

Vilniaus universitetas
Matematikos ir informatikos fakultetas
Programų sistemų katedra

Vytautas Čyras

INTELEKTUALIOS SISTEMOS

Vilnius
2012

Turinys

Pratarmė	5
Pirma dalis	6
1. Dirbtinio intelekto samprata.....	7
1.1. <i>Dirbtinio intelekto istorija</i>	7
1.2. <i>Dirbtinio intelekto klasifikacija ir tyrimo objektas</i>	8
1.3. <i>Žodžio „intelektualus“ samprata dirbtiniame intelekte</i>	10
1.4. <i>Hanojaus bokštai</i>	11
1.5. <i>Dirbtinio intelekto sampratos kitimas</i>	15
1.6. <i>Uždavinių sprendimo metodų principai</i>	16
2. Dirbtinio intelekto sistema kaip produkcių sistema	18
3. Valdymas su grįžimais ir procedūra BACKTRACK.....	26
3.1. <i>Predikato DEADEND pavyzdys</i>	28
4. Valdovių išdėstymo šachmatų lentoje uždavinys	30
5. Euristika	33
5.1. <i>Euristika valdovių uždavinyje</i>	34
5.2. <i>Dar viena euristika: valdovę statyti žirgo éjimu</i>	38
6. Patobulinta paieškos su grįžimais strategija. Procedūra BACKTRACK1.....	51
7. Uždavinys apie labirintą. Kelio paieška į gylį	53
7.1. <i>Kelio paieškos labirinte algoritmai BACKTRACK ir BACKTRACK1 variantai</i>	56
7.2. <i>Kai kurios produkcijų algebrinės savybės</i>	58
8. Kelio radimas labirinte paieškos į plotį būdu – bangos metodas.....	59
9. Paieškos į plotį algoritmas grafe be kainų	64
10. Algoritmas paieškai grafe, kai briaunos turi kainas.....	69
11. Paieškos į gylį algoritmas grafe be kainų	72
11.1. <i>Paieškos „i plotį“ skirtumas nuo „i gylį“</i>	75
11.2. <i>Sprendėjas ir planuotojas</i>	76
12. Prefiksinė, infiksinė ir postfiksinė medžio apéjimo tvarka	77
13. Bendras paieškos grafe algoritmas GRAPHSEARCH	82
14. Algoritmų BACKTRACK1 ir GRAPHSEARCH skirtumas.....	83
14.1. <i>Paieška grafe</i>	83
14.2. <i>Pavyzdys paieškai labirinte su salomis</i>	85
14.3. <i>Kontrpavyzdys – labirintas, kuriame geresnį rezultataą duoda BACKTRACK1</i>	89
15. Valdymo metodas „kopimas į kalną“	91
16. Manheteno atstumas	97
17. Algoritmo A* samprata	101
17.1. <i>A* pavyzdys, kai euristinė funkcija $h(n)$ yra Manheteno atstumas</i>	101
17.2. <i>A* pavyzdys keliu žemėlapyje</i>	105

18.	Tiesioginis ir atbulinis išvedimas produkcijų sistemoje	107
18.1.	<i>Tiesioginio išvedimo algoritmas</i>	107
18.2.	<i>Atbulinio išvedimo algoritmas</i>	109
18.3.	<i>Programų sintezės elementai</i>	111
18.4.	<i>Perteklinės produkcijos tiesioginiame išvedime</i>	113
18.5.	<i>Perteklinės produkcijos atbuliniame išvedime</i>	114
18.6.	<i>Tiesioginio išvedimo sudėtingumas</i>	115
19.	Rezoliucijų metodas	116
19.1.	<i>Irodymo remiantis rezoliucijos taisykle pavyzdys</i>	119
19.2.	<i>Pavyzdys: trys produkcinės taisyklės</i>	121
19.3.	<i>Pavyzdys: rezoliucijos naudojimas teoremu įrodymui</i>	124
20.	Ekspertinės sistemos	126
21.	Internetinė parduotuvė	132
22.	Tiuringo testas	141
23.	Egzamino bilietai	143
	Antra dalis	144
24.	Žinių vaizdavimas kaip dirbtinio intelekto šaka	145
25.	Duomenys, informacija ir žinios	147
26.	Žmogiškasis pažinimas ir žiniomis grindžiamos sistemos	150
27.	Žinių vaizdavimo klasifikavimas į procedūrinį ir deklaratyvų	152
27.1.	<i>Procedūrinis žinių vaizdavimas</i>	152
27.2.	<i>Deklaratyvus žinių vaizdavimas</i>	153
27.3.	<i>Įeities – išeities vaizdavimo pavyzdys</i>	158
28.	Žinių vaizdavimas ir žinių vadyba	159
29.	Logikos vaidmuo samprotavime	161
30.	Žinių vaizdavimo būdų apžvalga	164
30.1.	<i>Loginis žinių vaizdavimas</i>	164
30.2.	<i>Procedūrinis žinių vaizdavimas (siauraja prasme)</i>	164
30.3.	<i>Struktūrinis žinių vaizdavimas</i>	164
30.4.	<i>Tinklinis žinių vaizdavimas</i>	164
31.	Semantinis tinklas	167
32.	Bendrinė ir individuali sąvoka	171
32.1.	<i>Ryšio is-a interpretacijos</i>	172
32.2.	<i>Ryšio instance-of interpretacijos</i>	178
33.	Klasifikavimo vaizdavimas	180
33.1.	<i>Algebrinis požiūris į klasifikavimą</i>	181
33.2.	<i>Santykio sąvoka</i>	182
33.3.	<i>Tiesinės erdvės faktorizavimas pagal tiesinį poerdvi</i>	186
34.	Semantiniai tinklai pagal Russell ir Norvig	190
34.1.	<i>Paveldėjimas</i>	191

34.2.	<i>Santykis ir ryšys</i>	193
34.3.	<i>Atvirkštinis santykis</i>	194
35.	Koncepciniai modeliai duomenų bazėse	201
36.	Žinių vizualizavimas	207
37.	Teisės informatika	209
38.	Argumentavimas teisėje	213
38.1.	<i>Trys bylos</i>	214
38.2.	<i>Svoriu suteikimas vektoriaus koordinatėms</i>	215
38.3.	<i>Ar penki faktoriai lygiaverčiai?</i>	217
38.4.	<i>Formalizavimas remiantis faktoriais</i>	219
38.5.	<i>Formalizavimas remiantis vertybėmis</i>	222
38.6.	<i>Trečioji teorija, sukonstruota remiantis hipotetine byla</i>	228
39.	Literatūra	232

Vadovėli išleisti rekomendavo VU Matematikos ir informatikos fakulteto taryba
(2010-04-13, protokolo Nr. 6)

Vytautas Čyras. Intelektualios sistemos. Elektroninė knyga. ISBN 978-9955-33-561-0.
Prieiga internetu: <<http://www.mif.vu.lt/~cyras/konspektas-intelektualios-sistemos.pdf>>. 2012-12-26

Pratarmė

Ši mokymo medžiaga yra parengta ir kaip skaitomo kurso „Intelektualios sistemos“ paskaitų konspektas. Remiamasi atskirais kursais „Dirbtinis intelektas“ ir „Žinių vaizdavimas“. *Dirbtinis intelektas* (DI) gali būti suprantamas kaip tarpšakinė disciplina, kurioje susipynusios informatikos, matematikos, technikos mokslų, filosofijos ir kitų mokslų gijos. Pasaulyje yra visuotinai pripažintų, bet skirtinį požiūrių į DI. Šioje mokymo medžiagoje DI pristatomas kaip *informatikos* (angl. *computer science*) šaka, o *Žinių vaizdavimas* (ŽV) kaip DI sudėtinė dalis.

Kaip išvardinti mokslai turi savo specifiką, taip ir DI bei ŽV turi joms būdingą *dvasią* (angl. *spirit*), kurią ir siekiamas pristatyti šia medžiaga. Supaprastintai dalyko dvasia yra tai visa tai, kas slypi už teksto. Tačiau problemos esmė, kaip tą dvasią pavaizduoti.

Autorius gina požiūrį, kad DI paskirtis yra visų pirma tarnauti žmoniškajam pažinimui, o tik paskui intelektualizuotų sistemų kūrimui. Žodžių „žinios“ ir „pažinimas“ bendra šaknis apibūdina autoriaus požiūrio giluminį pradą ir sieki.

DI dėstomas dvasia, kuri vyrauja Nilso Nilsono knygoje „Dirbtinio intelekto principai“ (1982) anglų kalba ir jos vertime į rusų kalbą. Tiesa, kurse betarpiskai remiamasi tik pirmaisiais trimis knygos skyriaus. Kita įtakojanti knyga yra [Lugger 2005].

Dėstoma jau klasika tapusi medžiaga. Vadovaujamas Nilsono požiūriu – dirbtinis intelektas kaip produkcijų sistema. *Dirbtinio intelekto sistema* yra suprantama kaip trejetas: *globali duomenų bazė, produkcijų aibė ir valdymo sistema*. Kurse pagrindinis dėmesys yra skiriamas uždavinių sprendimui naudojant paiešką (angl. *problem solving by search*).

Žinių vaizdavimo kaip ir dirbtinio intelekto sąvoka yra siejama su žodžiu „intelektualus“. Kalbant apie žmogaus kuriamus artefaktus ir sistemas geriau tinka „intelektualizuotos. Pastarąjį autorius supranta kaip programų sistemą, kurioje naudojami DI principai. Tokiu būdu žiūrima per programų sistemų inžinerijos ir DI prizmę.

Autorius vadovaujasi matematiniu metodu, tačiau siekia atsižvelgti į metodus, naudojamus socialiniuose ir humanitariniuose moksluose. Siekiama pateikti fundamentalią medžiagą. Tekste remiamasi kitais šaltiniais ir daug cituojama – vadovaujamas humanitarinių ir socialinių mokslų tradicija.

Sąvokai „dirbtinio intelekto dvasia“ galima pasiūlyti tokią pačią sampratą kaip dirbtinio intelekto principų visuma. DI dvasios išdėstymas yra DI principų atskleidimas.

Kuo skiriasi metodas nuo principo? Metodas taikomas esant apibrėžtai sąlygų sudėčiai. Principas paprastai taikomas kai situacija nėra aiški. Jeigu uždaviniui išspręsti nėra žinomas metodas, tai tenka vadovautis principais.

DI jauna disciplina – sukako 50 metų – ir negali pasigirti tokiomis senomis principų tradicijomis kaip, pavyzdžiu, filosofija arba teisė. Yra teisės principų, kurių amžius keli tūkstantmečiai, žinomų iš senovės Graikijos ir Romos imperijos civilizacijų: *Istatymui visi lygūs* (Aristotelis); *Tebūnie išklausyta ir antroji pusė* (*Audiatur et altera pars*).

Rengiant tekštą talkino nemažai studentų, kurie klausėsi paskaitų ir konspektavo. Valdas Birbilas, Antanas Vilimas, Indrė Lukauskaitė ir Marek Meškevič buvo pirmieji, užrašę paskaitų turinį elektronine forma. Toliau talkino Vera Afanasjeva, Dalius Masiliūnas, Laimis Vaikus, Rimantas Žakauskas, Eglė Saulėnaitė, Aivaras Ruzveltas, Vaidas Bielskis, Dainius Kunigauskas ir kiti. Šis konspektas iš viso nebūtų sukurtas, jeigu ne Justo Arasimavičiaus kūrybinis darbas konspektuojant paskaitų metu ir iš rankraščio perkeliant į failą. Toliau konspekto versiją tobulino ir daugiausiai prisidėjo Edgaras Abromaitis. Atsakomybė už netikslumus ir klaidas tenka V. Čyrui kaip skaitomo kurso ir konspekto autorui.

Pirma dalis

1. Dirbtinio intelekto samprata

1.1. Dirbtinio intelekto istorija

2006 metais buvo minima termino „dirbtinis intelektas“ (angl. *artificial intelligence*, AI) 50 metų sukaktis, (žr. <http://www.ki2006.fb3.uni-bremen.de/50years.htm>). Šio termino sudarymas priskiriamas Džonui Makarčiui (John McCarthy) ir laikoma, kad pirmą kartą buvo jo įvestas 1956 metais JAV Dartmuto mieste (*Dartmouth*) vykusioje konferencijoje (žr. taip pat [McCarthy 1958]). Nuo to laiko dirbtinio intelekto (DI) samprata visą laiką buvo plečiama. Ankstyvojo DI tyrimų kryptys tapo savarankiškomis DI arba informatikos šakomis ir šiuo metu gali būti tik istoriškai siejamos su DI. Per trumpą DI istoriją progresas šioje srityje buvo lėtesnis negu pirmieji lūkesčiai. Tačiau DI tyrimuose sukurtos programas įtakojo pažangą kitose srityse, pvz., informacijos paiešką internete, žmogiškojo supratimo automatizavimo ribų supratimą.

Kadangi dirbtinis intelektas yra siejamas su matematine logika, o matematinė logika siejama su filosofijos šaka logika, kurios pradininkais laikomi senovės Graikijos mąstytojai Sokratus, Platonas, Aristotelis ir kt., tai mes DI galėtume sieti su senovės Graikija. Tačiau tokis požiūris nėra visuotinai priimtas.

DI gali būti siejamas su elektroninio kompiuterio sukūrimu 194X-aisiais metais ir tolimesniais technologiniais tyrimais. Tradiciškai DI siejamas su Alano Tiuringo (*Alan Turing*) iškeltu klausimu „ar gali mašina mąstyti?“ [Turing 1950; Tiuringas 1961]. Todėl A. Tiuringas yra laikomas vienu iš DI pradininkų.

Ivairių autorų DI apibrėžimai rodo skirtingus požiūrius į DI ir užduočių motyvaciją.

Dirbtinis intelektas yra informatikos (angl. *computer science*) šaka, nukreipta į kūrimą tokią mašiną, kurios gali užsiimti elgesiu, žmonių laikomu protingu. Galimybė sukurti „protinges“ mašinas sudomino žmones nuo senų laikų. Šiaisiai laikais po 50 metų DI programavimo metodų tyrimų svajonės apie sumanias mašinas tampa realybė. Tyrėjai kuria sistemas, kurios gali imituoti žmogaus mąstymą, suprasti kalbą, nugalėti geriausius žmones-šachmatininkus ir pasireiškia kitokiu „išradingu“.

1956 metų Dartmuto konferencijos organizatorius Marvinas Minskis (*Marvin Minsky*) 2006 metais DI 50-mečio minėjime Bremeno universitete šitaip pristato ankstyvojo DI tyrimų kryptis, (žr. http://www.ki2006.fb3.uni-bremen.de/pdf/2_minsky.pdf) :

- 1957 Arthur Samuel: A machine that plays master-level Checkers.
- 1957 Newell, Shaw and Simon. Proving logical theorems.
- 1960 Herbert Gelernter: Proving theorems in Geometry
- 1960 James Slagle: Symbolic Integral Calculus
- 1963 Lawrence G. Roberts: 3-D Visual Perception
- 1964 Thomas G. Evans: Solving Geometry Analogy problems.
- 1965 Daniel Bobrow: Solving word problems in Algebra
- 1969 Engelman, Martin and Moses: the MACSYMA project.
- 1969 Minsky & Papert: A robot builds structures with wooden blocks.
- 1970 Patrick Winston: A robot learns to recognize such structures.
- 1970 Terry Winograd: A program understands many sentences.
- 1972 Sussman: A program recognizes some bugs in programs.
- 1974 Eugene Charniak: A program understands some simple stories.

1.2. Dirbtinio intelekto klasifikacija ir tyrimo objektas

Mes skaitydami paskaitas į DI žiūrime kaip į informatikos šaką. Yra ir kitokių nuomonių. Koks dirbtinio intelekto tyrimo objektas? Ką tūliai dirbtinis intelektas kaip mokslo disciplina? Apie ką kalba DI? Atsakymus pateikiame citatose.

Olandijos leidykla „Kluwer Academic Publishers“ (dabar Springer) 2003 m. pateikė tokią informatikos klasifikaciją ir DI vietąjoje:

Computer and Information science

- Algorithms and computer theory – algoritmai ir kompiuterių teorija
- **Artificial intelligence (AI) – dirbtinis intelektas (DI)**
- Computer architecture – kompiuterių architektūra
- Computer – human interaction – kompiuterio ir žmogaus sąveika
- Computing – kompiuterija
- Data base systems – duomenų bazių sistemos
- Data communication networks – duomenų perdaravimo tinklai
- Design automation – projektavimo automatizavimas
- Electronic commerce – elektroninė komercija
- Electronic publishing – elektroninė leidyba
- General and management topics – bendrosios ir valdymo temos
- Information retrieval – informacijos gavimas
- Multimedia system and application – daugialypės terpės sistemos ir taikomosios programos
- Neural networks – neuroniniai tinklai
- Operating systems – operacinės sistemos
- Programming languages – programavimo kalbos
- Real-time systems – realaus laiko sistemos
- Software engineering – programų sistemų inžinerija
- System and control theory – sistemų ir valdymo teorija

Dirbtinis intelektas dar skirstomas į šias šakas:

- Artificial intelligence languages – dirbtinio intelekto kalbos
- Agents – agentai
- Cognitive science – kognityvinis mokslas (sinonimas pažinimo mokslas)
- Computer vision – kompiuterinis matymas
- Expert systems – ekspertinės sistemos
- Genetic programming and algorithms – genetiniai algoritmai ir programavimas
- **Knowledge representation – žinių pavaizdavimas**
- Machine learning – mašinų mokymasis
- Machine translation – mašininis vertimas
- Natural languages – natūraliosios kalbos
- Robotics and automation – robotika

Springer leidyklos tinklapyje (žr. <http://www.springer.com/>) pateikiama yra kiek kitokia informatikos klasifikacija:

Computer Science

- **Artificial Intelligence**
- Bioinformatics
- Communication Networks
- Database Management & Information Retrieval
- General Issues
- Hardware
- Human-Computer Interaction (HCI)
- Image Processing Information Systems and Applications
- Media Design
- Security and Cryptology
- Software Engineering
- Theoretical Computer Science

Žemiau pateikiama DI vieta JAV priimtoje kompiuterijos (angl. *computing*) šakų klasifikacijoje (žr. <http://www.acm.org/class/1998/>):

- A. General Literature – bendra literatūra
- B. Hardware – techninė įranga
- C. Computer Systems Organization – kompiuterinių sistemų sandara
- D. Software – programinė įranga
- E. Data – duomenys
- F. Theory of Computation – programavimas matematinėmis mašinomis
- G. Mathematics of Computing – kompiuterijos matematika
- H. Information Systems – informacinės sistemos
- I. Computing Methodologies – kompiuterijos metodikos
- J. Computer Applications – kompiuteriniai taikymai
- K. Computing Milieux – apie kompiuteriją

Dirbtinis intelektas yra šakoje „I. Computing Methodologies“:

- I.0 GENERAL – bendroji dalis
- I.1 SYMBOLIC AND ALGEBRAIC MANIPULATION – simboliniai ir algebriniai skaičiavimai
- I.2 ARTIFICIAL INTELLIGENCE – dirbtinis intelektas**
- I.3 COMPUTER GRAPHICS – kompiuterinė grafika
- I.4 IMAGE PROCESSING AND COMPUTER VISION – vaizdų apdorojimas ir kompiuterinis matymas
- I.5 PATTERN RECOGNITION – struktūrų atpažinimas
- I.6 SIMULATION AND MODELING – imitavimas ir modeliavimas
- I.7 DOCUMENT AND TEXT PROCESSING – dokumentų ir teksto apdorojimas

Dirbtinis intelektas yra skaidomas į šakas, tarp kurių yra „žinių vaizdavimas“. Mes laikomės tokios „žinių vaizdavimo“ sampratos, kokia yra šakoje „žinių vaizdavimo formalizmai ir metodai“ (žr. žemiau).

Yra galima laikytis ir kitokios disciplinos „žinių vaizdavimas“ sampratos, nes terminai „žinios“ ir „vaizdavimas“ gali būti aiškinami įvairiai. Pavyzdžiu, informacijos ir komunikacijos moksluose terminas „žinios“ yra interpretuojamas kitaip negu informatikoje.

Dar kitokia termino „žinios“ samprata yra vadyboje, tiksliau, *žinių vadyboje* (angl. *knowledge management*).

I.2 ARTIFICIAL INTELLIGENCE

I.2.0 General – bendra dalis

I.2.1 Applications and Expert Systems – aplikacijos ir ekspertinės sistemos

I.2.2 Automatic Programming – automatinis programavimas

I.2.3 Deduction and Theorem Proving – dedukcija ir teoremų įrodynėjimas

I.2.4 **Knowledge Representation Formalisms and Methods** – žinių vaizdavimo formalizmai ir metodai

Frames and scripts – freimai ir skriptai

Modal logic – modalumų logikos

Predicate logic – predikatų logika

Relation systems – sąryšių sistemos

Representation languages – vaizdavimo kalbos

Representations (procedural and rule-based) – procedūrinis ir taisyklemis grindžiamas vaizdavimas

Semantic networks – semantiniai tinklai

Temporal logic – laiko logika

I.2.5 Programming Languages and Software – programavimo kalbos ir programinė įranga

I.2.6 Learning – kompiuterio apmokymas

I.2.7 Natural Language Processing – natūraliosios kalbos apdorojimas

I.2.8 Problem Solving, Control Methods, and Search – uždavinių sprendimas, valdymo metodai ir paieška

I.2.9 Robotics – robotika

I.2.10 Vision and Scene Understanding – matymo ir aplinkos suvokimas

I.2.11 Distributed Artificial Intelligence – išskirstytas dirbtinis intelektas

Vienas iš dirbtinio intelekto tikslų apibūdinimą yra toks (žr. Judea Pearl, Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference, 14 (1988), cituota pagal [MacCrimmon, Tillers 2002, 55]):

„Dirbtinio intelekto tikslas yra pateikti intelektualaus elgesio kompiuterinius modelius, svarbiausia, sveiko proto samprotavimo“ („The aim of artificial intelligence is to provide a computational model of intelligent behavior, most importantly, commonsense reasoning.“)

1.3. Žodžio „intelektualus“ samprata dirbtiniame intelekte

Mes laikomės požiūrio, kad intelektualus yra tik žmogus. Programų sistema, kompiuteris, mašina arba bet koks kitas žmogaus sukurtas *artefaktas* (angl. *artifact*) yra ne intelektualus, bet „intelektualus“ (kabutėse), kitaip žodžiais, intelektualizuotas. Pavyzdžiui:

- intelektualus žmogus;
- „intelektuali“ programų sistema;
- „intelektualus“ kompiuteris;
- „intelektuali“ mašina.

Tačiau DI literatūroje kalbant apie „intelektualias“ sistemas arba mašinas yra priimta kabutes praleisti.

Trumpai kalbant, DI nagrinėja, kaip kompiuterius padaryti labiau „intelektualiai“. Dirbtinio intelekto disciplina nagrinėja klausimus: kas yra laikomas „intelektualių“? ką reiškia būti dirbtiniai intelektuali (angl. *artificially intelligent*)?

DI siekia sukurti (angl. *build*) ir suprasti intelektualias esybes (angl. *intelligent entities*). Ką reiškia „intelektualus“? „Intelektualiai“ yra laikomi šie gebėjimai:

- 1) jausti, priimti (angl. *perceive*);
- 2) suprasti (angl. *understand*);
- 3) prognozuoti (angl. *predict*);
- 4) manipoliuoti – atlikti kažkokį veiksmą (angl. *manipulate*).

Intelekto požymis yra gebėjimas spręsti blogai suformuluotus uždavinius. DI pastangos yra sukurti mašinas, kurios gali matyti, judėti ir veikti [MacCrimmon, Tillers 2002, 55]. Tokį teiginį kaip DI apibrėžimą anksčiau vartojo kiti autoriai.

DI tyrimo objektas yra kompiuteris, tiksliau, kaip „masto“ kompiuteris, kitais žodžiais, kaip padaryti, kad kompiuteris „mąstyti“.

A. Tiuringas suformulavo klausimą „ar gali mašina mąstyti?“ (angl. *can machine think?*) [Turing 1950]. Jis suformulavo testą, kuris vėliau buvo pavadintas Tiuringo testu. Testas trumpai skamba taip. Žmogus A per terminalą su klaviatūra bendrauja su dyiem agentais B ir C, iš kurių vienas yra žmogus o kitas mašina, bet A nežino kuris yra kuris. Žmogaus A tikslas yra nustatyti, kuris agentas yra žmogus ir kuris mašina. Jeigu A nepavyksta atskirti, tai laikoma, kad mašina „masto“ (praeina Tiuringo testą).

Pavyzdžiui, Alice – tai programa, kuri Tiuringo testo atveju atstotų mašiną. Ją galima pasiekti „Artificial Intelligence Foundation“ tinklapyje (žr. <http://www.alicebot.org/>).

Viena iš DI šakų yra *kognityvistika* (angl. *cognitive science*). Kognityvistikos tyrimo objektas yra žmogus, tiksliau, kaip masto žmogus. I kognityvistiką mūsų medžiagoje nesigilinama, nes gilinamasi į kitą DI paradigmą – kaip „masto“ kompiuteris.

Tik žmogui yra būdingas žmogiškasis intelektas. Kompiuteris gi pasižymi dirbtiniu intelektu. Mechaninių skaičiavimo mašinelių sukūrimas XIX a. ir aritmometrai (pvz. finansų apskaitoje XX a. plačiai naudotas „Feliks“) pademonstravo, kad sugebėjimas skaičiuoti nėra išskirtinis žmogiškojo intelekto požymis. Žmogui pavyko automatizuoti aritmetinius veiksmus. Kalkuliatorius šiuo metu nėra laikomas DI artefaktu.

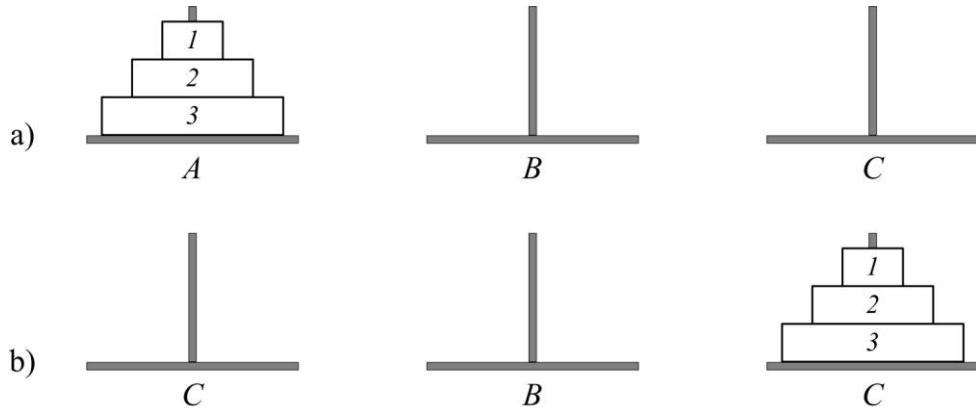
Prieš 40-50 metų DI varomoji jėga buvo natūralios kalbos apdorojimas (angl. *natural language processing*). Natūralios kalbos mašininio apdorojimo ir supratimo kriterijus yra loginis išvedimas (angl. *entailment*).

Veiksmų sekos, kitaip dar vadinamo *plano*, radimas yra priskiriamas DI. Pavyzdys gali būti gerai žinomas Hanojaus bokštų uždavinys.

1.4. Hanojaus bokštai

Turime tris virbus *A*, *B* ir *C*. Pradinėje būsenoje ant *A* yra n diskų – nuo didžiausio apačioje ir mažiausio viršuje, o ant *B* ir *C* tuščia; 1.1 pav. parodytas atvejis $n=3$. Uždavinio tikslas – perkelti diskus ant virbo *C* (žr. 1.2 pav.). Keliamas tik viršutinis diskas. Draudžiama didesnį diską dėti ant mažesnio.

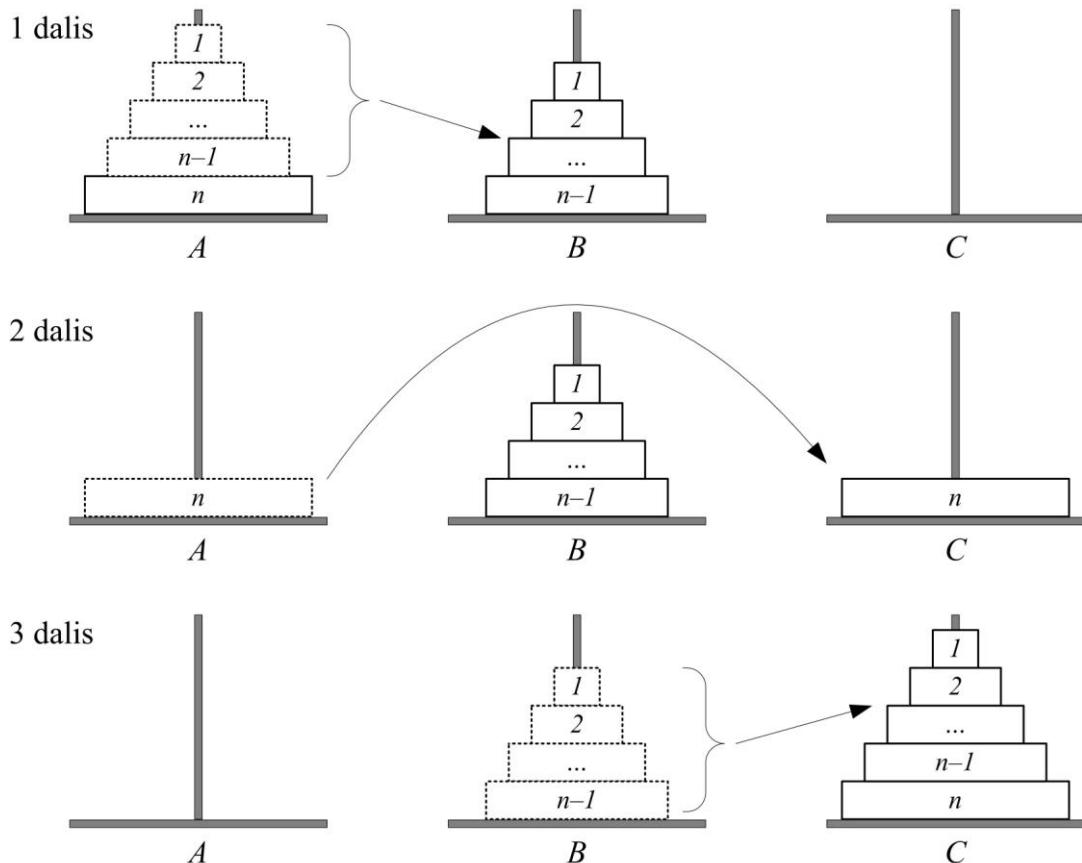
Uždavinio apibendrinimą žr. http://en.wikipedia.org/wiki/Towers_of_Hanoi.



1.1 pav. a) Hanojaus bokštų uždavinio pradinė būsena $\{(3,2,1), (), ()\}$. b) Terminalinė būsena $\{(), (), (3,2,1)\}$

Diskų perkėlimas yra intelektualus uždavinys. Intelektas glūdi algoritme, kuris rekursinis ir susideda iš dalių:

- 1 dalis. Perkelti iš viso $n-1$ diską nuo A ant B;
- 2 dalis. Perkelti diską n nuo A ant C;
- 3 dalis. Perkelti iš viso $n-1$ diską nuo B ant C.



1.2 pav. Trys rekursinio algoritmo dalys

Veiksmų seka kai $n=3$:

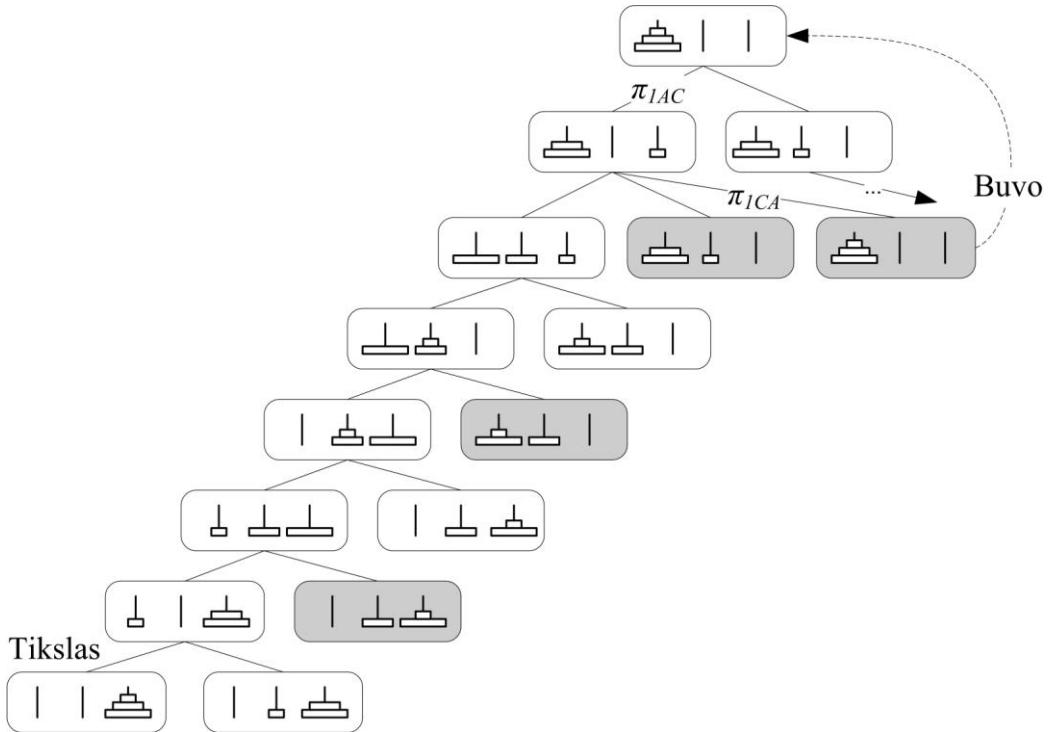
- $\{(3,2,1), (), ()\}$ – pradinė būsena;
1. $\{(3,2), (), (1)\}$ – diskas 1 nuo virbo A perkeliamas ant C;
2. $\{(3), (2), (1)\}$ – diskas 2 nuo virbo A perkeliamas ant B;
3. $\{(3), (2,1), ()\}$ – diskas 1 nuo virbo C perkeliamas ant B;
4. $\{(), (2,1), (3)\}$ – diskas 3 nuo virbo A perkeliamas ant C;
5. $\{(1), (2), (3)\}$ – diskas 1 nuo virbo B perkeliamas ant A;
6. $\{(1), (), (3,2)\}$ – diskas 2 nuo virbo B perkeliamas ant C;
7. $\{(), (), (3,2,1)\}$ – diskas 1 nuo virbo A perkeliamas ant C.

Šios sekos radimas yra žmogiškajam intelektui įveikiamas uždavinys. Tačiau ši uždavinį gali spręsti ir kompiuteris. Rekursinė procedūra:

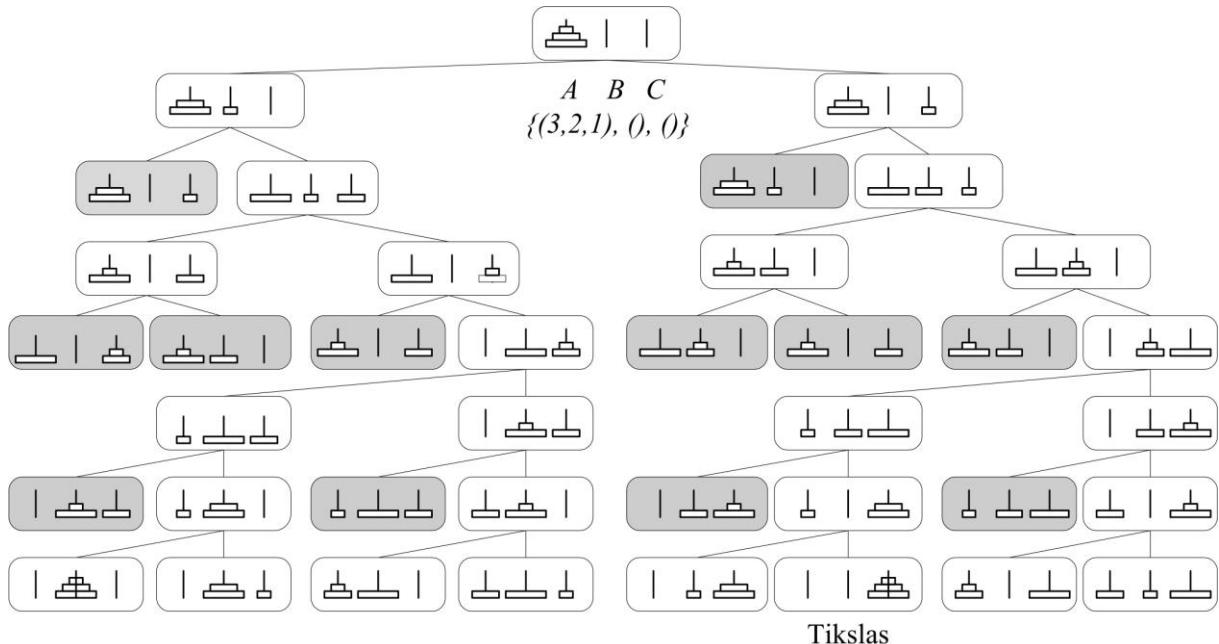
```
procedure hb(x, y, z: char; n: integer);
{x, y, z yra virbų vardai; n – diskų skaičius.}
{x – nuo, y – tarpinis, z – ant.}
begin
  if n > 0 then
    begin
      hb(x, z, y, n-1); {1. Perkelti n-1 diską ant tarpinio.}
      writeln(' Nuo virbo ', x,' perkeliama ant ', z); {2.}
      hb(y, x, z, n-1); {3. Perkelti n-1 diską ant tikslą. }
    end
  end
end
```

Bendru atveju perkėlimų skaičius yra eksponentinis, tiksliau, 2^{n-1} .

Tokią elegantišką rekursinę procedūrą pasiseka parašyti tik nedidelei daliai dirbtinio intelekto uždavinių. Tolimesniuose skyriuose aptariami paieškos algoritmai būsenų perėjimo grafuose (angl. *state transition graphs*). 1.3 pav. pateiktas GRAPHSEARCH_L_GYL algoritmo paieškos medis Hanojaus bokštų uždaviniui, kai $n=3$.



1.3 pav. Algoritmo GRAPHSEARCH_L_GYLĮ paieškos medis Hanojaus bokštų uždavinui $n=3$. Pilkai pažymėtos būsenų viršūnės, kurios jau yra sąraše ATIDARYTA arba UŽDARYTA. Algoritmas GRAPHSEARCH nagrinėjamas tolimesniuose skyriuose



1.4 pav. GRAPHSEARCH_L_PLOTI algoritmo paieškos medis Hanojaus bokštų uždavinyje. Pilkai pažymėtos būsenų viršūnės, kurios jau yra sąraše ATIDARYTA arba UŽDARYTA

Iteracinis Hanojaus bokštų algoritmas nėra taip žinomas kaip nagrinėtas rekursinis; žr. p.vz. [Brachman & Levesque 2004, p. 133–134]. Virbai sustatomi ratu: *A*, *B* ir *C*. Disko perkėlimo taisyklės:

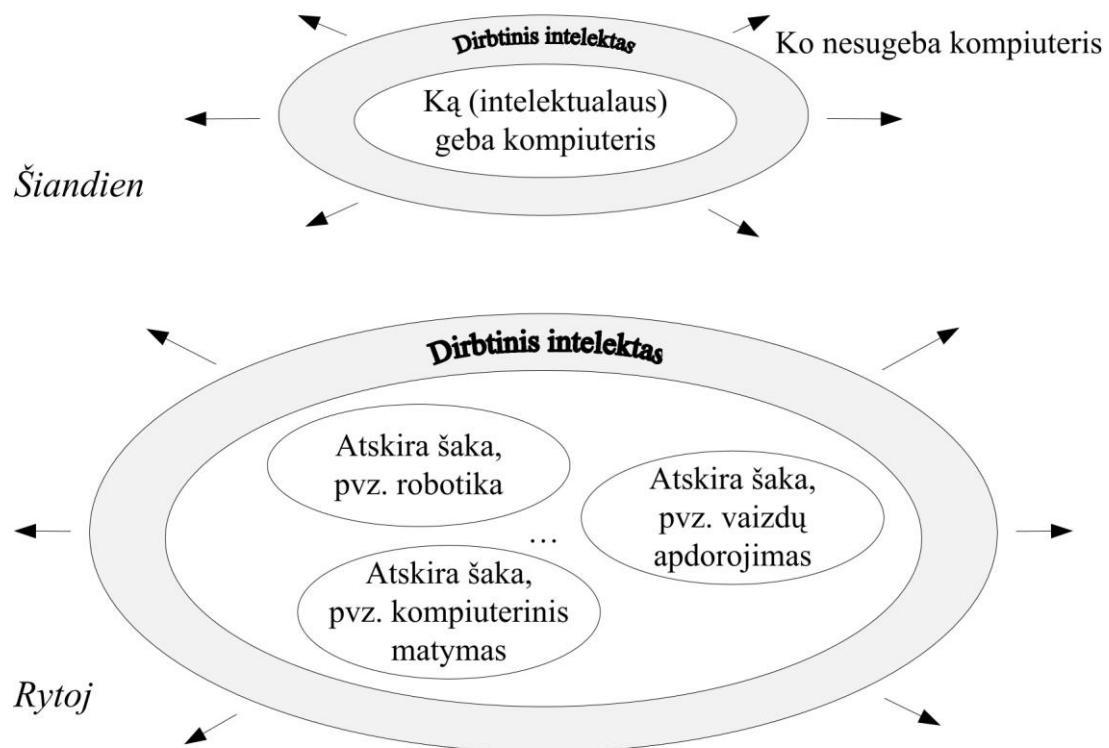
1. Perkelti ant artimiausio galimo virbo pagal laikrodžio rodyklę, jeigu *n* lyginis, arba prieš laikrodžio rodyklę, jeigu *n* nelyginis.

2. Tą patį diską kelti du kartus iš eilės draudžiama.

Kaip pratimą rekomenduojame parašyti programą.

1.5. Dirbtinio intelekto sampratos kitimas

Iškirtinos dvi DI savybės. Pirma, DI nagrinėja informatikos ribą: ką (intelektualaus) gali kompiuteris ir ko negali. Antra, dirbtinio intelekto samprata laikui bėgant kinta. Kas šiandien priskiriama DI, rytoj gali būti nebepriskiriama DI (žr. 1.5 pav.). Šitaip formuoja atskirose šakose, pavyzdžiu, robotika, vaizdų apdorojimas (angl. *pattern recognition*), kompiuterinis matymas (angl. *machine vision*).



1.5 pav. Kas šiandien priskiriama DI, rytoj gali būti nebepriskiriama DI.
Šitaip formuoja atskirose šakose

1.6. Uždavinių sprendimo metodų principai

Uždavinių, kurių sprendimui taikomi kompiuteriai, išankstinis klasifikavimas turi prasmę pats savaime. Žmogui susidūrus su nauju uždaviniu, uždavinio priskyrimas tam tikrai klasei gali iš anksto pasufleruoti sprendimo būdą. Uždavinio pavyzdys yra naftos buvimo prognozės sudarymas (yra naftos ar nėra), atsižvelgiant į seismogramą (trimatį žemės gelmių vaizdą, gautą iš tam tikrų prietaisų).

Pateiksime uždavinių klasifikaciją, kuris priimta kitose ne dirbtino intelekto dalykinėse srityse:

1. interpretavimas (analizė) (angl. *interpret (analysis)*)
 - 1.1. indentifikavimas (atpažinimas) (angl. *identify (recognise)*)
 - 1.1.1. stebėjimas (auditas, tikrinimas) (angl. *monitor (audit, check)*)
 - 1.1.2. diagozė (derinimas) (angl. *diagnose (debug)*)
 - 1.2. numatymas (simuliavimas) (angl. *predict (simulate)*)
 - 1.3. valdymas (angl. *control*)
2. konstravimas (sintezė) (angl. *construct (synthesis)*)
 - 2.1. specifikavimas (apribojimas) (angl. *specify (constrain)*)
 - 2.2. projektavimas (angl. *design*)
 - 2.2.1. planavimas (apdorojimas) (angl. *plan (process)*)
 - 2.2.2. konfigūravimas (struktūrizavimas) (angl. *configure (structure)*)
 - 2.2. surinkimas (gamyba) (angl. *assemble (manufacture)*)
 - 2.3.1. modifikavimas (taisymas) (angl. *modify (repair)*)

Analizė ir sintezė yra bendri visoms dalykinėms sritims metodai. Analizė yra skaidymas į dalis, o sintezė – konstravimas iš dalii.

Dirbtiniame intelekte yra priimtas tokis uždavinių sprendimo metodų (angl. *problem solving methods*) klasifikavimas:

1. klasifikavimas (angl. *classification*)
 - 1.1. neabejotinas klasifikavimas (angl. *certain classification*)
 - 1.1.1. sprendimų medžiai (angl. *decision trees*)
 - 1.1.2. sprendimų lentelės (angl. *decision tables*)
 - 1.2. euristinis klasifikavimas (angl. *heuristic classification*)
 - 1.3. modeliu grindžiamas klasifikavimas (angl. *model-based classification*)
 - 1.3.1. poaibius padengiantis klasifikavimas (angl. *set covering classification*)
 - 1.3.2. funkcinis klasifikavimas (angl. *functional classification*)
 - 1.4. statistinis klasifikavimas (angl. *statistical classification*)
 - 1.5. precedentais grindžiamas klasifikavimas (angl. *case-based classification*)
2. konstravimas (angl. *construction*)
 - 2.1. euristinis konstravimas (angl. *heuristic construction*)
 - 2.1.1. skeleto konstravimas (angl. *skeletal construction*)
 - 2.1.2. bandymų ir klaidų metodas (angl. *propose-and-revise strategy*)
 - 2.1.3. bandymų ir pakeitimų metodas (angl. *propose-and-exchange strategy*)
 - 2.1.4. mažiausią įsipareigojimų metodas (angl. *least-commitment strategy*)
 - 2.2. modeliais grindžiamas konstravimas (angl. *model-based construction*)
 - 2.3. precedentais grindžiamas konstravimas (angl. *case-based construction*)
3. imitavimas (angl. *simulation*)
 - 3.1. vieno žingsnio imitavimas (angl. *one-step simulation*)
 - 3.2. daugiažingsnis imitavimas (angl. *multiple-step simulation*)

3.2.1. skaitinis daugiažingsnis imitavimas (angl. *numerical multiple-step simulation*)

3.2.2. kokybinis daugiažingsnis imitavimas (angl. *qualitative multiple-step simulation*)

Šie uždavinių sprendimo metodai yra laikomi uždavinių sprendimo principais. Tuo pačiu šie principai įeina į dirbtinio intelekto principus.

Kuo skiriasi metodas nuo principio? Metodas taikomas aiškioje situacijoje. Principas taikomas ir nebūtinai aiškioje (t. y. ir neaiškioje) situacijoje. Jeigu uždaviniui išspręsti nėra žinomas metodas, tai tenka vadovautis principais.

Apibūdindami principio skirtumą nuo metodo, mes vadovaujamės analogija su skirtumu tarp principio ir normos. Norma yra apibrėžiama kaip visuotinai priimta elgesio taisyklė. Norma yra taikoma aiškioje situacijoje, tiksliau, esant tam tikrai normoje įtvirtintai aplinkybių sudėčiai. Normų pavyzdžiai:

1. Jeigu einama pro duris, tai vyros praleidžia moterį, išskyrus, kai vyros yra geriamo amžiaus arba moteris labai jauna.
2. Jeigu einama pavojingu maršruto (pavyzdžiui neapšviestais laiptais), tai vyros eina pirmas.

Principas yra „vyros suteikia pirmenybę moteriai“.

Termino „metodas“ prasmę galima iliustruoti formulėmis, kurias Albertas Čaplinskas pateikė dėstydamas programų sistemų inžineriją:

$$\text{Metodas} = \text{schema} + \text{procedūros}$$

$$\text{Metodika} = \text{metodas} + \text{rekomendacijos}$$

Šios formulės nors ir supaprastina termino prasmę, bet atspindi esmę.

2. Dirbtinio intelekto sistema kaip produkcių sistema

Toliau vadovaujamės [Nilsson 1982] 1.1 poskyriu.

Daugumoje dirbtinio intelekto sistemų galima išskirti tris esminius komponentus: duomenis, operacijas ir jų valdymą. Kitaip tariant, dažnai tokiose sistemose galima išskirti tam tikrą centrinę duomenų struktūrą, kurią vadinsime *globalia duomenų bazę*, bei veiksmus, atliekamus su ja, pagal iš anksto griežtai apibrėžtas operacijas. Šie veiksmai paprastai valdomi remiantis tam tikra globalia valdymo strategija. Bendru atveju, tokios sistemos yra vadinamos produkcių sistemomis. Taigi, produkcių sistema vadinsime trejetą:

1. globali duomenų bazė (sutrumpintai – duomenų bazė arba GDB);
2. produkcių aibė $\{\pi_1, \pi_2, \dots, \pi_m\}$;
3. valdymo sistema.

Šis trejetas yra modelis. Taigi jis yra kartu ir stiprybė, ir silpnybė – priklausomai nuo požiūrio taško.

Remiantis tokiu modeliu dalykinės srities žinios gali būti klasifikuojamos pagal tai, kur yra vaizduojamos. *Deklaratyviosiomis žiniomis* yra vadinamos žinios, vaizduojamos globalioje duomenų bazėje. *Procedūrinėmis žiniomis* vadinamos žinios, vaizduojamos produkcių aibėje. *Valdymo žiniomis* vadinamos žinios, vaizduojamos valdymo strategijoje.

Globali duomenų bazė – tai duomenų struktūra, kurią naudoja dirbtinio intelekto produkcių sistema. Priklasomai nuo konkrečios probleminės srities uždavinio, ši duomenų bazė gali būti tiek įprasta sveikųjų skaičių matrica, tiek sudėtinga duomenų bazių valdymo sistema. Produkciai iš produkcių aibės yra taikoma globaliai duomenų bazei. Pritaikius produkcių globali duomenų bazę yra pervedama iš vienos būsenos į kitą. Produkciai gali būti taikomi, jeigu globalios duomenų bazės būsena tenkina tos produkcijos pradinę sąlygą. Kurią būtent produkcių iš produkcių aibės parinkti ir taikyti, sprendžia valdymo sistema. Po kiekvienos produkcijos taikymo, valdymo sistema patikrina, ar globalios duomenų bazės būsena yra terminalinė. Jeigu globali duomenų bazė tenkina terminalinės būsenos sąlygą, tai algoritmas baigia darbą. Jeigu globalios duomenų bazės būsena nėra terminalinė, o valdymo sistema nebegali taikyti nei vienos produkcijos, tai laikoma, kad uždavinio sprendinys neegzistuoja.

Produkcių sistemos algoritmas yra užrašomas taip:

```
procedure PRODUCTION
{1} DATA := pradinė duomenų bazė;
{2} until DATA patenkina terminalinę sąlyga do
{3} begin
{4}   select išrinkti produkcią  $\pi_i$  iš aibės
        tų produkcių, kurias galima taikyti duomenų
        bazei DATA
{5}   DATA :=  $\pi_i$ (DATA) {Rezultatas, gautas pritaikius  $\pi_i$ 
        duomenų bazei DATA. Produkciai atvaizduoja
        (t. y. perveda) duomenų bazę iš vienos
        būsenos į kitą}
{6} end
```

Procedūros rezultatas (mes jį vadinsime uždavinio sprendiniu) yra produkcių seką $\langle \pi_{i1}, \pi_{i2}, \dots, \pi_{in} \rangle$, tokia kai pradinė būsena pervedama į terminalinę būseną (jų gali būti daug). Terminalinių būsenų aibė gali būti apibrėžiama predikatu, kurio reikšmė yra true jeigu būsena tenkina terminalinę sąlygą, ir false priešingu atveju.

Tai *nedeterminuota* procedūra. Operatorius **select** išrenka produkciją *nedeterminuotu* būdu. Išrinkimo algoritmas nėra nusakomas – tinka bet koks pasirinkimas. Visos produkcijos turi vienodą prioritetą ir nekviečia viena kitos. Išrenkama nebūtinai pirmoji produkcija iš aibės visų galimų taikyti produkcijų. Operatoriuje **select** ir glūdi intelektas – ką pasirinkti? Tokiu būdu procedūra PRODUCTION apibrėžia visą *determinuotu* algoritmulį šeimą.

PRODUCTION vykdo paiešką „i gylį“.

Kaip matyti iš procedūros PRODUCTION produkcijų sistemos elementais yra dar dvi sąvokos: pradinė GDB būsena ir terminalinė GDB būsena. Pradinė būsena yra viena. Terminalinų būsenų gali būti keletas; tai priklauso nuo uždavinio.

Ši procedūra PRODUCTION yra suprantama kaip valdymo sistemos *paradigma* (angl. *paradigm*). Žodžio „paradigma“ reikšmės aiškinimas remiantis anglų kalbos žodynu Oxford English Dictionary (žr. <http://dictionary.oed.com/>):

paradigma (angl. *paradigm*)

1. Šablonas arba modelis, egzempliorius; (taip pat) kažko tipinis atskiras atvejis, pavyzdys (Pattern or model, an exemplar; (also) a typical instance of something, an example).
2. a. *gramatika*. Tradicinėse lotynų, graikų ir kitų linksniuojuojamujų kalbų gramatikose: šablonas arba lentelė, rodanti visas konkretaus daiktavardžio, veiksmažodžio ar būdvardžio linksnių formas, naudojama kaip modelis kitų žodžių asmenavimui ar linksniavimui (In the traditional grammar of Latin, Greek, and other inflected languages: a pattern or table showing all the inflected forms of a particular verb, noun, or adjective, serving as a model for other words of the same conjugation or declension).
4. Koncepcinis ar metodinis modelis, pagrindžiantis mokslo arba disciplinos teorijas ar panaudojimą tam tikru laiku; (taigi) bendrai priimta pasauležiūra (A conceptual or methodological model underlying the theories and practices of a science or discipline at a particular time; (hence) a generally accepted world view).

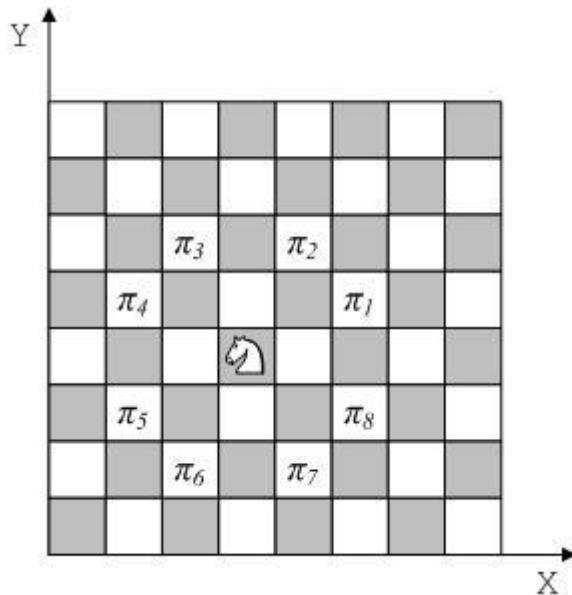
Kaip produkcijų sistemos pavyzdį išnagrinėsime „žirgo“ uždavinį: apeiti visą šachmatų lentą žirgu, kiekviename langelyje pabuvojant tik po vieną kartą. Pradinė žirgo padėtis – langelis [1,1].

8	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0
	1	2	3	4	5	6	7	8

2.1 pav. Šachmatų lenta vaizduojama dvimačiu masyvu. Pradinė žirgo padėtis yra langelyje [1,1]

Šiame uždavinyje globali duomenų bazė yra vaizduojama dvimačiu 8×8 sveikujų skaičių masyvu. Pradinėje būsenoje visi jo laukai turi reikšmę 0, išskyrus langelį [1,1], kurio reikšmė yra 1; žr. 2.1 pav.

Produkcijų aibę sudaro 8 produkcijos $\{\pi_1, \pi_2, \pi_3, \pi_4, \pi_5, \pi_6, \pi_7, \pi_8\}$, t. y. visi galimi žirgo éjimai šachmatų lentoje:



2.2 pav. Aštuoni galimi žirgo éjimai šachmatų lentoje – tai aštuonios produkcijos $\{\pi_1, \pi_2, \pi_3, \pi_4, \pi_5, \pi_6, \pi_7, \pi_8\}$

Pritaikius produkciją, globali duomenų bazė pervedama iš vienos būsenos į kitą. Nauja žirgo pozicija vaizduojama éjimo numeriu langelyje, į kurį buvo žengta. Šiame uždavinyje terminalinė globalios duomenų bazés būsena yra išreiškiama tokiu teiginiu: visi masyvo langeliai yra užpildyti skaičiais nuo 1 iki 64, kurie išdėstyti žirgo éjimais. Tuo atveju, jeigu globaliai duomenų bazei negalima taikyti nei vienos produkcijos ir masyve yra laukų, kurių reikšmė lygi 0, tai žirgu apeiti lentos nepavyko.

Toliau pateikiama programa Paskalio kalba. Programa parašyta atsižvelgiant į tekstą iš Valentinos Dagienės, Gintauto Grigo ir Kęstučio Augučio knygos [Dagienė, Grigas, Augutis 1986].

Uždavinio ir programos esmė nesikeičia, kai lentos dydis yra ne 8×8 , o $N \times N$ langelių. Paprasčiausias atvejis yra, kai $N=5$. Siūloma patiems nusibraižyti 4×4 langelių lentą ir išitikinti, kad žirgu nepavyks jos apeiti.

```

program ŽIRGAS;
const N      = 5;          {Šachmatų lento dydis}
    NN     = 25;         {Langelių skaičius šachmatų lentoje 5x5}
type INDEX = 1..N;
var
    LENTA : array[INDEX, INDEX] of integer; {Globali duomenų bazė}
    CX, CY : array[1..8] of integer;   {Produkcijų aibė visada iš 8 prod.}
    I, J   : integer;
    YRA    : boolean;

procedure INICIALIZUOTI;
var   I, J : integer
begin
    {1) Suformuojama produkcijų aibė}
    CX[1] := 2;    CY[1] := 1;
    CX[2] := 1;    CY[2] := 2;
    CX[3] := -1;   CY[3] := 2;
    CX[4] := -2;   CY[4] := 1;
    CX[5] := -2;   CY[5] := -1;
    CX[6] := -1;   CY[6] := -2;
    CX[7] := 1;    CY[7] := -2;
    CX[8] := 2;    CY[8] := -1;
    {2) Inicializuojama globali duomenų bazė}
    for I := 1 to N do
        for J := 1 to N do
            LENTA[I, J] := 0;
end; {INICIALIZUOTI}

procedure EITI (L : integer; X, Y : INDEX; var YRA : boolean);
{Jéjimo parametrai L - éjimo numeris; X, Y - paskutinė žirgo padėtis}
{Išėjimo parametras (t.y. rezultatas) YRA}
var
    K      : integer; {Produkcijos eilės numeris}
    U, V   : integer; {Nauja žirgo padėtis}
begin
    K := 0;
    repeat {Perrenkamos produkcijos iš 8 produkcių aibės}
        K := K + 1;
        U := X + CX[K]; V := Y + CY[K];
        {Patikrinama, ar duomenų bazė tenkina produkcijos taikymo sąlyga}
        if (U >= 1) and (U <= NN) and (V >= 1) and (V <= NN)
        then {Neišeinama už lento krašto.}
            {Patikrinama, ar langelis laisvas, t. y. ar žirgas jame nebuvvo}
            if LENTA[U, V] = 0
            then
                begin
                    {Nauja žirgo pozicija pažymima éjimo numeriu}
                    LENTA[U, V] := L;
                    {Patikrinama, ar dar ne visa lenta apeita}
                    if L < NN
                    then
                        begin {Jeigu lenta neapeita, tai bandoma daryti éjima}
                            EITI(L+1, U, V, YRA);
                            {Jei buvo pasirinktas nevedantis į sékmę éjimas,}
                            {tai grįztama atgal, t. y. langelis atlaisvinamas}
                            if not YRA then LENTA[U, V] := 0;
                        end
                    else YRA := true; {Kai L=NN}
                end;
            until YRA or (K = 8); {Sékmė arba perrinktos visos 8 produkcijos}
end; {EITI}

```

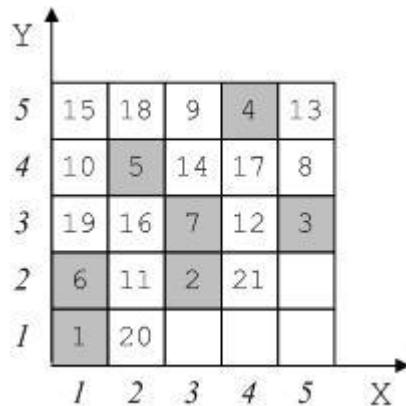
```
begin {Pagrindinė programa (main program) }
{1. Paruošiama globali duomenų bazė ir užpildoma produkcių aibė}
INITIALIZUOTI; YRA := false;
{2. Pažymima pradinė žirgo padėtis: langelis [1,1]}
LENTA [1, 1] := 1;
{3. Kviečiama procedūra sprendinio paieškai. Daryti antraėjimą, }
{   stovint langelyje X=1 ir Y=1, ir gauti atsakymą YRA           }
EITI(2, 1, 1, YRA);
{4. Jeigu sprendinys rastas, tai spausdinama lenta}
if YRA then
    for I := N downto 1 do
        begin
            for J := 1 to N do
                write(LENTA[I,J]);
                writeln;
            end;
        else writeln('Sprendinys neegzistuoja');
    end.
```

Sprendinys 5x5 šachmatų lentoje yra tokis:

	X
5	21 16 11 4 23
4	10 5 22 17 12
3	15 20 7 24 3
2	6 9 2 13 18
1	1 14 19 8 25
	X

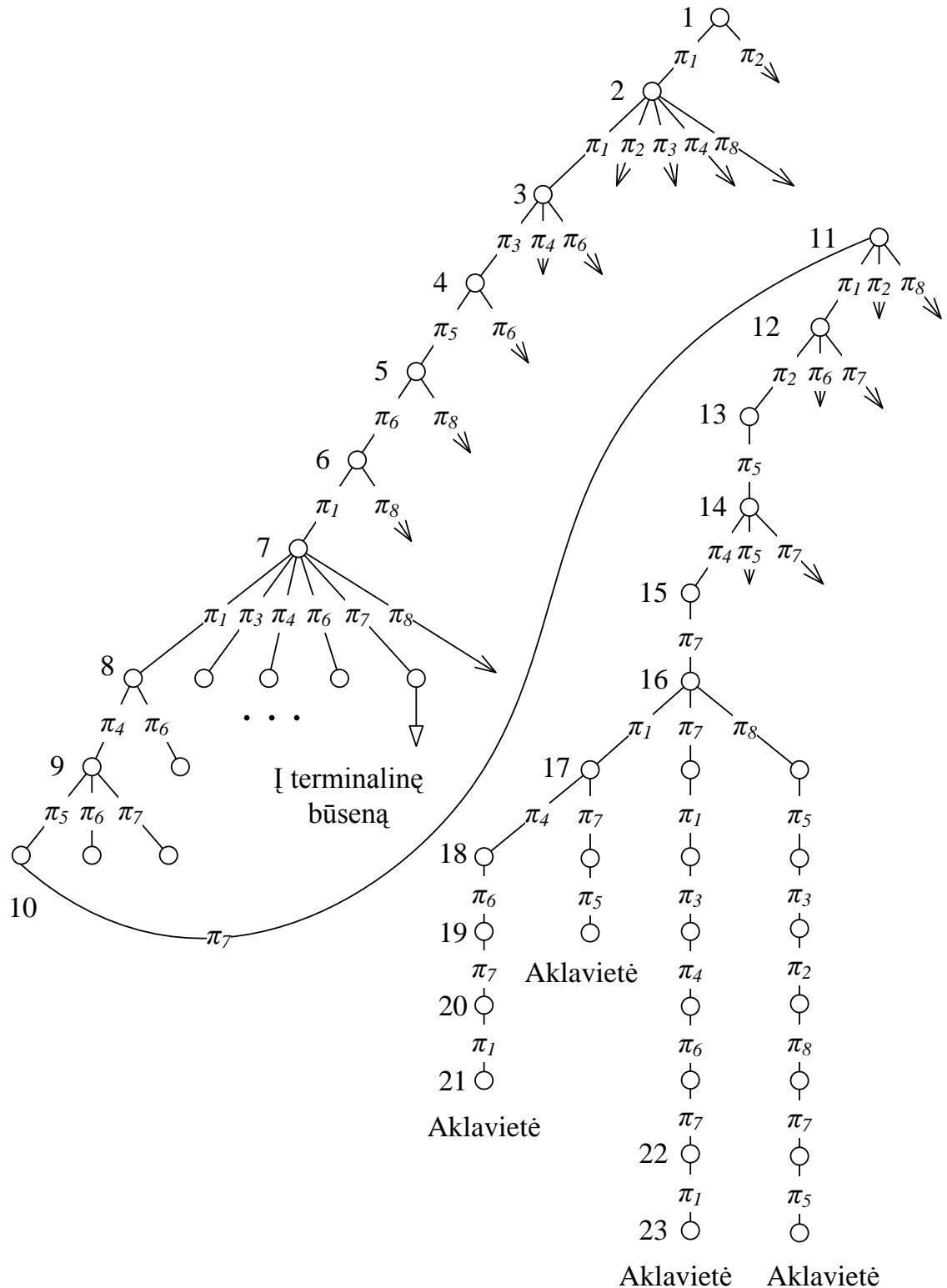
2.3 pav. Žirgas apeina 5x5 šachmatų lentą

Žirgas pirmają aklavietę 5x5 lentoje pasiekia po 21 įjimo. Ši globalios duomenų bazės būsena (t. y. šachmatų lenta) yra tokia:



2.4 pav. Žirgas apeina 5x5 šachmatų lentą. Išskirti pirmieji 7 éjimai, sutampantys su sėkmingu apéjimu, pateiktu 2.3 pav. Aštuntasis éjimas neveda į tikslą

Šioje būsenoje žirgas toliau žengti negali, t. y. šioje būsenoje negalima pritaikyti jokios iš 8 produkcijų. Žirgui tenka grįžti atgal ir bandyti eiti kitur. Pastebime, kad šioje būsenoje tik pirmieji septyni éjimai sutampa su sėkmingu apéjimu, pavaizduotu 2.3 pav. Tokiu būdu, 2.4 pav. pateiktoje būsenoje žirgui jau aštuntuoju éjimu parinkta į tikslą nevedanti produkcija.



2.5 pav. Paieškos medis iki pirmosios aklavietės. Jis vaizduoja žirgo ėjimus 5x5 šachmatų lentoje iki pasiekiamos pirmoji aklavietė, kai žirgas negali eiti toliau ir turi gržti. Lentos būsena yra pateikta 2.4 pav.

Pastebime, kad uždavinyje apie žirgą, terminalinių GDB būsenų yra daugiau negu viena (tuo atveju kai sprendinys apskritai egzistuoja). Akivaizdu kad egzistuoja mažiausiai du žirgo apėjimai, kurie yra simetriški pagrindinės kylančios ištisinės atžvilgiu.

Kas ir kaip yra užprogramuota?

1. LENTA, X ir Y – tai globali duomenų bazė;
2. CX ir CY – tai produkcių aibė;
3. Paieškos į gylį su grįžimas algoritmas EITI – tai valdymo sistema.

Šie trys punktai ir yra trejetas, sudarantis produkcių sistemą žirgo uždavinyje.

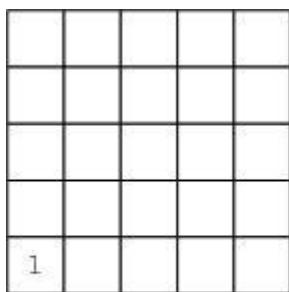
Planas, t. y. keliais arba atsakymas, yra $\langle \pi_{i1}, \pi_{i2}, \pi_{i3}, \dots, \pi_{i24} \rangle$, kur $\pi_{ij} \in \{\pi_1, \pi_2, \pi_3, \pi_4, \pi_5, \pi_6, \pi_7, \pi_8\}$, t. y. $i_j \in \{1, 2, \dots, 8\}$.

Žiniomis grindžiamos sistemos (angl. *knowledge-based system*) architektūroje išskiriame du elementus:

1. Samprotavimas (angl. *reasoning*). Tai valdymas (protavimas, mąstymas). Išvedimo mašiną galima įsigyti. Ji nepriklauso nuo dalykinės srities;
2. Žinių bazė (angl. *knowledge base*). Ją sudaro GDB (globali duomenų bazė) ir produkcių aibė. Žinių bazės negalima įsigyti, nes ji priklauso nuo dalykinės srities. „Dievas yra detalėse, o ne teoremore“ – teigia žymus DI autoritetas.

Pratimai

1. Nubraižykite paieškos medį, kuris perveda globalią duomenų bazę (šachmatų lentą su žirgu) iš pradinės būsenos 2.6 a pav. į terminalinę 2.6 b pav.. Pavyzdys jau pateiktas 2.5 pav.



a)

21	16	11	4	23
10	5	22	17	12
15	20	7	24	3
6	9	2	13	18
1	14	19	8	25

b)

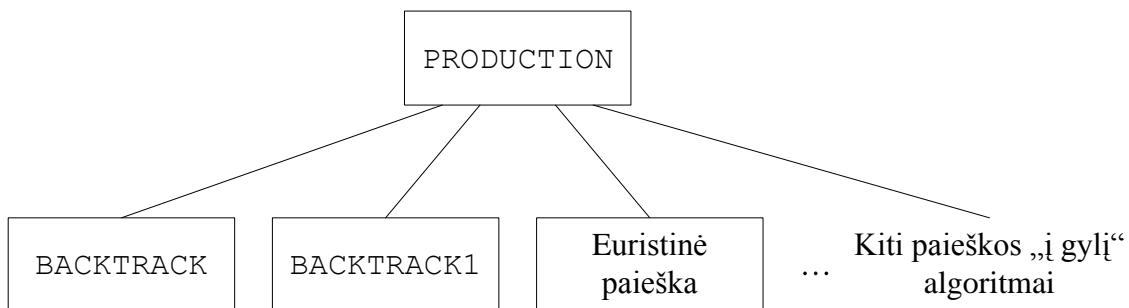
2.6 pav. Pradinė ir terminalinė žirgo būsenos 5×5 lentoje

2. Parašykite programą, kuri nagrinėtam žirgo uždavinui spausdina **visus** galimus sprendinius.

3. Valdymas su grīžimais ir procedūra BACKTRACK

Terminas „valdymas su grīžimais“ žymi vieną iš valdymo sistemos algoritmų, tiksliau algoritmų šeimą. Ši šeima, aprašoma žemiau pateikiama procedūra BACKTRACK, ir jos modifikacijomis. Nilsonas ir kiti autorai dar vartoja žodį strategija – valdymo strategija. Pastarasis terminas yra mažiau formalus negu algoritmas. Valdymas su grīžimais yra svarbus DI principas.

Procedūra BACKTRACK yra atskiras PRODUCTION atvejis. Kitų algoritmų pavyzdžiai yra BACKTRACK1 bei euristinė paieška (žr. 3.1 pav.).



3.1 pav. Procedūra BACKTRACK yra atskiras PRODUCTION atvejis

Valdymą su grīžimais paaikinsime valdovių išdėstymo šachmatų lentoje uždavinio pavyzdžiu. Aštuonias valdoves reikia išdėstyti šachmatų lentoje taip, kad jos nekirstų viena kitos. Ši uždavinį galima pavaizduoti produkcijų sistema:

1. Globali duomenų bazė (GDB) – tai šachmatų lenta;
2. Produkcijų aibė, kurioje 64 produkcijos, yra $\{\pi_{i,j}, i, j = 1, 2, \dots, 8\}$. Čia produkcija $\pi_{i,j}$ reiškia valdovę pastatymą į langelį $[i,j]$. Pastebime, kad tokią produkcijų aibę galima sumažinti nuo 64 iki 8 produkcijų $\{\pi_k, k = 1, 2, \dots, 8\}$, kur π_k – statyti einamają valdovę į k -tajį stulpelį. Pastaruoju atveju vadovaujamas prielaida, kad pirmoji valdovė statoma į pirmosios eilutės stulpelį $k1$, antroji – į antrosios eilutės stulpelį $k2$ ir t. t.;
3. Valdymo sistema – procedūra BACKTRACK.

Pradinė GDB būsena yra tuščia lenta. Terminalinė GDB būsena charakterizuojama tokiu predikatu: 8 valdovės sustatytos taip, kad nekerta viena kitos. Valdovių sustatymo planas yra $\langle \pi_{k1}, \pi_{k2}, \dots, \pi_{k8} \rangle$, kur $k1, k2, \dots, k8 \in \{1, 2, \dots, 8\}$.

Nedeterminuota procedūra PRODUCTION (žr. 2 skyrių) vaizduoja ištisą valdymo algoritmų šeimą. Terminą „valdymo strategija“ suprasime kaip plano, suprantamo kaip produkcijų seka, paiešką. Algoritmo esmė, kaip yra realizuojamas nedeterminuoto pasirinkimo operatorius **select**. Toliau yra nagrinėjamas *valdymo su grīžimais* algoritmas, kuris sutrumpintai vadinamas „valdymu su grīžimais“ (angl. *backtracking*).

Valdymo su grīžimais algoritmas dažnai būna pakankamai efektyvus uždaviniams, kuriuose atliekama nedidelės apimties paieška, kaip, pavyzdžiui, ankstesniame skyriuje išnagrinėtame žirgo uždavinyje. Šio algoritmo privalumas yra paprasta realizacija. Paiešką su grīžimais vaizduoja procedūra BACKTRACK [Nilsson 1985, 2.1 poskyris] žemiau.

Procedūra BACKTRACK kaip rezultatą grąžina produkcijų sąrašą.

```
recursive procedure BACKTRACK (DATA)
{DATA – einamoji globalios duomenų bazės būsena}
    { 1}      if TERM(DATA) then return NIL;
    { 2}      if DEADEND(DATA) then return FAIL;
    { 3}      RULES := APPRULES(DATA);
    { 4}      LOOP:   if NULL(RULES) then return FAIL;
    { 5}          R      := FIRST(RULES);
    { 6}          RULES := TAIL(RULES);
    { 7}          RDATA := R(DATA);
    { 8}          PATH   := BACKTRACK(RDATA);
    { 9}          if PATH = FAIL then goto LOOP;
{10}      return CONS(R, PATH);
end;
```

1 žingsnis. Patikrinama, ar nepasiekta globalios duomenų bazės (GDB) terminalinė būsena. Predikato TERM reikšmė yra true, jeigu einamoji GDB būsena DATA tenkina terminalinę sąlygą (t. y. tikslas pasiektas), ir false priešingu atveju. Jeigu TERM reikšmė yra true GDB pradinėje būsenoje, tai procedūra grąžina tuščią produkcijų sąrašą, žymimą NIL, ir sėkmingai baigia darbą.

2 žingsnis. Tikrinama, ar einamoji GDB būsena gali pasitaikyti kelyje i tikslą. Predikato DEADEND reikšmė yra true, jeigu tokia GDB būsena apskritai nepasitaiko kelyje i GDB terminalinę būseną. Tokiu atveju procedūra BACKTRACK baigia darbą ir grąžinamas nesėkmės požymis FAIL. Predikatu DEADEND yra programuojamos tokios žinios, kai programuotojas iš anksto žino, kad tam tikra GDB būsena niekada negali pasitaikyti kelyje i terminalinę būseną.

3 žingsnis. Funkcija APPRULES sudaro aibę tokį produkciją, kurios gali būti taikomos GDB einamajai būsenai DATA. Ši produkcijų aibė yra vadinama *konfliktine aibe* (angl. *conflict set*).

4 žingsnis. Jeigu konfliktinė aibė yra tuščia, t. y. nėra nė vienos produkcijos, kurią būtų galima taikyti, tai procedūra grąžina nesėkmės požymį FAIL ir baigia darbą.

5 žingsnis. Iš produkcijų aibės (sarašo) RULES yra paimama pirmoji produkcija. Funkcijos FIRST argumentas RULES yra sarašas, o reikšmė – šio sarašo pirmasis elementas. Funkcija FIRST yra žinoma iš algoritminės kalbos LISP, skirtos programavimui sarašinėmis duomenų struktūromis (angl. *LISP Processing*). Pavyzdžiui, FIRST($\langle \pi_1, \pi_2, \pi_3, \pi_4, \pi_8 \rangle$) = π_1 .

6 žingsnis. Iš produkcijų aibės RULES yra pašalinama pirmoji produkcija. Funkcijos TAIL argumentas RULES yra sarašas. O grąžinama reikšmė yra šis sarašas, bet be pirmojo elemento. Pavyzdžiui, TAIL($\langle \pi_1, \pi_2, \pi_3, \pi_4, \pi_8 \rangle$) = $\langle \pi_2, \pi_3, \pi_4, \pi_8 \rangle$.

7 žingsnis. Produkcija R globalią duomenų bazę iš būsenos DATA perveda į naują būseną RDATA.

8 žingsnis. Procedūra BACKTRACK yra kviečiama rekursyviai. Jos faktinis parametras yra 7 žingsnyje gauta nauja GDB būsena RDATA.

9 žingsnis. Jeigu 8 žingsnyje yra grąžintas nesėkmės požymis FAIL, tai reiškia, kad 5 žingsnyje paimta produkcija R neveda į terminalinę būseną. Tokiu atveju grįztama į 4 žingsnį. Bus imama kita produkcija.

10 žingsnis. Produkcijų sarašas yra grąžinamas kaip procedūros BACKTRACK rezultatas. Funkcija CONS prijungia elementą R prie sarašo PATH. Argumentų tipai yra CONS(atomas, sarašas). Pavyzdžiui, CONS($\pi_{j3}, \langle \pi_{j2}, \pi_{j1} \rangle$) = $\langle \pi_{j3}, \pi_{j2}, \pi_{j1} \rangle$. Paeiliui taikant produkcijas iš šio

sarašo, globali duomenų bazė yra pervedama iš pradinės būsenos į terminalinę būseną. Pavyzdžiu, $\pi_{j3}(\pi_{j2}(\pi_{j1}(\text{DATA})))$ yra terminalinė būsena.

3.1. Predikato DEADEND pavyzdys

Predikatas yra funkcija, kurios reikšmė yra *true* arba *false*. Kitais žodžiais, tai funkcija, kurios reikšmių aibė (dar vadinama funkcijos *kitimo sritimi*) yra $\{\text{true}, \text{false}\}$, t. y. – Būlio algebro tipo aibė. Argumentų aibė (dar vadinama *apibrėžimo sritimi*) nėra svarbi.

Predikatą DEADEND, kuris naudojamas procedūros BACKTRACK 2 žingsnyje, iliustruojame pavyzdžiu. Paimkime „15 kauliukų“ žaidimą (angl. 15-puzzle). Žaidimo esmė yra tokia: kauliukai sudedami į 4x4 dėžutę atsitiktine tvarka. Viena vieta lieka laisva. Kauliukas gali būti stumiamas keturiomis kryptimis: dešinėn, žemyn, kairėn ir aukštyn. Žaidėjo tikslas yra pasiekti terminalinę būseną, pateiktą 3.2 pav.



3.2 pav. a) Terminalinė būsena „15 kauliukų“ žaidime. b) Keturi tuščio langelio ėjimai – tai keturios produkcijos $\langle\pi_1, \pi_2, \pi_3, \pi_4\rangle$

Kauliuko stūmimas gali būti traktuojamas kaip tuščio lanelio stūmimas, bet priešinga kryptimi. Tokiu būdu yra keturios produkcijos: tuščias lanelis gali būti stumiamas kairėn, aukštyn, dešinėn arba žemyn.

Pradinė būsena realiame žaidime paprastai suformuojama šitaip: kauliukai išberiami ant stalo, sumaišomi ir atsitiktine tvarka sudedami atgal į dėžutę. Pradinės būsenos pavyzdys pateikiamas 3.3 a pav. Tiesa, šis pavyzdys nedaug skiriasi nuo terminalinės būsenos. Ši skirtumą sudaro tik viena perstata: kauliukai 1 ir 2 yra sukeisti vietomis.



