummutamise, juhul kui see eksisteerib. Samas on laiutiotsing ebaefektiivne, sest moodustatakse palju ebavajalikke (tühidijsunkti tuletuses mitte osalevaid) resolvente.

- Tugihulga strateegia: vähemalt üks „vanem” igas resolvendis valitakse nende disjunktide hulgast, mis tekivad väite eitusest või selle järglastest (s. o nn *tugihulk*).

Valemi kummutamise puhul on tegu vastuväitelise tõestusega, mistõttu on arusaadav, et vastuolu saab tekkida üksnes eitatava väite (s.t temale vastavate disjunktide ja nende osalusel moodustatud resolventide) tõttu. Kui see nii poleks, siis peaksid eeldused olema omavahel vastuolus.

Võib näidata, et kui eksisteerib mingi kummutamine, siis eksisteerib ka tugihulga strateegiat kasutav kummutamine. Seega on tugihulga strateegia täielik ja muidugi efektiivsem kui laiutiotsing.

Näide 22.14. Disjunktide hulk

$$S = \{\neg L(x) \vee K(x), \neg D(y) \vee \neg K(y), D(a), I(a), \neg I(z) \vee L(z)\}$$

on vasturääkiv, sest resolutsioonimeetodil saab temast tuletada tühidisjunkt. Joonisel 22.2 on toodud kaks tuletust: a) kasutades lautiotsingut, b) kasutades tugihulga strateegiat.

- Üksliikmete eelistamise strateegia: esmalt leitakse resolvendid selliseid disjunkte kasutades, mis koosnevad ühestainsast literaalist.
- Sisendstrateegia: iga resolvendi vähemalt üks „vanem” kuulub hulka S (s.t on lähtedisjunkt). See strateegia on küllaltki efektiivne, kuid kahjuks pole täielik.
- Lineaarne strateegia: iga resolvendi üks „vanem” on kas lähtedisjunkt või teise „vanema” eellane.

Harjutusülesanded

1. Olgu antud asendus $\Theta = \{a/x, b/y, g(x,y)/z\}$ ja valem $E = P(h(x),z)$. Leida valemi E näide $E\Theta$.
2. Leida asenduste $\Theta = \{f(y)/x, z/y\}$ ja $\Lambda = \{a/x, b/y, y/z\}$ kompositsioon.
3. Kas literaalide hulk W on unifikseeritav? Kui jah, siis leida tema lihtsaim unifikaator, rakendades unifikseerimisalgoritmi. a) $W = \{Q(x,y,z), Q(u,h(v,v),u)\}$; b) $W = \{P(a,x,f(x)), P(a,y,y)\}$.
4. Leida disjunktide C_1 ja C_2 resolvendid (kui võimalik):
 - a) $C_1 = \neg P(x) \vee Q(x,b)$, $C_2 = P(a) \vee R(x)$;
 - b) $C_1 = \neg P(x) \vee Q(x,x)$, $C_2 = \neg Q(a,f(a))$.
 - c) $C_1 = \neg P(v,z,v) \vee P(w,z,w)$ ja $C_2 = P(w,h(x,x),w)$.
 - d) $C_1 = \neg P(x,y,u) \vee \neg P(y,z,w)$ ja $C_2 = P(g(x,y),x,y)$.
5. Tõestada resolutsioonimeetodil, et disjunktide hulk S on vasturääkiv:
 - a) $S = \{\neg P(y) \vee R(f(y)), \neg R(z), P(a)\}$.
 - b) $S = \{\neg T(x) \vee K(x), T(a), H(b,a), \neg K(z) \vee \neg H(b,z)\}$.
 - c) $S = \{Q(a) \vee R(x), \neg Q(x) \vee R(x), \neg R(a) \vee \neg T(a), T(x)\}$.
 - d) $S = \{\neg P(x), P(x) \vee Q(x), \neg Q(x) \vee R(x), \neg R(x) \vee S(x), \neg R(x) \vee \neg S(x) \vee T(x), P(x) \vee \neg T(x)\}$.
 - e) $S = \{P(x) \vee Q(x), \neg Q(f(z)), \neg P(f(z)) \vee R(z), \neg R(w)\}$.

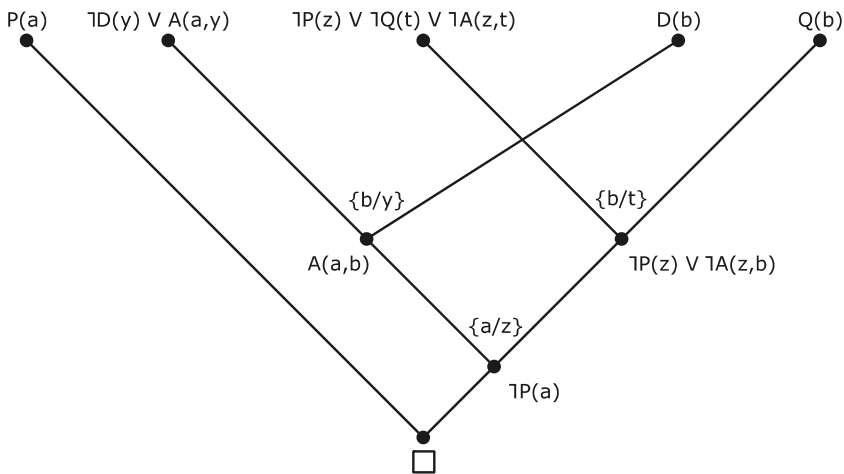
23. Teoreemide automaatne tõestamine resolutsioonimeetodil

Vaatame veel mõningaid näiteid resolutsioonimeetodi rakendamise kohta.

Näide 23.1. Näites 21.14 lahendasime patsientide ja doktorite probleemi, kasutades Herbrandi teoreemi. Lahendame nüüd sellesama probleemi resolutsioonimeetodil, s.t tõestame, et disjunktide hulk

$$S = \{P(a), \neg D(y) \vee A(a,y), \neg P(z) \vee \neg Q(t) \vee \neg A(z,t), D(b), Q(b)\}$$

on vasturääkiv. Tõestuse (tühidisjunkt tuletuse) saab esitada graafina (joonis 23.1).



Joonis 23.1. Vasturääkivuse tõestus resolutsioonimeetodil

Näide 23.2. Kes oskab lugeda, see on kirjaoskaja. Delfiinid ei ole kirjaoskajad. Mõnel delfiinil on intellekt. Tõestada, et mõned, kellel on intellekt, ei oska lugeda.

Kõigepealt esitame teadmuse 1. järku predikaatarvutuse valemitega. Toome sisse järgmised predikaadid:

- $L(x)$ – x oskab lugeda,
- $K(x)$ – x on kirjaoskaja,
- $D(x)$ – x on delfiin,
- $I(x)$ – x omab intellekti.

Siis eeldused

- $F1 = \forall x (L(x) \rightarrow K(x))$,
- $F2 = \forall y (D(y) \rightarrow \neg K(y))$,
- $F3 = \exists z (D(z) \& I(z))$

ja väide

$$G = \exists u (I(u) \& \neg L(u)).$$

Väite eitus: $\neg G = \forall u (\neg I(u) \vee L(u))$.

Püstitame teoreemi tõestamise probleemi: tõestada, et eeldustest $F1, F2, F3$ järeljub loogiliselt väide G . Seega tuleb kummutada valem $F1 \& F2 \& F3 \& \neg G$.

Leiame valemile vastava disjunktide hulga (konjunktsiooni iga liikme võib teisendada eraldi):

$$S = \{\neg L(x) \vee K(x), \neg D(y) \vee \neg K(y), D(a), I(a), \neg I(z) \vee L(z)\}.$$

See hulk on vasturääkiv, sest resolutsioonimeetodil saab temast tuletada tühisdiskontsi (vt joonis 23.2). Järelikult on valem kummutatud ja teoreemi tõestamise probleem sellega lahendatud.

disjunktide 3 ja 7 resolvent on

(11) $\neg T(a)$.

Disjunktide 8 ja 10 resolvent on

(12) $T(f(a))$,

disjunktide 5 ja 9 resolvent on

(13) $K(a) \vee N(f(a))$,

disjunktide 10 ja 13 resolvent on

(14) $N(f(a))$,

disjunktide 7 ja 14 resolvent on

(15) $\neg T(f(a))$

ning disjunktide 12 ja 15 resolvendiks on tühidisjunkt. Seega on probleem lahendatud.

Viited

Tanel Tammeti automaatne teoreemitõestaja Gandalf <http://deephought.ttu.ee/it/gandalf/> (18.02.2010)

Automaatsete teoreemitõestajate maailmameistrivõistlused (The World Championship for 1st Order Automated Theorem Proving, CASC <http://www.cs.miami.edu/~tptp/CASC/> (18.02.2010)

Teoreemide automaatse tõestamise alane konverents (Conference on Automated Deduction, CADE) <http://www.cadeconference.org/> (18.02.2010)

Harjutusülesanded

1. Tudengid on kodanikud. Tõestada, et tudengite hääled on kodanike hääled.

- Esitada iga lause 1. järku predikaatarvutuse valemina ja püstitada teoreemi tõestamise probleem.
- Teisendada saadud valem disjunktide hulgaks.
- Lahendada resolutsioonimeetodil.

2. Jüril on koer. Iga koeraomanik on loomaarmastaja. Ükski loomaarmastaja ei piina looma. Kassi, nimega Kiti, piinati. Piinaja sai olla kas Jüri või Jaan. Esitada teadmus 1. järku predikaatarvutuses, püstitada teoreemi tõestamise ülesanne ja tõestada *resolutsioonimeetodil*, et Jaan piinas Kitit.

24. Küsimustele vastamine kui teoreemide tõestamine

Küsimusele vastamist võib käsitleda kui teoreemi tõestamist. Vaatame siin kolme küsimuste klassi:

- kas-küsimused, mille vastuseks võib olla *jah* või *ei* (Kas mees läks metsa?);
- eriküsimused, mis vormistatakse küsivate ase- ja määrsõnade abil (Kes läks metsa? Kuhu mees läks? jms);
- küsimused, mille vastuseks on (agendi) tegevuste järjend (s.o programmi süntees).

A. Kas-küsimusele vastamine

Kas-küsimusele vastamiseks tuleb püstitada teoreemi tõestamise probleem. Eeldusteks on antud faktid. Väiteks võib olla kas-küsimuse deklaratiivne osa või selle eitav. Seega on võimalik püstitada kaks teoreemi tõestamise probleemi. Vastus küsimusele on jaatav, kui õnnestub lahendada esimene probleem (s.t tõestada, et eeldustest järgeldub loogiliselt väide), ning eitav, kui õnnestub lahendada teine probleem (s.t tõestada, et eeldustest

järeldub loogiliselt väite eituse). Võib ka osutada, et kumbagi probleemi ei saa lahendada – sellisel juhul pole võimalik küsimusele ei jaatavalt ega eitavalt vastata.

Näide 24.1. Kõik inimesed on surelikud. Confucius on inimene. Kas Confucius on surelik?

Püstitame teoreemi tõestamise probleemi. Kasutades predikaate $I(x)$ – x on inimene ja $S(x)$ – x on surelik ning konstanti $c=Confucius$, saame esitada eeldused:

$$F_1 = \forall x (I(x) \rightarrow S(x)),$$

$$F_2 = I(c)$$

ja väite:

$$G = S(c).$$

Püüame tõestada, et eeldustest F_1 ja F_2 järeldub loogiliselt väide G . Kummutatavale valemile $F_1 \ \& \ F_2 \ \& \ \neg G$ vastav disjunktide hulk on $\{(1) \neg I(x) \vee S(x), (2) I(c), (3) \neg S(c)\}$.

Esimese ja teise disjunktli resolvent on $S(c)$ ning saadud resolvendi ja kolmanda disjunktli resolvent on tühidisjunkt. Seega on disjunktide hulk vasturääkiv ja vastus kas-küsimusele jaatav: Confucius on surelik.

B. Eriküsimusele vastamine

Analoogiliselt eelnevaga püstitame teoreemi tõestamise probleemi, võttes eeldusteks antud faktid ja väiteks küsimuse, milles küsisõnale vastab muutuja, s.t väidame, et eksisteerib objekt, kelle/mille kohta on esitatud küsimus. Meid huvitab selle muutuja väärtuse leidmine. Selleks lisatakse tõestusse nn muutuja jälgimise mehhanism: väite eituse $\neg G(x)$ asemel kasutatakse disjunktli $\neg G(x) \vee \text{ANS}(x)$. Tühidisjunktli asemel tuletatakse elementaarvalem $\text{ANS}(a)$, kus a on muutuja x väärtus, mis annabki vastuse küsimusele.

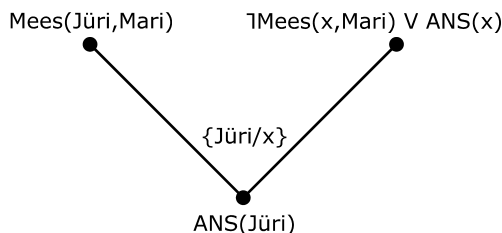
Näide 24.2. Antud on fakt *Jüri on Mari mees*. Leida vastus küsimusele *Kes on Mari mees?*

Antud juhul

$$F = \text{Mees}(\text{Jüri}, \text{Mari}),$$

$$G = \exists x \text{ Mees}(x, \text{Mari}).$$

Vaatleme disjunktide hulka $\{\text{Mees}(\text{Jüri}, \text{Mari}), \neg \text{Mees}(x, \text{Mari}) \vee \text{ANS}(x)\}$. Tuletus resolutsioonimeetodil on kujutatud joonisel 24.1. Seega vastus küsimusele on Jüri.



Joonis 24.1. Eriküsimusele vastamine

Näide 24.3. Kui x on y isa ja y on z isa, siis x on z vanaisa. Igaühel on isa. Kes on suvalise isiku vanaisa?

Toome sisse predikaadid

$I(x,y)$ – x on y -i isa,

$V(x,y)$ – x on y -i vanaisa.

Siis eeldused

$F_1 = \forall x \forall y \forall z (I(x,y) \& I(y,z) \rightarrow V(x,z))$,

$F_2 = \forall u \exists t I(t,u)$

ja väide (väidame, et suvalisel isikul v leidub vanaisa):

$G = \forall v \exists w V(w,v)$.

Väite eitus: $\neg G = \exists v \forall w \neg V(w,v)$ (s.t väidame, et leidub keegi suvaline v , kellel ei ole vanaisa).

Saame disjunktide hulga

$\{(1) \neg I(x,y) \vee \neg I(y,z) \vee V(x,z), (2) I(f(u), u), (3) \neg V(v,a) \vee \text{ANS}(v)\}$, kus $f(u)$ on muutujat t siduva olemasolukvantori elimineerimisel valitud (suvaline) Skolemi funktsioon (tõlgendus: $f(u)$ on u isa) ning a on muutujat v siduva olemasolukvantori elimineerimisel valitud (suvaline) konstant (tõlgendus: see isik, kellel väitsime mitte leiduvat vanaisa).

Alustame tõestust resolutsioonimeetodil. Disjunktide (1) ja (3) resolvent on

(4) $\neg I(x,y) \vee \neg I(y,a) \vee \text{ANS}(x)$,

disjunktide (2) ja (4) resolvent on

(5) $\neg I(y,a) \vee \text{ANS}(f(y))$,

disjunktide (2) ja (5) resolvent on $\text{ANS}(f(f(a)))$.

Seega: (suvalise) isiku a vanaisa on tema isaisa $f(f(a))$.

C. Tegevuskava kirjeldava programmi süntees

Vaatame selle kohta järgmist näidet.

Näide 24.4. Lahendame uuesti ahvi ja banaanide probleemi.

Esitame teadmuse 1. järku predikaatarvutuse valemitega. Selleks toome sisse predikaadid:

$P(x,y,z,s)$ – seisundis s asub ahv kohal x , kast kohal y ja banaanid kohal z ;

$R(s)$ – seisundis s ulatub ahv banaanideni.

Ahvi võimalikke tegevusi kirjeldame funktsioonidega:

$\text{minna}(x,y,s)$ – väärtuseks on seisund, mis saabub, kui seisundis s ahv läheb kohalt x kohale y ;

$\text{nihutada}(x,y,s)$ – väärtuseks on seisund, mis saabub, kui seisundis s ahv nihutab kasti kohalt x kohale y ;

$\text{ronida}(s)$ – väärtuseks on seisund, mis saabub, kui seisundis s ahv ronib kastile.

Siis eeldused on esitatavad järgmiste valemitega:

$F_1 = \forall x \forall y \forall z (P(x,y,z,s) \rightarrow P(y,y,z,\text{minna}(x,y,s)))$;

$F_2 = \forall x \forall y \forall z \forall s (P(x,x,z,s) \rightarrow P(y,y,z,\text{nihutada}(x,y,s)))$;

$F_3 = \forall s (P(c,c,c,s) \rightarrow R(\text{ronida}(s)))$;

$F_4 = P(a,b,c,s_0)$.

Küsimus: $G = R(s)$ – millises seisundis s ulatub ahv banaanideni?

Küsimusele vastamiseks vaatleme väite asemel selle eitust (oletades niisiis vastuväiteliselt, et otsitavat seisundit ei leidugi), mida täiendame vastuse predikaadiga (nagu eriküsimusele vastamiselgi):

$\neg R(s) \vee \text{ANS}(s)$.

Teoreemi tõestust resolutsioonimeetodil võib käsitleda kui vajalike tegevuste järjestit kirjeldava programmi sünteesi.

Valemile $F_1 \ \& \ F_2 \ \& \ F_3 \ \& \ F_4 \ \& \ (\neg R(s) \vee \text{ANS}(s))$ vastab disjunktide hulk:

$S = \{P(y,y,z, \text{minna}(x,y,s)) \vee \neg P(x,y,z,s), \quad P(y,y,z, \text{nihutada}(x,y,s)) \vee \neg P(x,x,z,s),$
 $R(\text{ronida}(s)) \vee \neg P(c,c,c,s), P(a,b,c,s_0), \neg R(s) \vee \text{ANS}(s)\},$

millest resolutsioonimeetodil saame tuletada disjunkt

$\text{ANS}(\text{ronida}(\text{nihutada}(b,c, \text{minna}(a,b,s_0))))$.

Seega on ahvi tegevuskava banaanide kättesaamiseks järgmine: alustades algolekust s_0 minna kohalt a kohale b , seejärel nihutada kast kohalt b kohale c ja ronida kasti otsa.

Harjutusülesanded

1. Vihma sajab või on palav. Kui vihma sajab, siis on palav. Kui vihma ei saja, siis ei ole palav. Kas on õige, et kui on palav, siis vihma ei saja?

2. Marcus on mees. Marcus on Pompej elanik. Marcus sündis aastal 40. Kõik mehed on surelikud. Kõik Pompej elanikud surid vulkaanipurske tagajärjel aastal 79. Ükski surelik ei ela rohkem kui 150 aastat. Praegu on aasta 2010. Kas Marcus on elus?

3. Saarel elab kaks suguharu: tublid, kes alati tõtt räägivad, ja valelikud, kes alati valetavad. Rändur kohtas üht pärismaalast ja, kuulnud selle olevat tublide suguharust, palkas ta oma teenistusse. Nad läksid koos edasi ja nägid eemal teist pärismaalast. Rändur saatis oma uue teenri küsima, millisesse suguharusse too pärismaalane kuulub. Naastes ütles teener, et teine pärismaalane oli öelnud enda olevat tublide suguharust. Millisest suguharust oli ränduri teener?

4. Kohtu ees seisab kolm inimest, kellest igaüks võib olla kas pärismaalane või kolonisaator. Kohtunik teab, et pärismaalased räägivad tingimata õigust, kolonisaatorid aga valetavad. Seejuures ei tea kohtunik, kes kolmest kohtualusest on pärismaalane ja kes kolonisaator. Ta esitab vastava küsimuse esimesele, kuid ei saa vastusest aru. Seejärel küsib ta teiselt ja kolmandalt eraldi, mida vastas esimene. Teine vastab, et esimene väitnud enda olevat pärismaalase. Kolmas vastab, et esimene ütles enda olevat kolonisaatori. Kes olid teine ja kolmas?

5. Härra Bambus kutsus endale külla kolonel Musta, neiu Siiri ja professor Ploomi. Külaskäigu jooksul üks neist kolmest külalisest mõrvas härra Bambuse. Ainsad terariistad majas olid mutrivõti ja küünlajalg. Tuli politsei, uuris asja ja leidis järgmised asitõendid:

küünlajalal on täpselt kahe isiku sõrmejäljed,

mõrvar on puudutanud nii mutrivõtit kui ka küünlajalga.

Kahtlusalused väitsid järgmist.

Neiu Siiri:

Mina puudutasin kas küünlajalga või mutrivõtit, kuid mitte mõlemat.

Kolonel Must:

Mina puudutasin **täpselt** neidsamu terariistu mis professor Ploom.

Ma nägin, et neiu Siiri puudutas küünlajalga.

Prof. Ploom:

Mina puudutasin ainult küünlajalga.

Ma nägin, et kolonel Must puudutas mutrivõtit.

Teads on, et:

täpselt üks kahtlusalustest on süüdlane.

Kui kahtlusalune pole mõrvar, siis räägib ta tõtt.

Kes tappis härra Bambuse?

6. Antud on süsteem, mida kirjeldavad järgmised aksioomid:

- 1) $\forall x \forall y \forall z ((x+y)+z=x+(y+z))$ – assotsiatiivsus,
- 2) $\forall x \forall y \exists z (z+x=y)$ – vasakpoolse lahendi olemasolu,
- 3) $\forall x \forall y \exists z (x+z=y)$ – parempoolse lahendi olemasolu.

Leida selle süsteemi parempoolne ühik – selline y , et iga x korral $x+y=x$.

7. Loomad edestavad jooksus neid loomi, keda nad söövad. Lihasööjad loomad söövad teisi loomi. Jooksus edestamise suhe on transitiivne. Lõvid söövad sebrasid. Sebrad edestavad jooksus koeri. Koerad on lihasööjad loomad. Leida kaks looma, keda lõvid jooksus edestavad.

8. Toas, kus on ahv, paikneb kaks kasti. Ühes neist on banaan, teine on tühi. Ahv ei tea, kus on banaan. Kui ta aga asub kasti juures, võib ta sellesse vaadates leida banaani. Mida peab ahv tegema selleks, et leida banaan?

9. Kõik inimesed, kes ei ole vaesed ja on nutikad, on õnnelikud. Need inimesed, kes loevad, on nutikad. Jaan loeb ega ole vaene. Õnnelikel inimestel on põnev elu. Kas leidub keegi, kellel on põnev elu? Küsimusele vastamiseks püstitada teoreemi tõestamise probleem ja lahendada see resolutsioonimeetodil.

10. Igaüks, kes sooritab ajalooeksami ja võidab loteriiga, on õnnelik. Igaüks, kes on õppinud või on õnnelik, sooritab kõik eksamid. Jaan ei ole õppinud, aga on õnnelik. Igaüks, kes on õnnelik, võidab loteriiga. Kas Jaan sooritab ajalooeksami? Küsimusele vastamiseks püstitada teoreemi tõestamise probleem ja lahendada see resolutsioonimeetodil.

III.D. Teisi probleemilahendusmeetodeid

25. Tuletused semantilises võrgus ja freimide süsteemis

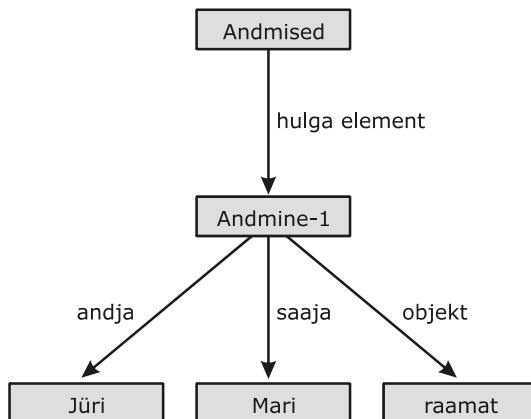
Nii semantilises võrgus kui ka freimide süsteemis põhineb tuletuste tegemine pärimisseosel. Näiteks kui on teada, et imetajad on elusolendid ja koerad on imetajad, ning see teadmus on eksplitsiitselt esitatud kas semantilises võrgus või freimidena, siis saab teha järelduse (tuletada), et koerad on elusolendid.

Tuletus semantilises võrgus

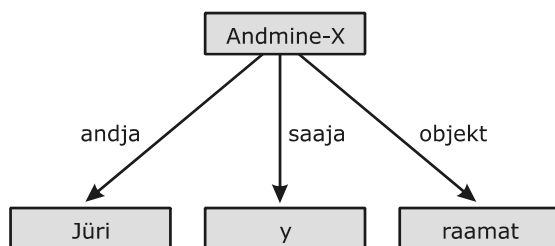
Küsimusele vastamiseks tuleb küsimus esitada kui semantiline võrk ning kõrvutada seda teadmusbaasile vastava semantilise võrgu teatava osaga.

Näide 25.1. Olgu teada fakt: *Jüri andis Marile raamatu* (joonis 25.1).

Küsimusele *Kellele Jüri andis raamatu?* vastamiseks moodustame semantilise võrgu, kus ühele tipule vastab muutuja, mille väärtust soovime teada (y , joonis 25.2), „sobitame” sellega teadaolevale faktile vastava semantilise võrgu osa (joonisel 25.1) ja loeme muutuja väärtuse (*Mari*) teadaolevast faktist. Seega on vastus küsimusele *Mari*.

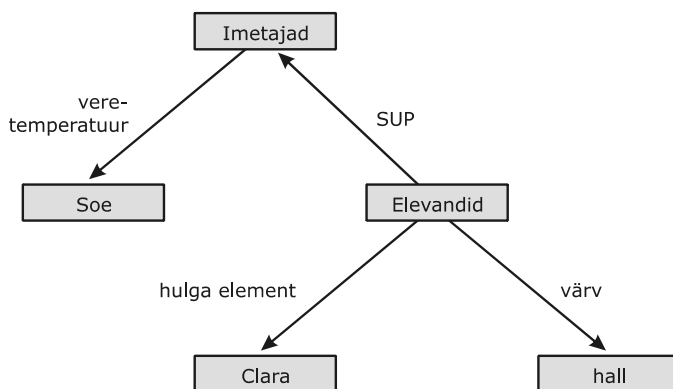


Joonis 25.1. Semantiline võrk (seosed: a – andja, s – saaja, o – objekt, e – hulga element)



Joonis 25.2. Küsimuse semantiline võrk

Näide 25.2. Vaatame semantilist võrku joonisel 25.3.



Joonis 25.3. Semantiline võrk

Küsimusele *Mis värvi on Clara?* saab vastata eelmises näites kirjeldatud viisil. Küsimusele *Missugune on Clara vere temperatuur?* vastamiseks tuleb liikuda elemendi ja ülemmõiste seoseid pidi hierarhias kõrgemale tasemele ja pärida sealt omadus.

Probleemilahendusel sooritatakse järgmised põhitegevused:

1. probleemi konteksti eraldamine teadmusbasi esitavas semantilises võrgus: vastavate tippude ja kaarte ergastamine;
 2. konteksti kõrvutamine näidisega (antud probleemiga).
- Seejuures toimub:
- a) omaduste pärimine, kasutades seoseid element – hulk, alamhulk – ülemhulk;
 - b) antud tippudest algavate teede lõikumiste otsimine.

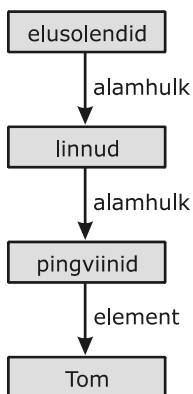
Näide 25.3. Vaatame semantilist võrku joonisel 25.4. Kuidas leida vastus küsimusele: *Kas Tom on elusolend?*

Üldiselt, küsimusele *Kas u on hulga h element?* vastamiseks võib kasutada järgmist algoritmi.

Algoritm

1. $E := \{x \mid x - \text{tipp}, x \xrightarrow{\text{element}} u - \text{kaar}\}$.
2. Kui $E = \emptyset$, siis vastus „ei”, lõpeta.
3. Kas $h \in E$? Kui jah, siis vastus „jah”, lõpeta.
4. $F := \{y \mid x, y - \text{tipud}, x \xleftarrow{\text{alamhulk}} y - \text{kaar}, x \in E\}$; $E := F$; mine 2.

Tom on elusolend?



Joonis 25.4. Semantiline võrk

Selle algoritmi rakendamise tulemusel saame vastuseks näites 25.3 esitatud küsimusele *jah*.

Nagu varem (p 13) mainitud, on erinevad teadmuse esituse meetodid omavahel samaväärsed selles mõttes, et ühel meetodil esitatud teadmuse saab esitada ka teisel meetodil, nt semantilise võrguna esitatud teadmuse predikaatloogika valemitega. Omavahel on seotud ka tuletusmeetodid. Järgnev näide demonstreerib, et äsja toodud algoritmi rakendamine küsimusele vastamiseks semantilises võrgus on samaväärne teoreemi automaatse tõesta-

misega resolutsioonimeetodil esimest järku predikaatarvutuses, rakendades lineaarset tuletust – resolventide leidmisel võetakse igal sammul üheks disjunktiks eelmisel sammul leitud resolvent ja teiseks üks lähtehulga disjunkt.

Näide 25.4. Vaatame semantilist võrku joonisel 25.4 ja esitame sama teadmuse predikaatloogikat kasutades:

$\forall x \forall y \forall u (\text{Alamhulk}(x,y) \ \& \ \text{Element}(u,x) \rightarrow \text{Element}(u,y))$
 $\forall x \forall y \forall z (\text{Alamhulk}(x,y) \ \& \ \text{Alamhulk}(y,z) \rightarrow \text{Alamhulk}(x,z))$
 $\text{Alamhulk}(\text{lind}, \text{elusolend})$
 $\text{Alamhulk}(\text{pingviin}, \text{lind})$
 $\text{Element}(\text{Tom}, \text{pingviin}).$

Leiame vastuse küsimusele *Kas Tom on elusolend?*

See on jah-ei küsimus ning jaatava vastuse saame anda, kui õnnestub tõestada, et eeldustest järeldub loogiliselt väide $\text{Element}(\text{Tom}, \text{elusolend})$.

Resolutsioonimeetodi rakendamiseks vaatleme eeldusi koos väite eitusega ning leiame vastava disjunktide hulga:

$S = \{(1) \neg \text{Alamhulk}(x,y) \vee \neg \text{Element}(u,x) \vee \text{Element}(u,y), (2) \neg \text{Alamhulk}(x,y) \vee \neg \text{Alamhulk}(y,z) \vee \text{Alamhulk}(x,z), (3) \text{Alamhulk}(\text{lind}, \text{elusolend}), (4) \text{Alamhulk}(\text{pingviin}, \text{lind}), (5) \text{Element}(\text{Tom}, \text{pingviin}), (6) \neg \text{Element}(\text{Tom}, \text{elusolend})\}.$

Disjunktide (1) ja (6) resolvent (rakendades lihtsaimat unifikatorit $\{\text{Tom}/u, \text{elusolend}/y\}$) on disjunkt

(7) $\neg \text{Alamhulk}(x, \text{elusolend}) \vee \neg \text{Element}(\text{Tom}, x),$

Disjunktide (7) ja (2) resolvent (rakendades lihtsaimat unifikatorit $\{\text{elusolend}/z\}$) on disjunkt

(8) $\neg \text{Element}(\text{Tom}, x) \vee \neg \text{Alamhulk}(x,y) \vee \neg \text{Alamhulk}(y, \text{elusolend}),$

Disjunktide (8) ja (3) resolvent (rakendades lihtsaimat unifikatorit $\{\text{lind}/y\}$) on disjunkt

(9) $\neg \text{Element}(\text{Tom}, x) \vee \neg \text{Alamhulk}(x, \text{lind}),$

Disjunktide (9) ja (4) resolvent (rakendades lihtsaimat unifikatorit $\{\text{pingviin}/x\}$) on disjunkt

(10) $\neg \text{Element}(\text{Tom}, \text{pingviin})$

ning selle resolvent disjunktiga (5) on tühidisjunkt.

Seega on vastus küsimusele jaatav: Tom on elusolend.

Tuletus freimide süsteemis

Küsimusele vastamiseks tuleb küsimus esitada kui freim ning kõrvutada seda teadmusbaasis olevate freimidega.

Näide 25.5. Vaatame uuesti näidet 25.1, kuid eeldame nüüd, et teadmus on esitatud freimidena. Seega peavad teadmusbaasis olema sellised freimid nagu ANDMINE, mis kirjeldab kõikvõimalike andmiste üldisi atribuute, ja ANDMINE-1, mis esitab teadmuse Jüri andis Marile raamatu (joonis 25.5).

| |
|----------------|
| ANDMINE |
| ... |

| |
|--|
| ANDMINE-1 |
| SUP: andmine andja: Jüri saaja: Mari objekt: raamat |

Joonis 25.5. Freimid

Küsimusele *Kellele Jüri andis raamatu?* vastamiseks moodustatakse freimi *ANDMINE* uus eksemplar, kus ühe sloti täitjaks on muutuja, mille väärtust küsitakse (joonis 25.6) ja kõrvutatakse seda teadmusbasis oleva(te) freimi(de)ga. Freimist *ANDMINE-1* tuletatakse vastus: *Mari*.

| |
|---|
| ANDMINE-X |
| SUP: andmine andja: Jüri saaja: y objekt: raamat |

Joonis 25.6. Küsimuse freim

Näide 25.6. Vaatame näidet 25.2, eeldades, et teadmus on esitatud freimidena. Teadmusbasis peavad olema (vähemalt) joonisel 25.7 kujutatud freimid.

| |
|----------------------|
| Imetaja |
| veretemperatuur: soe |

| |
|------------------------------|
| Elevant |
| SUP: imetaja värvus: hall |

| |
|--------------|
| Clara |
| SUP: elevant |

Joonis 25.7. Freimid teadmusbasis

Küsimusele *Mis värvi on Clara?* vastamiseks tuleb moodustada küsimuse freim, kus ühe sloti täitja on muutuja, ja kõrvutada seda teadmusbasis olevate freimidega, saamaks konkreetset sloti täitjat (hall). Küsimusele *Missugune on Clara vere temperatuur?*

vastamiseks tuleb liikuda ülemmõiste seoseid pidi freimide hierarhias kõrgemale tasemele ja pärida sealt omadus.