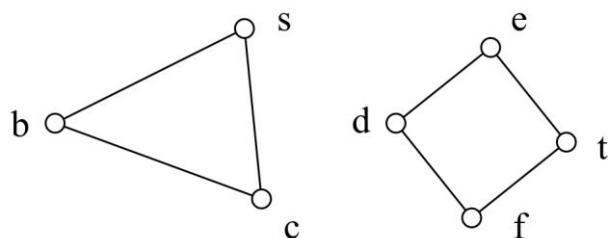
3.3 pav. a) Pradinės būsenos pavyzdys „15 kauliukų“ žaidime. Kauliukai 1 ir 2 yra sukeisti vietomis. Perstatę skaičius yra 1 lyginant su terminaline būsena (žr. 3.2 pav.). Kadangi perstatę skaičius yra nelyginis, tai ši būsena nėra sėkminga. b) Pradinės būsenos pavyzdys su lyginiu perstatu skaičiumi – dviem. Kauliukai 1 ir 2 bei 7 ir 8 yra sukeisti vietomis. Tokia būsena yra sėkminga

Kiek autorius prisimena iš įrodymo, prieš daugelį metų matyto moksleiviams skirtame matematinių uždavinių žurnale „Kvant“, tai „15 kauliukų“ uždavinys pasižymi tokia savybe.

Jeigu perstatų skaičius pradinėje būsenoje yra **nelyginis**, tai tokia būsena niekada nepasitaiko kelyje į terminalinę būseną. Jeigu perstatų skaičius pradinėje būsenoje yra **lyginis**, tai tokia būsena yra sėkminga (žr. 3.3 b pav.). Tokiu būdu, DEADEND (DATA) =true, jeigu perstatų skaičius yra nelyginis, ir false, jeigu lyginis. Plačiau žr. [Archer 1999].

Predikatas DEADEND (DATA) procedūriu būdu vaizduoja dalykinės srities gilias žinias (angl. *deep knowledge*), kada iš anksto žinoma, jog būsena DATA neveda į tikslą. Tokių žinių užprogramavimas leidžia sumažinti procedūros BACKTRACK paiešką (laiką ir atmintį) nesėkmingų pasirinkimų atvejais. Jeigu programuotojas tokių gilių žinių neturi, tai programuoja DEADEND (DATA) =false visais atvejais. Pastaruoju atveju procedūros BACKTRACK 2 žingsnis yra interpretuojamas kaip tuščias operatorius SKIP.

Kito uždavinio pavyzdys. Tegu ieškomas kelias grafe iš pradinės viršūnės s į terminalinę viršūnę t. Tada DEADEND (DATA) =true, jeigu pradinė viršūnė s ir terminalinė viršūnė t yra skirtinguose „kontinentuose“, t. y. nesusijusiose pografiuose. Pavyzdžiu, 3.4 pav. pateiktame grafe keliautojas, vien tik pažvelges į GDB ir neženges nė žingsnio, gali pasakyti, kad iš s į t nukeliauti neįmanoma, kadangi šios viršūnės yra skirtinguose „kontinentuose“.



3.4 pav. Kelias grafe iš pradinės viršūnės s į į terminalinę t neegzistuoja, nes šios viršūnės yra skirtinguose „kontinentuose“. Galima užprogramuoti: DEADEND (DATA) =true, jeigu pradinė s ir terminalinė t yra nesusijusiose pografiuose

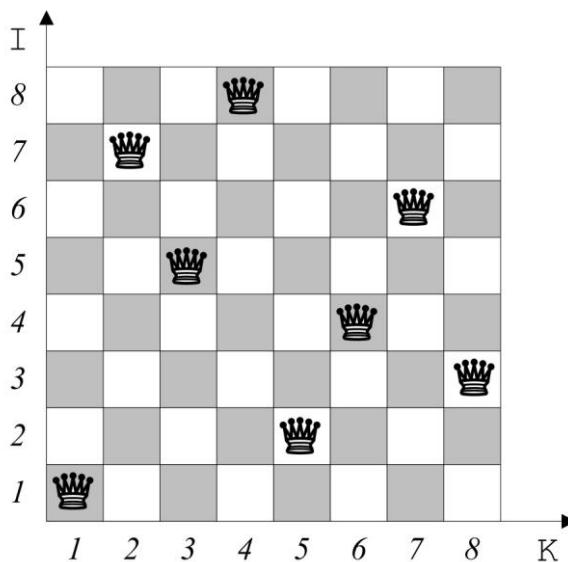
Pratimai

1. Ar įmanoma išspręsti kauliukų dėžutėje uždavinį, kai dėžutės plotis nelygus jos ilgiui?
2. Parašykite programą, kuri išspręstų kauliukų uždavinį duotų matmenų dėžutei.

4. Valdovių išdėstymo šachmatų lentoje uždavinys

Šis uždavinys yra žinomas kaip galvosūkis ir pratimas programuotojams. Aštuonias valdoves šachmatų lentoje reikia išdėstyti taip, kad jos nekirstų viena kitos.

Valdovių išdėstymą nusako vienmatis masyvas X , kur $X[I]=K$, čia I yra valdovės numeris kaip šachmatų eilutės numeris, o K – stulpelio, kuriame stovi valdovė, numeris. Šio uždavinio sprendinys egzistuoja. Vienas išdėstymų yra pateiktas 4.1 pav.: $X[1]=1$, $X[2]=5$, $X[3]=8$, $X[4]=6$, $X[5]=3$, $X[6]=7$, $X[7]=2$, $X[8]=4$.



4.1 pav. 8 valdovių išdėstymas šachmatų lentoje, kai nekerta viena kitos

Taigi uždavinio sprendinys yra koduojamas skaičių nuo 1 iki 8 perstata. Uždavinio sprendimui galima panaudoti „grubios jėgos“ (angl. *brute force*) algoritmą – perrinkti visas perstatas. Blogiausiu atveju, teks perrinkti 8 faktorialą $8! = 8 \cdot 7 \cdot 6 \cdots 2 \cdot 1 = 40320$ perstatų. $N \times N$ langelių šachmatų lentoje perstatų yra $N!$. Faktorialas yra aproksimuojamas eksponentinio sudėtingumo funkcija: $N! = 2^{\alpha \cdot N}$, kur α yra kažkoks realus skaičius, priklausantis nuo N , o N artėja į begalybę.

Valdovių uždaviniui spręsti taikysime procedūrą BACKTRACK. Pirmają valdovę statysime į pirmąją eilutę, antrają valdovę – į antrąją eilutę ir taip toliau iki aštuntosios. Produkcija π_k vaizduoja eilinės valdovės statymą į k -tajį stulpelį. Pavyzdžiu, aukščiau minėtą išdėstymą atitinka kelias iš pradinės būsenos (pradinėje būsenoje lenta tuščia):

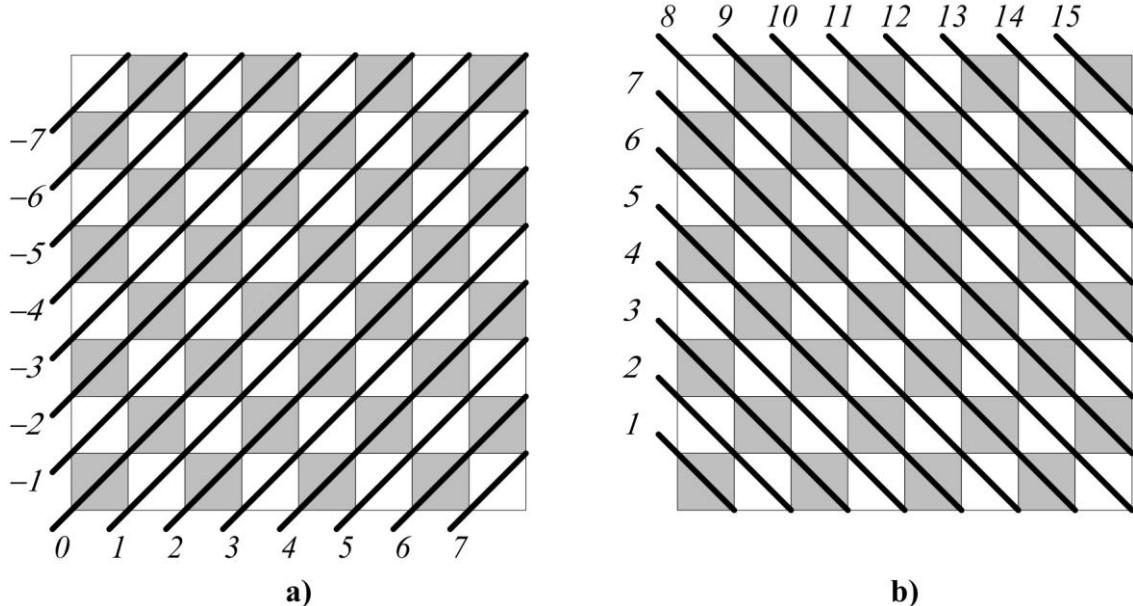
$$\text{PATH} = \langle \pi_1, \pi_5, \pi_8, \pi_6, \pi_3, \pi_7, \pi_2, \pi_4 \rangle$$

Pagal ši pažymėjimą kelio I -toji produkcija yra $\pi_{X[I]}$. Kitais žodžiais, $\text{PATH}[I] = \pi_{X[I]}$, kur $I = 1, 2, \dots, 8$.

Toliau pateikiama programa Paskalio kalba. Programa užrašyta atsižvelgiant į tekstą iš [Dagienė, Grigas, Augustis, 1986]. Šis uždavinys taip pat nagrinėjamas vikipedijoje (žr. http://en.wikipedia.org/wiki/Eight_queens_puzzle).

Uždavinio ir programos esmė nesikeičia, kai lento dydis yra ne 8×8 , o $N \times N$ langelių.

Programoje naudojamos kylančių ir besileidžiančių įstrižainių sąvokos. Yra tikrinama, ar valdovės nekerta per įstrižainę. 4.2 pav. iliustruoja kylančias įstrižaines, kurios sunumeruotos nuo -7 iki 7 (bendru atveju nuo $-(N-1)$ iki $(N-1)$), o 4.2 b pav. iliustruoja besileidžiančias įstrižaines, kurios sunumeruotos nuo 1 iki 15 (bendru atveju nuo 1 iki $2 \cdot N-1$). Pavyzdžiui, atsižvelgiant į programos kintamuosius I ir K , jeigu $I=2$, $K=3$, tai vertikalė yra $K=3$, kylančioji įstrižainė $K-I = 3-2 = 1$, ir besileidžiančioji įstrižainė $I+K-1 = 2+3-1 = 4$.



4.2 pav. a) Kylančios įstrižainės valdovių uždavinyje yra sunumeruotos nuo -7 iki 7 . Pagrindinė įstrižainė yra nulinė.
 b) Besileidžiančios įstrižainės yra sunumeruotos nuo 1 iki 15 .
 Pagrindinė įstrižainė yra aštuntoji

```

program VALDOVĖS;
const   N      = 8;
          NM1   = 7; {N-1}
          N2M1 = 15; {2*N-1}
var
  X      : array [1 .. N] of integer;
  VERT   : array [1 .. N] of boolean;           {Vertikalės}
  KYL    : array [-NM1 .. NM1] of boolean; {Kylančios istrižainės}
  LEID   : array [1..N2M1] of boolean;        {Besileidžiančios istrižainės}
  YRA    : boolean;
  I, J   : integer;
  BANDSK : longint;

procedure TRY (I : integer; var YRA : boolean);
{Iėjimas I - éjimo numeris. Išėjimas YRA - ar pavyko sustatyti}
  var K : integer;
begin
  K := 0;
  repeat
    K := K + 1; BANDSK := BANDSK + 1;
    if VERT[K] and KYL[K - I] and LEID[I + K - 1]
    then {Vertikalė, kylanti ir besileidžianti istrižainės yra laisvos}
    begin
      X[I] := K;
      VERT[K] := false; KYL[K - I] := false; LEID[I + K - 1] := false;
      if I < N then
        begin
          TRY(I + 1, YRA);
          if not YRA
          then {Nerastas kelias toliau}
            begin
              VERT[K] := true; KYL[K - I] := true;
              LEID[I + K - 1] := true; {Atlaivinamas langelis}
            end;
        end
        else YRA := true;
      end;
    until YRA or (K = N);
end; {TRY}

begin {Pagrindinė programa - main program}
  {1. Inicializuojami kintamieji}
  for J := 1 to N do VERT[J] := true;
  for J := -NM1 to NM1 do KYL[J] := true;
  for J := 1 to N2M1 do LEID[J] := true;
  YRA := false; BANDSK := 0;
  {2. Kviečiama procedūra}
  TRY(1, YRA);
  {3. Spausdinama lenta}
  if YRA then
    begin
      for I := N downto 1 do
      begin
        for J := 1 to N do
          if X[I] = J then write(1 : 3) else write(0 : 3);
          writeln;
      end;
      writeln('Bandymų skaičius: ', BANDSK);
    end
  else writeln('Sprendinys neegzistuoja');
end.

```

5. Euristika

Valdovių uždavinyje atliekamas pilnas produkcijų perrinkimas. Siekiant išvengti pilno perrinkimo galima pasiūlyti tokią taisykłę pasirinkti stulpelį, į kurį statoma valdovė. Pasirenkamas tas langelis, kuriamo valdovė kerta mažiausiai per ilgesnį iš dviejų ištrižainių.

Euristika – tai praktikos padikuota pasirinkimo taisykla (angl. *rule of thumb*), skirta rasti sprendimą be varginančios paieškos (paprastai suprantamos kaip perrinkimas).

Būdvardis **euristinis** (angl. *heuristic*) apibūdina kiekvieną sprendimo būdą, kuris pagerina uždavinio sprendimo vidutinę charakteristiką, bet nebūtinai blogiausio atvejo charakteristiką [Russell, Norvig, 2003, p. 94].

Euristikos savyoka kildinama iš graikiško žodžio „heuriskein“ (atrasti). Legenda byloja, kad senovės graikų filosofas Archimedes sušuko „eureka“ (atradau), kai suvokė apie jėgą, kuri išstumia kietą kūną, panardintą į skystį. Dabar ši jėga vadinama *Archimedo jėga*.

Anglų kalboje žodis „heuristic“ yra vartojamas ir kaip būdvardis, ir kaip daiktavardis. Žodžio „euristika“ reikšmės aiškinimas remiantis anglų kalbos žodynu *Oxford English Dictionary* (žr. <http://www.oed.com/>):

euristinis (angl. *heuristic*)

- a. Skirtas išsiaiškinti arba atrasti (Serving to find out or discover) ...
- c. *kompiuteriai* ... 1964 T. W. McRae ... „Euristinio“ programavimo procedūra kompiuteris atlieka paiešką tarp galimų sprendimų kiekvienoje programos stadijoje, įvertina šiai stadijai „gerą“ sprendimą ir pereina prie kitos stadijos. Iš esmės euristinis programavimas yra panašus į uždavinį sprendimą bandymu ir klaidų būdais, kuriais mes naudojamės kasdienybėje. ... (*Computers* ... 1964 T. W. McRae ... Under an ‘heuristic’ programming procedure the computer searches through a number of possible solutions at each stage of the programme, it evaluates a ‘good’ solution for this stage and then proceeds to the next stage. Essentially heuristic programming is similar to the problem solving techniques by trial and error methods which we use in everyday life. ...)

euristika (angl. *heuristic*)

- b. ... Euristinis procesas arba metodas mėginančių uždavinio sprendimą; taisykla arba informacijos vienetas naudojamas tokiam procese. ... 1957 A. Newell et al. ... Procesas, kuris gali išspręsti duotą uždavinį, bet negarantujantis išsprendimo, yra vadinamas šio uždavinio euristika. *Ten pat* Glausciai, mes naudosime „euristika“ kaip termino „euristinis procesas“ sinonimą. ... (A heuristic process or method for attempting the solution of a problem; a rule or item of information used in such a process. ... 1957 A. Newell et al. ... A process that may solve a given problem, but offers no guarantees of doing so, is called a heuristic for that problem. *Ibid.* For conciseness, we will use ‘heuristic’ as a noun synonymous with ‘heuristic process’. ...)

Dirbtinis intelektas ankstyvojoje vystimosi fazėje kartais buvo vadinamas *euristiniu programavimu* (angl. *heuristic programming*). Kaip alternatyva grubios jėgos (angl. *brute force*) algoritmų programavimui, kuris suprantamas kaip perrinkimas (paprastai eksponentinio sudėtingumo), buvo siūlomas euristinis programavimas.

Euristika gali būti grindžiama ir ankstesniu patyrimu. Jeigu panašus uždavinys jau buvo spręstas ankčiau arba netgi ankstesnis sprendimas buvo nesėkminges, tai patirtis gali padėti.

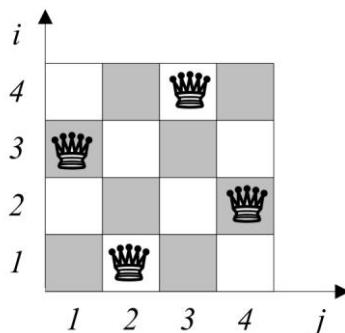
Pavyzdžiui, vairuotojas atkreipia dėmesį į savo automobilio blogą variklio darbą – variklis trūkčioja. Vairuotojas iš karto nesuvokia gedimo priežasties. Tačiau automobilį jis vairuoja daugelį metų ir prisimena, kad panaši situacija buvo iškilusi anksčiau. Tą kartą automobilį sutaisės mechanikas pasakė, kad gedimas buvo jungiamojo vamzdelio sandarumo pažeidimas, dėl kurio į variklį patekdavo oro ir nurodė gedimo vietą. Taigi vairuotojas sustabdė automobilį ir pastebėjo, kad tas pats vamzdelis vėl nesandarus. Todėl nusprendė ji sutvarkyti.

Euristika skirta pasiūlyti sprendimą, bet be garantijos. Kadangi neatliekama gili analizė, tai gali būti pateikiamas ir neteisingas sprendimas arba jo iš viso nerandama. Nesėkmė pagrista tuo, kad euristika pateikia „priimtiną“ variantą, o ne teisingą atsakymą. Euristika gali būti labiau priimtinės variantas negu tikslus sprendimas šiais atvejais:

- 1) yra per daug variantų, kuriuos reikia patikrinti;
- 2) kiekvieno varianto įvertinimo funkcija yra per daug sudėtinga;
- 3) įvertinimo funkcija yra nežinoma ir apytikslis įvertinimas gali atlkti euristikos vaidmenį.

5.1. Euristika valdovių uždavinyje

Euristika toliau iliustruojama valdovių išdėstymo uždaviniu (žr. 4 skyrių). Naudojant euristiką, bus sumazintas valdovių išdėstymo variantų skaičius. Paprastumo dėlei paimkime 4×4 langelių lentą. Joje valdoves išdėstyti įmanoma, pavyzdžiui:



5.1 pav. Vienas iš galimų nesikertančių valdovių išdėstymų 4×4 lentoje

Euristika, kuri suprantama kaip vienos eilutės produkcijų sutvarkymas, imama tokia:

$$\pi_{i,j} \leq \pi_{i,k}, \text{ jeigu } diag(i,j) \leq diag(i,k),$$

kur $diag(i,j)$ yra ilgis ilgesnės iš kylančios ir besileidžiančios įstrižainės, einančių per langelį $[i,j]$. Formaliai, $diag(i,j) = \max(kyl(i,j), leid(i,j))$. Čia $kyl(i,j)$ yra kylančios įstrižainės einančios per langelį $[i,j]$ ilgis, o $leid(i,j)$ – atitinkamos besileidžiančios įstrižainės ilgis.

Euristikos pagrindimas: kuo daugiau vadovė kerta per įstrižainę, tuo ji pavojingesnė kitoms valdovėms. Todėl pirmenybė teikiama valdovės statymui į tokį langelį, kuriame ji būtų mažiausiai pavojinga. Šią euristiką pavadiname „ilgesnės įstrižainės“ euristika. Toliau ji aiškinama remiantis [Nilsson 1982].

Ilgis kylančios įstrižainės, einančios per langelį $[1,1]$, yra 4, o besileidžiančios 1. Taigi:

$$diag(1,1) = \max(4,1) = 4$$

Analogiškai apskaičiuojame ilgius įstrižainių, einančių per kitus pirmosios eilutės langelius, t. y. $diag(1,j), j=1,\dots,4$:

$$diag(1,1)=4; diag(1,2)=\max(3,2)=3; diag(1,3)=\max(2,3)=3; diag(1,4)=\max(1,4)=4$$

Iš čia matome, kad

$$diag(1,2) \leq diag(1,3) \leq diag(1,1) \leq diag(1,4)$$

Taigi vadovaudamiesi aukščiau apibrėžta euristika, sutvarkome produkcijas valdovių statymui į pirmają eilutę:

$$\pi_{1,2} \leq \pi_{1,3} \leq \pi_{1,1} \leq \pi_{1,4}$$

Analogiškai apskaičiuojame ilgius įstrižainių, einančių per antrosios eilutės langelius, t. y. $diag(2,j), j=1,\dots,4$:

$$diag(2,1)=\max(3,1)=3; diag(2,2)=\max(4,3)=4; diag(2,3)=\max(3,4)=4; \\ diag(2,4)=\max(2,3)=3$$

Vadovaudamiesi euristika sutvarkome produkcijas valdovių statymui į antrają eilutę:

$$\pi_{2,1} \leq \pi_{2,4} \leq \pi_{2,2} \leq \pi_{2,3}$$

Analogiškai apskaičiuojame ilgius įstrižainių, einančių per trečiosios eilutės langelius, t. y. $diag(3,j), j=1,\dots,4$:

$$diag(3,1)=\max(2,3)=3; diag(3,2)=\max(3,4)=4; diag(3,3)=\max(4,3)=4 \\ diag(3,4)=\max(3,2)=3$$

Vadovaudamiesi euristika sutvarkome produkcijas valdovių statymui į trečiąją eilutę:

$$\pi_{3,1} \leq \pi_{3,4} \leq \pi_{3,2} \leq \pi_{3,3}$$

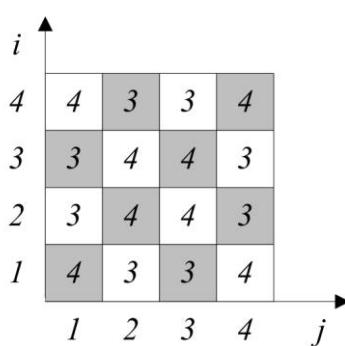
Analogiškai apskaičiuojame ilgius įstrižainių, einančių per ketvirtosios eilutės langelius, t. y. $diag(4,j), j=1,\dots,4$:

$$diag(4,1)=\max(1,4)=4, diag(4,2)=\max(2,3)=3, diag(4,3)=\max(3,2)=3 \\ diag(4,4)=\max(4,1)=4$$

Vadovaudamiesi euristika sutvarkome produkcijas valdovių statymui į ketvirtąją eilutę:

$$\pi_{4,2} \leq \pi_{4,3} \leq \pi_{4,1} \leq \pi_{4,4}$$

Funkcijos $diag(i,j)$ reikšmės pateikiamas 5.2 pav.



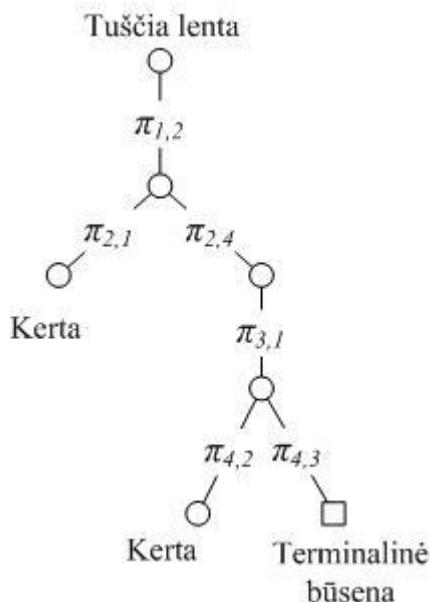
5.2 pav. Ilgesnės iš dviejų įstrižainių (kylančios ir besileidžiančios), einančių per langelį $[i,j]$, ilgio $diag(i,j)$ priskyrimas 4×4 lentos langeliams

Reziumuojamas visų keturių eilučių produkcijų sutvarkymas:

$$\begin{aligned}
 \pi_{1,2} \leq \pi_{1,3} \leq \pi_{1,1} \leq \pi_{1,4} & \quad - \text{ pirmoji eilutė} \\
 \pi_{2,1} \leq \pi_{2,4} \leq \pi_{2,2} \leq \pi_{2,3} & \quad - \text{ antroji eilutė} \\
 \pi_{3,1} \leq \pi_{3,4} \leq \pi_{3,2} \leq \pi_{3,3} & \quad - \text{ trečioji eilutė} \\
 \pi_{4,2} \leq \pi_{4,3} \leq \pi_{4,1} \leq \pi_{4,4} & \quad - \text{ ketvirtirtoji eilutė}
 \end{aligned}$$

5.3 lentelė. Produkcijų sutvarkymas pagal euristiką valdovėms 4×4 lentoje

Remdamiesi šiuo produkcijų sutvarkymu, gauname paieškos medį, pateiktą 5.4 pav.

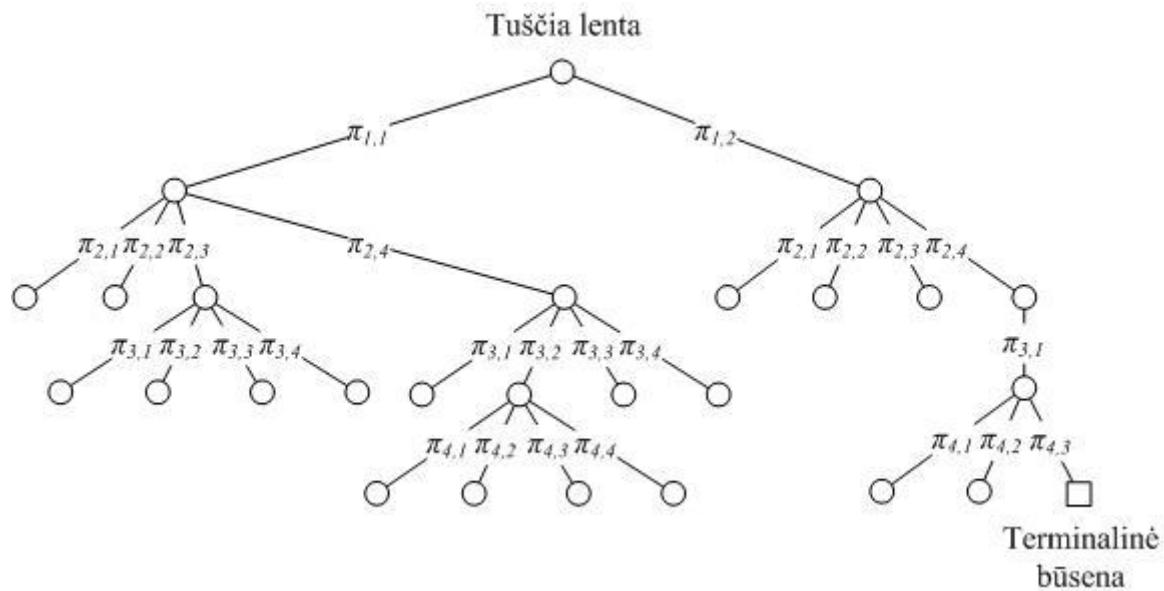


5.4 pav. Paieškos medis, kai remiamasi euristiniu produkcijų sutvarkymu.
Briaunų skaičius – kaip bandymų skaičius – yra 6

Algoritmo (strategijos) sudėtingumo matas yra bandymų skaičius. Medžio atveju – tai briaunų skaičius.

Paaiškinsime medį 5.4 pav. ir valdovių statymą, kai remiamasi produkcijų sutvarkymu pagal euristiką. Pirmoji valdovė statoma į langelį [1,2] (t. y. $[i=1, j=2]$), nes remiantis 5.2 lentele pirmoji eilutė pradedama $\pi_{1,2}$. Analogiškai antrają valdovę bandome statyti į langelį [2,1], nes antroji eilutė pradedama $\pi_{2,1}$. Tačiau čia ją kertą prieš tai pastatyta pirmoji valdovė. Todėl antrają valdovę statome į langelį [2,4], nes $\pi_{2,4}$ yra sekanti pagal sutvarkymą antrosios eilutės produkcija. Trečioji valdovė statoma į langelį [3,1], nes trečioji eilutė pradedama $\pi_{3,1}$. Ketvirtąją valdovę bandome statyti į langelį [4,2], nes ketvirtirtoji eilutė pradedama $\pi_{4,2}$. Tačiau čia ją kertą ir pirmoji, ir trečioji valdovės. Todėl ketvirtąją valdovę statome į langelį [4,3], nes $\pi_{4,3}$ yra sekanti pagal sutvarkymą ketvirtosios eilutės produkcija. Čia pastebime, kad pasiekėme terminalinę būseną – valdovės nekerta viena kitos. Bandymų skaičius, kaip matyti 5.2 pav., yra 6.

Jeigu nesiremtume euristiniu produkcijų sutvarkymu, t. y. eilutėje i produkcijos būtų tvarkomos j didėjimo tvarka ($\pi_{i,1} \leq \pi_{i,2} \leq \pi_{i,3} \leq \pi_{i,4}$), tai paieškos medis būtų kaip 5.5 pav.



5.5 pav. Paieškos medis valdovų išdėstymui 4×4 šachmatų lentoje pilnu perrinkimu, t. y. kai nesiremiamą jokią euristika. Briaunų skaičius – kaip bandymų skaičius – yra 26

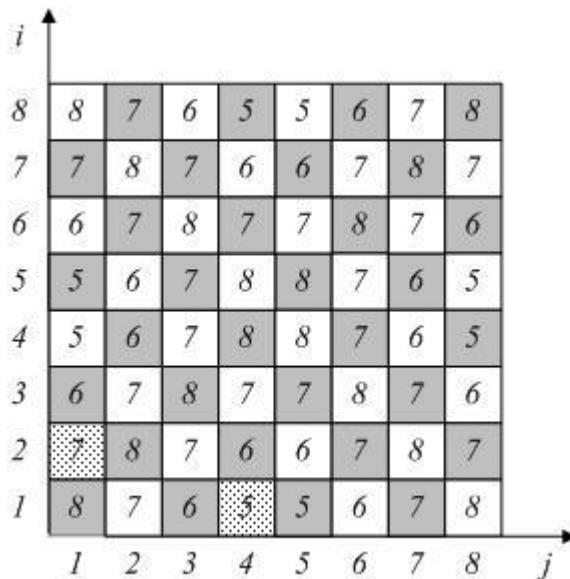
Pirmoji valdovė statoma į langelį $[1,1]$ (t. y. $[i=1, j=1]$) pagal produkciją $\pi_{1,1}$. Antrają valdovę statome į $[2,1]$ pagal $\pi_{2,1}$. Tačiau čia ją kerta pirmoji valdovė. Todėl antrają valdovę statome pagal $\pi_{2,3}$. Trečiąją valdovę statome pagal $\pi_{3,1}$. Tačiau čia ją kerta pirmoji valdovė. Toliau trečiosios valdovės negalima statyti ir pagal jokią iš $\pi_{3,1}, \pi_{3,2}, \pi_{3,3}, \pi_{3,4}$, nes visur kerta. Todėl tenka grįžti per vieną eilutę ir antrają valdovę statyti pagal $\pi_{2,4}$. Ir taip toliau. Rezultatas yra paieškos medis 5.5 pav. su terminaline būseną.

Paieškos efektyvumo matas yra briaunų skaičius paieškos medyje. Palygine paieškos medžius 5.4 pav. ir 5.5 pav. matome, kad euristinė paieška yra efektyvesnė. Paieškos medis 5.4 pav. turi 6 briaunas, o paieškos medis 5.5 pav. – 26 briaunas.

5.2. Dar viena euristika: valdovę statyti žirgo ējimu

Žvelgiant į valdovių uždavinio sprendinius, galima padaryti „atradimą“, kad valdovės gali būti bandomos statyti žirgo ējimu. I tokią statymo taisyklę žiūrėsime kaip į dar vieną euristiką, vadinamą „valdovės statymas žirgo ējimu“ (sutrumpintai „žirgo ējimas“).

Šios euristikos tyrimą pradėjo 2005 m. pradėjo Justas Arasimavičius būdamas studentu ir klausydamas paskaitą, o pratesė Edgaras Abromaitis 2008 m. Šio euristikos efektyvumas pranoko lūkesčius. Perrinkimas yra tiesinis su nedidelėmis išimtimis, kai lentos $N \times N$ dydis auga. Tokio gero rezultato iki tyrimo net nebuvo tikėtasi.

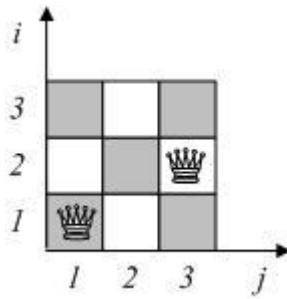


5.6 pav. Funkcijos $diag(i,j)$ – ilgesnės iš dviejų įstrižainių, einančių per langelį $[i,j]$ – reikšmės 8×8 šachmatų lento langeliuose

Poskyryje 5.1 buvo nagrinėta „ilgesnės įstrižainės“ euristika. Pagal ją, valdovė bandoma statyti į tokį šachmatų lento $N \times N$ langelį, kurio ilgiausioji įstrižainė iš dviejų būtų kiek galima trumpesnė. Paveiksle 5.6 pav. yra pateiktas šachmatų lento pavyzdys, kur kiekvienam langeliui priskirtas ilgesnės įstrižainės ilgis: $diag(1,1)=8$, $diag(2,1)=7$, ir t.t. Čia $diag(i,j)$ yra ilgesnės įstrižainės, einančios per langelį $[i,j]$, ilgis.

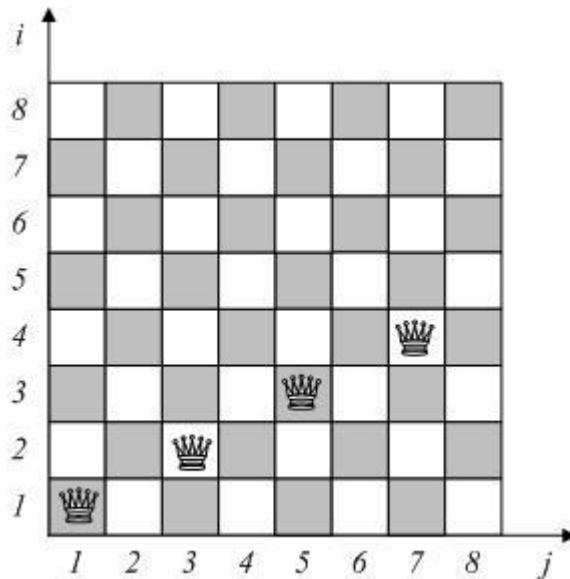
8×8 šachmatų lentoje, vadovaujantis „ilgesnės įstrižainės“ euristiką, pirmoji valdovė statoma į langelį [1,4]. Antroji valdovė bandoma statyti antrojoje eilutėje į ketvirtąjį stulpelį, tačiau čia ją kerta. Tada bandoma į penktąjį – ir vėl kerta. Tada į pirmąjį – jis nekertamas. Ir taip toliau, kol pasiekiamama terminalinė būsena.

Toliau pristatomą valdovės statymo euristiką „žirgo ējimas“. $N \times N$ šachmatų lentoje, valdovė statoma šachmatų žirgo ējimu „du langeliai kairėn, vienas langelis į viršų“ prieš tai pastatytos valdovės atžvilgiu (žr. 5.7 pav.).



5.7 pav. Valdovės statymas žirgo éjimu. Antroji valdovė statoma šachmatų žirgo éjimu „du langeliai kairėn, vienas lanelis į viršu“

Vienu iš euristikos motyvų yra pastebėjimas, kad 8×8 šachmatų lentoje galima padaryti net keturis tokius ejimus iš eilės (žr 5.8 pav.). Kitu motyvu yra pastebėjimas, kad keleto valdovių išsidėstymas žirgo éjimu pasitaiko tarp išdėstymo variantų, kuriuos perrenka valdymo su grįzimais procedūra BACKTRACK ir programa VALDOVĖS, pateikta 4 skyriuje.



5.8 pav. Vienu iš valdovės statymu euristika „žirgo éjimas“ motyvų yra pastebėjimas, kad 8×8 šachmatų lentoje galima padaryti net keturis tokius ejimus iš eilės

Apibrėžkime produkcių aibę euristikai „žirgo éjimai“. Ši aibė bus sudaryta iš triju produkcių: π_1, π_2, π_k .

π_1 – pirmosios valdovės statymo produkcija, pirmoji valdovė bus statoma į antrajį [2,1] arba ketvirtąjį [4,1] lentelės $N \times N$ pirmosios eilutės langelį. Į ketvirtąjį langelį statysime, tuo atveju, jei lentelės dydis bus skaičius apibrėžiamas pagal taisykłę $6n+3$ (čia $n = 1, 2, 3, \dots$), visais kitais atvejais valdovę statysime į antrajį lentelės $N \times N$ pirmosios eilutės langelį.

$$\pi_1 = \begin{cases} \bullet \text{ pirmają valdovę statyti į langelį } [i=1, j=2] - \text{kai } N \neq 6n+3; \\ \bullet \text{ pirmają valdovę statyti į langelį } [i=1, j=4] - \text{kai } N = 6n+3. \end{cases}$$

Šios pozicijos pasirinktos neatsitiktinai, tai rezultatas pirmosios pozicijos paieškos tyrimo atlikto pastebėjus valdovės éjimo žirgu efektyvumą bei siekiant minimalaus perrinkimo sprendžiant ši uždavinį (žr. 5.23 lentelę). Šio bandymo rezultatai parodė, kad antra ir ketvirta pozicijos (anksčiau minetu atveju) padengia nemažai lentelių.

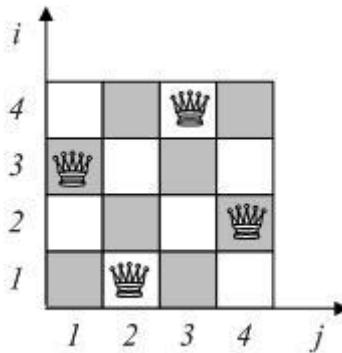
π_z – esminė euristikos produkcija, jau apibūdinta 5.7 paveiksle. Jeigu esame langelyje $[i,j]$, tai valdovę statome į langelį $[i+1,j+2]$, jei toks lanelis priklauso lentai, ir į langelį $[i+1,I]$, jei nepriklauso (t. y. $j+2 > N$).

$$\pi_z = \begin{cases} \bullet \text{ eilinę valdovę statyti į langelį } [i+1,j+2], \text{ kur } [i,j] \text{ yra paskutinė pastatyta valdovė, – kai šiuo statymu neišeinama už lentos ribų;} \\ \bullet \text{ eilinę valdovę statyti į langelį } [i+1,j=I] – \text{ kai } [i+1,j+2] \text{ išeina už lentos ribų.} \end{cases}$$

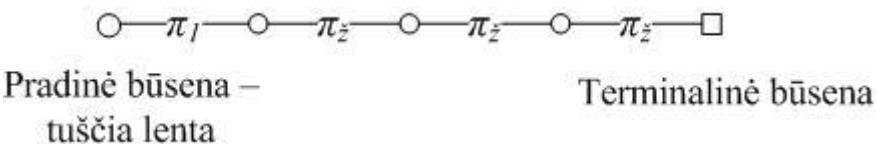
π_k – produkcija taikoma tada, kai valdovės negalima pastatyti pagal produkciją π_z (valdovė šios produkcijos siūlomame langelyje yra kertama). π_k produkcijos rezultatas yra tiesiog kitas greta esantis lanelis. Jeigu esame pozicijoje $[i,j]$, tai kitas lanelis yra $[i,j+1]$, tiksliau $[i,(j \bmod N) + 1]$, jei toks lanelis priklauso lentai, arba $[i,1]$, jei nepriklauso. Ši produkcija taikoma tol, kol patikrinami visi vienos eilutės laneliai, taigi nedaugiau kaip N kartų.

$$\pi_k = \begin{cases} \bullet \text{ eilinę valdovę statyti į langelį } [i,(j \bmod N) + 1], \text{ kur } [i,j] \text{ yra ankstesnis bandymas toje pačioje eilutėje. Galima sakyti, kad eilinė valdovė statoma į } [i,j+1], \text{ bet cikliškai, t. y. permetant į } j=1, \text{ jei } j+1 > N \text{ (t. y išeinama už krašto).} \end{cases}$$

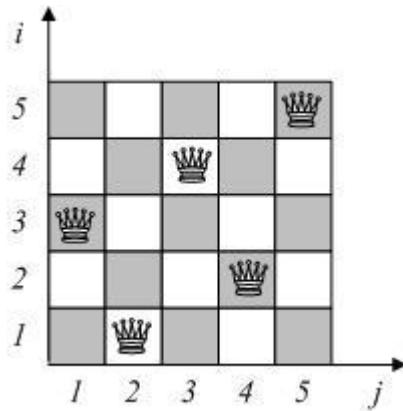
Paieškos medžiai ir sprendiniai lentoms taikant euristiką „žirgo éjimai“, kurių dydžiai nuo $N=4$ iki $N=9$, pateikti nuo 5.9 pav. iki 5.21 pav.



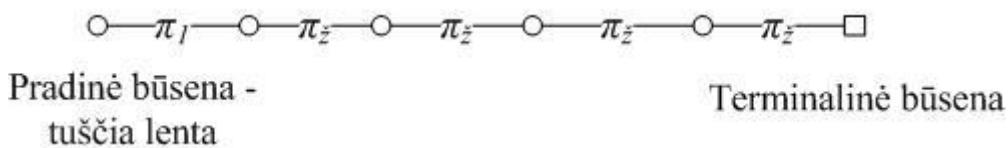
5.9 pav. Valdovių išdėstymo terminalinė būsena 4×4 šachmatų lentoje



5.10 pav. Valdovių išdėstymo 4×4 šachmatų lentoje paieškos medis, kai naudojama "žirgo ėjimo" euristika



5.11 pav. Valdovių išdėstymo terminalinė būsena 5×5 šachmatų lentoje



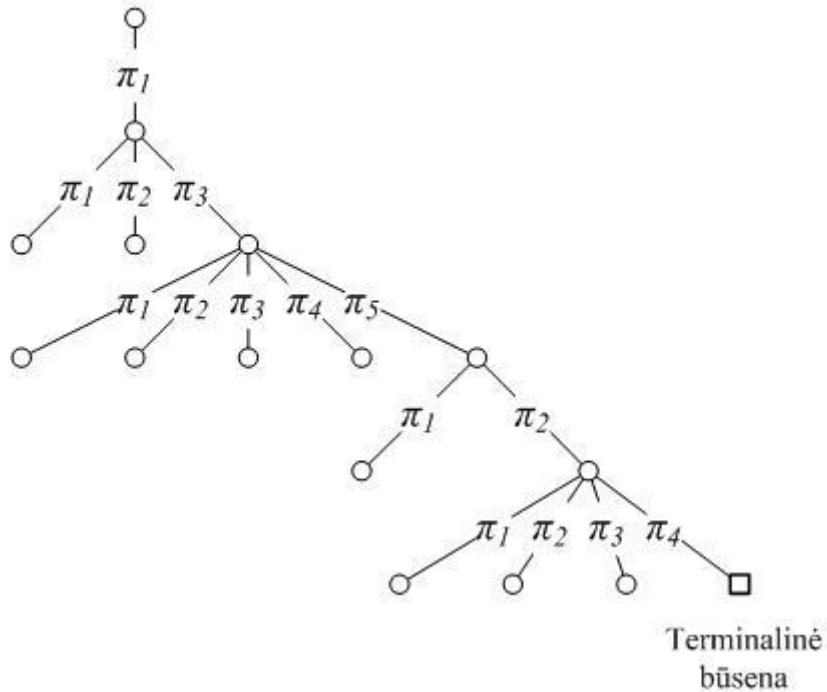
Pradinė būsena -
tuščia lenta

Terminalinė būsena

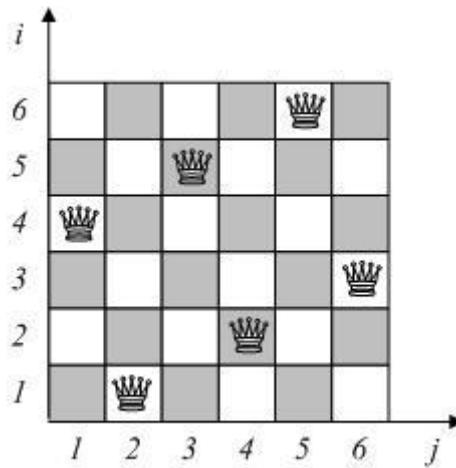
5.12 pav. Valdovių išdėstymo 5×5 šachmatų lentoje paieškos medis, kai naudojama "žirgo ėjimo" euristika

Palyginimui pateiksime 5.13 pav pateikiamas paieškos medis „grubios jėgos“ (angl. *brute force*) perrinkimui, t. y. be jokios euristikos 5×5 lentoje.

Pradinė būsena – tuščia lenta



5.13 pav. Valdovių išdėstymo 5×5 šachmatų lentoje paieškos medis, kai naudojamas „grubios jėgos“ (angl. *brute force*) perrinkimas



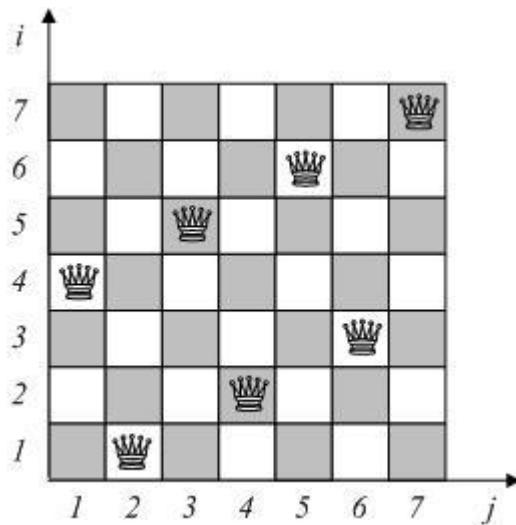
5.14 pav. Valdovių išdėstymo terminalinė būsena 6×6 šachmatų lentoje

$\circ - \pi_1 - \circ - \pi_2 - \square$

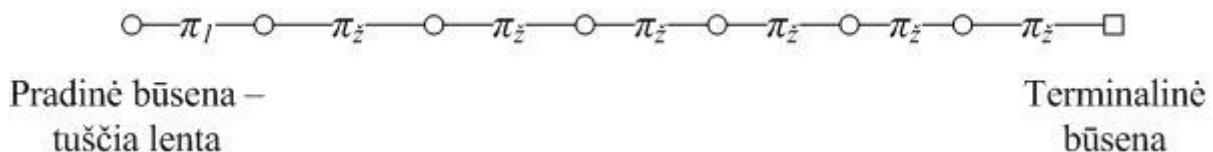
Pradinė būsena –
tuščia lenta

Terminalinė būsena

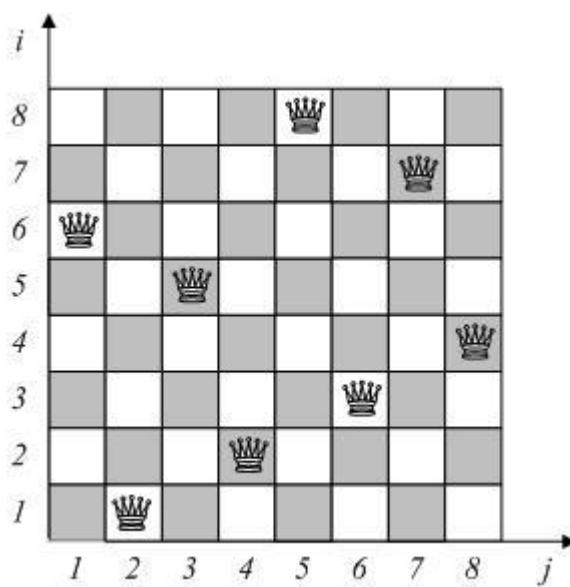
5.15 pav. Valdovių išdėstymo 6×6 šachmatų lentoje paieškos medis, kai naudojama "žirgo ėjimo" euristika



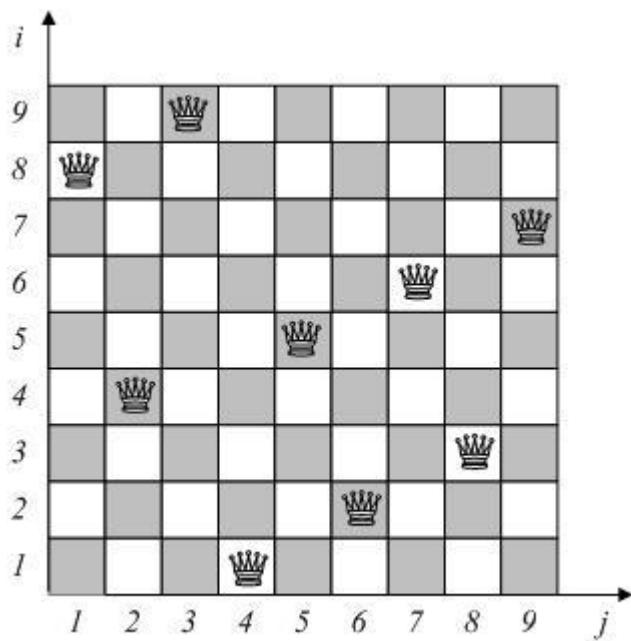
5.16 pav. Valdovių išdėstymo terminalinė būsena 7×7 šachmatų lentoje



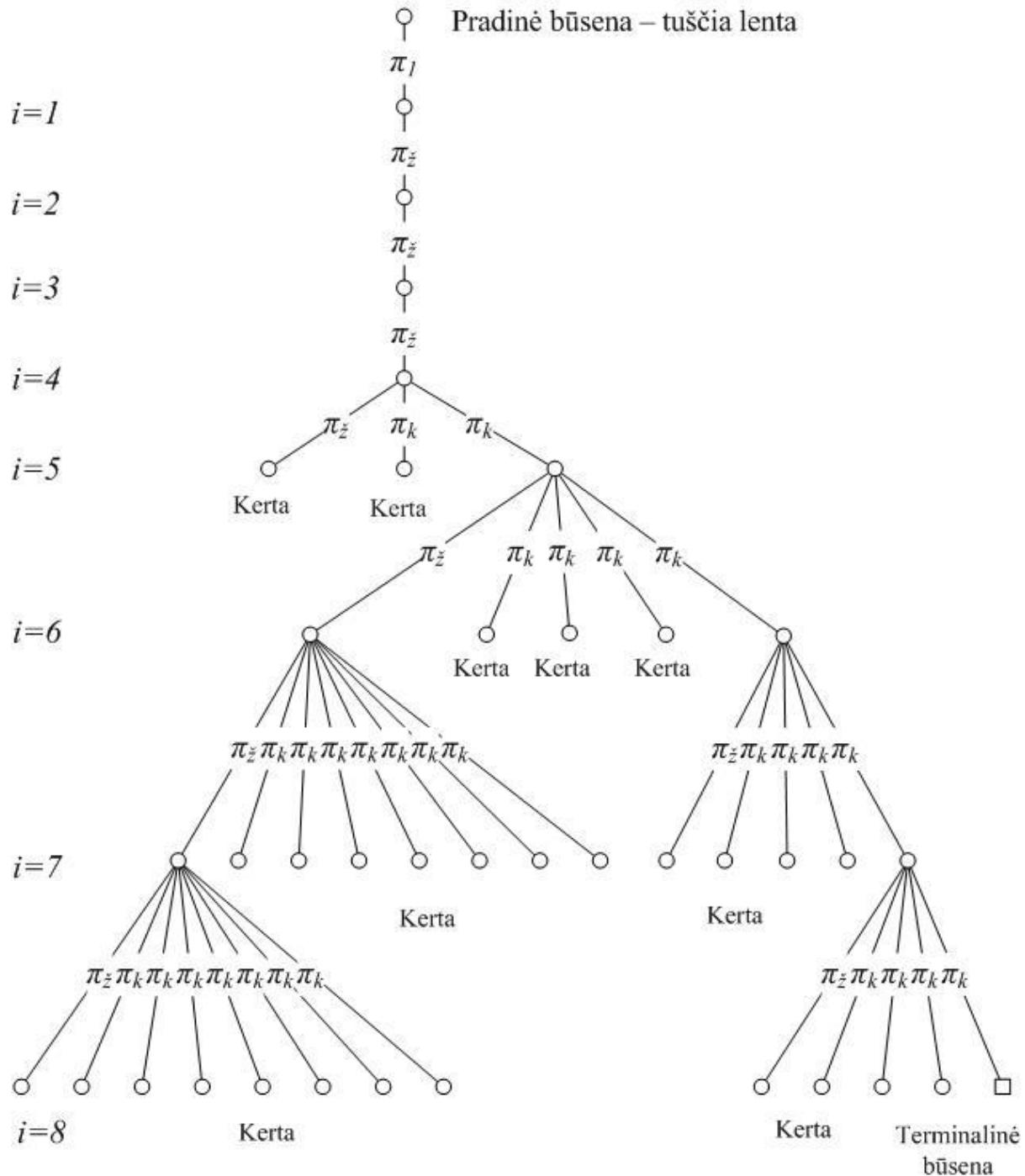
5.17 pav. Valdovių išdėstymo 7×7 šachmatų lentoje paieškos medis, kai naudojama "žirgo ėjimo" euristika



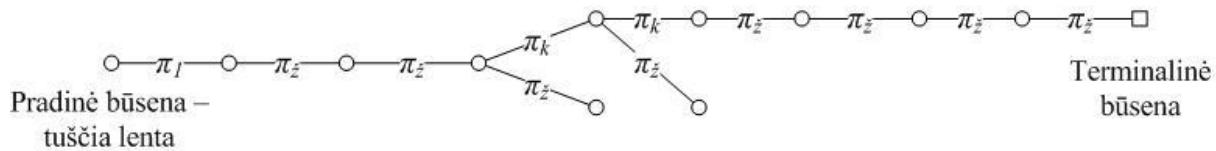
5.18 pav. Valdovių išdėstymo terminalinė būsena 8×8 šachmatų lentoje



5.20 pav. Valdovių išdėstymo terminalinė būsena 9×9 šachmatų lentoje



5.19 pav. Valdovių išdėstymo 8×8 šachmatų lentoje paieškos medis, kai naudojama "žirgo ėjimo" euristika



5.21 pav. Valdovių išdėstymo 9×9 šachmatų lentoje paieškos medis, kai naudojama "žirgo éjimo" euristika

5.18 pav. pirmieji keturi éjimai atlikti žirgo euristika, o nuo penktosios eilutės ši euristika derinama su produkcija π_k , t. y su perrinkimu. Paieškos évertis lentai 8×8 lygus 38 (žr. 5.2 lentelės, eilutę $N=8$). Tai atrodytų didelis skaičius, tačiau paieškos évertis, kai naudojama trumpiausios įstrižainės euristika, yra 204. O visiško perrinkimo évertis lygus 876.

Paimkime 9×9 lentą. Šios lento dydis tenkina savybę $N = 6n+3$. Taigi, pirmaja produkcija pasirenkamas ketvirtasis langelis (žr. 5.20 pav.). Toliau π_z derinama su π_k . Paieškos medyje tik dvi šakos veda į aklavietę. Taigi perrinkimas labai nedidelis. Paieškos évertis lygus 11 (žr. 5.2 lentelės, eilutę $N=9$).

Algoritmo vertinamo matas yra perrinkimų skaičius, kuris yra suprantamas kaip briaunų skaičius paieškos medyje. 5.22 lentelėje pateikiamas perrinkimų skaičius trims algoritmams: perrinkimas be euristikos, „trumpiausios įstrižainės“ euristika, „žirgo éjimo“ euristika.

Lentos dydis N	Perrinkimų skaičius – briaunų skaičius paieškos medyje		
	Perrinkimas be euristikos (angl. <i>brute force</i>)	„Trumpiausios įstrižainės“ euristika	„Žirgo éjimo“ euristika
4	26	6	4
5	15	15	5
6	171	69	6
7	42	87	7
8	876	204	38
9	333	874	11
10	975	437	10
11	517	200	11
12	3066	297	12
13	1365	684	13
14	26495	1742	300
15	20280	487	17
16	160712	111	16
17	91222	294	17
18	743229	7130	18
19	48184	24714	19
20	3992510	918	372
21	179592	48222	23
22	38217905	40744	22
23	584591	10053	23
24	9878316	5723	24
25	1216775	2887	25
26	10339849	265187	2196
27	12263400	986476	29
28	84175966	2602283	28
29	44434525	1261296	29

30	1692888135	52601	30
31	Daug	1850449	31
32	Daug	2804692	2866
33	Daug	2582396	35
34	Daug	35784	34
35	Daug	110473	35
36	Daug	19605979	36
37	Daug	135980	37
38	Daug	642244758	30532
39	Daug	193745	41
40	Daug	4685041	40

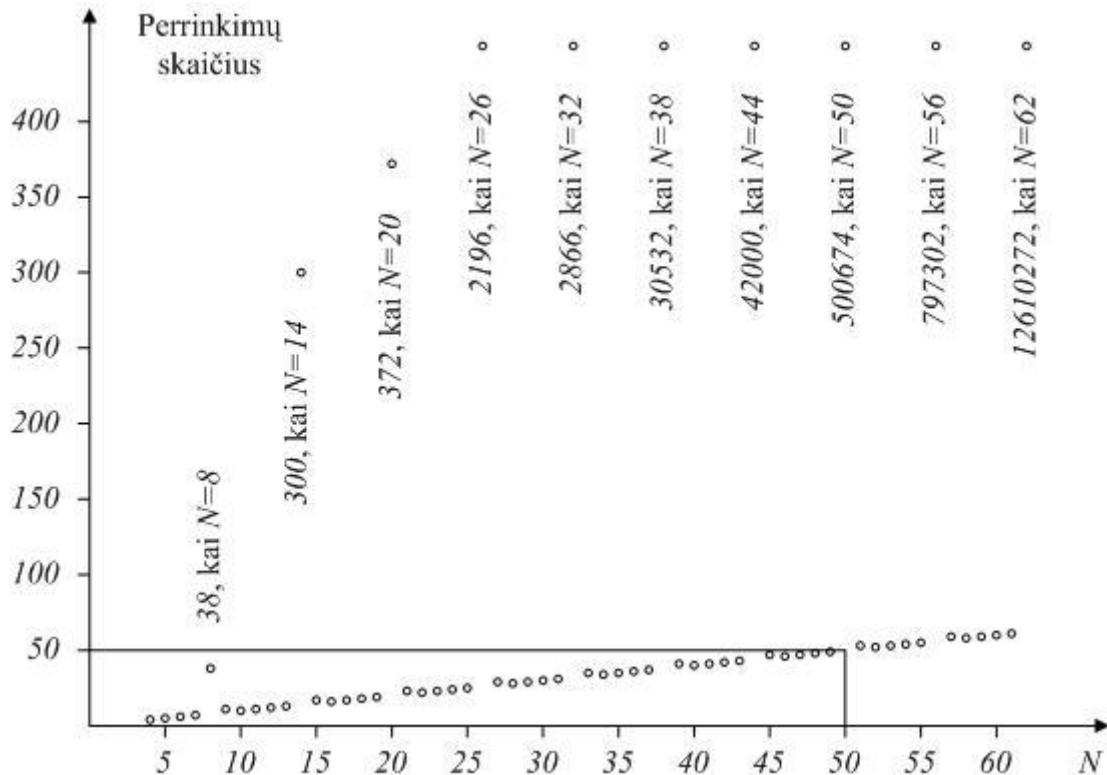
5.22 lentelė. Perrinkimų skaičius – briaunų skaičius paieškos medyje – trims algoritmams: perrinkimas be euristikos, „trumpiausios įstrižainės“ euristiką, „žirgo ėjimo“ euristika

Kaip galima pastebeti 5.22 lentelėje, euristikos užtikrina trumpesnę paiešką.

Toliau aptarsime perrinkimo su „žirgo ėjimo“ euristika teisiškumą, t. y. rezultato gavimą be grįžimų.

„Žirgo ėjimo“ euristika duoda rezultatą be grįžimų išspūdingai didelei lentų $N \times N$ aibei, būtent $N \in \{4, 5, 6, 7, 10, 11, 12, 13, 16, 17, 18, 19, 23\}$ ir kt.). Šioms lentoms paieškos medis turi lygiai N briaunų, kitais žodžiais, neturi atsišakojimų į aklavietes ir, tokiu būdu, neturi grįžimų.

5.23 pav. pateikiamas perrinkimų skaičiaus priklausomybės nuo N grafikas.



5.23 pav. Perrinkimų skaičiaus priklausomybė nuo lento dydžio N kai valdovės statomos „žirgo ėjimo“ euristika. Galima pastebėti šios priklausomybės teisiškumą labai didelei aibei N , t. y. $N \notin \{8, 14, 20, 26, 32, 38\}$ ir kt.)

Perrinkimų skaičius yra tiesinis, t.y. be grįžimų ne absoliučiai visiems N , bet su išimtimis $N \notin \{8, 14, 20, 26, 32, 38\}$ ir kt.).

Įrodyti hipotezę, „visiems N perrinkimų skaičius yra tiesinis“ nepavyks. Čia priminsime tokį sąmoni. Kaip klaidingai gali būti įrodinėjimas teiginys, kad „visi nelyginiai skaičiai yra pirminiai“. Vienetas yra pirminis pagal apibrėžimą; 3 yra pirminis; 5 yra pirminis; 7 yra pirminis. Toliau patikrinkime keletą kitų skaičių: pavyzdžiui, 11 yra pirminis; 13 yra pirminis. Taigi, galima daryti “išvadą”, kad teiginys teisingas.

Toliau pateikiamas pseudokodas programos, kuri išdėsto valdoves naudodamasi „žirgo ėjimo“ euristika.

```

program VALDOVES_EURISTIKA;
const   N      = 8;
          NM1   = 7; {N-1}
          N2M1 = 15; {2*N-1}
var
  X      : array [1 .. N] of integer;
  VERT   : array [1 .. N] of boolean;           {Vertikalės}
  KYL    : array [-NM1 .. NM1] of boolean; {Kylančios istrižainės}
  LEID   : array [1..N2M1] of boolean;        {Besileidžiančios istrižainės}
  YRA    : boolean;
  I, J   : integer;
  BANDSK : longint;

procedure TRY (I, J: integer; var YRA: boolean);
{Išėjimas I, J – eilutė ir stulpelis. Išėjimas YRA – ar pavyko sustatyti}
  var K: integer;
begin
  K := I;
  repeat
    BANDSK := BANDSK + 1;
    if VERT[I] and KYL[I - J] and LEID[I + J - 1] then
    begin {Vertikalė, kylanti ir besileidžianti istrižainės yra laisvos}
      X[J] := I;
      VERT[I] := false; KYL[I - J] := false; LEID[I + J - 1] := false;
      if j < N then
        begin
          if I + 2 > N then
            TRY(    1, J + 1, YRA)
          else
            TRY(I + 2, J + 1, YRA);
          if not YRA then
            begin {Nerastas kelias toliau}
              VERT[I] := true; KYL[I - J] := true;
              LEID[I + J - 1] := true; {Atlaivinamas langelis}
            end;
          end
          else YRA := true;
        end
        I := (I mod N) + 1;
      until YRA or (I = K);
end; {TRY}

procedure SOLVE;
begin
  if (N - 3) mod 6 = 0 then {N yra pavidalo  $6n + 3$ }
    TRY(4, 1, YRA)
  else
    TRY(2, 1, YRA)
end;

begin {Pagrindinė programa – main program}
  {1. Inicializuojami kintamieji}
  for J := 1 to N do VERT[J] := true;
  for J := -NM1 to NM1 do KYL[J] := true;
  for J := 1 to N2M1 do LEID[J] := true;
  YRA := false; BANDSK := 0;
  {2. Kviečiama procedūra}
  SOLVE;
  {3. Spausdinama lenta}
  if YRA then
    begin

```

```
for I := N downto 1 do
begin
  for J := 1 to N do
    if X[I] = J then write(1 : 3) else write(0 : 3);
  writeln;
end;
writeln('Bandymų skaičius: ', BANDSK);
end
else writeln('Sprendinys neegzistuoja');
end.
```

Pratimai

1. Nubraižykite paieškos medį valdovių išdėstymui 5×5 šachmatų lentoje „trumpiausios įstrižainės” euristika. Palyginkite su paieškos medžiu pilnam perrinkimui 5.13 pav.
2. Išdėstykite 8 rikius šachmatų lentoje taip, kad būtų kertamas kiekvienas langelis. Parašykite programą. Pasiūlykite euristiką. Pasiūlykite išdėstymo algoritmą be perrinkimo.
3. Patobulinkite valdovių išdėstymo euristiką „žirgo éjimas“.
4. Nubraižykite paieškos medį valdovių išdėstymui „žirgo éjimo“ euristika 21×21 šachmatų lentoje.

6. Patobulinta paieškos su grīžimais strategija. Procedūra BACKTRACK1

Procedūra BACKTRACK turi keletą esminių trūkumų. Pirmasis trūkumas yra tai, kad BACKTRACK gali „užsiciklini“ ir, tokiu būdu, nesustoti. Užsiciklinimo priežastis yra tai, kad BACKTRACK nesaugo nesėkminges būsenų.

Antras trūkumas pasireiškia tuo, kad paieška gali nueiti pernelyg gylyn. Tada BACKTRACK sukurs tiek daug duomenų bazės būsenų, kad jų pavaizdavimui neužteks kompiuterio atminties arba bus viršytas tam tikras laiko limitas. Todėl yra tikslina riboti rekursijos gylį.

Užsiciklinimas demonstruojamas kelio paieška labirinte kitame skyriuje 7.3 pav.

Siekiant išvengti minėtų trūkumų, procedūra BACKTRACK yra modifikuojama, įvedant aplankytų būsenų sąrašą bei rekursijos gylį, ir pavadinama BACKTRACK1. Ji žemiau pateikiama iš [Nilsson 1985] 2.1 poskyrio:

```
procedure BACKTRACK1 (DATALIST) returns PRODLIST;
{DATALIST - globalios duomenų bazės būsenų sąrašas}
{PRODLIST - procedūros rezultatas: produkcijų sąrašas}
{ 1} DATA := FIRST(DATALIST);
{ 2} if MEMBER(DATA, TAIL(DATALIST)) then
    return FAIL;
{ 3} if TERM(DATA) then return NIL;
{ 4} if DEADEND(DATA) then return FAIL;
{ 5} if LENGTH(DATALIST) > BOUND then return FAIL;
{ 6} RULES := APPRULES(DATA);
{ 7} LOOP: if NULL(RULES) then return FAIL;
{ 8} R := FIRST(RULES);
{ 9} RULES := TAIL(RULES);
{10} RDATA := R(DATA);
{11} RDATALIST := CONS(RDATA, DATALIST);
{12} PATH := BACKTRACK1(RDATALIST);
{13} if PATH = FAIL then goto LOOP;
{14} return CONS(R, PATH);
end procedure;
```

1 žingsnis. Iš globalios duomenų bazės būsenų sąrašo DATALIST imama pirmoji, t. y. einamoji duomenų bazės būsena. Laiko prasme ji buvo pasiekta vėliausiai. Funkcija FIRST grąžina pirmajį sąrašo elementą. Proceso pradžioje sąrašas DATALIST yra sudarytas iš vienos būsenos – pradinės globalios duomenų bazės būsenos.

2 žingsnis. Tikrinama, ar einamoji duomenų bazės būsena DATA nebuvo pasiekta anksčiau. Predikatas MEMBER(elementas, sąrašas) grąžina reikšmę true, jeigu elementas priklauso sąrašui, ir false, jeigu neprieklauso.

3 žingsnis. Patikriname, ar dar nepasiekta terminalinė būsena. Jeigu predikato TERM reikšmė yra true, tai procedūra grąžina tuščią produkcijų sąrašą, žymimą NIL, ir sėkmingai baigiamasi.

4 žingsnis. Tikriname, ar einamoji duomenų bazės būsena gali pasitaikyti kelyje į tikslą. Predikato DEADEND reikšmė yra true, jeigu einamoji duomenų bazės būsena niekada negali būti sutinkama kelyje siekiant tikslą. Tokiu atveju, procedūra BACKTRACK1 baigiamasi ir grąžinamas nesėkmės požymis FAIL. Predikatu DEADEND yra programuojamos tokios

žinios, kai programuotojas iš anksto žino, kad tam tikra globalios duomenų bazės būseną niekada nepasitaiko kelyje į terminalinę būseną.

5 žingsnis. Tikrinama, ar rekursijos gylis, t. y. sąrašo DATALIST elementų skaičius neviršija iš anksto apibrėžto ribinio rekursijos gylį BOUND. Jeigu ribinis rekursijos gylis pasiekta, tai procedūra BACKTRACK1 grąžina nesékmės požymį FAIL ir sustoja. Pavyzdžiu, LENGTH(<a,b,c>)=3, LENGTH(<a>)=1, LENGTH(<>)=0, LENGTH(<<a,b>,<c>,<d,e,f>,<g,h>>)=4. BOUND koncepcija – atsižvelgti į vartotojo nurodytus resursus. Tai priemonė prieš pernelyg gilią paiešką.

6 žingsnis. Funkcija APPRULES sudaro aibę tokų produkcijų, kurios gali būti taikomos einamajai globalios duomenų bazės būsenai DATA..

7 žingsnis. Jeigu taikytinų einamosios globalios duomenų bazės būsenai produkcijų aibė yra tuščia, t. y. nėra nė vienos produkcijos, kurią būtų galima pritaikyti, tai procedūra grąžina nesékmės požymį FAIL ir yra baigama.

8 žingsnis. Iš produkcijų aibės RULES yra paimama pirmoji produkcija.

9 žingsnis. Iš produkcijų aibės RULES yra pašalinama pirmoji produkcija.

10 žingsnis. Produkcija R globalią duomenų bazę iš būsenos DATA perveda į naujā būseną RDATA.

11 žingsnis. Prie globalios duomenų bazės būsenų sąrašo DATALIST prijungiamas naujai sugeneruota būsena RDATA. Gautoji aibė pavadinama RDATALIST.

12 žingsnis. Procedūra BACKTRACK1 yra kviečiama rekursyviai. Faktinis parametras yra 11 žingsnyje suformuotas būsenų sąrašas RDATALIST.

13 žingsnis. Jeigu 12 žingsnyje yra grąžintas nesékmės požymis FAIL, tai reiškia, kad 8 žingsnyje paimta produkcija R neveda į sėkmę. Tenka grižti į 7 žingsnį. Imti kitą produkciją.

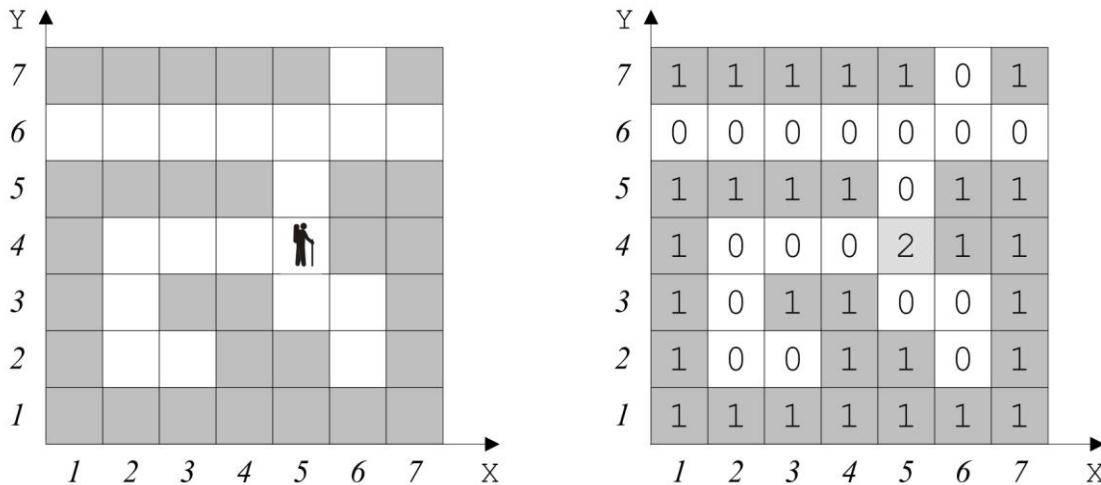
14 žingsnis. Kaip procedūros rezultatas yra grąžinamas produkcijų sąrašas. Funkcija CONS prijungia elementą R prie sąrašo PATH.

Procedūra BACKTRACK1 atlieka paiešką į gylį. Pavyzdžius panagrinėsime kituose skyriuose.

7. Uždavinys apie labirintą. Kelio paieška į gylį

Panaudosime algoritmą BACKTRACK1 kelio radimui iš labirinto. Agentas stovi labirinte. Jo tikslas yra pasiekti išėjimą iš labirinto. Išėjimų gali būti keletas.

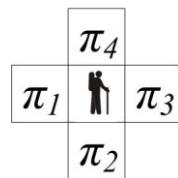
Paprasčiausiu atveju labirintas gali būti suprantamas kaip dvimatė lenta, kurios langeliai arba užpildyti, arba laisvi. Pavyzdys yra 7.1 pav.



7.1 pav. Labirinto pavyzdys ir pavaizdavimas dvimačiu sveikujų skaičių masyvu $LAB[*,*]$. Agentas stovi langelyje $X=5$, $Y=4$. Šiame labirinte yra trys išėjimai: $LAB[1,6]$, $LAB[7,6]$ ir $LAB[6,7]$. Užpildytas lanelis, t. y. „siena“, vaizduojamas sveikuojų skaičiumi 1, laisvas lanelis 0, o lanelis kuriame pradinėje būsenoje stovi agentas 2, t. y. $LAB[5,4] = 2$

Labirintą kompiuterio atmintyje pavaizduokime dvimačiu sveikujų skaičių masyvu. Užpildytas lanelis, t. y. „siena“, vaizduojamas sveikuojų skaičiumi 1, o laisvas lanelis – 0. Tegu pradinėje būsenoje agentas stovi laisvai pasirinktame langelyje. Šis masyvo elementas užpildomas reikšme 2.

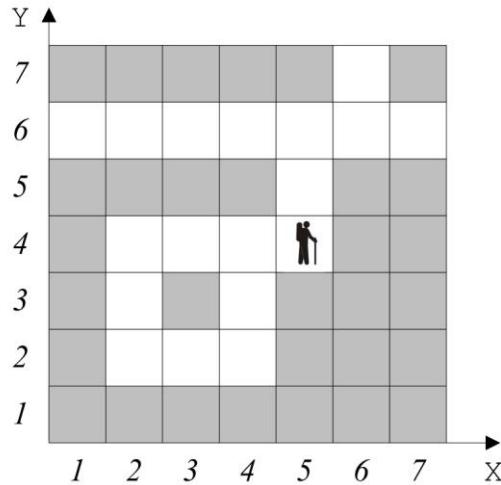
Agentas gali judėti tik laisvais langeliais. Susitarkime, kad iš einamojo lanelio keturios produkcijos $\{\pi_1, \pi_2, \pi_3, \pi_4\}$ yra surūšiuotos šia tvarka: π_1 – „i vakarus“, π_2 – „i pietus“, π_3 – „i rytus“ ir π_4 – „i šiaurę“ (žr. 7.2 pav.).



7.2 pav. Keturi agento ejimai – tai keturios produkcijos: „i vakarus“, „i pietus“, „i rytus“ ir „i šiaurę“ (žr. 7.2 pav.).

Laikoma, kad kelias išeiti iš labirinto rastas, kai agentas pasiekia bet kurį laisvą langelį labirinto krašte.

Procedūra BACKTRACK1 neužsiciklina bet kokiame labirinte, o BACKTRACK gali ir užsiciklinti. Kai labirinte yra „salų“, BACKTRACK gali užsiciklinti (bet gali ir neužsiciklinti – tai priklauso nuo produkčijų perrinkimo tvarkos). Pavyzdžiui, BACKTRACK užsiciklina labirinte 7.3 pav. Norint neužsiciklinti reikia įsiminti ankstesnes būsenas. Galima sakyti, kad reikia intelekto, kur intelektas suprantamas kaip gebėjimas įsiminti nueitą kelią.



7.3 pav. Labirinto, kuriame BACKTRACK užsiciklina, pavyzdys. Agentas stovi langelyje $X=5$, $Y=4$. Vadovaudamasis procedūra BACKTRACK, agentas keliauja aplink salą amžinu ciklu

Toliau pateikiama programa kelio iš labirinto paieškai valdymo su grįžimais algoritmu BACKTRACK1, t. y. į gylį. Ji spausdina kelią labirinte iš pradinio lavelio iki krašto.

```

program LABIRINTAS; {Kelio paieška algoritmu BACKTRACK1, t. y. i gyli}
const M = 7; N = 7; {Labirinto matmenys}
var LAB : array[1..M, 1..N] of integer; {Labirintas}
CX, CY : array[1..4] of integer; {4 produkcijos - X ir Y poslinkiai}
L, {Éjimo eilės numeris pradedant nuo 2. Žymi aplankytą langeli}
X, Y, {Pradiné agento padétis}
I, J, {Ciklo kintamieji}
BANDSK : integer; {Bandymų skaičius. Dél efektyvumo palyginimo}
YRA : boolean; {Ar kelias egzistuoja}

procedure EITI(X, Y : integer; var YRA : boolean);
var K, {Einamosios produkcijos numeris}
U, V : integer; {Nauja padétis pritaikius produkcijs}
begin {EITI}
{K1} if (X = 1) or (X = M) or (Y = 1) or (Y = N)
then YRA := true {TERM(DATA) = true, jeigu kraštas}
else
begin K := 0;
{K2} repeat K := K + 1; {Imama kita produkcijs. Ciklas per produkcijs}
{K3} U := X + CX[K]; V := Y + CY[K]; {Nauja agento padétis}
{K4} if LAB[U, V] = 0 {Langelis tuščias}
then
begin BANDSK := BANDSK+1; {Tik įdomumui. Variantu palyginimui}
{K5} L := L + 1; LAB[U,V] := L; {Langelis pažymimas}
{K6} EITI(U, V, YRA); {Rekursyvus kvietimas keliauti toliau}
if not YRA {Iš čia nebeisineame}
{K7} then begin
{K8} LAB[U,V] := -1; {0 BACKTRACK atveju}
L := L - 1;
end;
end;
until YRA or (K = 4);
end;
end; {EITI}

begin {Pagrindiné programa}
{1. Ivedamas labirintas}
for J := N downto 1 do
begin
for I := 1 to M do read(LAB[I,J]);
readln;
end;
{2. Ivedama ir pažymima pradiné agento padétis. Tai pradiné GDB būsena}
read(X, Y); L := 2; LAB[X,Y] := L;
{3. Užpildomas produkcijs}
CX[1] := -1; CY[1] := 0; {Kairén - i vakarus. 4 }
CX[2] := 0; CY[2] := -1; {Žemyn - i pietus. 1 * 3 }
CX[3] := 1; CY[3] := 0; {Dešinén - i rytus. 2 }
CX[4] := 0; CY[4] := 1; {Viršun - i šiaurę. }
{4. Priskiriamos pradinés reikšmés kintamiesiems}
YRA := false; BANDSK := 0;
{5. Kviečiama procedūra keliui rasti}
EITI(X, Y, YRA);
if YRA
then writeln('Kelias egzistuoja'); {Belieka atspausdinti kelią}
else writeln('Kelias neegzistuoja'); {Kelias iš labirinto neegzistuoja}
end.

```

Trys grįžimo variantai:

V1) LAB[U,V] := -1. Tai

BACKTRACK_SU_ATMINTIMI.

V2) LAB[U,V] := 0 ir yra LAB[U,V]:=L aukščiau. Tai BACKTRACK1. Aplink salą keliaujama antrą kartą, bet neužsiciklinama.

V3) LAB[U,V] := 0 ir nėra LAB[U,V]:=L. Tai klasikinis BACKTRACK. Programa užsiciklina.

Toliau pateikiami išplėsti šios programos komentarai.