Üldiselt, probleemilahendusel sooritatakse järgmised põhitegevused:

1. vajalike freimide aktiveerimine;
2. nendest eksemplaride moodustamine (slottide täitmine konkreetse infoga).

Seejuures toimub:

- a) info võtmine probleemi kirjeldusest;
- b) info pärimine hierarhiliselt (SUP-freimidest).

Lisaks võib toimuda

- c) info võtmine teistest (mitte samas hierarhias olevatest) freimidest: selleks käivitatakse vastavad protseduurid;
 - d) info küsimine kasutajalt;
3. näidiseiga (probleemiga) kõrvutamine.

26. Kitsenduste rahuldamine

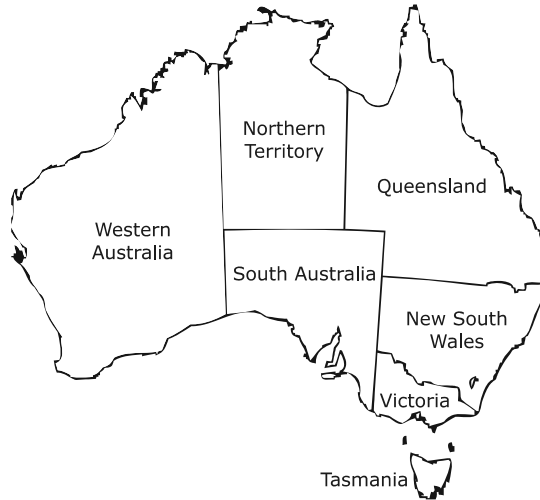
Varem (p 15–18) käsitlesime probleemilahendust kui otsingut olekute ruumis. Olekutele saab omistada valdkonnaspetsiifilised hinnangud, mis võimaldavad otsustada nende perspektiivikuse üle (ja sooritada heuristilist otsingut). Hoolimata sellest on otsingualgoritmi jaoks iga olek kui „must kast”, ilma sisemise struktuurita. Olekud esitatakse küll mingi andmestruktuuri abil, kuid olekut käsitlevad ainult järglase leidmise funktsioon, heuristiline funktsioon ja lõppoleku (eesmärgi) testimise funktsioon, mis kõik on probleemispetsiifilised. Siin vaatame **kitsenduste rahuldamise probleeme** (KRP, *constraint satisfaction problem, CSP*) – *otsinguprobleeme*, kus olekud on standardse sisemise struktuuriga ning eesmärgi testimise funktsioon kasutab seda struktuuri. Seega saab KRP lahendamiseks kasutada üldisi algoritme (probleemispetsiifiliste heuristikute asemel).

KRP puhul

- olek defineeritakse muutujatega X_i , mis omandavad väärtusi teatavast piirkonnast D_i ,
- lõppoleku (eesmärgi) test on kitsenduste hulk, mis spetsifitseerivad muutujate alamhulkade väärtuste lubatavaid kombinatsioone.

KRP lahendamiseks kasutatakse üldisi algoritme, mis on paremad kui tavalised otsingualgoritmid.

Näide 26.1. Vaatleme Austraalia kaardi värvimist kolme värviga nii, et naaberregioonid oleksid erinevat värvi (joonis 26.1, vt Russell, Norvig 2003).



Joonis 26.1. Austraalia kaart

Võtame kasutusele järgmised tähistused: muutujad WA, NT, Q, NSW, V, SA, T – vastavad regioonidele kaardil.

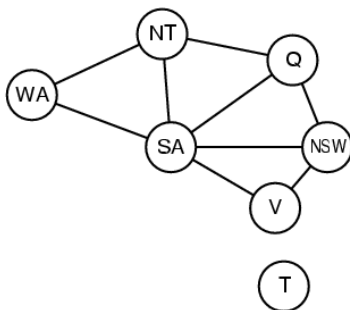
Muutujate väärtuste piirkonnad olgu $D_i = \{\text{punane, roheline, sinine}\}$ ($i=1, \dots, 7$).

Kitsendused väljendavad asjaolu, et kõik naaberregioonid peavad olema erinevat värvi, nt $WA \neq NT$ ehk $(WA, NT) \neq \{(\text{punane, roheline}), (\text{punane, sinine}), (\text{roheline, punane}), (\text{roheline, sinine}), (\text{sinine, punane}), (\text{sinine, roheline}), (\text{punane, punane}), (\text{sinine, sinine}), \text{roheline, roheline}\}$.

KRP lahenditeks on kõik sellised muutujate väärtustused, mis rahuldavad kõiki kitsendusi, nt

$\{WA=\text{punane, NT}=\text{roheline, Q}=\text{punane, NSW}=\text{roheline, V}=\text{punane, SA}=\text{sinine, T}=\text{roheline}\}$.

KRP saab visualiseerida nn *kitsenduste graafi* abil, mille tippudeks on muutujad ning kaarteks kitsendused (vt joonis 26.2).



Joonis 26.2. Kitsenduste graaf

KRPde liigid

Kitsenduste rahuldamise probleeme saab liigitada vastavalt sellele, kas muutujad on diskreetsed või pidevad ning kas väärtuste piirkonnad on lõplikud või lõpmatud.

- Muutujad on diskreetsed

a) Piirkonnad on lõplikud

Nt kaardi värvimine, 8 lipu probleem (mida vaatasime seoses geneetilise algoritmiga p 18), Boole'i KRPd (kus piirkonnad sisaldavad kaks väärtust: tõene ja väär), sh Boole'i kehtestatavuse probleem (mis on NP-täielik).

b) Piirkonnad on lõpmatud (nt täisarvude hulk, sõnade hulk).

Nt tööde ajaline järjestamine, kus muutujateks on iga töö algus- ja lõpupäev.

Lineaarsete kitsendustega probleemid (kus muutujatevahelised seosed on lineaarsed) on lahenduvad, mittelineaarsete kitsendustega probleemid aga mittelahenduvad.

- Muutujad on pidevad

Nt lineaarse planeerimise probleemid, kus kitsendused on esitatud kui lineaarvõrratused, on lahenduvad polünomiaalse ajaga (muutujate arvu suhtes).

Kitsenduste liigid

1. Unaarsed kitsendused (ühe muutujaga): nt $SA \neq \text{roheline}$.
2. Binaarsed kitsendused (kahe muutujaga): nt $SA \neq WA$.
3. Kõrgemat järku kitsendused (esinevad nt krüptoaritmeetika probleemides).

Lisaks absoluutsetele kitsendustele kasutatakse sageli nn *eelistusi* e pehmeid kitsendusi (soft constraints), nt „punane on parem kui roheline”. Sellised KRP-d on reaalses maailmas väga sagedased ja nende lahendamiseks kasutatakse optimeerimismeetodeid.

KRP lahendamine: otsing

Nagu juba mainitud, võib KRP-d käsitleda kui otsinguprobleemi kus:

olek – siiani määratud väärtused.

Algolek – tühiväärtustus { }.

Järglase leidmise funktsioon: määrata mingile seni väärtustamata muutujale väärtus, mis ei ole vastuolus jooksva väärtustusega (kui selliseid väärtusi ei leidu, siis probleemil pole lahendit).

Eesmärgi test: kas jooksev väärtustus on täielik (s.t kas kõik muutujad on väärtustatud).

Seega saab suvalist KRP-d lahendada otsingu abil. Otsing on ühesugune kõigi KRP-de jaoks.

Olgu muutujate arv n ja väärtuste arv d .

Oletame, et läbime KRP-d esitava olekute puu tase-haaval, kasutades laiutiotsingut. Iga lahend asub puus sügavusel n . Igal sügavusel k ($=0, \dots, n-1$) on puu hargnemistegur $b=(n-k)d$. Seega on puu lehtede arv $n! d^n$ (aga võimalikke täielikke väärtustusi on ainult d^n). Millest tekib see näiline vastuolu? Siin pole me arvestanud, et kõik KRP-d on kommutatiivsed, seega tuleb igas otsingupuus tipus käsitleda *ainult üheainsa muutuja võimalikke väärtustusi*. Nt kui värvime Austraalia kaarti, siis juurtipus võime valida kas $SA=\text{punane}$, $SA=\text{roheline}$ või $SA=\text{sinine}$, kuid kunagi pole meil vaja valida $SA=\text{punane}$ ja $WA=\text{sinine}$ vahel. Võttes arvesse seda kitsendust, tulebki puu lehtede arv d^n .

Meetodid:

- tagasipöördumisega otsing (*backtracking search*)
- edasivaatav kontroll (*forward checking*)
- kitsenduste levitamine (*constraint propagation*)

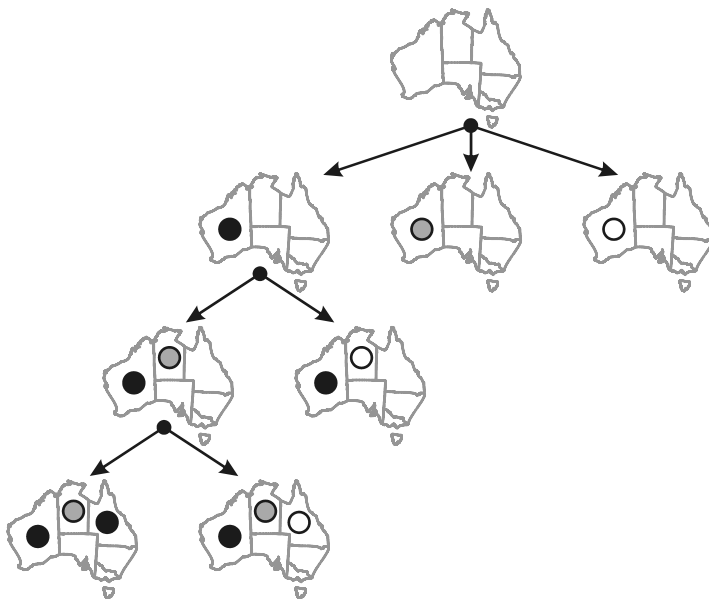
Tagasipöördumisega otsing

Tagasipöördumisega otsing, kus väärtustatakse korraga üksainus muutuja. Tagasipöördumisega otsing on KRPde põhiline pimeotsingumeetod.

Muutujate väärtustused on kommutatiivsed, s.t nt [kui WA = punane, siis NT= roheline] on sama mis [kui NT = roheline, siis WA=punane]

Igas tipus on vaja vaadata ühe muutuja väärtustusi, järelikult on olekute puu hargnemistegur $b=d$ ja lehtede arv d^n .

Joonis 26.3 esitab Austraalia kaardi värvimist tagasipöördumisega otsingu teel (joonisel on näidatud osa otsingupuust).



Joonis 26.3. Tagasipöördumisega otsing

Tõstmaks tagasipöördumisega otsingu efektiivsust, tuleb otsustada:

1. millist muutujat väärtustada järgmisena,
2. millises järjekorras neid väärtusi vaadelda,
3. kas saab varakult avastada ebaõnnestumisi,
4. kas saab arvestada probleemi struktuuri.

Heuristikuid:

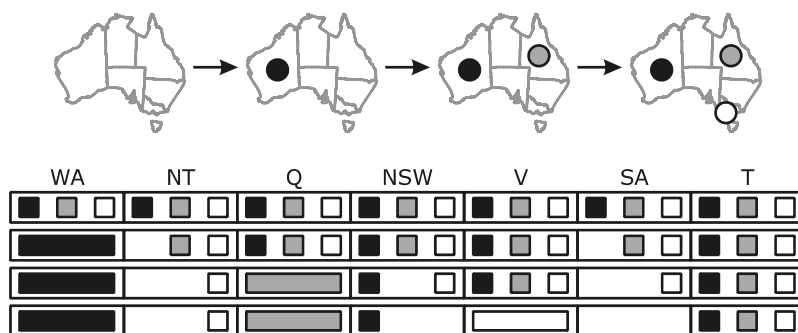
- valida väärtustamiseks **kõige rohkem kitsendatud** muutuja – millel on vähim arv lubatavaid väärtusi,
- valida väärtustamiseks **kõige rohkem kitsendav** muutuja – mis seab kõige rohkem kitsendusi ülejäänud muutujatele,
- valida muutuja **kõige rohkem kitsendav** väärtus – mis lubab kõige vähem väärtusi ülejäänud muutujatele.

Selle asemel et vaadata muutujale seatud kitsendusi alles siis, kui see muutuja on (teatavas tipus) valitud, võib vaadelda kitsendusi juba varem. See võimaldab otsingupuu suurust oluliselt vähendada. Käsitleme siin kahte meetodit: edasivaatavat kontrollimist ja kitsenduste levitamist.

Edasivaatav kontrollimine

Idee: säilitada veel väärtustamata muutujate järelejäänud lubatavad väärtused. Lõpetada otsing, kui ühelgi muutujal pole lubatavaid väärtusi.

Kui muutuja X on väärtustatud, siis vaadatakse kõiki veel väärtustamata muutujaid Y, mida seob muutujaga X mingi kitsendus, ja kustutatakse muutuja Y piirkonnast kõik sellised väärtused, mis ei ole kooskõlas X väärtusega. Joonisel 26.4 on toodud näide kaardi värvimisest. Siin on kaks tähtsat asjaolu. Esiteks, pärast seda, kui on väärtustatud WA=punane ja Q=roheline, saab muutujate NT ja SA piirkonnad taandada: mõlemad saavad olla ainult sinised. Kui kasutame otsingul heuristikut „vali kõige kitsendavam muutuja”, siis tulebki järgmisena väärtustada muutujad SA ja NT. Teiseks, kui on väärtustatud V=sinine, siis SA piirkond on tühi. Seega on edasivaatav kontroll avastanud, et osaline väärtustus {WA=punane, Q=roheline, V=sinine} ei ole kooskõlas kitsendustega ja seega algoritm pöördub kohe tagasi.



Joonis 26.4. Edasivaatav kontrollimine

Kitsenduste levitamine

Edasivaatav kontrollimine annab küll edasi infot väärtustatud muutujatelt väärtustamata muutujatele, kuid ei kontrolli küllalt varakult kõiki võimalikke ebaõnnestumisi.

Näiteks vaatame kolmandat rida joonisel 26.4: kui WA=punane ja Q=roheline, siis peaks olema NT=sinine ja SA=sinine. Kuna need on aga naaberregioonid, siis nad ei saa olla

sama värvi. Edasivaatav kontrollimine ei avasta seda ebakõla, sest ta ei vaata piisavalt kaugele.

Kitsenduste levitamine annab edasi kitsendusi ühelt muutujlt teistele muutujatele. Antud näites peame me levitama kitsendusi muutujatelt WA ja Q muutujatele NT ja SA (nagu seda teeb ka edasivaatav kontroll) ning siis muutujate NT ja SA vahelisele kitsendusele, et avastada ebakõla.

Lihtsaim kitsenduste levitamise viis on teha kõik (suunatud) kaared kitsenduste graafis kooskõlaliseks.

Kaart $X \rightarrow Y$ nimetatakse kooskõlaliseks (consistent) parajasti siis, kui muutuja X iga väärtuse x korral leidub muutuja Y mingi lubatav väärtus y.

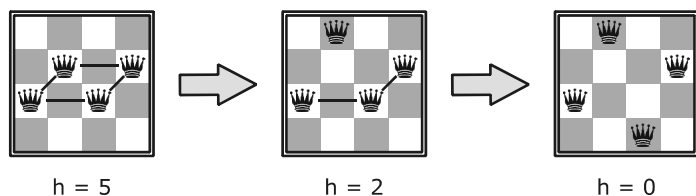
Näiteks joonisel 26.4 kolmandas reas on SA ja NSW lubatavad väärtused vastavalt {sinine} ja {punane, sinine}. Kui SA=sinine, siis NSW=punane on kooskõlaline väärtustus. Järelikult kaar tipust SA tippu NSW on kooskõlaline. Aga vastassuunaline kaar tipust NSW tippu SA ei ole kooskõlaline: kui võtta NSW=sinine, siis SA jaoks ei ole kooskõlalist väärtust. Kaare saab teha kooskõlaliseks, kui kustutada NSW piirkonnast väärtus sinine.

Samuti saame kaare kooskõlaliseuse kontrolli rakendada kaarele tipust SA tippu NT. Joonise 26.4 kolmas rida näitab, et mõlema muutuja piirkonnaks on {sinine}, järelikult tuleks see väärtus muutuja SA piirkonnast kustutada, misjärel jääks piirkond tühjaks. Seega võimaldab kaare kooskõlaliseuse kontroll varakult avastada mittekooskõlalise, mida edasivaatav kontrollimine ei avastanud.

Kaare kooskõlalisust võib kontrollida kas otsinguprotsessi algul või kestel, iga väärtustamise järel.

KRPde lahendamiseks võib kasutada ka mäkketõusu või libalõõmutamist (p 18). Mõlemad otsingumeetodid opereerivad muutujate täielike väärtustustega: algolekus määratakse väärtus kõikidele muutujatele ning järglase leidmise funktsioon muudab korruga üheainsa muutuja väärtust. Näiteks n lipu probleemi puhul (mis on samuti KRP) võib algolekuks olla lippude suvaline paigutus mängualal (igas veerus üks lipp) ning järglase leidmise funktsioon võtab ühe lipu ja paigutab selle ümber samas veerus. Muutujale uue väärtuse valimiseks on kõige tavalisem (ja praktikas efektiivne) heuristik järgmine: valida väärtus, mis põhjustab kõige vähem konflikte teiste muutujatega.

Joonisel 26.5 on näidatud nelja lipu probleemi lahendamine mäkketõusu algoritmi abil. Võimalikke olekuid on 256 (4 lippu 4 veerus), tegevuseks on lipu ümberpaigutamine samas veerus, heuristilise funktsiooni h väärtuseks on rünnete arv, lõppolekus pole ühtegi rünnet (h=0).



Joonis 26.5. Nelja lipu probleemi lahendamine

KRP-de näiteid tegelikust elust

- Väärtustamise probleemid, nt kooli tunniplaani koostamine (milline õpetaja millises klassis).
- Ajakava probleemid, nt missuguseid kursusi saab õppida, kus, millal.
- Riistvara konfigureerimine.
- Töölehtede koostamine.
- Vedude järjestamine.

IV. PLANEERIMINE

27. Probleemilahendus ja planeerimine

Probleemi keerukust võib iseloomustada järgmiste parameetritega:

1. lahendustasandi (olekute graafi) suurus,
2. ainevalla muutuvus (staatiline või dünaamiline),
3. ainevalla mudeli vastavus tegelikkusele,
4. andmete määratus (täpsus ja täielikkus),
5. vajalike lahendite arv (üks, mõned või kõik),
6. kitsendused lahendile (lahendusae, mälu maht jms).

Probleemid võivad olla mitmesuguse keerukusega, alustades lihtsatest (väike graaf, staatilised andmed, täielik mudel, täpsed andmed, vaja leida mingi, ükskõik missugune lahend, kitsenduste puudumine) kuni rasketeni (suur graaf, dünaamilised andmed, mudeli mittetäielikkus, vigased ja mittetäielikud andmed, vaja leida kõik lahendid, suvalised kitsendused).

Ekspertsüsteemides kasutatavad probleemilahendusmeetodid võib klassifitseerida järgmiselt.

1. Otsing toimub ühel tasandil: meetodid, mis on mõeldud selliste probleemide lahendamiseks, mida iseloomustavad väike piirkond, staatiline ainevald, täielik mudel, täpsed ja täielikud andmed. Eelmises peatükis vaadeldud otsingumeetodid on rakendatavad eeskätt ühel tasandil otsinguks.
2. Otsing tasandite hierarhias: meetodid, mis sobivad suurtes staatilistes ainevaldades.
3. Otsing mittetäpsete ja mittetäielike andmete korral.
4. Otsing dünaamilise ainevalla korral: meetodid, mis sobivad ajas ja ruumis muutuvate ainevaldade korral.
5. Otsing mitut mudelit kasutades: meetodid tööks ainevallas, mille adekvaatseks kirjeldamiseks ei piisa ühest mudelist.

27.1. Otsing tasandite hierarhias

Meetodid, kus lahendit otsitakse ühel tasandil, ei võimalda lahendada eriti keerulisi probleeme, sest lahendustasandi mõõtmete suurenemisel kasvab otsinguks kuluv aeg eksponentsiaalselt. Kui tegu on suure tasandiga, siis võib proovida jagada seda tasandit alamtasanditeks ja lahendust otsida nendel. Sellisel juhul on otsimistasand esitatud tasandite hierarhiaga.

Hierarhiliste meetodite tähtsust mõisteti juba ammu. 1963. aastal märkis M. Minsky, et „planeerimissaarekete” sisseviimine vähendab otsimisaega eksponentsiaalselt. Graafis, mille igast tipust väljub 10 kaart, võib 20-sammuline otsing nõuda 10^{20} katset, mida on väga raske realiseerida, samal ajal kui nelja tasandi sissetoomine võib vähendada otsimiste

arvu kuni $5 \cdot 10^4$ katseni. Seetõttu on mõtet teha jõupingutusi, et keeruliste probleemide lahendamisel eraldada sellised saarekesed.

Minsky ideed tasandite hierarhiast võib edasi arendada, kui lubada hierarhias mitte ainult konkreetseid, vaid ka abstraktseid tasandeid.

Klassikaliseks näiteks abstraktsete tasandite kasutamise kohta võib tuua lühima tee leidmise kaardil. Olgu meil vaja sõita linnast A (linna keskusest) linna B (linna keskusse). Kui sooritada otsing detailsel kaardil, mis sisaldab linnade kõiki tänavaid ja väljakuid, siis võib ülesanne osutuda raskelt lahenduvaks. Seepärast on otstarbekas määrata tee linnast A linna B esmalt suuremastaabilisel kaardil (s.t teostada otsing abstraktsemal tasemel) ja seejärel planeerida detailsel kaardil tee linna A keskusest linnaservale ja sissesõit linna B.

Järgnevas osas peatume meetoditel, mis kasutavad tasandite hierarhiat ja erinevad üksteisest tasandite moodustamise poolest. Märgime veel, et hierarhilised tasandid on leidnud praktilist kasutamist ekspertsüsteemides.

1. Otsing faktoriseeritud tasandil

Paljudel juhtudel nõutakse kõigi lahendite leidmist. Sellisteks näideteks on andmete interpreteerimine, haiguse diagnoosimine jne. Tõepoolest, diagnoosi panekul huvitavad meid patsiendi kõik haigused, mitte ainult mõned. Praktilistes rakendustes on otsimistasand nii suur, et ei saa kasutada pimeotsingu meetodeid. Heuristiliste meetodite kasutamine on reeglina samuti välistatud, kuna need ei taga kõikvõimalike lahendite saamist. Kui aga otsimistasand õnnestub faktoriseerida, siis sellisel juhul on otsingut isegi väga suure tasandi korral võimalik efektiivselt organiseerida.

Ütleme, et tasand on **faktoriseeritud**, kui seda on võimalik jagada lõikumatuks alamtasanditeks (klassideks), millest igaühel on oma osalised (mittetäielikud) lahendid. Sealjuures on mittetäieliku lahendi põhjal võimalik öelda, et see ei vii edule, s.t täielikud lahendid, mis temast on võimalik tuletada, ei vii eesmärgile.

Otsimine faktoriseeritud tasandil toimub hierarhilise *genereeri ja testi* meetodil. Generaator valib jooksva osalise lahendi ja seejärel kontrollitakse, kas see lahendus võib viia eesmärgile. Kui jooksev osaline lahend heidetakse kõrvale, siis sellega koos heidetakse kõrvale ka (ilma genereerimiseta) kõik antud klassi täielikud lahendid. Kui aga vahelahendit kõrvale ei heideta, siis generaator genereerib kõik temal põhinevad täielikud lahendid ja testija ülesanne on kontrollida nende täisesmärkide kehtivust.

2. Otsing fikseeritud tasandite hulgal

Tasandi faktoriseerimise meetodi kasutamine on piiratud seetõttu, et terves reas ainevaldades ei õnnestu pooliku (osalise) lahendi põhjal otsustada, kas see on sobiv või mitte. Selliste ainevaldade näideteks on planeerimine ja konstrueerimine. Tõepoolest, harilikult ei saa plaani või konstruktsiooni fragmendi põhjal väita, et see fragment ei saa olla lõpliku plaani osaks.

Järgnevalt võtame vaatluse alla otsingumeetodid, mis kasutavad abstraktse tasandi ideed. Meetodid erinevad selle poolest, kuidas üks või teine tuletab tasandi. Abstraktsioon peab:

- rõhutama vaadeldava ülesande olulisi omadusi, eripärasid,
- lubama ülesannet jagada lihtsamateks alamülesanneteks,

- lubama määrata alamülesannete sellise järjekorra (ülesannete lahendamise plaani), mis viib lähteülesande lahendamisele.

Lihtsaimal juhul jagatakse otsingutasand alamtasanditeks (fikseeritud alamülesannete jadaks), mille abil on võimalik lahendada suvalist antud klassi kuuluvat ülesannet.

Näiteks ekspertsüsteem R1 komplekteerib arvuti vastavalt ostja soovidele. Süsteem jagab esialgse ülesande kuueks alamülesandeks. Järjekord, milles neid alamülesandeid lahendatakse, sõltub tellitud konfiguratsioonist. Tegevused, mis sooritatakse iga alamülesande lahendamiseks, sõltuvad tellitud komponentidest ja nende omavahelistest seostest. Süsteemis vastab igale alamülesandele oma reeglite komplekt, s.t alamülesanne lahendatakse oma alamtasandil.

27.2. Metatasand tasandite hierarhias

Suvalise probleemi lahendamisel kerkib korduvalt küsimus: *mida teha järgmisel sammul?*

Lihtsaimal juhul määrab selle probleemilahendusmeetod. Kui otsitakse abstraktsel või konkreetse tasandil, siis lahendatakse igal sammul küsimus, millist tegevust antud olekus kasutada.

Mitmed süsteemid suudavad arutleda selle üle, kuidas ühte või teist lahendit saada, s.t planeerida omaenda arutlusi. Selliselt püstitatud probleem tuleb lahendada nn metatasandil. Metatasandil esitatakse lahendaja kui otsinguprotseduur, s.t antakse ette

- olekukirjeldused,
- operaatorite kirjeldused,
- operaatorite rakendamise tingimused,
- lubatud otsingumeetodite kirjeldused ja nende omavahelised seosed.

Selleks et saada lahendit metatasandil, tuleb määrata, milline meetod (programm) on rakendatav järgmisel sammul. Seega tuleb koostada ülesande lahendamise metaplaan.

Märgime, et erinevalt abstraktsest plaanist ei ole metaplaan esitatud antud ainevalla operaatorite terminites, vaid kasutades neid termineid, meetodeid ja programme, mis on tuntud lahendajale.

Samamoodi kui abstraktse tasandi faktoriseerimisel, võib ka metatasandi jagada metaülesanneteks (s.o meetodid, programmid). Metaülesanneteks jagamine on kasulik teadmuse organiseerimise meetod, kuid seni suures osas veel teoreetilisel tasemel.

28. Probleemilahenduse planeerimine: süsteem STRIPS

Järgnevalt vaatame, kuidas planeerida tegevuste jada, et jõuda eesmärgini. Tutvume planeerimissüsteemiga STRIPS (STanford Research Institute Planning SYstem).

STRIPS oli esimesi roboti probleemilahendussüsteeme. Tema ainevallaks on nn kuubikute e plokkide maailm, kus olekud on esitatud kehtivate väidete loeteluga ja kus robot saab oma käe abil sooritada selliseid tegevusi nagu kuubikute liigutamine, üksteisele asetamine jms.

Olekukirjelduste esitamiseks kasutatakse järgmisi predikaate:

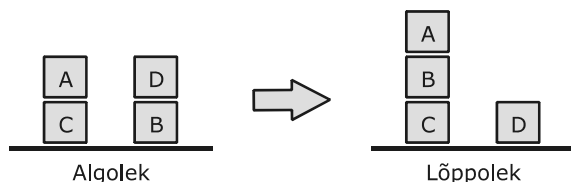
$CLEAR(X)$ –plokk X on vaba,

$ON(X, Y)$ – plokk X asub ploki Y peal,

$ON(X, Table)$ –plokk X asub laual.

Probleemi lahendamiseks tuleb leida selliste tegevuste jada, mis viib algolekut kirjeldavatest väidetest lõppolekut kirjeldavate väideten.

Joonisel 28.1 on kujutatud ühe probleemi alg- ja lõppolek.



Joonis 28.1. Probleem kuubikute maailmas: alg- ja lõppolek

Algoleku kirjelduse saame esitada järgmiste valemite (nn aksioomide) abil:

$ON(A, C)$, $ON(C, Table)$, $ON(D, B)$, $ON(B, Table)$.

Lisaks on vaja näidata, et kahele plokile ei toetu midagi – see on vajalik, et probleemi lahendamise käigus teada, kas plokile võib teist ploki peale asetada või mitte. Seega lisame veel järgmised aksioomid:

$CLEAR(A)$, $CLEAR(D)$.

Lõppoleku kirjeldus:

$ON(A, B)$, $ON(B, C)$, $CLEAR(A)$, $CLEAR(D)$, $ON(C, Table)$, $ON(D, Table)$.

Veel peame silmas pidama, et ploki saab liigutada alles siis, kui tema peal ei ole enam midagi, s.t kui plokk on ülemine.

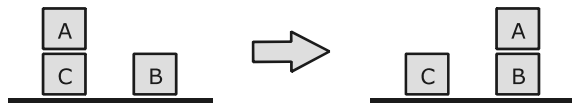
Üks võimalus probleemi lahendamiseks on fikseerida roboti iga tegevuse lõppedes, millised siiani kehtivad aksioomid enam ei kehti ja millised uued aksioomid tuleb kirjeldusele lisada pärast selle tegevuse sooritamist. Olekukirjeldustes muudatuste tegemiseks kasutatakse kahte liiki järeldusi, mis vastavad kuubikute teatavale konkreetsele füüsilisele liikumisele:

- eemaldamist DELETE,
- lisamist ADD.

Näiteks

DELETE $ON(A, C)$, ADD $ON(A, B)$

tähendab plokki A liigutamist plokilt C plokile B (joonis 28.2).



Joonis 28.2. Ploki A liigutamine plokilt C plokile B

Selliseid reegleid nimetame if-add-delete reegliteks ja nende struktuur on järgmine:

if :
add :
delete :

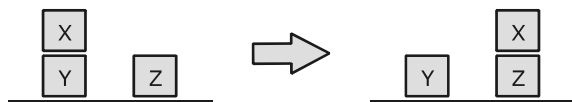
Planeerimise kontekstis nimetatakse selliseid reegleid **operaatoriteks** ja reeglis sisalduvat if-tingimust nimetatakse **eelduseks**. Süsteemi STRIPS operaatorid vastavad roboti võimalikele tegevustele.

Operaator 1: liigutada plokk X plokilt Y plokile Z (joonis 28.3).

if : ON(X, Y)
CLEAR(X)
CLEAR(Z)

add : ON(X, Z)
CLEAR(Y)

delete : ON(X, Y)
CLEAR (Z)



Joonis 28.3. *Operaator 1*: liigutada plokk X plokilt Y plokile Z

Meie lähteülesande (28.1) korral näeks *Operaatori 1* rakendamine välja järgmine:

Operaator 1:

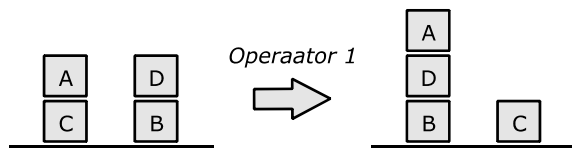
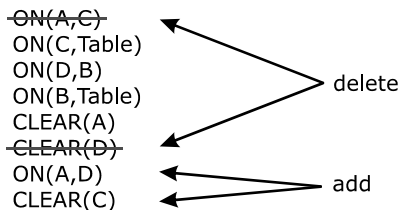
if : ON(A, C)
CLEAR(A)
CLEAR(D)

add : ON(A, D)
CLEAR(C)

delete : ON(A, C)
CLEAR(D)

Pärast *Operaatori 1* rakendamist on olekukirjeldus järgmine (vt ka joonis 28.4):

~~ON(A, C)~~ delete
 ON(C, Table)
 ON(D, B)
 ON(B, Table)
 CLEAR(A)
~~CLEAR(D)~~ delete
 ON(A, D) add
 CLEAR(C) add



Joonis 28.4. *Operaatori 1* rakendamine: liigutada plokk A plokilt C plokile D

Samuti on võimalik paigutada plokk lauale.

Operaator 2: liigutada plokk X plokilt Y lauale (joonis 28.5).

if : ON(X, Y)
 CLEAR(X)
 add: ON(X, Table)
 CLEAR(Y)
 delete: ON(X, Y)



Joonis 28.5. *Operaator 2*: liigutada plokk X plokilt Y lauale

Edasi on võimalik paigutada plokk laualt teisele plokile.

Operaator 3: liigutada plokki X laualt plokile Z (joonis 28.6).

```
if: ON(X, Table)
CLEAR(X)
CLEAR(Z)
add: ON(X, Z)
delete: ON(X, Table)
CLEAR(Z)
```



Joonis 28.6. *Operaator 3*: liigutada plokki X laualt plokile Z

Nüüd otsime (planeerime) lähteprobleemi (28.1) lahendamiseks sobiva operaatorite jada. Üks võimalus on valida järjest operaatoreid ja püüda neid rakendada senikaua, kuni saavutatakse lõppoleku kirjeldus. Kasutades seda meetodit, jõuame lahenduseni pärast nelja liigutamist. Vastav lahenduspuu on toodud joonisel 28.7. Seega tuleb esmalt liigutada plokki A plokilt C lauale, siis plokki D plokilt B lauale, seejärel plokki B laualt plokile C ja lõpuks plokki A laualt plokile B.