K1. BACKTRACK1 3 žingsnis. TERM(DATA). Tirkinama, ar agentas ant krašto.

K2. BACKTRACK1 8-9 žingsniai. Kintamasis K nurodo einamosios produkcijos numerį.

K3. BACKTRACK1 10 žingsnis. Globali duomenų bazė pervedama į naują būseną. Agentas pereina į kitą langeli.

K4. Tirkinama, ar langelis laisvas. Šitaip programuojama, ar K2 paimta produkcija priklauso aibei APPRULES(DATA), kuri yra BACKTRACK1 6 žingsnyje.

K5. BACKTRACK1 11 žingsnis. Pažymima kad šis agento langelis jau išnagrinėtas. BACKTRACK1 būsenas kaupia sąraše DATALIST. Taip išvengiama ir pakartotino ėjimo aplink salą iš kitos pusės ir užsicklinimo.

K6. BACKTRACK1 12 žingsnis. Procedūra kviečiama rekursyviai.

K7. Kadangi terminalinė būsena nėra pasiekta, tai tenka grįžti atgal.

K8. Trys grįžimo variantai:

V1) LAB[U,V] := -1. Užprogramuojame BACKTRACK1.

V2) LAB[U,V] := 0 ir yra LAB[U,V]:=L aukšciau. Tai BACKTRACK_SU_STEKU. Aplink salą keliaujama antrą kartą, bet neužsicklinama.

V3) LAB[U,V] := 0 ir nėra LAB[U,V]:=L. Tai klasikinis BACKTRACK. Programa užsicklina.

Šie trys grįžimo variantai V1, V2 ir V3 atspindi tris subtilius skirtumus valdymo su grįžimais strategijoje. Trys skirtingi paieškos medžiai pateikiami 7.4 pav.

7.1. Kelio paieškos labirinte algoritmai BACKTRACK ir BACKTRACK1 variantai

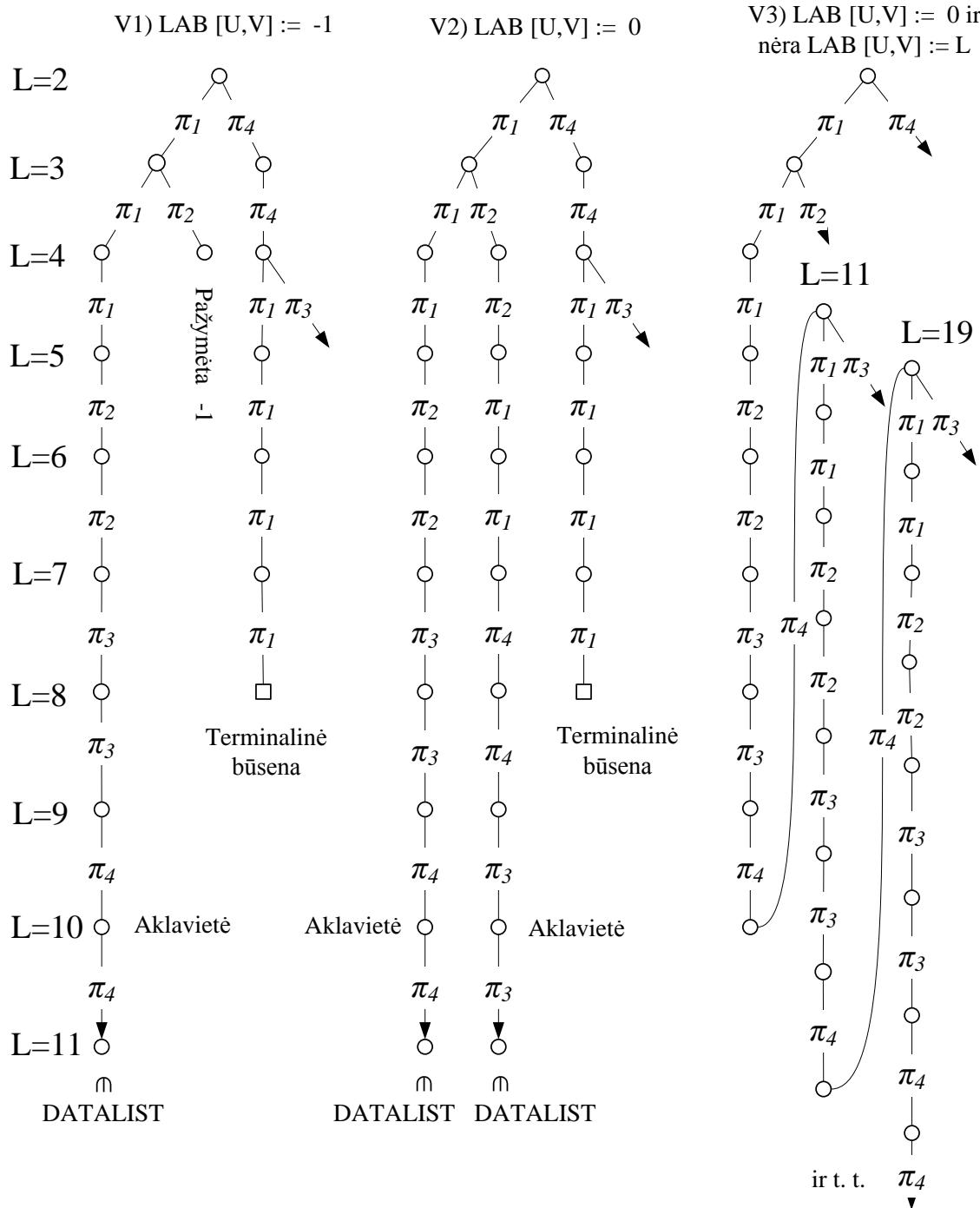
Valdymo su grįžimais šeimos BACKTRACK trys algoritmai pagal „gerumą“ sutvarkomi:

BACKTRACK < BACKTRACK1 < BACKTRACK_SU_ATMINTIMI

Čia „ \leq “ žymi dalinę tvarką. Už algoritmą BACKTRACK geresnis yra BACKTRACK1, o už pastarajį geresnis BACKTRACK_SU_ATMINTIMI. Algoritmas BACKTRACK1 dar įvardinamas sinonimu BACKTRACK_SU_STEKU. Sąvoka „geresnis“ yra suprantama kaip paieškos medžio briaunų skaičius, vidutiniu atveju. Toliau pavyzdžiais iliustruojama, kad atskirais kraštutiniais atvejais ši tvarka gali skirtis.

Šiuos variantus iliustravome kelio paieškos labirinte uždaviniu (žr. 7.3 pav.). BACKTRACK1 kaupia būsenas sąraše DATALIST. O BACKTRACK neturi jokio sąrašo būsenoms kaupti. Abiejų procedūrų rekursyvumas salygoja steko formavimą kompiuterio atmintyje. Sąrašas DATALIST atlieka dar vieno steko vaidmenį, nes Jame kaupiamos tik einamosios rekursijos šakos būsenos. BACKTRACK_SU_ATMINTIMI atmintyje pasižymėdamas kaupia ne tik einamosios šakos, o visas aplankytas GDB būsenas.

BACKTRACK atveju paieškos medis yra neišreikštinis. O BACKTRACK1 kaupia GDB būsenas sąraše DATALIST. Kadangi algoritmas žino, iš kurios senos būsenos jis pereina į naują, tai paieškos medis gali būti pavaizduotas išreikštiniu pavidalu.



7.4 pav. Trys paieškos medžių variantai rasti kelią labirinte iš 7.3 pav. V1 variantas) $\text{LAB}[U,V] := -1$. Šitaip užprogramuojamas $\text{BACKTRACK_SU_ATMINTIMI}$. V2 variantas) $\text{LAB}[U,V] := 0$ ir yra $\text{LAB}[U,V] := L$. Užprogramuojamas BACKTRACK1 , t.y. $\text{BACKTRACK_SU_STEKU}$. Aplink salą keliaujama antrą kartą, bet neužsiciklinama. V3 variantas) $\text{LAB}[U,V] := 0$ ir nėra $\text{LAB}[U,V] := L$. Šitaip užprogramuojamas klasikinis BACKTRACK . Programa užsiciklina.

7.2. Kai kurios produkcijų algebrinės savybės

Toliau pateikiamos dvi dirbtinio intelekto produkcijų sistemos algebrinės savybės:

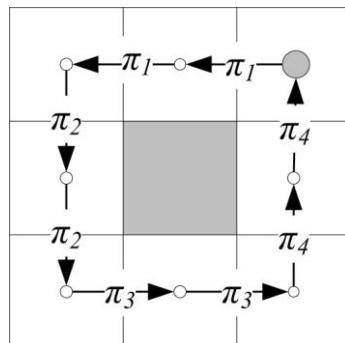
1 savybė. Produkcija gali būti suprantama kaip atvaizdavimas. Šio atvaizdavimo *apibrėžimo sritis* ir *kitimo sritis* yra ta pati aibė, būtent, globalios duomenų bazės būsenų aibė. Bet kuri produkcija π perveda GDB iš vienos būsenos į kitą: $\pi(\text{DATA}) = \text{RDATA}$.

2 savybė. Produkcijos π *atvirkštinė* produkcija yra žymima π^I . Tokiu būdu, pagal apibrėžimą $\pi^I \circ \pi = I$. Čia „ \circ “ žymi atvaizdavimų *superpoziciją*, o I žymi atvaizdavimą, kuris yra vadinamas *vienetiniu atvaizdavimu*. Pastarasis pasižymi tuo, kad $I(\text{DATA}) = \text{DATA}$, kiekvienai GDB būsenai DATA. Vienetinis atvaizdavimas bet kokią būseną atvaizduoja į save pačią, t. y. nekeičia šios būsenos. Pavyzdžiui, labirinto uždavinyje galioja savybė $\pi_I(\pi_3(\text{DATA})) = \text{DATA}$ (žr. 7.3 pav.). Taigi $\pi_3 = \pi_I^{-1}$ ir $\pi_4 = \pi_2^{-1}$.

Einant aplink salą, yra grįžtama į tą pačią būseną:

$$\pi_4(\pi_4(\pi_3(\pi_3(\pi_2(\pi_2(\pi_I(\text{DATA}))))))) = \text{DATA}$$

Šis ciklas yra pateiktas 7.4 pav. V3 variante.

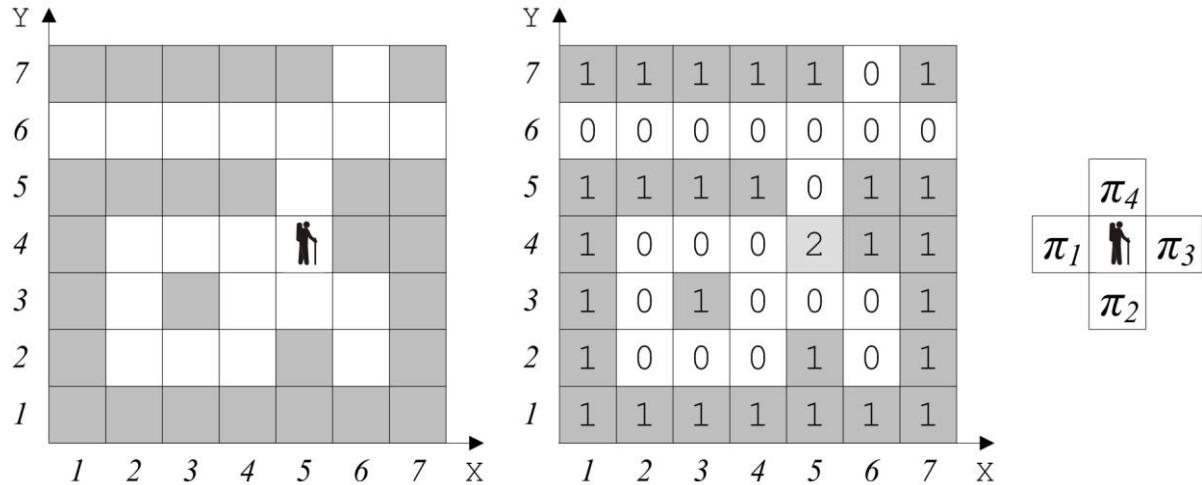


7.5 pav. Ciklo aplink salą pavyzdys yra produkcijų seka $\langle \pi_1, \pi_1, \pi_2, \pi_2, \pi_3, \pi_3, \pi_4, \pi_4 \rangle$

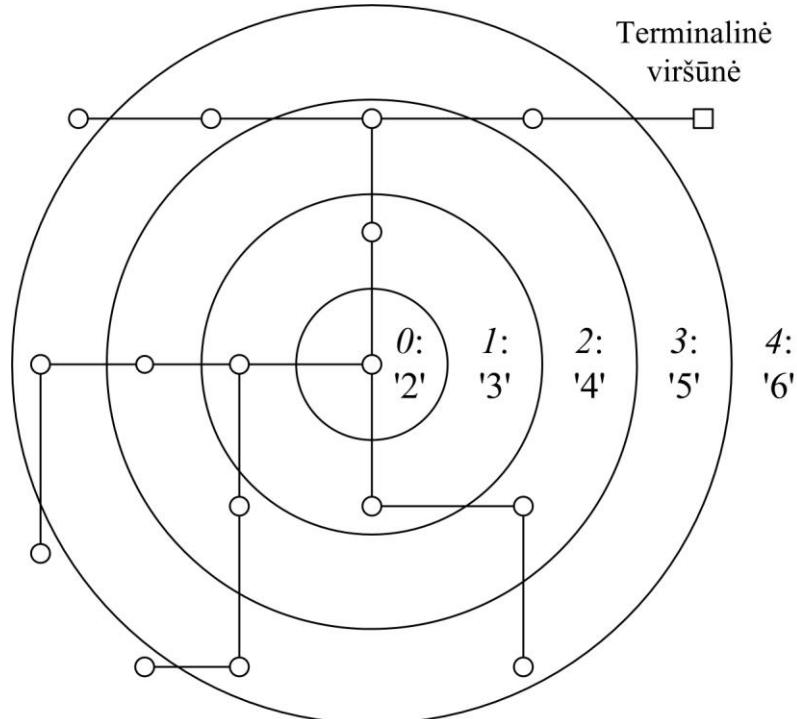
8. Kelio radimas labirinte paieškos į plotį būdu – bangos metodas

Dirbtinio intelekto produkcijų sistemoje be iki šiol aptarto paieškos būdo „i gylį“ yra svarbus paieškos „i plotį“ būdas. Jis dar vadinamas **bangos metodu**. Ši paieškos būdą iliustruosime kelio paieška labirinte. Šis algoritmas pasižymi tuo, kad jo rezultatas yra trumpiausias keliai iš visų galimų.

Susitarkime, kad iš einamojo langelio keturios produkcijos $\{\pi_1, \pi_2, \pi_3, \pi_4\}$ yra surūšiuotos šia tvarka: π_1 – „i vakarus“, π_2 – „i pietus“, π_3 – „i rytus“ ir π_4 – „i šiaurę“ (žr. 8.1 pav.).

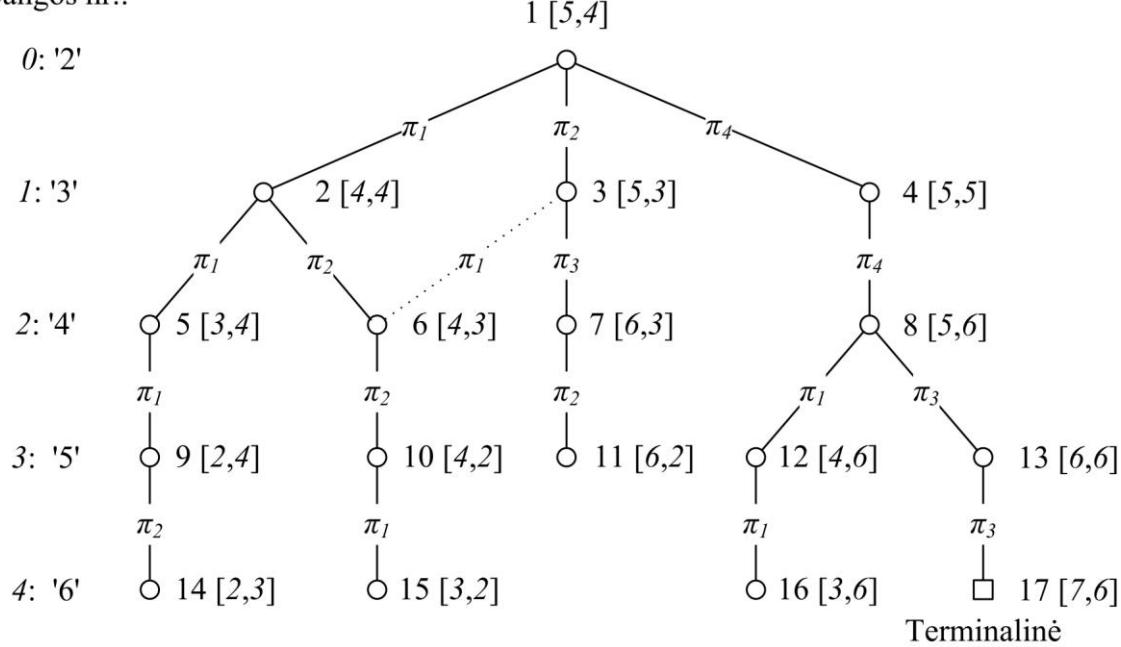


8.1 pav. Labirintas ir jo pavaizdavimas masyvu LAB [X, Y]. Agentas stovi langelyje X=5, Y=4. Keturi agento ėjimai $\{\pi_1, \pi_2, \pi_3, \pi_4\}$



8.2 pav. Paieškos medis. Atskira banga – tai aibė viršūnių, esančių tame pačiame paieškos medžio gylyje

Bangos nr.:

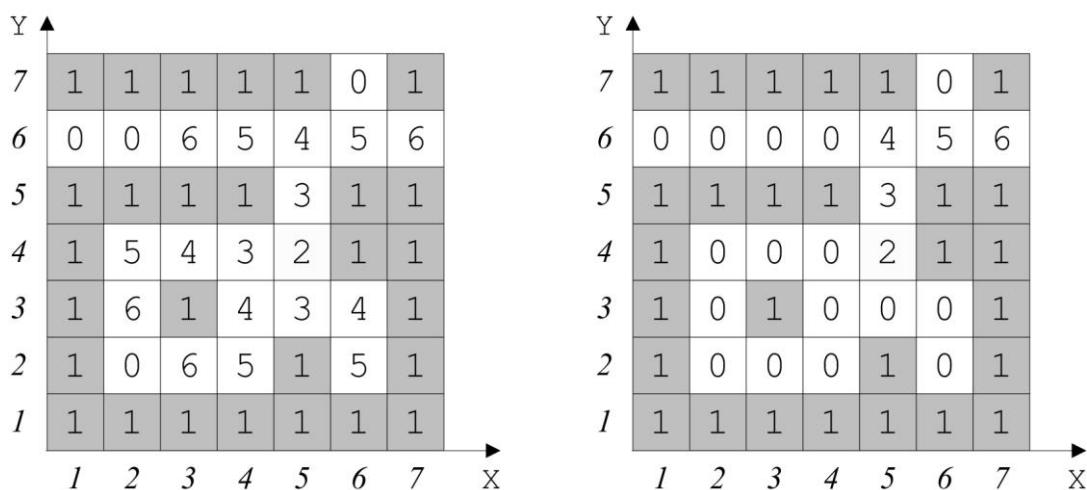


8.3 pav. Paieškos medis „pakratytas“ taip, kad vienos bangos viršūnės atsidurtų tame pačiame gylyje

Norėdami rasti kelią iš labirinto atliekame šiuos veiksmus:

1) Skleidžiame bangą. Gauname visų atidarytų langelių vaikus. Vaikais čia vadinsime tuos gretimus labirinto langelius (grafo terminais tai viršūnės), į kuriuos galima patekti iš einamojo langelio, pritaikius tinkamas produkcijas. Visus gautus vaikus pažymime kaip „atidarytus“. Langelius, iš kurių jie buvo pasiekti, pažymime kaip „uždarytus“.

2) Ar pasiekta terminalinė viršūnė? Patikriname, ar yra toks atidarytas lanelis, kuris tenkina terminalinės būsenos sąlygas. Jei taip, tai keliais rastas. Belieka surinkti jį grįžtant atgal. Jei ne, tai einame į 1 žingsnį.



8.4 pav. a) Globalios duomenų bazės būsena, įvykdžius paieškos į plotį algoritmą. b) Masyvo LAB būsena, surinkus kelią atgal

Atidaromi langeliai yra pažymėti iteracijos, t. y. bangos numeriu plius 2. Tokiu būdu, penkis kartus pakartojė minėtus veiksmus, pasiekiame būseną, pavaizduotą 8.6 a pav. Matome, kad vienas iš langelių, pažymėtu 6, (būtent langelis X=7, Y=6) tenkina terminalinės viršūnės sąlygą. Tokiu būdu išėjimas iš labirinto pasiekta. Surinkę rastą kelią atgal, t. y. keliaudami labirinto langeliais nuo rasto išėjimo atgal jiems priskirtų numerių mažėjimo tvarka, gauname masyvo LAB būseną, parodytą 8.4 b pav.

Toliau yra pateikiama programa, skirta kelio iš labirinto paieškai į plotį.

```

program LABIRINTAS_I_PLOTI (input, output);
const
    M = 7; N = 7; {Labirinto matmenys}
    MN = 49; {Langelių skaičius M*N}
var
    LAB, LABKOP : array[1..M, 1..N] of integer; {Labirintas ir jo kopija}
    CX, CY : array[1..4] of integer; {4 produkcijos}
    FX, FY : array[1..MN] of integer; {Masyvai fronto viršūnėms}
    UZD, {Skaitliukas uždaromai viršūnei}
    NAUJA, {Skaitliukas atidaromai viršūnei}
    K, {Produkcijos numeris}
    X, Y, {Pradinė keliautojo padėtis}
    U, V, I, J : integer;
    YRA : boolean;

procedure ATGAL(U, V : integer); {Surenkamas kelias atgal}
{Ieities parametrai: U, V - išėjimo iš labirinto langelio koordinatės, }
{ir globalus masyvas LABKOP. Išeities masyvas (rezultatas) LAB           }
    var K, UU, VV : integer;
begin {ATGAL}
    LAB[U,V] := LABKOP[U,V]; {Pažymimas išėjimo langelis}
    K := 0;
    repeat {Perrenkamos 4 producijos. Ieškomas toks langelis LABKOP[UU,VV], }
        {kurio reikšmė yra vienetu mažesnė už LABKOP[U,V]          }
        K := K + 1; UU := U + CX[K]; VV := V + CY[K];
        if (1 <= UU) and (UU <= M) and (1 <= VV) and (VV <= N)
        then {Neišeiname už labirinto ribų}
            if LABKOP[UU,VV] = LABKOP[U,V] - 1
            then
                begin
                    LAB[UU,VV] := LABKOP[UU,VV]; {Pažymime langelį masyve LAB}
                    U := UU; V := VV; K := 0; {Sukeičiame kintamuosius}
                end;
        until LABKOP[U, V] = 2;
end; {ATGAL}

begin {Pagrindinė programa - main program}
    { 1) Įvedamas labirintas}
    for J := N downto 1 do
    begin
        for I := 1 to M do
        begin read(LAB[I,J]);
            LABKOP[I,J]:= LAB[I,J];
        end
        readln;
    end;
    { 2) Įvedama pradinė keliautojo padėtis}
    read(X,Y); LABKOP[X,Y]:=2;

    { 3) Užpildomas 4 produkcijos}
    CX[1] := -1; CY[1] := 0; {Kairėn - į vakarus. 2   }

```

```

CX[2] := 0; CY[2] := -1; {Žemyn - į pietus. 1 * 3 }
CX[3] := 1; CY[3] := 0; {Dešinėn - į rytus. 4 }
CX[4] := 0; CY[4] := 1; {Viršun - į šiaurę. }

{ 4) Priskiriamos pradinės reikšmės kintamiesiems}
FX[1] := X; FY[1] := Y; UZD := 1; NAUJA := 1; YRA := false;

{ 5) Bangos algoritmas}
if (X = 1) or (X = M) or (Y = 1) or (Y = N)
then {Jeigu ant krašto (terminalinėje būsenoje), tai baigti darbą}
    begin YRA := true; U := X; V := Y;
    end;
if (X > 1) and (X < M) and (Y > 1) and (Y < N)
then
repeat {Ciklas per viršunes}
    X := FX[UZD]; Y := FY[UZD]; {Uždaromos viršunės koordinatės}
    K := 0;
    repeat {Ciklas per 4 produkcijas}
        K := K + 1; U := X + CX[K]; V := Y + CY[K];
        if LABKOP[U, V] = 0 {Jeigu kelias laisvas}
        then begin
            LABKOP[U,V] := LABKOP[X,Y] + 1; {Naujos bangos numeris}
            if (U = 1) or (U = M) or (V = 1) or (V = N) {kraštas}
            then YRA := true; {Sékmė. Čia galima kviesti ATGAL(U,V)}
            else begin {Nauja viršunė talpinama į fronto pabaiga}
                NAUJA := NAUJA + 1; FX[NAUJA] := U; FY[NAUJA] := V;
            end;
        end;
        until (K = 4) or YRA; {Perrinktos 4 produkcijos arba išėjimas rastas}
        UZD := UZD + 1; {Bus uždaroma (skleidžiama) kita viršunė}
    until (UZD > NAUJA) or YRA;

{ 6) Spausdinamas rezultatas}
if YRA
then begin
    writeln('Kelias iš labirinto egzistuoja');
    ATGAL(U,V); {Kelias surenkamas.}
    { Čia turi būti kviečiama procedūra keliui spausdinti}
    end
else writeln('Kelias iš labirinto neegzistuoja');
end.

```

Pasiekus terminalinę būseną paiešką nutraukiame. Atsakymas (planas) – tai produkcių sąrašas $\langle \pi_4, \pi_4, \pi_3, \pi_3 \rangle$.

Kelią labirinte (grafe) galima užrašyti ne tik produkcių sąrašu, bet ir langeliu (viršunių) sąrašu $\langle [5,4], [5,5], [5,6], [6,6], [7,6] \rangle$. Tokiu būdu pastebime, kad kelio užrašymas yra dualus. Kelią grafe galima užrašyti ir briaunų (t.y. produkcių) sąrašu, pvz., $\langle [v_1, v_2], [v_2, v_3], [v_3, v_4], [v_4, v_5] \rangle$, ir viršunių sąrašu $\langle v_1, v_2, v_3, v_4, v_5 \rangle$. Žemėlapyje kelias paprastai nurodomas miestų sąrašu, pvz., $\langle \text{Druskininkai}, \text{Vilnius}, \text{Kaunas}, \text{Klaipėda} \rangle$. Tačiau kelią galima nurodyti ir autostradų numeriais $\langle A4, A1, A1 \rangle$

Aukščiau pateiktoje programoje į masyvus FX ir FY yra talpinamos atidaromų viršunių koordinatės atitinkamai x ir y . Šie masyvai vadinami *frontu*. 8.4 lentelėje pateikiama bangos algoritmo interpretacija labirinto uždaviniui.

<i>i</i>	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	
Banga	'2'	'3'	'3'	'3'	'4'	'4'	'4'	'4'	'5'	'5'	'5'	'5'	'5'	'6'	'6'	'6'	'6'	
FX	5	4	5	5	3	4	6	5	2	4	6	4	6	2	3	3	7	
FY	4	4	3	5	4	3	3	6	4	2	2	6	6	3	2	6	6	
	UZD := 1	UZD = 1	NAUJA := 3	UZD = 1	NAUJA := 4	NAUJA := 5	UZD = 2	NAUJA := 6	UZD = 3	NAUJA := 8	UZD = 4	NAUJA := 9	UZD = 5	NAUJA := 10	UZD = 6	NAUJA := 11	UZD = 7	NAUJA := 12
	UZD := 1	UZD = 1	NAUJA := 2															

8.4 lentelė. Masyvų elementai FX[i] ir FY[i] programos vykdymo metu.

Jeigu uždaromos viršūnės numeris UZD tampa didesnis negu atidarytų viršūnių skaičius NAUJA, tai išėjimo nėra.

9. Paieškos į plotį algoritmas grafe be kainų

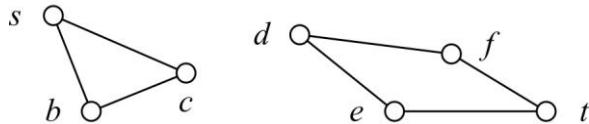
Pateiksime algoritmą rasti kelią grafe, kurio briaunos neturi kainos. Turime grafą. Turime pradinę viršūnę. Turime vieną terminalinę viršūnę. Paieškos į plotį algoritmas randa trumpiausią kelią. Tai kelias, turintis mažiausiai briaunų. Algoritmas pateikiamas vadovaujantis E. Hunt knygos vertimo į rusų kalbą [Hunt 1978] 10.1.1 poskyriu, p. 264.

Paieškos į plotį algoritmas grafe be kainų

Algoritmo iėjimas: 1) grafas be kainų, 2) pradinė viršūnė, 3) terminalinė viršūnė (gali būti ir keletas). Algoritmo išėjimas – trumpiausias kelias iš pradinės viršūnės į terminalinę.

1. Patalpinti pradinę viršūnę į sąrašą ATIDARYTA.

2. Jeigu sąrašas ATIDARYTA tuščias, tai kelias neegzistuoja. Baigt darbą. Tokia situacija yra gaunama, kai grafas yra *nesusijęs*, t. y. tame egzistuoja nepersikertantys viršūnių poaibiai tokie, kad pradinė viršūnė yra viename poaibyje, o terminalinė kitame, ir šių poaibių viršūnių nejungia jokia briauna (pavyzdys 9.1 pav.).



9.1 pav. Nesusijusio grafo pavyzdys. Kelias iš s į t neegzistuoja

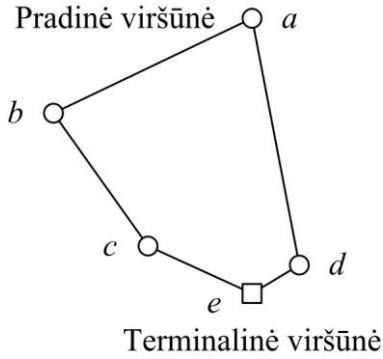
3. Uždaryti **pirmąją** viršūnę n iš sąrašo ATIDARYTA, t. y. perkelti iš sąrašo ATIDARYTA į sąrašą UŽDARYTA. Jeigu viršūnė n yra terminalinė (tikslo viršūnė), tai kelias rastas. Surinkti kelią atgal. Baigt darbą.

4. Paimti viršūnės n incidentinių viršūnių (kitais žodžiais – gretimų) aibę $S(n)$. Į sąrašo ATIDARYTA **pabaigą** patalpinti visus tokius vaikus $S(n)$, kurių nėra nei sąraše ATIDARYTA, nei sąraše UŽDARYTA. Formaliai, ATIDARYTA := CONS(ATIDARYTA, $S(n)/(ATIDARYTA \cup UŽDARYTA)$).

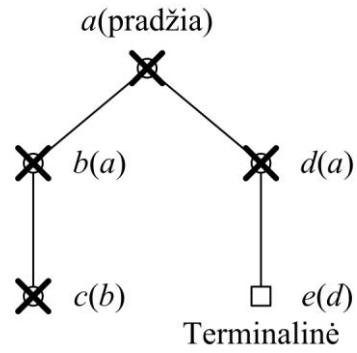
5. Pereiti prie 2 žingsnio.

Algoritmas veikia ir tuo atveju, kai terminalinių viršūnių yra keletas. Terminalinės viršūnės gali būti charakterizuojamos tam tikru predikatu TERM(viršūnė). Tokiu atveju algoritmas tikrina, ar uždaroma viršūnė tenkina ši predikatą.

Algoritmą iliustruojame grafu, pateiktu 9.2 pav. Ieškomas kelias iš pradinės viršūnės a į terminalinę viršūnę e . Paieškos medis parodytas 9.3 pav.



9.2 pav. Grafas, kurio briaunos be kainų.
Pradinė viršūnė a , terminalinė e



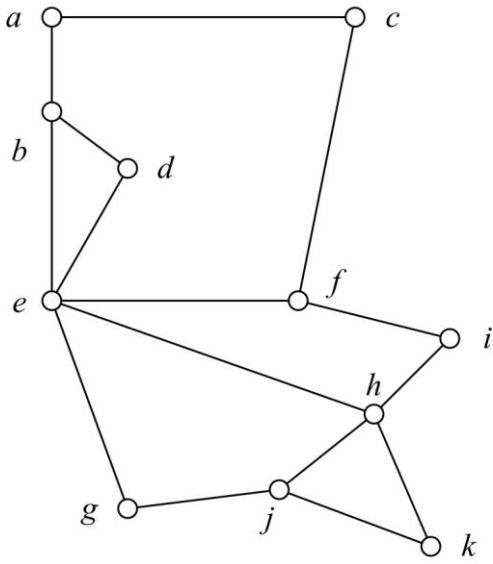
9.3 pav. Paieškos medis keliui iš a į e grafe, parodytame 9.2 pav.

Sąrašų ATIDARYTA ir UŽDARYTA būsenos algoritmo žingsniuose yra 9.4 lentelėje:

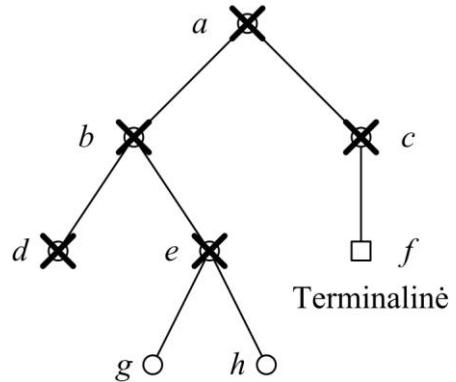
Nr.	ATIDARYTA	UŽDARYTA	Komentaras
1	$a(pradžia)$	\emptyset	Pradinė būsena
2	$b(a), d(a)$	$a(pradžia)$	$S(a) = \{b, d\}$. Uždaroma a ir atidaromos b bei d
3	$d(a), c(b)$	$a(pradžia), b(a)$	$S(b) = \{a, c\}$, bet $a \in$ UŽDARYTA
4	$c(b), e(d)$	$a(pradžia), b(a), d(a)$	$S(d) = \{a, e\}$, bet $a \in$ UŽDARYTA
5	$e(d)$	$a(pradžia), b(a), d(a), c(b)$	$S(c) = \{b, e\}$, bet $b \in$ UŽDARYTA, o $e \in$ ATIDARYTA. Todėl jų talpinti nebereikia
6	\emptyset	$a(pradžia), b(a), d(a), c(b), e(d)$	Uždaroma terminalinė e . Jos vaikai neanalizuojami

9.4 lentelė. Sąrašų ATIDARYTA ir UŽDARYTA būsenos algoritmo žingsniuose, kai ieškomas kelias iš pradinės a į terminalinę e grafe 9.2 pav.

Toliau sudėtingesnis pavyzdys. Ieškomas kelias iš pradinės a į terminalinę f 9.5 pav. Paieškos medis parodytas 9.6 pav.



9.5 pav. Grafas, kurio briaunos neturi kainų. Pradinė viršūnė a , terminalinė – f



9.6 pav. Paieškos medis ieškant kelio iš a į f grafe 9.5 pav.

9.7 lentelėje pateikiamos sąrašų ATIDARYTA ir UŽDARYTA būsenos algoritmo žingsniuose, kai ieškomas kelias iš pradinės viršūnės a į terminalinę f grafe, kuris yra 9.5 pav.

Nr.	ATIDARYTA	UŽDARYTA	Komentaras
1	$a(\text{pradžia})$	\emptyset	
2	$b(a), c(a)$	$a(\text{pradžia})$	$S(a) = \{b, c\}$
3	$c(a), d(b), e(b)$	$a(\text{pradžia}), b(a)$	$S(b) = \{a, d, e\}$, bet $a \in \text{UŽDARYTA}$
4	$d(b), e(b), f(c)$	$a(\text{pradžia}), b(a), c(a)$	$S(c) = \{a, f\}$, bet $a \in \text{UŽDARYTA}$
5	$e(b), f(c)$	$a(\text{pradžia}), b(a), c(a), d(b)$	$S(d) = \{b, e\}$, bet $b \in \text{UŽDARYTA}$
6	$f(c), g(e), h(e)$	$a(\text{pradžia}), b(a), c(a), d(b), e(b)$	$S(e) = \{b, d, f, g, h\}$, bet $b, d \in \text{UŽDARYTA}$ ir $f \in \text{ATIDARYTA}$
7	$g(e), h(e)$	$a(\text{pradžia}), b(a), c(a), d(b), e(b), f(c)$	Uždaroma terminalinė f . Jos vaikų talpinti nebereikia

9.7 lentelė. Sąrašų ATIDARYTA ir UŽDARYTA algoritmo žingsniuose, kai ieškomas kelias iš a į f grafe 9.5 pav.

Terminalinė viršūnė f yra uždaroma 7 žingsnyje. Jos vaikai $S(f) = \{c, e, i\}$ į sąrašą ATIDARYTA nebetalpinami. Surinkus kelią atgal, gaunama viršūnių seka: $\langle a(\text{pradžia}), c(a), f(c) \rangle$. Kelio ilgis yra dvi briaunos. Kelią galima vaizduoti tiek viršūnių sąrašu $\langle a, c, f \rangle$, tiek briaunų sąrašu $\langle (a, c), (c, f) \rangle$. Iš vieno pavaizdavimo yra gaunamas kitas.

Paieškos medis yra pateikiamas 9.6 pav. Terminalinė viršūnė f yra antroje bangoje. Algoritmas pradedą atidarineti ir trečiąjį bangą. Tokiu būdu pastebime, kad aukščiau pateiktas klasikinis algoritmas nėra optimizuotas. Ši algoritmą galima patobulinti, kad jis sustotų, kai **atidaroma** terminalinė viršūnė. Tuo tarpu klasikinis algoritmas sustoja kai

uždaro terminalinę viršūnę. Patobulintas algoritmas sutaupytu vieną bangą. Algoritmų teorijoje sudėtingumas laikomas tokiu pačiu, nepriklausomai ar atidaroma N ar $N + 1$ banga.

Išnagrinėkime dar vieną pavyzdį. Ieškomas kelias nuo pradinės viršūnės a iki terminalinės viršūnės h grafe 9.8 pav.

Paieškos į plotį algoritmą galima apibūdinti šitaip. Iš pradžių „pakratome“ grafą lyg „paémę“ už pradinės viršūnės – lyg laikydami už „pakarpos“. Briaunos išsitempią lyg guminės, ir viršūnės nusistovi pagal bangas. Kiekviena banga yra atskirame lygmenyje. Tada einame per bangas: per pirmąją, antrają, trečiąją, ir taip toliau. Suradę terminalinę viršūnę, surenkame kelią į viršų iki pradinės viršūnės.

Grafo 9.8 pav. viršūnių aibė yra:

$$V = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o\}$$

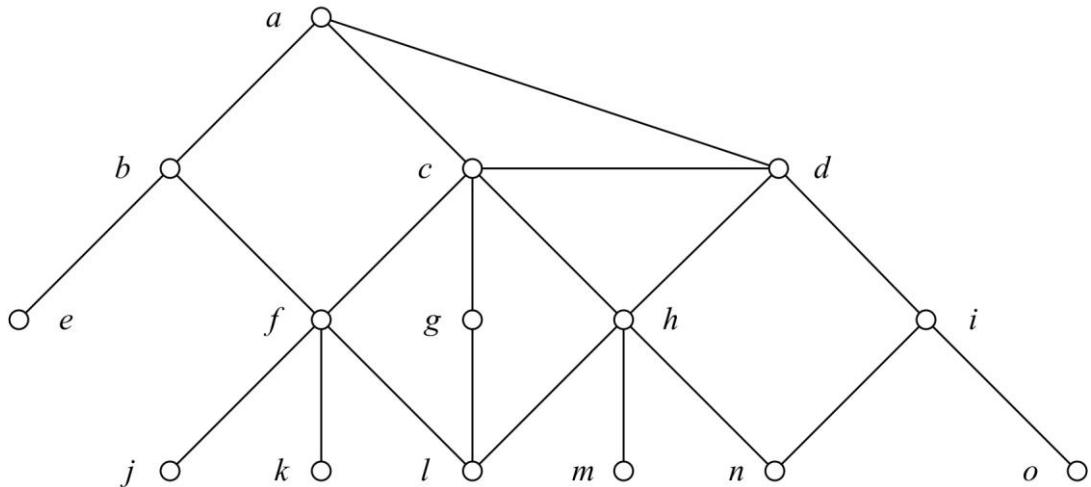
ir briaunų aibė:

$$E = \{(a,b), (a,c), (a,d), (b,e), (b,f), (c,f), (c,g), (d,h), (d,i), (f,j), (f,k), (g,l), (h,l), (h,m), (h,n), (i,n), (i,o)\}$$

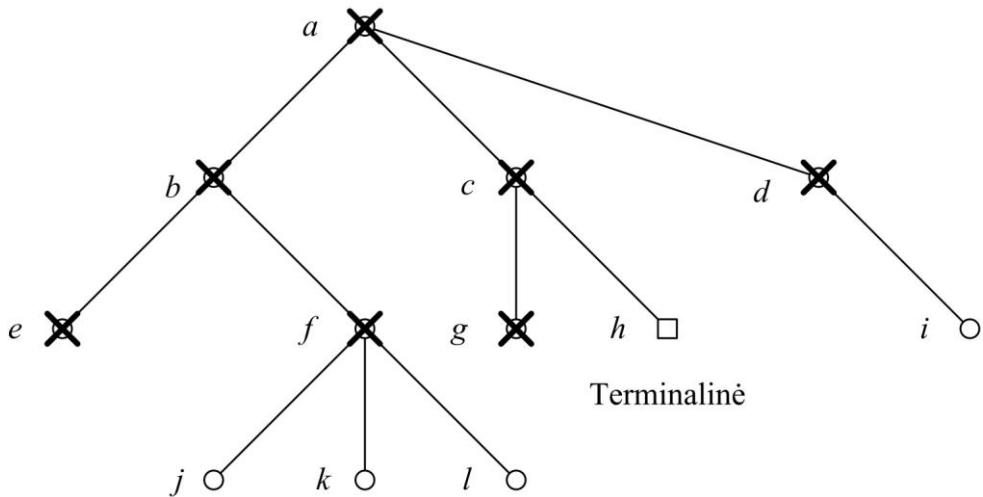
Grafas yra neorientuotas, t. y. briauna $(a,b) \in E$ tada ir tik tada, kai $(b,a) \in E$. Kaip žinia, grafas G yra formalizuojamas kaip pora $G = \langle V, E \rangle$, sudaryta iš viršūnių aibės V ir briaunų aibės E , kur $E \subset V \times V$. Briaunų aibė yra viršūnių aibės Dekarto sandauga iš savęs.

Grafa yra dviejų tipų *išreikštiniai* ir *neišreikštiniai*. Išreikštinis grafas vaizduojamas pateikiant aibes V ir E kaip sąrašus. Išreikštinę grafą yra įprasta vaizduoti grafiškai, pavyzdžiu, 9.5 pav ir 9.8 pav.

Neišreikštinis grafas nusakomas taisyklėmis, kaip yra generuoamos viršūnės ir briaunos. Yra įprasta, kad viršūnės vaizduoja globalios duomenų bazės būsenas, o briaunos – perėjimus tarp šių būsenų. Priimta, kad perėjimas siejamas su produkacija.



9.8 pav. Ieškomas kelias iš a į h . Neorientuotas grafas $G = \langle V, E \rangle$, kur $V = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o\}$, $E = \{(a,b), (a,c), (a,d), (b,e), (b,f), (c,f), (c,g), (d,h), (d,i), (f,j), (f,k), (g,l), (h,l), (h,m), (h,n), (i,n), (i,o)\}$.



9.9 pav. Paieškos medis grafe 9.8 pav. iš pradinės viršūnės a iki terminalinės viršūnės h

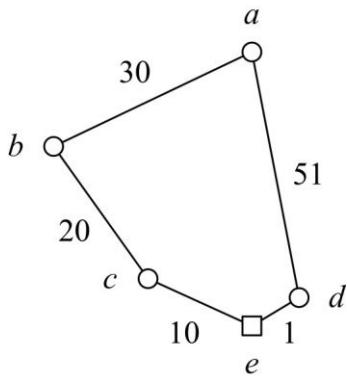
Atidaromas ir uždaromas viršūnes vaizduojame vienam fronte:

$$a, b, e, d, e, f, g, h, i, j, k, l$$

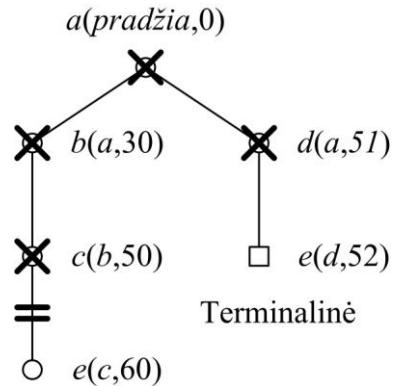
Pavyzdžiui, uždarant viršūnę d , jos vaikų aibė yra $S(d) = \{a, c, h, i\}$. Viršūnės a ir c jau priklauso sąrašui UŽDARYTA, o $h \in$ ATIDARYTA. Todėl į sąrašą ATIDARYTA talpinama tik i . Kai uždaroma viršūnė h , tai pastebima, kad ji yra terminalinė. Tada algoritmas sustoja.

10. Algoritmas paieškai grafe, kai briaunos turi kainas

Algoritmas pateikiamas vadovaujantis [Hunt 1978, 10.1.2]. Paieška iliustruojama grafu 10.1 pav. Paieškos medis pateiktas 10.2 pav.



10.1 pav. Grafas su kainomis. Pradinė viršūnė a , terminalinė – e



10.2 pav. Paieškos medis rasti kelią iš a į e grafe 10.1 pav.

ALGORITMO ĮĖJIMAS: 1) grafas, kurio kiekviena briauna turi savo kainą; 2) pradinė viršūnė; 3) terminalinė viršūnė.

ALGORITMO IŠĖJIMAS: kelias nuo pradinės viršūnės iki terminalinės

1. Patalpinti pradinę viršūnę į sąrašą ATIDARYTA.

2. Jeigu sąrašas ATIDARYTA tuščias, tai kelias neegzistuoja. Baigt darbą, grąžinant nesékmés požymį. Tokia situacija yra gaunama, kai grafas nesusijęs.

3. Iš sąrašo ATIDARYTA į sąrašą UŽDARYTA perkelti tokią viršūnę n , kurios kelio ilgio nuo pradinės viršūnės įvertinimas yra **mažiausias**. Tokiu būdu sąrašas ATIDARYTA turi būti rūšiuojamas. Jei ta viršūnė n yra terminalinė, tai kelias rastas. Surinkti kelią atgal. Baigt darbą.

4. Paimti viršūnės n incidentinių viršūnių aibę $S(n)$. Kiekvienai viršūnei iš n^* iš $S(n)$, kurios nėra sąraše UŽDARYTA, apskaičiuoti naują kelio įvertinimą. Formaliai, $\forall n^* \in S(n)/UŽDARYTA: KELIAS(s,n^*) := KELIAS(s,n) + p(n,n^*)$, kur p yra briaunos (n,n^*) ilgis. Į sąrašą ATIDARYTA įtraukti tas $S(n)$ viršūnes, kurių ATIDARYTA dar nėra. Formaliai, jeigu $n^* \notin ATIDARYTA$, tai $ATIDARYTA := CONS(ATIDARYTA, n^*)$. Jeigu $n^* \in S(n)$ jau priklauso ATIDARYTA, tai palyginti seno ir naujo kelio nuo s iki n^* ilgio įvertinimus. Jeigu naujas kelio įvertinimas yra geresnis, tai įtraukti n^* į ATIDARYTA bei:

1) pakeisti viršūnės n^* nuorodą, kad rodytų į n ;

2) viršūnei n^* priskirti naują kelio įvertinimą.

5. Pereiti prie 2 žingsnio.

Šis algoritmas nevadintinas nei „i gylį“, nei „i plotį“. Taip yra todėl, kad 3 algoritmo žingsnyje uždaroma iš sąrašo ATIDARYTA ta viršūnė, kurios įvertinimas yra mažiausias. Sąrašas ATIDARYTA formuojamas pagal mažiausią įvertinimą, o ne pagal principą „i gylį“ arba „i plotį“ talpinamas viršūnes.

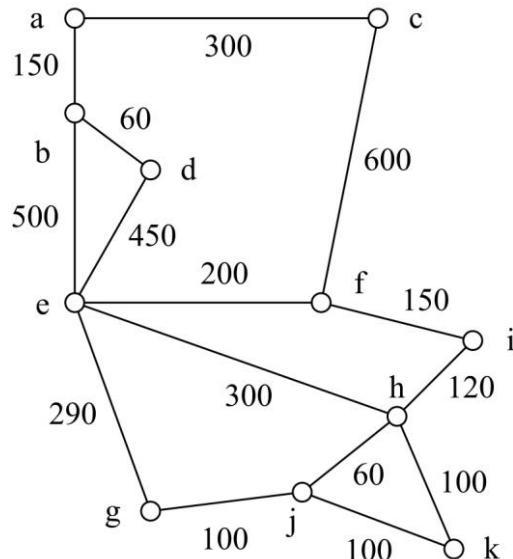
Zemiau lentelėje pateikiamas ATIDARYTA ir UŽDARYTA būsenos algoritmo žingsniuose.

	ATIDARYTA	UŽDARYTA	Komentaras
1	$a(pradžia, 0)$	\emptyset	
2	$b(a, 30), d(a, 51)$	$a(pradžia, 0)$	$S(a) = \{b, d\}$
3	$c(b, 50), d(a, 51)$	$a(pradžia, 0), b(a, 30)$	$S(b) = \{a, c\}$, bet $a \in UŽDARYTA$
4	$d(a, 51), e(c, 60)$	$a(pradžia, 0), b(a, 30), c(b, 50)$	$S(c) = \{b, e\}$, bet $b \in UŽDARYTA$
5	$e(d, 52)$	$a(pradžia, 0), b(a, 30), c(b, 50), d(a, 51)$	$S(d) = \{a, e\}$, bet $a \in UŽDARYTA$ ir $e \in ATIDARYTA$. Naujas įvertinimas $e(d, 52)$ yra geresnis negu senas $e(c, 60)$, todėl ir pasirenkamas
6	\emptyset	$a(pradžia, 0), b(a, 30), c(b, 50), d(a, 51), e(d, 52)$	Uždaroma terminalinė e

10.3 lentelė. Sąrašų ATIDARYTA ir UŽDARYTA būsenos algoritmo žingsniuose, ieškant kelio iš pradinės viršūnės a į terminalinę e
10.1 pav. Rastas kelias $\langle a(pradžia, 0), d(a, 51), e(d, 52) \rangle$

Atkreipkime dėmesį į nuorodos pakeitimą 5 žingsnyje. Uždarant viršūnę d , jos vaiko e kelio $e(d, 52)$ per d įvertinimas yra geresnis už jau sąraše esančių kelių per c įvertinimą $e(c, 60)$, nes $52 < 60$. Todėl pasirenkama briauna $e(d, 52)$ – su geresniu kelio įvertinimu. Algoritmo darbas baigiamas 6 žingsnyje, kai į sąrašą UŽDARYTA patalpinama terminalinė e . Rastas kelias $\langle a(pradžia, 0), d(a, 51), e(d, 52) \rangle$ arba trumpai tiesiog $\langle a, d, e \rangle$.

Toliau pateikiamas kitas pavyzdys. Grafas parodytas 10.4 pav.



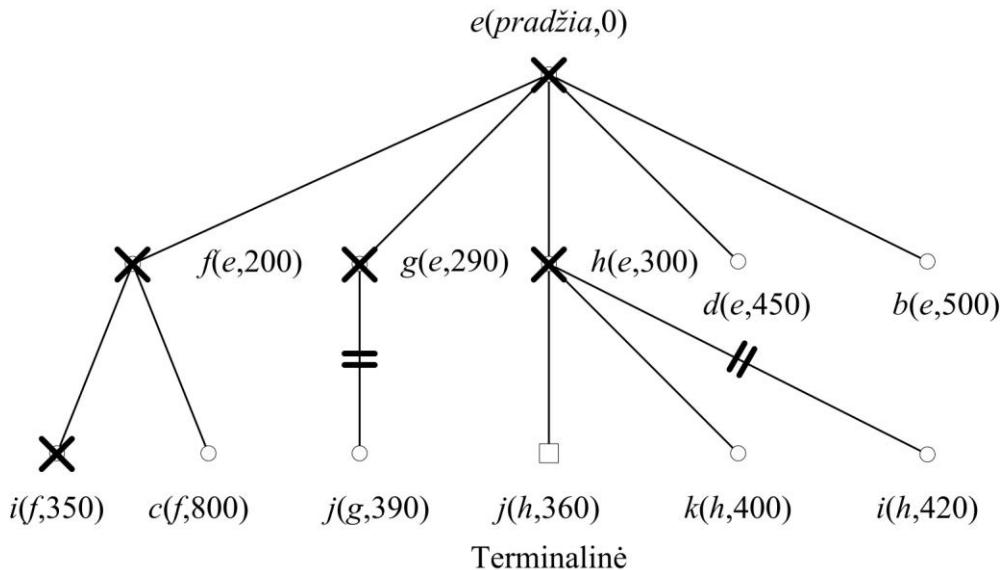
10.4 pav. Grafas, kurio briaunos turi kainas. Ieškomas kelias iš $e \in j$

Sąrašų ATIDARYTA ir UŽDARYTA būsenos atskiruose žingsniuose:

	ATIDARYTA	UŽDARYTA	Komentaras
1	$e(pradžia,0)$	\emptyset	
2	$f(e,200), g(e,290), h(e,300), d(e,450), b(e,500)$	$e(pradžia,0)$	$S(e) = \{b,d,f,g,h\}$
3	$g(e,290), h(e,300), i(f,350), d(e,450), b(e,500), c(f,800)$	$e(pradžia,0), f(e,200)$	$S(f) = \{c,e,i\}$, bet $e \in$ UŽDARYTA
4	$h(e,300), i(f,350), j(g,390), d(e,450), b(e,500), c(f,800)$	$e(pradžia,0), f(e,200), g(e,290)$	$S(g) = \{e,j\}$, bet $e \in$ UŽDARYTA
5	$i(f,350), j(h,360), k(h,400), d(e,450), b(e,500), c(f,800)$	$e(pradžia,0), f(e,200), g(e,290), h(e,300)$	$S(h) = \{e,i,j,k\}$, bet $e \in$ UŽDARYTA ir $i,j \in$ ATIDARYTA. Naujas i įvertinimas $i(h,420)$ yra blogesnis negu senas $i(f,350)$, todėl paliekame senajį. Naujas j įvertinimas $j(h,360)$ yra geresnis negu senas $j(g,390)$, todėl ir pasirenkamas
6	$j(h,360), k(h,400), d(e,450), b(e,500), c(f,800)$	$e(pradžia,0), f(e,200), g(e,290), h(e,300), i(f,350)$	$S(i) = \{f,h\}$, bet $f,h \in$ UŽDARYTA
7	$k(h,400), d(e,450), b(e,500), c(f,800)$	$e(pradžia,0), f(e,200), g(e,290), h(e,300), i(f,350), j(h,360)$	Uždaroma terminalinė j. Jos vaikų talpinti nebereikia

10.5 lentelė. Sąrašų būsenos algoritmo žingsniuose, ieškant kelio iš e į j
 10.4 pav. Rastas kelias $\langle e(pradžia,0), h(e,300), j(h,360) \rangle$

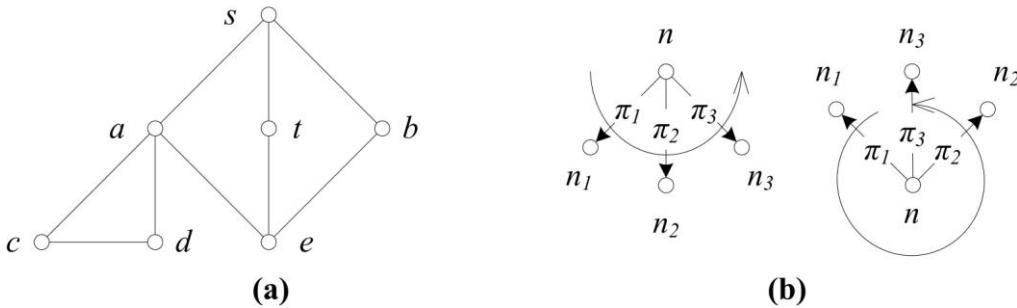
Paieškos medis atitinkantis aukščiau nagrinėtą uždavinį pateiktas 10.6 pav.



10.6 pav. Paieškos medis paieškai grafe 10.4 pav. iš e į j

11. Paieškos į gylį algoritmas grafe be kainų

Skirtumą tarp paieškos į gylį ir į plotį grafe be kainų pademonstruosime kelio paieška iš pradinės viršūnės s į terminalinę t grafe, parodytame 11.1 a pav. Tegu vaikai yra perrenkami „prieš laikrodžio rodyklę“, t.y. „dvyliktos valandos“ produkcija yra paskutinė (11.1 b pav.).



11.1 pav. a) Grafas, kurio briaunos neturi kainos. b) Briaunų (kaip produkcijų) perrinkimo tvarka „prieš laikrodžio rodyklę“. Produkcija „dvylikta valanda“ imama paskutinė, o ne pirmoji

Paieškos į plotį algoritmas randa kelią $\langle s,t \rangle$. Šis kelias bus randamas uždarant pirmają bangą (skaičiuojant nuo nulio; šiame gylyje yra s). Iš pradžių uždaroma viršūnė s ir atidaromi jos vaikai $S(s) = \{a, t, b\}$. Toliau pastarieji iš eilės uždarinėjami. Uždaręs viršūnę t , paieškos į plotį algoritmas baigia darbą.

Paieškos į gylį (angl. *depth-first search*) algoritmas grafe skiriasi uždaromos viršūnės vaikų talpinimo į sąrašą ATIDARYTA tvarka. Paieškos į plotį atveju, visi vaikai talpinami į sąrašo **pabaigą**, t. y. sąrašo (angl. *FIFO – First In First Out*) principu. O paieškos į gylį atveju – į jo **pradžią**, t. y. steko (angl. *LIFO – Last In First Out*) principu.

Paieškos į gylį algoritmas grafe, kai grafo briaunos neturi kainų yra užrašomas šitaip.

ALGORITMO JĒJIMAS: 1) grafas, kurio briaunos neturi kainų; 2) pradinė viršūnė; 3) terminalinė viršūnė.

ALGORITMO IŠJIMAS: kelias nuo pradinės viršūnės iki terminalinės

- Pradinę viršūnę patalpinti į sąrašą ATIDARYTA. Sąrašas UŽDARYTA šiame žingsnyje yra tuščias.

- Jeigu sąrašas ATIDARYTA tuščias, tai kelias neegzistuoja; Baigt darbą, grąžinant nesékmės požymį. Tokia situacija yra gaunama, kai grafas yra nesusijęs (žr. 9.1 pav.).

- Uždaryti **pirmąją** viršūnę n iš sąrašo ATIDARYTA, t. y. perkelti ją į sąrašo ATIDARYTA į sąrašą UŽDARYTA. Jeigu n yra terminalinė, tai kelias rastas. Surinkti jį, sekant nuorodomis atgal.

- Paimti viršūnės n incidentinių viršūnių aibę $S(n)$. Tas viršūnes, kurios neuždarytos, t. y. $S(n)/UŽDARYTA$, patalpinti į ATIDARYTA **pradžią**. Tas viršūnes, kurios kartojasi, t. y. $S(n) \cap ATIDARYTA$, pašalinti iš sąrašo ATIDARYTA pabaigos. Tokiu būdu pakartotinai atidaromoms viršūnėms yra keičiamos nuorodos, rodančios, iš kur į jas buvo ateita.

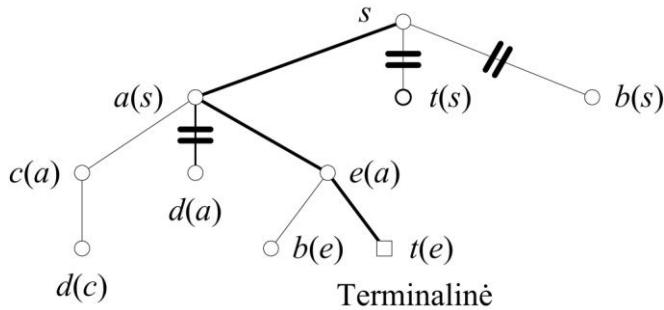
- Pereiti prie 2 žingsnio.

Panagrinėkime kelio iš s į t paiešką į gylį grafe 11.1 a pav. Sąrašų ATIDARYTA ir UŽDARYTA būsenos pažingsniui algoritmo vykdymo metu:

	ATIDARYTA	UŽDARYTA	Komentaras
1	$s(start)$	\emptyset	
2	$a(s), t(s), b(s)$	$s(start)$	$S(s) = \{a, t, b\}$
3	$c(a), d(a), e(a), t(s), b(s)$	$s(start), a(s)$	$S(a) = \{c, d, e, s\}$, bet $s \in UŽDARYTA$
4	$d(c), e(a), t(s), b(s)$	$s(start), a(s), c(a)$	$S(c) = \{d, a\}$, bet $a \in UŽDARYTA$ ir $d \in ATIDARYTA$. Todėl $d(c)$, kuri atidaroma vėliau, yra talpinama į ATIDARYTA pradžią, o $d(a)$ iš ATIDARYTA pabaigos pašalinama
5	$e(a), t(s), b(s)$	$s(start), a(s), c(a), d(c)$	$S(d) = \{c, a\}$, bet $c, a \in UŽDARYTA$
6	$b(e), t(e)$	$s(start), a(s), c(a), d(c), e(a)$	$S(e) = \{a, b, t\}$, bet $a \in UŽDARYTA$ ir $b, t \in ATIDARYTA$. Todėl $b(e)$ ir $t(e)$, kurios atidaromos vėliau, yra talpinamos į ATIDARYTA pradžią, o $t(s)$ ir $b(s)$ iš ATIDARYTA pabaigos pašalinamos
7	$t(e)$	$s(start), a(s), c(a), d(c), e(a), b(e)$	$S(b) = \{s, e\}$, bet $s, e \in UŽDARYTA$
8	\emptyset	$s(start), a(s), c(a), d(c), e(a), b(e), t(e)$	Uždaroma terminalinė t . Surenkamas kelias: t pasiekta iš e , e iš a , a iš s , kuri yra pradinė

11.2 lentelė. Sąrašų ATIDARYTA ir UŽDARYTA būsenos algoritmo žingsniuose, kai ieškomas kelias iš pradinės viršūnės s į terminalinę t 11.1 a pav. Rastas kelias $\langle s, a, e, t \rangle$ – trys briaunos

Paieškos medis parodytas 11.3 pav.



11.3 pav. Paieškos medis rasti kelią paieškos į gylį algoritmu iš pradinės viršūnės s į terminalinę t grafe 11.1 a pav. Viršūnėms d , b ir t yra keičiamos nuorodos: į $d(c)$ vietoje $d(a)$, į $b(e)$ vietoje $b(s)$ ir į $t(e)$ vietoje $t(s)$. Šitaip nukirstos šakos pažymėtos „=“

Pastebėsime, kad paieškos į gylį algoritmu rastas kelias $\langle s, a, e, t \rangle$ nėra trumpiausias. Jo ilgis – trys briaunos. Paieškos į plotį algoritmu rastas kelias $\langle s, t \rangle$ turi vieną briauną. Paieškos į gylį algoritmas iliustruoja trumpalaikės atminties principą. Yra įsimenama vėliau atėjusi viršūnė.

Paieškos grafe „i gylį“ algoritmo interpretavimo žingsniai yra parodyti lentelėje žemiau:

	ATIDARYTA	UŽDARYTA	Komentaras	Paieškos medis „i gylį“
1	$s(start)$	\emptyset	Pradinė viršūnė s patalpinama į ATIDARYTA	
2	$a(s), t(s), b(s)$	$s(start)$	Uždaroma pirmoji viršūnė s iš ATIDARYTA, t. y. Perkeliamas į UŽDARYTA. Jos incidentinės $S(s) = \{a,t,b\}$ patalpinamos į ATIDARYTA	
3	$c(a), d(a), e(a), t(s), b(s)$	$s(start), a(s)$	Uždaroma pirmoji viršūnė a iš ATIDARYTA. Jos incidentinės yra $S(a) = \{c,d,e,s\}$. s yra UŽDARYTA, todėl nenagrinėjama. Tos viršūnės, kurios nepriklauso UŽDARYTA, t. y. $\{c,d,e\}$, patalpinamos į ATIDARYTA pradžią	
4	$d(c), e(a), t(s), b(s)$	$s(start), a(s), c(a)$	Uždaroma pirmoji viršūnė c iš ATIDARYTA. Jos incidentinės yra $S(c) = \{d,a\}$. a yra UŽDARYTA, todėl nenagrinėjama. d patalpinama į ATIDARYTA pradžią , o iš pabaigos pašalinama. Tokiu būdu pakeičiama nuoroda į $d(c)$ vietoje $d(a)$	
5	$e(a), t(s), b(s)$	$s(start), a(s), c(a), d(c)$	Uždaroma pirmoji viršūnė d iš ATIDARYTA, t. y. Perkeliamas į UŽDARYTA. Jos incidentinės $S(a) = \{c,a\}$ yra UŽDARYTA, todėl nenagrinėjamos	
6	$b(e), t(e)$	$s(start), a(s), c(a), d(c), e(a)$	Uždaroma pirmoji viršūnė e iš ATIDARYTA. Jos incidentinės yra $S(e) = \{a,b,t\}$. a yra UŽDARYTA, todėl nenagrinėjama. b ir t patalpinamos į ATIDARYTA pradžią , o iš pabaigos pašalinamos. Keičiamos jų nuorodos: į $b(e)$ vietoje $b(s)$ ir $t(e)$ vietoje $t(s)$	
7	$t(e)$	$s(start), a(s), c(a), d(c), e(a), b(e)$	Uždaroma pirmoji viršūnė b iš ATIDARYTA. Jos incidentinės $S(b) = \{s,e\}$ yra UŽDARYTA, todėl nenagrinėjamos	

8	\emptyset	$s(start), a(s), c(a), d(c), e(a), b(e), t(e)$	Uždaroma pirmoji viršūnė t iš ATIDARYTA. Ji terminalinė, todėl kelias rastas. Sekant nuorodomis atgal, kelias surenkamas: t iš e , e iš a , a iš s , kuri pradinė. Kelias $\langle s,a,e,t \rangle$.	
---	-------------	--	---	--

11.4 lentelė Sąrašų ATIDARYTA ir UŽDARYTA būsenos paieškos „*i gylį*“ žingsniuose, kai ieškomas kelias iš s į t grafe 11.1 a pav. Randamas kelias $\langle s,a,e,t \rangle$ – trys briaunos

11.1. Paieškos „*i ploti*“ skirtumas nuo „*i gylį*“

Pademonstruosime paieškos *i ploti* ir *i gylį* skirtumus. Nagrinėkime grafą 11.1 a pav. Paieškos „*i ploti*“ interpretavimo žingsniai yra parodyti lentelėje žemiau:

	ATIDARYTA	UŽDARYTA ir frontas	Komentaras	Paieškos medis „ <i>i ploti</i> “
1	s	\emptyset	Pradinė viršūnė s patalpinama į ATIDARYTA	
2	a,t,b	s Frontas: s,a,t,b	Uždaroma pirmoji viršūnė s iš ATIDARYTA, t. y. perkeliama į UŽDARYTA. s incidentinės $S(s) = \{a,t,b\}$ patalpinamos į ATIDARYTA	
3	t,b,c,d,e	s,a Frontas: s,a,t,b,c,d,e	Uždaroma pirmoji viršūnė a iš ATIDARYTA, t. y. perkeliama į UŽDARYTA. Jos incidentinės yra $S(a) = \{c,d,e,s\}$. s yra UŽDARYTA, todėl ji nenagrinėjama. Tos viršūnės, kurios nepriklauso nei ATIDARYTA, nei UŽDARYTA, t. y. c , d ir e , patalpinamos į ATIDARYTA pabaiga	
4	b,c,d,e	s,a,t Frontas: s,a,t,b,c,d,e	Uždaroma pirmoji viršūnė t iš ATIDARYTA. Ji terminalinė, todėl kelias rastas. Sekant nuorodomis atgal, kelias surenkamas: t iš s , kuri pradinė. Rastas kelias $\langle s,t \rangle$.	

11.5 lentelė Sąrašų ATIDARYTA ir UŽDARYTA būsenos paieškos „*i ploti*“ žingsniuose, kai ieškomas kelias iš s į t grafe 11.1 a pav. Randamas kelias $\langle s,t \rangle$ – viena briauna. Primename, kad paieškos *i ploti* algoritmas randa trumpiausią kelią

11.2. *Sprendėjas ir planuotojas*

Iš pradžių priminsime paieškos į gylį ir į plotį principus, o paskui paaiškinsime *sprendėjo* ir *planuotojo* sąvokas. Paieška į gylį siejama su trumpalaikės atminties, t. y. steko principu – veliau gimusi viršūnė yra uždaroma anksčiau. Paieška į plotį siejama su sarašo principu, t. y. veliau gimusi viršūnė yra ir uždaroma vėliau. Sarašus ATIDARYTA ir UŽDARYTA galime palyginti su gyvaisiais ir mirusiais. Gyvujų eilė gali būti peržiūrima. O mirusiuju jau niekas netrikdo.

Algoritmas į gylį arba į plotį pasirenkamas atsižvelgiant į skirtingus motyvus. Paieška į gylį paprastai naudojasi uždavinio *sprendėjas* (angl. *problem-solver*). Sinonimas yra *sprendžiantis agentas* (angl. *solving agent*). Iprasta kad sprendėjo uždavinio prigimtis yra spręsti jį vieną kartą. Pavyzdžiui, ištūkti iš labirinto. Sprendėjas neturi prieš akis labirinto žemėlapio. Ištūkti iš šalto labirinto yra jo gyvybės ar mirties klausimas.

Paieška į plotį paprastai naudojasi *planuotojas* (angl. *planner*). Sinonimas yra *planuojantis agentas* (angl. *planning agent*). Racionalu, kad planuotojas prieš akis turi žemėlapį. Pavyzdžiui, planuotojas sėdi patogiame krėslė šiltame kambaryje ir ieško trumpiausio maršruto krovinui gabenti. Rastas kelias savo esme yra tam tikro uždavinio sprendimo planas. Vieną kartą rastas planas paprastai taikomas daug kartų. Todėl yra prasminga ieškoti trumpiausio plano. Paieška į plotį randa trumpiausią kelią, kuris, kaip minėjome, yra suprantamas kaip planas. Pavyzdžiui, grafe rastu trumpiausiu keliu vežėjui yra prasminga daug kartų gabenti krovinį iš pradinio miesto s į terminalinį t. Planavimas paprastai atima daugiau planuotojo resursų, negu sprendėjo. Bet planavimo algoritmas vykdomas tik vieną kartą. O vežėjas gabendamas trumpiausiu keliu sutaupo kiekvieną kartą.

Algoritmą paieškai į gylį galima palyginti su pasivaikščiojimu neturint tikslų rasti trumpiausią kelią. Paieškos į plotį tikslas ne tik vaikščioti, bet ir rasti trumpiausią kelią.

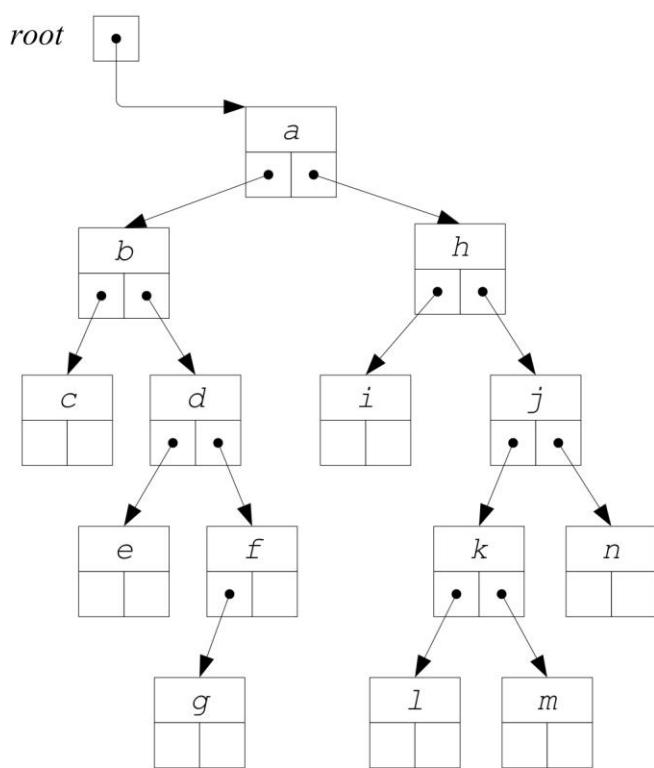
Galima iškelti klausimą, kuris algoritmas geresnis? Keliant tokį klausimą turi būti apibrėžtas vertinimo kriterijus. Paieška į plotį naudojama ieškant trumpiausio kelio. Paieška į gylį siejama su „intelekto“ sąvoka. Laikoma kad žmogaus trumpalaikė atmintis yra pritaikyta paieškai į gylį. Tokiu būdu vienoje situacijoje gali geriau tiki vienas algoritmas, pavyzdžiui, kai iš anksto nežinomas dalykinės srities būsenų grafas, kitoje – kitas, kai žinomas.

12. Prefiksinė, infiksinė ir postfiksinė medžio apėjimo tvarka

Iki šiol aptarėme medžio apėjimo tvarką „i gylį“ ir „i plotį“. Yra ir kitokių visuotinai priimtų būdų. Jeigu medyje yra N viršūnių, tai apėjimo būdų yra tiek, kiek yra viršūnių perstatų, t. y. N faktorialas $N! = N \cdot (N-1) \cdot (N-2) \dots 1$. Čia apėjimo būdas suprantamas kaip viršūnių išvardinimo seka, t. y. viršūnių $\{1, 2, 3, \dots, N\}$ perstata.

Keletas tvarkų – prefiksinė, infiksinė ir postfiksinė – yra išskiriamos ir paprastai pristatomos kurse „Duomenų struktūros ir algoritmai“. Šios tvarkos pristatomos dvejetainio medžio atveju. Tačiau prefiksinė ir postfiksinė medžio apėjimo tvarkos gali būti taikomos bet kokiems medžiams.

Dvejetainis medis trumpai yra apibrėžiamas, kaip medis kurio kiekviena viršūnė turi du, vieną arba nei vieno vaikų. Viena viršūnė medyje yra išskiriama ir vadinama šaknimi. Medis, prasidedantis kurioje nors viršūnėje, vadinamas pomedžiu.



12.1 pav. Dvejetainio medžio pavyzdys iš [Jensen, Wirth 1982, p. 66]. Jį vaizduoja simbolių eilutę
 $abc\dots de\dots fg\dots hi\dots jkl\dots m\dots n\dots$

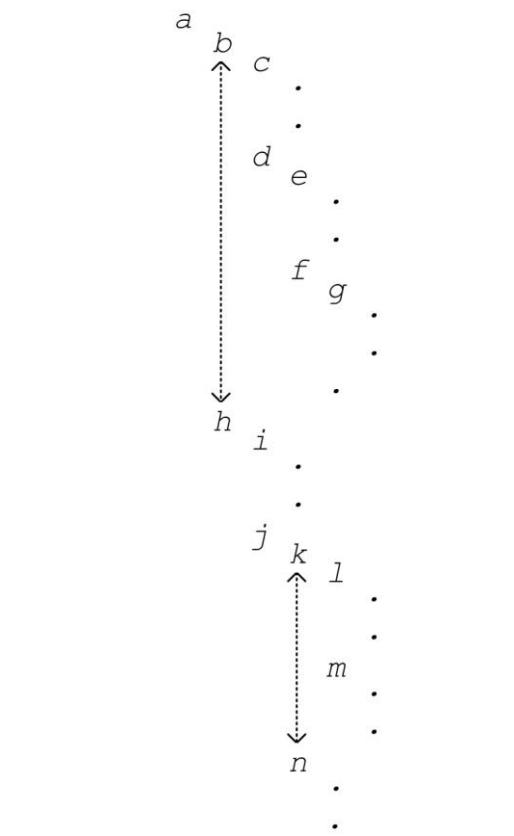


Fig. 12.2. Izomorfiškas pav. 12.1 medis, pavaizduotas vadovaujantis knygos turinio metafora

Prefiksinė dvejetainio medžio apėjimo tvarka (angl. *prefix order*) – medžio viršūnės vardinamos taip: viršūnė, kairysis pomedis, dešinysis pomedis. Mes vaizduojame 100, kur 1 žymi vardinimą (t. y. informacijos, siejamos su viršūne apdorojimą), o 0 – pomedžio apėjimą.

Infiksinė dvejetainio medžio apėjimo tvarka (angl. *infix order*) – medžio viršūnės vardinamos taip: kairysis pomedis, viršūnė, dešinysis pomedis. Mes vaizduojame 010.

Postfiksinė dvejetainio medžio apėjimo tvarka (angl. *postfix order*) – medžio viršūnės vardinamos taip: kairysis pomedis, dešinysis pomedis, viršūnė. Mes vaizduojame 001.

Toliau dvejetainio medžio pavyzdys 12.1 pav. ir apėjimo programos pateikiamos vadovaujantis K. Jensen ir N. Virtos knygos vertimo į rusų kalbą [Jensen, Wirth 1982, 65-67] 12.1 poskyriu. Šiame poskyryje pristatomos apėjimo procedūros. Priminsime, kad N. Virtas yra žinomas kaip Paskalio programavimo kalbos autorius. Procedūros iliustruoja ne tik medžio apėjimą, bet ir rekursiją.

Medis, pavaizduotas 12.1 pav. suformuojamas, įvedus tokią simbolių eilutę:

abc..de...fg...hi..jk1..m...n..

Šią eilutę galima suprasti pasitelkiant knygos turinio metaforą. Simbolių eilutė vaizduoja, kaip knygos struktūrinės dalys – skyriai ir jų poskyriai – yra su atitraukimais žymimos turinyje (žr. 12.2 pav). Knygos skaitytojas turinio medži suvokia prefiksine tvarka.

```
program traversal(input, output);
type ptr = ^node;
node = record
    info : char;
    llink, rlink : ptr;
end;
var root : ptr;
    ch : char;

procedure enter(var p:ptr);
begin { enter }
    read(ch);
    write(ch);
    if(ch ≠ '.') then
        begin
            new(p);
            p^.info:=ch;
            enter(p^.llink);
            enter(p^.rlink);
        end;
    else p := nil;
end; { enter }

procedure preorder(p:ptr);
begin
    if p ≠ nil then
        begin
            write(p^.info);
            preorder(p^.llink);
            preorder(p^.rlink);
        end;
end; { preorder }

procedure inorder(p:ptr);
begin
    if p ≠ nil then
        begin
            inorder(p^.llink);
            write(p^.info);
            inorder(p^.rlink);
        end;
end; { inorder }
```

```
procedure postorder(p:ptr);
begin
  if p ≠ nil then
  begin
    postorder(p^.llink);
    postorder(p^.rlink);
    write(p^.info);
  end;
end; { postorder }

begin
  write(' '); enter(root); writeln;
  write(' '); preorder(root); writeln;
  write(' '); inorder(root); writeln;
  write(' '); postorder(root); writeln;
end.
```

Programa spausdina keturias eilutes:

abc..de..fg...hi..jkl..m..n..	– tai pradiniai duomenys
abcdefghijklmn	– prefiksinė tvarka, procedūra preorder
cbedgfaihlkmjn	– infiksinė tvarka, procedūra inorder
cegfdbilmknjha	– postfiksinė tvarka, procedūra postorder.

Dvejetainio medžio 12.1 pav. apėjimas **į gylį** yra tokis:

abcdefghijklmn

Pastebime, kad jis sutampa su prefiksine medžio apėjimo tvarka. Dvejetainiame medyje kairioji briauna (t. y. `llink`) yra traktuojama kaip produkcija π_1 , o dešinioji briauna (t. y. `rlink`) – kaip produkcija π_2 , kur produkcijų aibę sudaro dvi produkcijos: $\{\pi_1, \pi_2\}$.

Dvejetainio medžio 12.1 pav. apėjimas **į plotį** yra tokis:

abhcdijefknglm

Kokių tikslų informatikos kurse yra dėstomas *prefiksinė*, *postfiksinė* ir *infiksinė* medžių apėjimo tvarkos? Kompiliatorius aritmetinę išraišką transliuoja į užrašą, kuris vadinamas *postfiksine forma*. Šitaip kompiliatorius visą programą gali transliuoti į stekinės mašinos kodą. Šis kodas gali būti interpretuojamas, ir tokiu būdu programa vykdoma.

Infiksinė aritmetinės išraiškos užrašymo forma – tai mums įprasta aritmetinės išraiškos užrašymo forma. Paprastai dar naudojami skliaustai kaip metasimboliai, norint nurodyti veiksmus su įdėtinėmis išraiškomis, pavyzdžiu:

$$((A+B) * (C-D)+E) * F$$

Šią išraišką atitinkantis sintaksinis medis pavaizduotas 12.3 pav.

Prefiksinė aritmetinės išraiškos užrašymo forma užrašoma rekursyviai – operacijos ženklas, pirmasis operandas, antrasis operandas:

$$*+ *+AB-CDEF$$

Postfiksinė aritmetinės išraiškos užrašymo forma užrašoma rekursyviai – pirmasis operandas, antrasis operandas, operacijos ženklas:

$AB+CD-*E+F*$

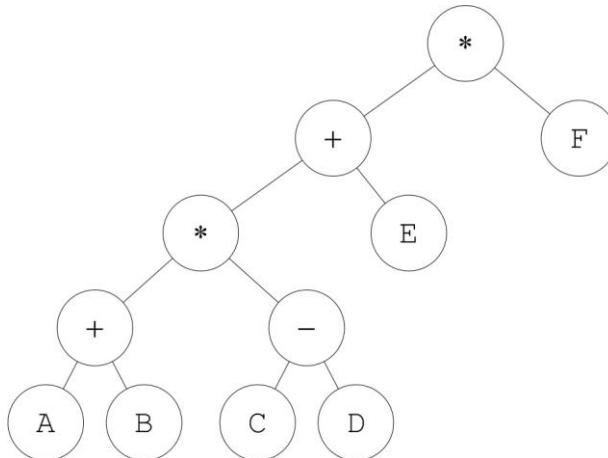


Fig. 12.3. Aritmetinės išraiškos
 $((A+B)*(C-D))+E)*F$
 sintaksinis medis

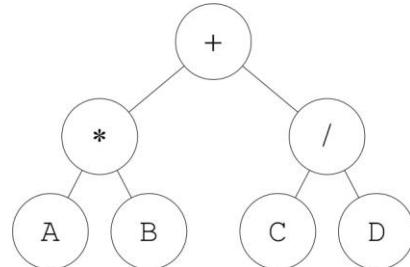


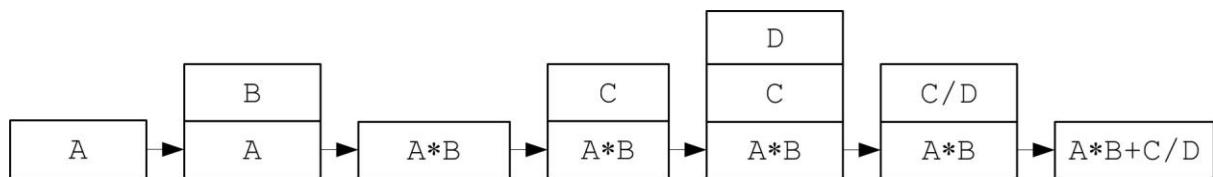
Fig. 12.4. Aritmetinės išraiškos $A*B+C/D$
 sintaksinis medis

Kitas pavyzdys iliustruoja aritmetinės išraiškos kompiliavimą į tarpinę kalbą, skirtą stekinei mašinai. Pavyzdžiuui, aritmetinė išraiška $A*B+C/D$ (jos sintaksinis medis yra 12.4 pav.) kompiliuojama į šią stekinės mašinos komandų seką:

```

Load A
Load B
Multiply
Load C
Load D
Divide
Add
  
```

Stekinės mašinos atminties – steko – kitimas laiko taktais, kai vykdomas aukščiau pateiktas kodas, yra pateiktas 12.5 pav.



12.5 pav. Steko turinio kitimas laiko taktais, kai skaičiuojama aritmetinė išraiška $A*B + C/D$. Rodyklė vaizduoja steko būsenos perėjimą iš steko būsenos laiko momentu t į būseną laiko momentu $t+1$

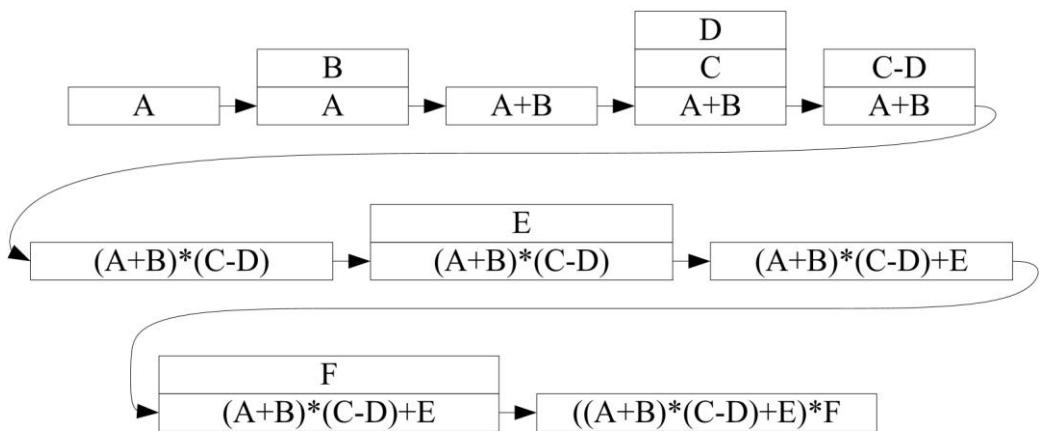
Kitas pavyzdys iliustruoja dar sudėtingesnės aritmetinės išraiškos, $((A+B)* (C-D)+E)*F$, kurios sintaksinis medis pateiktas 12.3 pav. steko turinio kitimas laiko taktais. Pastebime, kad infiksine medžio apėjimo tvarką $A+B*C-D+E*F$, kurioje nėra skliaustų, žmogus interpretuotų kitaip. Tačiau kompiiatorius operuoja aritmetinės išraiškos sintaksiniu medžiu. Šitaip operacijų atlikimo tvarka neiškraipoma. Nurodyta aritmetinė išraiška su skliaustais yra kompiliuojama į tokią stekinės mašinos programą:

```

Load A
Load B
Add
Load C
Load D
Subtract
Multiply
Load E
Add
Load F
Multiply

```

Stekinės mašinos atminties – steko – kitimas laiko taktais, kai vykdomas aukščiau pateiktas kodas, yra pateiktas 12.6 pav.:



12.6 pav. Steko turinio kitimas laiko taktais, kai skaičiuojama aritmetinė išraiška $((A+B)*(C-D)+E)*F$. Rodyklė vaizduoja steko būsenos laiko momentu t perėjimą į būseną laiko momentu $t+1$

Pratimai

1. Užrašykite išraišką $A+B*C-(D+E)*F-B$ prefiksine, infiksine ir postfiksine formomis.
2. Iš kokį stekinės mašinos kodą sutransliuoja išraiška iš pratimo aukščiau?
3. Pavaizduokite steko turinio kitimą žingsniais, kai skaičiuojama išraiška 1 pratime.

13. Bendras paieškos grafe algoritmas GRAPHSEARCH

Ankstesnėse temose išnagrinėjome algoritmus „i gylį“ ir „i plotį“. Apjunkime šias paieškas į vieną procedūrą. Ši procedūra sukuria išreikštinį paieškos medį grafe. Pastarasis grafas gali būti tiek išreikštinis, tiek neišreikštinis. Neišreikštinio grafo pavyzdžiais gali būti žaidimų, tokį kaip 8 kauliukai, kryžiukai-nuliukai, šaškės, šachmatai ir kt. būsenų medžiai.

ALGORITMO IĖJIMAS: 1) grafas; 2) pradinė viršūnė; 3) terminalinė viršūnė (jų gali būti keletas).

ALGORITMO IŠĖJIMAS: kelias nuo pradinės viršūnės iki terminalinės

1. Sukurti paieškos medį T , susidedantį iš vienos viršūnės – pradinės viršūnės s . Patalpiname s į sąrašą ATIDARYTA.
2. Sukurti sąrašą UŽDARYTA, kyris iš pradžių yra tuščias. Formaliai $UŽDARYTA := \emptyset$.
3. Jeigu ATIDARYTA tuščias, tai nesėkmė. Grąžiname nesėkmės požymį ir baigiamo darbą. Sąrašo ATIDARYTA išsisémimas reiškia, kad kelias iš s į terminalinę viršūnę neegzistuoja.
4. Paimti **pirmąją** viršūnę n iš sąrašo ATIDARYTA ir perkelti į UŽDARYTA.
5. Jeigu n yra terminalinė viršūnė, tai sėkmė. Sekdami nuorodomis atgal nuo n iki s , surenkame kelią ir baigiamo STOP.
6. Išskleisti n , t. y. paimti n incidentinių viršūnių aibę $S(n)$. Tuos $n^* \in S(n)$, kurių nėra nei sąraše ATIDARYTA, nei UŽDARYTA, t. y. $n^* \in (S(n) / (ATIDARYTA \cup UŽDARYTA))$, patalpinti į T ir suformuoti nuorodas į n . Patalpinti šiuos n^* į ATIDARYTA.
7. Kiekvienai viršūnei $n^* \in S(n)$, kuri priklauso ATIDARYTA, t. y. $n^* \in (S(n) \cap ATIDARYTA)$, nuspręsti, ar perorientuoti nuorodą. O visų kitų, t. y. $n^* \in (S(n) \cap UŽDARYTA)$, nenagrinėti.
8. Sutvarkyti ATIDARYTA pagal kokią nors schemą arba euristiką. Pavyzdžiui,
 - 1) pagal kainą;
 - 2) „**i gylį**“, t. y. $S(n) / UŽDARYTA$ patalpinti į ATIDARYTA **pradžią**, o $S(n) \cap ATIDARYTA$ iš naujojo ATIDARYTA pabaigos pašalinti, kad nesikartotų. Sutvarkyti **steko** principu, t. y. LIFO (Last In First Out).
 - 3) „**i plotį**“, t. y. $S(n) / UŽDARYTA$ patalpinti į naujojo ATIDARYTA **pabaigą**, o $S(n) \cap ATIDARYTA$ iš naujojo ATIDARYTA **pradžios** pašalinti. Sutvarkyti **eilės** principu, t. y. FIFO (First In First Out).
9. Pereiti prie 3 žingsnio.

GRAPHSEARCH algoritmo aiškinimą galima rasti [Nilsson 1982] 2.2.2 skyriuje.

14. Algoritmų BACKTRACK1 ir GRAPHSEARCH skirtumas

Keliais pavyzdžiais pademonstruosime skirtumą tarp algoritmo BACKTRACK1 ir GRAPHSEARCH_I_GYL. Kaip buvo minėta ankščiau, BACKTRACK1 įvardinamas sinonimu BACKTRACK_SU_STEKU. Čia steko vaidmenį atlieka procedūros parametras DATALIST. Jo tipas yra sąrašas, bet grįžtant iš rekursijos yra grįztama į ankstesnę sąrašo būseną. Tokiu būdu DATALIST „kvėpuoja“ kaip stekas.

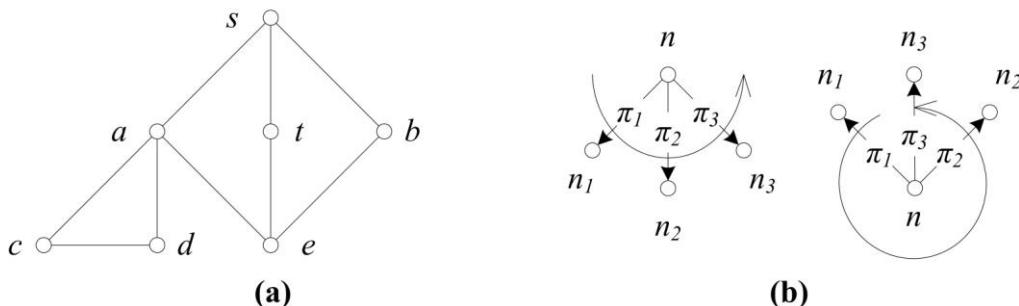
Algoritmų skirtumas lakoniškai apibūdinamas dviem teiginiais.

1. Procedūra BACKTRACK_SU_STEKU vadovaujasi sąvokomis a) produkcija ir b) globalios duomenų bazės būsena, o GRAPHSEARCH_I_GYL – a) atidarytos viršūnės ir b) uždarytos viršūnės.
2. Procedūra BACKTRACK_SU_STEKU aplink „salas“ eina *keletą* kartų, nes sąraše DATALIST kaip steke saugo tik einamajį kelią. Procedūra GRAPHSEARCH_I_GYL aplink salą eina tik *vieną* kartą – už laiką ji moka atmintimi.

Skirtingų kriterijų (mūsų atveju – atminties ir laiko) pusiausvyros principas – už laiką mokėti atmintimi – informatikoje yra visuotinai žinomas.

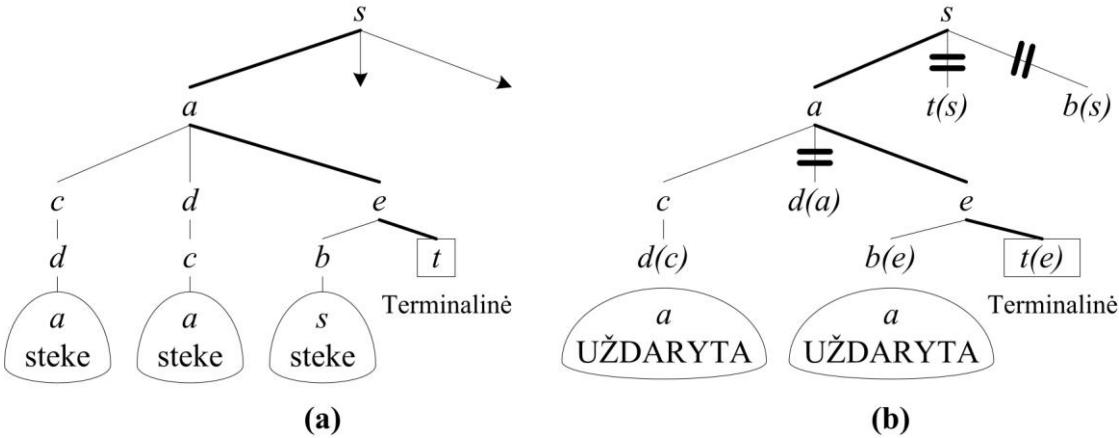
14.1. Paieška grafe

Pradékime palyginimą nuo paieškos grafe. Algoritmų skirtumus išnagrinėkime grafe, kuris buvo pateiktas 11.1 a pav. Tegu ir produkcijų sutvarkymas tas pats kaip 11.1 b pav.



14.1 pav. a) Grafas, kurio briaunos neturi kainos. b) Produkcijų sutvarkymas „prieš laikrodžio rodykle“ – „dvylitka valanda“ imama paskutinė

Procedūros BACKTRACK1 paieškos medis parodytas 14.2 a pav.:



14.2 pav. Paieškos medžiai: a) BACKTRACK1,
b) GRAPHSEARCH_L_GYL. Randamas tas pats keliai $\langle s,a,e,t \rangle$

Paaškinsime paieškos medžių 14.2 a pav. Agentas stovėdamas viršūnėje s mato tris produkcijas – lyg tris tunelius. Įveskime veiksmų „kainius“. Tegu pasižiūrėjimas į produkciją kainuoja 1 Lt, o produkcijos pritaikymas, t. y. GDB pervedimas į kitą būseną – 100 Lt. Agentui iš viršūnės s pasiekti a kainuoja 103 Lt: 3 Lt už pasižiūrėjimą į tris produkcijas ir 100 Lt už perėjimą pagal pirmają produkciją.

Procedūros GRAPHSEARCH_L_GYL paieškos medis (14.2 b pav.) yra mažesnis negu BACKTRACK1 (14.1 a pav.). Priežastis: agentas nežengia į pomedį žemyn nuo viršūnės d . Atkreipiame dėmesį į skirtumus tarp šių paieškos medžių. Pirma, GRAPHSEARCH_L_GYL atidaro viršunes t ir b , t. y. patalpina jas į sąrašą ATIDARYTA. Tuo tarpu BACKTRACK_SU_STEKU „mato“ tik produkcijas, o viršunių t ir b „tunelio gale“ nemato. Antra, BACKTRACK1 ciklą apeina iš vienos pusės, t. y. $\langle a,c,d \rangle$, o po to iš kitos pusės – $\langle a,d,c \rangle$. Tuo tarpu GRAPHSEARCH_L_GYL ciklą apeina tik iš vienos pusės – $\langle a,c,d \rangle$.

Atidarytų ir uždarytų viršunių frontas GRAPHSEARCH_L_GYL baigus darbą yra tokis:

$s(start) \{a(s) \ t(s) \ b(s)\} \ {e(a) \ d(a) \ e(a)} \ d(e) \ {b(e) \ t(e)}$

Čia viršūnės iš sąrašo UŽDARYTA yra pavaizduotos perbrauktos, o viršūnės, kurių nuorodas procedūra keitė, yra pabrauktos vingiuotai.

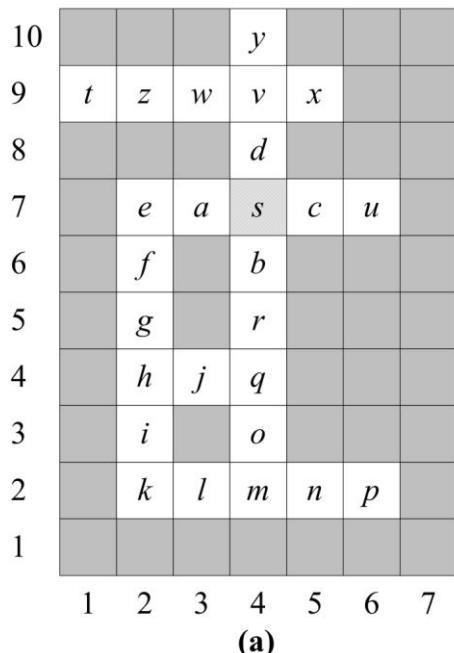
Procedūros vykdymo metu tris kartus ankstesnė nuoroda buvo keista vėlesne:

Situacija	Atidaryta anksčiau	Atidaroma vėliau. Todėl ji pasirenkama
1	$d(a)$	$d(c)$
2	$b(s)$	$b(e)$
3	$t(s)$	$t(e)$

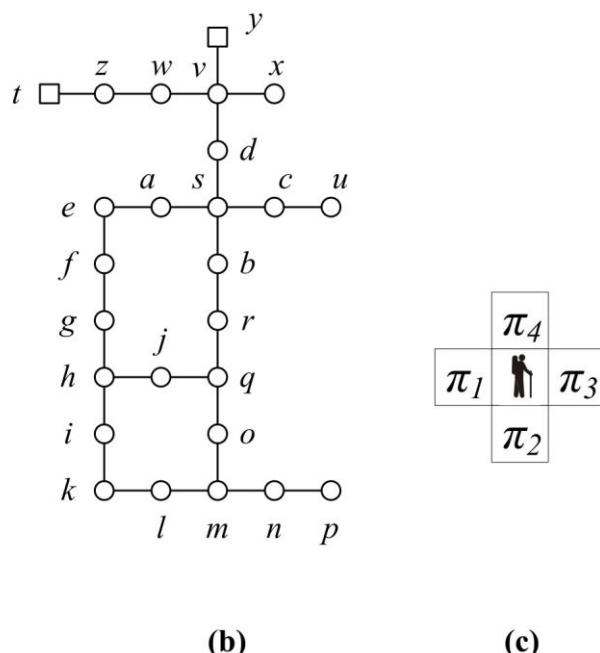
Abi procedūros randa tą patį kelią $\langle s,a,e,t \rangle$. BACKTRACK1 naudoja daugiau laiko. Jos paieškos medyje yra daugiau briaunu: 11 talpinimų į steką, iš jų 3 žingsniai patikrinti, kad talpinama viršūnė jau yra steke. GRAPHSEARCH_L_GYL – naudoja daugiau atminties. Pastarosios procedūros paieškos medyje yra 10 viršunių. Tuo tarpu BACKTRACK1 steko gylis yra 5.

14.2. Pavyzdys paieškai labirinte su salomis

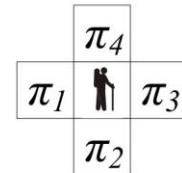
Toliau pateikiamas sudėtingesnis pavyzdys. Labirintas gali būti suprantamas kaip grafas. Pastarojo viršūnė atitinka labirinto langelį, o briauna tarp viršūnių dedama, kai langeliai yra gretimi. Paieška demonstruojama kelio radimu nuo pradinės viršūnės s iki krašto labirinte pateiktame 14.3 a pav. Išėjimų iš labirinto – terminalinių viršūnių – yra dvi: t ir y . Remiantis produkcijų sutvarkymu parodytu 14.3 c pav. randamas išėjimas per t . Keturios briaunas produkcijos yra surūšiuotos šia tvarka: „i vakarus“, „i pietus“, „i rytus“ ir „i šiaure“.



(a)



(b)

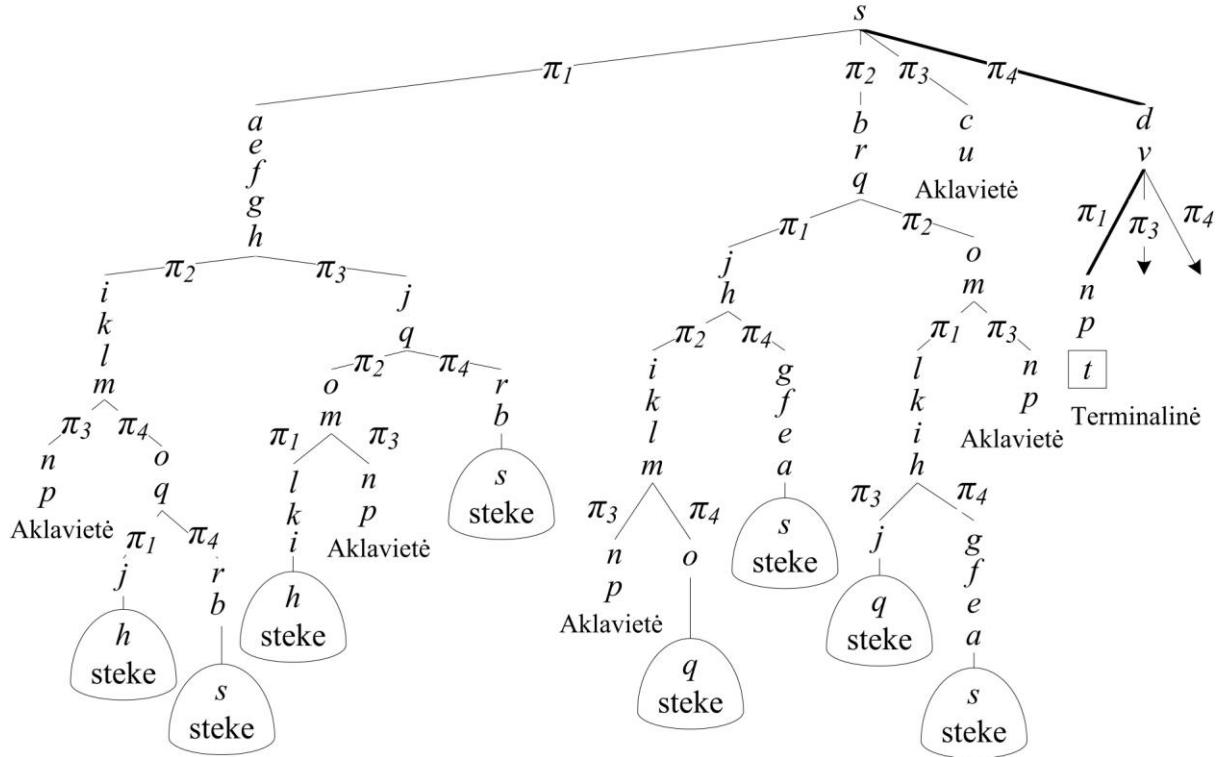


(c)

- 14.3 pav. a) Labirintas, kuriame ieškomas kelias iš s . Yra du išėjimai – t ir y .
 b) labirinto pavaizdavimas grafu. Pastarojo viršūnė atitinka langelį, o briauna – langelių gretimumo santykį. c) briaunų sutvarkymas

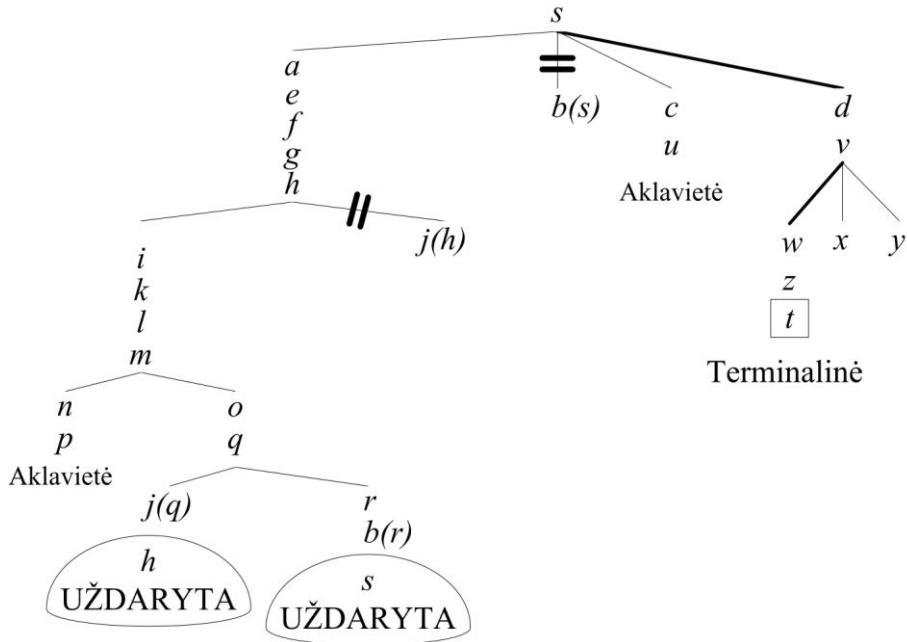
BACKTRACK1 paieškos medis algoritmui baigus darbą yra pateiktas 14.4 pav. Šis paieškos medis yra neišreikštinis. Stekui „susitraukiant“, kelias yra „užmirštamas“. Abu algoritmai nagrinėjamame grafe 14.3 a pav. randa tą patį kelią $\langle s,d,v,w,z,t \rangle$.

GRAPHSEARCH_L_GYLĮ paieškos medis algoritmui baigus darbą yra pateiktas 14.5 pav. Pomedžiai žemyn nuo $j(h)$ ir $b(s)$ nėra nagrinėjami, nes į sąrašą ATIDARYTA vėliau (t. y. pagal principą „i gylį“) patalpinamos atitinkamai viršūnės $j(q)$ ir $b(r)$. Šiu pomedžių nukirtimas ir sudaro esminį GRAPHSEARCH_L_GYLĮ skirtumą nuo BACKTRACK1. GRAPHSEARCH paieškos medyje yra mažiau viršūnių, tačiau atmintyje saugomi sąrašai ATIDARYTA ir UŽDARYTA.



14.4 pav. BACKTRACK1 paieškos medis – neišreikštinis. Kelias $\langle s, d, v, w, z, t \rangle$

Skirtingai nuo BACKTRACK1 neišreikštinio paieškos medžio, GRAPHSEARCH jis yra išreikštinis. Jis braižomas pagal į frontą talpinamų viršūnių nuorodas.



14.5 pav. GRAPHSEARCH_L_GYL paieškos medis. Jis išreikštinis

Toliau lentelėje pateikiamos GRAPHSEARCH_L_GYL atidarytų ir uždarytų viršūnių sąrašų būsenos algoritmo iteracijose.

	Viršūnių frontas: ATIDARYTA ir UŽDARYTA kartu	Komentaras
1	$s(start)$	Atidaroma pradinė viršūnė s
2	$s(start) \{a(s) b(s) c(s) d(s)\}$	Uždaroma $s(start)$ ir atidaromi – talpinami į fronto pabaigą pagal produkcijos numerį – keturi jos vaikai $S(s) = \{a, b, c, d\}$
3	$s(start) \{a(s) b(s) c(s) d(s)\} e(a)$	Uždaroma $a(s)$ ir imamos jos incidentinės $S(a) = \{e, s\}$. I fronto pabaigą talpinama $e(a)$. s jau yra fronte uždaryta ir toliau nenagrinėjama. (Toliau uždarytų viršūnių neminėsime – dėl lakoniškumo)
4	$s(start) \{a(s) b(s) c(s) d(s)\} e(a) f(e)$	Uždaroma $e(a)$ – „i gylį“ – ir atidaromas jos vaikas $f \in S(e)$
5	$s(start) \{a(s) b(s) c(s) d(s)\} e(a) f(e) g(f)$	Uždaroma $f(e)$ ir atidaromas $g \in S(f)$
6	$s(start) \{a(s) b(s) c(s) d(s)\} e(a) f(e) g(f) h(g)$	Uždaroma $g(f)$ ir atidaromas $h \in S(g)$
7	$s(start) \{a(s) b(s) c(s) d(s)\} e(a) f(e) g(f) h(g) \{i(h) j(h)\}$	Uždaroma $h(g)$ ir atidaromi $i, j \in S(h)$
8	$s(start) \{a(s) b(s) c(s) d(s)\} e(a) f(e) g(f) h(g) \{i(h) j(h)\} k(i)$	Uždaroma $i(h)$ ir atidaromas jos vaikas $k \in S(i)$
9	$s(start) \{a(s) b(s) c(s) d(s)\} e(a) f(e) g(f) h(g) \{i(h) j(h)\} k(i) l(k)$	Uždaroma $k(i)$ ir atidaromas jos vaikas $l \in S(k)$
10	$s(start) \{a(s) b(s) c(s) d(s)\} e(a) f(e) g(f) h(g) \{i(h) j(h)\} k(i) l(k) m(l)$	Uždaroma $l(k)$ ir atidaromas jos vaikas $m \in S(l)$
11	$s(start) \{a(s) b(s) c(s) d(s)\} e(a) f(e) g(f) h(g) \{i(h) j(h)\} k(i) l(k) m(l) \{n(m) o(m)\}$	Uždaroma $m(l)$ ir atidaromi jos vaikai $n, o \in S(m)$
12	$s(start) \{a(s) b(s) c(s) d(s)\} e(a) f(e) g(f) h(g) \{i(h) j(h)\} k(i) l(k) m(l) \{n(m) o(m)\} p(n)$	Uždaroma $n(m)$ ir atidaromas jos vaikas $p \in S(n)$
13	$s(start) \{a(s) b(s) c(s) d(s)\} e(a) f(e) g(f) h(g) \{i(h) j(h)\} k(i) l(k) m(l) \{n(m) o(m)\} p(n)$	Uždaroma $p(n)$ ir, kadangi ji aklavietėje, niekas neatidaroma. Kitame žingsnyje bus uždaroma sekanti atidaryta viršūnė
14	$s(start) \{a(s) b(s) c(s) d(s)\} e(a) f(e) g(f) h(g) \{i(h) j(h)\} k(i) l(k) m(l) \{n(m) o(m)\} p(n) q(o)$	Uždaroma $o(m)$ ir atidaromas jos vaikas $q \in S(o)$
15	$s(start) \{a(s) b(s) c(s) d(s)\} e(a) f(e) g(f) h(g) \{i(h) j(h)\} k(i) l(k) m(l) \{n(m) o(m)\} p(n) q(o) \{j(q) r(q)\}$	Uždaroma $q(o)$ ir atidaromi jos vaikai $j(q), r \in S(q)$. Reikia pastebeti, kad j jau yra ATIDARYTA kaip $j(h)$. Pagal principą „i gylį“ yra perorientuojama j nuoroda: pasirenkama vėliau atidaryta $j(q)$
16	$s(start) \{a(s) b(s) c(s) d(s)\} e(a) f(e) g(f) h(g) \{i(h) j(h)\} k(i) l(k) m(l) \{n(m) o(m)\} p(n) q(o) \{j(q) r(q)\}$	Uždaroma $j(q)$. Naujų vaikų ji neturi.
17	$s(start) \{a(s) b(s) c(s) d(s)\} e(a) f(e) g(f) h(g) \{i(h) j(h)\} k(i) l(k) m(l) \{n(m) o(m)\} p(n) q(o) \{j(q) r(q)\} b(r)$	Uždaroma $r(b)$ ir atidaromas jos vaikas $b(r) \in S(r)$. Reikia pastebeti, kad b jau yra ATIDARYTA kaip $b(s)$. Pagal principą „i gylį“ yra perorientuojama j nuoroda: pasirenkama vėliau atidaryta $b(r)$

18	$s(start) \{a(s) b(s) c(s) d(s)\} e(a) f(e) g(f)$ $h(g) \{i(h) j(h)\} k(i) l(k) m(l) \{n(m) o(m)\}$ $p(n) q(o) \{j(q) r(q)\} b(r)$	Uždaroma $b(r)$. Naujų vaikų ji neturi.
19	$s(start) \{a(s) b(s) e(s) d(s)\} e(a) f(e) g(f)$ $h(g) \{i(h) j(h)\} k(i) l(k) m(l) \{n(m) o(m)\}$ $p(n) q(o) \{j(q) r(q)\} b(r) u(c)$	Toliau uždaroma $c(s)$ ir atidaromas jos vaikas $u \in S(c)$
20	$s(start) \{a(s) b(s) e(s) d(s)\} e(a) f(e) g(f)$ $h(g) \{i(h) j(h)\} k(i) l(k) m(l) \{n(m) o(m)\}$ $p(n) q(o) \{j(q) r(q)\} b(r) u(c)$	Uždaroma $u(c)$ ir, kadangi ji aklavietėje, niekas neatidaroma. Kitame žingsnyje bus uždaroma sekanti atidaryta viršūnė
21	$s(start) \{a(s) b(s) e(s) d(s)\} e(a) f(e) g(f)$ $h(g) \{i(h) j(h)\} k(i) l(k) m(l) \{n(m) o(m)\}$ $p(n) q(o) \{j(q) r(q)\} b(r) u(c) v(d)$	Toliau uždaroma $d(s)$ ir atidaromas jos vaikas $v \in S(d)$
22	$s(start) \{a(s) b(s) e(s) d(s)\} e(a) f(e) g(f)$ $h(g) \{i(h) j(h)\} k(i) l(k) m(l) \{n(m) o(m)\}$ $p(n) q(o) \{j(q) r(q)\} b(r) u(c) v(d) \{w(v)$ $x(v) y(v)\}$	Uždaroma $v(d)$ ir atidaromi jos vaikai $w, x, y \in S(v)$
23	$s(start) \{a(s) b(s) e(s) d(s)\} e(a) f(e) g(f)$ $h(g) \{i(h) j(h)\} k(i) l(k) m(l) \{n(m) o(m)\}$ $p(n) q(o) \{j(q) r(q)\} b(r) u(c) v(d) \{w(v)$ $x(v) y(v)\} z(w)$	Uždaroma $w(v)$ ir atidaromas jos vaikas $z \in S(w)$
24	$s(start) \{a(s) b(s) e(s) d(s)\} e(a) f(e) g(f)$ $h(g) \{i(h) j(h)\} k(i) l(k) m(l) \{n(m) o(m)\}$ $p(n) q(o) \{j(q) r(q)\} b(r) u(c) v(d) \{w(v)$ $x(v) y(v)\} z(w) t(z)$	Uždaroma $z(w)$ ir atidaromas jos vaikas $t \in S(z)$
25	$s(start) \{a(s) b(s) e(s) d(s)\} e(a) f(e) g(f)$ $h(g) \{i(h) j(h)\} k(i) l(k) m(l) \{n(m) o(m)\}$ $p(n) q(o) \{j(q) r(q)\} b(r) u(c) v(d) \{w(v)$ $x(v) y(v)\} z(w) t(z)$	Uždaroma $t(z)$ ir pastebima, kad ji yra terminalinė. Surenkamas kelias: t pasiekta iš z , z iš w , w iš v , v iš d , d iš s , kuri pradinė. Rastas kelias $\{s, d, v, w, z, t\}$. Algoritmas baigia darbą. Kelios viršūnės lieka atidarytos

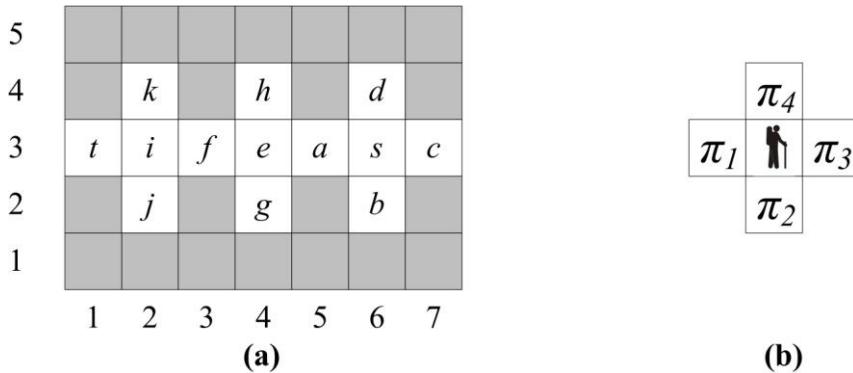
14.6 lentelė. Algoritmo GRAPHSEARCH_I_GYL iteracijos

Nuorodų keitimasis atliktas du kartus:

Situacija	Atidaryta anksčiau	Atidaroma vėliau. Todėl ji pasirenkama
1	$j(h)$	$j(q)$
2	$b(s)$	$b(r)$

14.3. Kontrpavyzdys – labirintas, kuriame geresnį rezultatą duoda BACKTRACK1

Ankstesčiai pavyzdžiai buvo pademonstruotas skirtumas tarp algoritmų BACKTRACK1 ir GRAPHSEARCH_L_GYL_I. Pastarasis pranašesnis taip pat prasme, kad jo paieškos medis (ir tokiu būdu vykdymo laikas) yra mažesnis. Tačiau tai pasiekiamai atminties kaina. Toliau 14.7 pav. pateikiamas kontrpavyzdys – grafas, kuriame BACKTRACK1 yra pranašesnis.



14.7 pav. a) Labirintas, kuriame ieškomas kelias nuo langelio s iki krašto. Yra du išėjimai – terminaliniai langeliai t ir c . b) Produkcių tvarka

BACKTRACK1 randa išėjimą „tiesiu taikymu“, t. y. be perrinkimo: $\langle s, a, e, f, i, t \rangle$. Algoritmu įvertinimui įveskime kainos kriterijus.

Tegu BACKTRACK_SU_STEKU „ikainiai“ yra tokie:

- 1 – už **pasižiūrėjimą**, t. y. produkcijos patalpinimą į sąrašą RULES, naudojant funkciją APPRULES (žr. 2 skyrių);
- 100 – už **patalpinimą** į steką. Tieka moka agentas už „grūdo“ padėjimą langelyje.

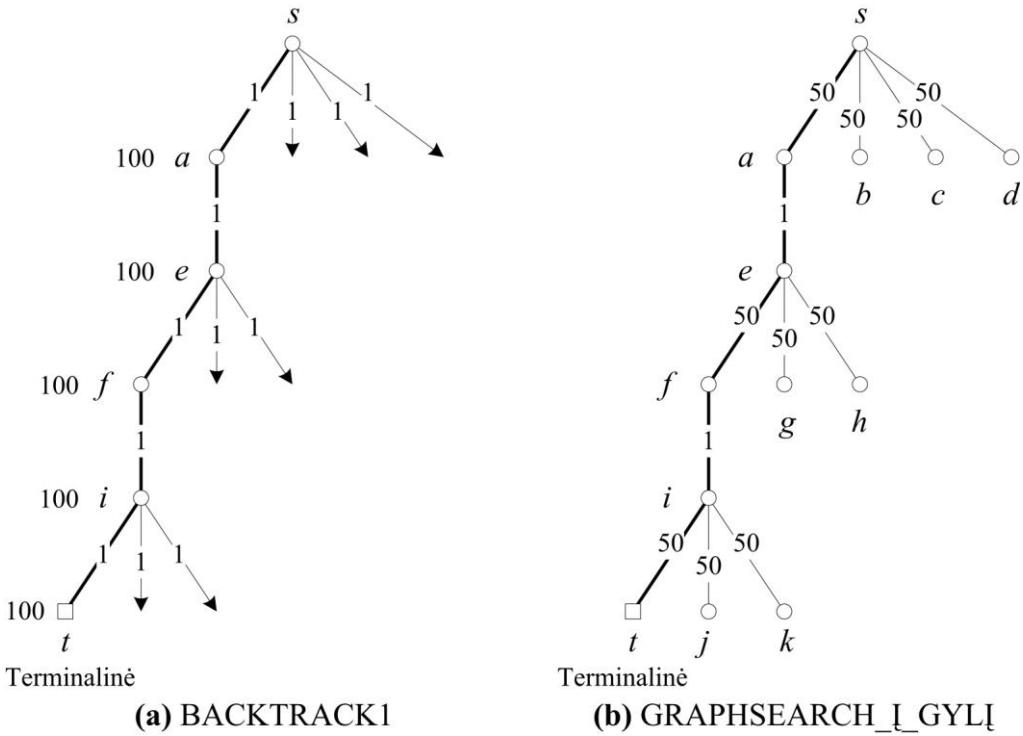
Tegu GRAPHSEARCH_L_GYL_I „ikainis“ yra toks:

- 50 – už viršūnės **atidarymą**, t. y. viršūnės patalpinimą į sąrašą ATIDARYTA.

Abu algoritmai randa tą patį kelią. Bet šiame grafe BACKTRACK1 kaina yra geresnė negu GRAPHSEARCH_L_GYL_I. BACKTRACK1 paieškos medžio kaina yra $5 \cdot 101 + 3 \cdot 1 + 2 \cdot 1 + 2 \cdot 1 = 512$ (14.8 a pav.), o GRAPHSEARCH_L_GYL_I – $12 \cdot 50 = 600$ (14.8 b pav.).

Šiame pavyzdyste procedūrai GRAPHSEARCH_L_GYL_I, kurios ikainis dvigubai mažesnis, kelio paieška yra brangesnė. Taip atsitinka todėl, kad BACKTRACK1 pirmiausia tik pasižiūri, kur galima eiti ($n \cdot 1$), paskui pasirenka pirmajį kelią, ir tik tada „žengia“, t. y. padeda „grūdą“ ($1 \cdot 100$). GRAPHSEARCH_L_GYL_I iš karto atidaro n viršūnių ($n \cdot 50$). Pasižiūrėjimas yra pigus, bet „grūdas“ yra brangus; jo kaina daug didesnė už pasižiūrėjimą.

Šiuo pavyzdžiu pademonstravome, kad GRAPHSEARCH_L_GYL_I ne visais atvejais yra pranašesnis už BACKTRACK1. Algoritmus metaforiskai palyginkime kaip automobilį ir riedučius. Laikykime, kad BACKTRACK1 yra riedučiai, o GRAPHSEARCH_L_GYL_I – automobilis. Tolimam keliui automobilis yra pranašesnis. Tačiau kaimyną gretimame kieme greičiau pasiekime riedučiais. Juk automobilį reikia užvesti, išvažiuoti iš kiemo, nuvažiuoti keliu iki kaimyno kiemo ir rasti vietą parkavimui.



14.8 pav. a) BACKTRACK1 paieškos medis; kaina $5 \cdot 100 + 12 \cdot 1 = 512$. b) GRAPHSEARCH_L_GYL; kaina $12 \cdot 50 = 600$ yra blogesnė

Pratimai

1. Pažymėkite produkcijas paieškos medžiuose, pateiktuose 14.2 a pav. ir 14.4 pav.
2. Kuris algoritmas – GRAPHSEARCH_L_GYL ar BACKTRACK1 – duoda geresnį rezultataj, paieškai kelio iš viršunės *s* iki krašto labirinte, pateiktame 14.9 pav?

7		<i>u</i>					<i>x</i>	
6		<i>t</i>	<i>r</i>	<i>p</i>			<i>q</i>	
5				<i>o</i>	<i>k</i>	<i>l</i>	<i>m</i>	<i>n</i>
4		<i>a</i>	<i>s</i>	<i>b</i>	<i>j</i>			
3		<i>c</i>		<i>v</i>	<i>i</i>			
2		<i>d</i>	<i>e</i>	<i>f</i>	<i>g</i>	<i>h</i>		
1						<i>w</i>		
	1	2	3	4	5	6	7	8
								9

14.9 pav. Labirintas, kuriame ieškomas kelias nuo langelio *s* iki krašto. Labirinte yra 4 išėjimai – *u*, *x*, *n* ir *w*

Nubraižykite paieškos medžius su kainomis: pažiūrėjimas – 1, „grūdo“ padėjimas – 100, viršunės atidarymas – 50.

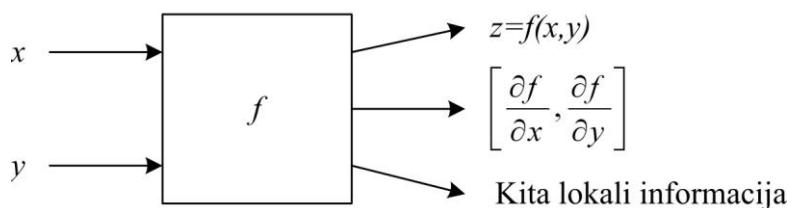
15. Valdymo metodas „kopimas į kalną“

„Kopimo į kalną“ metodas yra paimtas iš matematikos. Dirbtiniame intelekte jis nėra visuotinai priimtas dėl šių priežasčių: 1) lokalaus maksimumo problema ir 2) plato.

„Kopimo į kalną“ metodo esmė yra tokia. Iš *lokaliros informacijos* yra daroma prielaida apie *globalią informaciją*, t.y. apie sprendinį.

Norint rasti tašką, kuriame funkcija $f(x)$, apibrežta intervale $[a,b]$, įgyja maksimumą, nuo pradinio duoto taško x_0 reikia judeti funkcijos didėjimo kryptimi. Tą kryptį rodo funkcijos išvestinė. Renkamasi iš dviejų krypčių: x didėjimo arba mažėjimo kryptimi.

Skaičiavimo matematikoje yra žinomas „gradientinis metodas“. Kai reali funkcija priklauso nuo keleto argumentų, pvz., $f(x,y)$, tai judama didžiausio „statumo“ – gradiento – kryptimi. Funkcijos $f(x,y)$ maksimizavimo uždavinį galima suformuluoti šitokia metafora. Parašiutininkas (sprendžiantis, o ne planuojantis agentas) naktį nuleidžiamas ant kalno šlaito, t.y. taške $[x_0, y_0]$. Jo tikslas užlipti į viršūnę, t.y. rasti tašką $[x_{max}, y_{max}]$, kuriame funkcija $f(x,y)$ įgyja absolютų maksimumą, kur $x \in [a_1, b_1]$ o $y \in [a_2, b_2]$. Agentas kopia didžiausio gradiento kryptimi. Yra naktis ir agentas nemato viso kalnų peizažo, t.y. jam neprieinama globali informacija apie kalnus – funkciją $f(x,y)$. Agentas vadovaujasi tik lokalia informacija – gradientu (žr. 15.1 pav.).



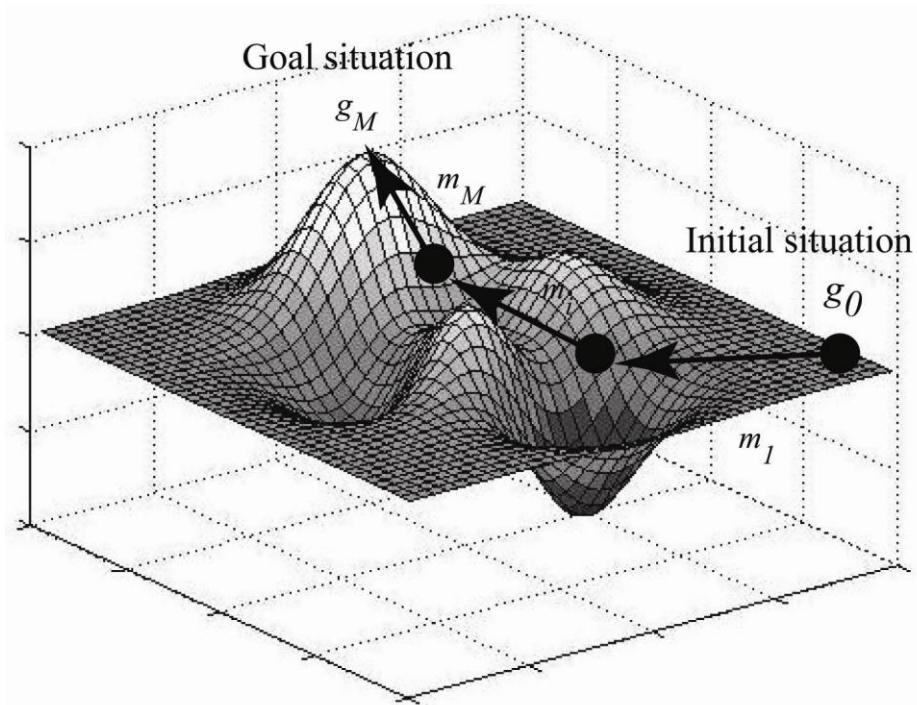
15.1 pav. Sprendžiantis agentas vadovaujasi lokalia informacija apie funkcijos $f(x,y)$ reikšmę tame taške, kur stovi. Priklasomai nuo agento intelektualumo be funkcijos reikšmės jis sužino didžiausio gradiento kryptį ir kitą lokalią informaciją, pvz., antrają išvestinę

Gradiento vektorius pasirenkamas kaip vedantis stačiausia kryptimi aukštyn. Toks vektorius yra vienas iš 360° agento horizonto, kuris suprantamas kaip liečiamoji plokštuma tame funkcijos $f(x,y)$ taške. Tokiu būdu agentas turi pasirinkti, kuriuo azimutu nuo 0° iki 360° eiti. Pavyzdys pateiktas 15.2 pav.

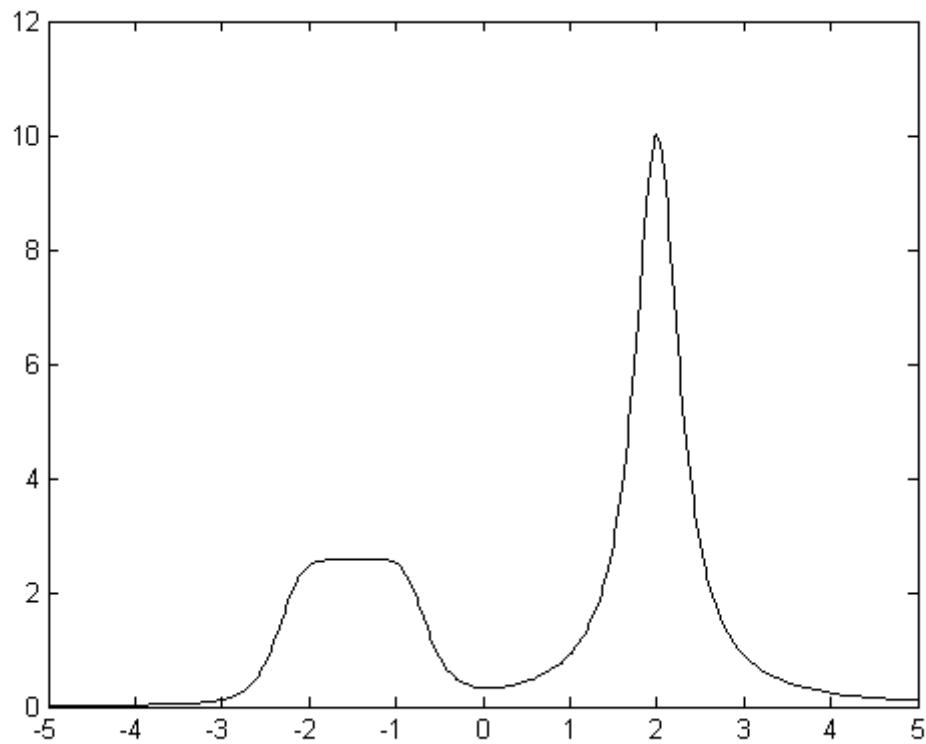
Darydamas žingsnius $\langle [x_0, y_0], [x_1, y_1], [x_2, y_2], \dots \rangle$ didžiausio gradiento kryptimi agentas gali pasiekti *lokalų* maksimumą, t.y. užkopti į žemesnę viršūnę, negu absoliutaus maksimumo. Tokiu būdu gradientinis metodas garantuoja absoliutaus maksimumo pasiekimą tik „gražioms“ funkcijoms – kurios turi tik vieną maksimumą.

Be lokalaus maksimumo, agentas susiduria su kitu pavojumi – plato. Atsidūrės ant plokštikalnės agentas nežino kuriuo azimutu eiti.

Dvimatėje erdvėje, t.y. $z=f(x)$ atveju, agento pasirinkimas daug paprastesnis. Jis renkasi tarp dviejų krypčių – x didėjimo ar mažėjimo. Pavyzdys pateiktas 15.3 pav.



15.2 pav. Kopimas į kalną $\langle [x_0, y_0], [x_1, y_1], [x_2, y_2], \dots \rangle$ didžiausio gradiento kryptimi kaip funkcijos $z=f(x,y)$ maksimizavimas. Agentas nemato reljefo lyg iš paukščio skrydžio



15.3 pav. Kopimas į kalną $\langle x_0, x_1, x_2, \dots \rangle$ teigiamos išvestinės kryptimi dvimatėje erdvėje kaip funkcijos $z=f(x)$ maksimizavimas

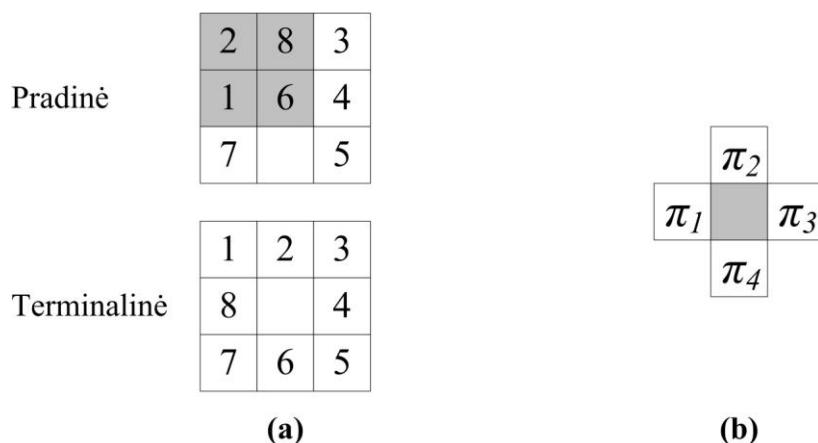
Gradientiniai metodai paprastai yra taikomi „gražioms“ funkcijoms. Pavyzdžiai reikalavimų „gražioms“ funkcijoms griežtėjimo tvarka yra šie:

1. funkcija tolydi;
2. funkcija turi išvestinę;
3. funkcijos išvestinė yra tolydi intervale $[a,b]$;
4. funkcijos išvestinė turi vienodą ženklinę (t. y. arba neteigiamą, arba neneigiamą) intervale $[a,b]$;
5. kitos sąlygos.

Tokiomis geromis savybėmis pasižymi funkcijos, konstruojamos iš elementarių funkcijų tokiu kaip polinomas, eksponentė, trigonometrinės funkcijos ir kt.

Dirbtinio intelekto produkcijų sistemoje yra išskiriamos dvi pagrindinės valdymo strategijos: negrižtamoji (angl. *irrevocable*) ir grįžtamoji (angl. *tentative*). Taikant negrižtamają strategiją, produkcija yra pasirenkama ir pritaikoma negrižtamai, t. y. be galimybės grįžti į ankstesnę būseną. „Kopimas į kalną“ yra viena iš negrižtamųjų strategijų. Ir priešingai, „valdymas su grįžimais“ BACKTRACK yra grįžtamosios strategijos pavyzdys.

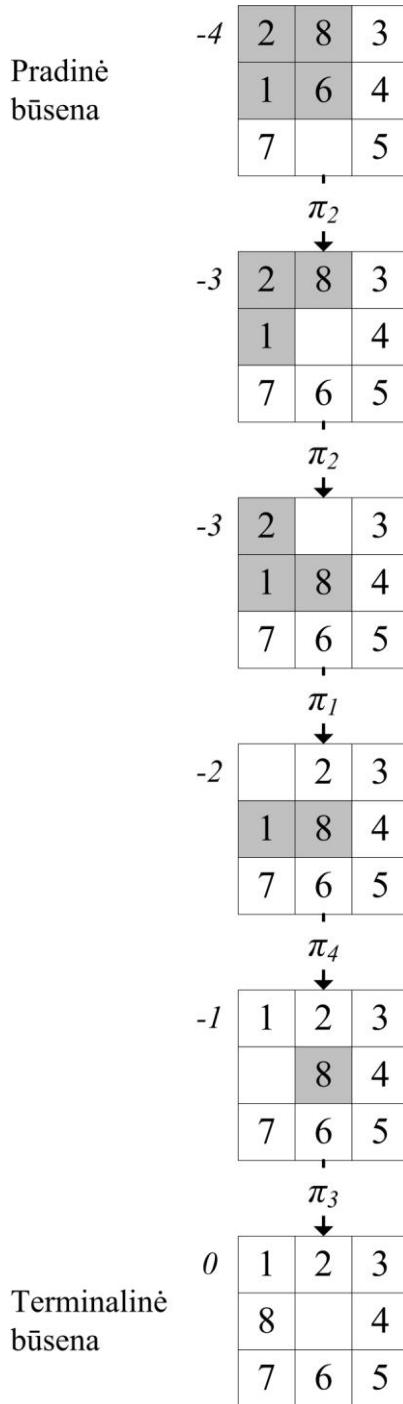
„Kopimo į kalną“ metodą iliustruojame paieška 8 kauliukų galvosūkyje (angl. *8-puzzle*). Apibrėžkime funkciją $f(n)$ nuo globalios duomenų bazės būsenos n tokiu būdu. Tegu $f(n) = -w(n)$, kur $w(n)$ yra skaičius kauliukų, kurie ne savo padėtyje lyginant su terminaline būseną. Pavyzdžiu, funkcijos reikšmė pradinėje padėtyje (15.4 pav.) yra -4 . Funkcijos reikšmė terminalinėje būsenoje yra 0 , nes visi kauliukai stovi savo vietose.



15.4 pav. a) Pradinė ir terminalinė būsenos 8 kauliukų galvosūkyje. b) Keturi tuščio lanelio ėjimai – tai produkcijos $\langle \pi_1, \pi_2, \pi_3, \pi_4 \rangle$

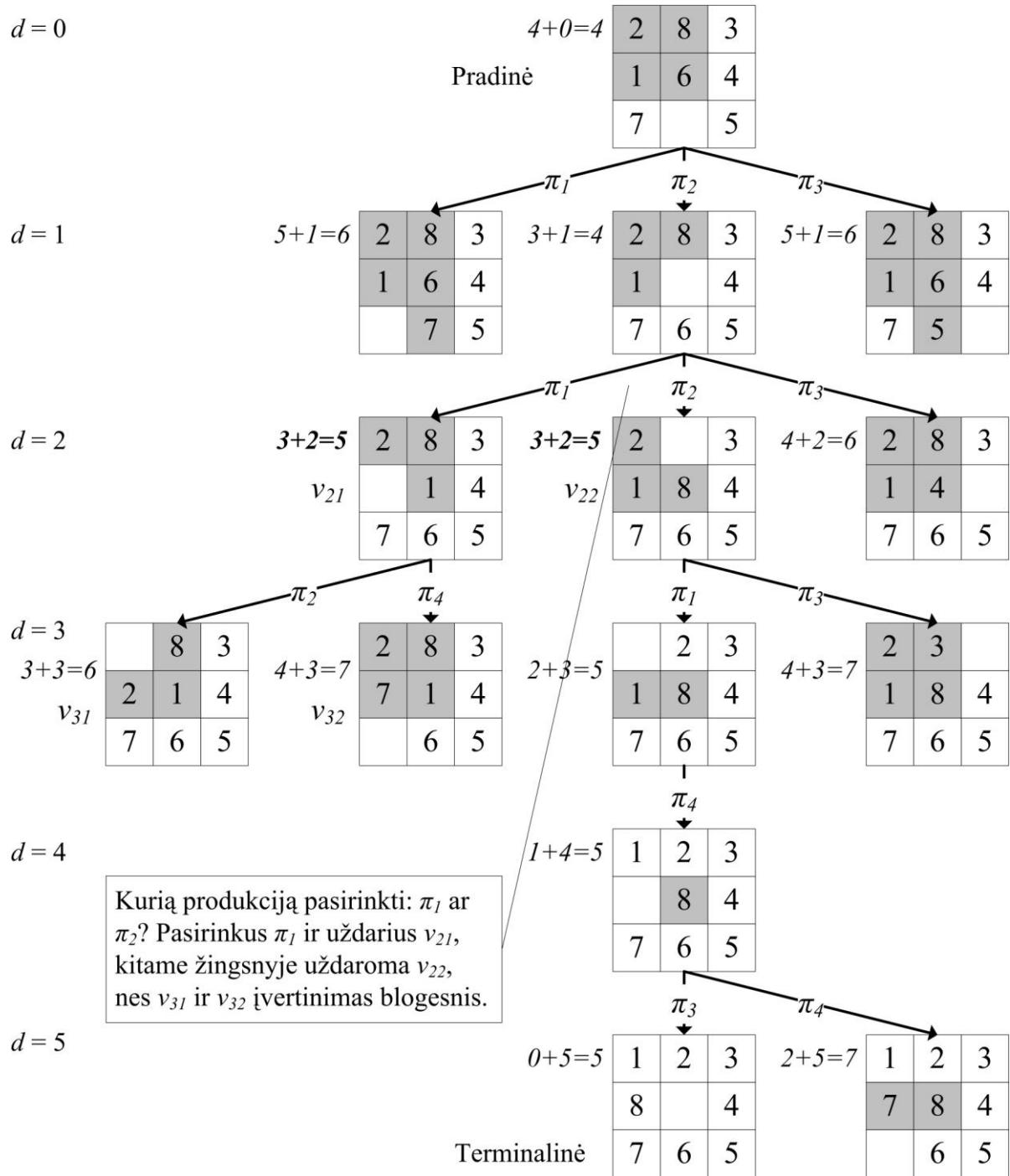
Pradinėje būsenoje funkcijos reikšmė yra blogiausia. Pereinant į kitas būsenas, siekiama padidinti f reikšmę, t. y. ją maksimizuoti ir tokiu būdu pasiekti terminalinę būseną $f(n) = 0$. Ėjimas atliekamas pastumiant kurį nors kauliuką į tuščią lanelį. Toks kauliuko pastūmimas yra traktuojamas kaip tuščio lanelio stūmimas priešinga kryptimi ir suprantamas kaip produkcija. Susitarkime, kad iš lanelio keturios produkcijos $\langle \pi_1, \pi_2, \pi_3, \pi_4 \rangle$ yra surūšiuotos šia tvarka: „i vakarus“, „i šiaurę“, „i rytus“ ir „i pietus“ (žr. 15.4 b pav.).

15.5 pav. vaizduoja f kilimą iš pradinės būsenos iki terminalinės kurioje $f(n) = 0$. Funkcijos f reikšmė yra nurodyta prie kiekvienos būsenos.



15.5 pav. Funkcijos $f(n) = -w(n)$, kur $w(n)$ yra skaičius kauliukų ne savo vietoje, maksimizavimas 8 kauliukų žaidime

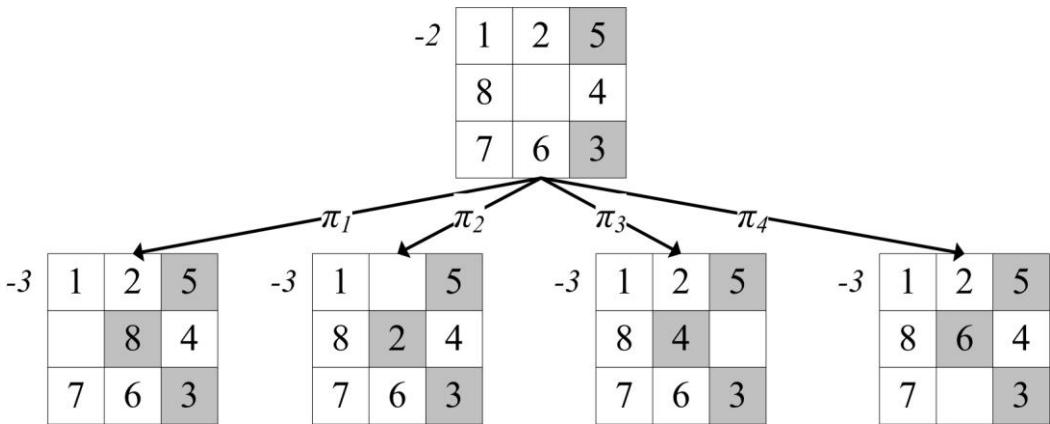
15.6 pav. vaizduoja paieškos medį, kai taikomas „kopimo į kalną” metodas ir pasirenkama funkcija $f(n) = w(n) + d(n)$, kur $d(n)$ – būsenos gylis. Ši f minimizuojama. Kuo būsena giliau, tuo ji blogesnė. Paieškos medis ir frontas su uždaromomis viršūnėmis pagrindžia antrosios produkcijos pasirinkimą iš $\{\pi_1, \pi_2\}$ gylyje $d = 2$.



15.6 pav. Funkcija $f(n) = w(n) + d(n)$, kur $w(n)$ yra kauliukų skaičius ne savo vietoje, o $d(n)$ – būsenos gylis. Funkcija f yra minimizuojama.

Taigi šiame pavyzdyste „kopimo į kalną“ metodas yra sėkmingas. Jis atveda į terminalinę būseną. Tačiau bendru atveju „kopimo į kalną“ funkcija gali pasiekti lokalujį maksimumą. Tada kitame žingsnyje neaišku kur eiti, nes visur blogiau.

Lokalaus maksimumo situacija pateikta 15.7 pav. Kiekviena produkcija perveda į būseną, kurioje funkcijos reikšmė yra blogesnė negu pradinėje.



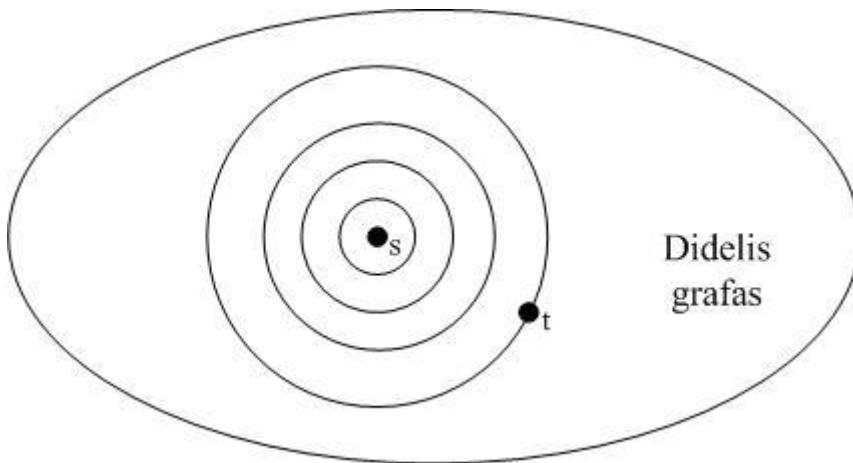
15.7 pav. Pradinė būsena yra lokaliame maksimume. Kiekviena produkcija perveda į būseną, kurioje funkcijos reikšmė yra blogesnė

Funkcijos f esmė yra tokia: tai lokali informacija, pagal kurią sprendžiama apie sprendinio globalias savybes. Tokios funkcijos atradimas gali būti suprantamas kaip intelektuali veikla. Funkcija f yra euristinė funkcija.

Funkcija $f(n) = w(n) + d(n)$, kur $d(n)$ yra būsenos gylis paieškos medyje, yra pateikiama [Nilsson 1982] 2.4.1 poskyryje, kaip pavyzdys euristinės funkcijos, skirtos būsenos išvertinimui procedūroje GRAPHSEARCH. Tenka pastebėti, kad paėmus $f(n)=d(n)$ yra gaunama paieška į plotį.

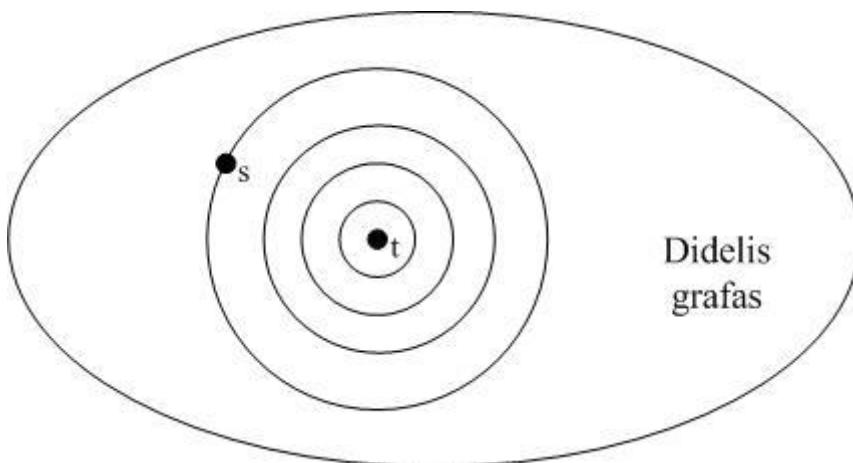
16. Manheteno atstumas

Tegu turime labai didelį grafą. Ieškosime kelio iš pradinės viršūnės s į terminalinę t . Ieškoti kelio galime paieškos į plotį būdu. Pirmaja banga atidarome viršūnes, esančias per vieną žingsnį nuo pradinės s . Antraja banga atidarome viršūnes, esančias per du žingsnius. Kai atidarome viršūnę t , tai reiškia suradome kelią; belieka jį surinkti.



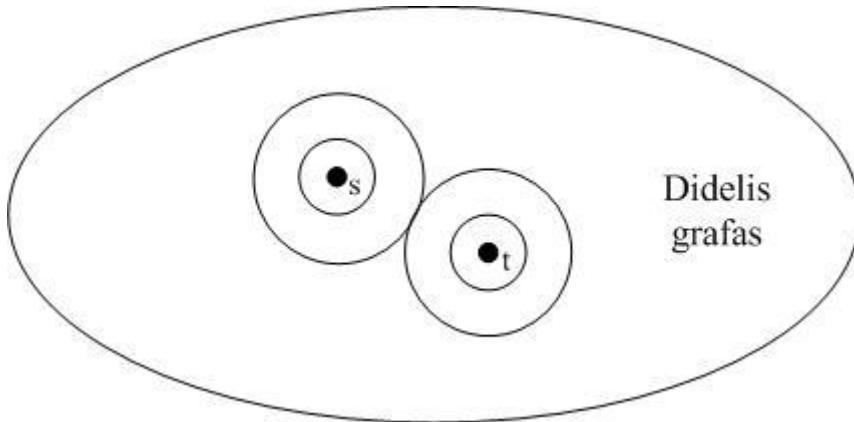
16.1 pav. Grafo pavyzdys kelio paieškai į plotį iš viršūnės s į t

Galime pradėti ir kita kryptimi, nuo viršūnės t . Kiekviena banga atidarome viršūnes esančias vis per daugiau žingsnių nuo t iki atidarome s ir tokiu būdu randame kelią.



16.2 pav. Grafo pavyzdys kelio paieškai į plotį iš s į t , pradedame nuo t

Paieškos rezultatai iškelia klausimą - kaip skleisti viršūnes geriau? Viršūnes reikia skleisti paraleliai – iš s ir iš t . Kada bangos „susitinka“, kelias rastas. Šitas būdas yra geresnis, tuo požiūriu, kad atskleidžiame mažiau viršūnių.



16.3 pav. Grafo pavyzdys kelio paieškai į plotį iš s ir t , pradedame nuo s ir t vienu metu

Pateikiamas teorinis įrodymas. Tegu kelio ilgis yra N briaunų. Jeigu skleidžiame bangą nuo s , viršunių skaičius bus $O(N^\alpha)$. Jeigu skleidžiame atgal nuo t iki s , įvertinimas yra tokis pat - $O(N^\alpha)$. Jeigu skleidžiame iš dviejų viršunių tai

$$O(N^\alpha) > O((N/2)^\alpha) + O((N/2)^\alpha)$$

Svarbiausia yra tai, kad pirmu būdu viršunių daugiau. Kai $\alpha = 2$, tai:

$$N^2 > (N/2)^2 + (N/2)^2 = N^2/4 + N^2/4 = N^2/2$$

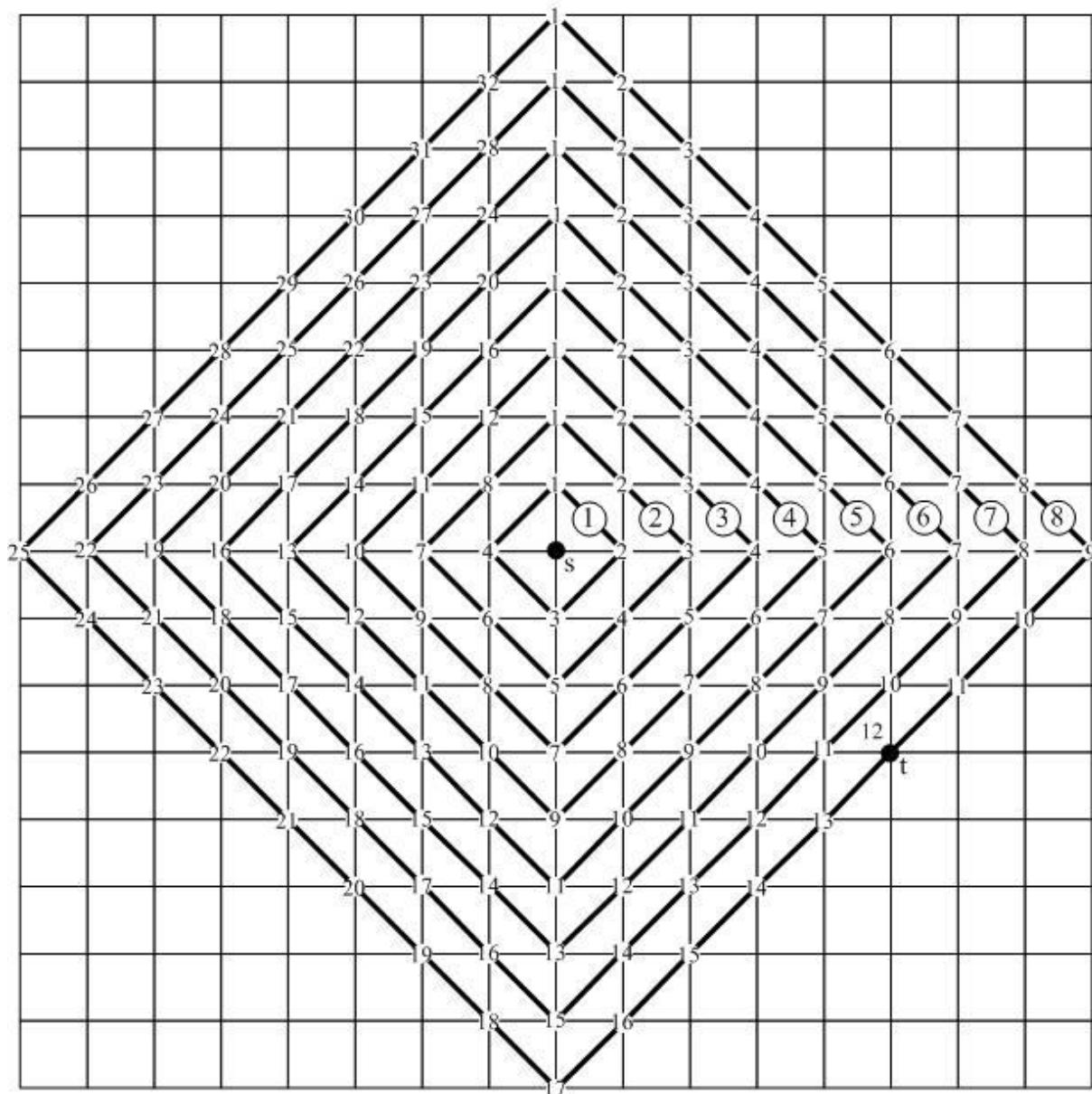
Kai $\alpha = 3$, tai

$$N^3 > (N/2)^3 + (N/2)^3 = N^3/8 + N^3/8 = N^3/4$$

Čia yra pateiktas įrodymo demonstravimas. Norint pateikti įrodymą bendru atveju, reikėtų daryti prielaidas apie grafą. Nenagrinėsime subtilių atvejų, pavyzdžiui kai $\alpha = 1,5$. Apsiribosime tik dvimate bei trimate erdvėmis, t. y. $\alpha \in \{2, 3\}$.

Idėjos esmė: viršunių atskleidimo požiūriu geriausia skleisti ir vieną ir antrą.

Turime grafą, kurio viršunes yra dvimatės gardelės (masyvo) viršūnės. Grafo viršūnės yra susikirtimai. Tokiu grafu galime pavaizduoti miesto gatvių tinklą. Briaunos vaizduotų gatves, o viršūnės – sankryžas. Ieškosime kelio tarp sankryžų s ir t . Pradedame nuo s . Pirmu žingsniu (pirmaja banga) pasiekiami 4 viršunes, antru žingsniu – 8 ir t. t. kol pasiekiami terminalinę viršūnę t . Grafas ir bangos yra pateikiamos 16.4 pav.:



16.4 pav. Manheteno gatvių tinklas. Kelio paieška iš s į t

Atidaromų grafo viršūnių skaičius kiekvienai bangai pateikiamas 16.5 lentelėje.