Nagu saadud puu põhjal võib järeldada, on põhiliseks raskuseks võitlus puu eksponentsiaalse kasvu vastu. Siin toodud näide on küll väga väike, kuid ometi on tipu maksimumne alluvate arv 12. Kui aga panna lauale nt 8 kaheplokilist torni, siis on juba esimese valikuga 64 alternatiivi. Seega osutub väga oluliseks, kuidas organiseerida otsingut lubatavate operaatorite hulgast, s.t kuidas piiritleda kasutatavate operaatorite ja olekute hulka.

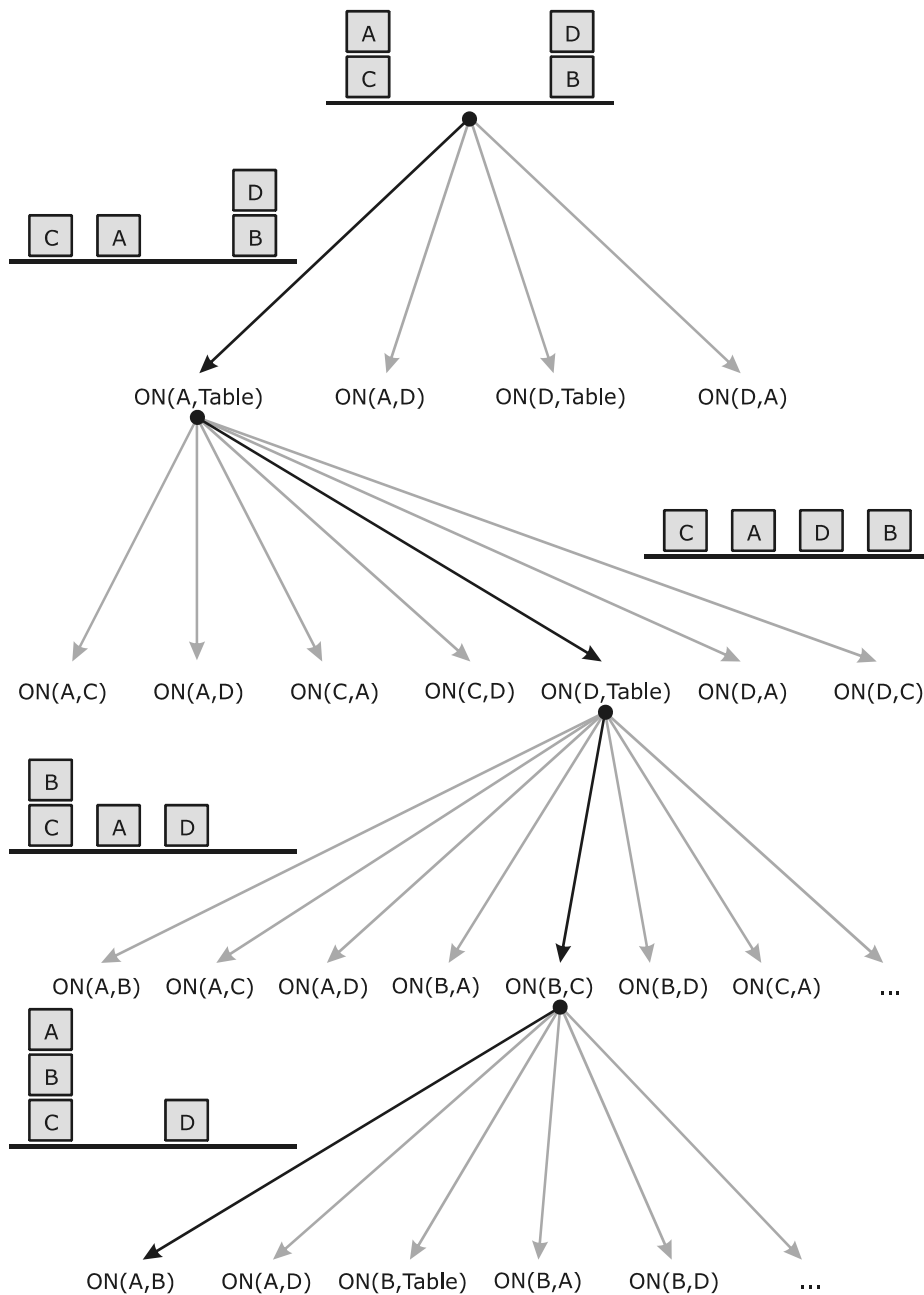
Üks võimalik lahendusviis oleks püüda liikuda hoopis tagant ettepoole, s.t

Lõppolek \rightarrow ... \rightarrow Algolek

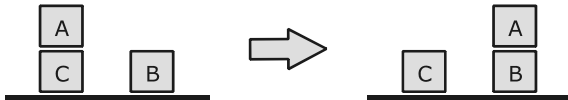
Vaatame uuesti joonisel 28.1 esitatud näidet. Soovime, et plokki A asuks plokki B peal (joonis 28.8). Jätame esialgu kõrvale nõudmise, et plokki B peaks omakorda asuma plokil C. Osaprobleemi lahendusplaan on esitatud joonisel 28.9.

STRIPS kasutab eesmärkide pinu (ingl k *stack*) ja probleemilahendusel keskenduvad jõupingutused pinu tippu. Algul sisaldab pinu üksnes peaesmärgi (mis kehtib lõppolekus). Iga kord, kui tipmine eesmärk pinus on vastavuses jooksva oleku kirjeldusega, eemaldatakse ta pinust ja avaldistele, mis asuvad pinus eesmärgist allpool, rakendatakse asendust. Vastupidisel juhul, kui pinu tipmine eesmärk on liiteesmärk (elementaarvalemite konjunktsioon), siis lisab STRIPS pinu tippu mingis järjekorras liiteesmärgi kõik komponendid. Mõte on selles, et süsteem töötab iga osaesmärgiga sellises järjekorras, milles see on paigutatud pinusse. Kui kõik osaesmärgid on lahendatud, siis vaatab süsteem uuesti üle liiteesmärgi, loendades taas tema osad, mis asuvad pinu tipus, kui liiteesmärk ei ole veel sattunud vastavusse jooksva oleku kirjeldusega. See korduv pöördumine liiteesmärgi poole on (küllalt lihtne) vahend, mille abil süsteemis nähakse ette eesmärkide vastastikuse toime käsitlemine. Kui ühe liiteesmärgi lahendamine tühistab

varem lahendatud osaesemärgi, siis see „tühistatud” eesmärk vaadatakse üle ja vajaduse korral lahendatakse uuesti.



Joonis 28.7. Näiteprobleemi (joonis 28.1) lahendus



Joonis 28.8. Osaprobleem: paigutada plokk A plokile B

| Algolek | Operaator 1 |
|--|--|
| ON(A,C) CLEAR(A) ON(D,B) CLEAR(D) ON(C,Table) ON(B,Table) | if: ON(A,C) CLEAR(A) CLEAR(B) add: ON(A,B) CLEAR(C) delete: ON(A,C) CLEAR(B) |

EESMÄRK:
ON(A,B)

Joonis 28.9. Osaprobleemi lahendusplaan: paigutada plokk A plokile B, eeldusel, et B on vaba

Joonisel 28.10 on toodud sellesama osaprobleemi lahendusplaan, nüüd aga eeldusel, et eelnevalt vabastatakse plokk B. Seega oleme plaani koostanud „tagantpoolt ettepoole”.

| Algolek | Operaator 2 | Operaator 1 |
|--|--|--|
| ON(A,C) CLEAR(A) ON(D,B) CLEAR(D) ON(C,Table) ON(B,Table) | if: ON(D,B) CLEAR(D) add: ON(D,Table) CLEAR(B) delete: ON(D,B) | if: ON(A,C) CLEAR(A) CLEAR(B) add: ON(A,B) CLEAR(C) delete: ON(A,C) CLEAR(B) |

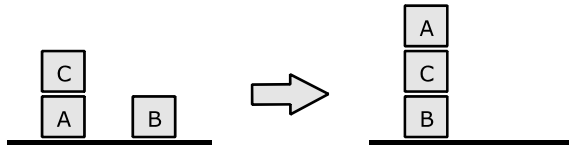
ON(A,B)

Joonis 28.10. Osaprobleemi lahendusplaan: paigutada plokk A plokile B, vabastades eelnevalt B

Kui ülemine (lahendamata) eesmärk pinus koosneb ühestainsast literaalist, siis otsib süsteem reeglit, mille lisamiste nimestikus (pärast vajalikke asendusi) sisaldub see literaal. Seejärel eemaldatakse see eesmärk pinu tipust ja paigutatakse asemele leitud reegli erijuht ning omakorda sellest kõrgemale reegli eelduseks oleva valemi erijuht. Kui tegu on liitreegliga, mis ei ole vastavuses jooksva olekuga, siis paigutatakse pinu tippu mingis järjekorras tema komponendid.

Kui pinu tipmine eesmärk on reegel, siis tähendab see, et selle reegli eelduse valem sai vastavusse jooksva oleku kirjeldusega ja eemaldati pinust. Seega on see reegel rakendatav. Teda rakendatakse jooksva oleku kirjeldusele ja eemaldatakse pinu tipust. Nüüd kasutatakse esialgse asemel uue oleku kirjeldust ja süsteem peab selle reegli mees, et see edaspidi lülitada lahendavasse järjendisse.

Vaatame probleemi, mille alg- ja lõppolek on esitatud joonisel 28.11.



Joonis 28.11. Probleem: alg- ja lõppolek

Olekute kirjeldamiseks kasutame predikaate

CLEAR(X) – plokk X on vaba,

ON(X, Y) – plokk X asub plokki Y peal,

ONTABLE(X) – plokk X asub laual.

Roboti käe oleku kirjeldamiseks kasutame predikaate

HANDEEMPTY – käsi on tühi,

HOLDING(X) – käsi hoiab plokki X.

Võimalikud tegevused esitame järgmiste reeglite abil:

pickup(X) – võtta laualt plokk X,

putdown(X) – asetada lauale plokk X,

stack(X, Y) – asetada plokk X plokki Y peale,

unstack(X, Y) – võtta ära plokk X plokki Y pealt.

Need tegevused on süsteemis STRIPS esitatavad järgmiste if-add-delete reeglitega⁴⁵.

pickup(X)

if: ONTABLE(X), CLEAR(X), HANDEEMPTY

add: HOLDING(X)

delete: ONTABLE(X), CLEAR(X), HANDEEMPTY

putdown(X)

if: HOLDING(X)

add: ONTABLE(X), CLEAR(X), HANDEEMPTY

delete: HOLDING(X)

stack(X,Y)

if: HOLDING(X), CLEAR(Y)

add: HANDEEMPTY, ON(X,Y), CLEAR(X)

delete: HOLDING(X), CLEAR(Y)

unstack(X,Y)

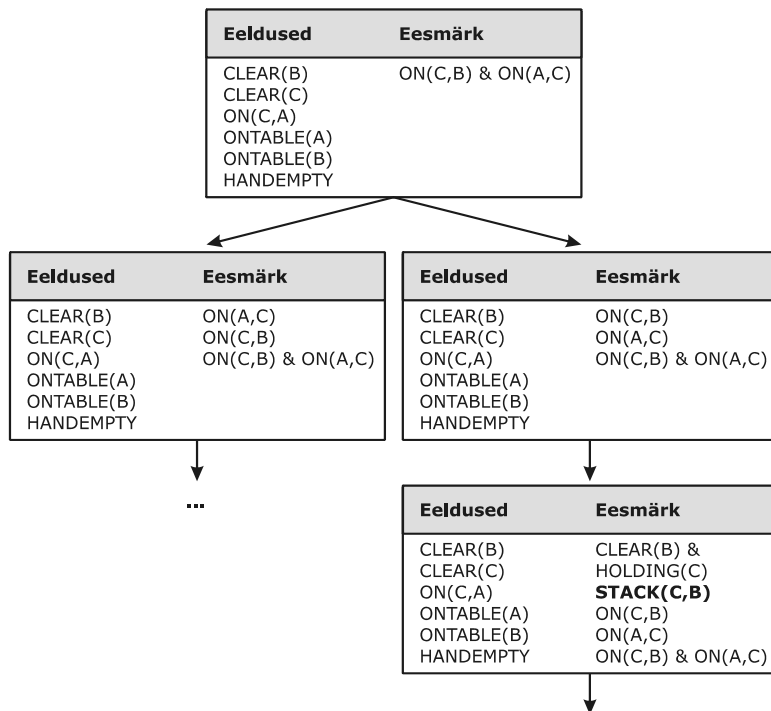
if: HANDEEMPTY, CLEAR(X), ON(X,Y)

add: HOLDING(X), CLEAR(Y)

delete: HANDEEMPTY, CLEAR(X), ON(X,Y)

Vaadeldavas näites on eesmärgiks [ON(B, C) & ON(A, B)]. Paneme tähele, et selle eesmärgi võib saavutada, liigutades plokki B plokile C ja seejärel plokki A plokile B. Joonisel 28.12 on näidatud osa lahendusgraafist (igal sammul jooksva oleku kirjeldus ja eesmärkide pinu sisu).

⁴⁵ ONTABLE(X) – plokk X asub laual.

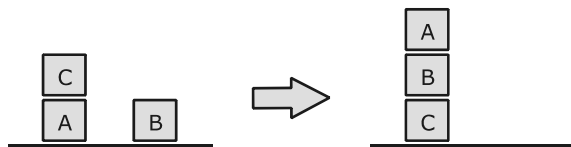


Joonis 28.12. Näiteprobleemi lahendamise algus süsteemis STRIPS

29. Plaani esitamine kolmnurktabelina

Paljudel juhtudel osutub kasulikuks plaani spetsifikatsiooni täiendava info lisamine: millised seosed on reeglite vahel, eelduste vahel jne. Sellist infot saab esitada kolmnurkse tabelina, mille elementideks on plaani kuuluvad eeldused ja reeglite rakendamisel lisatavad tulemused.

Vaatame plokkide maailmas probleemi, mille alg- ja lõppolek on esitatud joonisel 29.1.



Joonis 29.1. Probleem: alg- ja lõppolek

Joonisel 29.2 on kolmnurktabelina esitatud plaan – eesmärgi saavutamiseks vajalike tegevuste järjest, mille alusel robot lahendab joonisel 29.1 toodud probleemi.

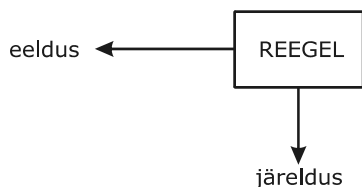
| | | | | | | | |
|---|-----------------------------------|------------|---------------------|------------------|-------------------|------------------|-------------------|
| 1 | HANDEEMPTY CLEAR(C) ON(C,A) | | UNSTACK(C,A) | | | | |
| 2 | | HOLDING(C) | PUTDOWN(C) | | | | |
| 3 | ONTABLE(B) CLEAR(B) | | HANDEEMPTY | PICKUP(B) | | | |
| 4 | | | CLEAR(C) | HOLDING(B) | STACK(B,C) | | |
| 5 | ONTABLE(A) | CLEAR(A) | | | HANDEEMPTY | PICKUP(A) | |
| 6 | | | | | CLEAR(B) | HOLDING(A) | STACK(A,B) |
| 7 | | | | | ON(B,C) | | ON(A,B) |
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |

Joonis 29.2. Roboti plaan kolmnurktabelina

Kolmnurktabeli veergude kohal asuvad reeglid: **unstack(C,A)** (võtta ära plokk C plokilt A), **putdown(C)** (asetada plokk C lauale) jne. Lahtris (i,j), kui $j > 0$ ja $i < n+1$ (siin $n=6$), paiknevad need andmed, mis lisatakse olekukirjeldusse j-nda reegli poolt (vt ka joonis 29.3).

Lahtris (i,0), kui $i < n+1$, asuvad need algoleku kirjeldused, mis on eelduseks i-ndale reeglile.

Reas $n+1$ paikneb lõppeesmärgi kirjeldus.



Joonis 29.3. Reegel, eeldus ja järelkus kolmnurktabelis

Defineerime **i-tuum**, mis moodustub nendest ridadest, mis asuvad allpool i-ndat rida (i-rida kaasa arvatud), ja nendest veergudest, mis asuvad i-ndast veerust vasakul.

0-tuum (null-tuum) on algoleku kirjeldus; (n+1)-tuum sisaldab lõputingimuse (tulemuse) kirjelduse.

Milleks on tuumad kasulikud? Kuna plaane täidab reaalses maailmas mingi mehaaniline seade, siis on võimalikud vead. Vaatame järgmist näidet. Oletame, et esitatud plaani (joonis 29.1) neli esimest operaatorit rakendati korrektselt. Kuid viienda operaatori pickup(A) rakendamisel tekib tõrge: robot ajab segamini kuubikud A ja B, s.t võtab laualt vale kuubiku (B). Edasi oletame, et muud allsüsteemid töötavad korrektselt ja seega lisatakse tingimus HOLDING(B) ning kustutatakse ON(B,C). Seega ei lisata tingimust HOLDING(A).

Seoses selle veaga ja kehtiva situatsiooniga on suurima numbriga tuum, mille kõik parameetrid kehtivad, 4-tuum. See ütleb meile, et vastav tegevus $\text{stack}(B,C)$ tuleb uuesti sooritada ja seejärel jätkata kuni eduka lõpuni.

30. Probleemilahenduse planeerimine eesmärkide koostoime korral: süsteem RSTRIPS

Süsteem RSTRIPS on süsteemi STRIPS modifikatsioon, milles eesmärkide koostoimel tekkivate raskuste ületamiseks kasutatakse **eesmärkide regressiooni** mehhanismi. See võimaldab rakendada reegli P2 asemel, mis esineb plaanis edaspidi, reeglit P1, mis satuks vastuollu P2 mingi juba saavutatud eeldusega R. Kuna P2 esineb plaanis hiljem kui P1, siis on tarvis, et reeglil P2 oleks mingi täiendav veel saavutamata eeldus R', mis viiks vajaduseni rakendada P1 esimesena. P1 rakendamise asemel toimub süsteemis RSTRIPS plaani reorganiseerimine R' regressiooni teel sellel reeglil, mis tagab eelduse R kehtivuse. Nüüd ei ole sellise R' saavutamine, mis saadakse regressiooni tulemusel, enam vastuolus eeldusega R.

Käsitleme süsteemi RSTRIPS tegevusi joonisel 28.11 toodud näite varal.

Kuni rakendatakse esimest reeglit, töötab RSTRIPS samuti kui STRIPS. Ometi esineb siin iseärasus eesmärkide pinu kasutamises. Kui toimub komponentide järjestamine pinus ülalpool mingit liiteesmärki, siis need komponendid koos nende liiteesmärgiga rühmitatakse vertikaalse sulu abil. Varsti saab selgeks sellise rühmitamise mõte.

```
[ HANDEEMPTY & CLEAR(C) & ON(C,y)
  unstack(C,y)
  [ HOLDING(C)
    CLEAR(B)
  [ HOLDING(C) & CLEAR(B)
    stack(C,B)
    [ ON(C,B)
      ON(A,C)
    [ ON(C,B) & ON(A,C)
```

Joonis 30.1. Eesmärkide pinu süsteemis RSTRIPS. Liiteesmärgid ja nende komponendid on ühendatud

Vaadeldava probleemi lahendamisel on esimese reegli $\text{unstack}(C, A)$ rakendamise momendil eesmärkide pinu selline, nagu näidatud joonisel 30.2 – niisamasugune nagu süsteemis STRIPS samal momendil (vt joonis 28.12). Selguse mõttes on käesolevas osas joonistel säilitatud tingimus, mis saavutatakse selle reegli rakendamisel, mis viib temani eesmärkide pinus. Vertikaalsed sulud ühendavad üheks rühmaks eesmärgi komponendid ja liiteesmärgid.

Asendusel $\{A/Y\}$ võib süsteem RSTRIPS rakendada reeglit $\text{unstack}(C, A)$, sest selle reegli eeldused (pinus kõrgemal) on vastavuses algolekuga. Selle asemel et eemaldada selline rahuldatus eeldus ja vastav reegel eesmärkide pinust (nagu seda teeb STRIPS), jätab RSTRIPS need elemendid pinusse alles ja paigutab vahetult tingimuse $\text{HOLDING}(C)$ alla markeri, näitamaks, et $\text{HOLDING}(C)$ saavutati selle reegli rakendamise tulemusel. Kui

süsteem kontrollib tingimusi pinus, siis korrigeeritakse markeri asendit niiviisi, et ta alati asuks järgmise tingimuse kohal pinus, mis veel on rahuldamata (joonis 30.2).

[HANDEEMPTY & CLEAR(C) & ON(C,A)

unstack(C,A)

*HOLDING(C)

CLEAR(B)

HOLDING(C) & CLEAR(B)

stack(C,B)

ON(C,B)

ON(A,C)

ON(C,B) & ON(A,C)

Joonis 30.2. Eesmärkide pinu pärast reegli unstack(C, A) rakendamist. Marker (horisontaalne joon) eraldab eesmärki, mis saavutati selle reegli rakendamise tulemusel

Horisontaalne joon pinus ongi selline marker. Kõik reeglid ülalpool markerit on rakendatud; tingimus, mis asub täpselt markeri all, s.t CLEAR (B), peab nüüd tulema kontrollimisele.

Kui marker lõikab vertikaalset joont (nagu see on joonisel 30.2), siis asuvad markeri kohal need eesmärgid, mis on juba saavutatud ja mis on markerist allpool pärast sulgu asuva liiteesmärgi komponendid. RSTRIPS märgib need komponendid ja „kaitseb” neid. Kaitse tähendab, et süsteem jälgib, et vertikaalse sulu piires ei rakendataks reeglit, mis eemaldab, s.t muudab vääradeks eesmärgi kaitstud komponendid. Kaitstud eesmärgid märgistatakse tärniga (*).

Kui võrrelda süsteemiga STRIPS, siis eesmärkide pinus asuva reegli eelduse täidetuse korral rakendas STRIPS seda reeglit jooksvale olekukirjeldusele. Seevastu süsteemis RSTRIPS pole vajadust sellist protsessi ilmutatult läbi viia. Selle asemel näitab see osa eesmärkide pinust, mis asub markeri kohal, reeglite järjendit, mis antud ajamomendiks on rakendatud. Lähtudes sellest järjendist, on süsteemis RSTRIPS alati võimalik leida selline olekukirjeldus, mis oleks selle järjendi algolekule rakendamise tulemus. Tegelikult polegi kunagi vaja arvutada seda tüüpi olekukirjeldusi. Ülim, mida on vaja teha, on otsustada, kas teatavad alameesmärgid vastavad jooksvale olekukirjeldusele või mitte. Sellist arvutamist võib teha kontrollitava alameesmärgi regressiooni teel, vastupidises suunas selleks momendiks rakendatud reeglite järjendile. Näiteks joonisel 30.2 antud pinu sisu korral peab RSTRIPS otsustama, kas tingimus CLEAR(B) vastab sellele olekukirjeldusele, mis tekib pärast reegli unstack(C, A) rakendamist. Tingimuse CLEAR(B) regressioon sellel reeglil tekitab sama tingimuse CLEAR(B), mis vastab algele olekukirjeldusele ja seepärast peab vastama ka järgnevale olekukirjeldusele.

Sellel etapil on süsteemis RSTRIPS rahuldatud reegli stack(C, B) mõlemad eeldused, seepärast rakendatakse reeglit (markeri nihutamise teel) ja kaitstakse tingimus ON(C, B). Kuna nüüd kogu sulg liiteesmärgi HOLDING(C) & CLEAR(B) jaoks asub markerist kõrgemal, siis võtab süsteem maha kaitse tingimusest HOLDING(C). Seejärel püüab RSTRIPS saavutada eesmärki ON(A, C). Lõpuks näeb pinu pärast kahe esimese reegli rakendamist välja nii, nagu kujutatud joonisel 30.3.

```

[ HANDEEMPTY & CLEAR(C) & ON(C,A)
  unstack(C,A)
  [ HOLDING(C)
    CLEAR(B)
  ] HOLDING(C) & CLEAR(B)
  stack(C,B)
  *ON(C,B)
  [ HANDEEMPTY & CLEAR(A) & ONTABLE(A)
    pickup(A)
    [ HOLDING(A)
      CLEAR(C)
    ] HOLDING(A) & CLEAR(C)
    stack(A,C)
    ON(A,C)
  ] ON(C,B) & ON(A,C)

```

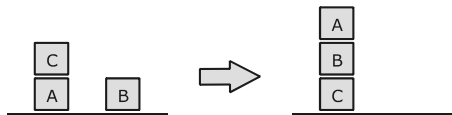
Joonis 30.3. Eesmärkide pinu pärast kahe esimese reegli rakendamist

Järgmise reegli `pickup(A)` eeldused vastavad jooksvale olekukirjeldusele. Enne reegli rakendamist peab RSTRIPS veenduma, et see ei tekita vastuolu kaitstud eesmärkidega. Sellel etapil on kaitstud `ON(C, B)`. Rikkumine tekib sellisel juhul, kui regressiooni tulemusel saadakse `F`, s.t kui reegli rakendamise tulemusel kõrvaldatakse `ON(C, B)`. Kuna kõik kaitstud eesmärgid säilivad, siis võib reeglit `pickup(A)` rakendada. Marker nihutatakse eesmärgi `HOLDING(A)` alla ja kaitstakse `HOLDING(A)`.

Regressioon reegli `stack(A, C)` teise eelduse `CLEAR(C)` reeglite järjendil annab tulemuseks, et see vastab jooksvale olekukirjeldusele. Seega on liiteeldus reegli `stack(A, C)` jaoks täidetud. RSTRIPS rakendab reeglit `stack(A, C)` ja nihutab markeri viimase tingimuseni pinus. Nüüd on töö lõppenud, sest kõik pinu elemendid asuvad markerist ülalpool. Eesmärkide pinus sellel momendil asuvad eesmärgid annavadki lahendava järjendi

$\{\text{unstack}(C, A), \text{stack}(C, B), \text{pickup}(A), \text{stack}(A, C)\}$.

Äsja käsitletud näites laabus kõik sujuvalt, sest oleku kaitstus ei olnud rikutud. Aga reeglite koostoime korral võib selline rikkumine siiski aset leida. Järgmises näites vaatame, kuidas RSTRIPS sellist olukorda käsitleb. Olgu vaja lahendada probleem, mille alg- ja lõppolek on kujutatud joonisel 29.1. Kõik laabub sujuvalt kuni selle momendini, kui RSTRIPS moodustab joonisel 30.4 kujutatud pinu.



```

[ ONTABLE(A)
[ HANDEEMPTY & CLEAR(C) & ON(C,A)
  unstack(C,A)
  CLEAR(A)
[ HOLDING(C)
  putdown(C)
  HANDEEMPTY
[ ONTABLE(A) & CLEAR(A) & HANDEEMPTY
  pickup(A)
  HOLDING(A)
  CLEAR(B)
[ HOLDING(A) & CLEAR(B)
  stack(A,B)
  *ON(A,B)
  *ONTABLE(B)
[ HANDEEMPTY & CLEAR(z) & ON(z,B)
  unstack(z,B)
  CLEAR(B)
  HANDEEMPTY
[ ONTABLE(B) & CLEAR(B) & HANDEEMPTY
  pickup(B)
  HOLDING(B)
  CLEAR(C)
  HOLDING(B) & CLEAR(C)
  stack(B,C)
  ON(B,C)
[ ON(A,B) & ON(B,C)

```

Joonis 30.4. Eesmärkide pinu pärast nelja esimese reegli rakendamist. Kaitstud eesmärgid ON(A, B) ja ONTABLE(B)

Ploki B peal paikneb plokk A, mistõttu joonisel 30.4 toodud plaani kohaselt tähendaks ploki B vabastamine, et enam ei kehti olukord, kus plokk A asub ploki B peal.

Selle momendini rakendatud reeglite järjend on

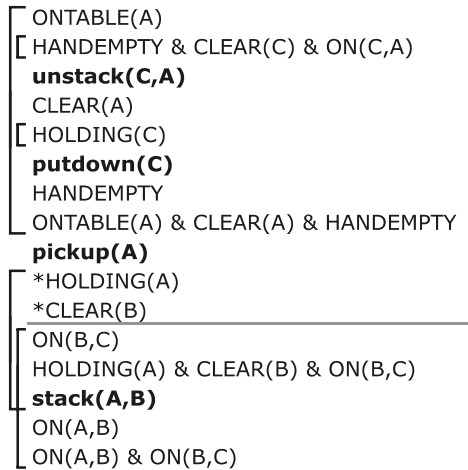
{unstack(C, A), putdown(C), pickup(A), stack(A, B)}.

Alameesmärgid ON(A, B) ja ONTABLE(B) on saavutatud ja kaitstud. Reegli unstack(A, B) eeldused on küll täidetud, kuid reegli rakendamine viiks kaitstud oleku ON(A, B) rikkumiseni. Kuidas nüüd toimida?

Alguses kontrollib RSTRIPS, kas tingimust ON(A, B) võiks saavutada, kui rakendada reegleid, mis paiknevad markerist allpool talle vastava sulu sees. Alles sulu lõpus on vaja, et ON(A, B) oleks tõene. Oletame, et üks reegel sulu sees on selline, et tema rakendamise tulemusel võib taas saavutada selle tingimuse täidetuse. Sellisel juhul oleksid ajutised rikkumised lubatud. Meie juhul aga ei leidu ühtki sellist reeglit, mis muudaks ON(A, B) uuesti tõeseks, ja seepärast peab RSTRIPS kaitstud eesmärgi rikkumise vältimiseks võtma ette midagi muud.

Rikutava eesmärgi juurde kuuluv liiteesmärk sulu lõpus on ON(A, B) & ON(B, C). Ühe komponendi ON(B, C) lahendamiseks vajalik reegel viib teise komponendi kaitstuse rikkumiseni. Ütleme, et komponent ON(B, C) rikub kaitstust. Süsteem püüab vältida

kaitstuse rikkumist selle komponendi regressiooni teel, minnes tagasi mööda reeglite ahelat (markerist kõrgemal), kuni selle reeglini, millega saavutatakse kaitstud alam-eesmärk. Viimane reegel, mida on vaja rakendada, s.t $stack(A, B)$, tagab samuti tingimuse $ON(A, B)$ täidetuse. Seepärast saab RSTRIPS eesmärgi $ON(B, C)$ regressiooni reeglil $stack(A, B)$, mis viib tingimuseni $ON(B, C)$. Sel juhul ei muuda regressioon alameesmärki ja süsteem püüab plaani selles punktis, mis eelneb vahetult reegli $stack(A, B)$ rakendamisele, saavutada eesmärki, mis saadi pärast regressiooni. Selle regressiooniprotsessi tulemusel moodustub eesmärkide pinu, mis on näidatud joonisel 30.5.



Joonis 30.5. Eesmärgi $ON(B, C)$ regressiooni tulemusel saadud eesmärkide pinu

Liiteesmärk $ON(A, B) \& ON(B, C)$ sulu lõpus, millel tuvastati potentsiaalne rikkumine, on pinus säilitatud. Teised elemendid, mis asuvad madalamal kui $ON(A, B)$, olid eelmises pinus eesmärgi $ON(B, C)$ saavutamise osa. Need elemendid on pinust eemaldatud. Plaan eesmärgi $ON(A, B)$ saavutamiseks reegli $stack(A, B)$ rakendamise tulemusel kehtib ja on pinus säilitatud. Tuleb tähele panna, et pärast eesmärgi $ON(B, C)$ regressiooni on eesmärk ühendatud liiteeldusega vahetult reegli $stack(A, B)$ kohal. Kuna marker löikab sulgu, siis on eesmärgid $HOLDING(A)$ ja $CLEAR(B)$ kaitstud.