Bangos nr.	Viršūnių skaičius
0	$1 + 0$
1	4
2	8
3	12
4	16
5	20
6	24
7	28
8	32

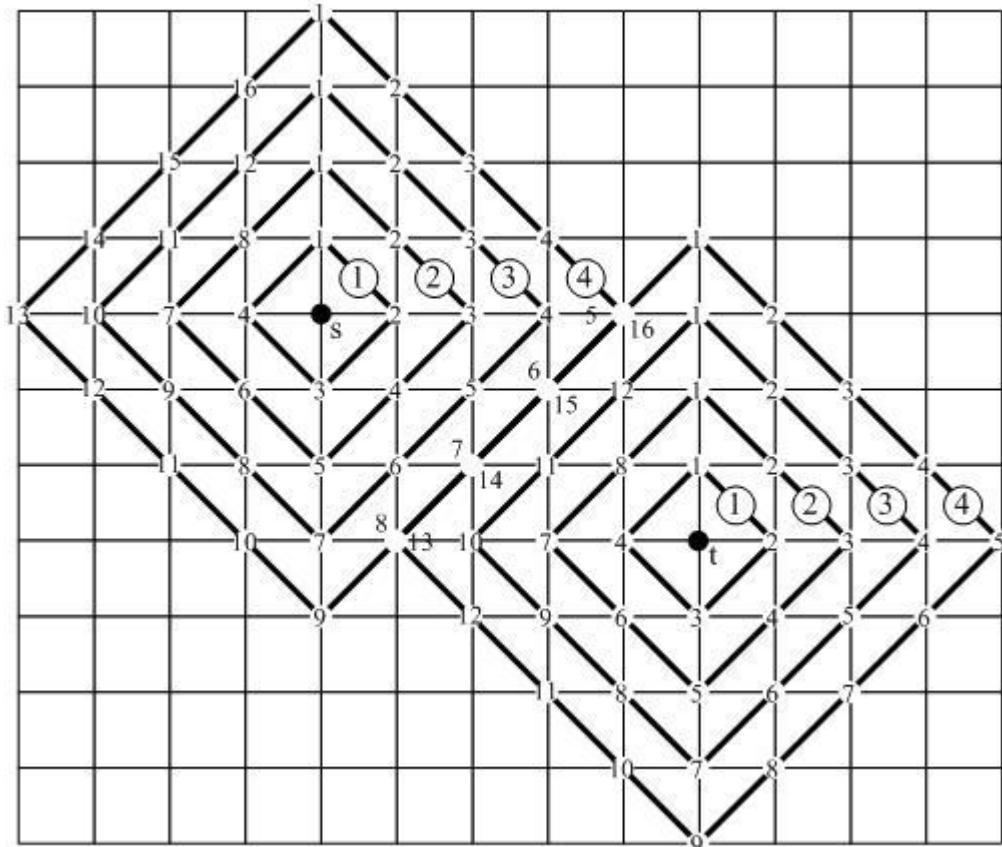
16.5 lentelė. Atidaromų viršūnių skaičius bangose 16.4 pav., iš viso 145

Kelio ilgis tarp s ir t yra $5 + 3 = 8$ briaunos, todėl, kaip beeitume, prireiks 8 bangų. Nesunkiai pastebime priklausomybę tarp bangos numerio ir atidaromų bangų viršūnių skaičiaus. Pasinaudodami ja galime išvesti formulę, pagal kurią galima greitai nustatyti, kiek iš viso bus atidaryta viršūnių, jei pasiekiame terminalinę viršūnę po N bangų.

$$0 + 4 + 8 + \dots + 4N = 4(1 + 2 + 3 + \dots + N) = 4(N(N+1)/2) = 2(N^2 + N) = O(N^2)$$

N -taja banga apriboto kvadrato plotas yra $2(N^2 + N) + 1$. Iš čia matome, kad 8 briaunų ilgio keliu rasti reikės atidaryti 145 viršūnes. Pradėjus nuo t gaunamas tas pats įvertinimas.

Dabar pradėkime paiešką dviem bangomis. Grafas ir bangos yra parodytos 16.6 pav.



16.6 pav. Paieška iš s į t Manheteno grafe, kai skleidžiamos abi bangos iš karto

Atidaromų grafo viršūnių skaičius kiekvienai bangai pateikiamas 16.7 lentelėje.

Bangos nr.	Viršūnių skaičius
0	$1 + 1 + 0 + 0$
1	$4 + 4$
2	$8 + 8$
3	$12 + 12$
4	$16 + 16$

16.7 lentelė. Atidaromų viršūnių skaičius bangose 16.6 pav., iš viso 82 viršūnės

Šitaip lygiagrečiai skleidžiant dvi bangas kelias randamas mažesniu atidaromų viršūnių skaičiumi. Pastebime, kad skleidžiamos bangos persidengia 4 viršūnėmis.

17. Algoritmo A* samprata

Algoritmas A* (tariama A su žvaigždute) yra GRAPSHEARCH algoritmas, kai naudojama tokia viršūnės n įvertinimo funkcija:

$$f(n) = g(n) + h(n)$$

kur $g(n)$ yra trumpiausias kelias nuo pradinės viršūnės iki n , o $h(n)$ yra euristinė funkcija įvertinti atstumui nuo n iki terminalinės viršūnės. Pavyzdžiu, žemėlapje $h(n)$ gali būti 1) atstumas tiesia linija (t. y. oru) nuo miesto n iki galinio miesto arba 2) Manheteno atstumas.

Agentas yra suprantamas kaip sprendējas. Jis nemato viso „žemėlapio“ ir vadovaujasi įvertinimo funkcija $f(n)=g(n)+h(n)$.

Algoritmas A* publikuotas [Hart, Nilsson, Raphael 1968], o vėliau vadovėlyje [Nilsson 1982]. Algoritmas pateikiamas ir kituose dirbtinio intelekto vadoveliuose, pavyzdžiu, [Russell, Norvig 2003, Luger 2005] ir Vikipedijoje (žr. http://en.wikipedia.org/wiki/A*_search_algorithm). Algoritmas iš pradžių įvardijamas A, o vėliau dėl tam tikro optimalumo pavadinamas A*.

Dirbtiniame intelekte skiriamos dvi paieškos strategijos: informuota paieška ir neinformuota paieška.

Neinformuota paieška – kai paieškos algoritmui nėra žinomas žingsnių skaičius arba kelio įvertinimas nuo einamosios būsenos iki terminalinės. Neinformuotos paieškos pavyzdžiai yra paieška „i gylį“, paieška „i plotį“ ir paieška grafe, kai grafo briaunos turi kainą.

Informuota paieška – kai paieškos algoritmas remiasi euristiniu įvertinimu nuo einamosios būsenos iki terminalinės. Informuotos paieškos pavyzdžiai yra „kopimas į kalną“ ir A* paieška.

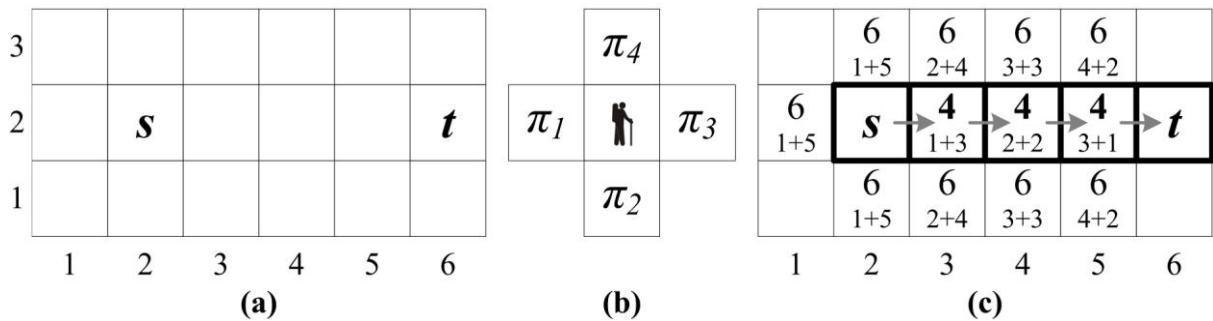
Informuota paieška yra efektyvesnė. Paieškos medyje yra mažiau viršūnių, bet efektyvumas pasiekiamas euristinio įvertinimo kaina.

17.1. A* pavyzdys, kai euristinė funkcija $h(n)$ yra Manheteno atstumas

Langelių lentoje, kuri gali būti suprantama kaip labirintas ir kaip grafas, agentas keliauja iš pradinio langelio (viršūnės) s į terminalinį t (žr. 17.1 pav.). Lentoje gali būti kliūčių (sienų). Vertinimo funkcijoje $f(n)=g(n)+h(n)$ dėmuo $g(n)$ yra nueito kelio nuo s iki n ilgis. Tegu euristinė funkcija $h(n)$ yra Manheteno atstumas nuo n iki t. Tegu keturi agento ėjimai (produkcijos) $\{\pi_1, \pi_2, \pi_3, \pi_4\}$ yra surūšiuoti šia tvarka: π_1 – „i vakarus“, π_2 – „i pietus“, π_3 – „i rytus“ ir π_4 – „i šiaurę“ (žr. 17.1 pav.). Priminsime kad Manheteno atstumas $d(n_1, n_2)$ tarp langelių $n_1=(x_1, y_1)$ ir $n_2=(x_2, y_2)$ yra apibrėžiamas taip:

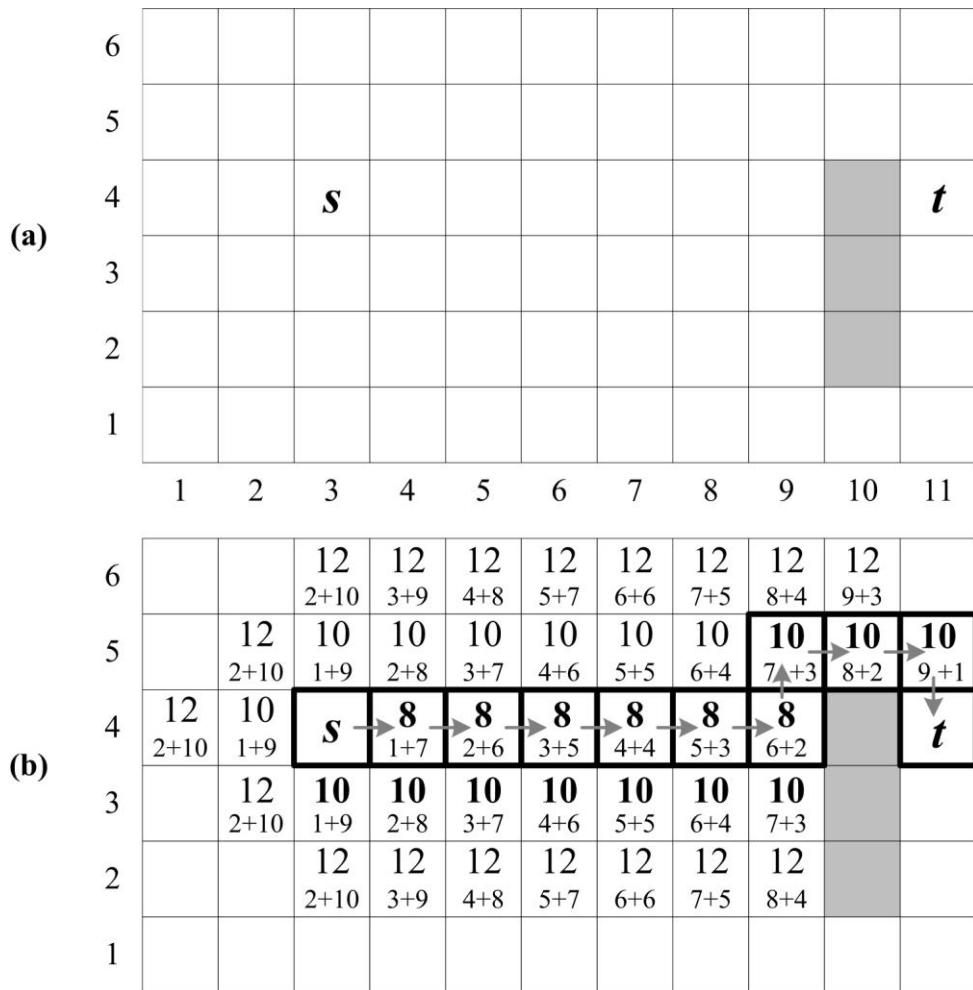
$$d(n_1, n_2) = |x_2 - x_1| + |y_2 - y_1|$$

Manheteno atstumas yra trumpiausias atstumas, kai leidžiama keliauti tik vertikaliai ir horizontaliai – keturiomis produkcijomis $\{\pi_1, \pi_2, \pi_3, \pi_4\}$ – t. y. draudžiama eiti istrižai.



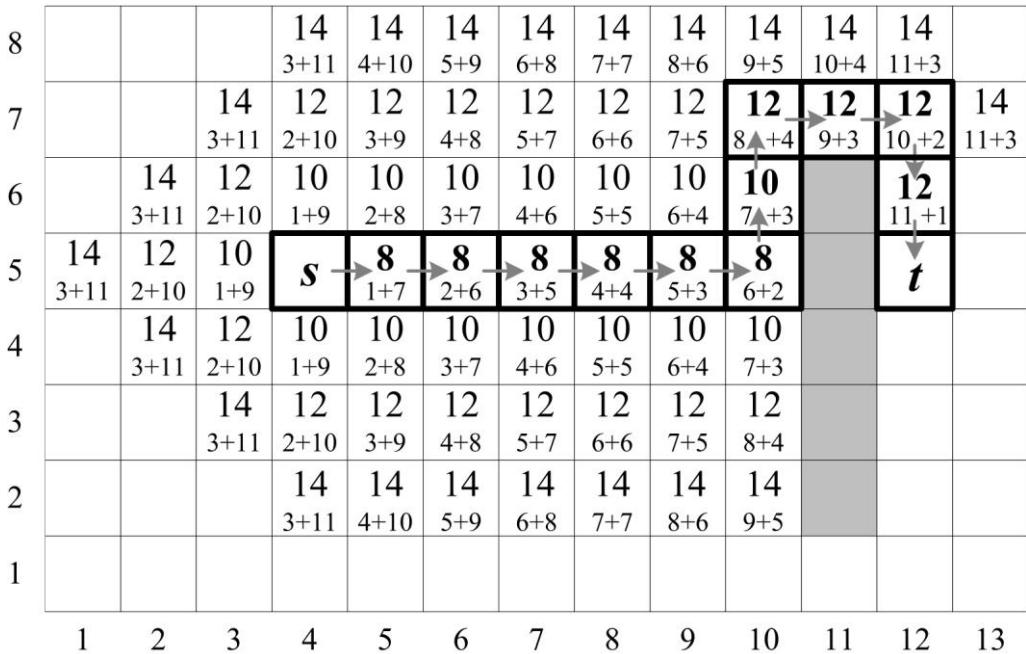
- 17.1 pav. a) Agentas keliauja iš pradinio lavelio (viršūnės) **s** į terminalinį **t**.
 b) Keturi agento ėjimai – tai keturios produkcijos: „i vakarus“, „i pietus“, „i rytus“ ir „i šiaurę“. c) Uždaromos ir atidaromos viršūnės ir rastas keliai iš **s** į **t**

Toliau pateikiamas antras – sudėtingesnis – pavyzdys, kai lentoje yra kliūtys, suprantamos kaip langeliai, į kuriuos negalima žengti. (žr. 17.2 pav.).



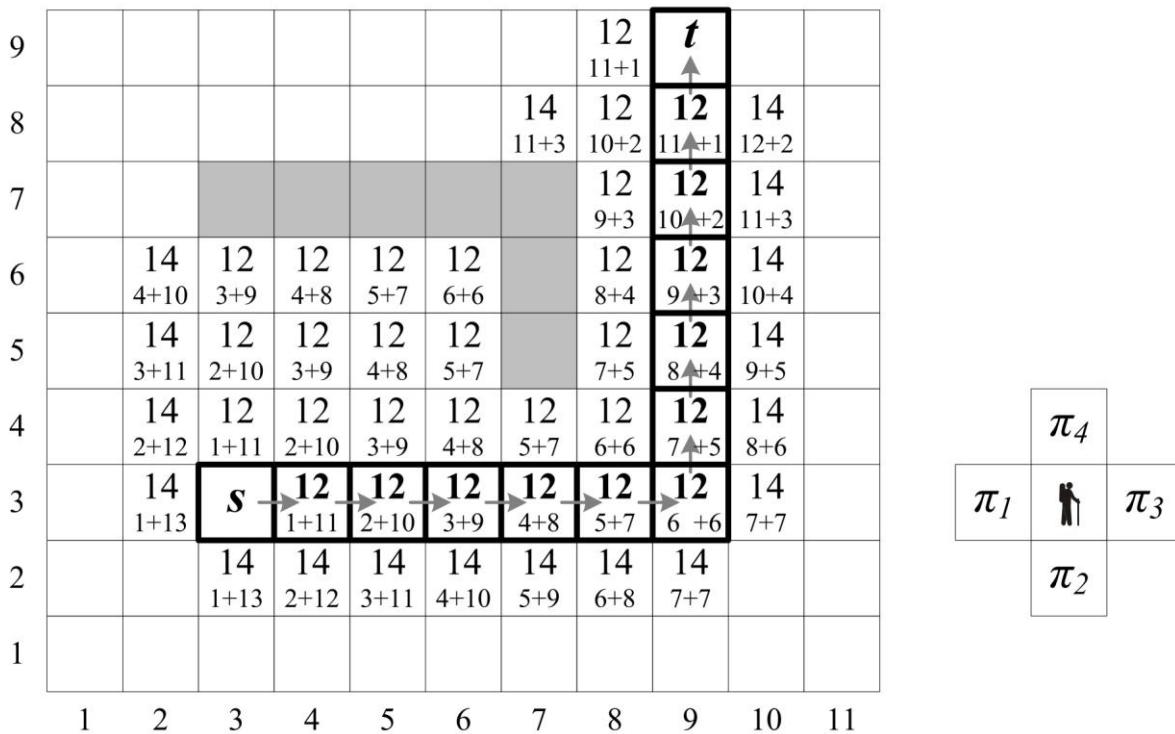
- 17.2 pav. a) Agentas keliauja iš pradinio lavelio **s** į terminalinį **t**. Paryškintos kliūtys, į kurias negalima žengti. b) Uždaromos ir atidaromos viršūnės bei rastas keliai

Toliau 17.3 pav. pateikiamas pavyzdys – su 5 langelių kliūtimi.



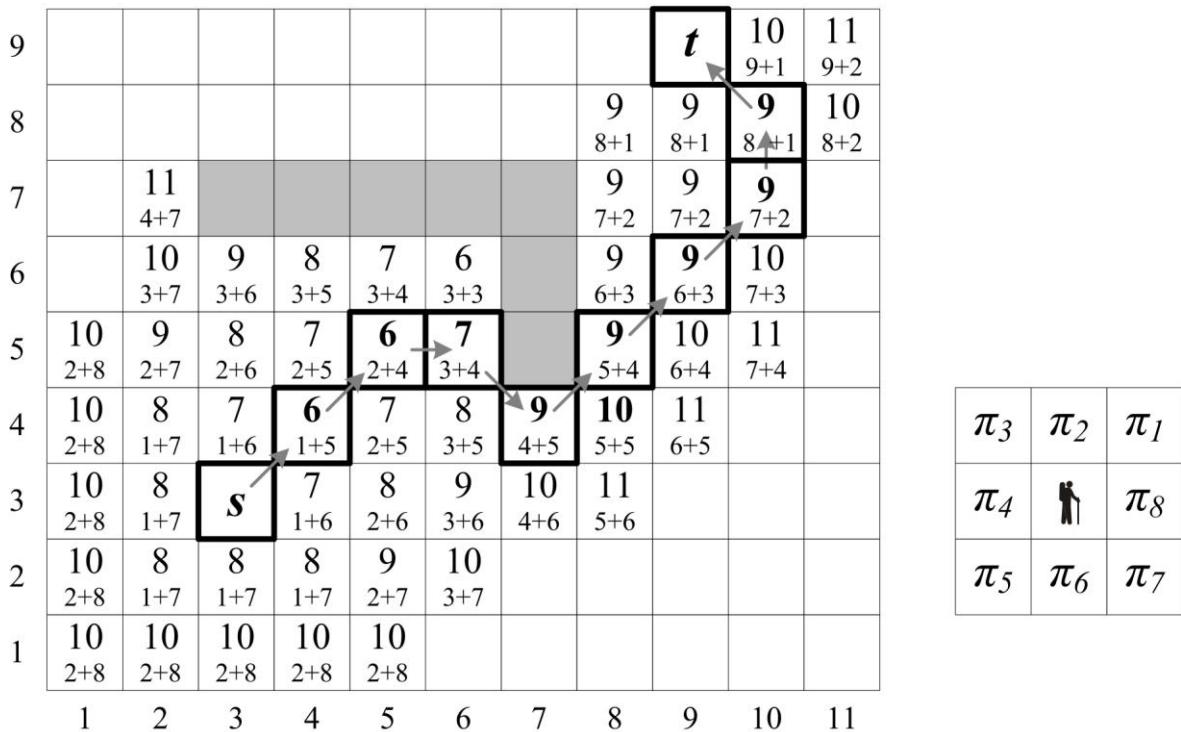
17.3 pav. Uždaromos ir atidaromos viršūnės bei rastas kelias iš s į t

Toliau 17.4 pav. pateikiamas pavyzdys, kai s it t yra ne toje pačioje eilutėje ir aplenkiamą L formos kliūtis.



17.4 pav. Aplenkiamą L formos kliūtis kliūtis. Parodytas rastas kelias iš s į t

Toliau 17.5 pav. pateikiamas pavyzdys, kai randamas kelias tokiam pat labirintė kaip aukščiau 17.4 pav., bet agentas turi 8 produkcijas, t.y. gali eiti ir įstrižainėmis. Rekomenduojama Vikipedijoje pasižiūrėti panašų dar didesį pavyzdį, kuriam frontas animuojamas skirtingais laiko momentais; žr. http://en.wikipedia.org/wiki/A*_search_algorithm.



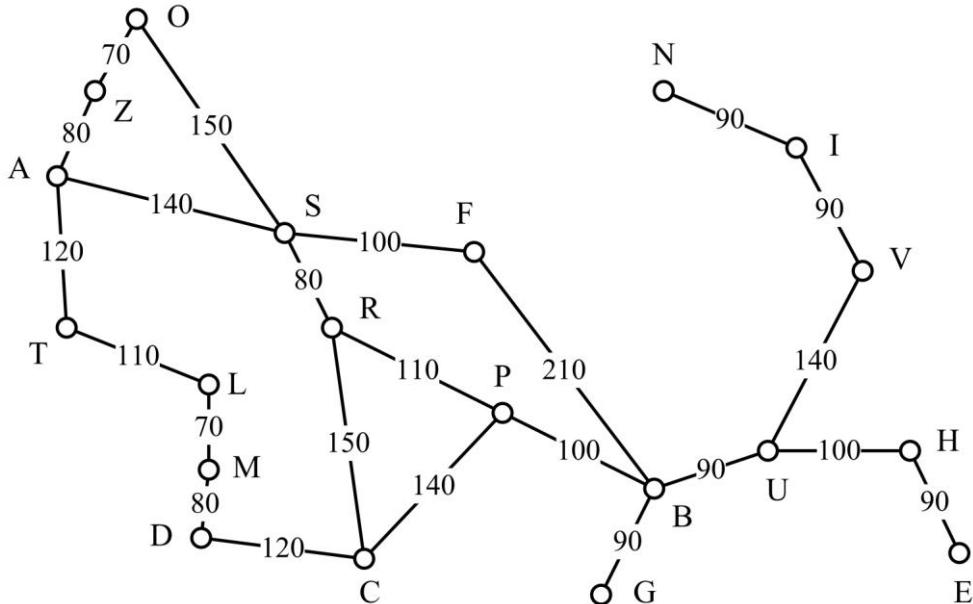
17.5 pav. Kelias iš s į t , kai yra aštuonios produkcijos – einama ir įstrižainėmis

Kaip galima pastebeti iš visų pavyzdžių nuo 17.1 iki 17.5 pav., banga turi „elipsės“ nuo s link t formą. Šitaip banga skleidžiama efektyviau, t. y. fronte yra žymiai mažiau viršūnių, negu skleidžiant apskritimais. Koncentrinis skleidimas gaunamas, kai $f(n)=g(n)$, t. y. neatsižvelgiama į euristinę funkciją $h(n)$ – terminalinio lanelio padėti (žr. 16.4 pav.).

Algoritmas A* yra optimalus, kai euristinė funkcija $h(n)$ yra optimistinė [Russell, Norvig 2003]. Euristika vadinama *optimistinė*, jeigu jos duodamas atstumo įvertis yra ne didesnis negu tikras atstumas. Agento kambaryste su kliūtimis uždavinyje tikras atstumas dėl kliūčių paprastai yra didesnis negu Manheteno. Tokiu būdu euristinė funkcija, imama kaip Manheteno atstumas yra optimistinė.

17.2. A* pavyzdys kelių žemėlapyje

Toliau algoritmas A* aiškinamas vadovaujantis [Russell, Norvig 2003] ketvirtu skyriumi p. 94-98. Ieškomas kelias iš viršūnės (miesto) A į B grafe (kelių žemėlapyje) 17.6 pav.



17.6 pav. Grafas, vaizduojantis Rumunijos kelių žemėlapį. Ieškomas kelias iš viršūnės (miesto) A į B. Funkcija $g(n)$ apskaičiuojama pagal atstumus tarp viršūnių. Šis supaprastintas pavyzdys paimtas vadovaujantis [Russell, Norvig 2003] 3.2 pav. p. 63

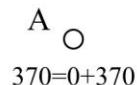
Euristinės funkcijos $h(n)$ reikšmės pateikiamos 17.7 lentelėje. Tai atstumas tiesia linija nuo n iki B.

Miestas	Atstumas	Miestas	Atstumas
A	370	M	240
B	0	N	230
C	160	O	380
D	240	P	100
E	160	R	190
F	180	S	250
G	80	T	330
H	150	U	80
I	230	V	200
L	240	Z	370

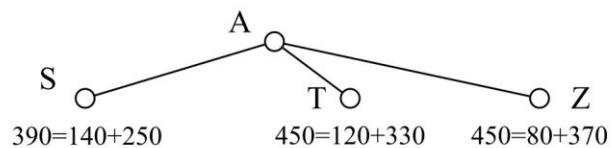
17.7 lentelė. Euristinės funkcijos $h(n)$ reikšmės. Tai atstumas tiesia linija nuo miesto n iki galinio miesto B

Algoritmo A* paieškos medžiai skirtingais žingsniais pateikiami 17.8 pav.

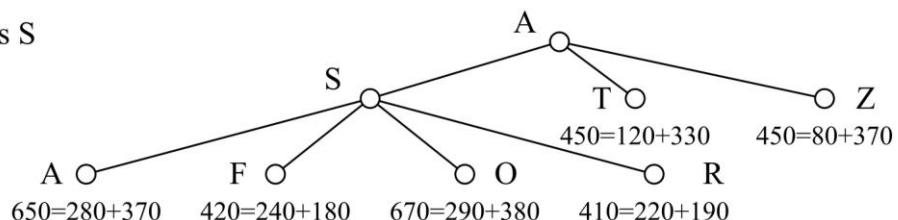
(a) Pradinė būsena



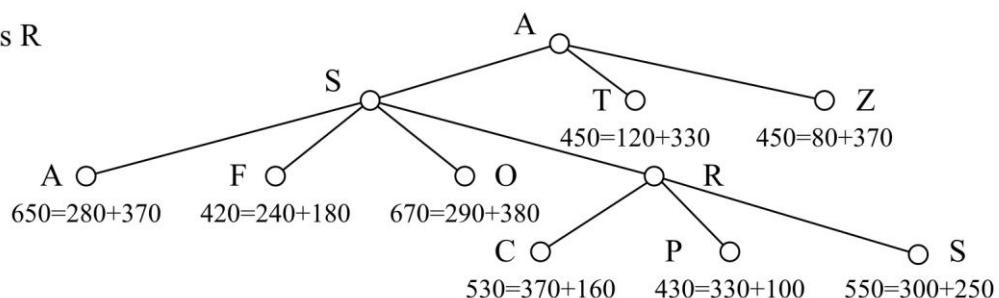
(b) Išskleidus, t. y. uždarius A



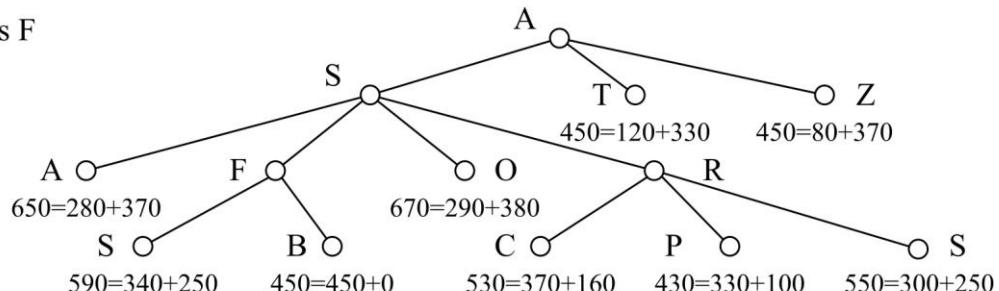
(c) Uždarius S



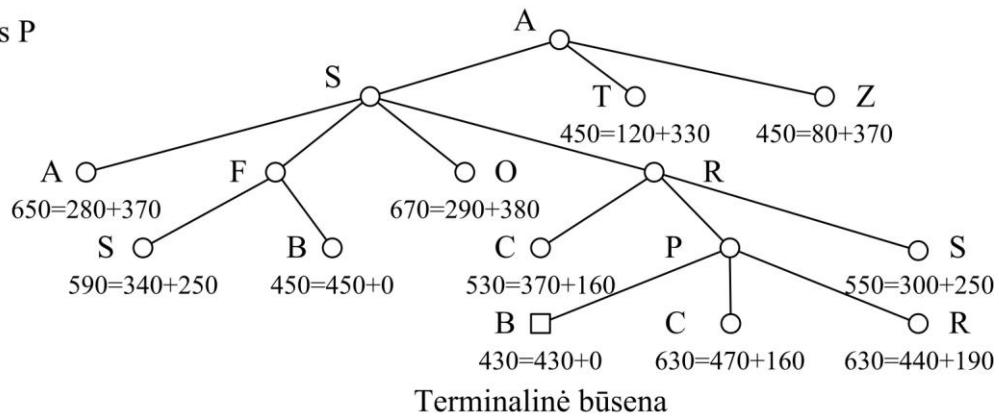
(d) Uždarius R



(e) Uždarius F



(e) Uždarius P



17.8 pav. Algoritmo A* paieškos medžiai skirtingais žingsniais. Viršūnės žymimos $f(n)=g(n)+h(n)$, kur $h(n)$ yra atstumas tiesia linija iki B

18. Tiesioginis ir atbulinis išvedimas produkcijų sistemoje

Tiesioginio ir atbulinio išvedimo sąvoką paaiškinsime paprasto pavidalo produkcijų sistemos atveju. Yra vadovaujamasi pavyzdžiu iš [Waterman 1989].

Paimkime produkcijų sistemą:

$$\begin{aligned}\pi_1: & F, B \rightarrow Z \\ \pi_2: & C, D \rightarrow F \\ \pi_3: & A \rightarrow D\end{aligned}\tag{18.1}$$

Produkcijos semantika apibrėžiama šitaip. Jeigu globalios duomenų bazės būsenoje yra kartu visi objektais, nurodyti kairėje produkcijos pusėje (π_1 atveju, tai F ir B abudu), tai produkciją galima taikyti. Pritaikius produkciją, nauja GDB būsena papildoma objektu, kuris yra nurodytas dešinėje produkcijos pusėje (π_1 atveju, tai Z).

Bendru atveju, produkcijų aibė yra sudaryta iš N produkcijų. Kairiąjį produkcijos pusę sudaro objektų vardai iš tam tikros vardu aibės, dar vadinamos alfabetu. Dešiniają pusę sudaro vieno objekto vardas. Jeigu dešiniajają pusę sudarytų keli objektais, pavyzdžiu, $\pi_j: A, B, C \rightarrow J, K$, tai produkcija gali būti perrašyta į dvi:

$$\begin{aligned}\pi_{j1}: & A, B, C \rightarrow J \\ \pi_{j2}: & A, B, C \rightarrow K\end{aligned}$$

Šitaip kablelis tarp objektų kairiojoje produkcijos pusėje yra suprantamas kaip konjunkcijos operacija $\&$, o kablelis dešiniojoje pusėje – kaip disjunkcija \vee .

Objektą A galima traktuoti kaip dvejetainį kintamąjį. Matematinės logikos terminais jis yra vadinamas *propoziciniu kintamuoju*. Kai sakoma, kad turimas objektas A, tai reiškia, kad dvejetainio kintamojo A reikšmė yra *true*. Kitaip tariant turimas faktas A.

Tarkime, kad pradinė GDB būsena yra {A,B,C}, o tikslas – Z.

Uždavinys yra formuluojamas taip: rasti produkcijų seką, kuri pradinę GDB būseną pverda į terminalinę būseną. Terminalinė būsena yra bet kokia būsena, kurioje yra tikslas objektas (pateiktame pavyzdyme Z).

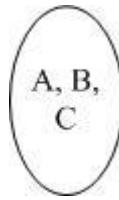
Produkcijų sekai gauti aptariami du algoritmai: tiesioginis išvedimas (angl. *forward chaining*) ir atbulinis išvedimas (angl. *backward chaining*). Tiesioginis išvedimas pradeda nuo pradiniių duomenų ir juda link tikslas. Atbulinis išvedimas pradeda nuo tikslas ir juda link duomenų.

18.1. Tiesioginio išvedimo algoritmas

Pradedama nuo pradinės GDB būsenos, t. y. nuo faktų.

Produkcijos perrenkamos iteracijomis. Kiekvienoje iteracijoje produkcijos perrenkamos iš eilės. Jeigu paimtą produkciją galima taikyti, tai ji yra taikoma, paskui pažymima, kad daugiau nebūtų taikoma kitose iteracijose ir valdymas perduodamas kitai iteracijai. Priešingu atveju imama kita produkcija. Jeigu produkcijos išsemtos, tai nesékmė – ieškoma produkcijų sekai neegzistuoja.

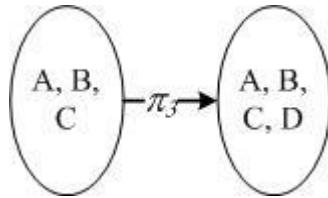
Šio algoritmo vykdymas demonstruojamas produkcijų sistemoje (18.1). Pradedama nuo pradinės būsenos {A,B,C}, pavaizduotos 18.1 pav.



18.1 pav. Pradinė globalios duomenų bazės būsena {A,B,C}

1 iteracija. Esamoje būsenoje {A,B,C} produkcijos π_1 negalima taikyti, nes objekto F nėra GDB būsenoje. Produkcijos π_2 taip pat negalima taikyti, nes nors ir yra objektas C, bet nėra D. Produkciją π_3 galima taikyti, nes objektas A yra. Ji taikoma. Pereinama į naują GDB būseną {A,B,C,D}, kuri gaunama papildžius pradinę būseną objektu D iš π_3 dešiniosios pusės. Tai nėra terminalinė būsena, nes joje nėra tikslo Z.

Produkcijos π_3 taikymas yra pavaizduotas 18.2 pav.

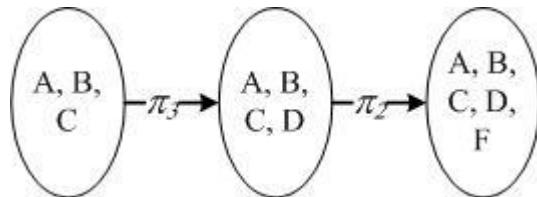


18.2 pav. Pritaikius produkciją π_3 yra pereinama į naują GDB būseną {A,B,C,D}, kuri pavaizduota dešinėje

Pereinama prie kitos iteracijos.

2 iteracija. Esamoje būsenoje {A,B,C,D} produkcijos π_1 negalima taikyti, nes objekto F nėra šioje būsenoje. Produkciją π_2 galima taikyti, nes yra ir C, ir D. Ji taikoma. Pereinama į būseną {A,B,C,D,F}, kuri gaunama papildžius einamają būseną objektu F iš π_2 dešiniosios pusės. Tai nėra terminalinė būsena, nes joje nėra tikslo Z.

Produkcijos π_2 taikymas pavaizduotas 18.3 pav.

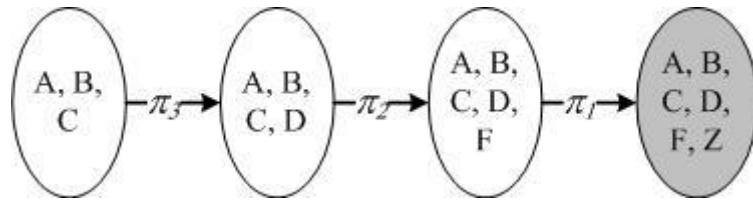


18.3 pav. Pritaikius produkciją π_2 yra pereinama į naują GDB būseną {A,B,C,D,F}, kuri pavaizduota dešinėje

Pereinama prie kitos iteracijos.

3 iteracija. Esamoje būsenoje {A,B,C,D,F} produkciją π_1 galima taikyti, nes ir F ir B yra šioje būsenoje. Ji taikoma. Pereinama į būseną {A,B,C,D,F,Z}, kuri gaunama einamają būseną papildžius objektu Z iš π_1 dešiniosios pusės. Tai terminalinė būsena. Algoritmas baigia darbą.

Produkcijos π_1 taikymas pavaizduotas 18.4 pav.

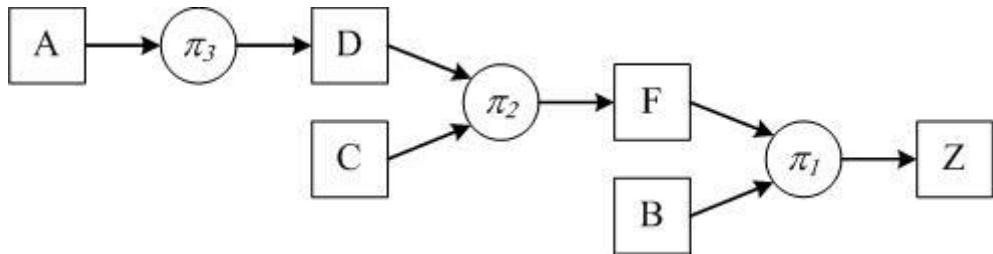


18.4 pav. Pritaikius produkciją π_1 yra pereinama į naują GDB būseną $\{A,B,C,D,F,Z\}$, kuri pavaizduota dešinėje. Tai terminalinė būsena, nes joje yra tikslas Z

Gautas rezultatas yra tokia produkcijų seka, kuri dar vadinama *planu*:

$$\langle \pi_3, \pi_2, \pi_1 \rangle \quad (18.2)$$

Ši produkcijų seka gali būti transformuota į *dvidalį grafi* (angl. *bipartite graph*, rus. *двойно́й гра́ф*), pateiktą 18.5 pav. Tai toks grafas, kurio viršūnės yra dviejų rūsių. Šiuo atveju tai viršūnės, žymimos stačiakampiais, ir viršūnės, žymimos skrituliais. Viršūnė-stačiakampis vaizduoja objektą ir yra pažymėta šio objekto vardu. Viršūnė-apskritimas vaizduoja produkciją ir yra pažymėta šios produkcijos vardu. Informatikoje koncepciniame modeliavime tokio pavidalo grafas yra vadinamas *semantiniu grafu* (*semantiniu tinklu*, *koncepciniu grafu*). Jis vaizduoja ir objektus, ir produkcijas. Iš tokio koncepcinio grafo nesunku „ištraukti“ produkcijų seką.



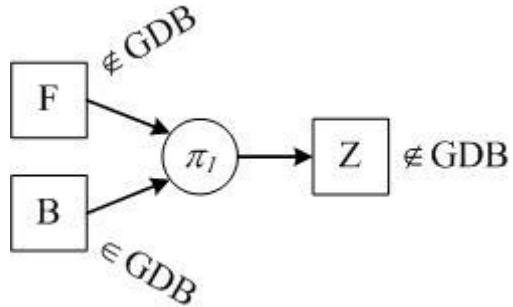
18.5 pav. Semantinis grafas, vaizduojantis objekto (fakto) Z išvedimą iš objektų (faktų) $\{A,B,C\}$ produkcijų sistemoje (18.1). Gaunama produkcijų seka $\langle \pi_3, \pi_2, \pi_1 \rangle$

18.2. Atbulinio išvedimo algoritmas

Skirtingai nuo tiesioginio išvedimo atbulinis išvedimas pradeda nuo tikslo (šiame pavyzdje – nuo Z), o ne nuo duomenų. Atbulinio išvedimo algoritmas pateikiamas produkcijų sistemos (18.1) pavyzdžiu.

1 iteracija. Produkcijos (18.1) yra perrenkamos nuo pirmosios. Ieškoma tokios produkcijos, kurios dešinėje pusėje yra tikslo objektas Z. Tokia produkcija yra π_1 . Ji pažymima, kad nebūtų išrenkama kitose iteracijose. Imama išrinktos produkcijos π_1 kairioji pusė $\{B,F\}$. Objektas B yra pradinėje GDB būsenoje $\{A,B,C\}$, o objekto F nėra. Todėl F padaromas nauju potiksliu.

Ši iteracija pavaizduota 18.6 pav.

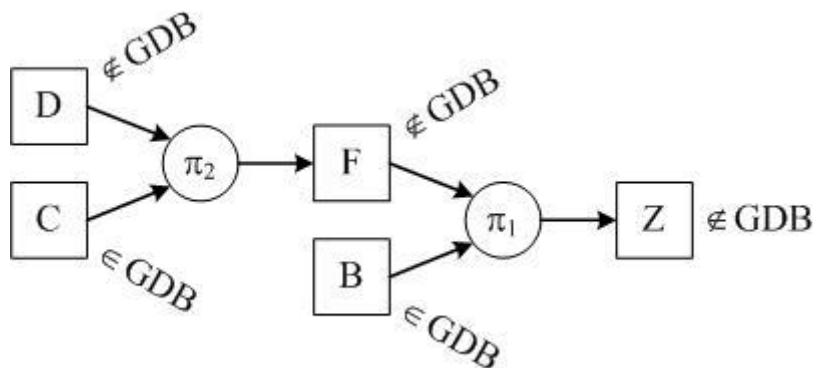


18.6 pav. Produkcijos π_1 semantinis grafas. Jis vaizduoja objekto Z išvedimą „atgal“, taikant produkciją π_1 . Produkcijos įėties objektas B yra būsenoje {A,B,C}, o objekto F nėra. Todėl F padaromas nauju potiksliu

Pereinama prie kitos iteracijos.

2 iteracija. Naujas tikslas – F. Produkcijos yra perrenkamos nuo pirmosios. Ieškoma tokios produkcijos, kurios dešinėje pusėje yra F. Tokia produkcija yra π_2 . Ji pažymima, kad nebūtų išrenkama kitose iteracijose. Imama išrinktos produkcijos π_2 kairioji pusė {C,D}. Objektas C yra pradinėje būsenoje {A,B,C}, o objekto D nėra. Todėl D padaromas nauju potiksliu.

Ši iteracija pavaizduota 18.7 pav.

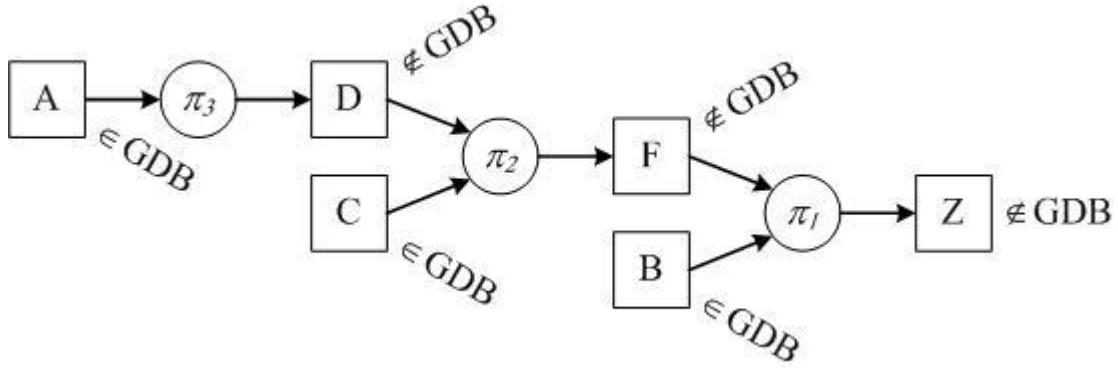


18.7 pav. Semantinis grafas, vaizduojantis objekto F išvedimą „atgal“, taikant produkciją π_2 . Objektas C yra pradinėje būsenoje {A,B,C}, o objekto D nėra. Todėl D padaromas nauju potiksliu

Pereinama prie kitos iteracijos.

3 iteracija. Naujas tikslas – D. Produkcijos yra perrenkamos nuo pirmosios. Ieškoma tokios produkcijos, kurios dešinėje pusėje yra D. Tokia produkcija yra π_3 . Šiame pavyzdyme ji vienintelė iš likusių nepažymėtų. Ji pažymima. Imama išrinktos produkcijos π_3 kairioji pusė {A}. Objektas A yra pradinėje būsenoje {A,B,C}, todėl išvedimas sėkmingai baigiamas. Algoritmas baigia darbą.

Gautas semantinis grafas pavaizduotas 18.8 pav.



18.8 pav. Semantinis grafas, vaizduojantis objekto Z išvedimą iš $\{A, B, C\}$ produkcių sekos $\langle \pi_3, \pi_2, \pi_1 \rangle$

18.3. Programų sintezės elementai

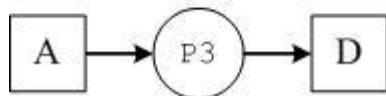
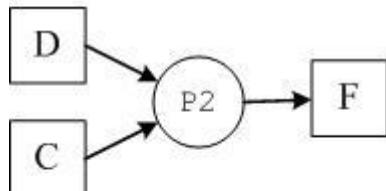
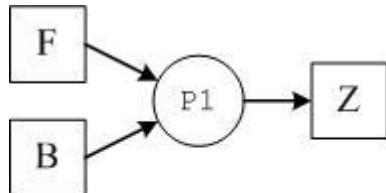
Ir tiesioginio, ir atbulinio išvedimo (18.1) pavidalo produkcių sistemoje rezultatas yra produkcių seka – planas. Ši seka vadinama planu todėl, kad ji planuojama vieną kartą, o vykdoma gali būti daug kartų.

Paimkime procedūras (18.3). Jų semantika yra pavaizduota produkciomis (18.1). Čia programos semantika yra suprantama kaip pora (*įėjimas, išėjimas*). Ir įėjimas, ir išėjimas yra objektų vardų aibė. Objektus yra išprasta realizuoti kaip globalius kintamuosius algoritminėje kalboje, pvz., Pascal, Fortran, C. Visų šių vardų aibė yra vadinama *alfabetu*. Šiame pavyzdyme alfabetas yra $\{A, B, C, D, E, F, Z\}$.

procedure P1; Z := f1(B, F)	(18.3)
procedure P2; F := f2(C, D)	
procedure P3; D := f3(A)	

Procedūromis P1, P2, P3 yra užprogramuotos tam tikros funkcijos f1, f2, f3. Jos vadinamos *funcinėmis priklausomybėmis* tarp atitinkamų objektų. Tokio pavidalo procedūros yra skirtos pavaizduoti procedūrines žinias apie tam tikrą dalykinę sritį. Programų sistemoje tokios procedūros paprastai yra apjungiamos į biblioteką.

Procedūrų P1, P2, P3 semantika yra vaizduojama atitinkamai produkciomis π_1 , π_2 ir π_3 . Grafiškai ir procedūrų semantika, ir jas atitinkančios produkcijos yra vaizduojamos tokio paties pavidalo semantiniaisiais grafais (žr. 18.9 pav.).



18.9 pav. Procedūrų P_1 , P_2 ir P_3 semantikos grafinis pavaizdavimas semantiniai grafaus. Procedūros semantika yra suprantama kaip pora (iėjimas, išėjimas). Tokiu būdu, procedūros semantikos pavaizdavimas tekstu sutampa su produkčija.

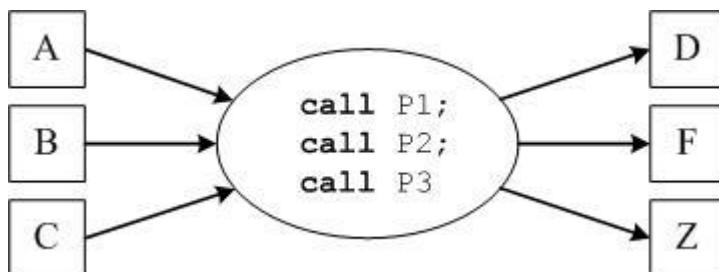
Kadangi produkcijos yra siejamos su procedūromis, tai ir tiesioginio, ir atbulinio išvedimo rezultatas, t. y. produkcių seka (18.2), yra transformuojama į šitokią *sintezuotą programą*:

```

call P3;
call P2;
call P1
  
```

(18.4)

Šios programos semantika (iėjimas, išėjimas), yra pavaizduota semantiniu grafu 18.10 pav.



18.10 pav. Sintezuotos programos semantikos vaizdavimas semantiniu grafu

Ši programa gali būti vykdoma daug kartų, prieš tai atlikus planavimą ir sintezę tik vieną kartą. Sintezuotą programą tikslina vykdyti cikle su skirtingu reikšmiu A, B, C įvedimu:

```

for J := 1 to 900 do { Ciklas kartojamas 900 kartu }
begin
    readln (A, B, C); { 1) Įvedamos A, B ir C reikšmės }

    call P3;           { 2) Kviečiama sintezuota programa }

    call P2;
    call P1;

    writeln(Z);        { 3) Spausdinama Z reikšmė }

end

```

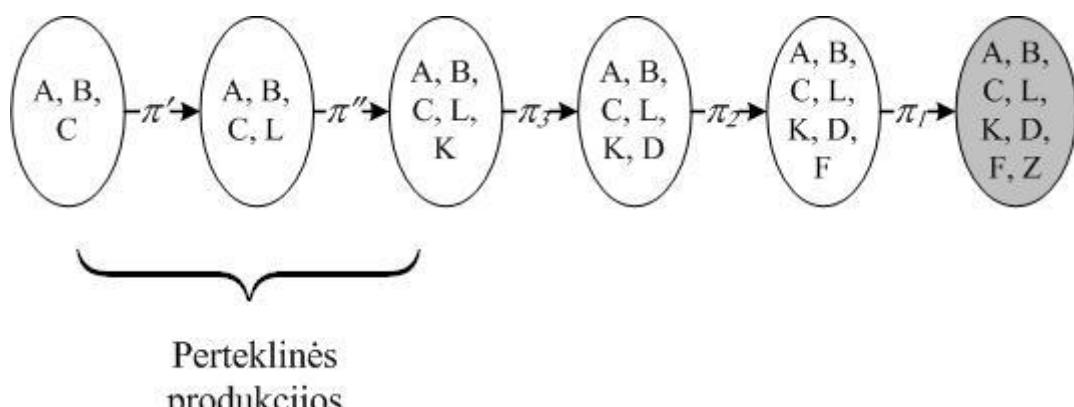
18.4. Perteklinės produkcijos tiesioginiame išvedime

Perteklinės produkcijos produkcių sekoje, kurią randa tiesioginio išvedimo algoritmas, yra iliustruojamos pavyzdžiu.

Imama produkcių sistema (18.1) ir papildoma dviem produkciomis π' ir π'' :

$$\begin{aligned}
 \pi': & A \rightarrow L \\
 \pi'': & L \rightarrow K \\
 \pi_1: & F, B \rightarrow Z \\
 \pi_2: & C, D \rightarrow F \\
 \pi_3: & A \rightarrow D
 \end{aligned} \tag{18.5}$$

Tegu turimų faktų aibė yra $\{A, B, C\}$, o tikslas – Z . Tiesioginio išvedimo algoritmu rastoje produkcių sekoje $\langle \pi', \pi'', \pi_3, \pi_2, \pi_1 \rangle$ yra dvi perteklinės produkcijos π' ir π'' . Jų sąlygojami padariniai yra perteklinių objektų L ir K išvedimas. Tiesioginis išvedimas pavaizduotas 18.11 pav.



18.11 pav. Tiesioginio išvedimo algoritmu rastoje produkcių sekoje $\langle \pi', \pi'', \pi_3, \pi_2, \pi_1 \rangle$ yra dvi perteklinės produkcijos π' ir π'' . Yra išvedami pertekliniai objektai L ir K

Šiame pertekliškume „tragedijos“ kaip nesékmés požymiu nėra. Tiesiog pailgėja ir išvedimo proceso laikas, ir sintezuotos programos vykdymo laikas. Jeigu sintezuota programa yra vykdoma keletą kartų, tai kiekvieną kartą yra gaminami pertekliniai objektai L ir K.

Motyvacija yra tokia. Analitikas, sudarinėdamas dalykinės srities modelį, žinias užrašo produkcijų pavidalu. Analitikas neturi išankstinės nuostatos, kad bus taikomas tiesioginio išvedimo būdas. Produkcijų aibė gali būti labai didelė. Iš anksto nėra žinoma, kurios produkcijos bus naudojamos tam tikram faktui išvesti, o kurios ne. Kaip pradinius duomenis gavęs tam tikrus faktus, algoritmas į planą gali įtraukti ir perteklinės produkcijas. Kalbant apie išvedimo algoritmą yra nagrinėjamas išvedimo metodas, o ne dalykinė sritis.

Perteklinės produkcijos atsiranda, nes tiesioginis išvedimas yra *monotoniniškas* matematinės logikos prasme.

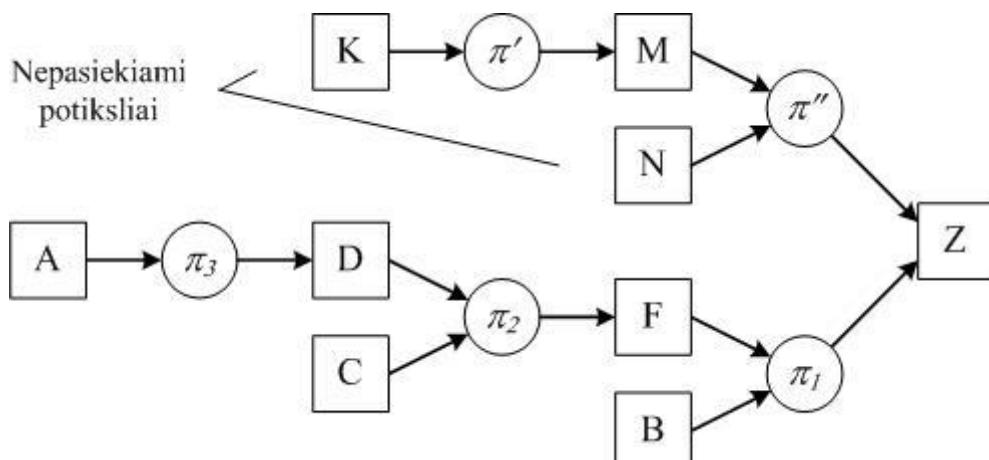
18.5. Perteklinės produkcijos atbuliniame išvedime

Perteklinės produkcijos gali atsirasti ir atbuliniame išvedime. Tik šiuo atveju atsiranda nepasiekiamų (perteklinių) tikslų. Kaip ir tiesioginiame išvedime, perteklišumas nėra nesékmés požymis. Jis tiesiog pailgina plano išvedimo laiką.

Toliau nagrinėjamas pavyzdys, kai prie produkcijų sistemos (18.1) yra pridėtos dvi papildomos produkcijos π' ir π'' :

$$\begin{aligned}\pi' &: K \rightarrow M \\ \pi'' &: M, N \rightarrow Z \\ \pi_I &: F, B \rightarrow Z \\ \pi_2 &: C, D \rightarrow F \\ \pi_3 &: A \rightarrow D\end{aligned}\tag{18.6}$$

Faktų aibė $\{A, B, C\}$ ir tikslas Z yra tie patys. Perteklinės produkcijos π' ir π'' nagrinėjamos išvedime – iš pradžių π'' o paskui gylyn π' . Tačiau jos į planą $\langle \pi_3, \pi_2, \pi_I \rangle$ nėra įtraukiamos. Dėl jų yra iškeliami pertekliniai tikslai K ir N – π'' iškelia potikslius M ir N, o π' potikslį K. Ir N ir K yra nepasiekiami. Atbulinio išvedimo sematinis grafas pateiktas 18.12 pav:



18.12 Atbuliniame išvedime yra dvi perteklinės produkcijos π' ir π'' . Jų potiksliai N ir K yra nepasiekiami. Šios produkcijos į randamą planą $\langle \pi_3, \pi_2, \pi_I \rangle$ nėra įtraukiamos

18.6. Tiesioginio išvedimo sudėtingumas

Teorema. Tiesioginio išvedimo sudėtingumas produkcijų sistemoje, turinčioje (18.1) pavidalą, yra $O(N^2)$, kur N yra produkcijų skaičius.

Irodymas. Pirmojoje iteracijoje atliekama N veiksmų (blogiausiu atveju). Vieno veiksmo metu tikrinama, ar galima taikyti produkciją. Antrojoje iteracijoje atliekama vienu veiksmu mažiau, t. y. $N-1$ veiksmų, nes pirmosios iteracijos metu išsirinkta produkcia yra pažymima, kad nebūtų taikoma kitose iteracijose. Trečiojoje iteracijoje atliekama $N-2$ veiksmų. Ir taip toliau. Paskutinė plano produkcia yra pasirenkama N -tosios iteracijos metu, kai po $N-1$ -osios iteracijos lieka nepažymėta viena produkcia. Tokiu būdu bendras atliekamų veiksmų skaičius (blogiausiu atveju) yra:

$$N + N-1 + N-2 + \dots + 1 = N \cdot (N-1)/2 = O(N^2)$$

Irodymo pabaiga.

Polinominis, netgi labai gero laipsnio – kvadratinis, o ne eksponentinis – sudėtingumas yra todėl, kad faktai yra suprantami kaip propoziciniai kintamieji, o ne kaip predikatų vardai. Tokiu būdu nekyla predikatų parametrų sutapatinimo problema, žinoma, pavyzdžiui, Prologo algoritminėje kalboje. Išvedimo Prologue sudėtingumas yra eksponentinis.

Pratimai

1. Naudodami 1) tiesioginį išvedimą ir 2) atbulinį išvedimą išveskite iš pradinės GDB būsenos $\{A,B,C\}$ tikslą Z produkcių sistemoje:

$$\begin{aligned}\pi_1: \quad &G, E \rightarrow H \\ \pi_2: \quad &H \rightarrow Z \\ \pi_3: \quad &A \rightarrow D \\ \pi_4: \quad &A, E \rightarrow F \\ \pi_5: \quad &B, D \rightarrow E \\ \pi_6: \quad &F \rightarrow G\end{aligned}$$

Parašykite planą ir nubraižykite semantinius grafus. Ar egzistuoja perteklinės produkcijos?

19. Rezoliucijų metodas

Išvedimas rezoliucijų metodu demonstruojamas pavyzdžiu iš [Thayse et al. 1990, p. 165]. Tegu turime teiginį „Profesorius gali egzaminuoti tik kito fakulteto studentus“ ir du faktus:

1. Žakas yra informatikos fakulteto profesorius;
2. Mari yra matematikos fakulteto studentė.

Reikia įrodyti kad profesorius Žakas gali egzaminuoti studentę Mari.

Du faktus užrašysime predikatais:

$$\text{Faktas F1: } \text{Prof}(\text{Info}, \text{Žakas}) \quad (19.1)$$

$$\text{Faktas F2: } \text{Stud}(\text{Mat}, \text{Mari}) \quad (19.2)$$

Pradinis teiginys užrašomas formaliau: jeigu y yra fakulteto x profesorius ir w yra fakulteto z studentas, tai profesorius y gali egzaminuoti studentą w (suprantama, kad $z \neq x$). Pastarasis teiginys užrašomas logikos formule:

$$\text{Prof}(x, y) \wedge \text{Stud}(z, w) \wedge \neg \text{Lygu}(x, z) \Rightarrow \text{Egz}(y, w) \quad (19.3)$$

Pastaroji formulė (19.3) suprantama kaip produkcinė taisykla R1 – vienintelė šiame pavyzdje. Čia $\text{Prof}(x, y)$, $\text{Stud}(z, w)$, $\text{Lygu}(x, z)$ ir $\text{Egz}(y, w)$ yra predikatai. Primenama, kad predikatas yra funkcija, kuri įgyja vieną iš dviejų reikšmių true arba false . $\text{Prof}(x, y)$ žymi, kad y yra fakulteto x profesorius. $\text{Stud}(z, w)$ žymi, kad w yra fakulteto z studentas. $\text{Lygu}(x, z)$ žymi, kad x ir z yra tas pats fakultetas. $\text{Egz}(y, w)$ žymi, kad dėstytojas y gali egzaminuoti studentą w .

Implikacija $F \Rightarrow G$ pašalinama, keičiant disjunkcija ir neigimu $\neg F \vee G$. Ši pakeitimą galima suprasti ir kaip termų perrašymo taisykla (angl. *term rewriting rule*):

$$\frac{F \Rightarrow G}{\neg F \vee G} \quad \text{Implikacijos pašalinimo taisykla} \quad (19.4)$$

Čia virš brūkšnio yra, kas perrašoma, o po brūkšniu – į ką perrašoma.

Iš matematinės logikos žinoma, kad ši perrašymo taisykla yra *validi*, (angl. *sound*). Kitais žodžiais, korektiška semantiškai, logiška, nes viršutinė ir apatinė formulė yra ekvivalentios: $F \Rightarrow G \equiv \neg F \vee G$. Šių formulų teisingumo lentelės sutampa:

F	G	$\neg F$	$\neg F \vee G$	$F \Rightarrow G$
t	t	f	t	t
t	f	f	f	f
f	t	t	t	t
f	f	t	t	t

Implikacijos pašalinimo taisykla (19.4) paaiškinama šiuo pavyzdžiu:

$$\frac{\text{lyja} \Rightarrow \text{šlapia}}{\neg \text{lyja} \vee \text{šlapia}}$$

Implikacijos pašalinimo taisykla yra apibendrinama šitaip:

$$\frac{F_1 \And F_2 \And \dots \And F_n \Rightarrow G}{\neg F_1 \vee \neg F_2 \vee \dots \vee F_n \vee G} \quad (19.5)$$

Kadangi $\neg\neg F \equiv F$ tai turima taip vadinama dvigubo neigimo pašalinimo taisyklė:

$$\frac{\neg\neg F}{F} \quad (19.6)$$

Toliau produkcija (19.3) perrašoma naudojant implikacijos pašalinimo taisyklę (19.5):

$$\neg Prof(x, y) \vee \neg Stud(z, w) \vee \neg\neg Lygu(x, z) \vee Egz(y, w)$$

Pastarojoje formulėje pašalinamas dvigubas neigimas:

$$\neg Prof(x, y) \vee \neg Stud(z, w) \vee Lygu(x, z) \vee Egz(y, w) \quad (19.7)$$

Tokiu būdu reziumuojama, kad teiginys „Profesorius gali egzaminuoti tik kito fakulteto studentus“ – produkcinė taisyklė R1 – yra perrašytas formule (19.7).

Toliau bus parodyta, kaip įrodoma rezoliucijos taisyklės pagalba.

Rezoliucijos taisyklė pateikiama toliau. Ją aiškinti pradedama nuo įrodymo taisyklės *modus ponens* (lotyniškai – teigimo būdas):,,1) Tegu F. 2) Tegu jei F, tai G. 3) Taigi G“.

$$\frac{\begin{array}{c} F \\ F \Rightarrow G \\ \hline G \end{array}}{\begin{array}{c} 1) \text{ Mažoji premisa (prielaida)} \\ 2) \text{ Didžioji premisa} \\ 3) \text{ Išvada} \end{array}} \quad \text{modus ponens}$$

Mes dar vaizduosime abi prielaidas rašydam i viena šalia kitos (tvarka nėra svarbi):

$$\frac{F, \quad F \Rightarrow G}{G} \quad \text{modus ponens}$$

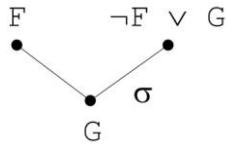
Pavyzdžiui, „1) Lyja. 2) Jei lyja, tai šlapia. 3) Taigi šlapia.“ *Modus ponens* taikymo pavyzdžiai yra 28 skyriuje. Pvz., „Sokratas yra žmogus. Jei x yra žmogus, tai mirtingas x. Taigi Sokratus mirtingas.“ Įrodoma keičiant kintamajį x konstanta Sokratus. Formaliai:

$$\frac{\begin{array}{c} 1) \text{ žmogus (Sokratus)} \\ 2) \forall x \text{ žmogus (x)} \Rightarrow \text{mirtingas (x)} \\ \hline 3) \text{mirtingas (Sokratus)} \end{array}}{\{ \text{Sokratus/x} \}} \quad \text{modus ponens}$$

Rezoliucijos taisyklė paprasčiausia forma užrašoma šitaip:

$$\frac{F, \quad \neg F \vee G}{G} \quad \sigma \quad \text{Rezoliucijos taisyklė} \quad \text{(paprasčiausia forma)} \quad (19.8)$$

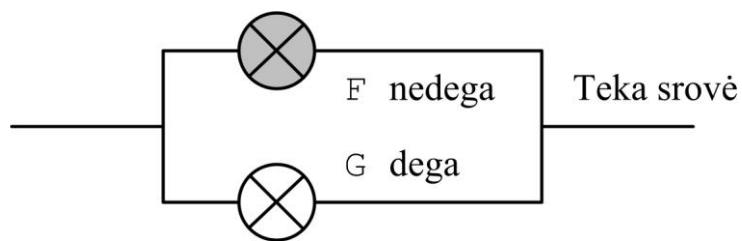
Grafiškai parodyta 19.1 pav. Aiškinima šitaip. Tegu turimos dvi formulės F ir $\neg F \vee G$. Pagal rezoliucijos taisyklę išvedama G. Pastaroji formulė vadinama *rezolvente*. Keitinys pažymėtas σ . Nagrinėjamos formulės turinčios pavidalą $\forall x_1 \forall x_2 \dots \forall x_n G(x_1, x_2, \dots, x_n)$. Plačiau žr. [Norgėla 2007, p. 26, 92, 158–163], [Nilsson 1998, p. 253–268] ir kt.



19.1 pav. Grafinis rezoliucijos taisyklės pavaizdavimas

Užrašymo forma (19.8) analogiška *modus ponens*. Prisiminkime, kad aukščiau parodėme formuliu $F \Rightarrow G$ ir $\neg F \vee G$ yra ekvivalentumą; žr. nuo (19.4). Rezoliucijos taisyklė taikoma, kai i vieną formulę įeina teigiamas disjunktas F , o i kitą – neigiamas disjunktas $\neg F$. Rezolventėje šių disjunktų néra.

Ir modus ponens, ir rezoliucijos taisyklės teisingumas iliustruojamas 19.2 pav. Tegu F ir G žymi lemputes. Jeigu G lemputė neveikia (t. y. nedega), tai daroma išvada, kad F veikia.



19.2 pav. Lygiagreti lempučių F ir G grandinė užrašoma disjunkcija $F \vee G$.

Teiginys „lemputė F nedega“ užrašomas $\neg F$. Iš šių dviejų formulų sekā, kad lemputė G dega

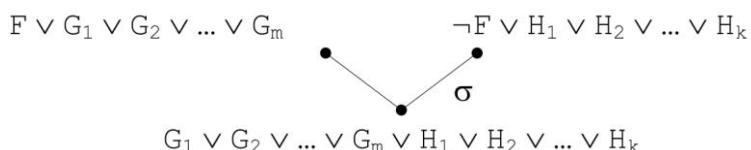
Kita rezoliucijos taisyklės forma, kai teigiamas ir neigiamas disjunktai sukeisti vietomis:

$$\frac{\neg F, \quad F \vee G}{F} \quad (19.9)$$

Bendru atveju rezoliucijos taisyklė užrašoma šitaip:

$$\frac{F \vee G, \quad \neg F \vee H}{G \vee H} \quad \{A_1/x_1, A_2/x_2, \dots A_n/x_n\} \quad \text{Rezoliucijos taisyklė (bendresnė)} \quad (19.10)$$

Rezolventė $G \vee H$ gaunama iš dviejų pradinių formulų pašalinant teigiamą ir neigiamą disjunktus. Ir G , ir H gali būti disjunkcija, pvz., $G_1 \vee G_2 \vee \dots \vee G_m$, kur $m \geq 0$. Iš F , G ir H įeinantys kintamieji x_i , jeigu tokiai yra, gali būti keičiami išraiškomis A_i (žr. 9.3 pav.) Keitinys σ turi pavidalą $\{A_1/x_1, A_2/x_2, \dots A_n/x_n\}$.



19.3 pav. Grafinis rezoliucijos taisyklės vaizdavimas. $m, k \geq 0$

19.1. *Irodymo remiantis rezoliucijos taisykle pavyzdys*

Toliau demonstruojami du įrodymai: *atbulinis* (kai įrodymas pradedamas nuo tikslų) ir *tiesioginis* (kai pradedama nuo duomenų).

Atbulinis įrodymas – nuo tikslų (tiksliau tikslų paneigimo) prie faktų. Šiame pavyzdyme tikslas yra įrodyti teoremą – teiginį „Žakas egzaminuoja Mari“, formaliai Egz (Žakas, Mari).

Įrodoma prieštaros būdu. Tegu profesorius Žakas negali egzaminuoti studentės Mari. Tai užrašoma kaip tikslų paneigimas:

$$\neg \text{Egz}(\text{Žakas}, \text{Mari}) \quad (19.11)$$

Imama ši priešinga prielaida (19.11) bei produkcija (19.7) ir taikoma rezoliucijos taisyklė (19.9). Siekiant neigiamą disjunktą $\neg \text{Egz}(\text{Žakas}, \text{Mari})$ suprastinti su teigiamu disjunktu Egz (y, w), keitinys yra $\{\text{Žakas}/y, \text{Mari}/w\}$.

$$\begin{array}{c} \neg \text{Egz}(\text{Žakas}, \text{Mari}), \quad \neg \text{Prof}(x, \text{Žakas}) \vee \neg \text{Stud}(z, \text{Mari}) \vee \text{Lygu}(x, z) \vee \text{Egz}(\text{Žakas}, \text{Mari}) \\ \hline \neg \text{Prof}(x, \text{Žakas}) \vee \neg \text{Stud}(z, \text{Mari}) \vee \text{Lygu}(x, z) \end{array}$$

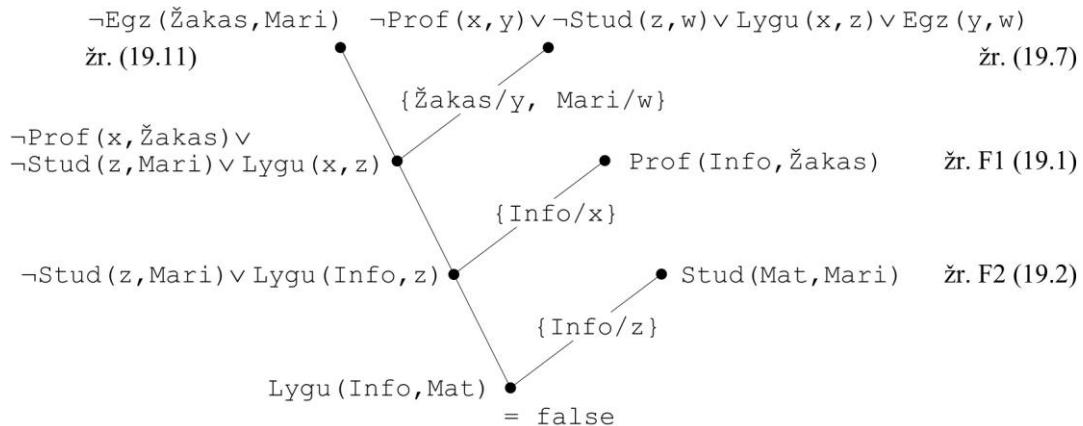
Toliau imama pastaroji rezolventė bei faktas F1 (19.1) ir taikoma rezoliucijos taisyklė (19.8). Keitinys $\{\text{Info}/x\}$.

$$\begin{array}{c} \neg \text{Prof}(\text{Info}, \text{Žakas}) \vee \neg \text{Stud}(z, \text{Mari}) \vee \text{Lygu}(\text{Info}, z), \quad \text{Prof}(\text{Info}, \text{Žakas}) \\ \hline \neg \text{Stud}(z, \text{Mari}) \vee \text{Lygu}(\text{Info}, z) \end{array}$$

Toliau imama pastaroji rezolventė bei faktas F2 (19.2) ir taikoma rezoliucijos taisyklė (19.9). Keitinys $\{\text{Mat}/z\}$.

$$\begin{array}{c} \neg \text{Stud}(\text{Mat}, \text{Mari}) \vee \text{Lygu}(\text{Info}, \text{Mat}), \quad \text{Stud}(\text{Mat}, \text{Mari}) \\ \hline \text{Lygu}(\text{Info}, \text{Mat}) \end{array}$$

Kadangi informatikos ir matematikos fakultetai yra skirtinių fakultetai, formaliai $\text{Lygu}(\text{Info}, \text{Mat}) = \text{false}$, tai gauta rezolventė yra false . Gaunama prieštara. Tokiu būdu, kadangi iš tikslų paneigimo išvedama prieštara, tai tikslas Egz (Žakas, Mari) yra teisingas. Įrodymo medis pateiktas 19.4 pav.



19.4 pav. Teoremos $Egz(\text{Žakas}, \text{Mari})$ atbulinio išvedimo medis: nuo tikslo paneigimo iki prieštaros gavimo

Atbulinio įrodymo pabaiga.

Tiesioginis įrodymas – nuo faktų prie tikslo. Duomenų bazėje yra faktai F1 (19.1) bei F2 (19.2) ir produkcija R1 (19.7). Reikia įrodyti $Egz(\text{Žakas}, \text{Mari})$.

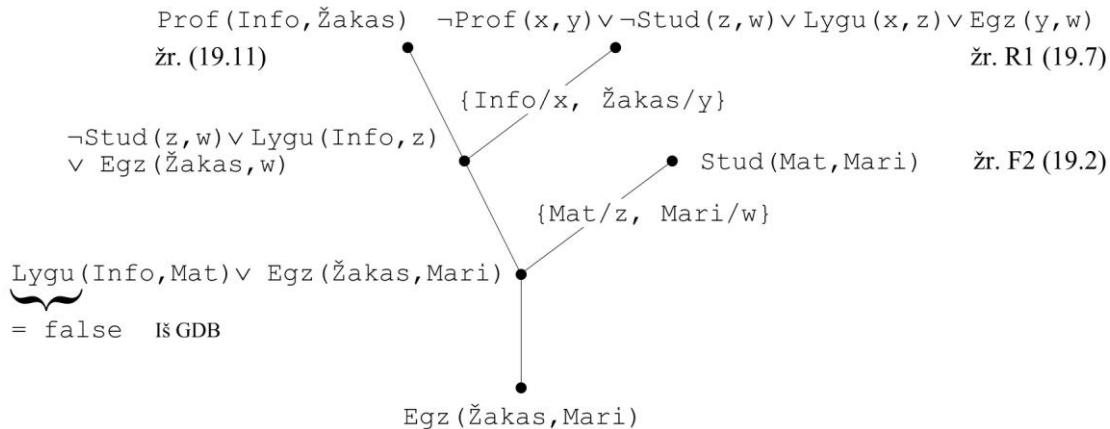
Imamas faktas F1 $\text{Prof}(\text{Info}, \text{Žakas})$ ir ieškoma produkcija, su kuria galima sutapatinti. Imama produkcija (19.7) ir taikoma rezoliucijos taisyklė (19.8) su keitiniu $\{\text{Info}/x, \text{Žakas}/y\}$:

$$\frac{\text{Prof}(\text{Info}, \text{Žakas}), \quad \neg \text{Prof}(\text{Info}, \text{Žakas}) \vee \neg \text{Stud}(z, w) \vee \text{Lygu}(\text{Info}, z) \vee \text{Egz}(\text{Žakas}, w)}{\neg \text{Stud}(z, w) \vee \text{Lygu}(\text{Info}, z) \vee \text{Egz}(\text{Žakas}, w)}$$

Toliau imamas faktas F2 $\text{Stud}(\text{Mat}, \text{Mari})$ ir pastaroji rezolventė. Taikoma rezoliucijos taisyklė (19.8) su keitiniu $\{\text{Mat}/z, \text{Mari}/w\}$:

$$\frac{\text{Stud}(\text{Mat}, \text{Mari}), \quad \neg \text{Stud}(\text{Mat}, \text{Mari}) \vee \text{Lygu}(\text{Info}, \text{Mat}) \vee \text{Egz}(\text{Žakas}, \text{Mari})}{\text{Lygu}(\text{Info}, \text{Mat}) \vee \text{Egz}(\text{Žakas}, \text{Mari})}$$

Kadangi $\text{Lygu}(\text{Info}, \text{Mat}) = \text{false}$ ir $\text{false} \vee H \equiv H$, tai išplaukia $\text{Egz}(\text{Žakas}, \text{Mari})$. Ką ir reikėjo įrodyti. Įrodymo medis pateiktas 19.5 pav.



19.5 pav. Teoremos $Egz(\text{Žakas}, \text{Mari})$ tiesioginio išvedimo medis – nuo faktų prie tikslo

Tiesioginio įrodymo pabaiga.

Įrodinėti galima tiek nuo duomenų, tiek nuo tikslo. Įrodymo esmė – su kuo tapatinti kintamuosius. Intelektas glūdi sekoje. Pati matematinė logika prie plano generavimo neprisideda. Logika yra priemonė plano sudarytojui (arba plano sudarymo programai).

19.2. *Pavyzdys: trys produkcinės taisyklės*

Demonstruojamas išvedimas pagal rezoliucijos taisyklę, naudojant 18 skyriuje įvestomis produkcijomis (18.1). Šios produkcijos užrašomos logikos formulėmis. Paskui remiantis implikacijos pašalinimo taisykle (19.5) suvedamos į *normaliąjį disjunkcinę formą*.

	Produkcija	Logikos formulė	Pašalinta implikacija
π_1 :	$F, B \rightarrow Z$	$F \& B \Rightarrow Z$	$\neg F \vee \neg B \vee Z$
π_2 :	$C, D \rightarrow F$	$C \& D \Rightarrow F$	$\neg C \vee \neg D \vee F$
π_3 :	$A \rightarrow D$	$A \Rightarrow D$	$\neg A \vee D$

Be šių trijų produkcijų dar yra trys faktai:

Faktas F1: A

Faktas F2: B

Faktas F3: C

Čia faktai suprantami kaip propoziciniai kintamieji, t.y. $A=true$, $B=true$ ir $C=true$. Reikia įrodyti Z .

Atbulinis įrodymas. Pradedama nuo tikslo paneigimo: $\neg Z$.

Ieškoma produkcija tapatinti. Tai π_1 . Toliau taikoma rezoliucijos taisyklė (19.9):

$$\frac{\neg Z, \quad \neg F \vee \neg B \vee Z}{\neg F \vee \neg B}$$

Gautai rezolventei $\neg F \vee \neg B$ ieškoma kita produkcija, su kuria galima tapatinti. Tai π_2 . Toliau taikoma rezoliucijos taisyklė (19.10):

$$\frac{\neg F \vee \neg B, \quad \neg C \vee \neg D \vee F}{\neg B \vee \neg C \vee \neg D}$$

Gautai rezolventei $\neg B \vee \neg C \vee \neg D$ ieškoma kita produkcija, su kuria galima tapatinti. Tai π_3 . Toliau taikoma rezoliucijos taisykla (19.10):

$$\frac{\neg B \vee \neg C \vee \neg D, \quad \neg A \vee D}{\neg B \vee \neg C \vee \neg A}$$

Kadangi globalioje duomenų bazėje yra faktai A, B ir C, t. y. $A=true$, $B=true$ ir $C=true$, tai išvedama false. Gaunama prieštara. Tai reiškia, kad tikslas yra teisingas.

Reikšmę false galima išvesti iš pastarosios rezolventės $\neg B \vee \neg C \vee \neg A$ taikant rezoliucijos taisykla (19.8) tris kartus. Tai parodoma tariant išvedimą.

Imamas faktas F1, t. y. A:

$$\frac{A, \quad \neg B \vee \neg C \vee \neg A}{\neg B \vee \neg C}$$

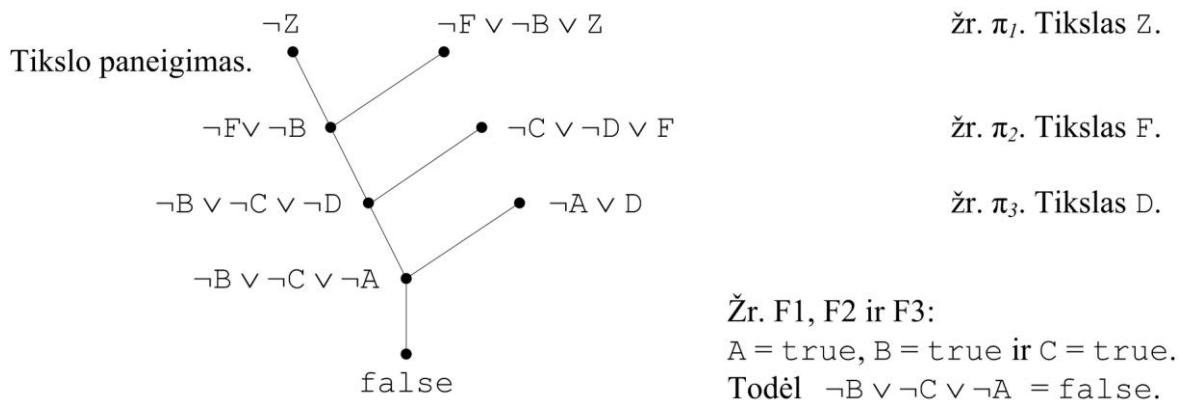
Imamas faktas F2, t. y. B:

$$\frac{B \quad \neg B \vee \neg C}{\neg C}$$

Imamas faktas F3, t. y. C:

$$\frac{C, \quad \neg C}{\emptyset}$$

Tuščias disjunktas reiškia prieštara. Tokiu būdu iš tikslo paneigimo $\neg Z$ gaunama prieštara. Tai reiškia, kad tikslas Z yra teisingas. Įrodymo medis pateiktas 19.6 pav.



19.6 pav. Teoremos Z atbulinio išvedimo medis: nuo tikslo paneigimo iki prieštaros

Atbulinio įrodymo pabaiga.

Dabar jau galima pastebėti, kad intelektas pasireiškia sutapatinimo sekoje π_1 , π_2 , π_3 . Gautas planas yra $\langle \pi_3, \pi_2, \pi_1 \rangle$.

Tiesioginis įrodymas. Pradedama nuo faktų.

Imamas faktas F1, t. y. A. Ieškoma produkcija, kurią bus galima pritaikyti pagal rezoliucijos taisykla. Tokia yra π_3 , t. y. $\neg A \vee D$. Taikoma rezoliucijos taisykla (19.8):

$$\frac{A, \quad \neg A \vee D}{D}$$

Ieškoma produkcija, kurią galima pritaikyti. Tai π_2 , t. y. $\neg C \vee \neg D \vee F$. Taikoma rezoliucijos taisyklė (19.8):

$$\frac{D, \quad \neg C \vee \neg D \vee F}{\neg C \vee F}$$

Imamas faktas F3, t. y. C. Taikoma rezoliucijos taisyklė (19.8):

$$\frac{C \quad \neg C \vee F}{F}$$

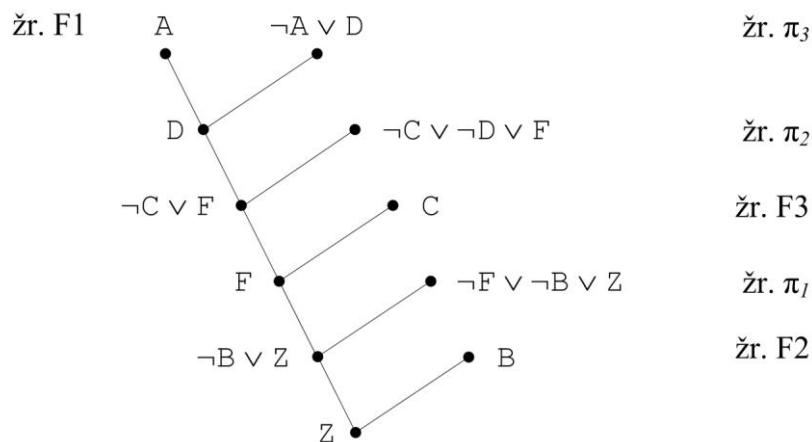
Ieškoma produkcija, kurią galima pritaikyti. Tai π_1 , t. y. $\neg F \vee \neg B \vee Z$. Taikoma rezoliucijos taisyklė (19.9):

$$\frac{F, \quad \neg F \vee \neg B \vee Z}{\neg B \vee Z}$$

Imamas faktas F2, t. y. B. Taikoma rezoliucijos taisyklė (19.9):

$$\frac{B, \quad \neg B \vee Z}{Z}$$

Gautas Z, ką ir reikėjo įrodyti. Įrodymo medis pateiktas 19.7 pav.



19.7 pav. Teoremos Z tiesioginio išvedimo medis: nuo faktų prie tikslų

Tiesioginio įrodymo pabaiga.

Kaip buvo minėta, intelektas pasireiškia sekos $\langle \pi_3, \pi_2, \pi_1 \rangle$ paieškoje.

19.3. Pavyzdys: rezoliucijos naudojimas teoremu įrodymui

Pateikiamas pavyzdys iš [Nilsson 1998, 16.5, 260–261] poskyrio „Using Resolution to Prove Theorems“. Tegu robotas žino, kad visi paketai 27 kambaryje yra mažesni negu bet kuris paketas 28 kambaryje. Tai užrašoma predikatų kalba:

$$(1) \forall x, y \text{ Package}(x) \& \text{Package}(y) \& \text{Inroom}(x, 27) \& \text{InroomL}(y, 28) \Rightarrow \text{Smaller}(x, y)$$

Sutrumpinus predikatų vardus ir eliminavus implikaciją gaunama disjunkcinė forma:

$$(2) \neg P(x) \vee \neg P(y) \vee \neg I(x, 27) \vee \neg I(y, 28) \vee S(x, y)$$

Tegu robotas žino, kad paketas A yra arba 27 arba 28 kambaryje (bet nežino kuriame), formaliai, $I(A, 27) \vee I(A, 28)$. Tegu dar žino, kad paketas B yra 27 kambaryje, $I(B, 27)$. Tegu dar žino, kad paketas B nėra mažesnis negu A, formaliai, $\neg S(B, A)$. Taigi:

(3) $P(A)$. Tai faktas.

(4) $P(A)$. Tai faktas.

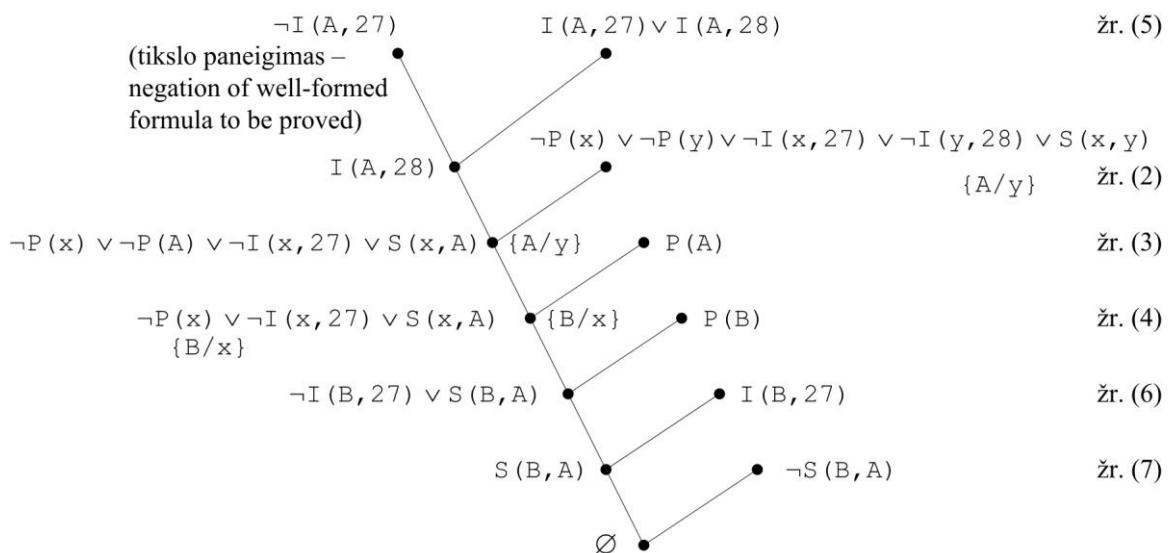
(5) $I(A, 27) \vee I(A, 28)$. Tai faktas, kurį žino robotas.

(6) $I(B, 27)$. Tai faktas, kurį žino robotas.

(7) $\neg S(B, A)$. Tai faktas, kurį žino robotas.

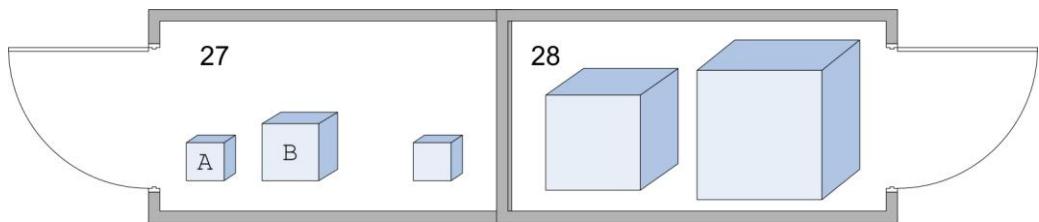
Robotas gali įrodyti, kad paketas A yra 27 kambaryje. Teorema – tikslas – yra $I(A, 27)$.

Rezoliucijos taisykla naudojama kiekvienam įrodymo žingsnyje. Pradedama nuo teoremos paneigimo: „Tegu $\neg I(A, 27)$ “ (žr. 19.8 pav.). Įrodymas vykdomas nuo viršaus į apačią. Dešinėje paveikslas pusėje pavaizduota, ką robotas žino. Pabaigoje tuščias disjunktas reiškia, kad gauta prieštara. Todėl teorema yra teisinga.



19.8 pav. Teoremos įrodymo medis (proof tree); žr. [Nilsson 1998, p. 261]

Galimas pasaulio modelis parodytas 19.9 pav.



19.9 pav. Galimas pasaulis, apie kurį žino robotas.

Pratimai

1. Duota produkcija:

$$\text{Mama}(X, Y) \ \& \ \text{Tétis}(Y, Z) \Rightarrow \text{Senelé}(X, Z)$$

ir du faktai:

„Adelé yra Broniaus mama“: Mama(Adelé, Bronius),

„Bronius yra Dainiaus tétis“: Tétis(Bronius, Dainius).

Irodykite tiesioginiu ir atbuliniu išvedimu teiginį:

„Adelé yra Dainiaus senelé“: Senelé(Adelé, Dainius).

20. Ekspertinės sistemos

Šis skyrius grindžiamas pavyzdžiu iš Brian Sawyer ir Dennis Foster vadovėlio (1986) anglų kalba ir jo vertimo į rusų kalbą.

Literatūroje pateikiami įvairūs *ekspertinės sistemos* (ES) (angl. *expert system*) apibréžimai. Ekspertinėje sistemoje saugomos tam tikros dalykinės srities specialistų – ekspertų – žinios ir remiantis jomis atliekamas loginis išvedimas atsakant į sistemai pateikiamus klausimus. Ekspertinė sistema apibūdinama kaip programų sistema, kurioje naudojami dirbtinio intelekto principai. Ekspertinė sistema skirta spręsti uždavinius, kuriems atliliki reikėtų dalykinės srities specialisto pagalbos. Tipiniai ekspertinių sistemų uždaviniai yra klasifikuojami šitaip: diagnostė, prognozė, konfigūravimas, sprendimų priemimas, planavimas ir kiti, kurie siejami su žmoniškuoju intelektu.

Terminas „ekspertinė sistema“ paplito prieš 20-30 metų, Jis išpopuliarėjo iš dalies dėl rinkodaros priežasčių. Šiuo metu populiарesnis sinonimas – *žiniomis grindžiama sistema* (angl. *knowledge-based system*). Pastarasis terminas prigijo su antros kartos ekspertinėmis sistemomis, kuriose visiškai buvo atskirta *žinių bazė* nuo *išvedimo mašinos*.

Terminas „*expert system shell*“ atitinka neužpildytą žiniomis ekspertinę sistemą, o „*expert system*“ – užpildytą. Tarp šių dviejų kraštutinių yra tarpiniai variantai.

Ekspertinių sistemų kūrimas susideda ne tik iš programinės ar aparatinės įrangos kūrimo, bet ir žinių bazės sudarymo, pasitelkiant atitinkamas srities ekspertus. Ekspertų turimas žinias reikia pavaizduoti programų sistemių suprantama kalba. Kartais sunku nustatyti, kokiais kriterijais remdamiesi specialistai daro atitinkamas išvadas.

Ekspertinės sistemos turi būti kuriamos taip, kad būtų paprasta jas tikrinti ir taisyti. Taip pat ekspertinė sistema turėtų pateikti paaiškinimą, kodėl buvo priimtas vienoks ar kitoks sprendimas. Norint išplėsti nagrinėjamą uždavinių aibę arba juos spręsti detaliau, turėtų pakakti papildyti žinių bazę naujomis žiniomis.

Bendroji ekspertinės sistemos architektūra yra pavaizduota 20.1 pav.



20.1 pav. Bendroji ekspertinės sistemos architektūra

Žinių bazė ir išvedimo mašina yra sudėtinės ekspertinės sistemos dalys. Kitos dalys mums nėra taip svarbios. Tokiu būdu 20.1 pav. pateikta supaprastinta bendrai priimta architektūra, žr. [Sawyer, Foster 1986]. Realiose ekspertinėse sistemoje naudotojai taip pat gali būti ekspertais, kurie užpilda ES žinių bazę *faktais* ir *produkcionis*. Gali būti išskiriamos dar smulkesnės ES sudėtinės dalys.

Pateiktoje architektūroje esminis yra žinių bazės ir išvedimo mašinos atskyrimas. Tai galime palyginti su duomenų bazių lentelių kaip duomenų atskyrimu nuo duomenų bazių valdymo sistemos (DBVS). Išvedimo mašina atitinka DBVS, o žinių bazė – DB lenteles. Atskyrimu siekiama, kad žinių bazę ir išvedimo mašiną galima būtų keisti nepriklausomai nuo viena kitos. Tačiau tai tėra siekiamybė, realiose sistemoje nepavyksta visiškai atskirti šiu dvių architektūros dalij. Čia galima palyginti su sunkumais, kurie kyla naudojant vieno gamintojo DBVS su kito gamintojo DB lentelėmis.

Ekspertinės sistemos žinių bazė nuo išprastos duomenų bazės skiriasi tuo, kad joje laikoma ne tik duomenys, bet produkcijos, kurių pagalba iš vienų duomenų išvedami kiti.

Toliau pateikiama demonstracinė ES iš [Sawyer, Foster 1986]. Pavyzdys iliustruoja žmogaus gyvenimo trukmės prognozę. Faktų bazę sudaro 20.2 pav. pateikti faktai. ES naudotojas pats pasirenka variantą, kuris 20.2 pav. vaizduojamas paryškintai

1) amžius:	2) lytis:	3) svoris:
1. 25_ir_mažiau	1. vyras	1. 55_ir_mažiau
2. 25-55	2. moteris	2. 55-85
3. 55_ir_daugiau		3. 85_ir_daugiau
4) sudėjimas:	5) cholesterolis:	6) druska:
1. smulkus	1. vidutinis	1. norma
2. stambus	2. daug	2. daug
7) rūko:	8) charakteris:	9) alkoholis:
1. taip	1. agresyvus	1. nevaroja
2. ne	2. švelnus	2. vidutiniškai
		3. daug

20.2 pav. Demonstracinės ekspertinės sistemos faktai. Naudotojo pasirinkimas paryškintas

Objektų reikšmės interpretuoojamos kaip simbolių eilutes, pvz., amžius = '25-55'

Žmogus, sudarantis reikšmių rėžius turi išmanyti dalykinę sritį. Žmogus, sudarantis žinių bazę neformaliai vadinas žinių inžinieriumi (angl. *knowledge engineer*).

20.3 lentelėje pavaizduotos produkcijos kurios yra žinių bazėje ir naudojamos gyvenimo trukmės išvedimo medyje. [Sawyer, Foster 1986] pavyzdyje yra apie 100 produkcijų.

Pr.1	Jeigu santykinis svoris normalus, tai bendras įvertinimas „taip“.
Pr.4	Jeigu širdies sustojimo rizika yra vidutinė, tai širdies pavojus žemas.
Pr.8	Jeigu amžius 25-55 metai ir lytis moteris, tai bendra gyvenimo trukmė 67 metai.
Pr.12	Jeigu svoris ne didesnis nei 55 kilogramai, sudėjimas smulkus ir lytis moteris, tai santykinis svoris yra normalus.
Pr.27	Jeigu cholesterolio suvartojama vidutiniškai, tai širdies sustojimo rizika vidutinė.
Pr.29	Jeigu druskos suvartojimas yra normos ribose, tai kraujo spaudimas vidutinis.
Pr.37	Jeigu bendras įvertinimas yra „taip“, širdies pavojus yra žemas, kraujo spaudimas yra vidutinis ir nerūko, tai perspektyva gera.
Pr.43	Jeigu charakteris agresyvus, tai asmenybės tipas A.
Pr.45	Jeigu asmenybės tipas A, tai rizikos lygis yra aukštas.
Pr.47	Jeigu alkoholio vartoja vidutiniškai, tai papildomas faktorius yra geras.
Pr.52	Jeigu perspektyva yra gera, rizikos lygis yra aukštas ir papildomas faktorius yra geras, tai faktorius yra nulis.

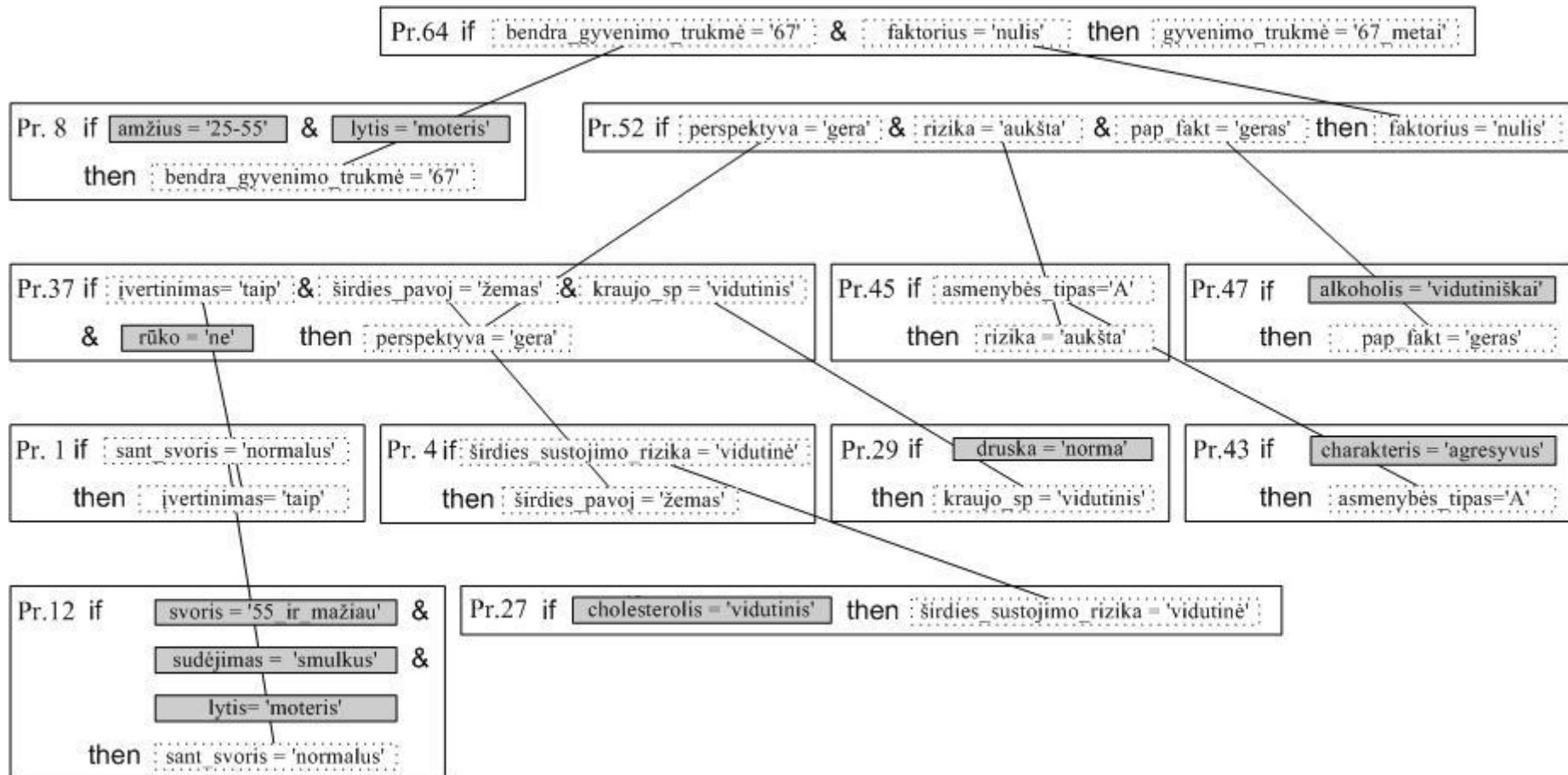
Pr.64

Jeigu bendra gyvenimo trukmė yra 67 metai ir faktorius yra nulis, tai gyvenimo trukmė yra 67 metai.

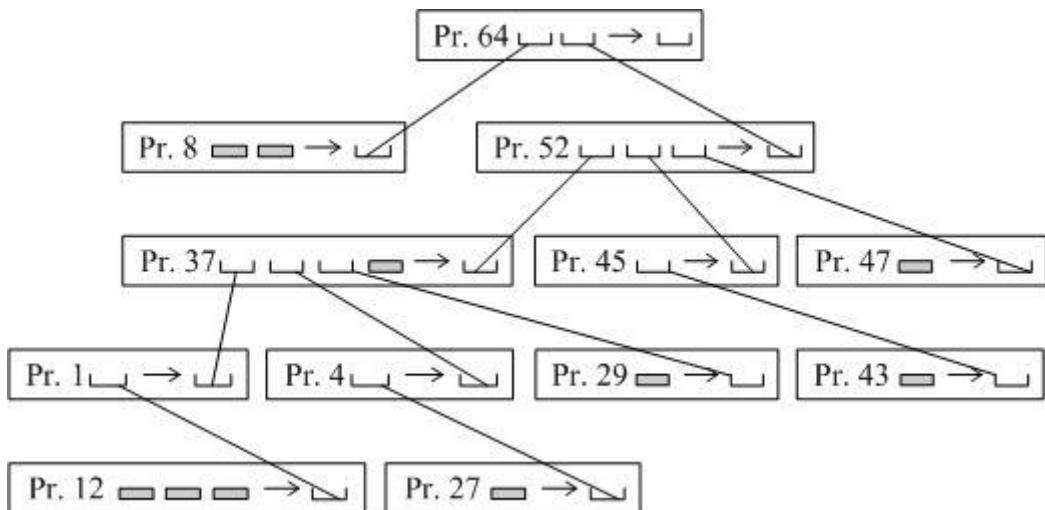
20.3 lentelė. Demonstracinės ekspertinės sistemos produkcijos, kurios naudojamos išvedimo medyje 20.4 pav.

Remdamasi pateiktais faktais ir produkcijomis ekspertinė sistema pateiks atsakymą, kad šiam konkrečiam žmogui prognozuojama gyvenimo trukmė yra 67 metai. Prognozės išvedimo medis pateiktas pateiktas 20.4 pav. Jeigu faktas yra žinomas, tai jis priklauso žinių bazei, tiksliau, faktų bazei.

Loginis išvedimas atliekamas atbulinio išvedimo algoritmu (angl. *backward chaining*). Tikslo objektas yra kintamasis gyvenimo_trukmė.



20.4 pav. Paieškos medis demonstracinėje ekspertinėje sistemoje. Pilki stačiakampiai žymi pradinius faktus. Jie toliau nėra išvedinėjami – tai lapai



20.5 pav. Supaprastintas išvedimo medis vaizduojantis išvedimą 20.4 pav. Pilki stačiakampiai reiškia faktus, esančius DB

Demonstracinė ekspertinė sistema ieško jai pateikto teiginio kaip tikslų įrodymo. Jeigu faktų įrodymui nepakanka, tai ekspertinė sistema užduoda papildomus klausimus ir tokiu būdu interaktyviai papildyti žinių bazę naujais faktais. Pavyzdžiui:

Sistema: Koks asmens charakteris?

1. agresyvus
2. švelnus

Naudotojas: agresyvus

Išvedimo algoritmas yra analogiškas atbuliniams išvedimui produkcijų sistemoje (žr. 18 skyrių „Tiesioginis ir atbulinis išvedimas produkcijų sistemoje“). Šio algoritmo sudėtingumas yra polinominis. Tokiu būdu jis yra mažesnio sudėtingumo negu eksponentinis išvedimas, kurį naudoja Prolog kalbos interpretatorius. Polinominio o ne eksponentinė sudėtingumą salygoja tai, kad produkcijose yra $o_i=r_i$ pavidalo faktai, o ne predikatai, pvz., $P_i(x,y)$. Kaip minėta, produkcijos yra tokio pavidalo:

$$o_1=r_1, o_2=r_2, \dots, o_n=r_n \rightarrow o_{n+1}=r_{n+1}$$

arba kitaip žodžiais:

$$\text{if } o_1=r_1 \ \& \ o_2=r_2 \ \& \ \dots \ \& \ o_n=r_n \text{ then } o_{n+1}=r_{n+1}$$

Termas $o_i=r_i$ yra suprantamas kaip teiginys „objekto o_i reikšmė yra lygi r_i “. Kiekvienas objektas o_i turi baigtinę reikšmių aibę $\{r_{i1}, r_{i2}, \dots, r_{im}\}$. Prologe produkcija turi, pavyzdžiui, tokį pavidala:

$$P_1(x,y) \ \& \ P_2(x) \ \& \ P_3(y) \rightarrow P_4(x,y)$$

Predikato $P_i(x,y)$ kintamieji tapatinami su termais, kurių aibė yra ne baigtinė, o begalinė.

Demonstracinės ekspertinės sistemos produkcijų aibė yra realizuota sąrašu, kurio elementas yra produkcija. Savo ruožtu šios produkcijos kairioji pusė yra porą $o_i=r_i$ sąrašas. Faktų bazė taip pat vaizduojama porą $o_i=r_i$ sąrašu.

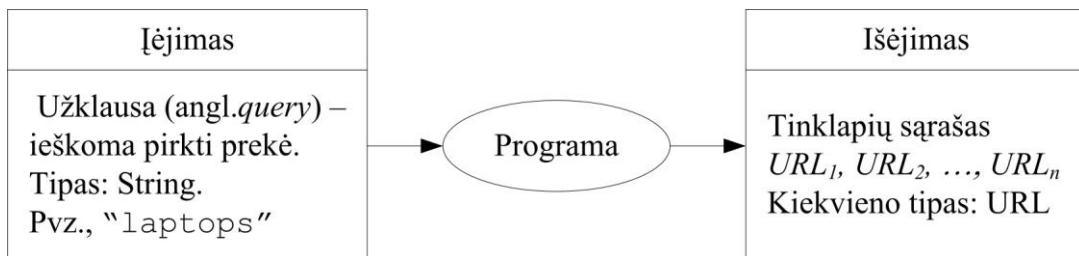
Naudotojo interfeiso realizacijos ypatybė yra tai, kad objekto duomenų struktūroje saugomas ir klausimas, kuris yra simbolų eilutė. Ši klausimą ekspertinė sistema užduoda naudotojui, kai jai nepasiseka išvesti šio objekto reikšmės, ir laukia atsakymo. Taigi, jeigu išvedimo metu kuriam nors konsekventui nepavyksta rasti tinkamo antecedento, tai antecedentas užklausiamas naudotojo. Tokiu būdu faktai gali būti surenkami programos vykdymo metu, o ne iš anksto.

Ekspertinė sistema (sinonimas – žiniomis grindžiama sistema) nuo informacinės sistemos skiriasi duomenų vaizdavimo būdu. Ekspertinėje sistemoje naudojamas *gilus* (angl. *deep*) žinių vaizdavimas. Informacinėje sistemoje naudojamas *plokščias* (angl. *flat*) duomenų vaizdavimas, pvz., lentelės duomenų bazių valdymo sistemoje.

21. Internetinė parduotuvė

Mes dėstome vadovaudamiesi [Russell, Norvig 2003, p. 344-348] skyriaus „Žinių vaizdavimas“ poskyriu, kuriame nagrinėjama internetinė parduotuvė. Reikalavimų analizė atliekama žinių vaizdavimo tikslu.

Nustatykime sąvokas, reikalingas internetinės parduotuvės programai. Pradėkime nuo programos įėjimo ir išėjimo. Programos įėjimas yra užklausa apie ieškomą pirkti prekę. Užklausos tipas yra String. Programos išėjimas yra sąrašas $URL_1, URL_2, \dots, URL_n$ tinklapių, kuriuose parduodama ieškoma prekė (žr. 21.1 pav.).



21.1 pav. Internetinės parduotuvės programos įėjimas ir išėjimas

Pavyzdžiui, pateikiama užklausa „laptops“ ir gaunamas sąrašas tokių tinklapių, kuriuose parduodami nešiojami kompiuteriai.

Programa suprantama kaip kompiuterinis agentas [Russell, Norvig 2003], kuris naršo po internetą ir ieško vartotojo nurodytos prekės (žr. 21.2 pav.).



21.2 pav. Kompiuterinio agento įėjimas ir išėjimas

Žmogus skiriasi nuo kompiuterinio agento:

- žmogus mato tinklapio vaizdą, asocijuoja „laptops“ su kompiuteriais ir žiūrėdamas į tinklapį sugeba padaryti išvadą, ar tinklapyje yra ieškoma prekė, ar nėra.
- kompiuteris tinklapio vaizdo „nemato“, o operuoja tik HTML tekstu. Jeigu tinklapis vaizduotų prekių kategorijas ne žodžiais, o piktogramomis, tai tik žmogus sugebėtų atskirti, kurioje kategorijoje gali būti norimos prekės.

Skirtingas žmogaus ir kompiuterio gebėjimas apdoroti vaizdą taikomas interneite atskirti žmogui nuo programos. Pavyzdžiui, prašoma atpažinti slaptažodį, kuris pateikiamas vaizde kaip iškraipytas tekstas CAPTCHA (angl. *completely automated public Turing test to tell computers and humans apart*). Žmogus slaptažodį atpažista, o kompiuteris ne.

Toliau apsiribojama tinklapiais, kuriuose nuspresti apie prekę galima iš HTML teksto. Žemiau pateikiamas tipinis tinklapis, kuriame gali būti siūloma pirkti nešiojamus kompiuterius (žr. 21.3 pav.).

Generic Online Store

Select from our fine line of products

- [Computers](#)
- [Cameras](#)
- [Books](#)
- [Videos](#)
- [Music](#)

```
<html>
<body>
<h1>Generic Online Store</h1>
<i>Select</i> from our fine line of products
<ul>
<li><a href="http://www.gen-store.com/compu">Computers</a>
<li><a href="http://www.gen-store.com/camer">Cameras</a>
<li><a href="http://www.gen-store.com/books">Books</a>
<li><a href="http://www.gen-store.com/video">Videos</a>
<li><a href="http://www.gen-store.com/music">Music</a>
</ul>
</body>
</html>
```

21.3 pav. Tipinis tinklapis, kuri žmogus suvokia kaip internetinę parduotuvę.
Apačioje HTML tekstas. Mes kursime parduotuvę adresu
<http://www.gen-store.com> ir pavadinsime GenStore

Žiūrėdamas į tinklapį, parodytą 21.3 pav., žmogus gali padaryti išvadą, šiame tinklapyje siūloma pirkti nešiojamus kompiuterius. Programai tokią išvadą padaryti sunkiau, bet įmanoma. Tam reikia susieti simbolių eilutę „Computers“ su nešiojamais kompiuteriais (tegu programai pateikiama užklausa „laptops“).

Toliau reikalavimai internetinės parduotuvės programai pateikiami „tikslinimo nuo viršaus link apačios“ metodu (angl. *top-down refinement*).

I etapas. Predikatas RelevantOffer

Tinklapis su HTML tekstu `page` yra relevantinis pasiūlymas užklausai `query` tada ir tik tada, kai 1) šis tinklapis yra relevantinis užklausai, ir 2) tame yra siūloma pirkti. Pavyzdžiu, tinklapis yra nerelevantinis, jeigu tame kalbama apie baldus arba pateikiama nešiojamų kompiuterių apžvalga. Formaliai, reikalavimas tinklapiui būti relevantišku pasiūlymu išreiškiamas tokiu predikatu:

RelevantOffer(`page`, `url`, `query`) \Leftrightarrow
Relevant(`page`, `url`, `query`) & **Offer**(`page`)

Čia kintamieji yra šių tipų: `page` – HTML tekstas, `url` – URL, `query` – String. Tipas URL yra String poaibis.

2 etapas. Predikatas **Offer**

Toliau predikatas **Offer** detalizuojama galimybę pirkti. Tinklapyje page yra pasiūlymas pirkti tada ir tik tada, kai HTML žymenyje „a“ arba „form“ sutinkama simbolių eilutė „pirkti“ (angl. *buy*) arba „kaina“ (angl. *price*):

$$\begin{aligned}\mathbf{Offer}(\text{page}) &\Leftrightarrow (\mathbf{InTag}('a', \text{str}, \text{page}) \vee \mathbf{InTag}('form', \text{str}, \text{page})) \\ &\quad \& (\mathbf{In}('buy', \text{str}) \vee \mathbf{In}('price', \text{str})) \\ \mathbf{InTag}(\text{tag}, \text{str}, \text{page}) &\Leftrightarrow \mathbf{In}('<' + \text{tag} + \text{str} + '</' + \text{tag}, \text{page}) \\ \mathbf{In}(\text{sub}, \text{str}) &\Leftrightarrow \exists i \text{ str}[i:i+\text{Length}(\text{sub})-1] = \text{sub}\end{aligned}$$

Čia predikatas $\mathbf{In}(\text{sub}, \text{str})$ išreiškia, kad simbolių eilutė sub įeina į simbolių eilutę str , o simbolis „+“ žymi eilučių konkatenaciją. Pavyzdžiui:

$$\begin{aligned}\mathbf{In}('KAD', 'ABRAKADABRA') &= \text{true} \\ \mathbf{In}('laptop', 'ABRAKADABRA') &= \text{false} \\ 'ABC' + 'DEFG' &= 'ABCDEFG'\end{aligned}$$

Šiame pavyzdyje pasiūlymas pirkti užkoduotas labai supaprastintai – žodžiai 'buy' ir 'price'. Pirma, 'buy' arba 'price' sutinkamas žymenyje ir, antra, tas žymuo sutinkamas tinklapyje page. Reikia pastebeti, kad ši specifikacija nėra pakankamai detali. Pavyzdžiui, ar turi prasmę didžiosios ir mažosios raidės eilutėse 'Buy' ir 'Price', asmenavimas, daugiskaita ir pan. Rimta detalizacija turi apimti kuo daugiau žinių apie sąvoką **Offer**, išskaitant ir morfologinę žodžių darybą.

Toliau etapais detalizuojamas predikatas **Relevant**.

3 etapas. Predikatas **OnlineStores** (store)

Predikatas nusako, ar identifikatorius store žymi internetinę parduotuvę. Mūsų paieškos strategija yra perrinkti visus tinklapius, pasiekiamus einant nuorodomis iš titulinio puslapio. Tinklapis page yra relevantinis užklausai query, tada ir tik tada, kai jis pasiekiamas nuorodų grandine iš mūsų kuriamos parduotuvės GenStore titulinio puslapio. Žinomas prekyvietės, tokios kaip Amazon, Ebay ir mūsų kuriama GenStore, tenkina predikatą **OnlineStores**, nes yra internetinės prekyvietės su žinomais interneto adresais.

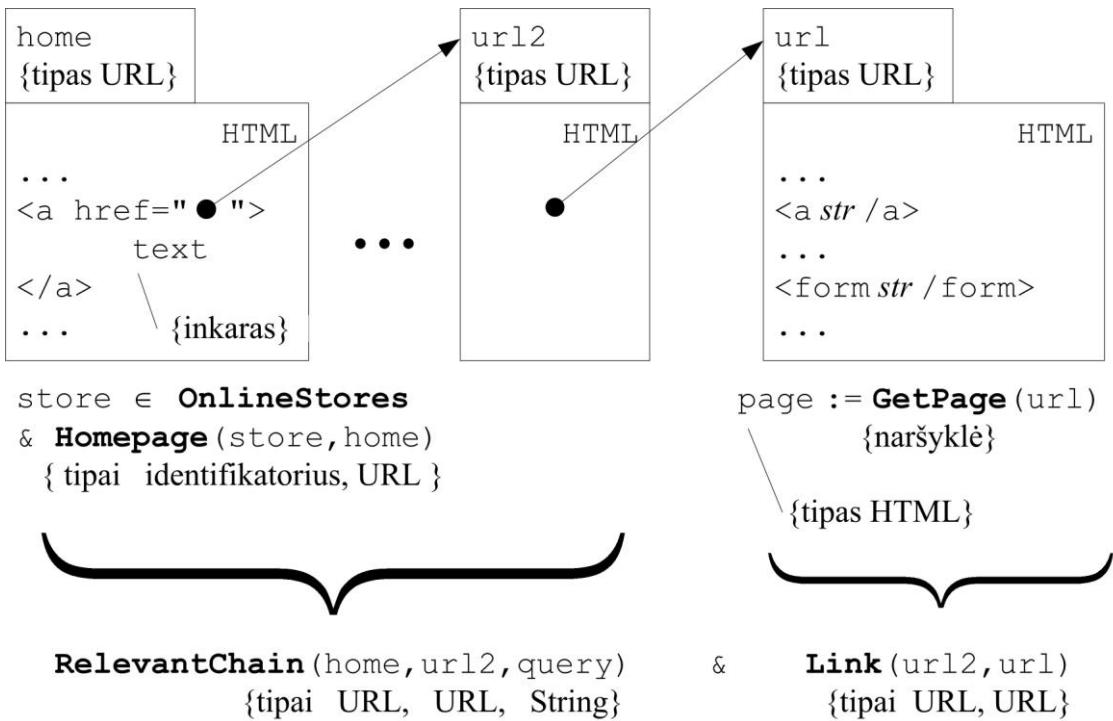
Kuriama elektroninė parduotuvė turi žinias apie tam tikras prekyvietes. Pavyzdžiui:

$$\begin{aligned}\text{Amazon} &\in \mathbf{OnlineStores} \\ &\& \mathbf{Homepage}(\text{Amazon}, 'http://www.amazon.com') \\ \text{Ebay} &\in \mathbf{OnlineStores} \\ &\& \mathbf{Homepage}(\text{Ebay}, 'http://www.ebay.com') \\ \text{GenStore} &\in \mathbf{OnlineStores} \\ &\& \mathbf{Homepage}(\text{GenStore}, 'http://www.gen-store.com')\end{aligned}$$

Čia vardai Amazon, Ebay, GenStore žymi individualias sąvokas. Jų tipas yra „identifikatorius“. Šiais trimis faktais pasakoma, kad pasaulyje yra tinklapiai, kurie yra prekyvietės.

4 etapas. Predikatas **Relevant**

Kuriamos parduotuvės idėja parodyta 21.4 pav. Stačiakampyje kairėje text tarp HTML žymenų „a“ yra vadinamas *inkaru* (angl. *anchor*). Inkarai matomi kompiuterio ekrane (žr. 21.3 pav.), pvz. „Computers“, „Books“. Inkarai sutinkami ir kituose tinklapiuose 21.4 pav.



21.4 pav. Aiškinama kuriamos parduotuvės specifikacija. Pavyzdžiui,
home = 'http://www.gen-store.com' ir store = GenStore

Toliau detalizuojamas predikatas **Relevant**:

Relevant(page, url, query) \Leftrightarrow
 $\exists \text{store } \exists \text{home } (\text{store} \in \text{OnlineStores} \& \text{Homepage}(\text{store}, \text{home}) \& \exists \text{url2 } \text{RelevantChain}(\text{home}, \text{url2}, \text{query}) \& \text{Link}(\text{url2}, \text{url}) \& \text{page} = \text{GetPage}(\text{url}))$ (21.1)

Kuriamos GenStore programos specifikacija (žr. 21.4 pav.) yra pradėti nuo titulinio tinklapio home ir eiti nuorodomis per internetą ieškant relevantinių tinklapių su siūlymais pirkti. Čia store tipas identifikatorius, home tipas URL.

Detalizuojant **Relevant** įvestas predikatas **Link**(from_url, to_url), kur from_url ir to_url tipas yra URL. Predikatu **Link** išreiškiama, kad tinklapyje from_url yra nuoroda į to_url tinklapį (žr. 21.4 pav.).

Tokiu būdu titulinis tinklapis yra home ir tikslas tinklapis – url. Priskyrimas page := **GetPage**(url) žymi, kad page bus priskirtas HTML tekstas esantis tinklapyje adresu url. Šis priskyrimas gali būti palygintas su naršyklės iškvietimu.

Predikatu **Homepage**(store, home) išreiškiama, kad prekyvietė vardu store turi titulinį tinklapį home.

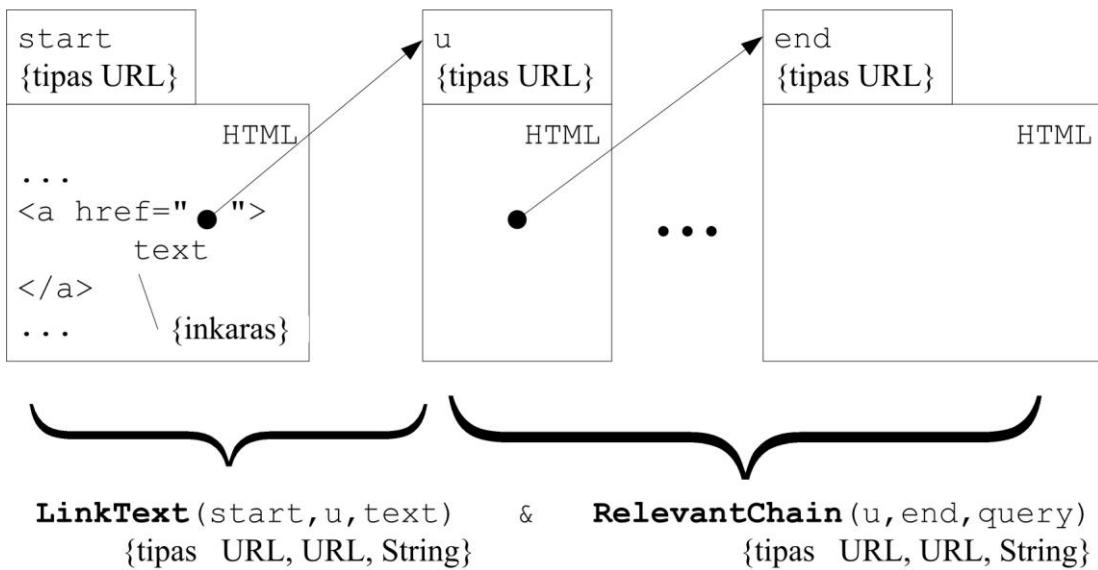
Taigi, perrenkami visi relevantiniai tinklapiai url, pasiekiami einant nuorodomis iš turimų prekyviečių store ∈ **OnlineStores** titulinio tinklapio home. Predikatas **offer** iš šių relevantinių tinklapių atsijoja tik tuos, kuriuose siūloma pirkti.

5 etapas. Predikatas **RelevantChain**

Detalizuojamas predikatas **RelevantChain** (*home, url2, query*); žr. (21.1) ir 21.5 pav. Iš esamo tinklapio start, kuriame esame, galima pereiti prie tinklapio u tik tada, kai start inkaras *text* (t. y. tekstas tarp HTML žymenų „a“) yra relevantinis užklausai *query*. Ir taip toliau rekursyviai. Grandine relevantinių nuorodų, perrenkami tinklapiai end.

$$\begin{aligned} \mathbf{RelevantChain}(\text{start}, \text{end}, \text{query}) &\Leftrightarrow (\text{start} = \text{end}) \\ &\vee (\exists u \exists \text{text} \mathbf{LinkText}(\text{start}, u, \text{text}) \\ &\quad \& \mathbf{RelevantCategoryName}(\text{query}, \text{text}) \\ &\quad \& \mathbf{RelevantChain}(u, \text{end}, \text{query})) \end{aligned} \quad (21.2)$$

Čia start tipas URL, end tipas URL, query tipas String.



21.5 pav. Predikatas **RelevantChain** (*start, end, query*)

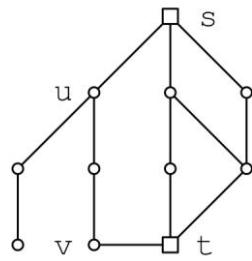
Pavyzdžiui, *query='laptops'* ir *start* yra Amazon titulinis tinklapis <http://www.amazon.com> arba mūsų GenStore titulinis <http://www.gen-store.com>. Tada iš *start* HTML teksto yra perrenkamos nuorodos į kitus tinklapius u. Predikatas **RelevantCategoryName**(*query, text*) paima tik tas nuorodas, kurių inkaras *text* relevantinis *query*. GenStore atveju, tai tik viena nuoroda <http://www.gen-store.com/compu>, nes iš penkių inkarų, tik 'Computers' yra relevantinis užklausai 'laptops' (žr. 21.3 pav.). Toliau rekursyviai ieškomi tinklapiai end iš <http://www.gen-store.com/compu>.

Verta priminti, kad specifikacijos rekursyviais predikatais yra naudojamos ir teorinėje informatikoje, ir dirbtiniame intelekte. Pavyzdžiui, tiesioginės paieškos (*forward chaining*) grafe (žr. 21.6 pav.) nuo pradinės viršunės s iki terminalinės t specifikacija yra tokia:

$$\text{ForwardChaining}(s, t) \Leftrightarrow \text{Link}(s, u) \& \text{ForwardChaining}(u, t)$$

Atbulinės paieškos (*backward chaining*) specifikacija:

`BackwardChaining(s, t) \Leftrightarrow BackwardChaining(s, v) & Link(v, t)`



21.6 pav. Paieška grafe nuo s iki t gali būti atliekama ir tiesioginės paieškos, ir atbulinės paieškos algoritmai

6 etapas. Žodžių priskyrimas kategorijoms

Iš pradžių aptarsime žodžių, kurie gali įeiti į užklausą, kategorizavimo problemą.

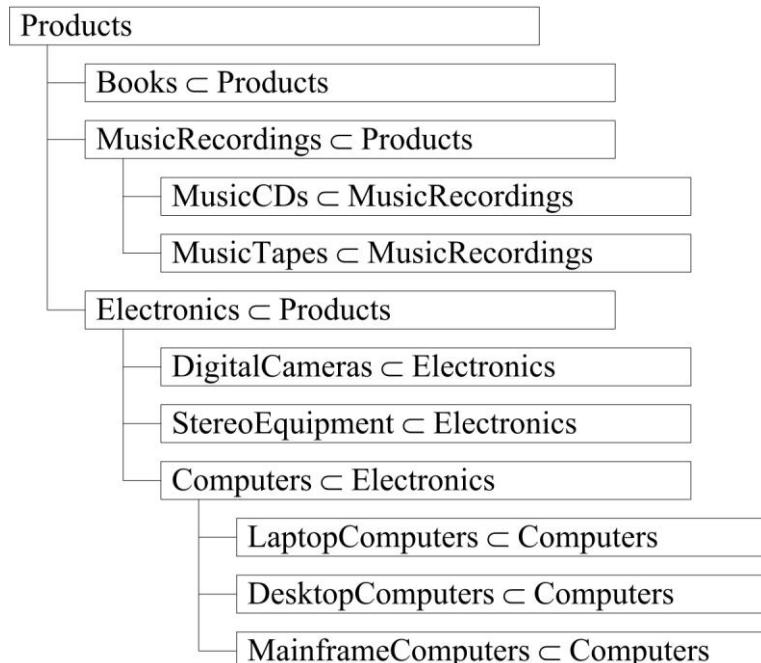
Galima pageidauti, kad užklausos forma būtų neformalizuota, pavyzdžiu `query = 'Man reikalingi laptopai'`. Žmogus gali užduoti užklausą ir kitais žodžiais, nes nežino kategorijų, kurias išreiškia inkaras `text`. Be to, vienas žodis gali būti priskiriamas kelioms kategorijoms. Pavyzdžiu, „lapiukas“ kategorizuojant gali būti siejamas su žvėreliu lape, nedideliu popieriaus lapu arba medžio lapu.

Primename kontekstą iš predikato **RelevantChain**(`start, end, query`) (21.2):

LinkText(`start, u, text`) & **RelevantChain**(`u, end, query`)

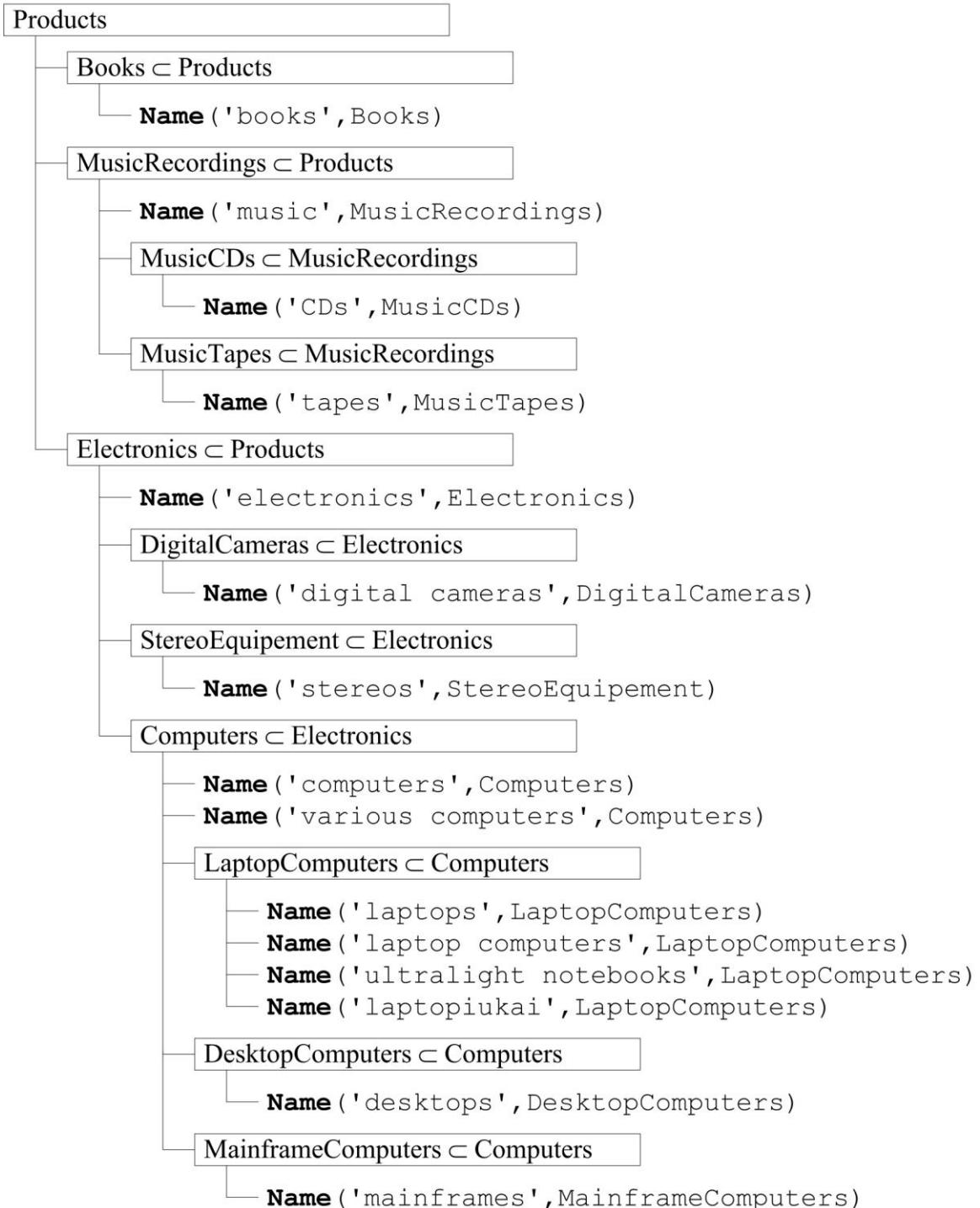
Čia, pvz., `text` reiksmė 'Computers', o `query` – 'Man reikalingi laptopai'.

Kategorizavimui naudojami du medžiai: 1) produktų kategorijų hierarchija ir 2) žodžių priskyrimo kategorijoms medis. Produktų kategorijų medžio pavyzdys parodytas 21.7 pav. Jo briaunos žymi ryšį *is-a*.



21.7 pav. Produktų kategorijų hierarchija. Medžio briaunos žymi ryšį *is-a*

Žodžiai, kurie įeina į užklausą query, yra priskiriami kategorijoms (žr. 21.8 pav.). Natūralioje kalboje žodžių yra daug – dešimtys tūkstančių. Kategorijų taip pat gali būti daug. Pavyzdžiu, tūkstančiai kategorijų yra muitinės tarifų lentelėse, rengiamose duomenų baze ir knyga. Kategorijas gali įvardinti ir žodžių junginiai, pvz., 'ultralight notebooks'.



21.8 pav. Žodžių priskyrimas kategorijoms

Pavyzdžiu, žodis „book“, atitinka sąvoką (kategoriją) Books. Tai išreiškiama predikatu **Name** ('book', Books). Žargono arba mažybiniai žodžiai taip pat priskiriami

kategorijoms, pvz., **Name** ('laptopiukai', LaptopComputers). Išvystyta programa taip pat turėtų suprasti linksnius ir kitokią žodžių darybą.

Vienas žodis gali būti priskiriamas kelioms kategorijoms, pvz., 'lapiukas'. Tokiu būdu atkreipiamas dėmesys į dviprasmiškumo problemą kategorizuojant žodžius. Pavyzdžiui, „CDs“ gali būti priskiriamas, ne tik 'MusicCDs', bet ir **Name** ('CDs', CertificatesOfDeposit).

Predikatu **Name** yra pasirenkama, kad žodis siejamas su labiau specifine arba bendresne kategorija. Pavyzdžiui, **Name** ('laptops', LaptopComputers) arba **Name** ('laptops', Computers).

7 etapas. Predikatas **RelevantCategoryName** (query, text)

Buvo iškeltas klausimas, kaip susieti užklausą query su inkaru text (produktų kategorijomis). Tegu query='laptops'. Tada predikatas teisingas vienu iš trijų atvejų:

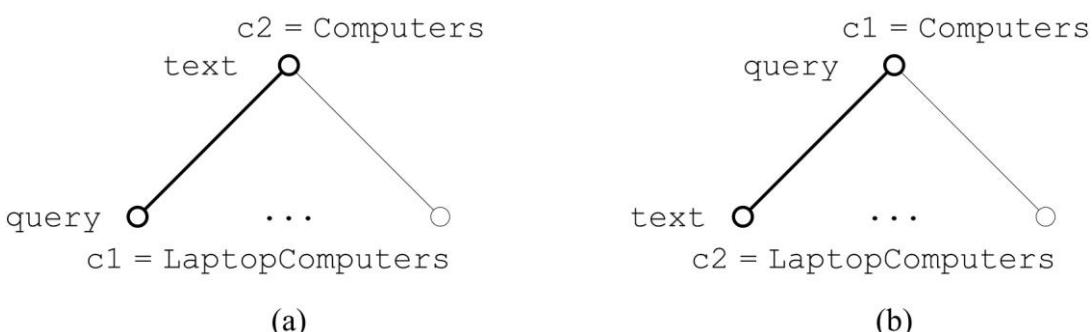
1. text ir query įvardina tą pačią kategoriją, pvz., text='laptop computers' ir 'laptops';
2. text įvardina bendresnę kategoriją, pvz. text='computers' (žr. 21.9 pav);
3. text įvardina labiau specifinę kategoriją, pvz. text='ultralight notebooks' (žr. 21.10 pav.).

Predikatas apibrėžiamas šitaip:

RelevantCategoryName (query, text) \Leftrightarrow

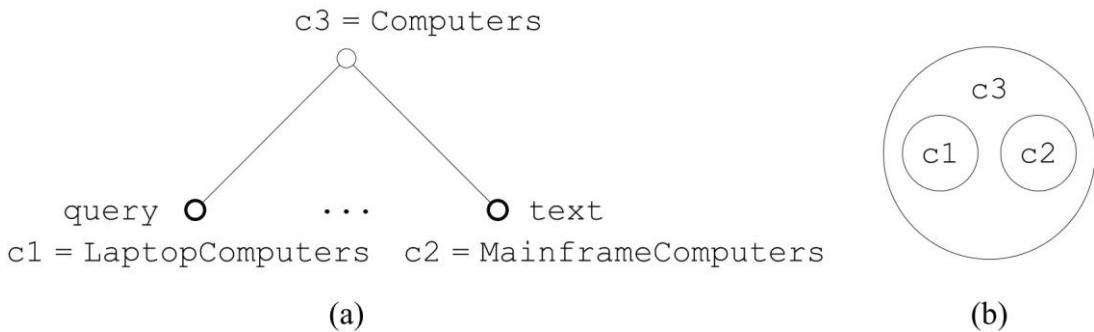
$\exists c_1 \exists c_2 \text{ Name}(\text{query}, c_1) \& \text{Name}(\text{text}, c_2) \& (c_1=c_2 \vee c_1 \subset c_2 \vee c_1 \supset c_2)$

Atvejis $c_1 \subset c_2$ pavaizduotas 21.9 a pav., o atvejis $c_1 \supset c_2$ 21.9 b pav.



21.9 pav. (a) Atvejis $c_1 \subset c_2$. (b) Atvejis $c_1 \supset c_2$.

Predikatas netenkinamas, kai nėra nei $c_1=c_2$, nei $c_1 \subset c_2$, nei $c_1 \supset c_2$. Tokiu atveju sakoma, kad kategorijos c_1 ir c_2 yra nepalyginamos. Pavyzdys yra, kai c_1 ir c_2 yra labiau specifinės kategorijos negu bendresnė kategorija c_3 . Pavyzdžiui $c_1 = \text{LaptopComputers}$, $c_2 = \text{MainframeComputers}$ ir $c_3 = \text{Computers}$ (žr 21.10 a pav.). Kategorijos c_1 ir c_2 kaip aibės yra aibės c_3 poaibiai, bet nesikerta (21.10 b pav.).



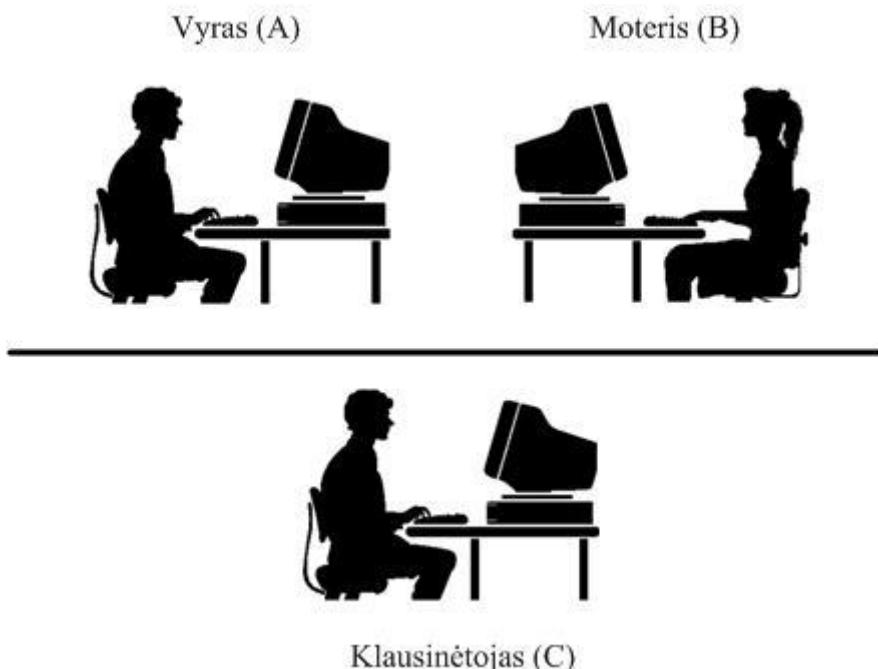
21.10 pav. (a) Predikatas netenkinamas, kai nėra nei $c_1=c_2$, nei $c_1 \subset c_2$, nei $c_1 \supset c_2$. Kategorijos c_1 ir c_2 nepalyginamos.

Užklausa gali būti, pavyzdžiu, „kompiuteris, telpantis ant atlenkiamo staliuko ekonominės klasės vietoje lėktuve Boeing 737“. Išvystyta programa turi apdoroti natūralią kalbą. Kaip matyti, iš anksto sunku numatyti visus atvejus, kai tenkinamas predikatas **Name**.

22. Tiuringo testas

Tiuringo testo samprata yra aptariama ir vadovėliuose [Luger 2005] ir interne (žr. http://en.wikipedia.org/wiki/Turing_test).

Alanas Tiuringas [Turing 1950] aptaria galimybę atsakyti į klausimą „Ar gali mašinos mąstyti?“. Šio straipsnio pradžioje jis teigia, jog iškeltas klausimas yra dviprasmiškas, todėl bando ji konkretizuoti. Tuo tikslu Tiuringas suformuluoją taip vadinamą *imitacinių žaidimų* (angl. *imitation game*). Šiame žaidime dalyvauja vyros (A), moteris (B) ir klausinėtojas (C). Pastarojo lytis néra svarbi. Klausinėtojas neturi tiesioginio kontakto su pašnekovais (tuomet imitacinis žaidimas netektų prasmės). Jis yra atskirame kambaryje (žr. 22.1 pav.). Klausinėtojo tikslas yra nustatyti, kuris iš dviejų žaidėjų yra vyros ir kuris moteris. Tuo tarpu kitų žaidėjų tikslas – įtikinti klausinėtoją, jog jis arba ji yra moteris. A ir B neprivalo sakyti tiesos, taigi jie gali meluoti.



22.1 pav. Imitacinis žaidimas. Žaidime dalyvauja vyros (A), moteris (B) ir klausinėtojas (C). Klausinėtojo tikslas yra nustatyti, kuris iš dviejų žaidėjų A ir B yra vyros ir kuris moteris. Ir A ir B siekia įtikinti klausinėtoją kad jie yra moteris; jie neprivalo sakyti tiesos

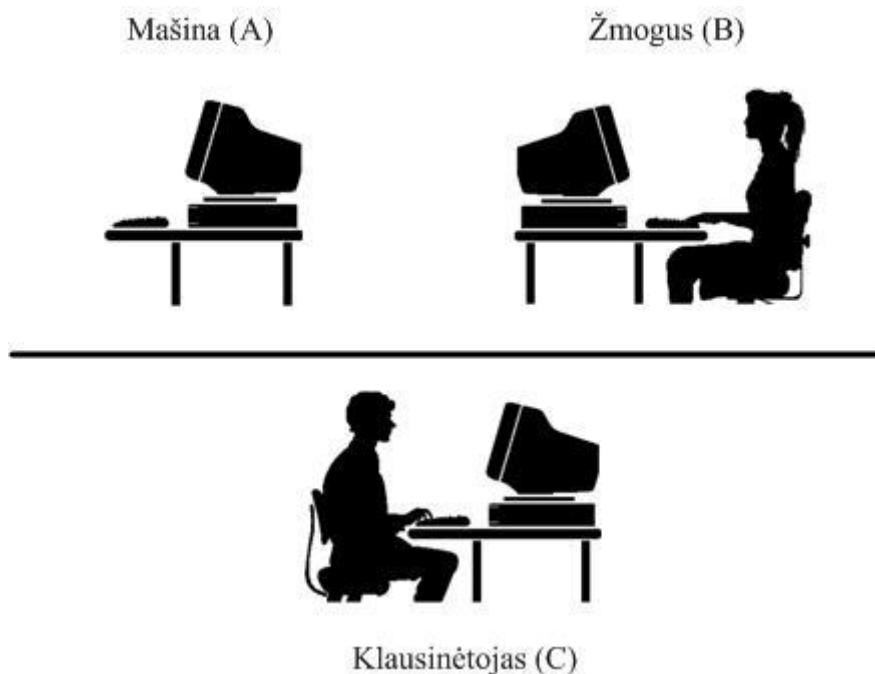
Klausimai užduodami ir atsakymai gaunami raštu arba pasitelkiant komunikacines technologijas, pvz., „teletaipo“ pagalba. Taigi, klausiantysis klausimus užduoda ir atsakymus gauna natūralia rašytine kalba. Klausimai gali būti užduodami bet kokie, pavyzdžiui: „kokis jūsų plaukų ilgis“.

Anot Tiuringo, klausimą „Ar gali mašinos mąstyti?“ galima perfrazuoti į jam ekvivalentų: „Kas atsitiks, jeigu vietoje A žais mašina? Ar šiuo atveju klausinėtojas klys taip pat dažnai, kaip ir tada, kai žaidžia tik žmonės?“ [Tiuringas 1961]. Vėliau Tiuringas šį klausimą dar kartą perfrazuoja:

„Jeigu paimsime tik vieną konkrečią skaitmeninę skaičiavimo mašiną S, tai kyla klausimas: ar teisingas teiginys, kad, keisdami tos mašinos atminties talpumą,

didindami veikimo greitį ir aprūpindami ją tinkama programa, galime priversti S patenkinamai žaisti A vietoje imitacinių žaidimų (o žaidėjas B bus žmogus)?“ [Tiuringas 1961].

Mes pastebėsime, jog naujai suformuluotoje problemoje nebedalyvauja moteris, tačiau žaidėjų tikslai išlieka nepakitę (žr. 22.2 pav.). Taigi Tiuringo testo tikslas yra ne išsiaiškinti, ar mašina gali imituoti moterų imitaciame žaidime, bet išsiaiškinti, ar mašina apskritai gali imituoti žmogų. Daugumoje vėlesnių Tiuringo testo formuluočių lytis yra ignoruojama, laikant, kad žaidime dalyvauja mašina (A), žmogus (B) ir klausinėtojas (C). Šiuo atveju, klausinėtojo tikslas yra nustatyti, kuris iš A ir B yra žmogus, o kuris mašina.



21.2 pav. Tiuringo testas yra suprantamas kaip imitacinis žaidimas. Šiuo atveju, klausinėtojo tikslas yra nustatyti, kuris iš A ir B yra žmogus, o kuris mašina

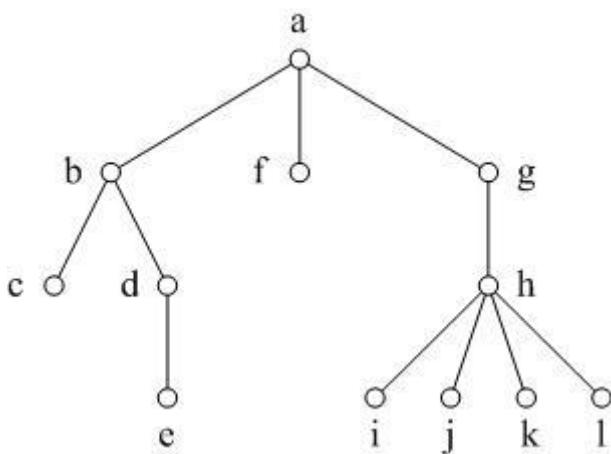
Esminis akcentas čia yra *imitacija*. Viena vertus, tai susiję su tuo, kad imitacinių žaidimo prigimtis yra apgaulė. Vyrui žaidime leidžiama sakyti viską, siekiant priversti klausinėtojų padaryti klaudingas išvadas. Tuo tarpu moters tikslas yra padėti klausinėtojui išsiaiškinti tiesą. Žaidime, kuriame dalyvauja mašina (vietoje A) ir žmogus, tikslai išlieka tie patys. Mašina stengiasi įtikinti klausinėtoją, jog ji yra žmogus. Mašinos gebėjimų neapsprendžia žaidime dalyvaujantis žmogus.

Laikui bėgant dirbtinio intelekto filosofai Tiuringo testo sampratą bandė perfrazuoti. Pavyzdžiui Džonas Searle (žr. http://en.wikipedia.org/wiki/Chinese_room) pasiūlė minties eksperimentą, kuris vadimas *kinų kambario argumentas* (angl. *Chinese Room argument*).

Žmogaus pašnekesi imituojančią programą Eliza dar 1976 m. pademonstravo J. Weizenbaum; žr. <http://en.wikipedia.org/wiki/ELIZA>. Vėlesnius pavyzdžius programas-robotus (angl. *chat bot*) galima išbandyti internte, žr. pvz. <http://nlp-addiction.com/eliza>.

23. Egzamino bilietai

1. Dirbtinio intelekto sistema kaip produkcių sistema, formalizmas, algoritmas PRODUCTION, pavyzdžiai. Dirbtinio intelekto samprata (remiantis skaityta literatūra).
2. Valdymo su grįžimais procedūros BACKTRACK ir BACKTRACK1. Valdymo su grįžimais esmė, užsiciklinimo galimybė. Pavyzdžiai. Euristikos samprata.
3. Uždavinys apie labirintą. Paieškos valdymo su grįžimais ir bangos būdais algoritmai ir programos.
4. Prefiksinė ir postfiksinė medžio apėjimo tvarka. Binariniai medžiai, bendro pavidalo medžiai. Parašykite procedūras darbui su bendro pavidalo medžiais: a) medžio įvedimui, b) apėjimui prefiksine tvarka ir c) apėjimui postfiksine tvarka. Pavyzdžiui, simbolių eilutė „a(b(cd(e))fg(h(ijkl)))“ vaizduoja medži:



5. Paieška į gylį ir į plotį medyje. Algoritmas paieškai į plotį grafe, kai grafo briaunos neturi kainos (trumpiausias kelias tarp dviejų viršunių).
6. Algoritmas paieškai grafe, kai grafo briaunos turi kainą.
7. Paieškos į gylį algoritmas grafe be kainų. Pavyzdys. Sprendėjas ir planuotojas.
8. Bendras paieškos grafe algoritmas GRAPHSEARCH. Neinformuotos procedūros, euristinė paieška. BACKTRACK1 ir GRAPHSEARCH skirtumai; kontrpavyzdys.
9. Algoritmo A* samprata. Pavyzdys, kai euristinė funkcija yra Manheteno atstumas.
10. Tiesioginis išvedimas ir atbulinis išvedimas produkcių sistemoje. Semantinis grafas. Programų sintezės elementai. Išvedimo sudėtingumas.
11. Tiesioginė ir atbulinė dedukcija pagal rezoliucijos taisyklę.
12. Ekspertinė sistema kaip dirbtinio intelekto sistema. Jos architektūra, demonstracinių sistemų pavyzdys.
13. Internetinės parduotuvės specifikacija pagal Russell & Norvig vadovėli.
14. Tiuringo testas ir dirbtinio intelekto filosofija (remiantis skaityta literatūra).
15. Neįmanomumas pasiekti keletą tikslų. Nusikaltėlio nubaudimo pavyzdys.

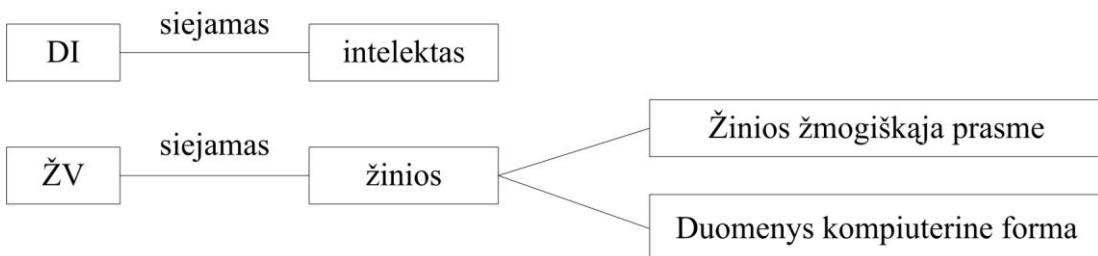
Antra dalis

24. Žinių vaizdavimas kaip dirbtinio intelekto šaka

Šiame vadovėlyje vadovaujamas požiūriu, kad *žinių vaizdavimas* (angl. *knowledge representation*) toliau ŽV, yra dirbtinio intelekto šaka. Egzistuoja ir kitokie požiūriai.

Žinių vaizdavimo ir samprotavimo žinyne [van Harmelen et. al 2008] pateikiamas toks apibūdinimas. „Žinių vaizdavimas ir samprotavimas yra dirbtinio intelekto iššūkių centre: suprasti intelekto ir pažinimo prigimtį tokiu mastu, kad kompiuteriai demonstruotų žmogui būdingus gebėjimus“. Pastebime, kad čia akcentuojamas pažinimas, o ne dirbtinio žmogaus sukūrimas.

Savokos „žinių vaizdavimas“ turinys atskleidžiamas remiantis žodžiu „žinios“ ir „vaizdavimas“ gramatininiu aiškinimu (žr. 24.1 pav.). Atskirame moksliniame tyrime būtų galima propaguoti kokią nors specifinę ŽV sampratą, pavyzdžiui, ŽV kaip pažinimas. Tačiau tai išeitų už vadovėlio ribų. Toliau vadovaujamas tokia ŽV samprata, kuri priimta paplitusiuose vadoveliuose, pvz. [Brachman, Levesque 2004; Russell, Norvig 2003; Stefik 1995].



24.1 pav. DI yra siejamas su žodžiu „intelektas“, o ŽV – su „žinios“

Savoka žinios (angl. *knowledge*) tam tikru mastu yra „mistine“. Jos turinys gali būti aiškinamas įvairiai, atsižvelgiant į žodžio žinios aiškinimą filosofijoje, jos šakoje kognityvistikoje (pažinimo teorijoje), sociologijoje, istorijoje ir informatikoje.

Mūsų nuostata, kad žinių vaizdavimas – tai žinių vertimas duomenimis. Savoka duomenys (angl. *data*) informatikoje nėra mistinė, o turi visuotinai priimtą turinį.

DI sampratą apsprendžia riba tarp žmogiškojo intelekto (angl. *human intelligence*) ir mašininio intelekto (angl. *machine intelligence*). DI sampratai yra būdinga, kad ji kinta. Šios ribos plėtimasis salygoja ir savokos *intelektuali sistema* (angl. *intelligent system*) sampratą. DI riba plečiasi, užgriebdama vis naujas sritis. Kas vakar buvo priskiriama žmoniškajam intelektui, šiandien gali būti priskiriama DI. Šia savybe paaiškinamas posakis „mes užsiiminėjome DI patys nežinodami, kad tai yra DI“.

Informatikos ir DI vystimasis salygoja kad kiekvienam dešimtmiečiui yra būdinga tam tikra DI samprata. Yra išskiriamos tam tikros *programavimo eros*. Analogiskai yra su DI ir ŽV. Todėl pateisinama, kad skirtinių autorai akcentuoja skirtingus požiūrius į DI ir ŽV.

Vadovaujantis [Nilsson 1982] *dirbtinio intelekto sistema* yra suprantama kaip *produkcijų sistema*, kuri, savo ruožtu, yra trejetas: *globali duomenų bazė*, *produkcijų aibė* ir *valdymo sistema*. Remiantis tokiu modeliu dalykinės srities žinios klasifikuojamos pagal tai, kur yra vaizduojamos. *Deklaratyvioms žiniomis* yra vadinamos žinios, vaizduojamos globalioje duomenų bazėje. *Procedūrinėmis žiniomis* vadinamos žinios, vaizduojamos produkcijų aibėje. *Valdymo žiniomis* vadinamos žinios, vaizduojamos valdymo strategijoje.

Žinių vaizdavimo vietą informatikoje atspindi neformali klasifikacija, parodyta 24.2 pav. Detaliau buvo apžvelgta 1.2 poskyryje.



24.2 pav. Žinių vaizdavimo vieta informatikoje

Klasifikavimą kaip abstrahavimąsi ir sąvoką sukūrimą autorius priskiria žmogiškojo intelekto sričiai. Plečiantis DI ribai yra užgriebiamos atskiro žmogiškojo intelekto sritys. Klasifikavimas gali būti siejamas su *duomenų gavyba* (angl. *data mining*). Pastaroji yra siejama ir su DI, o ne tik su duomenų bazė valdymo sistemomis ar statistika.

Žinių vaizdavimo būdai (t.y. kryptys, paradigmos) skirstomos į dvi dideles šakas:

1. **procedūrinis** (angl. *procedural*) žinių vaizdavimas. Algoritmai ir programos yra šio vaizdavimo pavyzdžiai (pavaizdavimas plačiąja prasme);
2. **deklaratyvus** (angl. *declarative*) žinių vaizdavimas. Labai apibendrintai ir negriežtai apibūdinsime kaip vaizdavimą duomenimis.

Deklaratyvus žinių vaizdavimas toliau yra skirstomas į keturias šakas:

1. **loginis**
 - 1) Teiginių logika
 - 2) Predikatų logika
 - 3) Deontinė logika. Pavyzdys: Prieš teiginį *A* yra vienas iš dviejų deontinių operatorių *O* arba *P*. *O(A)* reiškia „privalo būti“ *A*, angliskai „it is obligatory that *A*“. *P(A)* žymi „gali būti *A*“, angliskai „it is permitted that *A*“.
 - 4) Nurungimo logika (angl. *defeasibility logic*)
2. **procedūrinis** (siauraja prasme)
 - 1) Produkcių sistemos (pvz. $A, B, C \rightarrow D, E$)
3. **tinklinis** (angl. *network representation schema*)
 - 1) semantiniai tinklai (angl. *semantic networks*). Semantinis tinklas – tai grafas, kurio viršūnėms ir briaunoms suteikiama tam tikra prasmė.
 - 2) konceptiniai grafai (angl. *conceptual graphs*). Konceptinis grafas – tai grafas, kuris atspindi dalykinės srities **sąvokas**.
4. **struktūrinis** (angl. *structured knowledge representation*)
 - 1) freimai (angl. *frames*)
 - 2) objektai (angl. *objects*)

Supaprastintai skirtumą tarp procedūrinio ir deklaratyvaus žinių vaizdavimo galima charakterizuoti ir prisimenant skirtumą tarp programos ir jos specifikacijos. Procedūrinis vaizdavimas (kaip ir programa) atsako į klausimą „kaip“ (tiksliau, kaip sprendžiamas tam tikras uždavinys). Deklaratyvus vaizdavimas (kaip ir programos specifikacija) atsako į klausimą „kas“ (kas vaizduojama, koks uždavinys sprendžiamas), bet neatsako, kaip jį spręsti. Žinių vaizdavime pagrindinis dėmesys skiriamas deklaratyviems būdams.

25. Duomenys, informacija ir žinios

Vadovėlio autorius gina tokį požiūrį į žinių vaizdavimą kaip dalykinę veiklą:

Žinių vaizdavimas – tai žinių vertimas duomenimis.

Mums aktuali žinių sąvoka aptariama žiniomis grindžiamų sistemų vadovėliuose, pvz. [Stefik, 1995]. Pasitelkiamas gramatinis žodžio žinios aiškinimas. Žinių vaizdavimo kontekste, esminė sąvoka yra išvedimas (angl. *entailment*).

Kitokie duomenų informacijos ir žinių aspektai yra nagrinėjami komunikacijos moksle (angl. *communication science*).