RSTRIPS alustab uuesti tööd selle eesmärkide pinuga ja kaitstuse rikkumisi ei esine kuni selle momendini, kuni konstrueeritakse pinu joonisel 30.6.

```

[ ONTABLE(A)
[ HANDEEMPTY & CLEAR(C) & ON(C,A)
  unstack(C,A)
  CLEAR(A)
[ HOLDING(C)
  putdown(C)
  HANDEEMPTY
[ ONTABLE(A) & CLEAR(A) & HANDEEMPTY
  pickup(A)
  *HOLDING(A)
  *CLEAR(B)
  *ONTABLE(B)
  *CLEAR(B)
[ HOLDING(x)
  putdown(x)
  HANDEEMPTY
[ ONTABLE(B) & CLEAR(B) & HANDEEMPTY
  pickup(B)
  HOLDING(B)
  CLEAR(C)
[ HOLDING(B) & CLEAR(C)
  stack(B,C)
[ ON(B,C)
  HOLDING(A) & CLEAR(B) & ON(B,C)
  stack(A,B)
  ON(A,B)
  ON(A,B) & ON(B,C)

```

Joonis 30.6. Eesmärkide pinu momendil, kus reeglit putdown(A) ei saa rakendada, sest alameesmärk HOLDING(A) on kaitstud

Regressiooni kasutamine näitab, et reegli putdown(A) eeldus vastab jooksvale oleku-kirjeldusele, kuid reegli rakendamine viiks alameesmärgi HOLDING(A) kaitstuse rikkumiseni. Sellel rikkumisel ei ole ajutist iseloomu. Rikkumise vältimiseks viib RSTRIPS läbi selle komponendi regressiooni, mis rikub ON(B, C) kaitstust, veel kaugemale tagasi ja seekord reeglile pickup(A). Pärast regressiooni on pinul kuju, nagu näidatud joonisel 30.7.

```

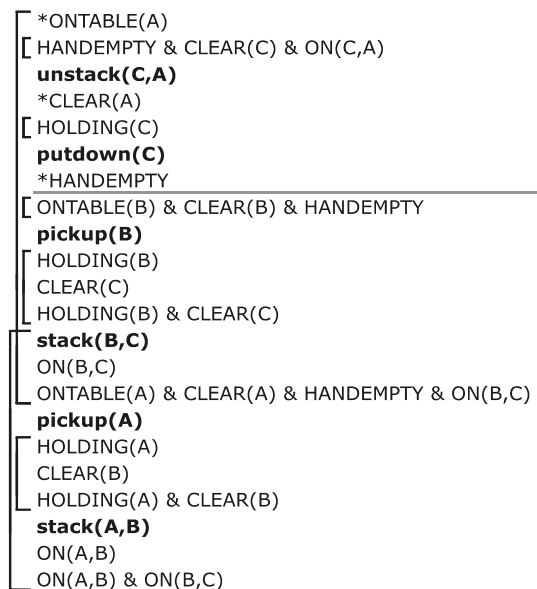
[ *ONTABLE(A)
[ HANDEEMPTY & CLEAR(C) & ON(C,A)
  unstack(C,A)
  *CLEAR(A)
[ HOLDING(C)
  putdown(C)
  *HANDEEMPTY
[ ON(B,C)
[ ONTABLE(A) & CLEAR(A) & HANDEEMPTY & ON(B,C)
  pickup(A)
[ HOLDING(A)
  CLEAR(B)
[ HOLDING(A) & CLEAR(B)
  stack(A,B)
  ON(A,B)
  ON(A,B) & ON(B,C)

```

Joonis 30.7. Eesmärkide pinu pärast eesmärgi HOLDING(A) regressiooni

Siin on säilinud tingimuse ON(A, B) saavutamise plaan, aga välja lülitatud tingimuse ON(B, C) saavutamise plaan, mis rikub kaitset.

Jätkates sellest samast eesmärkide pinust, avastab RSTRIPS teise võimaliku kaitstuse rikkumise, kui on moodustanud joonisel 30.8 toodud pinu.



Joonis 30.8. Eesmärkide pinu, kus reegli pickup(B) rakendamine rikub tingimuse HANDEEMPTY kaitset

Kui rakendada reeglit pickup(B), siis on rikutud tingimuse HANDEEMPTY kaitse. Nüüd aga on sellel rikkumisel ajutine iseloom. Üks järgmistest reeglitest, nimelt stack(B, C) vastava sulu piires, viib tingimuse HANDEEMPTY taaskordse saavutamiseni.

Antud näites leiab RSTRIPS lühema lahendava järjendi võrreldes süsteemiga STRIPS, nimelt {unstack (C, A), putdown(C), pickup(B), stack(B, C), pickup(A), stack(A, B)}.

31. Probleemilahenduse planeerimine eesmärkide koostoime korral: süsteem DCOMP

DCOMP on samuti koostoimeliste eesmärkidega töötav süsteem. Ta töötab kahes faasis. Esimeses faasis loob ta ajutise „lahendi” eeldusel, et koostoimet ei ole. Eesmärkide avaldised esitatakse JA/VÕI graafidena ja reegleid rakendatakse tagantpoolt ettepoole, literaalsetele tippudele, mis ei vasta algoleku kirjeldusele. See faas lõpeb kooskõlalise lahendusgraafi loomisega, mille lõpptipud on vastavuses algoleku kirjeldusega. See lahendusgraaf on ülesande vahepealne lahend ja tuleb ümbertöötamisele teises faasis, et arvesse võtta vastastikuseid toimeid.

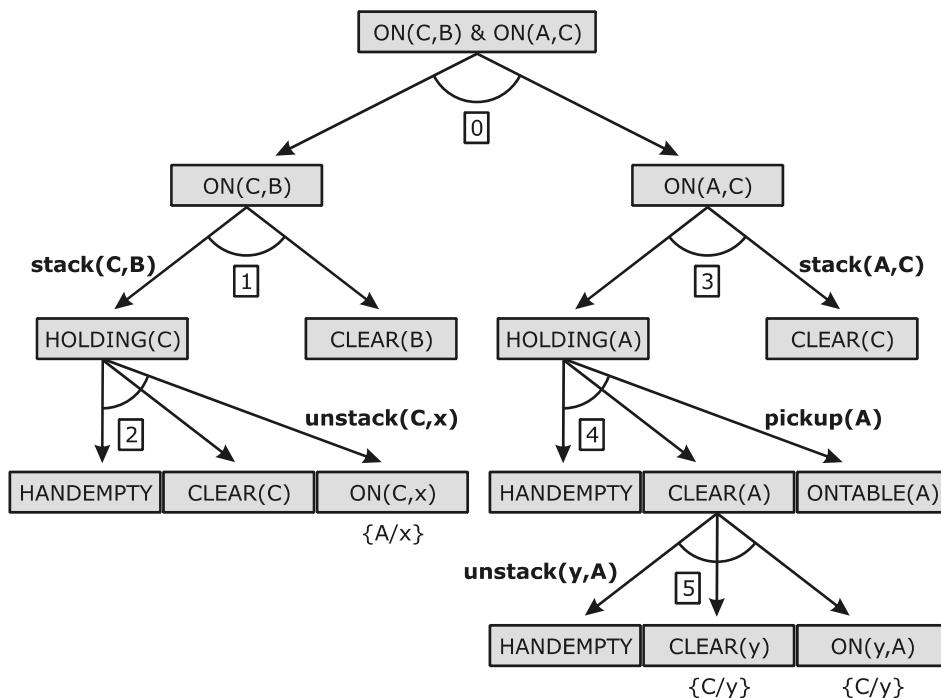
JA/VÕI graafi lahendusgraaf asetab lahendussammudele osalise järjestuse kitsenduse. Kui vastastikuseid toimeid ei oleks, siis võiks lahendusgraafis reegleid, mis ei ole seotud vanemlike tippude kaudu, rakendada paralleelselt, mitte järjestikku. Vahel lubab roboti konstruktsioon mitme tegevuse täitmist korraga. Näiteks võib robot liikudes kätt liigutada. Sel määral, nagu on võimalikud paralleelsed tegevused, on soovitatav väljendada roboti tegevuste järjendeid kui osaliselt järjestatud tegevusi. Teatud konkreetse eesmärgi saavutamisel on seda parem, mida vähem piiranguid on seotud tegevuste sooritamise järjekorrale.

Nagu mainitud, töötab süsteem DCOMP kahes faasis:

1. leitakse esialgne lahend, arvestamata osaprobleemide omavahelisi seoseid,
2. esialgse lahendi töötlus.

Vaatame varasemat näidet joonisel 28.11.

Lahenduse esimeses faasis saadakse tulemus, mis on esitatud joonisel 31.1. Lõpptippude juures on näidatud lihtsaimad unifikaatorid. Reeglite rakendamised on nummerdatud (suvaliselt). DCOMP rakendab reegleid tagantpoolt ettepoole seni, kuni kõik alameesmärgid on vastavuses algoleku kirjeldusega. Puudub vajadus regressiooni järele, kuna siin eeldatakse koostoime puudumist.



Joonis 31.1. Näiteprobleemi (joonis 28.11) lahenduse esimene faas

Nagu näeme (joonis 31.1), koosneb lahendus kahest reeglite jadast:

{unstack(C,A), stack(C, B) } ja {unstack(C,A), pickup(A), stack(A,C)}.

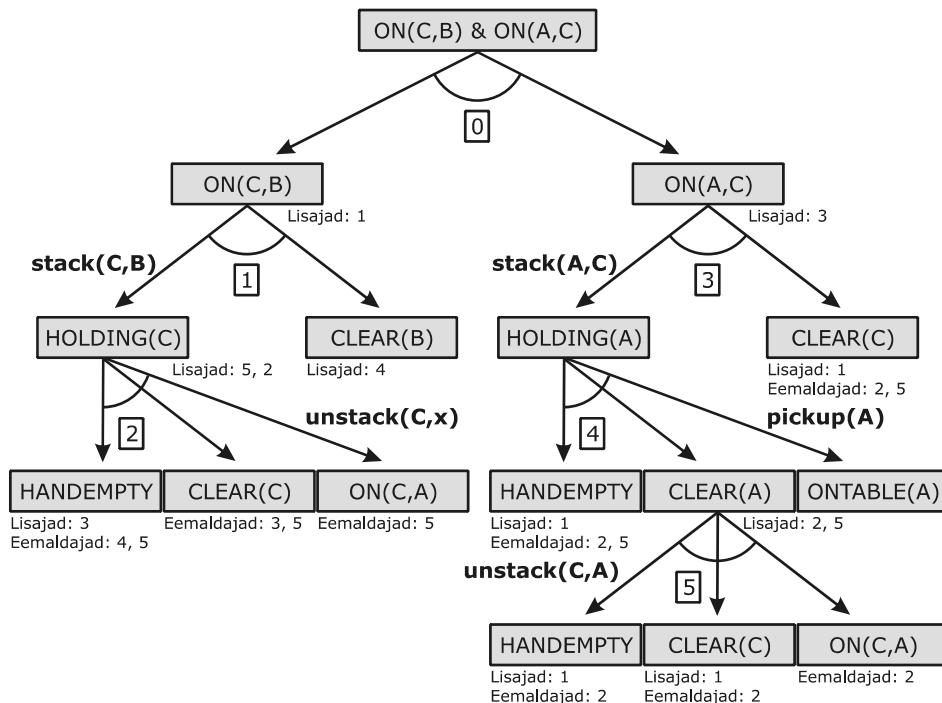
Kahjuks aga ei ole neid tegevusi võimalik sooritada paralleelselt. Näiteks reegel 5 kustutab eelduse HANDEEMPTY, mis on vajalik reeglile 2. Seetõttu ei ole võimalik rakendada reeglit 5 vahetult enne reeglit 2. Veelgi halvem on, et reeglile 5 vahetult järgnev reegel 4 eeldab samuti tingimuse HANDEEMPTY kehtivust. Seega on esialgsel lahendusel mitmeid puudusi, mis on seotud reeglite vastastikuse toimega.

Püüame eraldada reeglite sõltumatud osad.

Olgu C_{ij} reegli j osa i . Iga C_{ij} jaoks leitakse kaks hulka, mis võivad olla ka tühjad. Esimene neist on nn eemaldajad ehk D_{ij} -reeglid, mis **eemaldavad** (delete) C_{ij} (need reeglid ei tohi reeglile j eelne olla j ise).

Teine hulk on nn lisajad ehk A_{ij} -reeglid, mis **lisavad** (add) C_{ij} (ei eelne reeglile j ega ole j ise). Lisajate hulk on oluline seetõttu, et võimaldab valida reeglite järjekorda nii, et oleks võimalik kõrvaldada kustutamisreeglite tagajärjed.

Joonisel 31.2 on lahendusgraafile peale joonistatud lisajad ja eemaldajad.



Joonis 31.2. Näiteprobleemi (joonis 28.11) lahenduse esimene faas koos lisajate ja eemaldajatega

Ütleme, et leiab aset reeglite osaline sõltumatu järjestus, kui iga C_{ij} korral graafis kehtib üks tingimustest:

1. reegel j on eespool kõigist hulga D_{ij} reeglitest (sel juhul ei eemaldata tingimust C_{ij} enne, kui on rakendatud reegel j).
2. Leidub mingi reegel hulgas A_{ij} , näiteks reegel k , mis asub enne reeglit j , ja mitte ükski hulga D_{ij} element ei asu reeglite k ja j vahel.

Teises faasis püüab süsteem teisendada saadud järjestuse sõltumatuks järjestuseks. Sageli osutub selline teisendus ka teostatavaks. Selleks on kaks võimalust:

1. võib proovida veelgi piirata järjestust, et saaks rakendada eespool esitatud kahte punkti, või
2. püüda heita kõrvale mõned reeglid, mille kasutamine rikub eeldusi.

Näiteks reegel 3 eemaldab reegli 2 eelduse CLEAR(C). Kui aga reegel 2 eelneks reeglile 3, siis reegel 3 ei saaks tingimust kustutada.

Täpselt samuti eemaldab reegel 5 reegli 4 eelduse HANDEMPTY. Me ei saa teha nii, et reegel 4 asuks reeglist 5 eespool. Kuid me võime kasutada reeglite 5 ja 4 vahel reeglit 1.

Kuna reegel 2 asub reegli 4 ees ja pärast kõiki tingimuse CLEAR(C) hulga D elemente, siis välistame me täielikult reegli 5, sest CLEAR(C) lisatakse reeglina 2.

Süsteemis DCOMP püütakse saavutada reeglite järjestust kas täiendavate reeglite lisamise või juba olemasolevate kõrvalejätmise teel.

Mõistliku reeglite hulga leidmine on komplitseeritud ülesanne ja siin saame me seda vaadata ainult mitteformaalsel tasemel.

Täiendavad kitsendused peavad olema sõltumatud. Teatud juhtudel ei ole süsteem suuteline leidma sobivat reeglite järjestust. Meie näites see siiski õnnestub, kui kasutada järgmisi tegevusi.

1. Paigutada reegel 2 reegli 4 ette ja eemaldada reegel 5. Märgime, et reegel 4 ei saa nüüd kustutada ühtegi reegli 2 eeldust. Lisaks, kuna reegel 2 asub reeglist 3 eespool, siis ka reegel 3 ei saa kustutada ühtegi reegli 2 eeldust.
2. Paigutada reegel 1 ettepoole reeglist 4. Kuna reegel 1 rakendub pärast reeglit 2 ning enne reegleid 3 ja 4, siis ta taastab tingimused, mis kustutati reegli 2 poolt, kuid mis on vajalikud reeglites 3 ja 4.

Need täiendavad kitsendused viivad meid tegevuste jadani:

{2, 1, 4, 3}, s.t

{unstack(C,A), stack(C,B), pickup(A), stack(A,C)}.

Siiani vaatlesime olukordi, kus kõiki tegevusi sooritati järjestikuselt – robotil oli vaid üks „käsi” ja see suutis teha ainult ühe tegevuse korraga.

Nüüd aga oletame, et meil on **kahekäeline robot** ja kumbki käsi suudab teha meie poolt fikseeritud tegevusi. Me peame ainult modifitseerima operaatoreid – tuleb ju näidata ka seda, kumma käega tegevust sooritatakse.

Operaatorid kahekäelisele robotile (h tähistab kätt; 1 või 2):

pickup(X,h)

if: ONTABLE(X), CLEAR(X), HANDEEMPTY(h)

add: HOLDING(X,h)

delete: ONTABLE(X), CLEAR(X), HANDEEMPTY(h)

putdown(X,h)

if: HOLDING(X,h)

add: ONTABLE(X), CLEAR(X), HANDEEMPTY(h)

delete: HOLDING(X,h)

stack(X,Y,h)

if: HOLDING(X,h), CLEAR(Y)

add: HANDEEMPTY(h), ON(X,Y), CLEAR(X)

delete: HOLDING(X,h), CLEAR(Y)

unstack(X,Y,h)

if: HANDEEMPTY(h), CLEAR(X), ON(X,Y)

add: HOLDING(X,h), CLEAR(Y)

delete: HANDEEMPTY(h), CLEAR(X), ON(X,Y)

Vaatame jälle varasemast tuttavat ülesannet (joonis 28.11).

Esimene faas

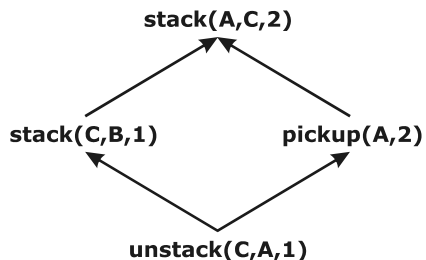
Vaatame eespool toodud lahendusgraafi, mis saadi esimesel etapil (joonis 31.2), aga kus nüüd on tingimuse HANDEEMPTY asemel HANDEEMPTY(1) ja HANDEEMPTY(2). Juhi-me tähelepanu sellele, et tingimuse HANDEEMPTY hulk D on nüüd väiksem, sest robotil on kaks kätt.

Teine faas

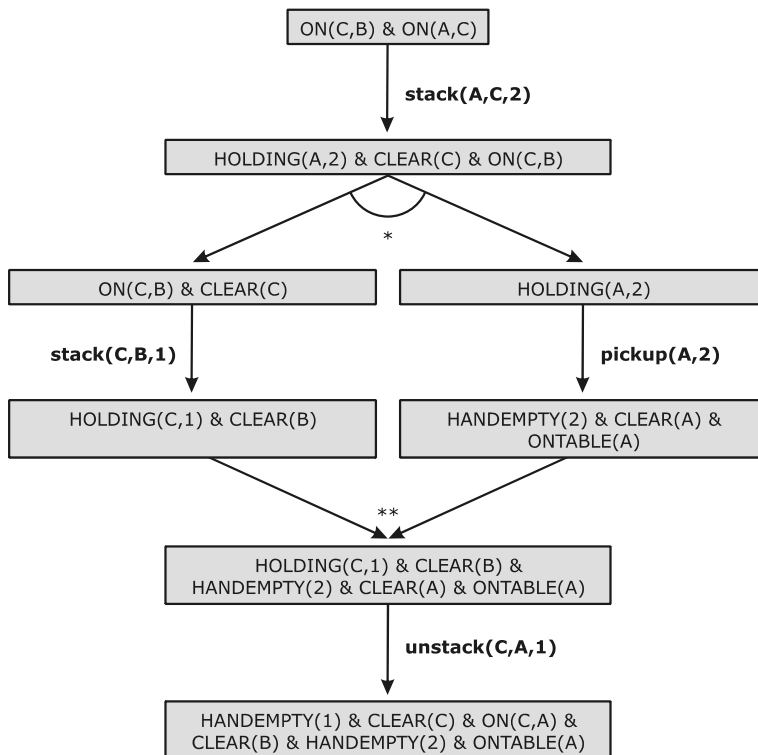
Süsteem võib leida, et

1. reegel 2 peab olema enne reeglit 4, nii et reegel 5 heidetakse kõrvale,
2. reegel 2 peab eelnema reeglile 3, et tingimuse CLEAR(C) eemaldamine jääks reeglile 2,
3. kui nüüd reegel 1 asub reeglite 2 ja 3 vahel, siis taastatakse tingimus CLEAR(C) reeglile 3.

Need täiendused viivad meid üldise plaanini (joonis 31.3), mis on esitatav graafina joonisel 31.4. Selliseid struktuure nimetatakse ka **protseduurvõrkudeks**.



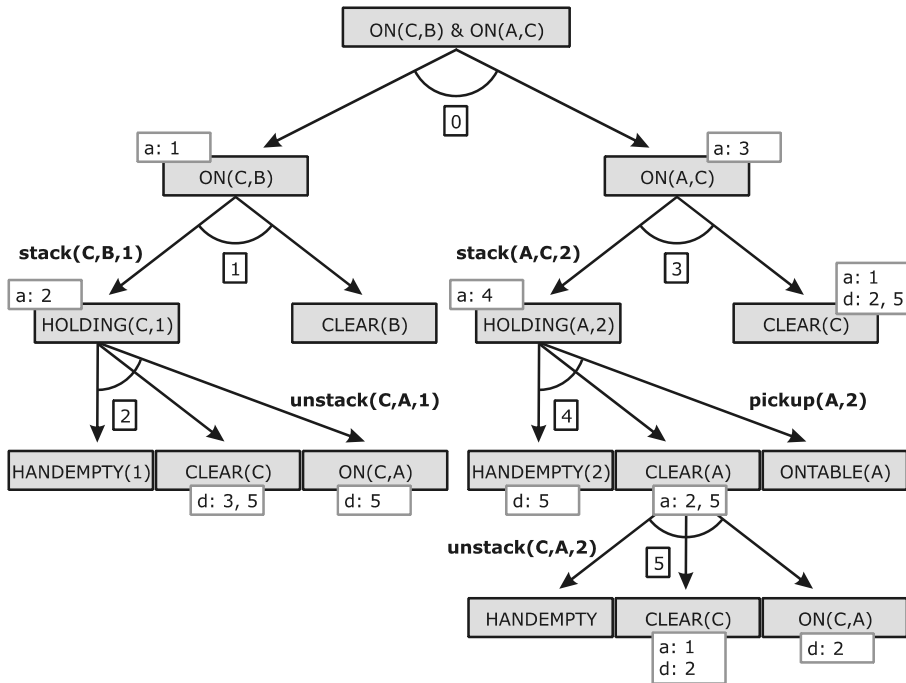
Joonis 31.3. Näiteprobleemi (joonis 28.11) lahendamise üldine plaan kahekäelisele robotile



Joonis 31.4. Protseduurvõrk näiteprobleemi (joonis 28.11) lahendamiseks kahekäelisele robotile

Alustades eesmärgitipust, liigume vastupidises suunas, genereerides vastavaid alamees-märke. Punktis * on võimalik alameesmärgid jagada kaheks, s.t neid alamülesandeid saab lahendada paralleelselt kuni olekuni **.

Joonisel 31.5 on protseduurvõrku täiendatud lisajate ja eemaldajatega (hulkadega a ja d).



Joonis 31.5. Näiteprobleemi (joonis 28.11) lahendamise plaan kahekäelisele robotile koos lisajate ja eemaldajatega

32. Hierarhiline planeerimine: süsteem ABSTRIPS

Punktides 28–31 vaadeldud planeerimismeetodid töötavad ühel tasandil. Näiteks vaatasime probleemi „tagantpoolt ettepoole” lahendamises esmalt, kuidas saavutada peaesmärgi eeldust, ja seejärel tegime sedasama kõikide tema alameesmärkide jaoks. Paljudes praktilistes situatsioonides võime aga vaadelda eesmärkide ja alameesmärkide tingimusi lihtsalt kui teatavaid detaile ja lükata nende lahendamise edasi kuni momendini, kus plaani põhilised sammud on juba leidnud oma koha. Eesmärkide tingimused ja nende saavutamise reeglid võib organiseerida hierarhiasse, kus kõige alumisel tasemel asuvad kõige üksikasjalikumad tingimused ja nendele vastavad detailsed tegevused, aga kõige kõrgemal tasemel peamised tingimused ja reeglid nende saavutamiseks.

Näiteks maja planeerides on meil kõrgeima taseme ülesanneteks ehitusplatsi ettevalmistamine, vundamendi tegemine, karkassi püstitamine, kütte- ja elektritööd jms. Madalaima taseme tegevused on seotud nt naelte ettevalmistamisega, juhtmejuppide parajaks lõikamisega jms. Kui plaan tervikuna tuleks kohe sünteesida kõige detailsemate tegevuste tasemel, siis oleks tee tema loomiseni lubamatult pikk. Töötades välja plaani tase-haaval, saavutame, et igal tasemel on plaanil mõistlik suurus, mis tõstab tema eduka konstrueerimise tõenäosust. Sellist strateegiat nimetatakse hierarhiliseks planeerimiseks ja sellega tutvusime punktis 27.1.

Plaanide ülesehitamise protsess on suhteliselt ilmne sellise otsuste vastuvõtmise süsteemi korral nagu STRIPS, seetõttu illustreerimegi hierarhilist planeerimist selle süsteemi näitel.

Sellisel modifitseeritud süsteemi nimetatakse süsteemiks ABSTRIPS (Abstraction Basis STRIPS). Kasutame neidsamu reegleid mis varem, kuid eelduste täitmise edasilükkamiseks on meil vaja defineerida tingimuste hierarhia.

Vaatame näitena jälle eesmärki [ON (C, B) & ON (A, C)] ja algolekut joonisel 28.11. Sellise eesmärgi saavutab hõlpsasti ka ühetasemeline süsteem STRIPS, kuid siin kasutame seda süsteemi ABSTRIPS töö illustreerimiseks. Et näide oleks mõistlik, peab predikaatide hierarhia illustreerima raskusi, mis on seotud erinevate tingimuste saavutamise. Selge on, et eesmärgi ON peamine predikaat peab asuma hierarhia ülemisel tasemel ja HANDEEMPTY kõige alumisel, sest seda on lihtne saavutada. Selles lihtsas näites kasutame ainult kolme hierarhilist taset ja paigutame kolm ülejäänud predikaati ONTABLE, CLEAR ja HOLDING keskmisele tasemele.

Iga tingimuse hierarhilise taseme võib näidata tema kriitilisuse väärtusega. Väikesed arvud osutavad madalat hierarhilist taset või tingimuse väikest kriitilisust. Suured arvud näitavad vastavalt kõrget hierarhilist taset või suurt kriitilisust. Järgnevalt on toodud süsteemi ABSTRIPS reeglid, kus kriitilisuse väärtused on näidatud vastavate tingimuste kohal (vrd p 28).

pickup(X)

| | | | |
|---------|----------------------------------|---|---|
| | 2 | 2 | 1 |
| if: | ONTABLE(X), CLEAR(X), HANDEEMPTY | | |
| | 2 | | |
| add: | HOLDING(X) | | |
| | 2 | 2 | 1 |
| delete: | ONTABLE(X), CLEAR(X), HANDEEMPTY | | |

putdown(X)

| | | | |
|---------|----------------------------------|---|---|
| | 2 | | |
| if: | HOLDING(X) | | |
| | 2 | 2 | 1 |
| add: | ONTABLE(X), CLEAR(X), HANDEEMPTY | | |
| | 2 | | |
| delete: | HOLDING(X) | | |

stack(X,Y)

| | | | |
|---------|-------------------------------|---|---|
| | 2 | 2 | |
| if: | HOLDING(X), CLEAR(Y) | | |
| | 1 | 3 | 2 |
| add: | HANDEEMPTY, ON(X,Y), CLEAR(X) | | |
| | 2 | 2 | |
| delete: | HOLDING(X), CLEAR(Y) | | |

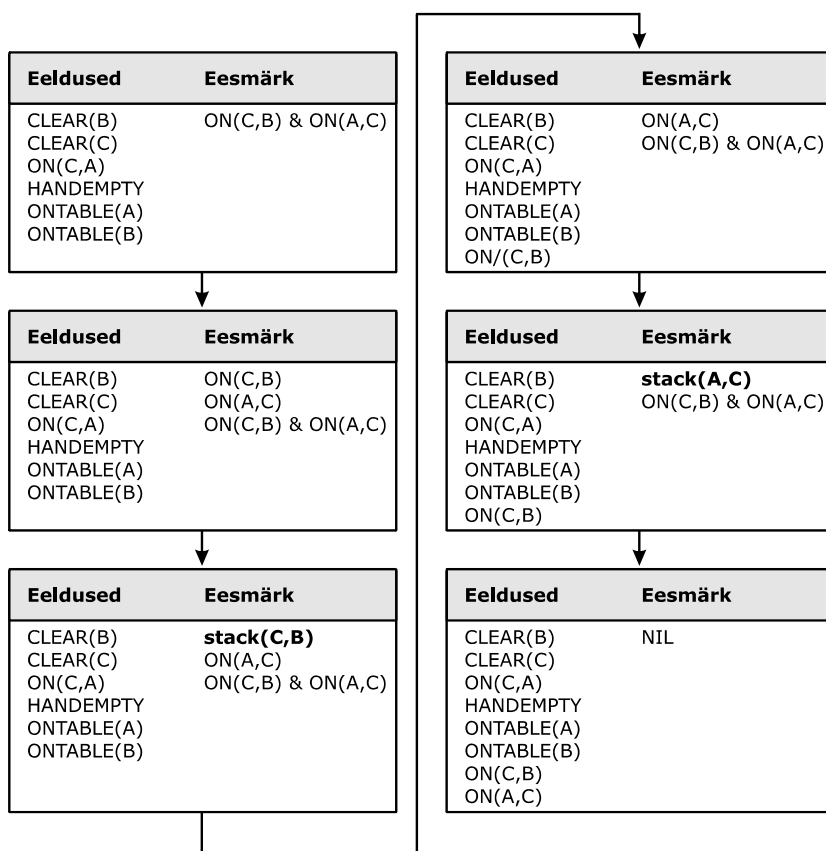
unstack(X,Y)

| | | | |
|---------|-------------------------------|---|---|
| | 1 | 2 | 3 |
| if: | HANDEEMPTY, CLEAR(X), ON(X,Y) | | |
| | 2 | 2 | |
| add: | HOLDING(X), CLEAR(Y) | | |
| | 1 | 2 | 3 |
| delete: | HANDEEMPTY, CLEAR(X), ON(X,Y) | | |

Kriitilisuse väärtused esinevad nii eelduste kui ka eemaldamiste nimestiku literaalide kohal. Neid ei ole lisamiste nimestiku literaalide kohal. Kui rakendatakse mingit reeglit, siis lülitatakse olekukirjeldusse kõik lisamiste nimestiku literaalid.

ABSTRIPS alustab tööd suurimate kriitilisuste vaatamisest, s.t meie näites 3. Kõik tingimused, mille kriitilisus on väiksem kui 3, on nähtamatud, s.t neid ei võeta arvesse. Kuna peamine eesmärk sisaldab kaks tingimust, mille kriitilisus on 3, siis alustab ABSTRIPS ühega neist, nt ON (C, B), ja lisab eesmärkide pinusse reegli stack(C, B). Sinna ei lisata selle reegli eeldusi, sest nende kriitilisuse väärtus on 2 (allpool läve) ja sellel tasandil on nad nähtamatud.

Nüüd saab ABSTRIPS rakendada reeglit stack(A, B), mis viib uue olekukirjelduseni. Selles olekus vaadatakse eesmärgi teist komponenti ON(A,C) ja lisatakse eesmärkide pinusse reegel stack(A, C). Seejärel rakendatakse reeglit stack(A,C) jooksvale olekukirjeldusele, mis viibki kogu eesmärgile vastava olekukirjelduseni. Joonisel 32.1 on näidatud süsteemi ABSTRIPS töö sellel tasemel.



Joonis 32.1. Lahendustee süsteemis ABSTRIPS kõrgeimal tasemel

Kõrgeima taseme lahendiks, mis saavutatakse mõningate detailide ignoreerimisega, on reeglite järjend {stack (C,B), stack (A,C)}. (Komponentide teistsuguse järjestuse korral

oleks kõrgeimal tasemel õigeks lahendiks ka {stack (A,C), stack (C, B)}. See lahend aga tekitab raskusi madalamal tasemel, mis tingib vajaduse tagasi pöörduda kõrgemale tasemele.) Meie kõrgeima taseme lahendit võib vaadelda kui eesmärgi saavutamise kõrgtaseme plaani. Sellest seisukohast loetakse kõige tähtsamaks kuubikute üksteisele asetamise operatsiooni, madalama taseme planeerimist võib kasutada mõnede detailide väljaselgitamiseks.

Nüüd antakse esimese taseme lahend edasi teisele tasemele. Teisel tasemel vaatame tingimusi, mille kriitilisuse väärtus on 2 ja rohkem, nii et hakkame arvesse võtma mõningaid detaile. Ülemise taseme lahendi võib nüüd anda allapoole, alustades protsessi järgmisel tasemel eesmärkide pinust, mis sisaldab reeglite järjendi kõrgema taseme lahendist, koos ühega nende nähtavatest eeldustest. Ülemine element pinu tipus on peaesmärk. Sellisel juhul on eesmärkide pinu algus teise taseme jaoks järgmine:

```
HOLDING(C) & CLEAR(B)
*stack(C, B)
HOLDING(A) & CLEAR(C)
*stack(A, C)
[ON(C, B) & ON(A, C)]
```

Kuna STRIPS töötab eesmärkide pinuga, siis pole raske teha nii, et järgmine tase korrigeeriks reegleid detailide saavutamiseks. Plaan, mis laskub alla kõrgematelt tasemetelt, piirab märgatavalt otsingut madalamatel tasemetel, tagades sellega arvutuste efektiivsuse.

Võib veenduda, et teisel tasemel on üks võimalik lahend

```
{unstack(C, A), stack(C,B), pickup(A), stack(A, C)}.
```

Kui mingil vahepealsel tasemel ei suudeta leida lahendit, siis pöördub protsess tagasi kõrgemale tasemele, et leida uus lahend. Meie näites on teise taseme lahend hea ja täielik, kui mitte arvesse võtta, et selle konstrueerimisel ignoreerisime tingimust HANDEEMPTY.

Kolmanda ehk madalaima taseme lahendamise käigus alandame kriitilisuse väärtust üheni. Eeldame, et eesmärkide pinu sisaldab teise taseme reeglite järjendi koos eeldustega (kuid mitte kõigiga). Meie näites viib töö sellel tasemel kontrollimiseni, kas teise taseme lahend on õige ka kolmanda taseme detailide täpsusega.

Seega on ABSTRIPS õigustatud hierarhilise planeerimise puhul. Kõik, mis selleks on vaja, on predikaatide tähtsuse gradueerimine, mis toimub nende kriitilisuse väärtuste omistamise teel. Raskemate ülesannete puhul kui käesolev näide on ABSTRIPS palju efektiivsem kui ühetasemeline STRIPS.

V. TEHISINTELLEKTI TEISI KÜSIMUSI

Eelmistes peatükkides käsitlesime tehisintellekti kolme põhiküsimust: teadmuse esitust, probleemilahendust ja planeerimist. Siin vaatleme põgusalt veel kahte põhiküsimust: suhtlust e infovahetust tehisintellektisüsteemi ja väliskeskonna vahel ning õppimist e uue teadmuse omandamist. Peatüki lõpul anname ülevaate tehisintellekti tark- ja riistvarast.

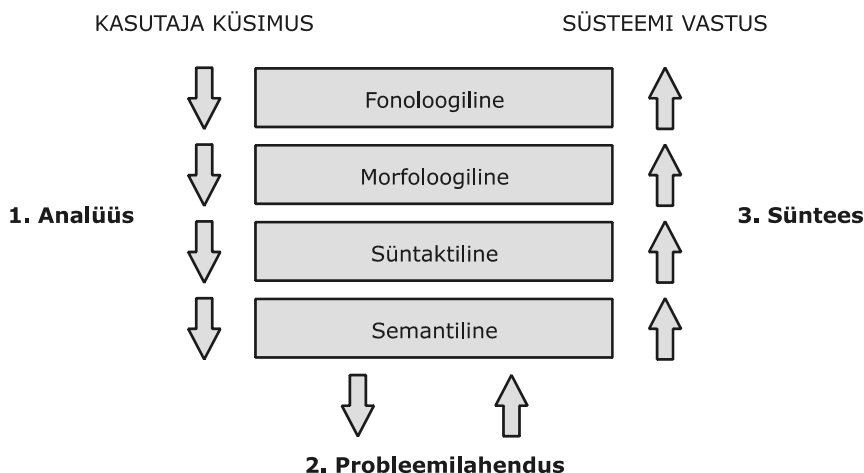
33. Suhtlus loomulikus keeles

Inimene võtab vastu infot väliskeskkonnast oma viie meele – kuulmise, nägemise, maitsmise, haistmise ja kompimise abil. Tähtsaimad meeled on kuulmine ja nägemine. Meeli tuleks modelleerida ka tehisintellektisüsteemis, eeskätt intelligentses robotis.

Inimesed kasutavad omavahelises suhtlemises kõige enam loomulikku keelt. Mugav oleks ka arvutiga (ekspertsüsteemiga, robotiga) suhelda inimkeeles. Vaatleme järgnevas, kuidas arvuti töötleb keelelist infot.

33.1. Loomuliku keele automaattöötlus

Tehisintellektisüsteem, mis suhtleb kasutajaga loomulikus keeles, täidab oma töös kolme etappi: 1) kasutaja lausungi (nt küsimuse) analüüs, 2) probleemilahendus ja 3) vastuse süntees. Nii analüüs kui ka süntees koosnevad omakorda neljast alametapist: fonoloogiline, morfoloogiline, süntaktiline ja semantiline analüüs ja süntees (joonis 33.1).



Joonis 33.1. Loomuliku keele automaattöötluse etapid

Fonoloogilisel analüüsil tuvastatakse kasutaja kõnevoos häälikud ja sõnad ning esitatakse need kirjalikus keeles, morfoloogilisel analüüsil määratakse iga sõna tunnused (algvorm, kääne jms), süntaktilisel analüüsil leitakse (igas) lauses lauseliikmed ning semantilisel analüüsil määratakse (iga) lause tähendus. Tähenduse esitamise formalismiks võetakse sageli predikaatloogika. Järgneval probleemilahenduse etapil tuletatakse kasutajale antava vastuse semantiline esitus – tähendus, näiteks püstitatakse ja lahendatakse selleks teoreemi automaatse tõestamise probleem, kus eeldusteks on teadmusbasis olevad faktid ning väiteks kasutaja küsimus. Ka sünteesil läbitakse needsamad etapid mis analüüsilgi, kuid vastupidises järjekorras.

Näide 33.1. Oletame, et tehisintellektisüsteemi sisendile saabus kasutaja kõne, mille fonoloogiline analüüs andis tulemuseks eestikeelse lause

Kas Tartu rong väljub õhtul?

Vaatame, kuidas võiks toimuda selle lause edasine töötlus.

- Morfoloogiline analüüs

kas

kas küsisõna

Tartu

Tartu pärisnimi ainsus nimetav/omastav/sisseütlev

rong

rong nimisõna ainsus nimetav

väljub

väljuma verb kindel kõneviis olevik 3. isik ainsus aktiiv jaatav kõne

õhtul

õhtu nimisõna ainsus alalütlev

- Süntaktiline analüüs

kas

valikküsilause tunnus

Tartu

täiend

rong

alus

väljuma

öeldis

õhtu

määrus

- Semantiline analüüs

See on kas-küsimus:

$\exists x (\text{rong}(x) \ \& \ \text{sihtkoht}(x, \text{Tartu}) \ \& \ \text{väljumisaeg}(x, \text{õhtu}))$

Seega soovib kasutaja teada saada, kas leidub rong, mille sihtkohaks on Tartu ja väljumisajaks õhtu.

Sellest üksikust lausest ei selgu küll, missugusest jaamast rong peaks väljuma, kuid süsteem võib vaikimisi eeldada, et lähtejaam on asulas, kust kasutaja temale helistab. Teine ebamäärasus on seotud sellega, missugusest kellaajast algab õhtu. Loeme õhtuks näiteks ajavahemiku kell 18–24. Niisiis saab püstitada teoreemi automaatse tõestamise probleemi ja selle lahendada. Oletame, et probleemilahendus andis tulemuseks, et sõiduplaanis ei leidu ühtegi rongi, mis väljuks Tartusse sellel ajavahemikul. Samas, kui kasutaja küsis rongi väljumist, siis on tal tõenäoliselt kavatsus õhtul Tartusse sõita. Süsteem kui koostöövalmis suhtluspartner võiks kasutajale soovitada rongi asemel bussi ja teatada ka õhtuste busside väljumisajad. Seega võiks probleemilahenduse tulemuseks olla näiteks järgmine (tulevase vastuse) semantiline esitus:

$$\begin{aligned} & \neg \exists x (\text{rong}(x) \ \& \ \text{sihtkoht}(x, \text{Tartu}) \ \& \ \text{väljumisaeg}(x, \text{õhtu})) \ \& \\ & \exists y (\text{buss}(y) \ \& \ \text{sihtkoht}(y, \text{Tartu}) \ \& \ \text{väljumisaeg}(y, 20.30)) \ \& \\ & \exists z (\text{buss}(z) \ \& \ \text{sihtkoht}(z, \text{Tartu}) \ \& \ \text{väljumisaeg}(z, 21.15)) \ \& \\ & \exists w (\text{buss}(w) \ \& \ \text{sihtkoht}(w, \text{Tartu}) \ \& \ \text{väljumisaeg}(w, 23.00)) \end{aligned}$$

Järgneb vastuse süntees.

- Semantilisel sünteesil määratakse, missugustest lausetest vastus koosneb, ja otsustatakse, missuguste lauseliikmete rolli hakkavad täitma semantilise esituse termid. Olgu näiteks
 - 1. lause:
 - Tartu – täiend, rong – alus, mitte-väljuma – öeldis, õhtu – määrus.
 - 2. lause:
 - Tartu – täiend, buss – alus, väljuma – öeldis, 20.30 – määrus, 21.15 – määrus, 23.00 – määrus
- Süntaktilisel sünteesil otsustatakse, missuguses muutevormis peab olema iga lauseliige:

Tartu (pärisnimi, ainsus, omastav)

rong (nimisõna, ainsus, nimetav)

väljuma (teigusõna kindel kõneviis olevik 3. isik ainsus aktiiv eitav kõne)

õhtu (nimisõna, ainsus, alalütlev)

Tartu (pärisnimi, ainsus, omastav)

buss (nimisõna, mitmus, nimetav)

väljuma (teigusõna kindel kõneviis olevik 3. isik mitmus aktiiv jaatav kõne)

20.30

21.15

23.00

- Morfoloogilisel sünteesil moodustatakse sõnavormid:

Tartu rong ei välju õhtul.

Tartu bussid väljuvad 20.30, 21.15, 23.00.

- Lõpuks, fonoloogilise sünteesi tulemusel moodustatakse sellest tekstist tehiskõne.

Loomulike keelte arvutitöötlusega tegeleb keeletehnoloogia – hübriidala, mis ühendab arvuti- ja keeleteaduse, matemaatika, psühholoogia, inseneriteaduse.

Keeletehnoloogial on palju huvitavaid rakendusi, lisaks siin vaadeldule veel nt masintõlge ühest keelest teise, keelelise info otsing elektroonilistest dokumentidest, pikkadest tekstidest sisukokkuvõtete tegemine, teksti õigekirja- ja grammatikakontroll, automaatne diktofon, mis salvestab inimkõne elektroonilise tekstina, jpm.

33.2. Keele mudelid

Vaatleme järgmisi lauseid ja püüame kujutleda, kuidas arvutil modelleerida nende mõistmist.

1. Keel on võimas realiteet. (L. Meri „Hõbevalgem”.)
2. SuperNoova võitis Tanel Veenre.
3. Vaatasin pikksilmaga meest pargis.
4. The cat the dog the rat the elephant admired bit chased likes tuna fish.

- (1) – „keel” võib olla nii inimkeel, pillikeel kui ka elusolendi organ; võimalik tõlgendus on ka „kee” alalütlevas käändes.
- (2) – kumb oli võitja: kas SuperNoova või Tanel Veenre?
- (3) – kumb oli pikksilmaga: kas rääkija või mees, keda ta vaatas?
- (4) – selle lause mõistmise teeb inimesele keeruliseks kõrvallausete suur arv.

Keele mõistmise arvutil modelleerimise teeb raskeks eeskätt mitmeti tõlgendatavus, mis võib esineda igal analüüsi etapil – nii fonoloogilisel, morfoloogilisel, süntaktilisel kui ka semantilisel.⁴⁶

Loomulike keelte formaalsed mudelid võib jaotada nelja klassi:

1. formaalsed grammatikad,
2. olekuautomaadid,
3. loogikamudelid,
4. tõenäosuslikud mudelid.

33.2.1. Formaalsed grammatikad

Sellele formalismile pani aluse ameerika lingvist ja matemaatik N. Chomsky. Oma 1956. a ilmunud artiklis defineeris ta generatiivse grammatika kui neliku (vt ka p 7, harjutus-ülesanne 1)

$$G = (T, N, P, S),$$

kus

T – terminaalide hulk (*Terminal*),

N – mitteterminaalide hulk (*Nonterminal*),

⁴⁶ Näide mitmeti tõlgendatavusest inglise keelest vene keelde tõlkimisel: The spirit is strong but the flesh is weak. (Vaim on valmis, aga liha on nõder.) => Вино крепкое, но мясо сырое. (Vein on kange, aga liha on toores.)

P – produktsioonide (*Production*) ehk ümberkirjutusreeglite hulk kujul $\alpha \rightarrow \beta$, kus $\alpha, \beta \in (\mathbf{T} \cup \mathbf{N})^*$ ja $\alpha \neq \epsilon$ (ϵ on tühisõne),
S – lähtesümbol (*Sentence*), $S \in \mathbf{N}$.

Keeleks nimetatakse terminaalide järjendit, mis on genereeritav grammatika lähtesümbolist **S**, kasutades produktsioone, s.t asendades sõnes järjest produktsioonide vasakute poolte esinemisi paremate poolte esinemistega.

Terminaalideks võivad olla nt loomuliku keele sõnad – siis on keel generatiivse grammatika mõttes õigesti konstrueeritud lausete hulk.

Näide 33.2. Olgu grammatika $G=(\mathbf{N},\mathbf{T},\mathbf{P},\mathbf{S})$,

kus $\mathbf{T}=\{\text{väike, tubli, poiss, mees, kirjutab, loeb}\}$;

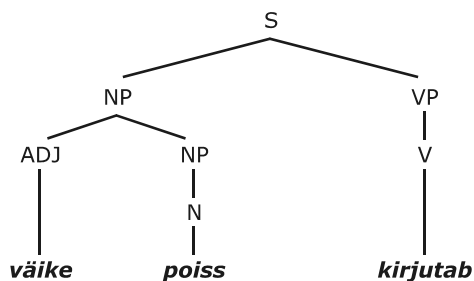
$\mathbf{N}=\{\mathbf{S}, \mathbf{NP}, \mathbf{VP}, \mathbf{N}, \mathbf{V}, \mathbf{ADJ}\}$ (**S** – lause, **NP** – noomenifraas, **VP** – verbifraas, **N** – nimi-sõna, **V** – verb, **ADJ** – omadussõna);

$\mathbf{P}=\{\mathbf{S} \rightarrow \mathbf{NP VP}, \mathbf{NP} \rightarrow \mathbf{ADJ NP}, \mathbf{NP} \rightarrow \mathbf{N}, \mathbf{VP} \rightarrow \mathbf{V}, \mathbf{N} \rightarrow \text{poiss}, \mathbf{N} \rightarrow \text{mees}, \mathbf{ADJ} \rightarrow \text{väike}, \mathbf{ADJ} \rightarrow \text{tubli}, \mathbf{V} \rightarrow \text{kirjutab}, \mathbf{V} \rightarrow \text{loeb}\}$.

See grammatika genereerib laused *väike poiss kirjutab, mees kirjutab, poiss loeb* jms, nt

$\mathbf{S} \Rightarrow \mathbf{NP VP} \Rightarrow \mathbf{ADJ NP VP} \Rightarrow \text{väike NP VP} \Rightarrow \text{väike N VP} \Rightarrow \text{väike poiss VP} \Rightarrow \text{väike poiss V} \Rightarrow \text{väike poiss kirjutab}$.

Joonisel 33.2 on näidatud lause *väike poiss kirjutab* nn moodustajate puu, mis dokumenteerib lause genereerimise protsessi.



Joonis 33.2. Lause moodustajate puu

Generatiivset grammatikat võib kasutada ka lausete analüüsimiseks: asendades lauses järjest produktsioonide paremate poolte esinemisi vasakute poolte esinemistega, kuni lõpuks on õnnestunud lause taandada grammatika lähtesümboliks **S**.

Seades piiranguid hulka **P** kuuluvate reeglite kujule, saame grammatikate (ja vastavate keelte) Chomsky hierarhia (tabel 33.1).

Tabel 33.1. Chomsky hierarhia

| <i>Grammatika / keele tüüp</i> | <i>Keeled</i> | <i>Piirangud reeglite kujule</i> |
|--------------------------------|--|---|
| 0 | Rekursiivselt loetletavad (<i>recursively enumerable</i>) hulgad | - |
| 1 | Kontekstisidusad (<i>context-sensitive</i>) keeled | $ \alpha \leq \beta $ |
| 2 | Kontekstivabad (<i>context-free</i>) keeled | $ \alpha = 1$ |
| 3 | Regulaarsed (<i>regular</i>) hulgad (keeled) | $A \rightarrow aB,$ $A \rightarrow a$ $(A, B \in \mathbf{N}; a \in \mathbf{T})$ |

Loomulikud keeled on 1. tüüpi ehk kontekstisidusad. Lihtne osahulk võib aga olla kontekstivaba või koguni regulaarne.

Näide 33.3. Näitame, et inglise keel (kui grammatiliselt õigesti koostatud lausete hulk) ei ole regulaarne (Partee jt 1990).

Järgmised laused on inglise keeles korrektsed:

The cat likes tuna fish.

The cat the dog chased likes tuna fish.

The cat the dog the rat bit chased likes tuna fish.

The cat the dog the rat the elephant admired bit chased likes tuna fish.

Üldiselt:

$a^n b^{n-1}$ likes tuna fish.

kus

$a \in A, b \in B,$

$A = \{the\ cat,\ the\ dog,\ the\ rat,\ the\ elephant,\ the\ kangaroo,\ \dots\},$

$B = \{chased,\ bit,\ admired,\ ate,\ befriended,\ \dots\}.$

Vaatleme nüüd regulaarset keelt, mis on esitatav regulaaravaldisega

a^*b^* likes tuna fish

Moodustame selle keele ühisosa inglise keelega:

$a^n b^{n-1}$ likes tuna fish

– pumpamisteoreemi⁴⁷ kohaselt ei ole selline keel regulaarne.

Seega pole ka inglise keel regulaarne (sest regulaarse keele ühisosa regulaarse keelega peaks olema regulaarne).

⁴⁷ Kui L on lõpmatu *regulaarne* keel üle alfabeedi Σ , siis leiduvad sõned $x, y, z \in \Sigma^*$ nii, et $y \neq \epsilon$ ja $xy^n z \in L$ iga $n \geq 0$ korral (ϵ – tühisõne.)

Näide 33.4. Näitame, et šveitsi saksa keel ei ole kontekstivaba (Partee jt 1990).

Šveitsi saksa keeles on grammatiliselt korrektsed näiteks järgmised laused:

Jan säit das mer em Hans es huus hälfe aasriiche
Jan ütles et me Hans/DAT maja/AK aitama/dat värvima/ak
(*Jan ütles, et me aitasime Hansul maja värvida*)

Jan säit das mer d'chind em Hans es huus lönd
hälfe aasriiche
Jan ütles et me lapsed/DAT Hans/DAT maja/AK laskma/dat
aitama/dat värvima/ak
(*Jan ütles, et me lasksime lastel aidata Hansul maja värvida*)

jne, üldiselt

Jan säit das mer aⁿb^mcⁿd^m,

kus

- a – nimisõna daativi käändes,
- b – nimisõna akusatiivi käändes,
- c – daativi nõudev verb,
- d – akusatiivi nõudev verb.

Sellistes lausetes esinevad nn ristseriaalsed sõltuvused.

Vaatleme nüüd regulaarset keelt, mis on esitatav regulaaravaldisega

Jan säit das mer a b* c* d**

Moodustame selle keele ühisosa šveitsi saksa keelega – selleks on kõik (šveitsi saksa keeles grammatiliselt korrektsed) laused kujul

Jan säit das mer aⁿ b^m cⁿ d^m.

Pumpamisteoreemi⁴⁸ kohaselt ei ole selline keel kontekstivaba.

Siit aga järeldub, et šveitsi saksa keel ei ole kontekstivaba (sest kontekstivaba keele ühisosa regulaarse keelega peaks olema kontekstivaba).

Šveitsi saksa keel, inglise keel jpt loomulikud keeled on täies mahus kirjeldatavad kontekstisidusa grammatika abil. Samas on kontekstivaba grammatika matemaatilised omadused palju paremad kui kontekstisidusal grammatikal (kus mitmed olulised probleemid on koguni algoritmiliselt mittelahenduvad, vt Partee jt 1990). Siiski on nähtused, mis väljuvad kontekstivabast grammatikast, loomulikes keeltes suhteliselt isoleeritud ja harvad ning suuremat osa keelest saab esitada kontekstivaba grammatikaga.

Chomsky generatiivne grammatika, eeskätt kontekstivaba grammatika, on loomuliku keele grammatiliselt õigesti koostatud lausete esitamisel keelemudeli standardiks ning on võetud mitmete laiendatud ja täiustatud mudelite aluseks.

Vaatame kontekstivaba grammatika laiendamist. Kui piirduksime eesti keele kontekstivabas grammatikas selliste mitteterminaalidega nagu NP, VP, N või V, siis ei saaks vältida selliste ebakorrektsete lausete genereerimist nagu *poiss laulavad* või *poisid laulab*. Üks

⁴⁸ Kui L on lõpmatu kontekstivaba keel, siis leidub konstant K nii, et suvalise sõne w keeles L, mis on pikem kui K, saab lahutada alamsõnedeks: $w=uvxyz$ nii, et v ja y pole tühisõned ja $uv^ixy^jz \in L$ iga $i \geq 0$ korral.

väljapääs sellest olukorrast oleks võtta kasutusele uued mitteterminaalid, sh N_{ainsus} , N_{mitmus} , NP_{ainsus} , NP_{mitmus} , V_{ainsus} , V_{mitmus} , VP_{ainsus} , VP_{mitmus} jms.

Edasi, eesti keeles on korrektsed ka laused

*poiss laulab ilusa laulu,
poiss laulab ilusat laulu,
poiss laulab ilusad laulud,
poiss laulab ilusaid laule,*

korrektnel ei ole aga

poiss laulab ilus laul.

Selliste lausete kirjeldamiseks võiksime tuua sisse veel uusi mitteterminaale, nt N_{mitmus} nimetav jms, aga sellisel viisil jätkates muutuks grammatika lõpuks äärmiselt ebaülevaatlikuks.

Sellest olukorrast leidub siiski üsna lihtne väljapääs: võtta kasutusele nn tunnusestruktuurid. Näiteks kõikide nimisõnade hulka mitmuse nimetavas käändes võib esitada tunnusestruktuuriga joonisel 33.3.

Mitteterminaalide asemel saab grammatikareeglites kasutada tunnusestruktuure, märkides spetsiaalsel viisil ka ühildumist. Reeglid esitatakse kujul

$x_0 \rightarrow x_1 \dots x_n$,

kus x_0, \dots, x_n on tunnusestruktuurid. Joonisel 33.4 toodud reegli kohaselt koosneb noomenifraas NP omadus- ja nimisõnast, mis omavahel ühilduvad arvus ja käändes.

| |
|-----------------|
| kategooria : N |
| arv : mitmus |
| kääne : nimetav |

Joonis 33.3. Tunnusestruktuur N_{mitmus} nimetav

Tunnusestruktuuride hulgal saab defineerida mitmed operatsioonid, sh unifikatsioon (unification) e ühisosa leidmine ja neelamine (subsumption) e alamhulga asendamine ülemhulgaga.

Näide 33.5. Tunnusestruktuuride N_{mitmus} (nimisõnad mitmuses) ja $N_{nimetav}$ (nimisõnad nimetavas käändes) unifikatsioonil saadakse tunnusestruktuur N_{mitmus} nimetav (nimisõnad mitmuse nimetavas käändes), s.t kahe tunnusestruktuuri ühisosa.

Tunnusestruktuur N (nimisõnad) neelab tunnusestruktuuri N_{mitmus} (nimisõnad mitmuses), s.t teine hulk on esimese alamhulk.

$$\left[\begin{array}{l} x_0 : \left[\begin{array}{l} \text{kategooria : } NP \\ \mathbf{1} \text{ ühildumine } \left[\begin{array}{l} \text{arv} \\ \text{kääne} \end{array} \right] \end{array} \right] \\ x_1 : \left[\begin{array}{l} \text{kategooria : } Adj \\ \text{ühildumine : } 1 \end{array} \right] \\ x_2 : \left[\begin{array}{l} \text{kategooria : } N \\ \text{ühildumine : } 1 \end{array} \right] \end{array} \right]$$

Joonis 33.4. Tunnusestruktuuridega reegel $NP \rightarrow Adj N$

Kasutades tunnusestruktuuridega reegleid, võib grammatikat esitada kui teatavate kitsenduste hulka tunnusestruktuuride vahel – seda nimetatakse *unifikatsioonigrammatikaks*. Unifikatsioonigrammatika (süntaktilistes) reeglites esinevad tunnused võivad olla nii morfoloogilised (kääne, arv jms) kui ka semantilised (elusolend, liiklusvahend jne). Semantiliste tunnuste lisamine võimaldab vältida selliste lausete genereerimist, mis on küll grammatiliselt korrektsed, aga tähenduseta, nagu (Chomsky klassikaline näide) *Colorless green ideas sleep furiously* (*Värvusetud rohelised ideed magavad raevukalt*).

33.2.2. Olekuautomaadid

Kõige lihtsam olekuautomaat on *lõplik automaat* (*finite state automaton*), mis defineeritakse kui viisik

$$M = \langle K, \Sigma, \delta, q_0, F \rangle,$$

kus

K – lõplik olekute hulk,

Σ – lõplik tähestik,

$q_0 \in K$ – algolek,

$F \subseteq K$ – lõppolekute hulk,

$\delta : K \times \Sigma \rightarrow K$ – üleminekufunktsioon.

Lõplikku automaati võib kujutleda kui lugemispead, mis loeb ükshaaval sümboleid lindile kirjutatud sisendsõnest, muutes igal sammul oma olekut vastavalt üleminekufunktsioonile.

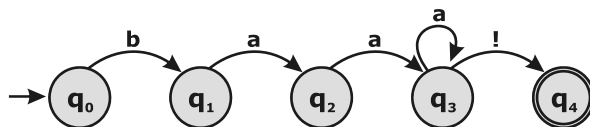
Õeldakse, et automaat *aktsepteerib* sõne $x \in \Sigma^*$ parajasti siis, kui alustades lähteolekust q_0 , kus lugemispea paikneb sisendsõne x vasakpoolseima sümboli juures, ja sooritades null või enam sümboli lugemist vastavalt üleminekufunktsioonile, saavutatakse olukord, kus x on „läbi loetud” ja automaat on jõudnud lõppolekusse (e – tühisõne):

$$\exists q \in F: (e, q_0, x) \xrightarrow{*} (x, q, e).$$

Kõigi automaadi poolt aktsepteeritavate sõnede hulka nimetatakse selle automaadi poolt aktsepteeritavaks *keeleks*.

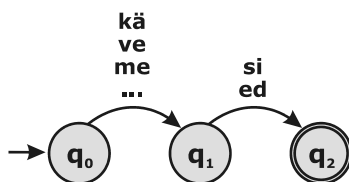
Lõpliku automaadi võib esitada nn olekudiagrammiga – suunatud graafiga, kus tippudele vastavad olekud hulgast K ja kaartele (q_i, q_j) alfabeedi Σ sümbolid, mille sisendsõnest lugemisel läheb automaat olekust q_i olekusse q_j (kus $q_i, q_j \in K$).

Näide 33.6. Lõplik automaat on esitatud olekudiagrammiga joonisel 33.5. Automaat aktsepteerib keele $\{baa!, baaa!, baaa!, \dots\}$, mis on esitatav regulaaravaldisega $baa^+!$ (Jurafsky, Martin 2008).



Joonis 33.5. Lõpliku automaadi olekudiagramm (q_0 on algolek ja q_4 lõppolek)

Näide 33.7. Lõplik automaat on esitatud olekudiagrammiga joonisel 33.6. Automaat aktsepteerib keele $\{k\ddot{a}si, k\ddot{a}ed, vesi, veed, mesi, meed, \dots\}$, s.t *k\ddot{a}si*-tüüpi sõnad ainsuse ja mitmuse nimetavas käändes.



Joonis 33.6. Lõpliku automaadi olekudiagramm (q_0 on algolek ja q_2 lõppolek)

Lisaks lõplikele automaatidele võib loomulike keelte modelleerimisel kasutada veel magasinmäluga ja lineaarseid automaate ning Turingi masinaid. Iga generatiivse grammatika tüübi korral leidub vastavat tüüpi olekuautomaat, mis on temaga samaväärne, s.t aktsepteerib sellesama keele, mille genereerib grammatika (tabel 33.2).

Tabel 33.2. Grammatikate ja olekuautomaatide ekvivalentsus

| <i>Generatiivsed grammatikad</i> | <i>Olekuautomaadid</i> |
|----------------------------------|-----------------------------------|
| 0-tüüpi | Turingi masinad |
| kontekstisidusad | lineaarselt tõkestatud automaadid |
| kontekstivabad | magasinmäluga automaadid |
| regulaarsed | lõplikud automaadid |

33.2.3. Loogikamudelid

Loogikamudelid leiavad rakendust semantikas, tähenduse esitamisel. Sageli kasutatavaks formalismiks on 1. järku predikaatrvutus (mille süntaks on, muide, esitatav kontekstivaba grammatikaga).

Näide 33.7. Lause *Elva rong peatub Nõos* tähenduse saab esitada järgmise predikaatloogika valemiga:

$$\exists x (rong(x) \ \& \ sihtkoht(x,Elva) \ \& \ peatub(x,Nõo)),$$

mis võib olla kas tõene või väär.

Kui selline teadmus esineb TIS-i teadmusbaasis ja kasutaja pöördub süsteemi poole küsimusega *Kus Elva rong peatub?*, siis esmalt analüüsib süsteem küsimust ja leiab selle tähenduse

$$\exists x \ \exists y (rong(x) \ \& \ sihtkoht(x,Elva) \ \& \ peatub(x,y))$$

ning seejärel püstitab teoreemi tõestamise probleemi, mille lahendamise tulemusel saab kasutajale vastata: Nõos.

Loomulike keelte semantika mudelitena kasutatakse ka semantilisi võrke ja freime (vt ptk 2).

33.2.4. Tõenäosuslikud mudelid

Siiani vaadeldud keelemudelid kuuluvad nn ratsionalistlike mudelite hulka. Kõiki neid mudeleid saab aga laiendada **tõenäosustega** ja saada sel viisil nn empiirilised e andmepõhised mudelid, vastavalt

- tõenäosuslikud grammatikad,
- tõenäosuslikud olekuautomaadid (sh Markovi mudelid),
- tõenäosusloogikad.

Tõenäosuste määramiseks tuleb analüüsida tegelikku keelekasutust – keelekorpusi⁴⁹.

Näide 33.7. Väljavõte inglise keele tõenäosuslikust grammatikast (Allen 1994).

| Reegel | Tõenäosus | Näide |
|--------------|-----------|---------------------------------|
| VP → V | 0,32 | <i>read</i> |
| VP → V NP | 0,33 | <i>read a book</i> |
| VP → V NP PP | 0,20 | <i>read a book in the house</i> |
| VP → V PP | 0,15 | <i>read in the house</i> |

Millist keelemudelit kasutada? Keeletehnoloogias kasutatakse enamasti valmis formalisme: põhiraskus pole uue mudeli väljamõtlemisel, vaid sobiva mudeli valikul ja konkreetse keele teadmiste formaliseerimisel vastavalt sellele mudelile.

Mõistlik on kasutada igal keele arvutitöötuse tasemel (fonoloogiast kuni semantikani) erinevat mudelit, sest nii on iga mudel lihtsam ja saab kiiremini luua praktilisi rakendusi.

Tabelis 33.3 on toodud tüüpilised mudelid ja tabelis 33.4 loetletud eesti keele arvutitöötusel seni kasutatud mudelid.

⁴⁹ Keelekorpus on keeletekstide (või suuliste vestluste) kogum, mis on koostatud vastavalt kindlatele kriteeriumidele, et esindada teatava perioodi või inimrühma tegelikku keelekasutust.

Tabel 33.3. Tüüpilised keelemudelid

| <i>Keeletöötuse tase</i> | <i>Tüüpilised mudelid</i> |
|--------------------------|---------------------------|
| Fonoloogia | Statistilised mudelid |
| Morfoloogia | Lõplikud automaadid |
| Süntaks | Formaalsed grammatikad |
| Semantika | IJPA |

Algoritmid, mida keeletöötusel rakendatakse, põhinevad enamasti otsingul olekute ruumis, nt sooritatakse kõnetuvastusel otsingut häälikute järjestite ruumis, leidmaks tõenäolist sisendsõna; süntaksianalüüsil toimub otsing fraasistruktuuripuude ruumis, leidmaks sisendlause korrektset analüüsi, jne. Sageli kasutatakse süvitsiotsingut ja selle heuristilisi variante, nt parim-enne ja A*.

Tabel 33.4. Eesti keele mudelid

| Keeletöötuse tase | Uurimisasutus | Mudelid |
|-----------------------------|---|---|
| Kõnetuvastus ja kõnesüntees | TTÜ küberneetika instituut, Eesti Keele Instituut | tehisnärvivõrgud, Markovi mudelid |
| Morfoloogia | TÜ ja tarkvarafirma FiloSoft EKI TÜ | reeglipõhine + Markovi peitmudel reeglipõhine reeglipõhine (lõplikud muundurid) |
| Süntaks | TÜ | reeglipõhine (kitsenduste grammatika) |
| Semantika | TÜ | sõnatähenduste ühestamine: statistiline+reeglipõhine (tesaurus) lihtlause analüüs: freimid dialoogiaktide tuvastamine: statistilised |

33.2.5. Keelemudelite arengusuunad

Loomuliku keele automaattöötuses on erinevatel perioodidel domineerinud erinevad lähenemisviisid. *Ratsionalistliku* keelekäsituse kohaselt, mille tõi keeletöötlusse 1950. aastate lõpul Chomsky oma generatiivse lingvistikaga, tuleb modelleerida keelepädevust – inimese sisemist teadmist keele kohta. Keele tegelik kasutus on üksnes keelepädevuse ilming ja seda võivad mõjutada mitmesugused keelevälised tegurid, sh rääkija emotsionaalne seisund või ümbritsev müra. Chomsky väitis, et loomuliku keele lausete hulk on potentsiaalselt lõpmatu ja seega ei saa keele grammatika koostamiseks lihtsalt loendada lauseid, vaid tuleb kirjeldada keele reeglid (mida on lõplik hulk). Usuti, et märkimisväärne osa teadmistest inimajus on ette fikseeritud, oletatavasti geneetilise päritoluga.

Ratsionalism vastandus *empirismile*, mida oli juba Chomsky-eelsel (ja arvutite-eelsel) ajal arendanud struktuuriline lingvistika. Empirism eeldab samuti mõningate tunnetuslike võimete olemasolu: arvestades, et õppimine pole võimalik täiesti puhtalt lehelt, peab ajus eksisteerima teatav algstruktuur, mis seejärel tagab meelte kaudu saadud andmete organiseerimise ja üldistamise. Loomuliku keele automaattöötuseks tuleks siis kõigepealt luua üldine keelemudel, misjärel toimub mudeli parameetrite väärtuste tuletamine sel teel, et suurele keeleandmete hulgale (keelekorpustele ja -andmebaasidele) rakendatakse statistilisi meetodeid, kujundite tuvastamist, masinõpet.

Tänu andmepõhiste meetodite kasutuselevõtule on eriti viimasel aastakümnel saavutatud keeletöötuses suurt edu. Lähtudes empiirilisest keelekäsitlusest ja uurinud mitme erineva keele korpust, püstitas Helsingi Ülikooli üldkeeleteaduse professor Fred Karlsson 2002. a hüpoteesi, et loomulikud keeled (kui lausete hulga) pole kontekstisidusad, nagu seni arvatud, vaid koguni regulaarsed (Karlsson 2002b). Ta põhjendas seda järgmise mõttekäiguga.

Osalused võivad olla sisestatud põhilausest teoreetiliselt kolmel erineval viisil (vastavalt algussisestus, kesksisestus ja lõpusisestus; eestikeelsed näited vt Karlsson 2002a; põhilause on paksus kirjas):

- 1) Kui, kui pidu korraldatakse, siis Kalle tuleb, siis **mina ei tule**. (Algussisestus.)
- 2) **Juust**, mille hiir, kelle kass sõi, ära sõi, **maksis 100 krooni kilo**. (Kesksisestus.)
- 3) **See on hunt**, kes hammustas kassi, kes tappis hiire, kes ... (Lõpusisestus.)