Charakterizuodami sąvokas „duomenys“, „informacija“, „žinios“ ir „išmintis“ sistemų teorijos specialistai [Bellinger, Castro, Mills 2004] pasitelkia pragmatines veiklas. Jie pradeda cituodami sistemų teorijos specialistą ir organizacijų kaitos profesorių Russell Ackoff [Ackoff 1985], kuris išskiria penkias kategorijas:

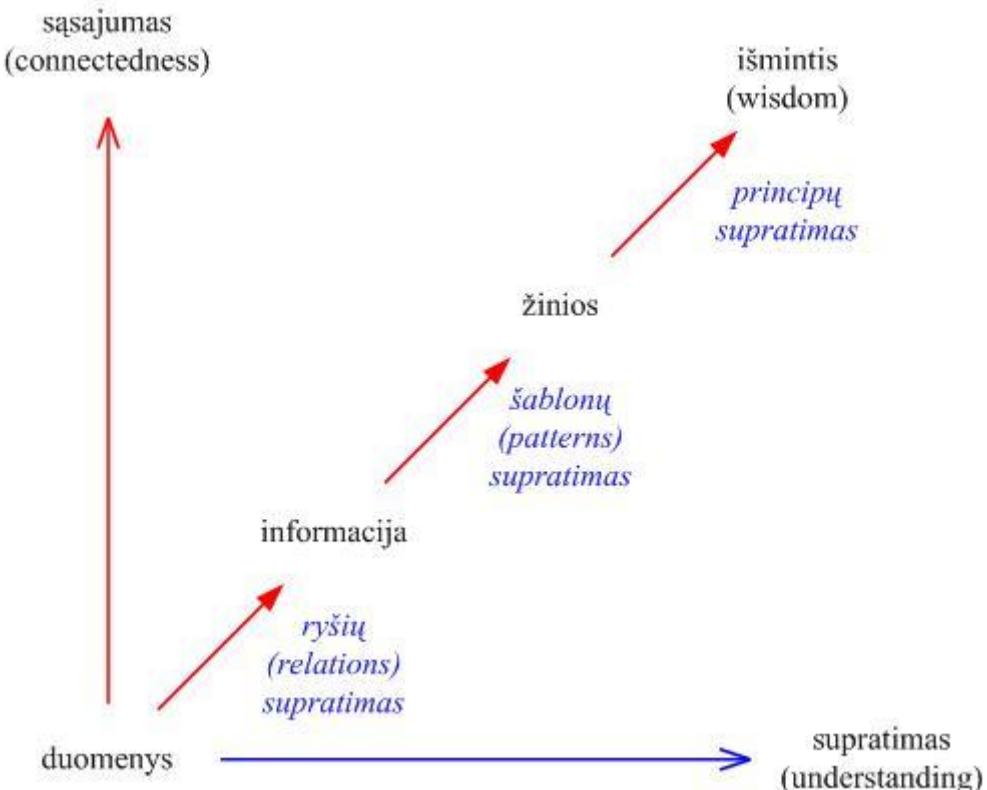
1. *duomenys*. Tai simboliai;
2. *informacija*. Tai duomenys, kurie yra apdorojami ir yra naudingi.; atsako į klausimus „kas“, „kā“, „kur“ ir „kada“;
3. *žinios*. Tai duomenų ir informacijos pritaikymas. Atsako į klausimą „kaip“;
4. *supratimas*. Tai „kodėl“ pripažinimas;
5. *išmintis*. Tai įvertintas supratimas.

Ackoff nuomone, pirmosios keturios kategorijos siejamos su praeitimi – kas yra arba buvo žinoma. Tik penktoji kategorija – išmintis – yra siejama su ateitim, nes apima įžvalgumą ir sumanymą.

[Bellinger, Castro, Mills 2004] išplečia aukščiau pateiktus Ackoff apibūdinimus ir pateikia savo modelį (žr. 25.1 pav.):

1. *duomenys* yra neapdoroti. Pavyzdžiui, kompiuterinės skaičiuoklės lentelė (angl. *spreadsheet*);
2. *informacija*. Tai duomenys, kuriems yra suteikta reikšmė. Kitaip sakant, tai interpretuoti duomenys;
3. *žinios*. Tai informacijos visuma, skirta tam tikriems tikslams. Kai asmuo iškala informaciją pats jos nesuprasdamas, tai tokia veikla nėra laikoma žinių įsisavinimu. Dauguma taikomųjų programų naudoja kompiuteryje saugomas žinias.
4. *supratimas* pasižymi interpoliavimu ir tikimybe. Supratimas yra kognityvinė ir analitinė veikla. Šioje veikloje iš turimų žinių yra gaunamos naujos žinios. Skirtumas tarp supratimo ir žinių gali būti lyginamas su skirtumu tarp išmokinimo ir įsiminimo. DI sistemos pasižymi supratimu, bet „dirbtinio“, o ne žmogiškojo intelekto prasme. DI sistemos gali sintezuoti naujas žinias iš anksčiau sukauptos informacijos ir žinių;
5. *išmintis* pasižymi ekstrapoliacijimu, o taip pat determinuotumo bei tikimybės nebuvimu. Išmintis iššaukia kitus sąmonės lygmenis, atskiru atveju, žmonėms skirtas norminges sistemas, pavyzdžiui, moralės kodeksą. Skirtingai nuo keturių ankstesnių sąvokų išmintis kelia klausimus, į kuriuos nėra lengvų atsakymų arba išviso nėra įmanoma atsakyti. Išmintimi grindžiamas filosofinis tyrimas. Išmintimi yra sprendžiama, kas yra gerai ir blogai, teisinga ir neteisinga. Gene Bellinger laikosi nuomonės, kad kompiuteriai niekada neturės išminties. Išmintis būdinga tik žmogui. Išminčiai būtina siela, kuri glūdi ne tik prote, bet ir širdyje. Siela yra tai, ko kompiuteriai niekada neturės.

[Bellinger, Castro, Mills 2004] keturias savokas – duomenis, informacija, žinias ir išmintį – vaizduoja semantinio grafo viršūnėmis, o penktą – supratimą – kaip briaunas (t.y. perėjimus) tarp šių viršūnių. Kadangi šių perėjimų yra trys, tai jiems priskiriami papildomi atributai, vaizduojantys supratimo lygmenį: ryšių, šablonų ir principų supratimą.



25.1 pav. Trys supratimo lygmenys – ryšių supratimas, šablonų supratimas ir principų supratimas – kaip perėjimai tarp duomenų, informacijos, žinių ir išminties [Bellinger, Castro, Mills 2004]
<http://www.systems-thinking.org/dikw/dikw.htm>.

Tokiu būdu turime trijų lygmenų supratimą:

1. ryšio supratimas kaip perėjimas nuo duomenų prie informacijos;
2. šablono supratimas kaip perėjimas nuo informacijos prie žinių;
3. principų supratimas kaip perėjimas nuo žinių prie išminties.

Duomenys vaizduoja faktą arba teiginį be ryšio su kitais dalykais. Pavyzdžiui, „lyja“. Duomenys nėra interpretuoti. Pavyzdžiui, Excel lastelėje parašytas skaičius 7. Šiam skaičiui nesuteikiama jokia prasmė. Jeigu pasakytume, kad tai 7 kilometrai, 7 mylios, 7 000 Lt ar 7 Lt – tai jau būtų informacija.

Informacija – tai interpretuoti duomenys. Reikšminga gali būti tik informacija, o ne duomenys, nes tik suteikus prasmę, galima lyginti reikšmingumą kokiui nors atžvilgiu. Informacija įkūnija tam tikro ryšio supratimą, pavyzdžiui, tarp priežasties ir pasekmės, pvz., „temperatūra nukrito iki 15 laipsnių ir lauke pradėjo lyti“.

Žinios – tai informacijos ir naujų faktų išvedimas. Žinios vaizduoja numatymo šabloną. Šis šablonas sujungia, kas yra turima, su tuo, kas atsitiks. Pavyzdžiui:

*„Jeigu oro drėgnumas didelis ir oro temperatūra stipriai nukrinta,
tai atmosferoje negali išsilaikyti drėgmė,
ir todėl lauke lyja.“*

Išmintis įkūnija žiniose glūdinčių fundamentalių principų supratimą. Pavyzdžiu:

„Lauke lyja, nes yra toks reiškinys – lietus. Lietus kaip gamtos reiškinys apima supratimą apie garavimą, oro sroves, temperatūros pokyčius, garų kondensaciją, lietų ir kt.“

Išmintis pasižymi ekstrapoliavimu.

Autoriaus nuomone aukščiau pateikti modeliai labiau aktualūs bibliotekininkystės, informacijos ir komunikacijos moksluose (angl. *library, information and communication sciences*). Pastaruosiuose esminė sąvoka yra informacija. O informatikoje svarbesnė yra duomenų sąvoka. Žinių vaizdavime svarbesnė yra naujų žinių išvedimo sąvoka.

Interpretacijos reikšmė perėjime nuo duomenų prie informacijos iliustruojama tokiu pavyzdžiu. „Aš važiavau mieste 50, o mane sustabdė kelių policininkas ir be pagrindo nubaudė už greičio viršijimą“. Tegu toks tekstas saugomas dokumentų valdymo sistemoje, kaip asmens parodymai. Interpretacija: 50 mylių per valandą. Tokia interpretacija yra labai tikėtina atsižvelgiant į neišreikštines žinias, kad vairuotojas yra užsienietis. 50 mylių per valandą yra 80 kilometrų per valandą. Kompiuteris tokios interpretacijos nežino, o ginčą nagrinėjantis pareigūnas numano. Todėl kompiuteris negali nuspręsti, ar vairuotojas pažeidė kelių eismo taisykles.

26. Žmogiškasis pažinimas ir žiniomis grindžiamos sistemos

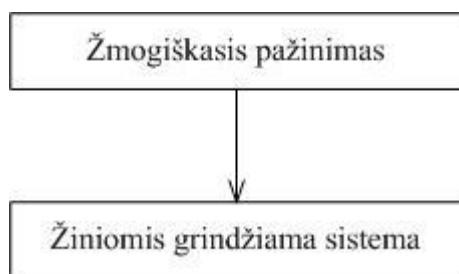
Autorius žinių vaizdavimą sieja su dviem sąvokomis. Tai:

1. *žmogiškasis pažinimas*. Visų pirmo, tai žmogui būdingo smalsumo patenkinimas. Antra, pažinimas kaip mokslinė, projektinė veikla, tarp kurios rezultatų yra ataskaita (angl. *report*) apie tam tikroje dalykinėje srityje atskleistus faktus ir dėsnius;
2. *žiniomis grindžiamos sistemos*.

Šios dvi sąvokos skirtos dviejų tipų siekiams. I šias sąvokas galime žiūrėti kaip i žmogaus dalykinės veiklos motyvus arba tikslus:

1. *pažinti* tam tikrą reiškinį (kuris sudaro dalykinę sritį);
2. *sukurti* žiniomis grindžiamą sistemą.

Atrodytų, kad tai alternatyvūs, vienas nuo kito nepriklausomi motyvai. Tačiau įsigilinus galima pastebėti, kad antrajam tikslui pasiekti yra būtinės pirmasis. Norint sukurti žiniomis grindžiamą sistemą tam tikroje dalykinėje srityje yra būtina šią dalykinę sritį pažinti. Modelis yra pateikiamas 26.1 pav.



26.1 pav. Dvi paveikslė interpretacijos. Pirma, žmogiškasis pažinimas yra autorui svarbesnis motyvas negu žiniomis grindžiamų sistemos. Antra, žmogiškasis pažinimas yra būtinės kaip žinių išgavimo stadija kuriant žiniomis grindžiamą sistemą.

Žiniomis grindžiama sistema (ŽGS) (angl. *knowledge-based system*, KBS) – tai kompiuterinė sistema, kuri naudoja žinias. Terminą *žiniomis grindžiama sistema* [Stefik 1995] trumpina iki **žinių sistema** (angl. *knowledge system*). Toks sutrumpinimas yra visiškai priimtinės vienos knygos rėmuose. Tačiau bendru atveju, pavartotas neišskiriant tam tikro konteksto, gali būti suprastas kaip žinių iš tam tikros dalykinės srities, o ne jos apdorojimo būdų, sistema. Čia mes darome analogiją su terminais *informacinė sistema* ir *informacijos sistema*.

Gali būti diskutuojama, koks šių sąvokų turinys. Informacinė sistema yra informatikoje prigijusi sąvoka. Ji siejama su tuo, *kaip* duomenys apdorojami.

Terminą „informacijos sistema“, naudojamą bibliotekininkystėje, galima aiškinti iš informatikos pozicijos. „Informacijos sistema“ žymi, *kas* yra apdorojama. Čia „informacija“ yra apdorojimo objektas. Informacijos sistema – tai sistema apie tam tikros dalykinės srities informaciją. Tai sistematizuota informacija. Terminas „informacijos sistema“ kildinamas iš bibliotekininkystės, informacijos ir komunikacijos mokslų. Pavyzdžiu, Universalioji dešimtainė klasifikacija, UDK (rus. Универсальная десятичная классификация, УДК), gali būti priskiriama prie informacijos sistemų. UDK – tai sisteminė, visas žinių sritis aprėpianti bibliotekinė-bibliografinė klasifikacija.

Žiniomis grindžiamos sistemos sudaro atskirą informacinių sistemų šaką, formaliai:

Žiniomis grindžiama sistema *is_a* informacinė sistema.

Žiniomis grindžiamos sistemos gyvavimo ciklo modelis būtų atskiras informacinės sistemos gyvavimo ciklo modelio atvejis. Informacinės sistemos gyvavimo ciklo pirmoji stadija yra reikalavimų analizė. Žiniomis grindžiamų sistemų pirmoji gyvavimo stadija yra **žinių išgavimas** (angl. *knowledge acquisition*), pvz., [Stefik 1995, p. 217]. O žinių išgavimo veikloje yra reikalaujama pažinti dalykinę sritį. Tiesa, **žinių inžinierius** neprivalo dalykinės srities pažinti taip giliai kaip dalykinės srities **ekspertas**.

Kaip matyti iš aukščiau, žiniomis grindžiamos sistemos sukūrimui yra būtina pažinti dalykinę sritį. Koks šio pažinimo mastas, tai jau kitas klausimas.

[Buchanan et al. 1990] rašyti apie žiniomis grindžiamas sistemas pradeda pastebėjimu, kad žinios turi būti prieinamos tiems, kam jos yra reikalingos. Žiniomis grindžiamų sistemų paskirtis yra apibūdinama šitaip: kaip laiku ir tiksliai pateikti žinias, naudingas tam tikrai užduočiai (angl. *task*). Kitaip sakant, kaip profesinę patirtį tam tikroje srityje pateikti sprendimų priemėjams kada jiems reikia ir kur reikia. Yra įprasta naudoti terminą „ekspertinė sistema“ dėl tokių priežascių:

1. ŽGS demonstruoja kompetenciją duotoje užduotyje tokiu lygiu, kaip ir duotos dalykinės srities ekspertai;
2. Žinios, esančios ŽGS, yra išgautos iš ekspertų;
3. ŽGS išvedimo metodai yra pagrįsti uždavinių sprendimo technika ir strategijomis, kurias naudoja ekspertai.

Kadangi šios trys charakteristikos yra tik siūlomojo, o ne privalomojo pobūdžio, t.y. nėra nei būtinos, nei pakankamos intelektualioms sistemoms, tai [Buchanan et al. 1990] suteikia pirmenybę terminui žiniomis grindžiama sistema. Terminas ŽGS žymi dirbtinio intelekto taikymą uždavinių sprendimo (angl. *problem-solving*) ir sprendimų priemimo (angl. *decision-making*) užduotims spręsti.

ŽGS architektūroje yra išskiriamos dvi sudėtinės dalys:

1. samprotavimo (angl. *reasoning*) (arba užduoties sprendimo) procesas;
2. žinių bazė (angl. *knowledge base*)

ŽGS skirta spręsti tik *tam tikro* tipo užduotims. Ankstyvajame dirbtinio intelekto tyrimų laikotarpyje mokslininkai tikėjos, kad DI tyrimų rezultatus vainikuos bendras užduočių sprendėjas (angl. *general problem solver*) [Ernst, Newell 1969]. Deja tokio universalaus sprendėjo iš principo neįmanoma sukurti. Matematinėje logikoje yra nagrinėjamos algoritmiškai neišprendžiamomis problemos.

Užduočių tipus, kurių sprendimui taikomas DI, klasifikavo [Chandrasekaran et al. 1992]. [Valente 1995, p. 136] išskiria tokias tipiniai (*generic*) DI užduotis: numatymas, valdymas, diagnozė, nurodymas.

27. Žinių vaizdavimo klasifikavimas į procedūrinį ir deklaratyvų

27.1. Procedūrinis žinių vaizdavimas

Programavimas algoritmine kalba gali būti siejamas su procedūriniu žinių vaizdavimu.

Šis teiginys iliustruojamas pavyzdžiu, demonstruojančiu kvadratinės lygties sprendime iškūnytasis žinias. Kvadratinės lygties $ax^2 + bx + c = 0$ šaknys yra

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Šaknis x_1 ir x_2 apskaičiuoja šios procedūros:

```
{ Procedūra apskaičiuoja šaknį x1 }
procedure šaknis1 (a, b, c: real; var x1: real);
    x1 := ( -b + sqrt ( b*b - 4*a*c ) ) / ( 2*a )
end

{ Procedūra apskaičiuoja šaknį x2 }
procedure šaknis2 (a, b, c: real; var x2: real);
    x2 := ( -b - sqrt ( b*b - 4*a*c ) ) / ( 2*a )
end
```

Kvadratinę lygtį su koeficientais AA, BB, CC, galima išspręsti programa, kuri nuosekliai iškviečia x_1 ir x_2 apskaičiavimo procedūras:

```
read(AA, BB, CC);
call saknis1(AA, BB, CC, XX1);
call saknis2(AA, BB, CC, XX2);
writeln(XX1, XX2)
```

Šios lygties šaknų apskaičiavimui toliau pateikiama efektyvesnė procedūra saknys. Pastaroji šaknies traukimo veiksmą atlieka tik vieną kartą, o ne du kartus, kaip kviečiant abi procedūras saknis1 ir saknis2 nuosekliai.

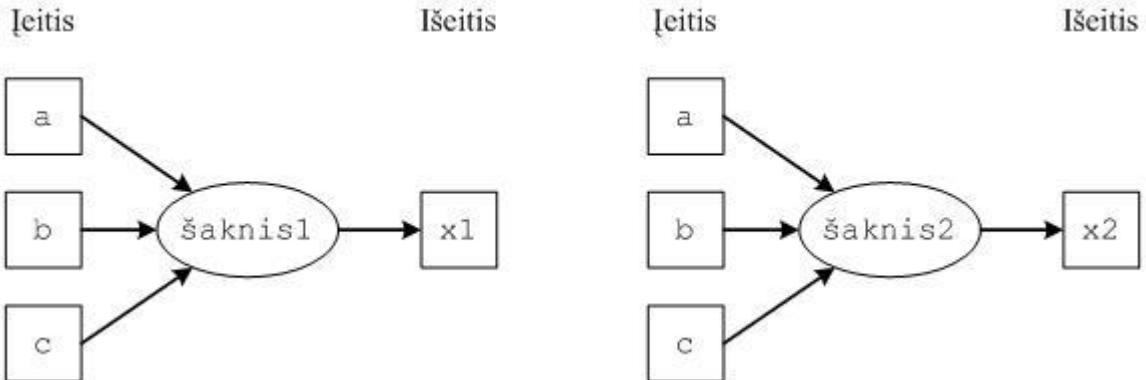
```
{ Procedūra apskaičiuoja šaknis x1 ir x2 }
procedure šaknys ( a, b, c: real; var x1, x2: real );
    var kint: real;
begin
    kint := sqrt(b*b - 4*a*c);
    x1 := ( -b + kint ) / ( 2*a );
    x2 := ( -b - kint ) / ( 2*a );
end;
```

Pirmoji ir antroji programa iškūnija skirtinges žinias apie sprendimą. Pirmuoju atveju sprendimo koncepcija yra paprastesnė, o antruoju – vykdymo laikas yra trumpesnis.

Algoritmas yra viena iš žinių vaizdavimo formų.

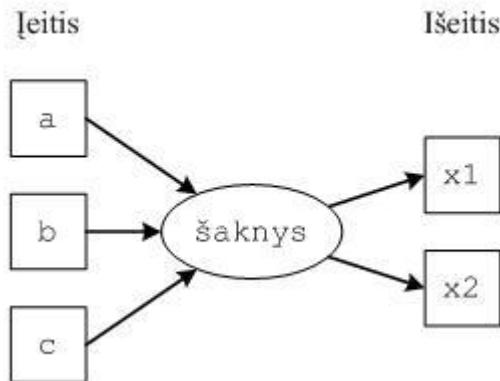
27.2. Deklaratyvus žinių vaizdavimas

Kvadratinės lygties šaknų apskaičiavimo procedūrų šaknis1 ir šaknis2 deklaratyvus pavaizdavimas, suprantamas kaip procedūros įėitis ir išeitis, yra pateiktas 27.1 pav.



27.1 pav. Lygties $ax^2 + bx + c = 0$ šaknų apskaičiavimo procedūrų šaknis1 ir šaknis2 deklaratyvus pavaizdavimas. Vaizduojama procedūrų įėitis ir išeitis. Tekstine forma $\{a, b, c\} \xrightarrow{\text{šaknis1}} \{x_1\}$ ir $\{a, b, c\} \xrightarrow{\text{šaknis2}} \{x_2\}$

Analogiškai procedūros šaknys deklaratyvus pavaizdavimas yra pateiktas 27.2 pav.



27.2 pav. Lygties $ax^2 + bx + c = 0$ šaknų apskaičiavimo procedūros šaknys deklaratyvus pavaizdavimas. Tekstine forma $\{a, b, c\} \xrightarrow{\text{šaknys}} \{x_1, x_2\}$

Deklaratyvus vaizdavimas atsako į klausimą „kas“, bet neatsako į klausimą „kaip“. Atsakymas į šį klausimą įkūnytas algoritmo realizacijoje.

Tegu *in* žymi procedūros įėitį (angl. *input*), *out* – išeitį (angl. *output*). Tokiu būdu $in(\text{šaknis1}) = \{a, b, c\}$ ir $out(\text{šaknis1}) = \{x_1, x_2\}$.

Procedūros P deklaratyvus pavaizdavimas yra suprantamas kaip pora susidedanti iš šios procedūros įėities ir išeities kintamųjų (objektų iš tam tikro alfabeto), formaliai $\langle in(P), out(P) \rangle$, sutrumpintai $\langle in P, out P \rangle$. Ši pora dar vadinama procedūros P semantika

jeities ir išeities terminais, ir žymima $sem(\mathbb{P})$. Tokia pora dar vadinama *funkcinę priklausomybę* (angl. *functional dependency, data dependency*) arba *skaičiavimo santykium* (rus. *вычислительное отношение*), žr. pvz. [Tyugu 1984]. Tokiu būdu:

$$sem(\text{šaknis1}) = in(\text{šaknis1}) \rightarrow out(\text{šaknis1}) = \{a, b, c\} \rightarrow \{x_1\} \quad (26.1)$$

$$sem(\text{šaknis2}) = in(\text{šaknis2}) \rightarrow out(\text{šaknis2}) = \{a, b, c\} \rightarrow \{x_2\} \quad (26.2)$$

$$sem(\text{šaknys}) = in(\text{šaknys}) \rightarrow out(\text{šaknys}) = \{a, b, c\} \rightarrow \{x_1, x_2\} \quad (26.3)$$

Mums svarbu kad sekos šaknis1; šaknis2 išeitis yra lygi procedūros šaknis1 išeities $\{x_1\}$ ir procedūros šaknis2 išeities $\{x_2\}$ sajungai $\{x_1, x_2\}$:

$$sem(\text{šaknis1}; \text{šaknis2}) = \{a, b, c\} \rightarrow \{x_1, x_2\} \quad (27.4)$$

Kaip matyti iš (27.4) ir (27.3) sekos šaknis1; šaknis2 ir atskiros procedūros šaknys semantika sutampa:

$$sem(\text{šaknis1}; \text{šaknis2}) = sem(\text{šaknys})$$

Reikia pastebėti kad sekos šaknis1; šaknis2 ir atskiros procedūros šaknys sutampa tik semantika jeities-išeities terminais. Procedūra šaknys atlieka mažiau veiksmų negu sekos šaknis1; šaknis2. Todėl pastarosios nesutampa kitomis savybėmis.

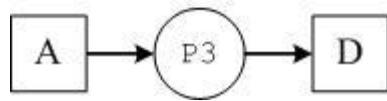
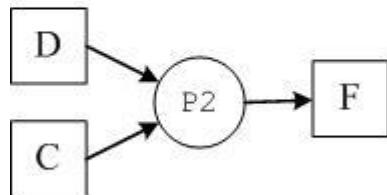
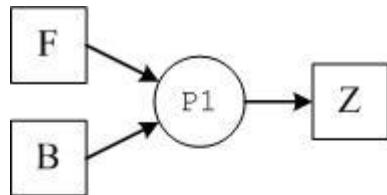
Toliau vėl prisimenama produkcių sistema (18.1), pateikta 18 skyriuje.

$$\begin{aligned} \pi_1: & F, B \rightarrow Z \\ \pi_2: & C, D \rightarrow F \\ \pi_3: & A \rightarrow D \end{aligned} \quad (18.1)$$

Šios trys produkcijos siejamos su trimis procedūromis (18.3):

procedure P1;	(18.3)
Z := f1(B, F)	
procedure P2;	
F := f2(C, D)	
procedure P3;	
D := f3(A)	

Procedūrų P1, P2, P3 semantika yra vaizduojama atitinkamai produkcionimis π_1 , π_2 ir π_3 . Grafiškai procedūrų semantika yra vaizduojama semantiniais grafais, pateiktais 27.3 pav.



27.3 pav. Procedūrų P_1 , P_2 ir P_3 semantikos grafinis pavaizdavimas semantiniai grafais. Procedūros semantika $sem(P)$ yra suprantama kaip pora $\langle in(P), out(P) \rangle$ ir žymima $in(P) \rightarrow out(P)$

Pateiktame pavyzdje alfabetas yra visų išeities ir įeities objektų sąjunga $\{A, B, C, D, F, Z\}$.

Procedūros P_1 įeitis ir išeitis:

$$\begin{aligned} in(P_1) &= \{F, B\} \\ out(P_1) &= \{Z\} \end{aligned}$$

Analogiškai procedūros P_2 įeitis ir išeitis:

$$\begin{aligned} in(P_2) &= \{D, C\} \\ out(P_2) &= \{F\} \end{aligned}$$

Bei procedūros P_3 įeitis ir išeitis:

$$\begin{aligned} in(P_3) &= \{A\} \\ out(P_3) &= \{D\} \end{aligned}$$

Iškeliamas klausimas: kokia yra procedūrų P_1 ir P_2 nuoseklios kompozicijos (paprastai kalbant, sekos) $P_1; P_2$ įeitis ir išeitis? Tiksliau, kaip išreiškiama nuoseklios kompozicijos $P_1; P_2$ semantika, jeigu žinoma P_1 ir P_2 semantika?

27.1 apibrėžimas. Procedūrų P_1 ir P_2 sekos $P_1; P_2$ įeitis ir išeitis apibrėžiama tokiomis formulėmis:

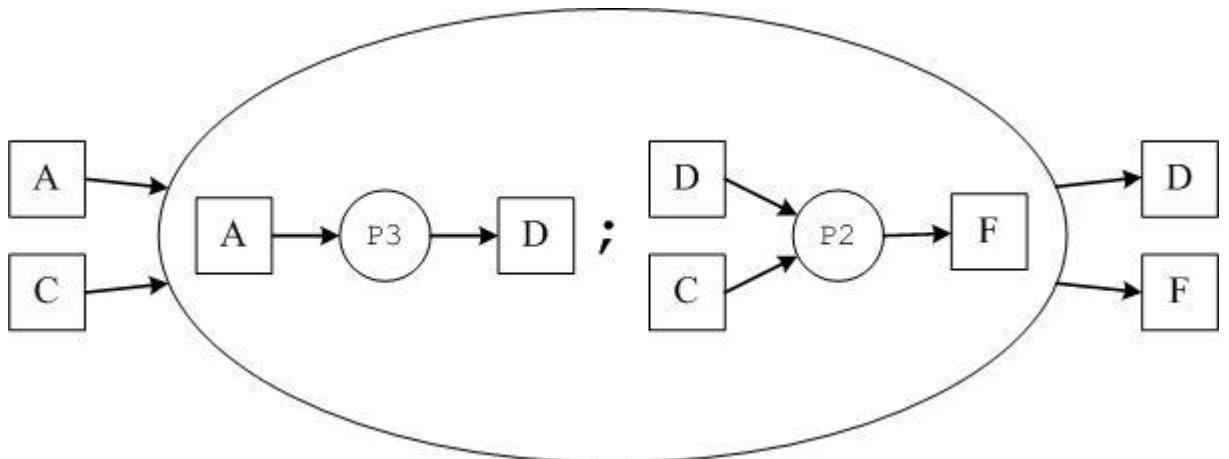
$$in(P_1; P_2) = in P_1 \cup (in P_2 / out P_1) \quad (27.5)$$

$$out(P_1; P_2) = out P_1 \cup out P_2 \quad (26.6)$$

Paprasčiau paaiškinama išeities formulė. Dviejų programų sekos išeitis yra lygi šių programų išeicių sajungai.

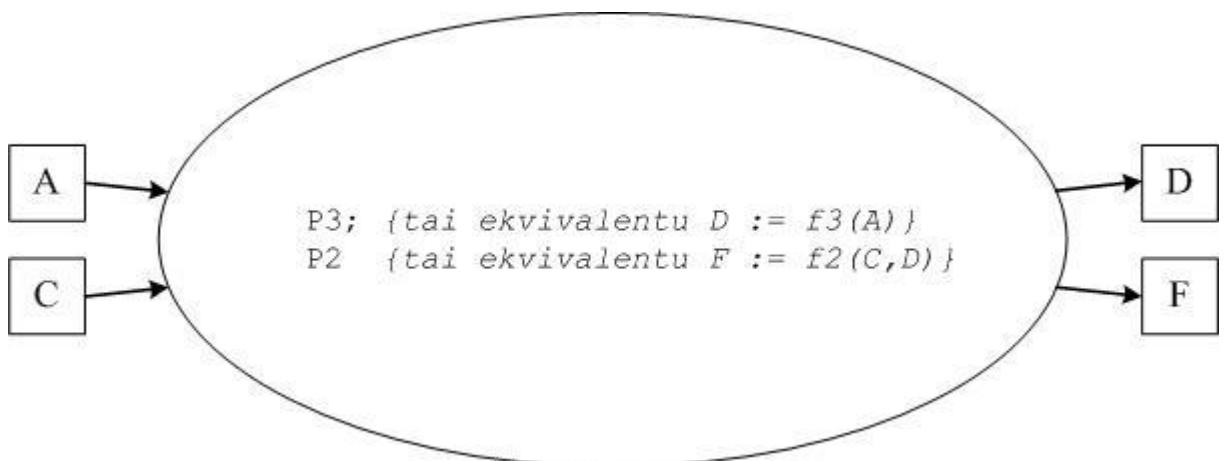
Dviejų programų įeitis yra lygi sajungai, kurią sudaro P_1 įeitis ir ta P_2 dalis, kuri nejineina į P_1 išeiti.

Šias formules iliustruoja procedūrų P_3 ir P_2 sekos įeities ir išeities pavaizdavimas 27.4 pav.



27.4 pav. Procedūrų sekos $P_3; P_2$ semantikos, suprantamos kaip įeitis ir išeitis pavaizdavimas semantiniu grafu

27.4 pav. pavaizduotą sekos $P_3; P_2$ semantika sutrumpintai pateikiama 27.5 pav.

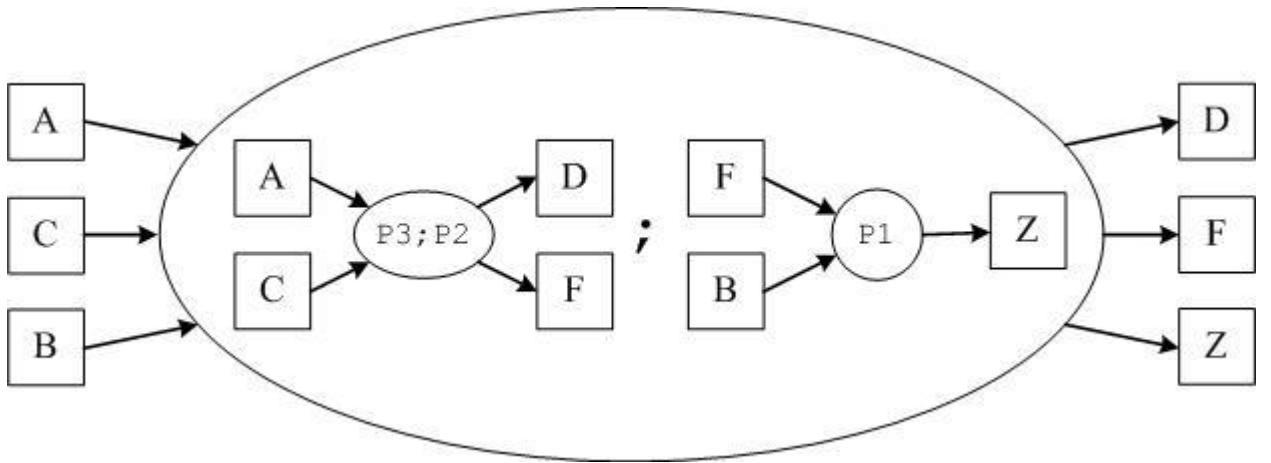


27.5 pav. Procedūrų sekos $P_3; P_2$ semantika

Trijų procedūrų P_3 , P_2 ir P_1 nuosekli kompozicija yra apibrežiama rekurentiškai, t.y. kaip nuosekli kompozicija pirmųjų dviųjų procedūrų, o paskui dar nuosekli kompozicija su trečiaja procedūra:

$$P_3;P_2;P_1 = (P_3;P_2);P_1 \quad (26.7)$$

Šią formulę iliustruoja procedūrų P_3 , P_2 ir P_1 sekos įėities ir išeities pavaizdavimas 27.6 pav.



27.6 pav. Trijų procedūrų sekos $P_3;P_2;P_1$ semantikos pavaizdavimas per $P_3;P_2$ ir P_1 seką

Trijų procedūrų sekos $P_3;P_2;P_1$ įėitis gaunama remiantis (27.7) ir (27.5) formulėmis:

$$in(P_3;P_2;P_1) = in(P_3;P_2) \cup (in(P_1) / out(P_3;P_2)) = \{A,C\} \cup \{\{F,B\} / \{D,F\}\} = \{A,C,B\}$$

Trijų procedūrų sekos $P_3;P_2;P_1$ išeitis gaunama remiantis (27.7) ir (27.6) formulėmis:

$$out(P_3;P_2;P_1) = out(P_3;P_2) \cup out(P_1) = \{\{D,F\} \cup \{Z\}\} = \{D,F,Z\}$$

Reziumuojama, kad aukščiau operacijomis tarp aibų įrodyta:

$$\begin{aligned} in(P_3;P_2;P_1) &= \{A,C,B\} \\ out(P_3;P_2;P_1) &= \{D,F,Z\} \end{aligned}$$

Šiame etape terminas „žinios“ suprantamas kaip pora \langle įėitis, išeitis \rangle . Bendru atveju trijų procedūrų sekos įėities ir išeities formulės yra gaunamos remiantis (27.7), (27.5) ir (27.6):

$$\begin{aligned} in(P_1;P_2;P_3) &= in(P_1;P_2) \cup (in(P_3)/out(P_1;P_2)) = \\ &= in(P_1) \cup (in(P_2)/out(P_1)) \cup (in(P_3)/(out(P_1) \cup out(P_2))) \end{aligned}$$

$$out(P_1;P_2;P_3) = out(P_1;P_2) \cup out(P_3) = out(P_1) \cup out(P_2) \cup out(P_3)$$

1.3 teorema. Procedūrų P_1, P_2, \dots, P_n nuoseklios kompozicijos įėitis ir išeitis yra:

$$in(P_1;P_2; \dots; P_n) = \bigcup_{i=1}^n (in(P_i) \setminus \bigcup_{j=1}^{i-1} out(P_j)) \quad (27.8)$$

$$out(P_1;P_2; \dots; P_n) = \bigcup_{i=1}^n out(P_i) \quad (27.9)$$

Irodymas matematinės indukcijos būdu. Tai padaryti paliekama kaip pratimas skaitytojui.

Aukščiau pateiktas formules galima paaiškinti. Sekos $P_1;P_2; \dots; P_n$ išeitis (27.9) yra lygi atskirų procedūrų išeicių sąjungai. Sekos $P_1;P_2; \dots; P_n$ įeitis (26.8) lygi sąjungai, kur iš P_i įeities atimamos P_1, P_2, \dots, P_{i-1} išeitys.

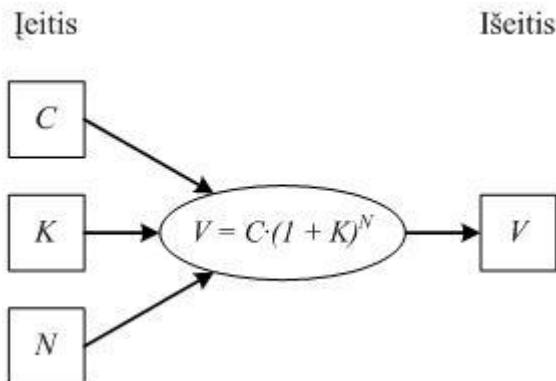
Nuoseklios kompozicijos esmė yra tame, kad anksčiau kviečiamos procedūros gamina rezultatus vėliau kviečiamoms. Kitais žodžiais, ankstesni sekos nariai „maitina“ vėlesnius.

27.3. *Įeities – išeities vaizdavimo pavyzdys*

Procedūros įeities-išeities pavaizdavimui imamas kitas pavyzdys. Klasikinė kapitalo augimo formulė:

$$V = C \cdot (1 + K)^N$$

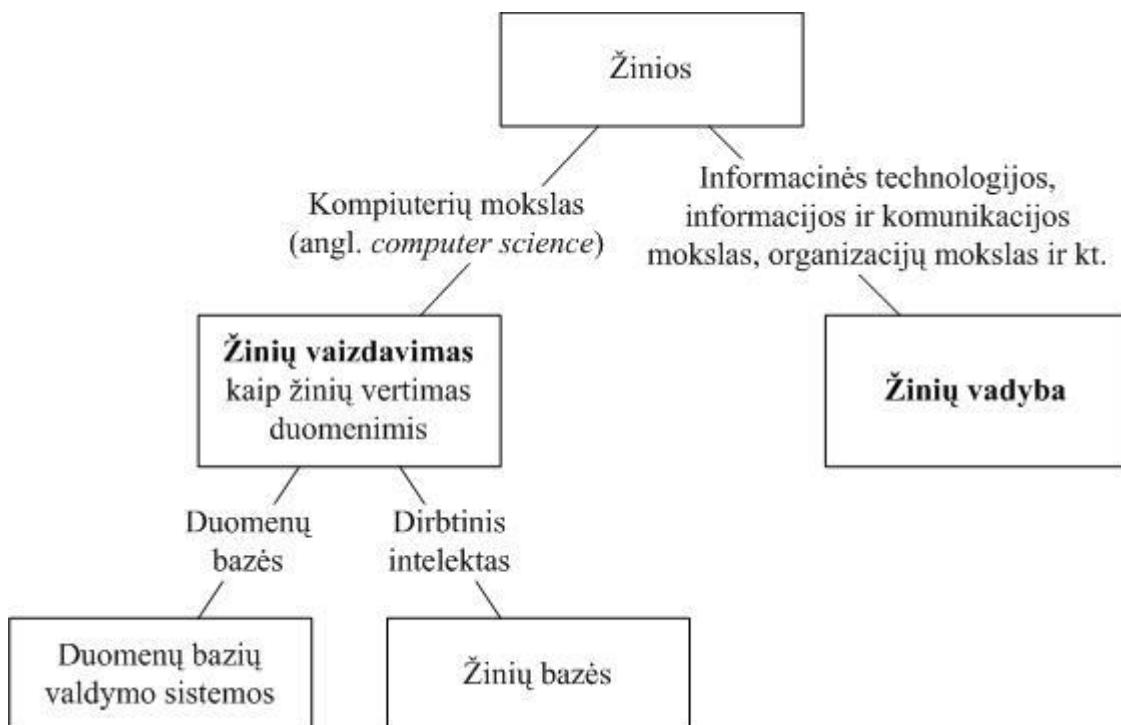
Čia V yra gaunama po N metų pinigų suma, įnešus į banką sumą C su palūkanų norma K (pavyzdžiu, $K=0,03$, kai metinės palūkanos yra 3 procentai). Šios formulės semantika pavaizduota 27.7 pav.



27.7 pav. Procedūros, kuri skaičiuoja kapitalo C vertę V po N metų esant palūkanoms K , įeities ir išeities pavaizdavimas.

28. Žinių vaizdavimas ir žinių vadyba

Kalbant apie žinias galima eiti dviem kryptimis: 1) žinių vaizdavimas, 2) žinių vadyba (angl. *knowledge management*) (žr. 28.1 pav.). Pirmoji kryptis paprastai priskiriama kompiuterių mokslui (angl. *computer science*), antroji – informacinių technologijų, informacijos ir komunikacijos mokslo ir organizacijų mokslo (angl. *organisation science*) šakoms. Žinių vadybos esmė yra didinti organizacijos vertę. Žinių vaizdavimo esmė, autoriaus nuomone, yra didinti kompiuterio intelekto ir žmogaus intelekto vertę.



28.1 pav. Žinių vaizdavimo ir žinių vadybos vieta kalbant apie žinias

Duomenų apdorojimas duomenų bazėse esmė yra išrinkimas (angl. *retrieval*). Žinių vaizdavimo ir dirbtinio intelekto esmė siejama su tokiomis sąvokomis kaip išplaukimasis, sąlygojimas (angl. *entailment*, *logical entailment*), samprotavimas (angl. *reasoning*). Tuo tarpu sąvoka samprotavimas (angl. *reasoning*) priskirtina žmogui ir sietina su išvedimu (angl. *inference*) dirbtiniame intelekte.

Sąvoka „išrinkimas“ siejama su algoritmu sudėtingumu, kuris yra polinominis, paprastai logaritminis, tiesinis, kvadratinis ar kubinis. Išvedimas dirbtiniame inteleekte siejamas su eksponentiniu sudėtingumu ar net algoritmiskai neišsprendžiamu uždavinių klase.

Kaip pavyzdys imamas išrinkimas duomenų bazių srityje. Reikia išrinkti įrašą atlyginimų lentelėje (žr. 28.2 lentelę) pagal užklausą, kurioje užduodama žmogaus pavardė, pavyzdžiu, Atlyginimas ("Butkaitis") =?

Pavardė	Atlyginimas (Lt)
Adoraitis	1000
Aldukaitis	900
Bajoraitis	1200
Butkaitis	800

Caraitis	1000
...	...
Žabaitis	1100

27.2 lentelė. Atlyginimų lentelė.

Laikoma kad atlyginimų lentelė surūšiuota pagal pavardes. Tada išrinkimo sudėtingumas gali būti sumažintas nuo tiesinio $O(N)$, kai peržiūrimi visi įrašai nuo pradžios iki galo, iki logaritminio $O(\ln N)$, kai naudojamas hešavimas (angl. *hashing*) arba indekso failas.

Dirbtinio intelekto metodų sudėtingumas yra eksponentinis $O(2^N)$. Pavyzdžiui, kelio radimas Manheteno labirinte, keliaujančio pirklio uždavinyje su N miestų. Dirbtiniame inteleekte nagrinėjami ir algoritmiškai neišsprendžiami uždaviniai.

29. Logikos vaidmuo samprotavime

Šis skyrius pradedamas pavyzdžiu iš [Brachman, Levesque 2004] įvado. Naujų žinių gavimas siejamas su loginiu išvedimu. Pavyzdžiui, tegu teisingi du teiginiai:

1. Pacientas Jonas yra alergiškas medikamentui M;

2. Kas alergiškas medikamentui M, tas alergiškas ir medikamentui M';

Iš šių teiginių išvedama, kad Jonui draudžiama skirti medikamentą M'.

Tai įrodoma šitaip:

Duota:

1. $\text{alerg}(\text{Jonas}, \text{M})$

2. $\forall x (\text{alerg}(x, \text{M}) \Rightarrow \text{alerg}(x, \text{M}'))$

Įrodyti: $\text{alerg}(\text{Jonas}, \text{M}')$

Įrodymas

I žingsnis:

$\forall x (\text{alerg}(x, \text{M}) \Rightarrow \text{alerg}(x, \text{M}'))$

Kintamasis x keičiamas konstanta Jonas

$\text{alerg}(\text{Jonas}, \text{M}) \Rightarrow \text{alerg}(\text{Jonas}, \text{M}')$

2 žingsnis

$\text{alerg}(\text{Jonas}, \text{M})$
 $\text{alerg}(\text{Jonas}, \text{M}) \Rightarrow \text{alerg}(\text{Jonas}, \text{M}')$

Taikoma įrodymo taisyklė *modus ponens*
(lot. teigimo būdas):

$P \Rightarrow Q$

$\text{alerg}(\text{Jonas}, \text{M}')$

Q

Tokiu būdu išvedama kad Jonas yra alergiškas medikamentui M', todėl Jonui draudžiama skirti M'.

Matematinėje logikoje nagrinėjama tik įrodymo struktūra, o ne turinys. Predikatai suprantami kaip teiginiai su parametrais. Čia galima priminti, kad predikatas tai funkcija, kurios reikšmė true arba false.

Toliau nagrinėjamas kitas išvedimo pavyzdys, turintis tokią pačią struktūrą. Tegu teisingi du teiginiai:

1. Sokratas yra žmogus;

2. Visi žmonės yra mirtingi;

Iš šių teiginių išplaukia, kad Sokratas yra mirtingas.

Tai įrodoma šitaip.

Duota:

1. $\text{žmogus}(\text{Sokratas})$

2. $\forall x (\text{žmogus}(x) \Rightarrow \text{mirtingas}(x))$

Įrodyti: $\text{mirtingas}(\text{Sokratas})$

Įrodymas

I žingsnis

$$\frac{\forall x \ (\text{žmogus}(x) \Rightarrow \text{mirtingas}(x)) \quad x \quad \text{keičiamas} \quad i \\ \text{žmogus}(\text{Sokratus}) \Rightarrow \text{mirtingas}(\text{Sokratus})}{\text{Sokratus}}$$

Pastaba: Kintamojo x keitimas konstanta A yra suprantamas kaip išvedimo taisyklė, vadinama *universal instantiation* (angl. *universal instantiation*; žr. [Klenk 2011]). Ji užrašoma:

$$\frac{\forall x \ P(x)}{P(A)} \quad \{A/x\}$$

Išraiška $\{A/x\}$ vadinama *keitiniu*.

2 žingsnis

$$\frac{\begin{array}{c} \text{žmogus}(\text{Sokratus}) \\ \text{žmogus}(\text{Sokratus}) \Rightarrow \text{mirtingas}(\text{Sokratus}) \end{array}}{\text{mirtingas}(\text{Sokratus})} \quad \text{Taikoma } \textit{modus ponens}$$

Tokiui būdu išvesta, kad Sokratus yra mirtingas.

Toliau nagrinėjamas dar vienas išvedimo pavyzdys, turintis tokią pačią struktūrą. Tegu teisingi du teiginiai:

1. Dzeusas yra Dievas;
2. Dievai yra nemirtingi;

Iš šių teiginių išplaukia, kad Dzeusas yra nemirtingas.

Tai įrodoma šitaip.

Duota:

1. dievas(Dzeusas)
2. $\forall x \ (\text{dievas}(x) \Rightarrow \neg \text{mirtingas}(x))$

Įrodyti: $\neg \text{mirtingas}(\text{Dzeusas})$

Įrodymas

I žingsnis

$$\frac{\forall x \ (\text{dievas}(x) \Rightarrow \neg \text{mirtingas}(x)) \quad x \quad \text{keičiamas} \quad i \\ \text{dievas}(\text{Dzeusas}) \Rightarrow \neg \text{mirtingas}(\text{Dzeusas})}{\text{Dzeusas}}$$

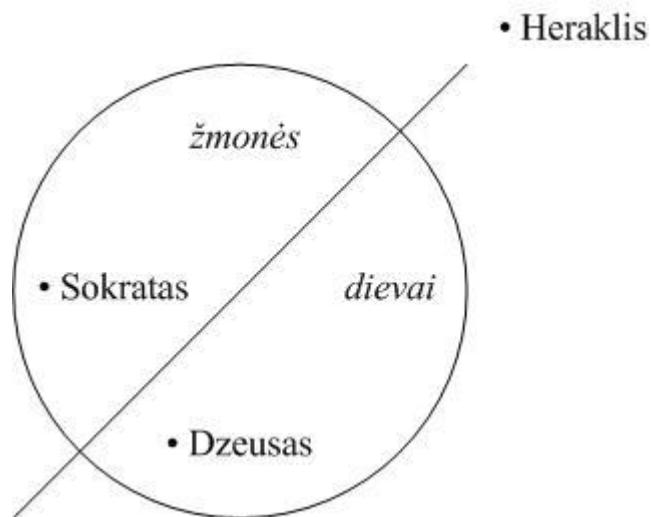
2 žingsnis

$$\frac{\begin{array}{c} \text{dievas}(\text{Dzeusas}) \\ \text{dievas}(\text{Dzeusas}) \Rightarrow \neg \text{mirtingas}(\text{Dzeusas}) \end{array}}{\neg \text{mirtingas}(\text{Dzeusas})} \quad \text{Taikoma } \textit{modus ponens}$$

Tokiu būdu išvesta, kad Dzeusas yra nemirtingas.

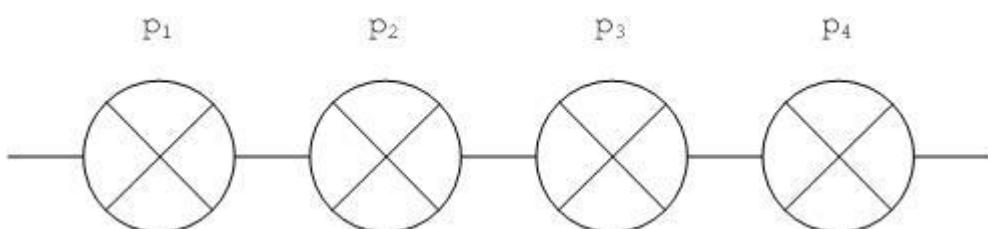
Kadangi teiginio „Sokratus yra dievas“ nėra pradinių teiginių aibėje, tai teiginys „Sokratus yra nemirtingas“ negali būti išvestas.

Remiantis požymiu mirtingas ar nemirtingas galima klasifikuoti būtybes į dvi aibes: žmonės ir dievai (žr. 29.1 pav.). Tačiau remiantis tokia klasifikacija yra problematiškas žinių modeliavimo klausimas – kuriai klasei priskirti pusdievius, pavyzdžiu Heraklį.



29.1 pav. Objektų aibės „dievai“ ir „žmonės“. Šių aibių sajunga yra predikato mirtingas apibrėžimo sritis

Logika nėra vaistas pati savaimė. Ji nenurodo uždavinio sprendimo kelio. Pavyzdžiui, tegu nedega keturių lempučių girlianda (žr 29.2 pav). Reikia nustatyti, kuri lemputė ar net kelios lemputės yra perdegusios.



29.2 pav. Tegu keturių lempučių girlianda nedega. Kuri lemputė (ar lemputės) yra perdegusi?

Logikos kalba $p_1 \& p_2 \& p_3 \& p_4 = \text{false}$. Čia $p_i = \text{false}$ žymi kad lemputė p_i yra perdegusi. Kadangi girlianda nedega, tai išplaukia, kad egzistuoja toks i , jog $p_i = \text{false}$. Formaliai, $\exists i \ p_i = \text{false}$. Tačiau neaišku kuris i . Be to, gali būti keletas tokų i . Gali būti $p_1 = \text{false}$ arba $p_2 = \text{false}$ arba $p_3 = \text{false}$ arba $p_4 = \text{false}$ arba $p_1 \& p_2 = \text{false}$ arba $p_1 \& p_3 = \text{false}$ ir t. t. nes gali būti perdegusios kelios lemputės.

30. Žinių vaizdavimo būdų apžvalga

Deklaratyvaus žinių vaizdavimo būdus yra priimta klasifikuoti [Sowa 2000; Brachman, Levesque 2004] į keturias grupes:

1. loginis žinių vaizdavimas;
2. procedūrinis žinių vaizdavimas (siauraja prasme);
3. tinklinis žinių vaizdavimas;
4. struktūrinis žinių vaizdavimas.

Toliau apžvelgiamos aukščiau išvardintos žinių vaizdavimo būdų grupės.

30.1. Loginis žinių vaizdavimas

Loginio žinių vaizdavimo būdų pagrindas:

1. teiginių logika; kitaip dar *teiginių skaičiavimas* (angl. *proposition calculus*);
 2. predikatų logika; kitaip dar *predikatų skaičiavimas* (angl. *predicate calculus*).
- Apibendrintai – loginio žinių vaizdavimo būdai yra grindžiami pirmos eilės logika.

30.2. Procedūrinis žinių vaizdavimas (siauraja prasme)

Procedūrinis žinių vaizdavimas siauraja prasme supaprastintai yra suprantamas kaip produkcijų sistema.

Produkcijų sistemos pavyzdys:

$\pi_1:$	$F, B \rightarrow Z$	Sintaksė:
$\pi_2:$	$C, D \rightarrow F$	kairioji pusė \rightarrow dešinioji pusė.
$\pi_3:$	$A \rightarrow D$	<i>if</i> antecedentas \rightarrow <i>then</i> konsekventas

Rodyklė „ \rightarrow “ yra interpretuojama kaip implikacijos operacija matematinėje logikoje.

30.3. Struktūrinis žinių vaizdavimas

Struktūrinis žinių vaizdavimas siejamas su freimo (angl. *frame*) sąvoka, kurią pasiūlė Marvinas Minskis. Freimas – tai duomenų struktūra skirta pavaizduoti stereotipinę situaciją, pavyzdžiui, buvimo kambaryje suvokimui arba samprotavimams apie atėjantį draugo gimtadienį. Freimas vaizduojamas tinklu iš mazgų ir ryšių. Struktūriniam žinių vaizdavimui gali būti priskiriami objektiniame programavime naudojami deklaratyvaus vaizdavimo būdai.

30.4. Tinklinis žinių vaizdavimas

Tinklinis žinių vaizdavimas (angl. *network representation schema*, *network representation*) apima tokias būdų grupes:

1. semantiniai tinklai (angl. *semantic networks*);
2. koncepciniai grafa (angl. *conceptual graphs*).

Abi šios sąvokos gali būti suprantamos labai neformaliai ir todėl yra giminingos. Yra autorių, kurie kalbdami apie tą patį naudoja arba pirmajį, arba antrajį terminą.

Semantinis tinklas yra grafas, kuriuo vaizduojamos klasifikavimo žinios apie objektus ir jų savybes. Viršūnėmis vaizduojamos sąvokos, o briaunomis ryšiai (angl. *relationships*).

Išskiriami šiu rūšių ryšiai, vadinami *universaliaisiais*:

1. *is-a*;
2. *instance-of*;
3. *part-of*.

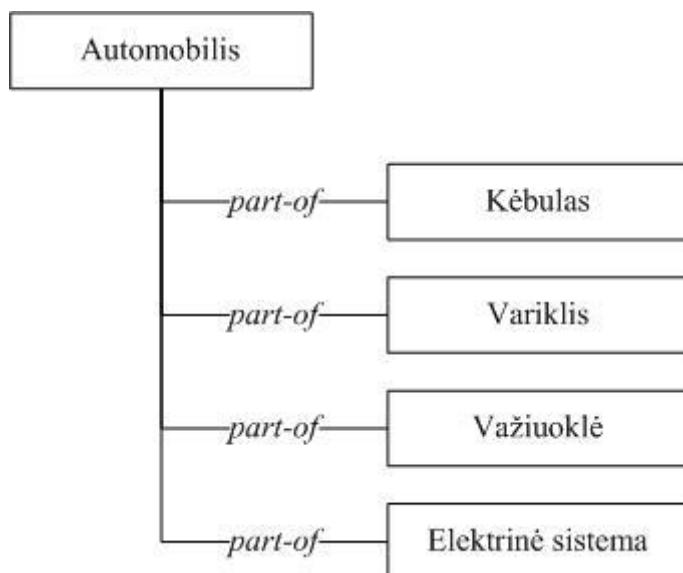
Tai bendrieji, nuo dalykinės srities nepriklausantys ryšiai.

Dalykinės srities žinių visuma remiasi konceptualizacija. Pastaroji apima objektus, sąvokas ir kitas esybes, kurias priimta laikyti egzistuojančiomis dalykinėje srityje, ir ryšius tarp objektų [Genesereth, Nilsson 1987]. Konceptualizacija yra abstraktus, supaprastintas pasaulio vaizdas, kurį norima pavaizduoti tam tikru tikslu.

Informatikoje terminu *ontologija* yra vadinama tam tikros dalykinės srities sąvokų visumos specifikavimas išreikštiniu pavidalu (angl. *explicit specification of a conceptualization*) [Gruber 1993]. Terminas ontologija yra paimtas iš filosofijos, kur ontologija yra suprantama kaip būties teorija, o epistemologija – pažinimo teorija. Supaprastintai galima sakyti, kad filosofijoje ontologija atsako į klausimą „kas (yra pasaulyje)“, o epistemologija – „kaip (žmogus pažista pasaulį)“.

Ontologiją informatikoje yra suprantama kaip sąvokų sąvadas. Tradiciškai sąvokų atributai nejeina į ontologiją.

Semantinis grafas 30.1 pav. pateiktas vaizduoja teiginį, kad automobilio sudėtinės dalys yra kėbulas, variklis, važiuoklė ir elektrinė sistema



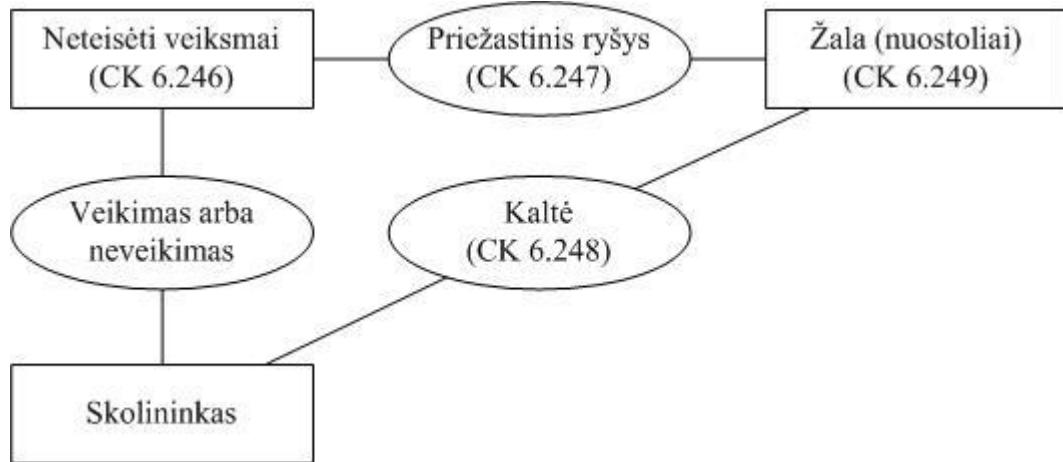
30.1 pav. Automobilių sudėtinės dalys yra kėbulas, variklis, važiuoklė ir elektrinė sistema

Semantinis grafas 30.2 pav. vaizduoja sąvokos „civilinė atsakomybė“ elementus. Šis semantinis grafas yra sudarytas remiantis Lietuvos Respublikos civilinio kodekso 6.245 straipsniu. Šis semantinis grafas pavaizduotas kaip dvidalis grafas. Viršūnėms ir briaunoms gali būti nesuteikiama griežta interpretacija. 29.2 pav. viršūnės vaizduoja sąvokas.

LR civilinio kodekso 6.245 straipsnis vadinas „Civilinės atsakomybės samprata ir rūšys“. Gretimi straipsniai skirti civilinės atsakomybės sąvokos elementams:

1. CK 6.246 str. „Neteisėti veiksmai“;
2. CK 6.248 str. „Kaltė kaip civilinės atsakomybės salygą“;

3. CK 6.247 str. „Priežastinis ryšys“;
4. CK 6.249 str. „Žala ir nuostoliai“.



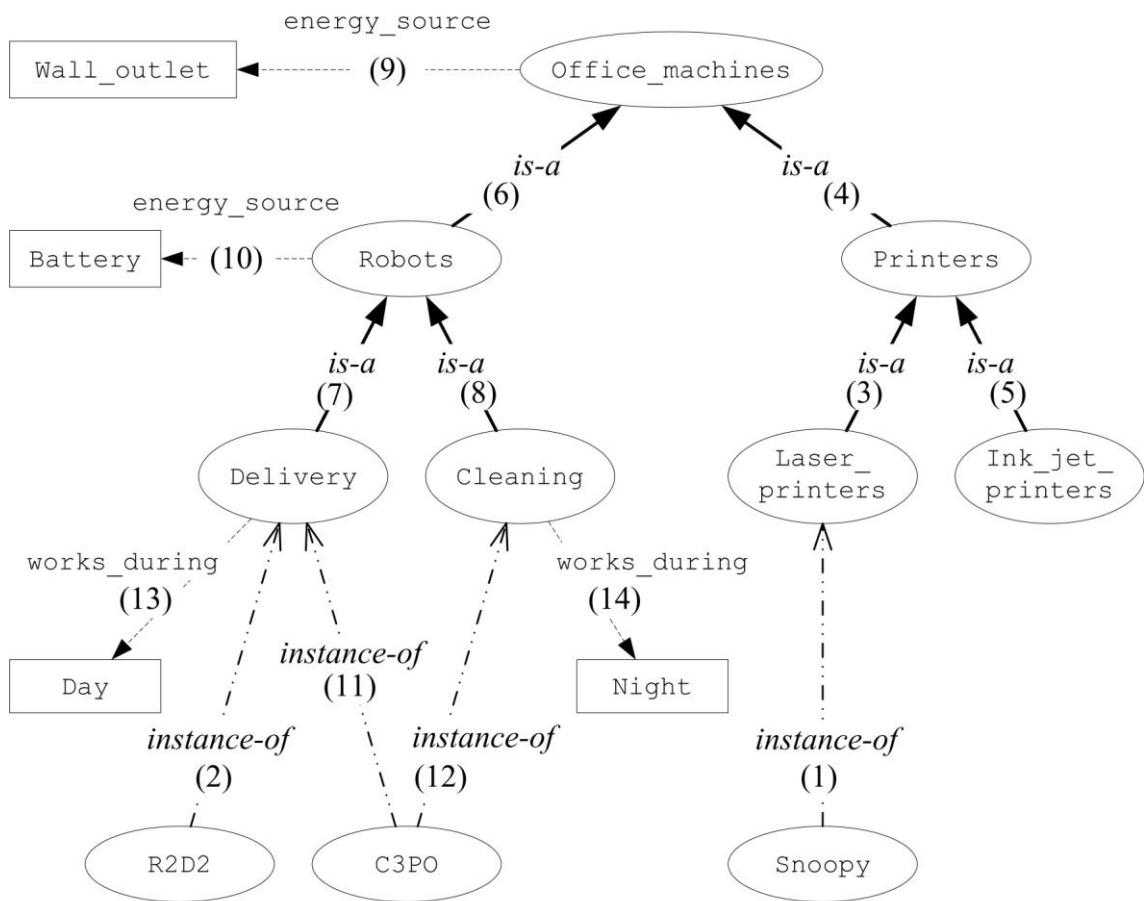
30.2 pav Sąvokos „civilinė atsakomybė“ semantinis grafas, sudarytas remiantis Lietuvos Respublikos civilinio kodekso 6.245 straipsniu

Semantinio tinklo viršūnėms ir briaunoms gali būti nesuteikiama griežta interpretacija. Toks semantikos nesuteikimas gali būti priimamas kaip silpnybė, kai kalbama apie žinių vaizdavimą kurį supranta kompiuteris. Tačiau tai priimama kaip stiprybė, kai kalbama apie žmogui skiriamą žinių vaizdavimą. Semantinis tinklas paprastai ir skiriamas žmogui.

31. Semantinis tinklas

Semantinio tinklo savoką 1909 m. pasiūlė amerikiečių matematikas Charles Peirce. Semantinių tinklų vaizdavimui nėra visuotinai priimto formalizmo kaip, pavyzdžiui, grafas. Ankstesniame skyriuje semantinis tinklas buvo apibrėžtas kaip grafas, kuriuo viršūnės vaizduoja savokas, o briaunos – ryšius. Pagrindinis semantinių tinklų privalumas – ne iki galio formalizuota semantika.

Toliau nagrinėjamas pavyzdys iš [Nilsson 1998 p. 308-313] 18.3 poskyrio „Žinių vaizdavimas tinklais“. Jame yra pateikiama tokia biuro technikos klasifikacija (žr. 30.1 pav.).



Žymėjimai:

- instance-of
- is-a
- > parametras, kitais žodžiais, atributas, savybė, požymis, slotas

31.1 pav. Semantinis tinklas vaizduoja biuro technikos klasifikaciją [Nilsson 1998, p. 308-313]

Biuro technika `Office_machines` klasifikuojama į robotus `Robots` ir spausdintuvus `Printers`. Robotai klasifikuojami į pristatymo `Delivery` ir valymo `Cleaning`. Spausdintuvai klasifikuojami į lazerinius `Laser_printers` ir rašalinius `Ink_jet_printers`. `Snoopy` yra lazerinių spausdintuvų egzempliorius. `R2D2` yra pristatymo robotų egzempliorius. `C3PO` yra robotų egzempliorius, kuris ir pristato, ir valo.

Biuro įrenginiai maitinami elektros srove iš rozetės, t.y. jų atributas `energy_source` turi reikšmę `Wall_outlet`. Robotai maitinami elektros srove iš baterijos, t.y. jų atributas `energy_source` turi reikšmę `Battery`. Pristatymo robotai dirba dieną, t.y. jų `works_during` atributo reikšmė yra `Day`. Valymo robotai dirba naktį, t.y. jų `works_during` atributo reikšmė yra `Night`.

30.1 pav. sunumeruoti ryšiai vaizduojami teiginiais, kurie užrašomi formulėmis:

- (1) `Laser_printer(Snoopy)`
- (2) `Delivery_robot(R2D2)`
- (3) $\forall x [Laser_printer(x) \Rightarrow Printer(x)]$
- (4) $\forall x [Printer(x) \Rightarrow Office_machine(x)]$
- (5) $\forall x [Ink_jet_printer(x) \Rightarrow Printer(x)]$
- (6) $\forall x [Robot(x) \Rightarrow Office_machine(x)]$
- (7) $\forall x [Delivery_robot(x) \Rightarrow Robot(x)]$
- (8) $\forall x [Cleaning_robot(x) \Rightarrow Robot(x)]$
- (9) $\forall x [Office_machine(x) \Rightarrow energy_source(x) = Wall_outlet]$
- (10) $\forall x [Robot(x) \Rightarrow energy_source(x) = Battery]$
- (11) `Delivery_robot(C3PO)`
- (12) `Cleaning_robot(C3PO)`
- (13) $\forall x [Delivery_robot(x) \Rightarrow works_during(x) = Day]$
- (14) $\forall x [Cleaning_robot(x) \Rightarrow works_during(x) = Night]$

Reikia pastebėti kad semantiniame grafe yra naudojama daugiskaita, pvz. `Robots`, o logikos formulėse predikatų vardai naudojami vienaskaitoje pvz. `Robot`. Tokiu būdu tarp sąvokų semantiniame grafe ir formulėse yra izomorfizmas.

Tiek remiantis (1), (3), (4), (9) ryšiais, tiek remiantis (1), (3), (4), (9) formulėmis logikos pagalba yra išvedama kad Snoopy maitinamas iš rozetės:

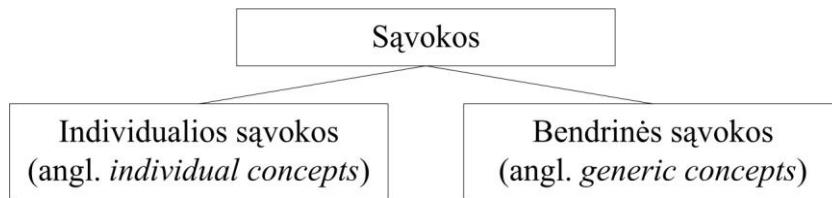
`Energy_source(Snoopy) = Wall_outlet`

Šiame pavyzdyme kyla problema išsiaiškinant kada dirba robotas C3PO: dieną ar naktį? Formaliai `works_during(C3PO) = Day` ar `Night`. Remiantis (11) ir (13) C3PO dirba dieną, `works_during(C3PO) = Day`. Remiantis (12) ir (14) C3PO dirba naktį, `works_during(C3PO) = Night`. Šiame pavyzdyme konceptualizuojama kad žinių bazėje atributui `works_during` skirta viena reikšmė. Ši situacija iliustruoja žinių bazių skirtumą nuo logikos. Logikas atsakytu kad formuliu (1)–(14) aibė prieštaringa. Žinių bazių specialistas pasiūlytų:

1. nustatyti reikšmę pagal nutylėjimą (angl. *by default*);
2. prioritetą tarp (11) ir (12);
3. euristiką pasirinkimui tarp (11) ir (12);
4. konceptualizuoti atributą `works_during` kaip daugiareikšmį, t.y. tipo `set-of {Day, Night}`.

Formalizujant semantinių grafo paprastai naudojamas grafo formalizmas. Yra išprasta naudoti dvidalių grafo (angl. *bipartite graph*). Primenama, kad grafas G yra pora: viršunių aibė V ir briaunų aibė E , formaliai – $G = \langle V, E \rangle$, kur $E \subset V \times V$. Kaip minėta, semantinio grafo

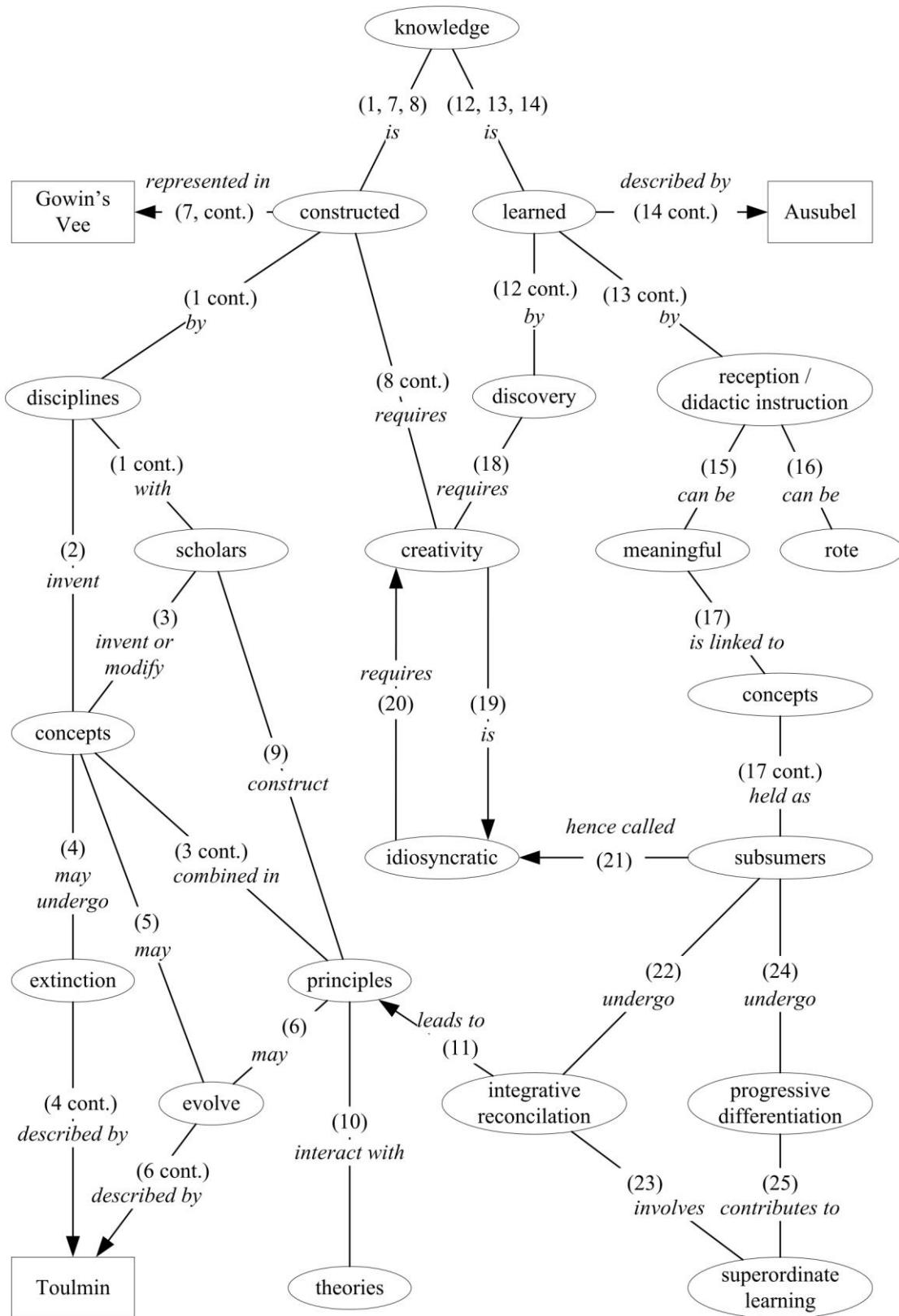
viršūnės vaizduoja savokas, o briaunoms yra suteikiama tam tikra prasmė. Savokos klasifikuojamos į individualias, pvz. Snoopy, ir bendrąsias, pvz. Robots.



31.2 pav. Savokos klasifikuojamos į individualias, pvz. Snoopy, ir bendrines, pvz. Robots.

Toliau nagrinėjamas semantinio grafo pavyzdys, sudarytas remiantis [Geldenhuys 1999, p. 13] (žr. 31.3 pav). Jis parodo pagrindinius pagrindinius teiginius apie žinių išgavimą ir sudarymą. Semantinis tinklas 31.3 pav. „koduoją“ viršūnėmis ir briaunomis šiuos sakinius:

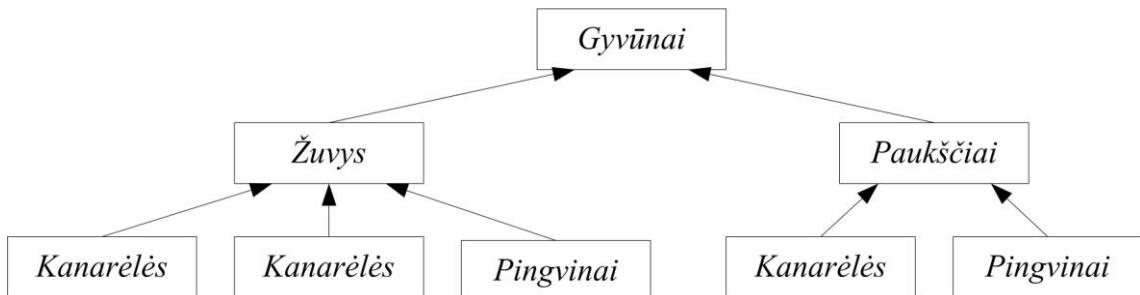
1. Knowledge is constructed by disciplines with scholars.
2. Disciplines invent concepts.
3. Scholars invent or modify concepts combined in principles.
4. Concepts may undergo extinction described by Toulmin.
5. Concepts may evolve.
6. Principles may evolve – described by Toulmin.
7. Knowledge (is) constructed represented in Govins's Vee.
8. Knowledge (is) constructed requires creativity.
9. Scholars construct principles.
10. Principles interact with theories.
11. Integrative reconciliation leads to principles.
12. Knowledge is learned by discovery.
13. Knowledge is learned by reception/didactic instruction
14. Knowledge is learned described by Ausubel.
15. Reception/didactic instruction can be meaningful.
16. Reception/didactic instruction can be rote (iškalimas).
17. Meaningful is linked to concepts held as subsumers.
18. Discovery requires creativity.
19. Creativity is idiosyncratic.
20. Idiosyncratic requires creativity.
21. Subsumers hence call idiosyncratic.
22. Subsumers undergo integrative reconciliation.
23. Integrative reconciliation involves superordinate learning.
24. Subsumers undergo progressive differentiation.
25. Progressive differentiation contributes to superordinate learning.



31.3 pav. Semantinis tinklas sudarytas remiantis [Geldenhuys 1999]. Jis parodo pagrindinius žinių išgavimo ir sudarymo teiginius

32. Bendrinė ir individuali sąvoka

Tarp bendrinių sąvokų yra *is-a* yra ryšys (angl. *relationship*). Tarp individualios sąvokos ir bendrinės yra *instance-of* ryšys. 32.1 pav. vaizduojamas labai supaprastintas gyvūnų klasifikavimas.

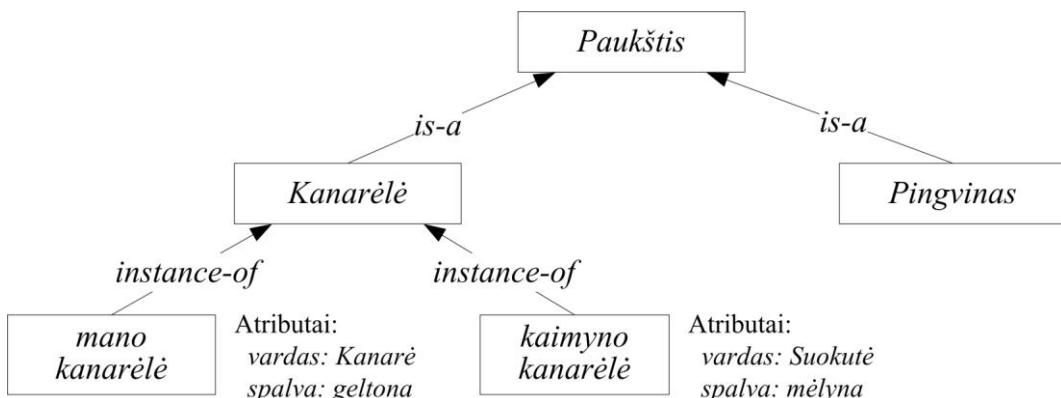


32.1 pav. Gyvūnų klasifikavimo pavyzdys

Šiame grafe pavaizduotos klasės ir jų ryšiai. Informatikoje įprasta klasės vardą rašyti vienaskaita, pavyzdžiui, *Gyvūnas*, *Žuvis*, *Paukštis*.

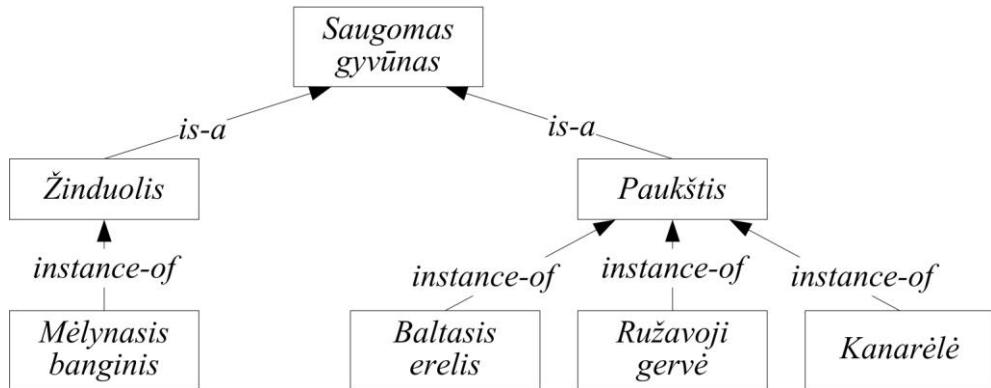
Galima iškelti klausimą: kas yra kanarélė – ar klasė, ar egzempliorius?

Pirmausia yra priimta modeliuoti, kad kanarélė yra klasė, žr. 32.2 pav. Klasė *Kanarélė* yra ryšyje *is-a* su klase *Paukštis*.



32.2 pav. *Kanarélė* kaip bendrinė sąvoka, o *mano kanarélė* – individuali

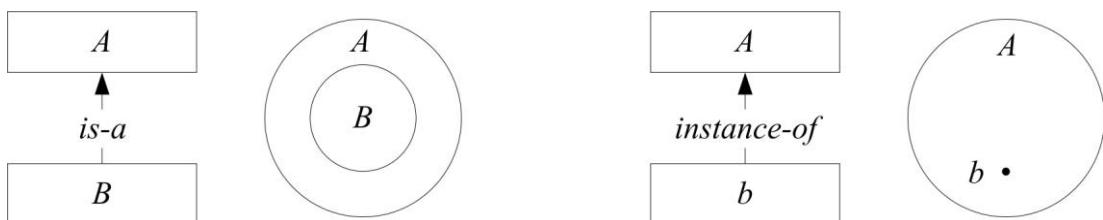
Kanarélė gali būti modeliuojama ir kaip individuali sąvoka, žr. 31.3 pav. Čia egzempliorius *Kanarélė* yra ryšyje *instance-of* su klase *Paukštis*. Ji žymi čia visas kanarèles pasaulyje kaip nykstančią rūšį. Šiame pavyzdyme medžio lapai suprantami kaip individualios sąvokos. Kiekviena saugoma rūsis suprantama kaip individuas. Kaip matyti, ta pati sąvoka viename kontekste gali būti suprantama kaip bendrinė, kitame kaip individuali.