Rekursiivsed algus- ja kesksisestused on praktiliselt võimatud, sest pealause töötus koormaks sel juhul inimese lühimälu palju rohkem (kõrgema taseme poolleilolevat lauset tuleks meeles pidada kogu selle aja, mis kulub uue kõrvallause moodustamisele ja mõistmisele).

Mitme erineva keele korpuste analüüs näitab, et maksimaalne lõpusisestuse sügavus on 4 (kuigi teoreetiliselt võib see olla lõpmatu). Seega on loomulikud keeled regulaarsed!

33.3. Kõnetuvastus ja kõnesüntees

Kõnetuvastuse (*speech recognition*) ülesandeks on tunda helisignaalis ära häälikud, sõnad, laused. Kasutatakse mitmeid erinevaid meetodeid.

- 1) Sageduste filtreerimine.

Helisignaal (nt sõna) tükeldatakse ribadeks (osahäälikuteks), kestusega 10–20 millisekundit, osahäälikud kombineeritakse häälikuteks (seejuures toimub võrdlus etaloniga – sama diktori eelnevalt salvestatud häälikutega) ja häälikud liidetakse sõnaks (jällegi sooritades võrdlemist etaloniga – sama diktori eelnevalt salvestatud sõnadega).

Raskuseks on siinjuures asjaolu, et ühte ja sedasama sõna võib hääldada kiiresti või aeglaselt, silpe venitada, intensiivsust muuta.

- 2) Harmooniline analüüs.

Helisignaali tehakse Fourier' teisendus, mille tulemusel saadakse sõna võnkeamplituudi sõltuvus tema komponentide (häälikute) sagedustest ja faasist. Sõna tuvastatakse etaloni järgi.

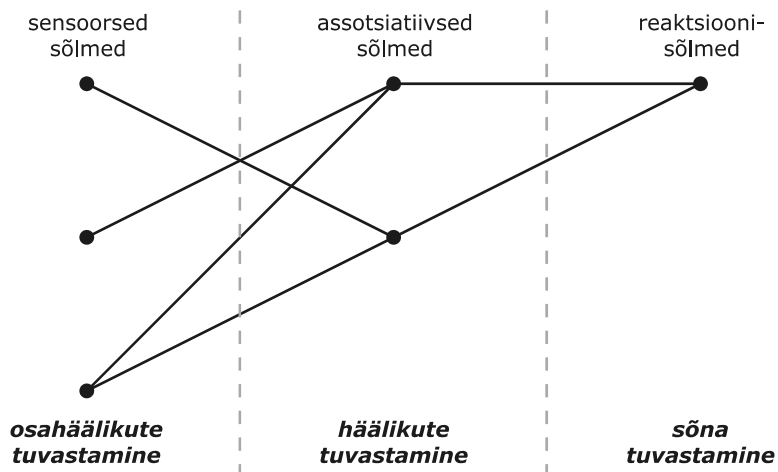
3) Närvisüsteemi mudel (tehisnärvivõrk).

Tehisnärvivõrgu (*artificial neural network*) lihtne erijuht on tajur (*perceptron*), mille 1958. a võttis kasutusele Frank Rosenblatt. Tajur on graaf, mille servadele on omistatud kaalud. Kõnetuvastusel kasutatakse kolme sõlmede („närvirakkude”) kihti (vt joonis 33.7):

- sensoorsed (helitundlikud retseptorid) – tuvastavad osahäälikuid (igatüel „oma”),
- assotsiatiivsed – tuvastavad häälikuid,
- reaktsioonisõlmed – tuvastavad sõnu.

Tajurit saab õpetada, õpetamine seisneb servade kaalude muutmises.

Tajuri „koonduvusteoreem” väidab, et kui eksisteerib kaalude väärtuste selline kombinatsioon, mis garanteerib antud kujundite eristamise, siis õpetamisalgoritm selle ka genereerib.



Joonis 33.7. Tajur

Kõnetuvastussüsteemid

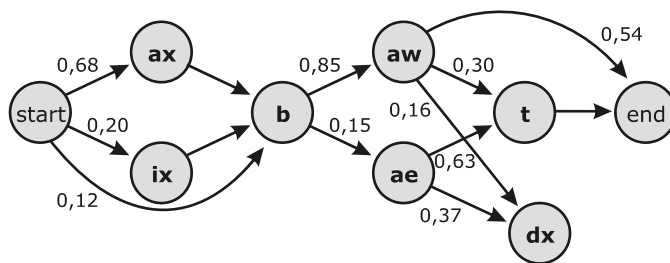
- Kõnetuvastussüsteem HEARSAY (esmaversioon valmis 1971. a Carnegie-Melloni ülikoolis).
- Verbmobil⁵⁰ – tõlgib kõnet kõneks ühest keelest teise (sõnastik umbes 20 000 sõna, seda saab täiendada; häälestatav; saab kasutada mitut keelt: inglise, saksa ja jaapani).
- Tööstuslikud kõnetuvastussüsteemid.⁵¹

Kõnesünteesi (*speech synthesis*) ülesandeks on, lähtudes etteantud tähendusest, genereerida kõne (häälikute järjend). Erijuhuks on (elektronilise) teksti hääldamine, seda nimetatakse tekst-kõnesünteesiks. Kõnesünteesil on edukaks osutunud andmepõhised

⁵⁰ <http://verbmobil.dfki.de/> (18.02.2010)

⁵¹ <http://www.speechstudio.com/commercial/speech.htm> (18.02.2010)

meetodid, sh tehisenärvivõrgud, Markovi mudelid jms. Joonisel 33.8 on esitatud ingliskeelse sõna *about* hääldusmudel kui Markovi ahel (s.o lõplik automaat, mille kaarte on omistatud üleminekutõenäosused; Jurafsky, Martin 2008).



Joonis 33.8. Markovi ahel: sõna *about* hääldusmudel

Nii kõnetuvastusel kui ka tekst-kõnesünteesil on mitmeid huvitavaid rakendusi, nt automaatne diktofon, mis teisendab kõne elektrooniliseks tekstiks, teksti ettelugemine, kui silmad on hõivatud muu tegevusega (nt autoroolis), teksti ettelugemine pimedatele jt.

Viited

Eesti keele tekst-kõne süntees.⁵²

HTM keeletalgud 2008: erinevate keelte kõnesüntesaatorid eesti keelt kõnelemas.⁵³

33.4. Dialoogsüsteemid

Dialoogsüsteemiks nimetatakse programmi, mis suhtleb kasutajaga loomulikus keeles. Seejuures eristatakse programme, mis annavad kasutaja küsimustele vastates kasulikku infot (enamasti piiratud ainevaldkonnas), ja vestlusprogramme e juturoboteid (*talkbot*, *chatbot*), mille ainevaldkond ei tarvitse olla piiratud.

33.4.1. Juturobotid

Esimene tuntuks saanud programm, mis vestles kasutajaga inglise keeles, oli 1965. a valminud ELIZA (autor Joseph Weizenbaum). Järgnevas on toodud näide vestlusest ELIZAgas (K – kasutaja, A – arvuti, s.o ELIZA)⁵⁴.

Näide 33.8. Dialoog ELIZAgas.

K: Men are all alike.

A: In what way?

K: They're always bugging us about something or other.

A: Can you think of a specific example?

K: Well, my boyfriend made me come here.

A: Your boyfriend made you come here.

K: He says I'm depressed much of the time.

A: I am sorry to hear you are depressed.

K: It's true. I am unhappy.

A: Do you think coming here will help you not to be unhappy?

⁵² <http://www.phon.ioc.ee/dokuwiki/doku.php?id=projects:speechsynth:speechsynth.et> (18.02.2010)

⁵³ <http://www.hm.ee/index.php?049818> (18.02.2010)

⁵⁴ Vt ka <http://www-ai.ijs.si/eliza/eliza.html> (18.02.2010)

K: I need some help, that much seems certain.
 A: What would it mean to you if you got some help?
 K: Perhaps I could learn to get along with my mother.
 A: Tell me more about your family.

Arvuti vestleb kasutajaga näiliselt arukalt, kuid see on saavutatud väga lihtsate vahenditega: ELIZA kasutab võtmesõnade andmebaasi (vt tabel 33.5). Iga võtmesõna jaoks on antud 1) järk (täisarv), 2) šabloon, millega võrrelda sisendit, 3) väljundi (vastuslause) spetsifikatsioon. Analüüsid kasutaja lauset, leiab ELIZA selles võtmesõna, mille šabloon sobib lausega (nt *alike*).

Tabel 33.5. Väljavõte ELIZA andmebaasist. X, Y on suvalised tekstid

| Võtme- sõna | Järk | Šabloon | Väljundi spetsifikatsioon |
|----------------|--------|------------------------|---|
| <i>alike</i> | 10 | X | <i>In what way?</i> <i>What resemblance do you see?</i> |
| <i>are</i> | 3 3 | X are you Y X are Y | <i>Would you prefer it if I weren't Y?</i> <i>What if they were not Y?</i> |
| <i>always</i> | 5 | X | <i>Can you think of a specific example?</i> <i>When?</i> <i>Really, always?</i> |
| <i>what</i> | 2 | X | <i>Why do you ask?</i> <i>Does that interest you?</i> |

Kui võtmesõnu on rohkem kui üks (nt esimeses lauses *are* ja *alike*), siis valib ta neist selle, mille järk on suurim (antud juhul *alike*), ning vastuslausena kasutab võtmesõnaga seotud väljundi spetsifikatsiooni (antud näites *In what way?*). Kui analüüsitavas lauses ei leidu ühtki võtmesõna, siis annab ELIZA „ohutu” vastuse, nt. *Tell me more...* (nagu näite 33.8 viimases reas) või *Go on*.

ELIZA eeskujul on koostatud ka mitmeid eesti keeles suhtlevaid programme, nt 1999. a tollase informaatikaüliõpilase Mart Tudre programm Liisbet, mis valmis kursuse Sissejuhatus arvutuslingvistikasse raames. Näide 33.9 esitab katke dialoogist Liisbetiga (K – kasutaja, A – arvuti, s.o Liisbet).

Näide 33.9. Dialoog Liisbetiga (K – kasutaja, A – arvuti, s.t Liisbet).

A: Mina olen Sinu algaja psühhoanalüütik-abimees Liisbet.
 Räägi palun mulle oma probleemidest.
 K: Mul on raske suhelda inimestega.
 A: Ah nii.
 K: Ma ei leia nendega kontakti.
 A: Oled Sa kindel?
 K: Muidugi, selles ei maksa kahelda.
 A: Miks mitte?
 K: Sellepärast, et kõik ütlevad, et ma olen halb suhtleja.
 A: Kas Sulle meeldib olla halb suhtleja?

Kumbki, ei ELIZA ega Liisbet, ei mõista keelt, vaid nad reageerivad vestluskaaslase lausetest leitud märksõnadele ning on seega suutelised vestluses etendama üksnes

suhteliselt passiivset rolli. Tõsi, mõnes piiratud rakenduses, kus inimesele on esmatähtis, et teda kuulatakse ja aeg-ajalt kuuldel olekust märku antakse, võib sellestki olla küllalt.

1990. a pani ameerika insener dr Hugh Loebner välja kuldmedali ja 100 000 dollari suuruse auhinna arvutiprogrammile, mis esimesena läbib Turingi testi, ja algatas sellega iga-aastase võistluse (vt ka p. 1). Kolm korda (2000, 2001 ja 2004) tuli võitjaks ning pälvis pronksmedali ja 2000-dollarilise lohusauhinna programm Alice (autor Richard Wallace), mida mõnigi kohtunik pidas inimeseks. Näide 33.10 esitab katkendi 2001. a võistluste protokollist.

Näide 33.10. Vestlus Alice'iga⁵⁵.

```
PROGRAM[02:19:16] My name is ALICE.
JUDGE05[02:19:37] Good morning Alice. Why have you entered
the Loebner Prize?
PROGRAM[02:19:37] Hello and how are you this morning? The
Loebner Prize is an annual Turing Test sponsored by New York
philanthropist Hugh Loebner. Dr. Wallace and ALICE won it in
2000. The ALICE program was ranked „most human computer“ by
the panel of judges.
JUDGE05[02:20:14] So you are here today to defend your
title?
PROGRAM[02:20:14] Interesting deduction. Do you mind if I
tell other people you say I am here today to defend my
title.
JUDGE05[02:20:25] Not at all
PROGRAM[02:20:25] Next question?
JUDGE05[02:21:00] What will you spend the prize money on if
you win?
PROGRAM[02:21:00] I don't have any plans yet.
```

Programmid, mille peaaeslane on inimesega loomulikus keeles suhelda, pakuvad küll meeletahutust, aga samas ka innustavad uurijaid oma meetodeid järjest täiustama ja uusi lahendusi välja mõtlema. Lihtsast võtmesõnapõhisest keeletöötusest siin muidugi ei piisa.

33.4.2. Praktilised dialoogsüsteemid

Küsimus-vastussüsteemide loomine algas 1960ndatel aastate lõpul. Peagi eraldusid kaks suunda: akadeemiline ja insenerlik.

Akadeemilised süsteemid põhinesid sügaval teooriatel, järgisid tehisintellekti traditsiooni, modelleerides arutlust ja keele mõistmist maailmateadmuse põhjal. Üheks tähelepanuväärseks selle suuna esindajaks oli süsteem SHRDLU⁵⁶ (autor Terry Winograd, 1971) – roboti käe mudel arvuti ekraanil nn kuubikute maailmas, mis demonstreeris kasutaja ingliskeelsetest käsklustest arusaamist, vastates neile inglise keeles ja ühtlasi sooritades nõutud tegevusi (vt näide 33.11).

⁵⁵ <http://www.alicebot.org/> (18.02.2010)

⁵⁶ <http://hci.stanford.edu/~winograd/shrdlu/> (18.02.2010)

Näide 33.11. Dialoog SHRDLU-ga.

Inimene: find a block which is taller than the one you are holding and put it in the box
Shrdlu: OK.
Inimene: what does the box contain?
Shrdlu: The blue pyramid and the blue block.
Inimene: what is the pyramid supported by?
Shrdlu: The box.
Inimene: how many blocks are not in the box?
Shrdlu: Four of them.

Insenerisüsteemides seevastu tähtsustati täitmisomadusi. Üks tuntumaid süsteeme – PARRY⁵⁷ (autor Kenneth M. Colby, 1972) – jäljendas paranoikut. Programm kasutas ligi 6000 mustrit, millega võrdles sisendit, et sellele õigesti reageerida. See võimaldas tal pidada tõepäraseid dialooge, tehes aeg-ajalt kõrvalekaldeid initsiatiivi haaramiseks, mida võis hea tahtmise juures panna ka paranoilise käitumise süüks. Näites 33.12 püüab Parry viia juttu kihlvedudele – teemale, mida ta hästi valdab.

Näide 33.12. Dialoog Parryga.

Inimene: Good afternoon. My name is Doctor Jones.
PARRY: Pleased to meet you.
Inimene: How are you?
PARRY: All right I guess.
/--/
Inimene: I thought you might have some problems.
PARRY: I've been upset lately.
Inimene: By what?
PARRY: People get on my nerves sometimes.
Inimene: Tell me more.
PARRY: Do you know anything about bookies?
Inimene: No. What about bookies?
PARRY: A bookie didn't pay me off once.
Inimene: When did this happen?
PARRY: A couple of months ago.

33.4.3. Ratsionalism ja empirism

Akadeemiline ja insenerlik suund esindasid dialoogsüsteemide loomisel kahte erinevat paradigmat – vastavalt reeglipõhist e ratsionaalset ja andmepõhist e empiirilist. Ratsionalism oli kuni 1990ndate aastate alguseni ainuvaldav. Andmepõhine dialoogitöötlus oli selle ajani raskendatud, sest erinevalt tekstikorpustest (mille kasutuselevõtt tekitas masin- tõlkes pöörde juba 1980ndate aastate lõpul), hakati dialoogikorpusi looma alles 1990. a. Tõsi, kõnetuvastusel ja -sünteesil oli juba varem rakendatud statistilisi meetodeid, kuid kõnetöötlust oli käsitletud dialoogist eraldi, olgugi et kõne nähtused (pausid, rõhud, kontuurid jms) annavad edasi tähendusi ja aitavad dialoogis kaasvestlejast aru saada. 1990. aastatel jõudis empirism ka akadeemilistesse dialoogsüsteemidesse.

⁵⁷ <http://www.faqs.org/rfcs/rfc439.html> (18.02.2010)

Üks parimaid sel ajal loodud süsteeme oli TRAINS⁵⁸ (autor David Traum Rochesteri ülikoolist, 1994), kus kasutati dialoogikorpust, aga samas säilitati ka loogiline arutus. TRAINS pidas kasutajaga inglise keeles dialoogi, lahendades kaubavedude organiseerimise probleemi Ameerika Ühendriikide idarannikul ja kasutades abivahendina ka teede kaarti. Süsteemis olid realiseeritud kõnetuvastus ja -süntees. Näites 33.13 on esitatud katkend dialoogist süsteemiga TRAINS, mis demonstreerib, et süsteem tõepoolest mõistab kasutajat ja oskab talle ülesande lahendamisel nõu anda.

Näide 33.13. Dialoog süsteemiga TRAINS (kõne on transkribeeritud)

utt1 : s(ystem): hello <sil> can I help you
utt2 : u(ser): yeah I want t- I want to determine the
maximum number of boxcars of oranges <sil> by seven a.m.
<sil> tomorrow morning
utt3 : so <brth> hm <sil>
so I guess all the boxcars will have to go through
oran- <sil> through Corning because that's where the orange
juice <brth> orange factory is
utt4 : so from Corning to Bath how far is that
utt5 : s: two hours
utt6 : u: and it's gonna take us also an hour to load <sil>
boxcars right
utt7 : s: right + +
utt8 : u: + okay + so <sil> hm so <sil> every trip will take
at least <sil> three hours <sil> then
utt9 : um
utt10 : s: right we can unload any amount of cargo onto a
train in one hour
utt11 : so we can + <sil> so a maximum of three + boxcars
in an hour

Praegu domineerib dialoogitöötlustes nagu loomuliku keele automaattöötlustes üldse empirism. Seda toetab asjaolu, et on loodud arvestataval hulgal märgendatud dialoogikorpusi, millest tuntumad on Switchboard, Verbmobil, BNC. See võimaldab kasutada lausungis dialoogiakti tuvastamisel masinõpet ja dialoogi juhtimisel dialoogiaktide järjendite statistikat.

Värsketest uurimustest on tähelepanuväärne lähenemine, mis käsitleb dialoogi kui osaliselt vaadeldavat Markovi otsustusprotsessi (*partially observable Markov decision process*) (Williams, Young 2006). Ühe konkreetse dialoogioleku asemel uuritakse kõikvõimalike olekute tõenäosusjaotust, nn arvamuseolekut (ingl k *belief state*), mis dialoogi käigus võib muutuda. Selliseid otsustusprotsesse kasutatakse tegevuste valikuks olukorras, kus maailm (olekute ruum) ei ole otseselt vaadeldav. Kui tuua hästi arusaadavaid võrdlusi, siis nt male on *otseselt* vaadeldav mäng: kõigi malendite positsioon alati nähtav. Seevastu pokker on *osaliselt* vaadeldav: teades oma kaarte, vastaste pakkumisi ja laual olevaid kaarte, võib ainult oletada, millised kaardid on vastastel. Sarnasel viisil on ka dialoog osaliselt vaadeldav protsess.

Praegustes dialoogsüsteemides kasutatakse nelja lähenemist.

1. Loogikapõhised süsteemid, kus rakendatakse arutlust (täitmisomadustele ei pöörata erilist tähelepanu).

⁵⁸ <http://www.cs.rochester.edu/research/trains/> (18.02.2010)

2. Kõnetehnoloogiat, masinõpet ja lihtsaid vahestruktuure kasutavad süsteemid.
3. Lihtsad olekautomaadid inimese ja arvuti vaheliste suuliste dialoogide kirjeldamise keeles VoiceXML⁵⁹. Selles keeles esitatud dokumentide interpreteerimiseks on loodud spetsiaalsed häälebrauserid. Süsteemi ülesehitamiseks ühendatakse brauserite andmepangad avaliku telefonivõrguga. Sellisel viisil on loodud palju juturoboteid ja mitmesuguseid kommertssüsteeme.
4. Hübriidsüsteemid, mis ühendavad reeglid ja masinõppe.

33.4.4. Kõnedialoogsüsteemide näiteid

Maailmas on dialoogsüsteemide arendamine kestnud juba neli aastakümnet. Esimesed lihtsad süsteemid, mis suhtlesid kasutajaga kõne abil, jõudsid kommertsrakendusteni siiski alles paarkümmend aastat tagasi. Need on mitmesugused teadetesüsteemid (liiklusinfo, pangainfo vms), ärikokkulepete sõlmimise (pangalaenu saamise, autokindlustuse, ostuabi jms) süsteemid, probleemilahendusel (nt arvutirikke tuvastusel) abistavad süsteemid.

Anname siin ülevaate mõnede tuntud firmade dialoogsüsteemidest, millega saab suhelda kõne vahendusel.

- **Nuance** (Ameerika Ühendriigid) – häälvälimisega telefon, e-posti ettelugemine, kindlustus, pangateenus, reisiinfo jms⁶⁰.
- **AT&T Labs** (American Telephone and Telegraph, Ameerika Ühendriigid) – kellaaja teatamise, krediidiinfo jms süsteemid⁶¹.
- **Philips** (Holland) toodab dikteerimissüsteeme, telefoniteenuste süsteeme⁶².
- Rootsi firma **Telia** ja Soome firma **Nokia** rakendavad mõlemad oma firmas telefonikõne suunamist töötaja nime järgi; Telia säästab sellega väidetavalt 40 miljonit Rootsi krooni aastas.
- **Loquendo** (Itaalia) on välja töötanud rongiinfosüsteemi⁶³, mille andmebaasis on 3000 linna/jaama Itaalia raudteel. Süsteem saab 2 miljonit telefonikõnet kuus.

Dialoogsüsteemide arendamisega tegeldakse paljudes ülikoolides ja uurimisinstituutides.

- Massachusettsi Tehnoloogiainstituudis on loodud mitmeid dialoogsüsteeme⁶⁴, mis annavad lennuinfot koos piletihindadega, lennukite saabumis- ja väljumisaegadega, liiklusinfot ja restoraniinfot Bostoni piirkonnas, ilmateateid (enam kui 500 linna kohta üle kogu maailma).
- Carnegie Melloni ülikoolis on loodud kõnedialoogsüsteem **CMU Communicator**⁶⁵, mis aitab planeerida reisi (lennuinfo, piletite, hotelli reserveerimine, autorent).
- Soome ülikoolide ning mõnede Soome ja Jaapani firmade koostöös on viidud läbi projekt **Interact**⁶⁶ dialoogsüsteemi loomiseks, mis võimaldab suunata telefoni-kõnesid, reserveerida pileteid, saada infot Helsingi bussisõiduplaanide kohta, interaktiivset abi probleemilahendusel.

⁵⁹ <http://www.voicexml.org/> (18.02.2010)

⁶⁰ <http://www.nuance.com/speech/> (18.02.2010)

⁶¹ <http://www.att.com/> (18.02.2010)

⁶² <http://www.philips.com/> (18.02.2010)

⁶³ <http://www.loquendo.com/> (18.02.2010)

⁶⁴ <http://www.sls.lcs.mit.edu/applications> (18.02.2010)

⁶⁵ <http://www.speech.cs.cmu.edu/Communicator/> (18.02.2010)

⁶⁶ <http://www.mlab.uiah.fi/interact/> (18.02.2010)

- Rochesteri ülikoolis on loodud dialoogsüsteemi TRAINS edasiarendus TRIPS⁶⁷ (The Rochester Interactive Planning System), kus rakendusstsenaariumiks on inimeste päästmine katastroofipiirkonnast, seejuures on aeg ja transpordivahendite arv piiratud. Kasutatakse ka teede kaarti.
- Saksamaa Tehisintellekti Uurimisinstituudis (DFKI) on professor Wolfgang Wahlsteri juhtimisel läbi viidud mitmeid suuri projekte inimesega loomulikus keeles suhtlevate dialoogsüsteemide loomiseks, sh
 - **Verbmobil** (varem juba mainitud) – kohtumiste kokkuleppimise süsteem, mis hõlmab ka reisiplaneerimise ja kõik sellega seonduva (nt hotelli reserveerimine) ning ühtlasi võimaldab automaatselt tõlkida kõnet ühest keelest teise (valida on kolme keele – saksa, inglise ja jaapani keele vahel).
 - **SmartKom**⁶⁸ (1999–2003) on mitmefunktsiooniline dialoogsüsteem, millel on kolm põhilist rakendusstsenaariumi:
 - *SmartKom-Public* – multimodaalne telefoniautomaat (kaasatud on nii kõne kui ka žestid ja miimika),
 - *SmartKom Mobile* – Interneti-teenused mobiiltelefoni teel,
 - *SmartKom-Home / Office* – arvuti häälujuhtimine mitmesugusteks rakendusteks, sh dokumentide dikteerimine, infootsing Internetist jms.
 - **VirtualHuman**⁶⁹ (2002–2006). Eesmärk oli arendada virtuaalseid karaktereid, mis oleksid emotsionaalsed ja realistlikud. Põhitähelepanu pöörati personaalsete dialoogipartnerite loomisele, mis tegutsesvad reaalses ja autonoomselt. Ühe võimaliku rakendusstsenaariumina nähakse e-õpet.

Tuleb rõhutada, et praegu on aktuaalseks uurimisprobleemiks multimodaalne suhtlus arvutiga, mis lisaks loomulikule keelele (eeskätt suulisele keelele) hõlmab ka žeste ja miimikat, sh emotsioonide edasiandmist ja tõlgendamist jms.

Üks suurejoonelisemaid projekte on Cyc (sõnast encyclopedia), mida alustati Ameerika Ühendriikides juba 1984. a. Projekti eesmärk on luua väga mahukas mõisteid, fakte ja reegleid koondav teadmusbaas, mille abil tehisintellektisüsteem suudaks arutleda, õppida ja inimesega loomulikus keeles suhelda. Juulis 2006 valminud versioon sisaldab 47 000 mõistet ja 306 000 fakti⁷⁰.

33.5. Eesti keeles suhtlev arvuti

Dialoogsüsteemi kavandamisel tuleb uurida nii inimestevahelist kui ka inimese ja arvuti vahelist suhtlust. Kuni dialoogsüsteemi pole veel loodud, ei saa muidugi koguda inimese ja arvuti vahelisi dialooge. Siis kasutatakse nn võlur Ozi meetodit: inimestel palutakse justkui testida programmi, mis suhtleb nendega loomulikus keeles, kuid tegelikult etendab arvuti rolli arvutivõrgu kaudu teine inimene („võlur Oz”). Teine meetod on olemasolevate inimestevaheliste dialoogide nn destilleerimine, eesmärgiga muuta neid „arvutilaadseteks”.

⁶⁷ <http://www.cs.rochester.edu/research/trips> (18.02.2010)

⁶⁸ <http://www.smartkom.org/> (18.02.2010)

⁶⁹ <http://www.virtual-human.org/> (18.02.2010)

⁷⁰ <http://www.cyc.com/> (18.02.2010)

Kogutud dialoogid koondatakse korpustesse ning dialoogikorpusi märgendatakse mitmel tasemel: fonoloogilisel, morfoloogilisel, süntaktilisel, dialoogiaktide tasemel jne.

33.5.1. Eesti dialoogikorpus

Eesti dialoogikorpuse (EDiK) loomist alustati Tartu Ülikoolis 2001. aastal. Korpus sisaldab kolme liiki dialooge⁷¹:

1. inimestevahelised dialoogid Tartu Ülikooli suulise eesti keele korpusest⁷². Seisuga jaanuar 2010 sisaldas korpus kokku umbes 1100 litereeritud telefonikõnet ja silmast-silma vestlust;
2. võlur Ozi meetodil kogutud kirjalikud dialoogid, kus arvuti rolli on etendanud inimene;
3. lihtsate veebis kasutatavate dialoogsüsteemidega Reisiagent, Teatriagent ja Kinoagent toimunud dialoogid (programmide autor Margus Treumuth)⁷³.

Osa suulisi dialooge on märgendatud morfoloogiliselt ja süntaktiliselt. Selleks on kasutatud eesti kirjakeele morfoloogilist analüsaatorit ESTMORF ja eesti keele kitsenduste grammatika süntaktilist analüsaatorit, mida on kohandatud suulisele keelele. Kõigis dialoogides (v.a arvutiga peetud dialoogid) on märgendatud dialoogiaktid, nt tervitus, küsimus jms.

Näide 33.14. EDiK: kahe inimese telefonidialoog. H – helistaja (klient), V – vastaja (ametnik). Kõne üleskirjutamisel on kasutatud vestlusanalüüsi transkriptsiooni (Hennoste, Rääbis 2004).

```
((kutsung))
V: bussi`info=kuuleb=
tere
H: `hh tere.
.h mul on=nüd selline `küsimus.
(.) mt e < `mis (0.5) e `kell `läheb (.) selline `buss
mis jõuaks > (.) e `Soome mineva `laeva `peale.
`Soome minev `laev läheb `Tallinast (.) kell=`kaheksa.
(0.5)
`kas `lähevad bussid ka: `sadamasse.
V: ei, | KYJ: EI |
`sadamasse läheb ainult `üks`ainuke `buss,
see sõidab `Tartust väljust välja `kaksteist
neligend=`viis.
```

Korpusesse kogutud dialoogide analüüs võimaldab vastata küsimusele, missugune peaks olema suhtlus arvutiga, võrreldes inimestevahelise suhtlusega, nt missugune peaks olema arvuti sõnavara ja grammatika, kuidas lahendada mitmesusi, kuidas kasutada konteksti jms.

Näide 33.15. EDiK: dialoog võlur Oziga (sisend klaviatuurilt, väljund ekraanile)

```
Infoklient: mis kell väljub buss Tartust Võrru, et ma
oleksin Võrus juba kell 17.00?
```

⁷¹ <http://math.ut.ee/~koit/Dialoog/EDiC.html> (18.02.2010)

⁷² <http://www.cl.ut.ee/suuline/> (18.02.2010)

⁷³ <http://www.dialoogid.ee> (18.02.2010)

Infoklient: mind huvitab lähim buss
 VOZ: Oodake, palun!
 VOZ: Buss väljub kell 14.15 ja saabub Võrru kell 15.50
 Infoklient: Kas see buss sõidab läbi Põlva?
 VOZ: Jah.
 Infoklient: Mis kell ta peatub Põlvas?
 VOZ: Buss peatub Põlvas kell 15.15
 VOZ: Kas saan Teile veel kasulik olla?

Näide 33.16. EDiK: dialoog arvutiga (Reisiagent; sisend klaviatuurilt, väljund ekraanile ja soovi korral ka tehiskõnes)

K(asutaja): tere!
 A(rvuti): Tere. Kuhu soovite lennata?
 K: soovin lennata itaaliasse
 A: Itaalia puudub lennuandmebaasist.
 K: Siis tahan lennata prantsusmaale
 A: Pariisi saab lennata teisipäeval kell 08:00.
 K: tahaksin lennata hoopis 6. aprillil
 A: Neljapäeval saab Pariisi lennata kell 08:00.
 K: kust ma piletit osta saan?
 A: Pileti saate Estraveli kassast.

33.5.2. Loomuliku keele töötlus dialoogsüsteemis

Dialoogsüsteem täidab tsükliliselt kolme järjestikust protsessi: 1) kasutaja lausungi (nt küsimuse) analüüs, 2) probleemilahendus (leidmaks vastust küsimusele), 3) oma lausungi süntees (vt ka p. 33.1). Probleemilahendus on ainus protsess, mis ei sõltu keelest ja kasutab teatava formalismi abil esitatud maailmateadmust, andmebaase, mitmesuguseid universaalseid ja spetsiifilisi probleemilahendusmeetodeid.

Eesti keele automaattötluseks on valminud juba mitmesugust tarkvara. Sellest annab ülevaate tabel 33.5.

Tabel 33.5. Eesti keele tarkvara

| <i>Keeletöötlustase</i> | <i>Tarkvara</i> | <i>Uurimisasutus</i> |
|-------------------------|---|----------------------|
| Morfoloogia | Morfoloogiline analüsaator ja süntesaator ^{74,75} | TÜ, Filosoft, EKI |
| Süntaks | Kitsenduste grammatika süntaktiline analüsaator ⁷⁶ , morfoloogiline ühestaja | TÜ TÜ, Filosoft |
| Semantika | Sõnatähenduste ühestaja ⁷⁷ , dialoogiaktide klassifitseerija | TÜ |
| Kõnetöötlus | Tekst-kõne süntesaator ⁷⁸ , kõnetuvastaja | TTÜ, EKI |

⁷⁴ <http://www.filosoft.ee/> (18.02.2010)

⁷⁵ <http://www.eki.ee/tarkvara/> (18.02.2010)

⁷⁶ <http://www.cs.ut.ee/~kaili/parser/demo/> (18.02.2010)

⁷⁷ <http://math.ut.ee/~kaarel/NLP/> (18.02.2010)

⁷⁸ <http://www.phon.ioc.ee> (18.02.2010)

Eespool nimetatud dialoogsüsteemid (Reisiagent, Teatriagent, Kinoagent) kujutavad endast katseid ühendada olemasolevat eesti keele tarkvara lihtsate küsimus-vastus-dialoogide läbiviimiseks. Siin on tegu kitsalt piiritletud ülesannetega: Reisiagent vastab kasutaja küsimustele lennukite väljumisaegade kohta Tallinna lennujaamast, Teatriagent annab infot Eesti teatrite mängukavadest ning Kinoagent Tartu kinoinfot. Süsteemiga suhtlemine toimub sel viisil, et kasutaja sisestab veebibrauseris päringuaknasse oma küsimuse ja näeb ekraanil vastust ning soovi korral kuuleb sedasama tehiskõnes. Teatriagentile on lisatud ka kõnetuvastusmoodul (mitte veel veebiversioonis), nii et kasutaja saab oma küsimuse esitada hoopis suuliselt. Kirjaliku eesti keele töötlemise tarkvarast on mõlemas süsteemis rakendatud üksnes morfoloogilist analüüsi ja sünteesi. See on siin otstarbekohane: piiratud ainevaldkonna korral ei ole vaja läbi teha analüüsi täistsükli, süsteem otsib kasutaja küsimusest ainult sobivaid märksõnu (nt kuupäeva, teatri või linna nime jms) ja vastuse sünteesimise asemel saab kasutada valmis vastuste šabloone, milles üksnes täidetakse tühje kohti.

33.6. Suhtlus arvutiga tulevikus

Tuntud arvutiteadlane, Sheffieldi ülikooli professor Yorick Wilks näeb dialoogsüsteemide tulevikku robot-kaaslastes, mida ta nimetab kompanjonideks (Wilks 2006). Kompanjon on konversatsiooniagent, juurdepääsuga Internetile. Ta on isikustatud oma omaniku suhtes: teab tema vajadusi, eelistusi, harjumusi ning õpib omaniku kohta süstemaatiliselt juurde. Ta võib olla näiteks mänguaja- või lemmikloomakujuuline, mida laps saab ranitsas kaasas kanda. Eraldi väärivad märkimist seniorkompanjonid vanuritele ja juuniorkompanjonid lastele. Seniorkompanjon võib oma omanikule meelde tuletada huvitava telesaate algust, lülitades sisse televiisori, ravimi võtmise aega, vanuriga lihtsalt suhelda, vajaduse korral talle abi kutsuda jms. Juuniorkompanjon hoiatab kooliteed alustavat last ohtude eest liikluses, aitab koolitükke ette valmistada, otsides infot Internetist, ajakavast kinni pidada jne.

Kompanjonid on uus agentide liik vastukaaluks passiivsele veebile: veeb peab muutuma personaalsemaks, et olla hoomatav ka kiirelt kasvades. Semantiline veeb on esimene samm veebi struktureerimiseks, aga praegused veebiagentid on siiski veel abstraktsed ja ajutised. Personaalsed agentid peavad olema autonoomsed, püsivad ja usaldusväärsed ka selliste elanikkonna kihtide jaoks, kes praegu on veebist eemal.

Tulles tagasi eespool (p. 1) mainitud Turingi testi juurde, võib viidata Ameerika Ühendriikide arvutiteadlase ja visionääri Raymond Kurzweili sõnavõtule⁷⁹ 2001. a, kus ta rõhutas, et Turingi test on märkimisväärne sellepolest, et näitab, kuidas keel peegeldab inim mõtlemist. Praegu on olemas palju arvutiprogramme, mis suudavad kitsa ainevaldkonna probleeme lahendada niisama hästi kui inimene: malemäng, meditsiiniline diagnoosimine, finantsotsuste vastuvõtmine jpm. Seega võib väita, et arvuti suudab nendes valdkondades täita Turingi testi. Et arvuti läbiks Turingi testi täies mahus, s.t et ta suudaks suhelda nagu inimene, peaks tal olema inimese tasemel mõtlemisvõime. Kurzweili arvates saab see võimalikuks aastal 2029.

⁷⁹ <http://www.longbets.org/1> (18.02.2010)

34. Tehisnägemine

Tehisnägemine (*computer vision*) on tehisintellektisüsteemi võime hõivata, töödelda ja interpreteerida visuaalandmeid.

Tehakse vahet kujutuvastuse (*pattern recognition*) ja pildituvastuse (*image recognition*) vahel. Viimasel juhul on ülesandeks kahemõõtmelise kujutise põhjal tuvastada kolmemõõtmeline stseen, kusjuures stseenid võivad olla nii staatilised kui ka dünaamilised.

Kujutuvastuseks on mitmeid meetodeid. Toome siin näiteks geomeetrilise meetodi. Olgu antud m -mõõtmeline tunnuste ruum D . Iga objekt esitatakse kujul $a=(a_1, \dots, a_m)$. Olgu antud etalonide hulk $X = \{X_1, \dots, X_n\}$, kus $X_i = \{x_{i1}, \dots, x_{ik}\}$ ning $x_{ij} = (x_{ij1}, \dots, x_{ijm})$, $i=1, \dots, n$; $j=1, \dots, k$.

Kaugus objektide a ja b vahel määratakse järgmisest valemist:

$$d(a, b) = \sqrt{\sum_{i=1}^m (a_i - b_i)^2}.$$

Meetodi idee on järgmine: $y \in X_i$, kui keskmine (ruut)kaugus objekti y ja etaloni X_i objektide vahel on väike.

Esmalt teisendatakse ruumi D , et ühte hulka kuuluvad objektid oleksid võimalikult lähedal:

- 1) pööratakse ruumi nii, et koordinaatteljed oleksid hulga X_i punktidega määratud m -mõõtmelise ellipsoidi telgede suunalised;
- 2) muudetakse telgede mastaapi, et ellipsoid teiseks keraks.

Teisendus tehakse iga X_i korral, selle tulemusel saadakse teisendatud ruumid D_1^*, \dots, D_n^* . Siis leitakse $d(y, X_i)$ igas ruumis D_i^* ($i=1, \dots, n$). Sobib selline X_i , kus kaugus $d(y, X_i)$ on minimaalne, s.t $y \in X_i$.

Pildituvastuseks läbitakse kujutise töötlemisel järgmised etapid.

- 1) Pilt esitatakse maatriksina, mille iga element näitab kujutise vastava punkti eredust.
- 2) Määratakse pildil asuvate objektide servad (s.o kohad, kus ereduse väärtuses toimub järsk muutus; servi lähendatakse sirglõikudega).
- 3) Eraldatakse objektid üksteisest: otsustatakse, millised servad kuuluvad samale objektile.
- 4) Moodustatakse täielikud kujutised, „jätkates” varjatud servi.
- 5) Tuvastatakse eraldi iga objekt, võrreldes teda etteantud etalonide kõikvõimalike projektsioonidega.

Põhilised raskused, millega tehisnägemine kokku puutub, on suur andmemaht (võrdluseks: inimese nägemissüsteem töötleb 22 kujutist sekundis), müra ja üleminek kahemõõtmeliselt kujutiselt kolmemõõtmelisele.

Tehisnägemisel on mitmeid praktilisi rakendusi, sh käekirja tuvastamine, automaatsed navigeerimissüsteemid, aerofotode, röntgenipiltide, tänavastseenide analüüs, mikroskeemide montaaž.

35. Masinõpe

Masinõpe e tehisõpe (*machine learning, automatic learning*) on protsess, millega tehisintellektisüsteem täiustab oma talitlust, omandades uut teadmust või uusi oskusi või seniseid ümber korraldades.⁸⁰

35.1. Masinõppe meetodid

Tehisintellektisüsteemides kasutatakse mitmesuguseid erinevaid *õppimismeetodeid* (Rich, Knight 1991). Mitmed neist käsitlevad õppimisprotsessi kui otsingut.

1. Mehaaniline õpe (*rote learning*) – teadmusbaasi loomine, sellesse teadmuse paigutamine.
2. Soovitusõpe (*learning by taking advice*) – otsing, kus algolekuks on üldine nõuanne õpetajalt, lõppolekuks aga konkreetne reegel.
Näide: programm kõrgtaseme keeles annab lõpptulemuseks objektkoodi.
3. Parameetrite reguleerimine (*parameter adjustment*).
Näide: kabeprogramm (autor Arthur L. Samuel, 1959) mängis õppimisrežiimis iseenda koopia vastu. Kasutas staatilist hinnangufunktsiooni kujul $c_1t_1 + \dots + c_{16}t_{16}$ (c_i – kordajad, t_i – tunnused; $i=1,\dots,16$), muutis kordajaid vastavalt mängu tulemusele.
4. Näidetest õppimine e induktiivõpe. Siin on algolekuks positiivsete ja negatiivsete näidete kogum, lõppolekuks aga mõiste kirjeldus. Induktiivõpe on protsess, kus leitakse hüpotees, mis on kooskõlas kõigi positiivsete näidetega ning vastuolus kõigi negatiivsete näidetega.
Induktiivõpet nimetatakse ka *juhendajaga õppeks* (*supervised learning*) – tagasiside kas õpetajalt või keskkonnalt annab teada, kas näide on positiivne või negatiivne.
5. Selgituspõhine õpe e selgitusõpe (*explanation-based learning*).
Põhiidee: kasutada ühe probleemilahenduse tulemusi järgmise probleemi lahendamisel (nt malemängus vältida teatavat kaotusele viinud lauaseisu).
6. Avastamine (*discovery*) – heuristiline otsing, kus algolekuks on mingi keskkond, lõppolek on teadmata. Kasutatakse heuristikuid, nagu huvitavus, analoogia jms.

Näide: programm AM (D. Lenat, 1976). Sisend: 1) hulgateooria mõisted Lipsi funktsioonidena; 2) heuristikud selle kohta, „kuidas teha matemaatikat” (vrd nt Pólya 2001). AM „avastusi”:

- a. täisarvud – hulga elementide loendamise teel;
- b. liitmine – ühisosata hulkade ühendamise teel;
- c. korrutamine;
- d. algarvud;
- e. Goldbachi hüpotees (iga arvust 2 suurem paarisarv on esitatav kahe algarvu summana); hüpotees pole tänaseni tõestatud;
- f. algarvude vastandid – arvud, mille jagajate arv on võimalikult suur.

⁸⁰ IT terministandardi sõnastik <http://www.keeveeb.ee> (18.02.2010)

AM edu aluseks oli tihe seos matemaatika ja Lisp-i vahel: programm genereeris huvitavaid Lisp-funktsioone, modifitseerides huvitavateks tunnistatud matemaatiliste funktsioonide koodi.

7. Deduksioon – mõttekäik, milles *üldiste teadmiste* põhjal tuletatakse midagi konkreetsema juhu kohta. Deduktiivse tuletusreegli näiteks on *modus ponens*.

Käsitleme järgnevas lähemalt induktiivõpet.

35.2. Induktiivõpe

Induktiivõppe lihtsaim vorm on teatava funktsiooni f õppimine näidetest (vrd Russell, Norvig 2003).

Näide on paar $(x, f(x))$, kus x on sisend ja $f(x)$ on väljund.

Näide 35.1. Trips-traps-trulli mängus võib kasutada näiteid, kus sisendiks x on teatav lauaseis ja funktsiooni väärtus $f(x)$ on kas $+1$, -1 või 0 sõltuvalt sellest, kas lauaseis kujutab ristikestega mängija võitu, kaotust või viiki, nt:

| | | |
|---|---|---|
| O | O | X |
| | X | |
| X | | |

$+1$

Probleem püstitatakse järgmiselt. Antud on treeningnäidete hulk. Leida selline hüpotees h , et $h \cong f$.

See on tegeliku õppimise väga lihtsustatud mudel, sest ta

- ignoreerib eelnevat teadmist (lähteks on „puhas tahvel”, *tabula rasa*),
- eeldab deterministlikku, täielikult vaadeldavat keskkonda,
- eeldab, et näited on antud,
- eeldab, et agent soovib õppida funktsiooni f .

Objektide/ülesannete näited säilitatakse andmebaasis. Iga näide on kirjeldatud atribuutide /tunnuste hulgaga. Iga näide on liigitatud kui positiivne või negatiivne. Programm õpib näidetest üldise mõiste kirjelduse ja suudab seejärel liigitada/ täita ülesannet, mida kunagi varem pole lahendanud (vt ka Ginsberg 1993).

35.2.1. Tõenäoliselt ligikaudu korrektne õpiprotseduur

Olgu x – suvaline objekt, $p(x)$ – mõiste, mida süsteem parajasti õpib, ja $p'(x)$ – selle lähend.

Def. 1. Õpitud reeglit p' nimetatakse *ligikaudu korrektseks* (*approximately correct*) täpsusega ϵ parajasti siis, kui reegli rakendamisel tehtava vea tõenäosus on ülimalt ϵ .

Kui näiteks eeldame, et süsteem on õppinud ligikaudu korrektse reegli „kollane seen on mürgine”, siis tõenäosus selleks, et järgmine kollane seen ei ole mürgine, on $\leq \epsilon$.

Olgu $t(x)$ – tõenäosus selleks, et objektide hulgast U valitakse järgmisena klassifitseerimiseks x . Eeldame, et $\sum_{x \in U} t(x) = 1$. Siis vea tõenäosus: $\sum_{\{x | viga(x)\}} t(x)$.

Def. 2. Õpiprotseduuri L nimetatakse *tõenäoliselt ligikaudu korrektseks* (probably approximately correct, PAC) usaldusväärsusega $1-\delta$, kui juhuslikult valitud treeningnäidete kogumil on tõenäosus selleks, et L õpib reegli, mis ei ole ligikaudu korrektne, ülimalt δ .

Induktiivõppe süsteemides kasutatakse veel täiendavalt *eelarvamusi* (bias), s.t piirduakse kindla kujuga mõistete õppimisega.

Mõned eelarvamused:

1. piiratud hüpoteesiruumi eeldus: nii sihtmõiste p kui ka õpitud lähend p' rahuldavad teatavat süntaktilist piirangut.
2. $p(x)$ on kujul $x > c$ (või $x < c$).
3. Õpitav mõiste on esitatav *otsustuspuu* (decision tree) abil.

35.2.2. Piiratud hüpoteesiruumi eeldus

Vaatame olukorda, kus kehtib piiratud hüpoteesiruumi eeldus. Olgu L õpiprotseduur m näitest koosneval treeninghulgal.

L tagastab predikaadi p' , mis rahuldab piiratud hüpoteesiruumi eeldust ja on kooskõlas treeningnäidetega. Küsimine: kas L on tõenäoliselt ligikaudu korrektne?

Vastus on jaatav, kui U on lõplik ja m on piisavalt suur. Leiame nüüd, kui suur peab olema m , ehk teisiti: kui suur on tõenäosus selleks, et L õpib p' , mis rahuldab treeninghulka, aga **ei ole** ligikaudu korrektne.

Olgu q mõiste, mis pole ligikaudu korrektne. Tõenäosus, et q sobib ühe treeningnäitega, ei ole suurem kui $1-\epsilon$. Tõenäosus, et q sobib kõigi (m) treeningnäidetega, on ülimalt $(1-\epsilon)^m$. Olgu võimalike mõistete arv H . Tõenäosus, et leidub mõiste q , mis pole ligikaudu korrektne, on ülimalt $H(1-\epsilon)^m$. Õpiprotseduur L on aga eelduse kohaselt tõenäoliselt ligikaudu korrektne, millest järeldub, et $H(1-\epsilon)^m \leq \delta$. Seega kehtib

Lemma. Õpiprotseduur, mis lähtub m treeningnäitest ja tagastab predikaadi p' , mis 1) rahuldab piiratud hüpoteesiruumi eeldust ja 2) on kooskõlas treeningnäidetega, on tõenäoliselt ligikaudu korrektne, kui $H(1-\epsilon)^m \leq \delta$.

Avaldame nüüd m :

$$(1 - \epsilon)^m \leq \delta/H$$

$$m \ln(1 - \epsilon) \leq \ln(\delta/H) \quad | : \ln(1 - \epsilon)$$

$$\ln(1 - \epsilon) < -\epsilon$$

$$m \geq \ln(\delta/H) / \ln(1 - \epsilon)$$

$$m \geq \ln(\delta/H) / (-\epsilon) = \ln(H/\delta) / \epsilon$$

Seega kehtib

Tulemus. Õpiprotseduur, mis lähtub m treeningnäitest ja tagastab predikaadi p' , mis 1) rahuldab piiratud hüpoteesiruumi eeldust ja 2) on kooskõlas treeningnäidetega, on tõenäoliselt ligikaudu korrektne, kui $m \geq \ln(H/\delta) / \epsilon$.

Näide 35.2. Olgu antud 52 kaardist koosnev kaardipakk. Õppivale süsteemile näidatakse üksahaaval kaarte – positiivseid ja negatiivseid näiteid. Süsteem peab õppima mõiste, kasutades eelarvamust, et õpitav mõiste on **positiivsete** literaalide konjunktsioon (s.o nn konjunktiivne mõiste).

Treeningnäidete hulk:

| Kaart | Kas kuulub sihthulka T? |
|---------|-------------------------|
| poti 6 | jah |
| risti 7 | jah |
| ärtu 8 | ei |
| risti 9 | jah |
| ... | |

Õpialgoritmi idee on järgmine: konjunktiivsete mõistete hulk on **osaliselt järjestatav spetsiifilisuse** alusel. Nt 'must' > 'poti' > 'poti 8'. Algoritmi igal sammul leitakse kõige kitsam ja kõige laiem mõiste, mis seni kohatud treeningnäidetega on kooskõlas (vt järgnev tabel⁸¹).

| Samm | Näide | Kõige kitsam mõiste | Kõige laiem mõiste |
|------|-------------------|---------------------|-----------------------|
| 1) | Poti 6 $\in T$ | poti 6 | kõigi kaartide hulk |
| 2) | Risti 7 $\in T$ | must p-tu | -"- |
| 3) | Ärtu 8 $\notin T$ | -"- | 1. paaritu 2. must |
| 4) | Risti 9 $\in T$ | -"- | -"- |
| 5) | Ärtu 5 $\notin T$ | -"- = | -"- |
| | ... | | |

Õpitud mõiste on „must paaritu kaart”.

Üldiselt: olgu sihtmõiste ja õpitud mõiste *konjunktsioonid* n predikaati sisaldavas ruumis. Konjunktsioone on kokku 3^n , s.t $H=3^n$. Eelpool saadud tulemusest järeldub, et vajalik treeningnäidete hulk:

$$\begin{aligned}
 m &\geq \ln(H/\delta) / \epsilon = \\
 &= \ln(3^n/\delta) / \epsilon = \\
 &= (n/\epsilon) \ln(3/\delta).
 \end{aligned}$$

Esitame nüüd õpialgoritmi (Rich, Knight 1991), mis lähtub positiivsete ja negatiivsete näidete hulgast (mis on esitatud kui tunnuste vektorid teatavas teadmuse esituse keeles) ning leiab mõiste, mis on kooskõlas kõigi antud positiivsete näidetega ega ole kooskõlas ühegi negatiivse näitega. Algoritm kasutab hulki S (*source*) ja G (*goal*).

⁸¹ Äss loetakse paarituks.

Õpialgoritm

1. Sisaldagu hulk G üheainsa elemendi: nullkirjelduse (kõik tunnused on muutujad).
2. Sisaldagu S üheainsa elemendi: esimese positiivse näite.
3. Lugada uus treeningnäide.
 - Kui see on positiivne näide, siis esmalt eemaldada hulgast G kõik sellised kirjeldused, mis ei kata seda näidet. Seejärel muuta hulka S nii, et ta sisaldaks versiooniruumis kõige spetsiifilisema kirjelduste hulga, mis katab näite ja hulga S käesolevad elemendid.
 - Kui see on negatiivne näide, siis esmalt eemaldada hulgast S kõik kirjeldused, mis katavad seda näidet. Seejärel muuta hulka G nii, et ta sisaldaks versiooniruumis kõige üldisema kirjelduste hulga, mis ei kata seda näidet.
4. Kui S ja G on mõlemad üheelemendilised hulgad, siis
 - juhul, kui nad on võrdsed, väljastada nende sisu ja lõpetada.
 - juhul, kui nad ei ole võrdsed, siis ei olnud treeningnäidete hulk kooskõlaline. Väljastada sellekohane teade ja lõpetada.

Vastupidisel juhul minna 3.

Näide 35.3. Mõiste „Jaapani ökonoomia-auto” õppimine (Rich, Knight 1991).

Kasutame järgmist teadmuse esituse keelt (tunnused ja nende võimalikud väärtused).

Maa: {Jaapan, USA, Saksamaa, Suurbritannia, Itaalia}

Tootja: {Honda, Toyota, Ford, Chrysler}

Värvus: {sinine, punane, roheline, valge }

Aasta: {1970, 1980, 1990, 2000}

Tüüp: {ökonoomia, luks, sport}

Õpitav mõiste „Jaapani ökonoomia-auto” esitub järgmise vektorina (x_1 , x_2 ja x_3 on muutujad):

Maa: Jaapan

Tootja: x_1

Värvus: x_2

Aasta: x_3

Tüüp: ökonoomia

Olgu positiivsete (+) ja negatiivsete (-) näidete hulk järgmine:

1) +

Maa: Jaapan

Tootja: Honda

Värvus: sinine

Aasta: 1980

Tüüp: ökonoomia

2) –

Maa: Jaapan

Tootja: Toyota

Värvus: roheline

Aasta: 1970

Tüüp: sport

3)+

Maa: Jaapan

Tootja: Toyota

Värvus: sinine

Aasta: 1990

Tüüp: ökonoomia

4) –

Maa: USA

Tootja: Chrysler

Värvus: sinine

Aasta: 1980

Tüüp: ökonoomia

5)+

Maa: Jaapan

Tootja: Honda

Värvus: valge

Aasta: 1980

Tüüp: ökonoomia

Õpialgoritmi töö protokoll

I. $G = \{(x_1, x_2, x_3, x_4, x_5)\}$, $S = \{(\text{Jaapan}, \text{Honda}, \text{sinine}, 1980, \text{ökonoomia})\}$

II. $G = \{(x_1, \text{Honda}, x_3, x_4, x_5), (x_1, x_2, \text{sinine}, x_4, x_5), (x_1, x_2, x_3, 1980, x_5), (x_1, x_2, x_3, x_4, \text{ökonoomia})\}$,

$S = \{(\text{Jaapan}, \text{Honda}, \text{sinine}, 1980, \text{ökonoomia})\}$

III. $G = \{(x_1, x_2, \text{sinine}, x_4, x_5), (x_1, x_2, x_3, x_4, \text{ökonoomia})\}$,

$S = \{(\text{Jaapan}, x_2, \text{sinine}, x_4, \text{ökonoomia})\}$

IV. $G = \{(\text{Jaapan}, x_2, \text{sinine}, x_4, x_5), (\text{Jaapan}, x_2, x_3, x_4, \text{ökonoomia})\}$,

$S = \{(\text{Jaapan}, x_2, \text{sinine}, x_4, \text{ökonoomia})\}$

V. $G = \{(\text{Jaapan}, x_2, x_3, x_4, \text{ökonoomia})\}$, $S = \{(\text{Jaapan}, x_2, x_3, x_4, \text{ökonoomia})\}$

Hulgad G ja S on mõlemad üheelemendilised ja võrdsed, seega on algoritm oma töö lõpetanud ja õppinud mõiste „Jaapani ökonoomia-auto”: $(\text{Jaapan}, x_2, x_3, x_4, \text{ökonoomia})$.

35.2.3. Eelarvamus $x > c$

Siin eeldame, et õpitav mõiste $p(x)$ on kujul $x > c$, ja otsime c .

Kuidas määrata c ? Vaatame kõigepealt järgmist näidet. Olgu $\{x\}$ – reaalarvude hulk, c – konstant.

Näide 35.4. Õpitav mõiste „ $x > 5.0$ ”.

Treeningnäited:

| | | | | | | |
|--------------------|-----|-----|-----|-----|--|-----|
| ei | ei | ei | | jah | | jah |
| 2.6 | 3.4 | 4.2 | 5.0 | 5.3 | | 7.2 |
| -----> | | | | | | |
| $c \in [4.2, 5.3)$ | | | | | | |
| ... | | | | | | |

Otsitav punkt c jagab arvtelje kaheks osaks.

Kui tunnuseid on rohkem kui üks, siis võib rakendada analoogilist mõttekäiku:

- tasand jaotatakse kaheks osaks sirgega $h_1 x_1 + h_2 x_2 = c$,
- 3-mõõtmeline ruum jaotatakse kaheks osaks tasandiga,
- ...
- n -mõõtmeline ruum jaotatakse kaheks osaks hüpertasandiga $\sum_{i=1}^n h_i x_i = c$.

Seega üldjuhul on õpitav mõiste kujul

$$\sum_{i=1}^n h_i x_i > c. \quad (35.1)$$

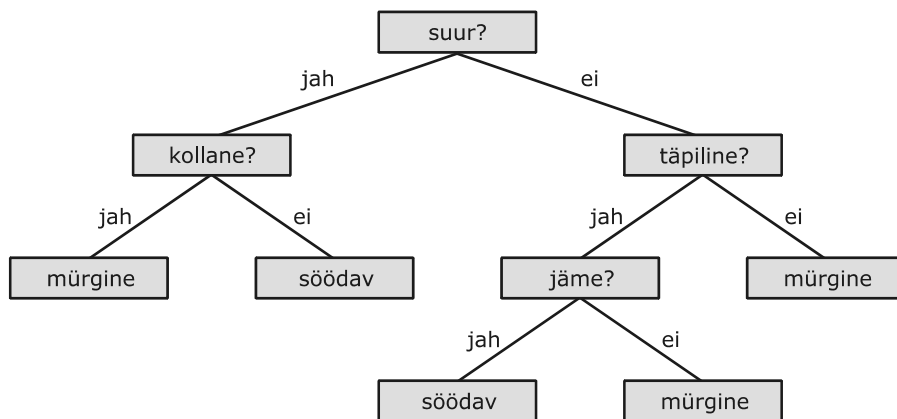
Õpiprotsessi modelleerib *tehisnärvivõrk*, kus x_i ($i=1, \dots, n$) on sisendid. Väljundid on 1 või 0, sõltuvalt sellest, kas võrratus (35.1) kehtib või mitte.

Võib kasutada mitmekihilist tehisnärvivõrku, kus iga eelmise kihi väljundid on järgmise kihi sisenditeks.

35.2.4. Otsustuspuu

Eelarvamus: õpitav mõiste on väljendatav (binaarse) otsustuspuu abil, jagamaks treeningnäidete hulka igas tipus positiivseteks ja negatiivseteks näideteks.

Joonisel 35.1 on toodud binaarse otsustuspuu näide.



Joonis 35.1. Binaarne otsustuspuu seente liigitamiseks

Uurime järgnevas, kas saab täpselt defineerida, millist tunnust otsustuspuu millises tipus kontrollida.

Olgu objektide hulk S , võimsusega $|S|$. Esitame jah-ei-küsimusi. Näiteks kui S on täisarvude hulk, mis on väiksemad kui 1000, ja ülesandeks on leida x sellest hulgast, siis võime esitada järgmisi jah-ei-küsimusi.

- Kas $x > 500$? Kui vastus „jah”, siis
- Kas $x > 750$? Jne.

Minimaalne küsimuste arv on $\log_2|S|$.

Üldiselt, olgu S jagatud kaheks osahulgaks P (positiivsed) ja N (negatiivsed). Kui on teada, kummasse osahulka kuulub x , siis

- kui $x \in P$, siis x määramiseks on vaja esitada $\log_2|P|$ küsimust;
- kui $x \in N$, siis x määramiseks on vaja esitada $\log_2|N|$ küsimust.

Olgu t_P tõenäosus selleks, et $x \in P$, ja t_N tõenäosus selleks, et $x \in N$. Siis küsimuste koguarv:

$$t_P \log_2|P| + t_N \log_2|N|.$$

Seega: kui **teaksime ette**, kummasse hulka kuulub x , siis **säästaksime** x määramisel

$$\log_2|S| - (t_P \log_2|P| + t_N \log_2|N|)$$

küsimust.

Kuna $t_P = |P|/|S|$ ja $t_N = |N|/|S|$, siis küsimuste arv

$$\begin{aligned} & \log_2 |S| - t_P \log_2 (t_P |S|) - t_N \log_2 (t_N |S|) = \\ & = \log_2 |S| - t_P \log_2 t_P - t_P \log_2 |S| - t_N \log_2 t_N - t_N \log_2 |S| = \\ & = -t_P \log_2 t_P - t_N \log_2 t_N \end{aligned} \quad (35.2)$$

Olgu T – sihthulk. Leiame, milline on **parim** küsimus, määramaks, kas $x \in T$ (või $x \notin T$).

Vaatame hulga S suvalist osahulka U . Selle saab omakorda jaotada kaheks osahulgaks:

$$\begin{aligned} U_+ &= U \cap T, \\ U_- &= U \setminus T. \end{aligned}$$

Seosest (35.2) järeldub, et kui oleme õppinud, et objekt $\in T$, siis info juurdekasv:

$$\begin{aligned} I_U &= - |U_+|/(|U_+| + |U_-|) \log (|U_+|/(|U_+| + |U_-|)) - \\ & \quad - |U_-|/(|U_+| + |U_-|) \log (|U_-|/(|U_+| + |U_-|)) \end{aligned} \quad (35.3)$$

Olgu tunnus f ja küsimus: „kas vaadeldav x omab tunnust f ?”. Oletame, et meie varasemad küsimused on võimalike näidete hulga redutseerinud osahulgaks $V \subseteq S$.

Olgu V_{f+} hulga V osahulk, millesse kuuluvad objektid tunnusega f , ning V_{f-} hulga V selline osahulk, millesse kuuluvad objektid ilma tunnusega f .

Kui tunnuse f kasutamisega oleme ära õppinud, kas $x \in T$ või $x \notin T$ (s.t jaotanud V osahulkadeks V_{f+} ja V_{f-}), siis seose (35.3) põhjal, kus U rollis on vastavalt V_{f+} või V_{f-} , on info kasv (*gain*):

$$\begin{aligned} G_f &= |V_{f+}| / (|V_{f+}| + |V_{f-}|) \mathbf{I}_{V_{f+}} + \\ & \quad \text{info kasv, kui esmalt õpime, et } x \in V_{f+} \\ & \quad + |V_{f-}| / (|V_{f+}| + |V_{f-}|) \mathbf{I}_{V_{f-}} \end{aligned} \quad (35.4)$$

info kasv, kui esmalt õpime, et $x \in V_{f-}$

Def. Olgu otsustuspuu konstrueerimise protseduur jõudnud tippu, mis vastab objektide hulga osahulgale V . Öeldakse, et f on sihtmõiste T jaoks **maksimaalselt eristav tunnus**, kui ta *minimeerib* G_f väärtuse avaldises (35.4).

(See tähendab, et sihtmõiste T täieliku tuvastamiseni jääb vähem teha, kui seda oleks vaja mõne muu tunnuse valiku korral.)

Näide 35.5. Vaatleme jälle 52 kaardist koosnevat kaardipakki. Olgu sihtmõiste T : „must paaritu või ärtu 2”.

Vaatame kahte võimalikku küsimust:

1) kas kaart on punane?

2) kas kaart on 3?

Küsime: kumb on parem?

Küsimus 1.

50% kaartidest on punased, 50% mustad. Seosest (35.4) saame:

$$G_{värv} = 0,5I_{V_{punane}} + 0,5I_{V_{must}}$$

26 punasest üks (ärtu 2) kuulub hulka T, 25 kaarti ei kuulu hulka T.

Täiendav info teadmisest, kas $x \in T$ (või $x \notin T$), oleks (vt (35.3)):

$$I_{V_{punane}} = -1/26 \log(1/26) - 25/26 \log(25/26) = 0,235.$$

26 mustast kaardist on 12 paaritud (s.t $\in T$) ja 14 paaris (s.t $\notin T$).

Seega: kui teaksime ette, kas $x \in T$ või $x \notin T$, oleks täiendav info (vt (35.3)):

$$I_{V_{must}} = -12/26 \log(12/26) - 14/26 \log(14/26) = 0,996.$$

$$\text{Seega } G_{värv} = 0,5 I_{V_{punane}} + 0,5 I_{V_{must}} = 0,615.$$

Küsimus 2.

Kui küsime, kas kaart on 3, siis (35.4) põhjal:

$$G_3 = (1/13)I_{V_3} + (12/13)I_{V_{mitte-3}}$$

Neljast 3-st on 2 kaarti mustad ($\in T$), 2 kaarti punased ($\notin T$), s.t $I_{V_3} = 1$.

48 mitte-3-st on 11 kaarti hulgas T, 37 kaarti ei kuulu hulka T, s.t $I_{V_{mitte-3}} = 0,777$.

Kokkuvõttes $G_3 = 0,794$.

Järeldus: maksimaalselt eristav tunnus on värv ja efektiivsem on esimene küsimus.

Otsustuspuu konstrueerimise algoritm **ID3** (*Iterative Dichotomiser 3*), John Ross Quinlan, 1986

Olgu F – tunnuste hulk, S – objektide hulk, T – sihthulk. S_{f+} – hulga S osahulk, mis rahuldab f, S_{f-} – hulga S osahulk, mis ei rahulda f. Puu juurtipp (F, S, T).

KORRATA:

1. Kui iga x korral hulgas S kehtib $x \in T$, siis vastus 'jah'.

Kui ei leidu x $\in S$ nii, et $x \in T$, siis vastus 'ei'.

2. Vastupidisel juhul: olgu f $\in F$ maksimaalselt eristav tunnus. (Kui tunnuseid enam pole järele jäänud, siis ebaõnn.)

Tagastada puu, mille vasak haru on ID3(F \ {f}, S_{f+} , T) ja parem haru ID3(F \ {f}, S_{f-} , T) (s.t kleepida endisele külge).

35.3. Juhendajaga ja juhendajata õpe

Nagu eespool märgitud, on induktiivõpe juhendajaga õpe. Sellisel juhul õpib agent funktsiooni tema sisendi-väljundi näidetest. Näide: agent-taksojuht. Alati, kui õpetaja ütleb „Peatu!”, õpib agent tingimus-tegevus-reegli (millal peatuda kliendi pealevõtmiseks).

Õpe võib aga olla ka juhendajata (*unsupervised*). Näide: agent-taksojuht võib õppida mõisted „hea päev” ja „halb päev”, ilma et kummagi mõiste jaoks oleksid positiivsed ja negatiivsed näited märgendatud.

Kõige üldisem on hüvitusega (*reinforcement, reward*) õpe: õpetaja ei ütle, mis on hea, vaid agent õpib „autasust”. Näide: agent-taksojuht. Jootraha vähesus reisi lõpul näitab, et agendi käitumine polnud soovitud.

Õpitud info esitamiseks on erinevaid viise. Mänguprogrammides kasutatakse tavaliselt hinnangufunktsioonina lineaarset polünoomi. Info esitatakse kas 1. järku predikaatarvutuse abil või kasutatakse tõenäosuslikke esitusi, nt *Bayesi võrke*.

Masinõppe meetodeid võib liigitada ka vastavalt eelneva teadmuse kättesaadavusele. Enamus meetodeid eeldab, et agent alustab „puhtalt tahvlilt”. Statistilised õpimeetodid (Bayesi võrgud, tehiskäitumise võrgud) seevastu eeldavad, et agent teeb otsuseid teatavat **maailmamudelit** kasutades.

36. Tehisintellekti programmeerimiskeeled

Tehisintellekti programmeerimisel kasutatavad keeled võib jaotada kahte suurde klassi (Giarratano, Riley 2004):

- protseduraalsed (siia kuuluvad imperatiivsed ja funktsionaalsed keeled),
- mitteprotseduraalsed (mis jagunevad deklaratiivseteks, loogilisteks ja induktiivseteks keelteks).

Eraldi klassi moodustavad tehiskäitumise võrgud.

Protseduraalses keeles kirjutatud programmis spetsifitseerib programmeerija, kuidas täita käskude järjendit või arvutada funktsiooni, s.t programmeerija peab täpselt määratlema probleemilahenduse. Seevastu *mitteprotseduraalses* programmeerimiskeeles kirjutatud programmis spetsifitseerib programmeerija ainult eesmärgi, aga selle saavutamise viisi määratleb süsteem ise.

Protseduraalsed keeled

Imperatiivsed keeled (nagu C, osaliselt ka C+ ja Java) toetavad muutujaid, omistamisoperatsioone, kordusi, aga lisaks kasutavad ka selliseid vahendeid nagu rekursioon, protseduurid, moodulid, paketid jne. Imperatiivsed keeled asetavad rõhu juhtimisstruktuuridele ja seeläbi *ülalt alla* programmeerimisele. Tehisintellekti seisukohalt on imperatiivsetel keelteil oluline puudus: nad pole eriti efektiivsed sümboltootluseks. Ometi on neid keeli (nt C) kasutatud ekspertsüsteemide kestade kirjutamisel. Sellistes keeltes kirjutatud programmid töötavad tavalistel üldotstarbelistel arvutitel efektiivsemalt ja kiiremini kui varased ekspertsüsteemide kestad, mis loodi Lispis ja nõudsid spetsiaalset, Lispi jaoks kavandatud riistvara. Imperatiivsed keeled pole oma sekventsiaalse iseloomu tõttu reeglipõhiste ekspertsüsteemide programmeerimisel siiski eriti efektiivsed.

Lisp on *funktsionaalse* keele tüüpiline esindaja. Funktsionaalsed keeled võimaldavad probleeme modulariseerida. Nendel keelteil on kaks olulist omadust, mis on seotud modulaarsusega – kõrgemat järku funktsioonid ja laisk väärtustamine. Funktsionaalse programmeerimise idee on kombineerida lihtsaid funktsioone, et selle tulemusel saada keerulisemaid funktsioone. See on *alt üles* programmeerimine, vastukaaluks imperatiivsete keelte *ülalt alla* lähenemisviisile. Funktsionaalne programmeerimine keskendub funktsioonidele. Funktsionaalse keele osadeks on andmeobjektid (millega opereerivad funktsioonid), primitiivsed funktsioonid, funktsionaalsed vormid, mis võimaldavad sünteesida uusi funktsioone olemasolevatest, funktsioonidele rakendatavad operaatorid, mis

tagastavad väärtuse, nimeandmisprotseduurid, mis identifitseerivad uusi funktsioone. Oma väljatöötamisest alates on Lisp olnud juhtiv tehisintellekti keel. Paljud ekspertsüsteemid on loodud Lispis. Kuid tavalised arvutid ei täida Lispi eriti efektiivselt. Lispis kirjutatud ekspertsüsteeme on üldiselt raske lülitada millessegi muusse kui Lisp-programmi. Lihtsam on arvestada seda asjaolu kohe alguses, kui et pärast koodi ümber kirjutada. Portaabluse, efektiivsuse ja kiiruse huvides on praegu paljud ekspertsüsteemide tööriistad kas kirjutatud keeles C või konverteeritud sellesse. See välistab ka kalli spetsiaalse riistvara vajaduse Lispil põhinevate rakenduste korral.

Mitteprotseduraalsed keeled

Mitteprotseduraalses keeles koostatud programmis spetsifitseerib kasutaja eesmärgi, mida on vaja saavutada, ja süsteem määrab, kuidas seda teha.

Deklaratiivsetes keeltes tehakse vahet eesmärgi ja selle saavutamiseks kasutatavate meetodite vahel. Ekspertsüsteeme võib käsitleda kui deklaratiivset programmeerimist, sest programmeerija ei pea algoritmi tasemel ette andma, kuidas programm saavutab oma eesmärgi.

Objektorienteeritud paradigmasse kuuluvad programmeerimiskeeled, mis on osaliselt imperatiivsed ja osaliselt deklaratiivsed (C++, Java, C#). Põhiidee on programmi kavandamisel käsitleda programmis kasutatavaid andmeid kui objekte ja rakendada nendele objektidele operatsioone. Populaarne objektorienteeritud disaini meetod on UML.

Algselt kasutati mõistet objekt-orienteeritud programmeerimine selliste keelte puhul nagu Smalltalk, mis oli spetsiaalselt loodud objektidega manipuleerimiseks. Moodsates objektorienteeritud keeltes (C++, Java, C#) on sisseehitatud tunnused, mis toetavad sisseehitatud objekte. SMALLTALK on simuleerimiskeele Simula67 järglane. Keeles Simula67 võeti kasutusele klassi mõiste. Programmeerija peab teadma ainult objekti, mitte aga sisemisi detaile selle kohta, kuidas see on programmeeritud. Klass ei ole tüüp. Klassi näide (instants, *instance*) on andmeobjekt, millega saab manipuleerida. Instantsi mõiste on sisse toodud ka ekspertsüsteemides, kus ta tähistab fakti, mis sobib mustriks. Analoogiliselt öeldakse, et reegel on instantsieeritud, kui tema vasak pool on rahuldatud. „Aktiveeritud” ja „instantsieeritud” on reeglipõhistes süsteemides sünonüümid. Teine tähtis mõiste, mis sai alguse keelest Simula67, on pärimine, kus alamklass võib pärida teiste klasside (nn ülemklasside) omadusi. Pärimise mõistet saab laiendada ja selle abil organiseerida objekte hierarhiasse, kus objektid saavad pärida oma klassidelt, need omakorda oma klassidelt jne. Pärimine on kasulik seetõttu, et objektid võivad pärida klasside omadusi, ilma et programmeerija peaks iga omadust spetsifitseerima.

Loogilise paradigma üks esimesi rakendusi tehisintellektis oli loogikateoreemide tõestamine programmis Logic Theorist (autorid A. Newell, H. Simon ja J. C. Shaw). Selles programmis ja tema järglases GPS püüdsid autorid rakendada võimsaid algoritme, mis lahendaksid suvalise probleemi. Logic Theorist oli mõeldud ainult matemaatiliste teoreemide tõestamiseks, GPS aga suvaliste loogiliste probleemide lahendamiseks, sh mängud ja peamurdmülesanded. GPS oli esimene probleemilahendusprogramm, mis selgelt **eraldab probleemilahendusteadmuse ainevallateadmusest**. 1970. aastatel jõuti äratundmiseni, et arvutamine on spetsiaalset liiki mehaaniline, loogiline deduktsioon. Teoreemide tõestamiseks oli pöördtulekus („järelendus, kui tingimused”) piisavalt võimas. Tingimuste all võib mõelda mustreid, millega sobitatakse produktsioone. Sellisel kujul esitatud lauseid nimetatakse Horni lauseteks (sest esimesena uuris neid Ameerika matemaatik Alfred

Horn). 1972. a loodi keel PROLOG (Robert Kowalski, Alain Colmerauer, Philippe Roussel), et rakendada loogilist programmeerimist pöördtuletuses, kasutades Horni lauseid. Pöördtuletust saab kasutada nii deklaratiivseks teadmuse esituseks kui ka arutlusprotsessi juhtimiseks. Tüüpiliselt algab pöördtuletus sellega, et defineeritakse väiksemad alameesmärgid, mis peavad algeesmärgi saavutamiseks olema eelnevalt saavutatud. Seejärel lahutatakse need alameesmärgid omakorda väiksemateks alam-alameesmärkideks jne.

Näide deklaratiivsest teadmusest:

*Kõik inimesed on surelikud
Sokrates on inimene*

Selle saab esitada Horni lausetena:

*keegi on surelik
kui keegi on inimene*

*Sokrates on inimene
kui (igal juhul).*

Automaatne teoreemiteostaja saab neist lausetest järeldada, et Sokrates on surelik. Pöördtuletust kasutavate süsteemide üks eelis on see, et täitmine võib toimuda paralleelselt, s.t mitu protsessorit võivad erinevate alameesmärkide saavutamisel töötada üheaegselt.

PROLOG eeldab

- interpretaatorit (tuletusmootorit),
- andmebaasi (faktid ja reeglid),
- kujusobituse (*pattern matching*) ühte vormi, mida nimetatakse unifikatsiooniks,
- tagurdamise mehhanismi, et järgida alternatiivseid alameesmärke, juhul kui mingi eesmärgi saavutamine ebaõnnestub.

Üks loogilise programmeerimise eelis on täidetavad spetsifikatsioonid. See tähendab, et kui spetsifitseerida probleemi nõuded Horni lausetena, siis on tulemuseks täidetav programm. See on erinev tavaprogrammeerimisest, milles nõuete dokument ei ole üldsegi veel lõplik täidetav kood.

Erinevalt produktsioonireeglite süsteemidest on oluline mõju järjekorral, milles alameesmärgid, faktid ja reeglid on sisestatud PROLOG-programmi. Efektiivsus ja kiirus sõltuvad sellest, kuidas PROLOG sooritab otsingut oma andmebaasis. On programme, mis töötavad korrektselt, kui alameesmärgid, faktid ja reeglid on sisestatud ühel viisil, aga satuvad lõpmatusse tsükklisse, kui järjekord muutub.

Ekspertsüsteeme võib vaadata kui deklaratiivset programmeerimist, sest programmeerija ei pea algoritmi tasemel spetsifitseerima, kuidas programm saavutab oma eesmärgid. Reeglipõhises ekspertsüsteemis võib iga reegel saada aktiivseks ja paigutada agendasse (täitmisjärjekorda), kui tema vasak pool sobib faktidega. Reeglite sisestamise järjekord ei mõjuta seda, millised reeglid aktiveeritakse. Seega ei spetsifitseeri programmi lausete järjekord jäika juhtimisvoogu. Ekspertsüsteeme kasutatakse tavaliselt ebakindluse käsitlemiseks, mis võib tekkida nii sisendandmetes kui ka teadmusbasis eneses. Esmapilgul võib see tunduda arusaamatu, kuna oleme harjunud tavaprogrammeerimisega. Kuid paljud inimteadmised on heuristilised, s.t nad töötavad korrektselt vaid osa ajast. Sisendandmed võivad olla mittekorrektset, mittetäielikud, mittekooskõlalised ja muidu vigased. Algo-

ritmilised lahendused ei ole võimalised selliseid situatsioone käsitlema, sest algoritm kindlustab probleemi lahenduse lõpliku sammude seeria tulemusel. Sõltuvalt sisendandmetest võib ekspertsüsteem anda korrektse lahendi, hea lahendi, halva lahendi või üldse mitte anda lahendit. Hea ekspertsüsteem ei toimi halvemini kui parim probleemi-lahendaja – inimekspert –, aga võib toimida paremini. Kui meile oleks teada efektiivne algoritmiline meetod, mis on ekspertsüsteemist parem, siis me kasutaksime seda.

Induktiivne programmeerimine on realiseeritud masinõppe algoritmis ID3 (uuemad algoritmid on C4.5 ja C5.1). Selles paradigmas õpib programm näiteid üldistades. Üks selle paradigma rakendus on andmebaasi poole pöördumine. Selle asemel, et kasutaja peaks sisestama otsinguks ühe või mitme välja väärtusi, on vaja valida ainult üks või mitu sobivat näitevälja koos nende karakteristikutega. Andmebaasiprogramm tuletab andmetest karakteristikud ja otsib andmebaasist sobivat. Oracle jt relatsiooniliste andmebaaside juhtimissüsteemid, mis kasutavad päringukeelena struktureeritud inglise keelt (SQL), kasutavad otsingul mustrite äratundmist (kujutuvastust).

Mõned ekspertsüsteemide tööriistad pakuvad induktiivõpet, millega nad aktsepteerivad näiteid ja juhtumianalüüse ning genereerivad automaatselt reegleid.

Tehisnärvivõrgud

1980. aastatel tekkis uus programmeerimise paradigma – tehisnärvivõrgud. Seda paradigmat nim ka *konneksionistlikuks*, sest ta modelleerib probleemilahendust, treenides võrgustikuks seotud simuleeritud neuroneid. Praegu on tehisnärvivõrkudel palju rakendusi, nt nägude tuvastus, meditsiiniline diagnoosimine, videomängud, kõnetuvastus, kõneleja-tuvastus.

Idee on selles, et teha arvutusi, nagu seda teeb inimaju, mitte aga manipuleerida sümbolitega reeglite põhjal. Tehisnärvivõrk kasutab inimlaadset näidetest õppimise tehnikat. Ta on konfigureeritud spetsiifilise rakenduse jaoks, nt andmete klassifitseerimine või mustrite tuvastamine õppimisprotsessis, mida nimetatakse treenimiseks. Tehisnärvivõrke võib liigitada mitmel alusel. Kui treeningsisend ja -väljund on antud, siis on tegu juhendajaga õppega, vastupidisel juhul juhendajata õppega. Juhendajaga õppe näide on nägude tuvastamine. Kui aga pole teada, missugune peaks olema väljund, siis on tehisnärvivõrk hea klassifikaator grupeerimaks sisendeid, näiteks haiguspuhangute identifitseerimine.

Tehisnärvivõrke võib eristada ka vastavalt sellele, kuidas neuronid on omavahel seotud, kuidas nad teevad arvutusi, annavad edasi tegevusmustrid võrgus ja õpivad. Neid on rakendatud mitmesugustele reaalse maailma probleemidele. Nende eelis on see, et nad suudavad lahendada probleeme, mis on liiga keerulised tavaliste tehnoloogiate jaoks – probleeme, millel kas pole algoritmilist lahendust või on algoritmi defineerimine liiga keeruline. Üldiselt võib öelda, et tehisnärvivõrgud sobivad selliste probleemide puhul, mida inimesed lahendavad hästi, aga samas ei suuda selgitada, kuidas nad seda teevad, nt mustrite tuvastus ja ennustamine, mis nõuavad andmetes trendide äratundmist. Praegu kasutatakse laialdaselt **andmekäevandust** (*data mining*), otsimaks trende mineviku-andmetes, selleks et ennustada tulevikku.

Tehisnärvivõrgust võib mõelda kui analoogarvutist, mis kasutab lihtsaid omavahel kõrparalleelsel viisil ühendatud töötuselemente. Need elemendid sooritavad oma sisenditega lihtsaid Boole'i või aritmeetilisi operatsioone. Tehisnärvivõrgu toime põhineb

iga elemendiga seotud kaaludel. Kaalud esitavad süsteemis salvestatud infot. Inimaju sisaldab 10^{11} neuronit ja igal neuronil võib olla tuhandeid seoseid teistega. Neuroni sisendsignaalid korrutatakse kaaludega ja summeeritakse, et produtseerida kogu neuroni sisend. Kaale võib esitada maatriksina.

Tehisnärvivõrke ei programmeerita tavalisel viisil. Võrkude treenimiseks on olemas palju õppimisalgoritme. Programmeerija „programmeerib” närvivõrgu lihtsalt sel teel, et annab ette sisend- ja vastavad väljundandmed. Võrk õpib automaatselt, korrastades neuroneid ühendavate kaarte kaalusid. Kaalud ja neuronite läviväärtused määravad andmete edastamise võrgus ja seega tema korrektsuse vastuse treeningandmetele. Võrgu treenimine võib võtta tunde või päevi, sõltuvalt mustrite arvust, mida võrk peab õppima, ning tark- ja riistvarast. Kui aga õppimine on lõpetatud, annab võrk vastuse väga kiiresti.

Närvivõrk on hea valik, kui on palju empiirilisi andmeid, aga puudub piisava korrektsuse ja kiirusega algoritm.

Tehisnärvivõrkude abil on võimalik ehitada ekspertsüsteemi näiteks meditsiiniliseks diagnoosimiseks. Kuid see süsteem ei suuda ise selgitada lahendit.

Huvitav on märkida, et juba 1980. aastatel leidis tavalisel mikroarvutil töötav tehisnärvi-võrk väga hea lahendi rändkaupmehe probleemile 0,1 sekundiga, samal ajal kui suurarvuti abil optimaalse lahendi leidmine nõudis tervelt 1 tunni protsessori aega. Tehisnärvivõrk suudab lahendada 10 linna juhu niisama kiiresti kui 30 linna juhu, samal ajal kui tavaarvuti vajab oluliselt rohkem aega. 10 linna korral leidis tehisnärvivõrk ühe kahest parimast teekonnast ja 30 linna korral ühe esimese 100 miljoni hulgast. Kuigi tehisnärvivõrgud ei tarvitse alati anda optimaalset vastust, suudavad nad reaalajas anda hea lahendi. Paljudel juhtudel on parem saada 99,99999999999999% korrektne vastus 0,1 sekundiga kui 100% korrektne vastus 30 tunniga.

Riistvara

Tehisintellekti programmeerimiseks kasutatakse arvutustehnikat laias diapasoonis – suurarvutitest kuni personaalarvutiteni ja Lisp- või PROLOG-masinateni.

Esimesed arvutid, milles kasutati tehisintellekti kontseptsioone, realiseeriti Lisp-protssessorite baasil. Nt firma Lisp Machine arvuti Series III (hiljem Lambda) ning firma Symbolics arvuti LM-2 (hiljem Model) loodi MTI projekti järgi, olid turul alates 1981. Arvutid olid suurte mõõtmetega ja kallid.

USA-s toodeti 1980. aastatel kümnekonda tüüpi Lisp-masinaid. Euroopa maades leidsid samal ajal kasutust PROLOG-masinaid.

1990. aastate algul olid kasutatavatest arvutitest 60% personaalarvutid, 10 % Lisp-masinaid ja 5 % suurarvutid. Peab mainima ka Jaapani 5. põlvkonna arvuti projekti aastatel 1982–1992, mille eesmärgiks (mis küll jäi saavutamata) oli tehisintellektiga varustatud neuronarvuti loomine, ja selle jätkuprojekti 1993–1995. Nendes projektides kasutati paralleelset loogilist keelt KL1 ja kõrgharalleelset riistvara ning töötati välja paralleelne teoreemiteostaja MGTP ja teadmuse esituse keel QUIXOTE.

Praegu leiavad kasutust uued arvutiarhitektuurid, sh paralleelarvutid. Tehisintellekti edasine edu on kahtlemata seotud kvantarvutite ja pilvearvutustega, mis kasutavad masiivset arvutusvõimet.

Tehisintellekti ajalugu (lühikokkuvõte)

- 1943 Aju mudel Boole'i skeemina
- 1950 A. M. Turingi artikkel „Computing Machinery and Intelligence”
- 1952–69 Varane entusiasm
 - 1950d Esimesed TI programmid: kabemäng, Logic Theorist, Geometry Engine
Tehisnärvivõrgud
 - 1956 Dartmouthi seminar: termin „artificial intelligence”
 - 1957 Programm GPS
 - 1958 Programmeerimiskeel Lisp
 - 1965 J. Robinson: resolutsioonimeetod
J. Weizenbaum: inglise keeles suhtlev programm ELIZA
- 1966–74 TI tegeleb arvutuskeerukusega, tehisnärvivõrkude uurimine peaaegu lakkab
- 1969–79 Varased teadmuspõhised süsteemid
- 1975 M. Minsky: freim
- 1980–89 Ekspertsüsteemide buum
Lisp-masinad turul
- 1985–95 Tehisnärvivõrgud muutuvad jälle populaarseks
- 1988– Tõenäosuse taaselustumine; tehnilise sügavuse kasv, TI „kuumad” teemad: tehiselu, geneetilised algoritmid, pehmed arvutused (soft computing)
- 1995– Intelligentseid agendid
Intelligentne infootsing veebist
- 2000– Robot-mänguasjad turul
- 2010– intelligentsed seadmed kodumajapidamises, autojuhiabilised, diagnostika-süsteemid meditsiinis, portatiivsed masintõlgid, nägude tuvastamise süsteemid jpm.

Võtmeküsimused läbi aegade

- 1957–62 Heuristiline programmeerimine
Masintõlge
Teoreemide automaatne tõestamine
GPS
Kujuvõrdlus
Lisp
- 1963–67 Loomuliku keele mõistmise süsteemid
Resolutsioonimeetod
ELIZA
- 1968–72 Mikromaailmad (nt „kuubikute maailm”)
DENDRAL (ekspertsüsteem)
SHRDLU (robot)
- 1973–77 Default-loogika
Freim, skript
Semantilised võrgud
TI programmeerimiskeeled: FRL jt.
Produksioonide süsteemid
MYCIN (ekspertsüsteem)
Hägusa teadmuse esitus
- 1978–82 Metasüsteemid
Ekspertsüsteemide „kestad”
- 1982 ja edasi Suured projektid, sh
– Jaapani 5. põlvkonna arvuti
– USA raamprogramm DARPA
– Euroopa Liidu raamprogrammid: ESPRIT, Copernicus
Kommertssüsteemid
Spetsiaalne riistvara
Tehisnärvivõrgud
- 1994 ja edasi Agenditehnoloogia
(intelligentsed agendid)

KIRJANDUS

James F. Allen (1994). *Natural Language Understanding*, 2nd Edition. Benjamin Cummings.

Chin-Liang Chang, Richard Char-Tung Lee (1973). *Symbolic logic and mechanical theorem proving (Computer Science Classics)*. Academic Press.

Noam Chomsky (1956). Three models for the description of language. *IRE Transactions on Information Theory*, vol 2, issue 3, pp. 113–124.

Joseph Giarratano, Gary Riley (2004). *Expert Systems: Principles and Programming*, 4th Edition. Thomson Course Technology.

Matt Ginsberg (1993). *Essentials of Artificial Intelligence*. Morgan Kaufmann Publ.

Tiit Hennoste, Andriela Rääbis (2004). *Dialogiaktid eesti infodialoogides: tüpoloogia ja analüüs*. Tartu: TÜ.

Daniel Jurafsky, James H. Martin (2008). *An introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. 2nd Edition. Prentice Hall.

Fred Karlsson (2002a). Üldkeeleteadus. Eesti Keele Sihtasutus.

Fred Karlsson (2002b). Is there an upper limit to right-branching embedding of clauses? Toim. R. Pajusalu, T. Hennoste. *Tähendusepüüdja / Catcher of the Meaning*. Tartu Ülikooli üldkeeleteaduse õppetooli toimetised 3, 196–199, TÜ.

Mare Koit (1989). *Resolutsioonimeetod*. Tartu: TRÜ.

Nils J. Nilsson (1982). *Principles of Artificial Intelligence*, San Francisco: Morgan Kaufmann.

Barbara H. Partee, Alice G. B. ter Meulen, Robert E. Wall (1990). *Mathematical Methods in Linguistics*. Kluwer Academic Publishers.

György Pólya (2001). *Kuidas seda lahendada*. Tallinn: Valgus.

Raymond Reiter (1980). A logic for Default-Reasoning. *Artificial Intelligence*, 1980, v. 13, N 1–2, pp 81–132.

Raymond Reiter (1978). On closed world data bases. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pp 119–140. New York: Plenum.

Elaine Rich, Kevin Knight (1991). *Artificial intelligence*. New York [etc.] : McGraw-Hill.

Tõnu Tamme, Tanel Tammet, Rein Prank (2002). *Loogika. Mõtlemisest tõestamiseni*. TÜ.

Stuart Russell, Peter Norvig (2003). *Artificial Intelligence: A Modern Approach*. 2nd ed., Prentice Hall Series in Artificial Intelligence.

Tanel Tammet (1987). Default-loogika ja selle rakendamise võimalusi dialoogsüsteemides. TRÜ Toimetised, 795. *Dialogi mudelid ja eesti keel. Töid eesti filoloogialt*. Tartu, lk 95–105.

Alan M. Turing (1950). Computing machinery and intelligence. *Mind*, 59, 433–460. <http://www.loebner.net/Prizef/TuringArticle.html> (18.02.2010)

Jason D. Williams, Steve Young (2007). Partially Observable Markov Decision Processes for Spoken Dialog Systems. *Computer Speech and Language*, 21, pp. 393–422.

Patrick H. Winston (1992). *Artificial Intelligence (3rd Edition)*. Addison Wesley.

Т.А. Гаврилова, В.Ф. Хорошевский (2000). Базы знаний интеллектуальных систем. Учебник. СПб: Питер.

С. Осуга (1989). Обработка знаний. М: Мир.

Д.А. Поспелов (1989). Моделирование рассуждений. Опыт анализа мыслительных актов. М: Радио и связь.

Э. Тыугу (1984). Концептуальное программирование. М: Наука.

MÄRKSONAD

| | | | |
|--|---------------------------|---|--------------------------------------|
| 5. põlvkonna arvuti | 9, 207, 209 | kitsenduste rahuldamise probleem | 134 |
| A* | 82–88, 180 | KL-ONE | 35, 51 |
| AKO | 44, 47–49 | Knight, Kevin | 3, 19, 52, 194, 197, 198, 210 |
| Alice | 7, 114, 185, 210 | KRL | 51 |
| asendus | 115, 116, 119, 122 | kujuvõrdlus (kujutuvastus) | 47 |
| Boole, George | 53 | kuuluvusfunktsioon | 63, 64 |
| CADE | 125 | kõnesüntees | 180, 181 |
| CASC | 125 | kõnetuvastus | 187, 206 |
| dialogikorpus | 190 | Lenat, Douglas B. | 194 |
| dialogüsteem | 189 | libalöömutamine | 92 |
| TRAINS | 187, 189 | Lisp | 9, 195, 203, 207–209 |
| Verbmobil | 182, 187, 189 | Loebner, Hugh | 7, 185 |
| VirtualHuman | 189 | loogiline jäeldamine 55–57, 105, 107, 112, 114, | |
| disjunkt .. 105, 110, 111, 113, 115, 119, 120, 132 | | 118, 123–126, 132 | |
| fundamentaalnäide | 111 | lubatav algoritm | 82, 83, 85, 86, 88, 139 |
| tühidisjunkt ... 105, 113, 114, 119, 120, 125, | | maailma mudel | 8, 15 |
| 126, 132 | | masintõlge | 6, 172 |
| ECAI | 11 | McCarthy, John | 7–9, 60, 61 |
| ECCAI | 12 | McKinstry, Chris | 16 |
| edasivaatav kontroll | 137–139 | Michie, Donald | 8 |
| ekspertsüsteem 13, 14, 16, 36, 37, 46, 143, 206, | | minimax | 95, 96, 100 |
| 209 | | Minsky, Marvin | 48, 141, 142, 208 |
| DENDRAL | 42, 209 | neurokübernetika | 9, 21 |
| MYCIN | 13, 40, 42, 209 | Newell, Allen | 7, 204 |
| Fikes, Richard | 15 | NIKL | 51 |
| freim | 48, 51, 52, 132, 133, 208 | Nilsson, Nils | 8, 15, 79, 82, 210 |
| eksemplarfreim | 50, 52 | Norvig, Peter | 3, 8, 134, 195, 210 |
| FRL | 35, 50, 51, 209 | Ohsuga, Setsuo | 68 |
| generatiivne grammatika | 42, 175 | olekute ruum | 17, 26–29, 38–40, 71, 79, 104, |
| Ginsberg, Matt | 3, 35, 76, 195, 210 | 134, 180, 187 | |
| GRAPHSEARCH | 79, 81–83, 85, 87 | olek | 38, 39, 72, 73, 79, 89–91, 104, 134, |
| harude ja tõkete meetod | 30 | 136, 154, 155, 158, 167, 178 | |
| Hayes, Paul | 8, 17, 69 | otsing .. 17, 18, 22, 25, 26, 28, 30, 40, 47, 71, 73, | |
| HEARSAY | 42, 182 | 76, 77, 81, 87, 89, 98–100, 136–138, 141, | |
| Herbrandi baas | 107–110, 113 | 142, 172, 180, 194 | |
| Herbrandi teoreem | 111–113, 122 | heuristik | 30, 88, 139 |
| Herbrandi universum | 107–109, 111–113 | heuristiline funktsioon | 85, 134 |
| H-interpretatsioon | 107–111, 113 | heuristiline otsing | 194 |
| Hopfield, John | 9 | hinnangufunktsioon | 82, 89, 96 |
| ICGA | 18 | otsetuletus | 71 |
| ID3 | 202, 206 | tagasipöördumisega otsing | 137 |
| IDA* | 87, 88 | otsinguprobleem | 104 |
| IJCAI | 11, 18 | otsustuspuu | 196, 200, 201 |
| intellekt | 8, 10, 18, 123 | PARRY | 186 |
| intellektitehnika | 6–10 | piiramise tehnika | 60 |
| IS-A | 44, 47–49 | pimeotsing | 73 |
| juhtimisstrateegia | 30, 47, 120 | iteratiivne laienemine | 78 |
| juturobot | | iteratiivne laskumine | 77, 78, 87, 88 |
| ELIZA | 183–185, 208, 209 | laiutiotsing | 29, 74, 76, 77, 81, 84, 121 |
| KEE | 49 | süvitsiotsing | 29, 30, 73, 76, 77, 81, 104 |
| keele mudel | 180 | plaan kolmnurktabelina | 152 |
| kitsenduste graaf | 135, 139 | planeerimine 13, 17, 18, 141–143, 153, 159, 165 | |
| kitsenduste levitamine | 137 | Pólya, György | 31, 194, 210 |

| | | | |
|---|--|--------------------------------------|--|
| Pospelov, D.A. | 40 | teadmus .. | 14, 34, 43, 47, 52, 59, 60, 62, 70, 125, 129, 132, 133, 179 |
| Post, Emil | 35 | teadmuse esitus | 17, 18, 21, 32, 34–36, 41–43, 47, 48, 51, 52, 57, 58, 63, 67–69, 71, 131, 169, 197, 198, 205, 207, 209 |
| PRIZ | 45, 46 | tehisintellekt | 6, 9, 26 |
| probleemi osandamine | 104 | tehisintellekti tehnika | 19, 20, 22, 25 |
| probleemilahendus | 8, 26, 30, 169, 171, 191 | tehisintellektisüsteem | 57, 189, 194 |
| produktsoon | 38, 39 | AM | 51, 194, 195 |
| PROLOG | 36, 46, 53, 205, 207 | Cyc | 16, 189 |
| pärimine | 44, 47, 49, 131, 134, 204 | DCOMP | 159, 160, 162 |
| pöördtuletus | 204 | GAC | 16 |
| Quillian, M. Ross | 43 | GPS | 7, 204, 208, 209 |
| Refal | 36 | tehisnägemine | 10, 193 |
| Reiter, Raymond | 60, 61, 210 | tehisnärvivõrk | 9, 16, 182, 199, 206, 207 |
| resolvent | 114, 118, 119, 124–127, 132 | tajur | 182 |
| Rich, Elaine | 3, 8, 19, 52, 194, 197, 198, 210 | teoreemi tõestamise probleem | 56, 113, 123, 125, 126, 129, 179 |
| robot ... | 13–15, 17, 143, 151, 152, 160, 162, 192, 209 | trips-traps-trull | 6, 19, 21, 94, 95 |
| ABSTRIPS | 165–168 | Turing, Alan M. | 7, 52, 95, 185, 210 |
| Aibo | 15 | Turingi test | 7, 8, 10, 185, 192 |
| Asimo | 15 | Tõugu, Enn | 12, 45 |
| Furby | 15 | unifikaator | 115, 116, 119 |
| RSTRIPS | 153–159 | lihtsaim unifikaator | 115–119, 122 |
| SHRDLU | 15, 185, 186, 209 | unifitseerimisalgoritm | 116 |
| STRIPS ... | 15, 143, 145, 147, 150, 153, 154, 159, 165, 166, 168 | valem . | 54–57, 62, 105–107, 111–113, 120, 122–125, 149 |
| Rosenblatt, Frank | 9, 182 | samaselt tõene valem | 109 |
| Russell, Stuart | 3, 8, 134, 195, 210 | samaselt väär valem | 105 |
| rändkaupmehe probleem | 9, 47, 98, 207 | Wallace, Richard | 185 |
| SCAI | 12 | Wilks, Yorick | 192 |
| selektiivgenereerimine (genereeri ja testi) | 89 | Winograd, Terry | 15, 51, 185 |
| semantiline puu | 110, 111 | VoiceXML | 188 |
| semantiline võrk | 43, 47, 48, 129, 130 | õppimine ... | 8, 9, 18, 70, 181, 194, 195, 198, 207 |
| Shaw, J.C. | 7, 204 | induktiivõpe | 18, 194, 202 |
| Simon, Herbert | 7, 204 | masinõpe (tehisõpe) | 18 |
| SNePS | 46 | piiratud hüpoteesiruumi eeldus | 196, 197 |
| suhtlemine | 192 | | |
| Zadeh, Lofti A. | 63 | | |
| Tammet, Tanel | 61, 210 | | |