32.3 pav. *Kanarélė* kaip individuali sąvoka. Ji yra ryšyje *instance-of* su klase *Paukštis*

Buvimas ryšyje gali būti žymima infikcine notacija $B \text{ is-a } A$ arba prefiksine notacija $\text{is-}a(B,A)$. Abiem atvejais suprantama kaip predikatas – būti ar nebūti ryšyje.

Ryšiui $B \text{ is-a } A$ suteikiama semantika poaibis-aibė, t.y. $B \subset A$. Ryšiui $b \text{ instance-of } A$ suteikiama semantika aibės elementas-aibė, t.y. $b \in A$. Reikia pastebėti, kad abiem atvejais žmogus gali ištarti vienodai: „ B yra A “ (angl. “ B is A ”) ir „ b yra A “ (angl. “ b is A ”), nors prasmė yra skirtina.



32.4 pav. Interpretacija: A – bendrinė sąvoka, B – bendrinė sąvoka ir b – individuali sąvoka. Semantika aibių kalba: $B \subset A$ ir $b \in A$

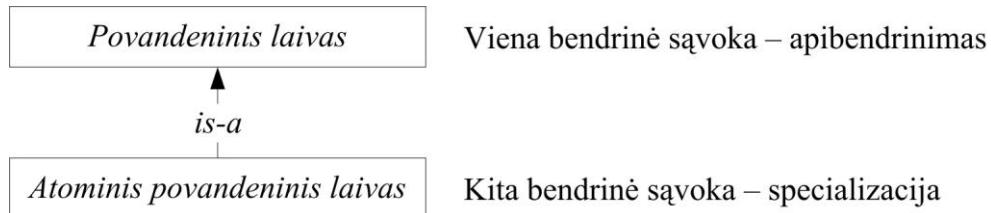
32.1. Ryšio is-a interpretacijos

Šis ir sekantis poskyriai parengti remiantis [Brachman 1988].

Ryšys *is-a* gali turėti tokias interpretacijas.

1. Poaibis ir viršaibis (angl. *subset/superset*).

Kai viršūnės vaizduoja aibes, tai briauna tarp viršūnių žymi poaibio santykį. Pavyzdžiu, teiginys „Atominis povandeninis laivas yra povandeninis laivas“ turi tokią interpretaciją *Atominis povandeninis laivas* \subset *Povandeninis laivas* (žr. 32.5 pav.). Kitaip tariant, „Kiekvienai esybei x , jeigu x priklauso atominiams povandeniniams laivams, tai x priklauso ir povandeniniams laivams“.



32.5 pav. Ryšys *is-a* kaip poabisi-viršaibis

2. Apibendrinimas ir specializacija (angl. *generalization/specialization*).

Apibendrinimas, kaip ryšys tarp predikatų išreiškia implikaciją. Pavyzdžiu, tegu sąvoka *Povandeninis laivas* yra sąvokos *Atominis povandeninis laivas* apibendrinimas. Tada *is-a* reiškia „kiekvienai esybei x , jeigu *Atominis_povandeninis_laivas*(x), tai *Povandeninis_laivas*(x)“.

Apibendrinimas ir specializacija yra suprantami taip, kaip objektiniame programavime. Sąvoka B yra sąvokos A *specializacija* ir A yra B *apibendrinimas*. Jeigu sąvokos formalizuojamos predikatais, tai yra teisinga $\forall x B(x) \Rightarrow A(x)$, kur \Rightarrow žymi implikaciją logikos prasme.

Kai sąvokos A ir B apibrėžiamos atributų rinkiniais, tai aukščiau nurodyta implikacija suprantama kaip implikacija tarp atitinkamų A ir B atributų.

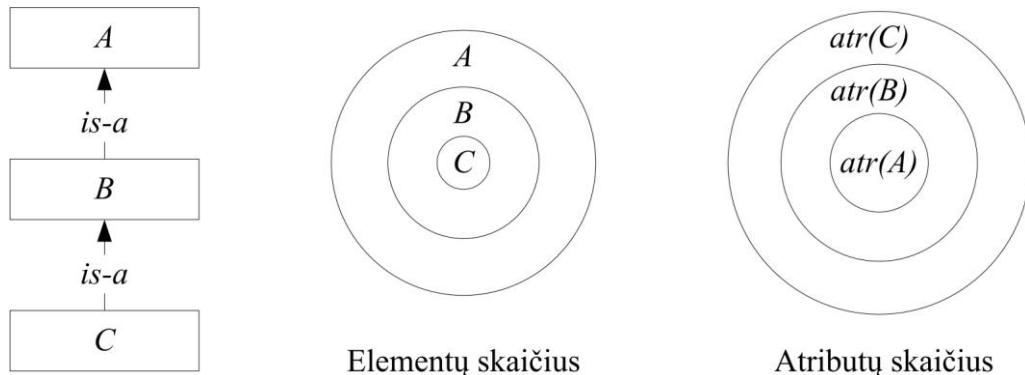
Labiau specifinę sąvoką B atitinkanti aibė turi mažiau elementų negu jos apibendrinimas A :

$$B \subset A \quad (32.1)$$

Tačiau labiau specifinę sąvoką B turi daugiau atributų negu A :

$$atr(B) \supset atr(A) \quad (32.2)$$

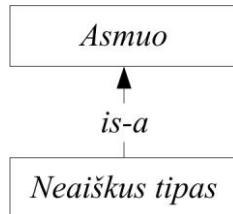
Čia *atr(sąvoka)* žymi sąvokos atributų aibę. Pastebime, kad atributų įdėjimas (32.2) yra *priešinga* kryptimi, negu aibės elementų skaičius (32.1). Labiau specifinę sąvoką pasižymi daugiau atributų. Tai parodyta 32.6 pav. Pavyzdžiu, sąvoka A turi atributus E ir F , o sąvoka B be E ir F dar ir papildoma atributą G , kitaip tariant, $E \& F \Rightarrow A$ ir $E \& F \& G \Rightarrow B$.



32.6 pav. Tipinė *is-a* ryšio interpretacija: Ryšys *is-a* suprantamas kaip apibendrinimas, jeigu žiūrima iš apačios į viršų, t. y. rodyklės kryptimi, ir specializavimas – jeigu žiūrima iš viršaus į apačią, t. y. prieš rodyklę. Tai tipinis klasifikavimo ryšys

3. AKO (angl. *a-kind-of*).

Šią interpretaciją iliustruoja sąvoka *Neaiškus_tipas* (angl. *stranger*). Sakinys „Neaiškus tipas yra asmuo“ žymi ryšį *Neaiškus_tipas is-a Asmuo* su interpretacija AKO.



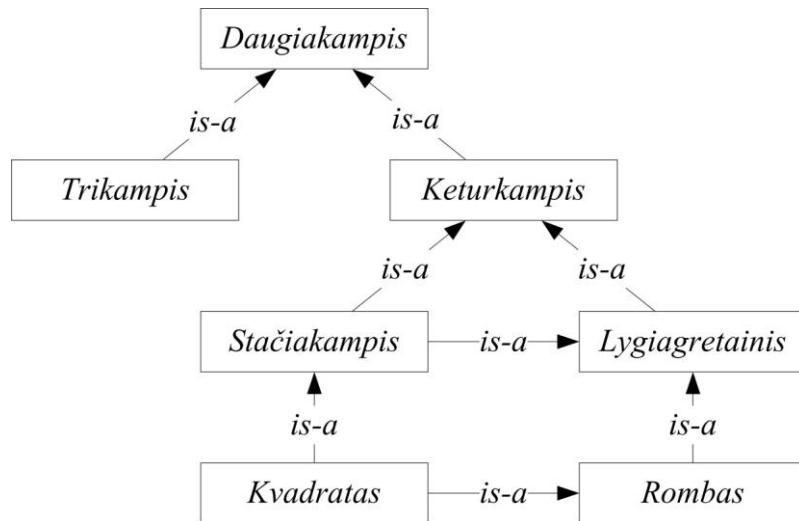
32.7 pav. Sakiniui „Neaiškus tipas yra asmuo“ žymi ryšį suteikiama interpretacija AKO

Šioje interpretacijoje sąvokai nesuteikiamas klasės vaidmuo. Tiesiog įvardijama buitiškai: „neaiškus tipas“. Bet paliekama galimybė paveldėti.

Pavyzdžiui, i degalinę užsukusį asmenį kasininkas iš pradžių gali priimti kaip neaiškų tipą. Nežinomas jo apsilankymo degalinėje tikslas: ar apsipirkti, ar apsižvalgyti, ar vogti. Jeigu išsirinkęs prekę, jis paduoda kreditinę kortelę, tai jis yra priimamas kaip pirkėjas.

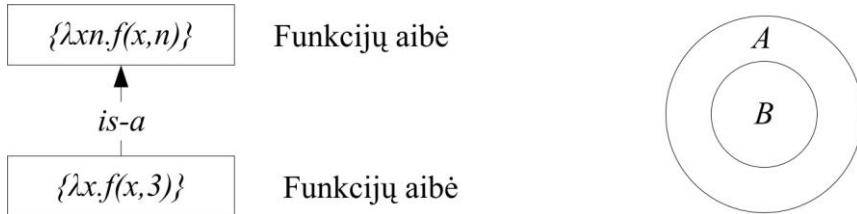
4. Koncepcinis įdėjimas (angl. *conceptual containment*).

Pavyzdžiui, „Trikampis yra daugiakampis turintis tris kraštines“, „Keturkampis yra daugiakampis turintis keturias kraštines“. Ryšiai tarp geometrinių figūrų daugiakampio, trikampio, keturkampio, lygiagretainio, rombo, stačiakampio ir kvadrato yra pateikiami 32.8 pav. Koncepcinis įdėjimas – iš apačios į viršų. Apėmimas – iš viršaus į apačią.



32.8 pav. Ryšiai *is-a* tarp geometrinių figūrų

Kitas pavyzdys yra λ -abstrakcija, naudojama funkcinio programavimo teorijoje. Pavyzdžiui, tegu yra nagrinėjamos dviejų argumentų funkcijos $\{\lambda xn.f(x,n)\}$. Šios funkcijų aibės specializacija yra vieno argumento funkcijų aibė $\{\lambda x.f(x,3)\}$, kuri gaunama sukonkretinus $n=3$ (žr. 32.9 pav.).



32.9 pav. *is-a* ryšys tarp dviejų aibiu, kurių elementai yra funkcijos

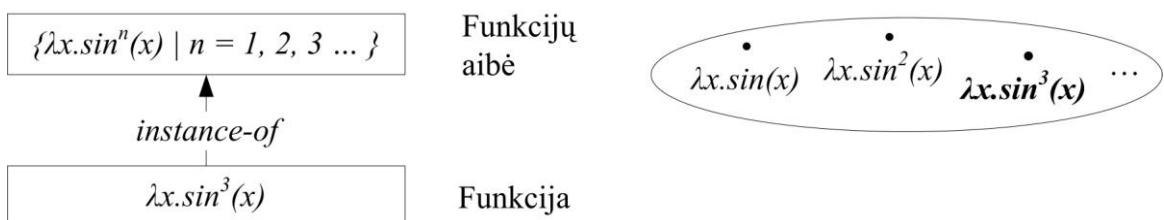
32.10 pav. demonstruoja *instance-of* (o ne *is-a*) ryšį tarp funkcijos $\lambda x. \sin^3(x)$ ir funkcijų šeimosa (t.y. aibės) $\{\lambda x. \sin^n(x) / n=1, 2, 3 \dots\}$. Reikia pastebėti, kad pastarają funkcijų aibę reikia skirti nuo vienos funkcijos $\lambda n. \lambda x. \sin^n(x)$. Pastaroji priklauso nuo parametruo n .

Funkcija turi apibrėžimo sritį ir kitimo sritį. Tai aibės. Funkcija (sinonimas „atvaizdavimas“) yra trejetas: apibrėžimo sritis, kitimo sritis, ir atvaizdavimo taisyklė.

I fiksuočių dviejų argumentų x ir n funkciją f galima žiūrėti trejopai.

1. Tai dviejų argumentų funkcija $\lambda x. f(x, n)$. Apibrėžimo sritis yra Dekarto sandaugos poaibis, t.y. aibės, kuriai priklauso x , ir aibės, kuriai priklauso n , Dekarto sandaugos poaibis.
2. Antra, tai funkcijų šeima $\{\lambda x. f(x, n) / n=1, 2, 3 \dots\}$. Tai funkcijų aibė.
3. Trečia tai funkcija $\lambda n. \lambda x. f(x, n)$ nuo argumento n ir kitimo sritimi antrajame punkte. Šis atvaizdavimas matematikoje žymimas $n \rightarrow \lambda x. f(x, n)$. Kitaip tariant, argumentui n priskiriama reikšmė $\lambda x. f(x, n)$, kuri savo ruožtu yra funkcija nuo x .

Skirtingai nuo 32.9 pav., kur vaizduojamas *is-a* ryšys, 32.10 pav. vaizduoja *instance-of* ryšį. 32.9 pav. vaizduoja *is-a*, nes f žymi bet kokią funkciją, kaip *Dramblys* žymi dramblio sąvoką. 32.10 pav. vaizduoja *instance-of*, nes f sukonkretinta.



32.10 pav. Ryšys *instance-of* tarp funkcijos $\lambda x. \sin^3(x)$ ir funkcijų aibės $\{\lambda x. \sin(x), \lambda x. \sin^2(x), \lambda x. \sin^3(x), \dots\}$

Programuojant dviejų argumentų funkciją $\lambda x. \sin^n(x)$ naudojama kėlimo laipsniu operacija. Programuojant specializuotą funkciją $\lambda x. \sin^3(x)$ kėlimas laipsniu kaip brangi operacija yra keičiamas sandauga $\lambda x. \sin(x) \cdot \sin(x) \cdot \sin(x)$. Tegu pastarosios funkcijos vadinamos FSIN ir FSIN3. Vadovaujantis trečiuoju požiūriu programuojama funkcija $\lambda n. \lambda x. f(x, n)$, žemiau pavadinta FEN.

```
procedure FSIN(x: real, n: real): real;
begin
    return sin(x) **n;
end
```

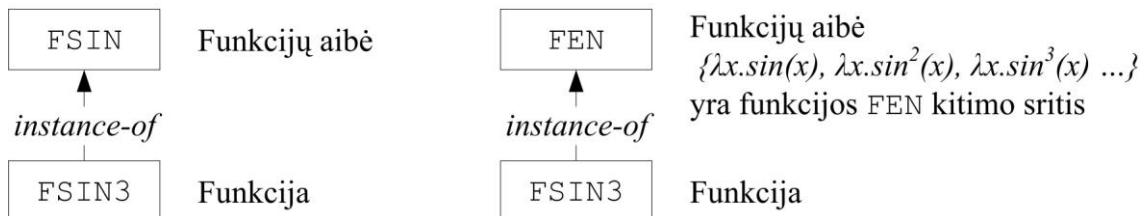
```

procedure FSIN3(x: real): real;
begin
    return sin(x) * sin(x) * sin(x);
end

procedure FEN(n: integer): procedure (x: real) real;
begin
    return sin(x)**n;
end

```

Požiūris į FSIN3 ir FSIN kaip esančias *instance-of* ryšyje (bet ne *is-a* ryšyje) yra modeliuojamas 32.11 pav. Kaireje. Tokiu modeliavimo būdu į FSIN žiūrima kaip į funkcijų šeimą nuo n, o ne kaip į funkciją nuo dviejų argumentų x ir n.



32.11 pav. *instance-of* ryšys tarp FSIN3 ir FSIN

Į FSIN3 ir FEN irgi galima žiūrėti kaip esančias *instance-of* ryšyje (žr. 32.11 pav. dešinėje). Čia vadovaujamasi trečiuoju požiūriu (žr. aukščiau ir 32.10 pav.).

Kitas pavyzdys iliustruoja ryšį tarp polinominių funkcijų. Tegu P yra funkcija n-tojo laipsnio polinomas $\lambda x^n.x^n+x^{n-1}+x^{n-2}+\dots+x^2+x+1$:

```

procedure P(x: real, n: real): real      { n ≥ 1 }
begin
    var rez := 1;
    for i := 1 to n
        rez := rez * x + 1;
    return rez;
end

```

Tegu P3 yra P, kai $n=3$ ir yra programuojama be ciklo:

```

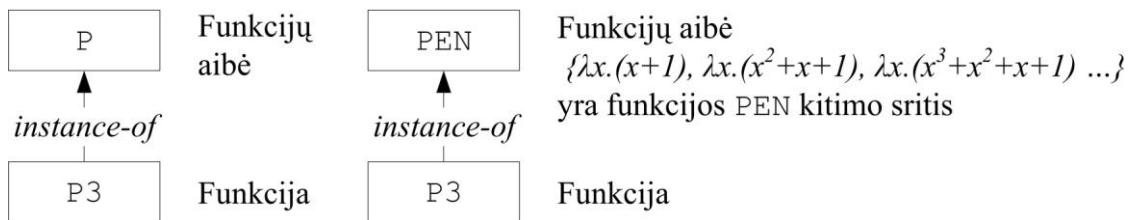
procedure P3(x: real): real
begin
    return x*x*x + x*x + x + 1;
end

```

Tegu PEN atitinka FEN iš ankstesnio pavyzdžio.

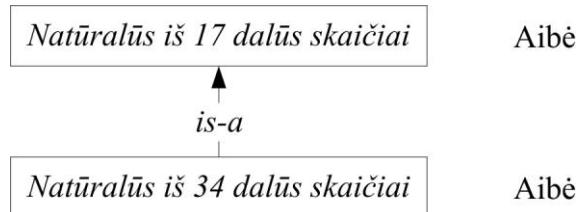
```
procedure PEN(n: integer): procedure (x: real) real;
begin
  var rez := 1;
  for i := 1 to n
    rez := rez * x + 1;
  return rez;
end
```

Požiūris į P3 ir P kaip esančias *instance-of* ryšyje (o ne *is-a* ryšyje) yra modeliuojamas 32.12 pav. Ir vėl tokiu modeliavimo būdu į P žiūrima kaip į funkcijų šeimą nuo n, o ne kaip į funkciją nuo dviejų argumentų x ir n.



32.12 pav. *instance-of* ryšys tarp P3 ir P

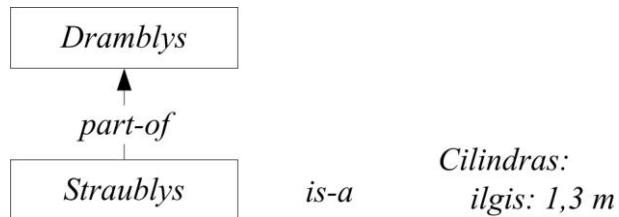
Dar vienas pavyzdys – ryšys *is-a* tarp dalių skaičių aibių. Natūralūs iš 34 dalūs skaičiai {34, 68, 102 ...} yra ryšyje *is-a* su iš 17 daliais skaičiais {17, 34, 51, 68, 85, 102, 119 ...} (žr. 32.13 pav.). Skaičiai dalūs iš 17 nėra įvardinti atskira sąvoka, kaip lyginiai ir nelyginiai.



32.13 pav. *is-a* ryšys tarp skaičių dalių iš 34 ir 17

5. Rolės reikšmės apribojimas.

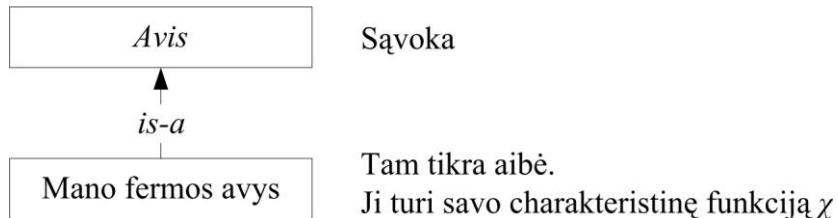
Interpretuojama kaip tipas sloto reikšmei užpildyti. Pavyzdžiui, „Dramblio straublys yra 1,3 m. ilgio cilindras“. Šio teiginio esmė, kad rolė (atributas, savybė) *Straublys* yra tam tikro tipo – būtent *Cilindras* (žr. 32.14 pav.). Tokia *is-a* interpretacija nėra klasifikavimo ryšys.



32.14 pav. Dramblio straublys yra (*is-a*) cilindras

6. Aibė ir jos charakterinė funkcija.

Tai néra klasifikavimo ryšys. Tai pvz. ryšys tarp tam tikros dramblių aibės ir *Dramblys* kaip sąvokos. Šitaip susiejama šios aibės charakterinė funkcija su sąvoka. Pavyzdžiuui, tegu kalbama apie avis savo fermoje. Tada turimas omenyje ryšys tarp visų avių fermoje ir avies sąvokos (žr. 32.15 pav.). Turima omenyje, kad laikomos ne karvės ir ne ožkos. Atitinkama charakterinė funkcija nusako, pavyzdžiuui, kad kaimyno avis Nr. 1234 nepriklauso mano fermos avims: $\chi_{\text{manoFermosAvys}}(\text{kaimynoAvis}1234) = \text{false}$.



32.15 pav. *is-a* kaip ryšys tarp avių mano fermoje aibės ir avies sąvokos

Aibės *A* charakterinė funkcija χ (tariama chi) apibrėžiama šitaip:

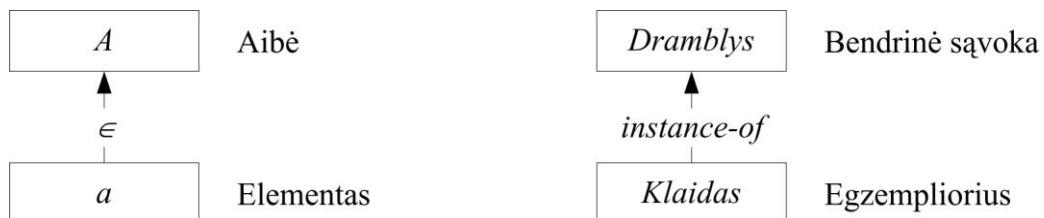
$$\begin{aligned}\chi_A(x) &= \text{true}, \text{ jeigu elementas } x \text{ priklauso aibei } A \\ &\text{false priešingu atveju}\end{aligned}$$

Pavyzdžiu: $\chi_{\text{dalūsių}}(34) = \text{true}$, $\chi_{\text{dalūsių}}(3) = \text{false}$.

32.2. Ryšio *instance-of* interpretacijos

Ryšys *instance-of* gali turėti tokias interpretacijas.

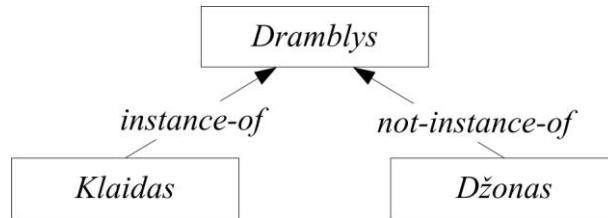
- Elemento ir aibės ryšys; $a \in A$ (žr. 32.16 pav). Pavyzdžiuui, „Klaidas yra dramblys“ reiškia „Klaidas yra dramblių aibės elementas“.



32.16 pav. $a \in A$ kaip *instance-of* ryšys. Tai išreiškia „Klaidas yra dramblys“

2. Predikatas.

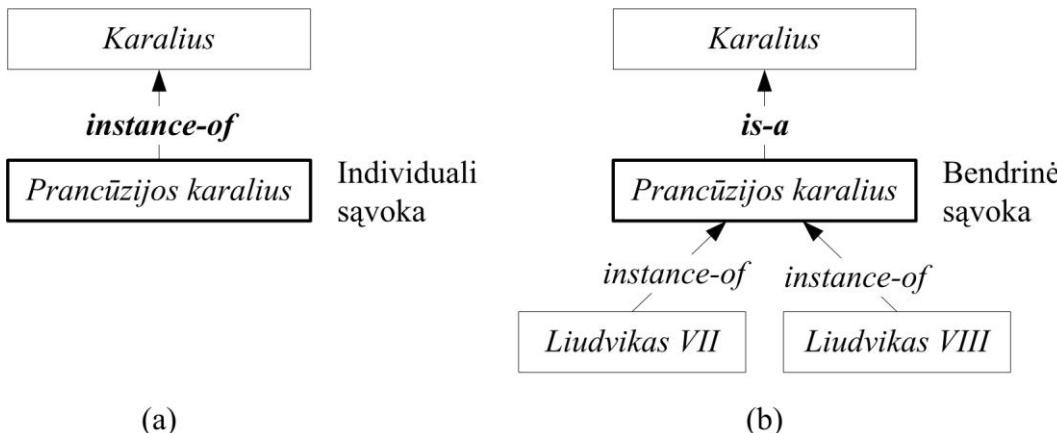
Pavyzdžiuui „Klaidas yra dramblys“ implikuoja predikatą $\text{Dramblys}(\text{Klaidas}) = \text{true}$. „Džonas néra dramblys“ reiškia $\text{Dramblys}(\text{Džonas}) = \text{false}$. „Klaidas néra povandeninis laivas“ reiškia $\text{Povandeninis_laivas}(\text{Klaidas}) = \text{false}$. Aptariamas predikatas atitinka aibės charakterinę funkciją χ .



32.17 pav. „Klaidas yra dramblys“ implikuoja predikatą Dramblys (Klaidas) = true. „Džonas nėra dramblys“ implikuoja Dramblys (Džonas) = false

3. Konceptinis įdėjimas (angl. *conceptual containment*).

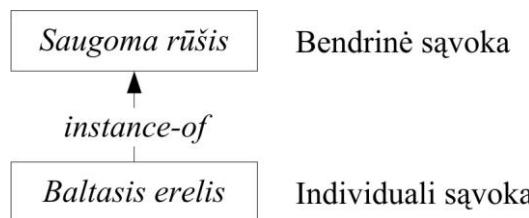
Pavyzdžiui, teiginys „Prancūzijos karalius yra karalius“ modeliuojamas *instance-of* ryšiu (32.18 a pav.). Tačiau reikia pastebėti, kad modeliavimas priklauso nuo konteksto. Jeigu kalbama apie Liudviką VII ir Liudviką VIII, tai *Prancūzijos karalius* modeliuojamas kaip bendrinė sąvoka (32.18 b pav.).



32.18 pav. Tarp sąvokų gali būti ir *instance-of*, ir *is-a* ryšys: (a) *instance-of* modeliuoja „Prancūzijos karalius yra karalius“; (b) *is-a*

4. Abstrakcija.

Iš saknio „Baltasis erelis yra saugoma rūšis“ ištraukiamą individualią sąvoką *Baltasis erelis*, bendrinę sąvoką *Saugoma rūšis* ir abstrakciją reiškiantis ryšys *instance-of* (32.19 pav.).



32.19 pav. „Baltasis erelis yra saugoma rūšis“ kaip *instance-of* ryšys

33. Klasifikavimo vaizdavimas

Pradedama nuo pavyzdžio, demonstruojančio klasifikavimą specifinėje dalykinėje srityje – mokesčių teisėje. Mokesčių administravimo įstatymo [MAI] 13 straipsnis nustato mokesčių sąrašą Lietuvoje. Siekiant iliustruoti šią dalykinę sritį, sąrašas pateikiamas nesutrumpintas.

13 straipsnis. Mokesčiai

Pagal ši įstatymą administruojami mokesčiai:

1. pridėtinės vertės mokestis;
2. akcizas;
3. gyventojų pajamų mokestis;
4. nekilnojamojo turto mokestis;
5. žemės mokestis;
6. mokestis už valstybinius gamtos išteklius;
7. naftos ir dujų išteklių mokestis;
8. mokestis už aplinkos teršimą;
9. konsulinis mokestis;
10. žyminis mokestis;
11. (neteko galios nuo 2005 m. liepos 1 d.);
12. paveldimo turto mokestis;
13. privalomojo sveikatos draudimo įmokos;
14. įmokos į Garantinį fondą;
15. valstybės rinkliava;
16. loterijų ir azartinių lošimų mokestis;
17. mokesčiai už pramoninės nuosavybės objektų registravimą;
18. pelno mokestis;
19. valstybinio socialinio draudimo įmokos;
20. pertekliaus mokestis cukraus sektoriuje;
21. gamybos mokestis cukraus sektoriuje;
22. (neteko galios nuo 2007 m. balandžio 26 d.);
23. muitai;
24. atskaitymai nuo pajamų pagal Lietuvos Respublikos miškų įstatymą;
25. mokestis už valstybės turto naudojimą patikėjimo teise;
26. socialinis mokestis;
27. cukraus pramonės restruktūrizavimo laikinasis mokestis;
28. papildomos baltojo cukraus gamybos kvotos ir pridėtinės izogliukozės gamybos kvotos vienkartinio išsipirkimo mokestis.

Informatika galėtų iškelti klausimą, ar šie mokesčiai klasifikuojami į kokias nors nepersikertančias klasės? Mokesčių teisei skirtoje knygoje [Marcijonas, Sudavičius 2003, p. 18] rašoma: „Pagal mokesčių mokėtojų ypatybes skiriami juridinių asmenų, fizinių asmenų (gyventojų) ir bendri mokesčiai. Juridiniai asmenys moka pelno, fiziniai asmenys (gyventojai) – pajamų mokestį; bendriems mokesčiams priklauso žyminis mokestis, valstybės rinkliava, nuompinigiai (užmokestis) už valstybinę žemę bei valstybinio fondo vandens telkinius ir t. t.“. Iš šios citatos išplaukia, kad aukščiau išvardinti mokesčiai negali būti suskaidyti į dvi nepersikertančias aibes, pirma, mokesčiai, kuriuos moka juridiniai asmenys, ir, antra, mokesčiai, kuriuos moka fiziniai asmenys. Persikirtimas yra įvardintas kaip bendri mokesčiai.

Gaunamas tokis aibės „mokesčiai“ suskaidymas į tris poaibius:

1. juridinių asmenų mokesčiai;

mokesčis.mokėtojo_tipas = {'juridinis_asmuo'}

JA=1 & FA=0

2. fizinės asmenų mokesčiai;

```
mokestis.moketojo_tipas = {'fizinis_asmuo'}
JA=0 & FA=1
```

3. bendri mokesčiai.

```
mokestis.moketojo_tipas = {'juridinis_asmuo', 'fizinis_asmuo'}
JA=1 & FA=1
```

Čia **type** `mokestis.moketojo_tipas` = **subset_of**
`{'juridinis_asmuo', 'fizinis_asmuo'}.2 bitai: JA ir FA.`

33.1. Algebrinis požiūris į klasifikavimą

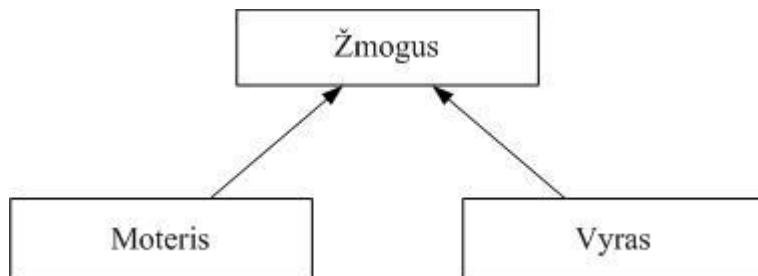
Tegu turima aibė $A=\{a_1, a_2, \dots\}$.

33.1 apibrėžimas. Aibės A suskaidymu (angl. *partition*) $A=\{A_1, A_2, \dots, A_m\}$ yra vadinama aibės A poaibių aibė tokia, kad:

- 1) poaibiai nesikerta (angl. *disjoint*): $A_i \cap A_j = \emptyset$, kai $i \neq j$;
- 2) poaibių aibė yra išsemianti (angl. *exhaustive*): $\forall a \in A, \exists i \in \{1, 2, \dots, m\}$ tokis, kad $a \in A_i$. Kitais žodžiais, kiekvienam aibės A elementui a egzistuoja tokis poaibis $A_i \subset A$, kad jo elementas yra a .

Toliau pateikiamas pavyzdys, aibės Žmogus suskaidymas $ZMOGUS=\{\text{Moteris}, \text{Vyras}\}$. Skaidoma pagal atributą žmogus.lytis. I suskaidymo elementą *Moteris* (kaip aibės Žmogus poaibi) patenka tokie elementai $x \in \text{Žmogus}$, kurių atributas lytis turi reikšmę 'moteris', t.y. $x.lytis='moteris'$. I suskaidymo elementą *Vyras* patenka tokie elementai, kurių atributas lytis turi reikšmę 'vyras', t.y. $x.lytis='vyras'$.

Klasifikavimas pavaizduotas 33.1 pav.



33.1 pav. Klasifikavimas kaip suskaidymas: $\text{Žmogus}=\text{Moteris} \cup \text{Vyras}$;
 $ZMOGUS=\{\text{Moteris}, \text{Vyras}\}$

Aibė paprastai yra skaidoma pagal kokį nors *ekvivalentumo* santykį.

33.2 apibrėžimas. Santykis \sim tarp aibės A elementų yra ekvivalentumo santykis, tada ir tik tada, kai jis yra:

- 1) *tranzityvus*: iš $x \sim y$ ir $y \sim z$ sekā $x \sim z$, kur $x \neq y, y \neq z$;
- 2) *refleksyvus*: $x \sim x$ kiekvienam $x \in A$.

33.3 apibrėžimas. Aibės A suskaidymas pagal joje apibrėžtą ekvivalentumo santykį \sim vadinamas faktoriaibe ir žymimas A/\sim .

I vieną suskaidymo elementą (jis yra aibės A poaibis) patenka visi tarpusavyje ekvivalentūs aibės A elementai.

Aibės faktorizavimas į faktoriaibę yra vienas iš būdų paimti jos suskaidymą.

33.4 pavyzdys. Čia pateikiamas natūraliųjų skaičių aibės faktorizavimo pavyzdys.

Imamas ekvivalentumo santykis natūraliųjų skaičių aibėje $N=\{0,1,2,3,4,\dots\}$. $x \sim y$ jeigu dalijant iš 3 jų liekanos yra lygios, t.y. $(x \bmod 3) = (y \bmod 3)$.

Pagal šį ekvivalentumo santykį aibės A faktoraibė yra sudaryta iš trijų elementų: $\{0,3,6,9,\dots\}$, $\{1,4,7,10,\dots\}$ ir $\{2,5,8,11,\dots\}$. Formaliai:

$$A/\sim = \{\{0,3,6,9,\dots\}, \{1,4,7,10,\dots\}, \{2,5,8,11,\dots\}\}$$

Pabrėžiama, kad aibės suskaidymo elementai, savo ruožtu yra aibės, t.y. aibės A poaibiai.

33.2. Santykio sąvoka

Santykis (rus. *отношение*, angl. *relation*) yra griežta matematinė sąvoka.

33.5 apibrėžimas. Binarinis santykis R tarp aibės A elementų yra Dekarto sandaugos $A \times A$ poaibis, t.y. $R \subset A \times A$.

Binarinis santykis gali būti suprantamas kaip dvivietis predikatas $r(x, y)$, kur $r(x, y) = \text{true}$, kai x ir y yra santykyje R , ir $r(x, y) = \text{false}$, kai x ir y nėra santykyje R .

Binarinis santykis vaizduojamas dviejų stulpelių lentele.

Pavyzdžiui, 33.4 pavyzdyme apibrėžtas ekvivalentumo santykis $x \sim y$, jeigu dalijant iš 3 jų liekanos yra lygios, t.y. $(x \bmod 3) = (y \bmod 3)$, vaizduojamas tokia Dekarto sandauga:

$$\begin{aligned} &\{(0,0), (0,3), (0,6), \dots, (3,0), (3,3), (3,6), \dots, (6,0), (6,3), (6,6), \dots, (9,0), (9,3), (9,6), \dots, \\ &(1,1), (1,4), (1,7), \dots, (4,1), (4,4), (4,7), \dots, (7,1), (7,4), (7,7), \dots, (10,1), (10,4), (10,7), \dots, \\ &(2,2), (2,5), (2,8), \dots, (5,2), (5,5), (5,8), \dots, (8,2), (8,5), (8,8), \dots, (11,2), (11,5), (11,8), \dots\} \end{aligned}$$

Ši Dekarto sandauga pavaizduota 33.2 lentelėje. Pastaroji padalinta į tris stulpelius, kurie ir vaizduoja natūraliųjų skaičių suskaidymą į tris poaibius: $\{0,3,6,9,\dots\}$, $\{1,4,7,10,\dots\}$ ir $\{2,5,8,11,\dots\}$.

$(x \bmod 3) = 0$ $(y \bmod 3) = 0$	$(x \bmod 3) = 1$ $(y \bmod 3) = 1$	$(x \bmod 3) = 2$ $(y \bmod 3) = 2$
0~0	1~1	2~2
0~3	1~4	2~5
0~6	1~7	2~8
0~9	1~10	2~11
0~12	1~13	2~14
...
3~0	4~1	5~2
3~3	4~4	5~5
3~6	4~7	5~8
3~9	4~10	5~11
3~12	4~13	5~14
...
6~0	7~1	8~2
6~3	7~4	8~5
6~6	7~7	8~8
6~9	7~10	8~11
6~12	7~13	8~14
...
9~0	10~1	11~2
9~3	10~4	11~5
9~6	10~7	11~8
9~9	10~10	11~11
9~12	10~13	11~14
...

33.2 lentelė. Natūraliųjų skaičių aibės faktorizavimas pagal ekvivalentumo santykį: $x \sim y$, jeigu dalijant iš 3 jų liekanos yra lygios, t.y. $(x \bmod 3) = (y \bmod 3)$

Šiame pavyzdyste santykį nusakanti lentelė yra begalinė.

Santykio ir ryšio sąvokos yra skiriamos. Santykis yra tarp aiškiai įvardintos aibės elementų. Kai kalbame apie ryšį, tai turimas omens aukštesnis lygmuo. Ryšys gali būti suprantamas kaip santykis. Ryšys paprastai yra tarp tam tikrų esybių (sąvokų), kurių ekstencionalo aibė ir intencionalo aibė nėra griežtai fiksuojamos. Ryšiu tarp x ir y priskiriama ir kitų savybių, pvz., $1:n$, $m:n$, o ne tik buvimas santykije.

Elementai x ir y yra santykije R tada ir tik tada, kai teisingas tam tikras predikatas $r(x, y)$.

Pateikiamas įrodymas, kad 33.4 pavyzdyste apibrėžtas ekvivalentumo santykis $x \sim y$, yra ekvivalentumo santykis. Įrodymas pagal apibrėžimą.

Pirma įrodomas tranzityvumas.

$x \sim y$, kai $(x \bmod 3) = (y \bmod 3)$. Tai galima perrašyti: $x = 3 \cdot k_1 + r$ ir $y = 3 \cdot k_2 + r$. Analogiškai $y \sim z$ galima perrašyti: $y = 3 \cdot k_2 + r$ ir $z = 3 \cdot k_3 + r$. Kadangi visų liekana r vienoda, tai teisinga lygybė $(x \bmod 3) = (z \bmod 3)$. Iš čia seká $x \sim z$. Tai ir reikėjo įrodyti.

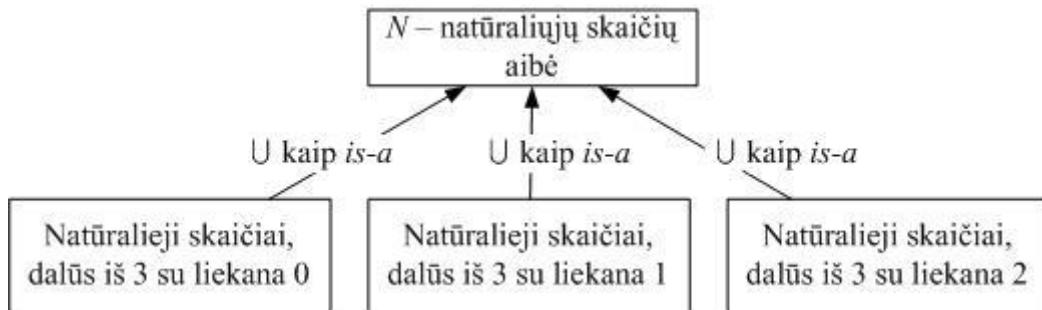
Toliau įrodomas refleksyvumas. $x \sim x$, nes $(x \bmod 3) = (x \bmod 3)$.

Kadangi ekvivalentumas yra santykis, tai jis suprantamas kaip tam tikras predikatas $\text{equiv}(x, y)$. $\text{equiv}(x, y) = \text{true}$, kai yra tenkinama kažkokia salyga. Pavyzdžiu, x dalinant iš trijų ir y dalinant iš trijų gaunama ta pati liekana. $\text{equiv}(x, y) = \text{false}$, kai ekvivalentumo salyga nėra patenkinta, pavyzdžiu, liekanos skirtingos.

Toliau aiškinamas faktoraibės pavaizdavimas.

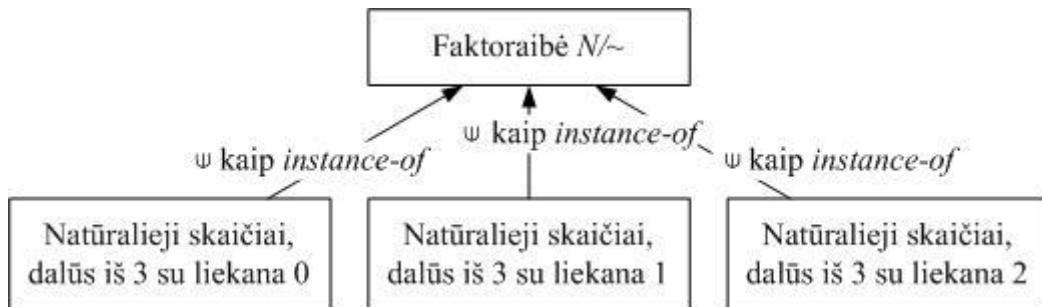
Aibės $A = N = \{0, 1, 2, 3, 4, \dots\}$, suskaidymas pagal apibrėžtą santykį \sim yra $A = A/\sim = \{A_1, A_2, A_3\}$, kur $A_1 = \{0, 3, 6, 9, \dots\}$, $A_2 = \{1, 4, 7, 10, \dots\}$, $A_3 = \{2, 5, 8, 11, \dots\}$.

A_1 , A_2 ir A_3 yra natūraliųjų skaičių aibės poaibiai, t.y. $A_1 \subset A$, $A_2 \subset A$, $A_3 \subset A$. Tai vaizduojama 33.3 pav.



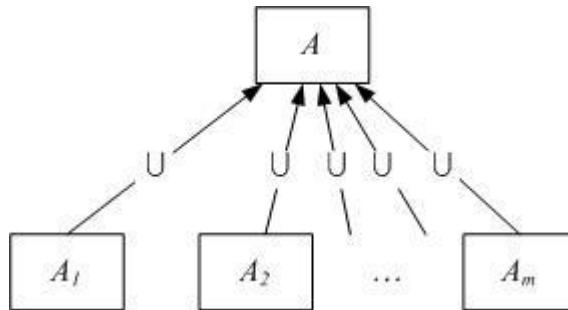
33.3 pav. Natūraliųjų skaičių aibės suskaidymas pagal ekvivalentiškumo santykį

Savo ruožtu A_1 , A_2 ir A_3 yra faktoraibės A elementai, t.y. $A_1 \in A$, $A_2 \in A$, $A_3 \in A$. Tai vaizduojama 33.4 pav.

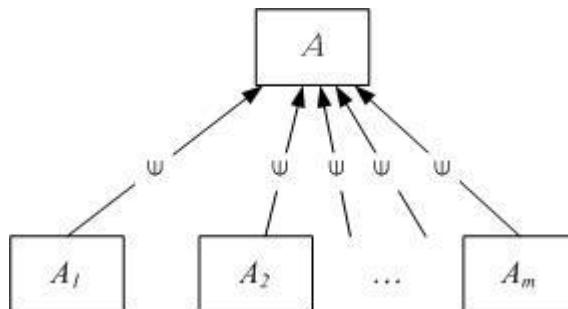


33.4 pav. Natūraliųjų skaičių faktoraibės elementai

Bendru atveju $A/\sim = \{A_1, A_2, A_3, \dots, A_N\}$. Čia $A_i \in A/\sim$ ir $A_i \subset A$.



33.5 pav. Suskaidymo $A=\{A_1, A_2, \dots, A_m\}$, kur $A_i \subset A$ pavaizdavimas. Viršūnėje aibė A



33.6 pav. Kitoks suskaidymo $A=\{A_1, A_2, \dots, A_m\}$, kur $A_i \subset A$ pavaizdavimas ir tokiu būdu $A_i \in A$. Viršūnėje yra suskaidymas A

Santykis gali būti ne tik dvivietis, jis gali būti ir trivietis, ir daugia vietis.

33.6 pavyzdys. Santykis Skrydis (Kas_skrenda, Iš_kur, Į_kur, Kada) gali būti nustatomas lentele (žr. 33.7 lentelę).

Kas_skrenda	Iš_kur	Į_kur	Kada
Adomas	Vilnius	Londonas	2009-03-14
Birutė	Kaunas	Paryžius	2009-05-27

33.7 lentelė. Keturvietį santykį Skrydis (Kas_skrenda, Iš_kur, Į_kur, Kada) nustatanti lentelė

33.7 teorema. Jeigu objektų x ir y atributo atr reikšmės sutampa, t.y. $x.atr = y.atr$, tai objektai yra ekvivalentiškumo santykyje.

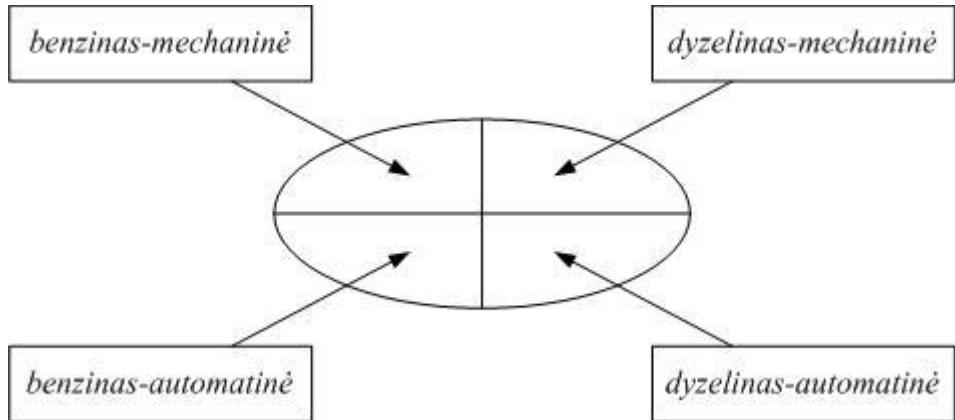
Įrodoma pagal apibrėžimą. Įrodymas paliekamas skaitytojui kaip pratimas.

33.8 pavyzdys. Imamas spausdintuvų atributas spausdinimo_tipas. Tegu jo reikštių aibė yra {lazerinis', 'rašalinis', 'adatinis', 'kitoks'}. Apibrėžiamos ekvivalentiškumo santykis $x \sim y$, kai $x.spausdinimo_tipas = y.spausdinimo_tipas$.

Tada organizacijoje esančių spausdintuvų aibė { s_1, s_2, \dots, s_n } suskaidoma į poaibius lazeriniai_spausdintuvai, rašaliniai_spausdintuvai, adatiniai_spausdintuvai ir kitokie_spausdintuvai.

33.9 pavyzdys. Automobilių aibė suskaidoma pagal atributą spalva. Kiek spalvų, tiek poaibų: raudoni, geltoni, žali ir t.t.

33.10 pavyzdys. Automobilių aibė suskaidoma pagal du atributus: *kuras* ir *greičių_dėžė*. Tegu atributo *kuras* reikšmių aibė yra {*'benzinas'*, *'dyzelinas'*}, ir atributo *greičių_dėžė* reikšmių aibė yra {*'mechaninė'*, *'automatinė'*}. Ekvivalentišumo santykis gaunamas konjunkcijos operacija tarp keleto atributų: $x \sim y$ tada ir tik tada, kai $(x.kuras = y.kuras) \ \& \ (x.greiciu_dede = y.greiciu_dede)$. Automobilių aibė suskaidoma į keturis poaibius: *benzin-mechaninė*, *dyzelin-mechaninė*, *benzin-automatinė*, *dyzelin-automatinė*. (žr. 33.8 pav.).



33.8 pav. Automobilių aibės suskaidymas pagal ekvivalentišumo santykį \sim .
 Čia $x \sim y$ tada ir tik tada, kai $(x.kuras = y.kuras) \ \& \ (x.greiciu_dede = y.greiciu_dede)$. Automobilių aibė, pavaizduota elipse, yra suskaidoma į keturis poaibius

33.3. Tiesinės erdvės faktorizavimas pagal tiesinį poerdvį

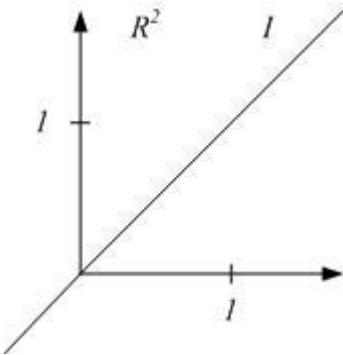
Matematikos šakoje abstrakčiojoje algebroje yra nagrinėjamas tiesinės erdvės suskaidymas į faktoraibę pagal tiesinį poerdvį. Toks skaidymas vadinamas tiesinės erdvės faktorizavimu.

Žemiau nagrinėjamas pavyzdys, kai imama dvimatiė tiesinė erdvė R^2 ir jos tiesinis perdvoris, žymimas I . Tiesinis perdvoris I apibrėžiamas kaip R^2 poaibis, kuriam galioja: jeigu $x \in I$, ir $y \in I$, tai $x+y \in I$, ir $t \cdot x \in I$, kur t yra realus skaičius. Tiesinis perdvoris gali būti imamas tokio pavidalio: $I = \{t \cdot v, \text{ kur } t \in R\}$. Čia v yra vadinamas baziniu vektoriumi.

Analogiškai gaunamas dvimatis perdvoris trimatėje erdvėje virš dviejų bazinių vektorių v_1 ir v_2 , t.y. plokštuma:

$$I = \{t_1 \cdot v_1 + t_2 \cdot v_2, \text{ kur } t_1 \text{ ir } t_2 \text{ yra realūs skaičiai, t.y. } t_1 \in R \text{ ir } t_2 \in R\}$$

Dvimatėje Euklidinėje erdvėje R^2 tiesinį poerdvį I sudaro, pavyzdžiui, 45 laipsnių kampu einanti tiesė, t.y. aibė vektorių, turinčių pavidalą (t,t) , kur t yra realus skaičius. Tokiu būdu, $I = \{t \cdot (1,1), \text{ kur } t \in R\}$ (žr. 33.9 pav.). Galima irodyti, kad tokia aibė I yra tiesinis perdvoris. Reikia irodyti, pirma, dviejų vektorių x ir y iš I suma $x+y$ priklauso I , antra, vektoriaus x sandauga iš skaliaro k priklauso I .



33.9 pav. Tiesinis poaibis $I = \{t \cdot (1,1), t \in R\}$ dvimatėje erdvėje R^2

Įrodomas. Iš $x \in I$ ir $y \in I$, sekā kad $x = t_1 \cdot (1,1) = (t_1, t_1)$ ir $y = t_2 \cdot (1,1) = (t_2, t_2)$. Tada:

1. $x+y = (t_1, t_1) + (t_2, t_2) = (t_1+t_2, t_1+t_2) = (t_1+t_2) \cdot (1,1) \in I$.
2. $k \cdot x = k \cdot (t_1, t_1) = (k \cdot t_1, k \cdot t_1) \in I$.

Įrodymo pabaiga.

Tiesinėje erdvėje ekvivalentiškumo santykis ~ apibrėžiamas šitaip.

33.11 apibrėžimas. Vektoriai x ir y yra ekvivalentiški ($x \sim y$) tada ir tik tada kai jų skirtumas priklauso tiesiniams poerdviui:

$$x \sim y \Leftrightarrow (x-y) \in I$$

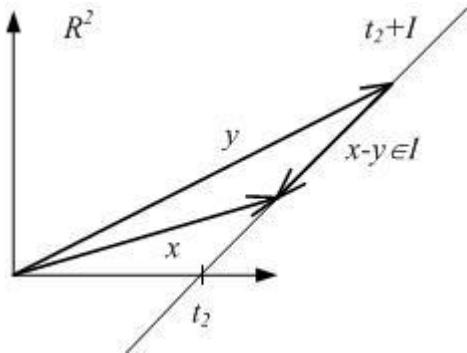
Įrodomas, kad aukščiau apibrėžtas santykis yra ekvivalentiškumo santykis, t.y. kad tenkina tranzityvumo ir refleksyvumo aksiomas.

Tranzityvumas. Turima $x \sim y$ ir $y \sim z$. Reikia įrodyti $x \sim z$. Tegu $x = (x_1, x_2)$, $y = (y_1, y_2)$ ir $z = (z_1, z_2)$. Iš $x \sim y$ sekā ir $x-y = (t_1, t_1)$. Iš čia sekā $x = y+(t_1, t_1)$. Iš $y \sim z$ sekā ir $y-z = (t_2, t_2)$. Iš čia sekā $y = z+(t_2, t_2)$. Tokiu būdu $x = y+(t_1, t_1) = z+(t_2, t_2)+(t_1, t_1) = z+(t_2+t_1, t_2+t_1)$. Iš čia sekā $x-z = (t_2+t_1, t_2+t_1) \in I$. Kadangi $x-z \in I$, tai $x \sim z$.

Refleksyvumas. Reikia įrodyti $x \sim x$. Kadangi $x-x = (0,0) \in I$, tai $x \sim x$.

Įrodymo pabaiga.

Faktoraibės R^2/\sim elementas yra žymimas t_2+I . Tai R^2 poaibis, turintis pavidalą $\{(t_2+t, t), t \in R\}$ (žr. 33.10 pav.). Kitais žodžiais, tai tiesė, pasvirusi 45° kampu ir nutolusi atstumu t_2 nuo taško $(0,0)$. Iš visų ši R^2 poaibų žiūrima kaip į atskirą vienetą, dar vadinančią R^2/\sim tašku. Tokiu būdu į tiesę žiūrima kaip į faktoraibės elementą.

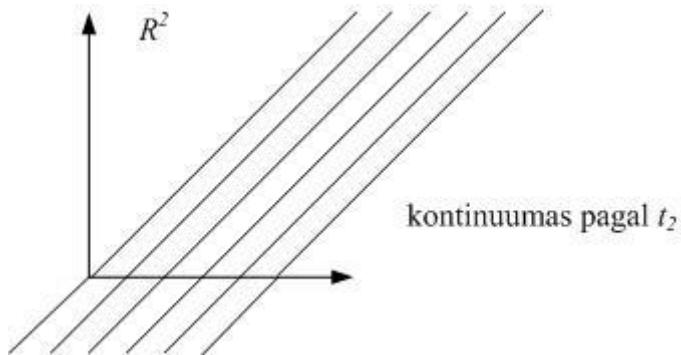


33.10 pav. Faktoraibės R^2/\sim elementas t_2+I yra R^2 poaibis, turintis pavidalą $\{(t_2+t, t), t \in R\}$. Rodoma, kad vektorių x ir y , iš t_2+I skirtumas priklauso I . Tokiu būdu pagal \sim apibrėžimą visi vektoriai iš t_2+I yra ekvivalentiški

Kaip matome, i tiesę galima žiūrėti dvejopai. Pirma, kaip į taškų aibę – R^2 poaibį, antra, faktoraibės R^2/\sim elementą.

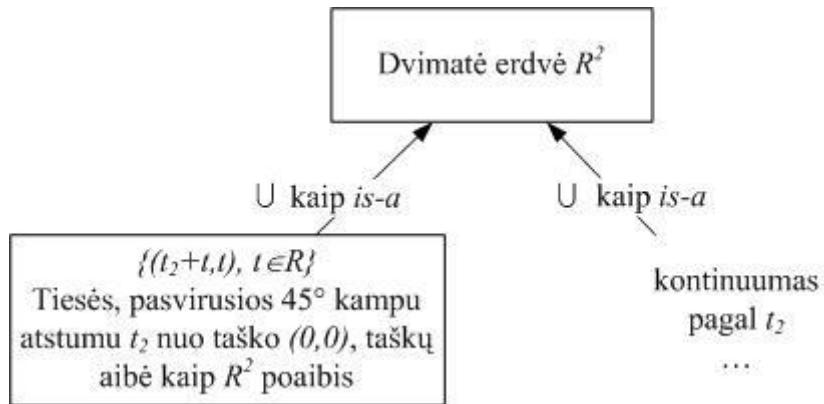
I tiesių rinkinių galima žiūrėti dvejopai. Pirma, kaip į dvimatę Euklidinę erdvę R^2 , antra, kaip į faktoraibę R^2/\sim . Pirmuoju požiūriu, i R^2 žiūrima kaip į sajungą tiesių, pasvirusiu 45° kampu ir nutolusių atstumu t_2 nuo taško (0,0) (žr. 33.11 pav.):

$$R^2 = \bigcup_{t_2 \in (-\infty, +\infty)} \{(t_2+t, t), t \in R\}$$



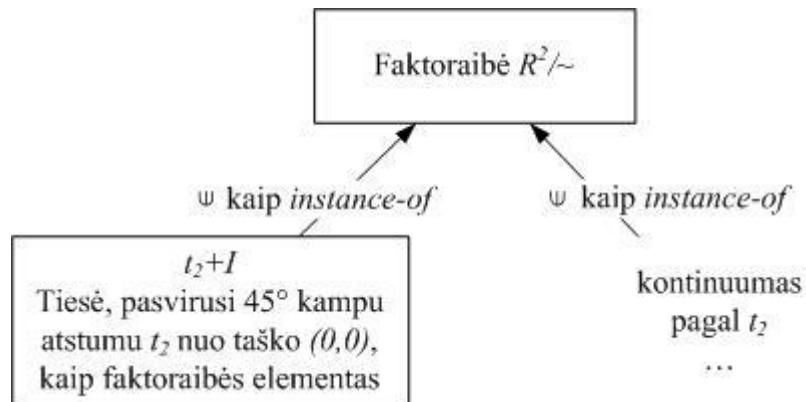
33.11 pav. Dvimatę Euklidinę erdvę R^2 yra sajunga tiesių, pasvirusiu 45° kampu ir nutolusių atstumu t_2 nuo taško (0,0)

Šis (pirmasis) požiūris į dvimatę Euklidinę erdvę R^2 kaip į tiesių sajungą pavaizduotas koncepciniu modeliu, pateiktu 33.12 pav.



33.12 pav. Dvimatės Euklidinės erdvės R^2 koncepcinis modelis. Iš R^2 yra žiūrima kaip į sąjungą tiesių, pasvirusių 45° kampu ir nutolusių atstumu t_2 nuo taško $(0,0)$

Antruoju požiūriu, tiesės yra faktoraibės R^2/\sim elementai kaip aibės elementai, t.y. $t_2+I \in R^2/\sim$ kiekvienam $t_2 \in R$ (žr. 33.13 pav.). Čia iš tiesų žiūrima kaip į tašką – faktoraibės elementą.



33.13 pav. $t_2+I \in R^2/\sim$ kiekvienam $t_2 \in R$. Pastaruoju požiūriu faktoraibės R^2/\sim elementas t_2+I yra atskiras vienetas – tiesė

34. Semantiniai tinklai pagal Russell ir Norvig

Semantiniai tinklai detaliau nagrinėjami vadovaujantis [Russell, Norvig 2003, p. 350-352] 10 skyriumi „Žinių vaizdavimas“ (angl. *”Knowledge representation”*).

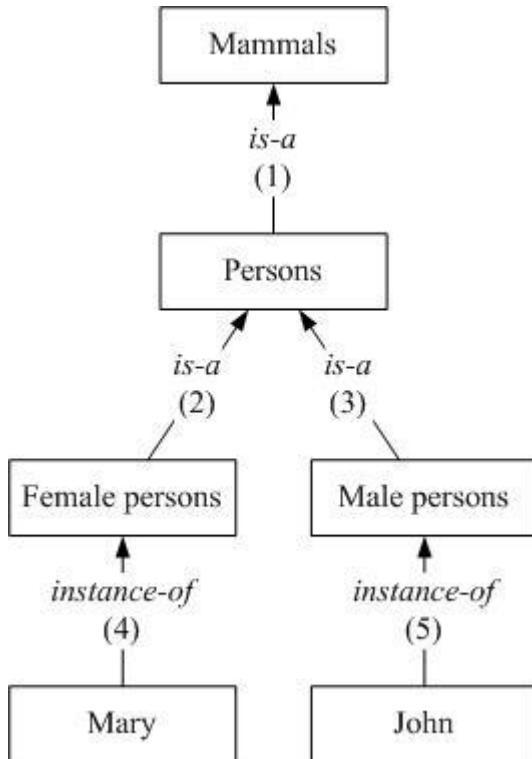
Semantinio tinklo sąvoką pasiūlė 1909 metais amerikiečių matematikas ir mąstytojas Charles Peirce. Nuo to laiko diskutuojama, ar semantiniai tinklai yra tam tikra logikos forma, ar kažkas daugiau. Šio vadovėlio autorius nuomone, semantiniai tinklai nėra vien tik logikos forma. Jeigu taip būtų, tai semantinis tinklas turėtų griežtą semantiką. O realybėje semantinių tinklų stiprybė yra tame, kad jiems nesuteikiama formaliai semantika. Interpretuojantis asmuo gali suteikti tokią prasmę, kokią nori.

Kyla rizika, kad skirtinių asmenys supras skirtingai tą patį pavaizdavimą, bet tokia ir yra kaina. Kartais sąmoningai siekiama suteikti interpretavimo laisvę. Teisėje tai įvardijama kaip atviros prasmės (angl. *open texture*) problema. Teisės mąstytojas Hartas kaip pavyzdį nagrinėja normą „transporto eismas parke draudžiamas“ (žr. [Bench-Capon 2002]). Keliami klausimai, pirma, kas priskiriama transporto priemonėms, antra, kokie yra išimtiniai važiavimo atvejai. Ar riedučiai, žaisliniai automobiliai, dviračiai, žoliaplovės yra transporto priemonės šios normos požiūriu? Ar gali vykti į nelaimės vietą greitoji pagalba ar taksi išskirtinėmis aplinkybėmis?

Jeigu semantiniams grafui būtų suteikiama vienintelė reikšmė, tai būtų nukertama šaka ant kurios sėdima. Filosofiškai žiūrint, kiekvienam reiškinyje galima ižvelgti ir teigiamų ir neigiamų vertinimo kriterijų. Pavyzdžiu, automobilio gebėjimas pasiekti didesnį greitį, gali būti vertinamas teigiamai, bet tam būtinas didesnis kuro suvartojimas, kuris vertinamas neigiamai.

Semantinis tinklas iš [Russell, Norvig 2003, p. 350-352] (žr. 34.1 pav.) užrašomas šiomis matematinės logikos formulėmis:

- (1) $\forall x \text{ Person}(x) \Rightarrow \text{Mammal}(x)$;
- (2) $\forall x \text{ Female_person}(x) \Rightarrow \text{Person}(x)$;
- (3) $\forall x \text{ Male_person}(x) \Rightarrow \text{Person}(x)$;
- (4) $\text{Female_person}(\text{Mary})$. Tai faktas. Predikatas lygus true.
- (5) $\text{Male_person}(\text{John})$. Tai faktas. Predikatas lygus true.



34.1 pav. Semantinis tinklas iš [Russell, Norvig 2003, p. 350-352]

Šiuos teiginius galima užrašyti ir ne tik predikatų, bet aibų ir jų elementų forma (kaip padaryta [Russell, Norvig 2003]), pavyzdžiui:

$$\forall x (x \in \text{Persons} \Rightarrow x \in \text{Mammals})$$

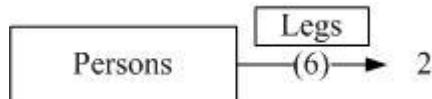
34.1 pav. stačiakampis žymi kategorijos vardą (savoką), o [Russell, Norvig 2003], naudojamos elipsės, pasakant jog tai aibės (intensionalai), t. y. suteikiant tam tikrą semantiką.

Jeigu nagrinėjama dviejų laukų lentelė, tai visi galimų įrašų aibė sudaro intensionalą. Tai laukų apibrėžimo sričių Dekarto sandaugą. O visi realiai duomenų bazėje esančių įrašų aibė sudaro ekstencionalą. Tai intencionalo poaibis.

I grafinę vaizdavimo formą galima iđetи daugiau prasmės negu aprašoma logikos formulėmis. Pavyzdžiui, gali būti skirtingai spalvinama ar paryškinama arba stačiakampiams suteikiamas skirtinges dydis.

34.1. *Paveldėjimas*

Savybė, kad žmogus turi dvi kojas, grafine forma yra pavaizduota 34.2 pav. Čia matyti, kad kategorija skirta ne tik aibei žymeti, bet ir suteikti savybes aibės elementams kaip objektams. Matematikoje aibės elementams nesuteikiame savybės, pavyzdžiui, spalva, kvapas, skonis ir pan. Tuo tarpu informatikoje objektiniame vaizdavime savybės suteikiame.



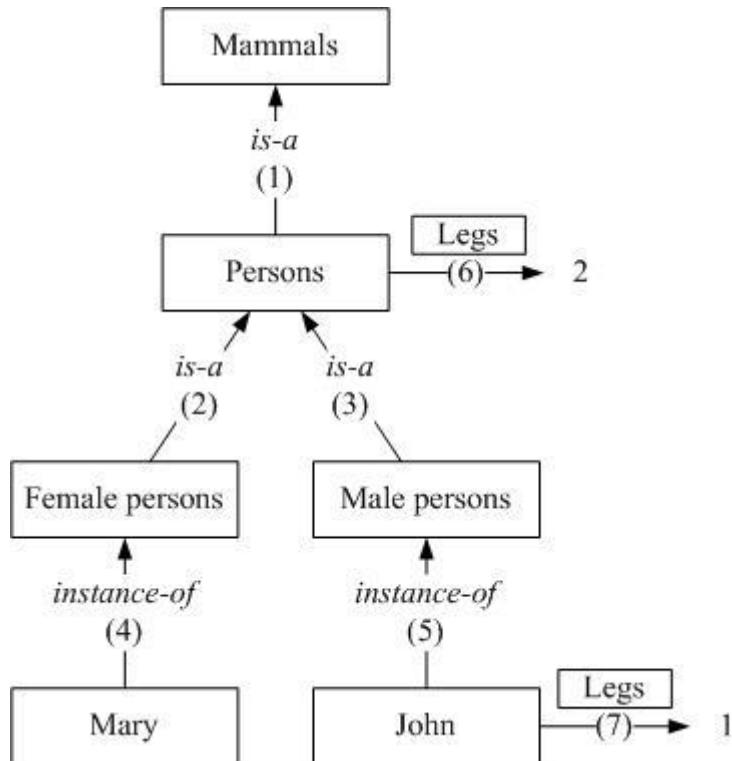
34.2 pav. Asmenys turi dvi kojas. Formaliai $Persons.Legs = 2$. Logikos kalba
 $\forall x \ Person(x) \Rightarrow Legs(x, 2)$

Grafe 34.1 pav. kylant aukštyn ir atsižvelgiant į 34.2 pav. išvedama, kad individuali savoka *John* turės savybę $John.Legs = 2$. Analogiskas teiginys $Legs(John, 2)$ išvedamas iš formulų (1)-(5) ir (6):

(6) $\forall x \ Person(x) \Rightarrow Legs(x, 2)$

Toliau norima pavaizduoti išimtį – *John* turi vieną koją, $John.Legs = 1$ (žr. 34.3 pav.). Turime predikatą:

(7) $Legs(John, 1)$



34.3 pav. *John* turi vieną koją. Formaliai $John.Legs = 1$ arba predikatu $Legs(John, 1)$

Nemonotoninis išvedimas semantiniame tinkle arba žinių bazėje gali sukelti sunkumus. Gali būti išvesti prieštarangi teiginiai $Legs(John, 2)$ ir $Legs(John, 1)$. Todėl yra objektiškai orientuotų kalbų, kuriose draudžiamas multipaveldėjimas.

Kai atsiranda išimtys, pvz., $Legs(John, 1)$, susitariama, kad speciali taisykla nugali bendrają taisyklę. Tačiau nuo tokio tipo susitarimų nukenčia išvedimo efektyvumas. Susitarimas, kad speciali taisykla nugali bendrają taisyklę, taip pat yra taisykla, vadinama *metataisykla*.

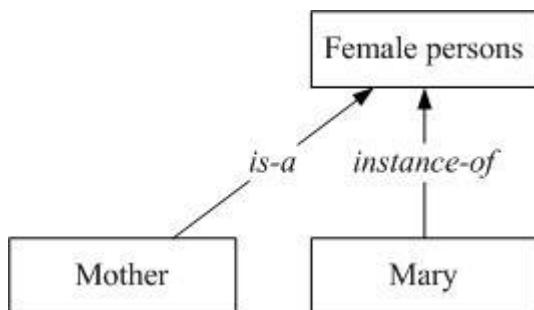
Išimtis apie vienakojį *John* gali būti užrašyta šitaip:

$\forall x \ (x \in Persons \ \& \ x \neq John) \Rightarrow Legs(x, 2)$

Tokio tipo kalba būtų sudėtinga dėl išimčių apdorojimo. Ji būtų nepatogi duomenų bazių valdymo sistemoje.

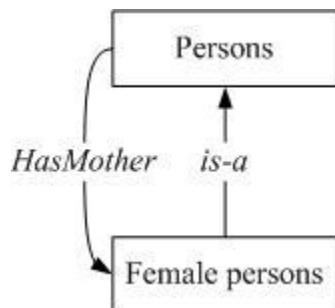
34.2. Santykis ir ryšys

Keliamas klausimas, ar motinos savoką *Mother* galima modeliuoti kaip pavaizduota 34.4 pav.



34.4 pav. Ar šitaip gali būti modeliuojama motinos savoka *Mother*? Ne

Deja, tai neteisingas modeliavimas. Motinos sudaro ne moterų poaibį, kaip modeliuotojui gali atrodyti iš pirmo žvilgsnio, o yra santykis tarp asmens ir jo motinos. Tai santykis tarp dviejų asmenų, kurių antrasis yra moteris. Šis santykis modeliuojamas ryšiu tarp savokų *Persons* ir *Female persons* (žr. 34.5 pav.) arba predikatu *HasMother* (*Person*, *Female_person*).



34.5 pav. Santykis *HasMother* tarp asmens ir jo motinos

Santykis suprantamas kaip lentelė. Žodžiu *HasMother* yra žymimas santykis kai kalbama apie Dekarto sandaugą ir ją atitinkančią lentelę (žr. 34.6 lentelę).

Asmuo	Asmens motina
<i>John</i>	<i>Catherine</i>
...	...

34.6 lentelė. Dvinaris santykis tarp asmens ir jo motinos vaizduojamas dviejų laukų lentele.

Logikos kalba vaizduojama kad *Catherine* yra *John* motina dviem predikatais.

(8) *Female_person* (*Catherine*)

(9) HasMother (John, Catherine)

Lentelėje asmeniui *John* galima priskirti dvi motinas, *Catherine* ir *Rūta* (žr. 34.7 lentelę) kas prieštarauja realybei. Todėl ryšiams gali būti nustatomas reikalavimas kardinalumui: *1:1*, *1:N* ir *N:M*. *HasMother* kardinalumas yra *1:1*.

Asmuo	Asmens motina
<i>John</i>	<i>Catherine</i>
<i>John</i>	<i>Rūta</i>
...	...

34.7 lentelė. Prieštaraujantis realybei santykis, kai asmuo *John* turi dvi motinas – *Catherine* ir *Rūta*

Santykio kryptis aukščiau pasirinkta, atsižvelgiant į lentelės lauką, kuris yra raktas.

Ryšio *HasMother* intensionalas yra *Persons × Female persons*. Kad asmens motina yra moteris galima užrašyti logikos kalba:

(10) $\forall x \ x \in \text{Persons} \Rightarrow (\forall y \ \text{HasMother}(x, y) \Rightarrow y \in \text{Female_persons})$

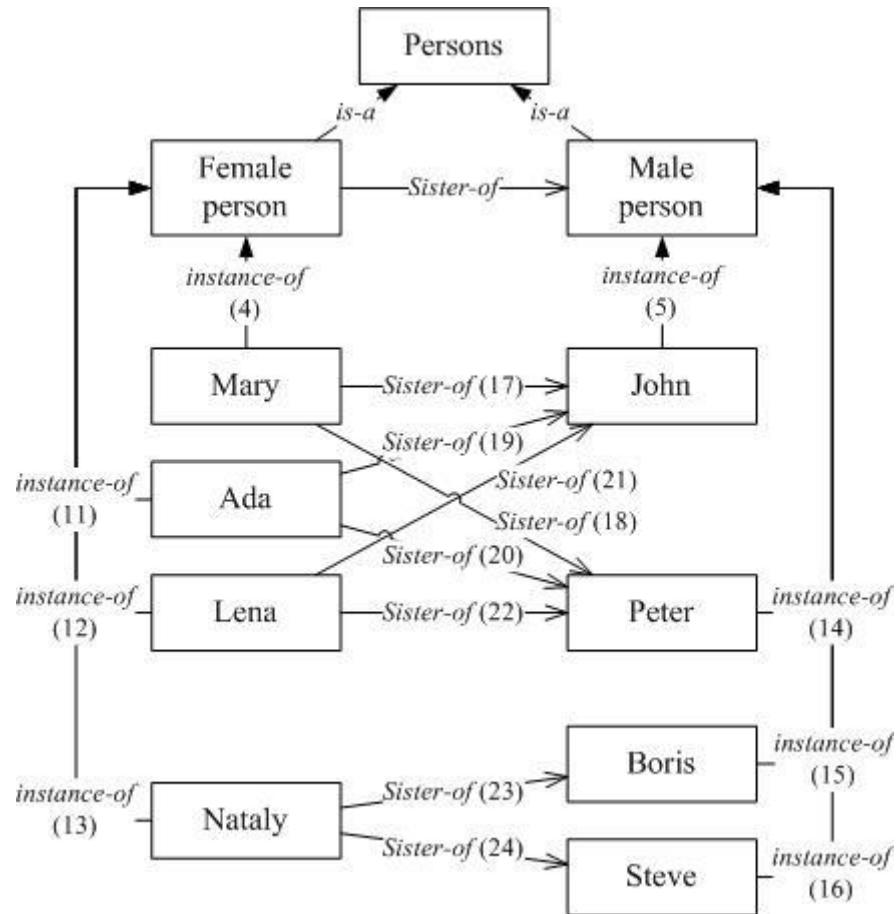
Logikos kalba gali būti užrašomas apribojimas ryšio kardinalumui:

„Asmens gimimo įraše gali būti nurodyta tik viena motina arba nė vienos“

Pastaruoju teiginiu vaizduojamos žinios, kad vaiko gimimo įraše gali nebūti duomenų apie motiną. Taip civilinėje metrikacijoje sprendžiamas vaikų, kurių biologinė motina nėra žinoma, lauko „Motina“ užpildymas. Šis laukas gali būti paliekamas tuščias arba rodyti į moterį, kuri nėra biologinė motina.

34.3. Atvirkštinis santykis

Toliau įvedamas dar vienas ryšys, būtent, *Sister-of*, tarp kategorijų *Female_person* ir *Male_person*. Be to semantinis tinklas papildomas faktais, kad turimos trys seserys *Mary*, *Ada* ir *Lena* bei jų du broliai *John* ir *Peter*. Taip pat pridedama viena moteris *Nataly* ir du jos broliai *Boris* ir *Steve* (žr. 34.8 pav.).



34.8 pav. Trys seserys *Mary*, *Ada* ir *Lena* turi du brolius *John* ir *Peter*. *Nataly* turi du brolius *Boris* ir *Steve*

Šie faktai vaizduojami predikatais:

- (11) Female_person(Ada)
- (12) Female_person(Lena)
- (13) Female_person(Nataly)
- (14) Male_person(Peter)
- (15) Male_person(Boris)
- (16) Male_person(Steve)
- (17) Sister-of(Mary, John)
- (18) Sister-of(Mary, Peter)
- (19) Sister-of(Ada, John)
- (20) Sister-of(Ada, Peter)
- (21) Sister-of(Lena, John)
- (22) Sister-of(Lena, Peter)
- (23) Sister-of(Nataly, Boris)
- (24) Sister-of(Nataly, Steve)

Ryšys *Sister-of* apibrėžtas tarp kategorijų *Female_person* ir *Male_person*. Tokiu būdu šiame semantiniame tinkle nėra vaizduojama situacija, kai dvi moterys yra seserys.

Santykis *Sister-of* nustatytas 34.9 lentele.

Female_person	Male_person
Mary	John
Mary	Peter
Ada	John
Ada	Peter
Lena	John
Lena	Peter
Nataly	Boris
Nataly	Steve

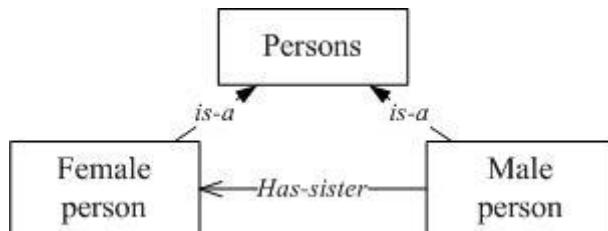
34.9 lentelė. Trys seserys *Mary*, *Ada* ir *Lena* turi du brolius *John* ir *Peter*.
Nataly turi du brolius *Boris* ir *Steve*

Kita išvedimo forma yra naudojimasis atvirkštiniais ryšiais. Įvedamas ryšys *Has-sister* tarp sąvokų *Male_person* ir *Female_person*. Šis ryšys apibrėžiamas predikatu, teigiančiu, kad *Has-sister* yra atvirkštinis *Sister-of* (žr. 34.10 pav.):

$$(25) \forall f \forall m \text{ Has-sister}(m, f) \Leftrightarrow \text{Sister-of}(f, m)$$

Pastarasis teiginys (25) reifikuojasi ryši *Has-sister*. Nuo predikato *Has-sister* pradedama žiūrėti į šį ryšį kaip į objektą (žr. 34.10 pav.).

34.1 Apibrėžimas. Reifikavimas – tai predikato arba funkcijos pavertimas algoritminės kalbos objektu [Russell, Norvig 2003, p 230].



34.10 pav. Ryšys *Has-sister* tarp sąvokų *Male_person* ir *Female_person* yra gaunamas reifikuojant predikatą *Has-sister* kaip atvirkštinį *Sister-of*

Kaip matome, nuo charakterizavimo predikatų terminais, galima pereiti prie naujo objekto, kuris yra atskira sąvoka. Sąvokos įvedimas yra būdingas žmogiškajam intelektui.

Tegu duomenų bazei pateikiama užklausa „Kas yra *John* seserys?“:
Sister-of(?, John)

Tai užklausa į Prolog panašia kalba. Sistemos prašoma rasti visas individualias sąvokas, kurios yra santykje *Sister-of* su *John*.

Šios užklausos sudėtingumas yra tiesiškai priklausomas nuo lentelės *Sister-of* ilgio *N*. Perrenkami visi lentelės įrašai iš eilės ir ieškoma tokį įrašą, kurių stulpelio *Male_person* reikšmė yra *John*. Randami trys tokie įrašai.

Ar galimas geresnis sudėtingumas? Klausimas formuluoamas, koks yra užklausos *Has-sister(John, ?)* sudėtingumas? Atsakymas: tiesinis pagal lentelės *Sister-of* ilgį *N*.

Irodymas. Iš (25) gaunama:

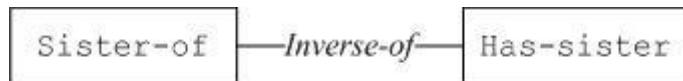
$$\text{Has-sister}(\text{John}, ?) \Leftrightarrow \text{Sister-of}(?, \text{John})$$

Įrodymo pabaiga.

Tuo tarpu kitos užklausos Sister-of(Ada, ?) sudėtingumas yra logaritminis, t.y $O(\ln(N))$. Tai pasiekiamas, predikatą atitinkančios reliacinės lentelės rūšiavimu pagal pirmajį parametru (t. y. Female_person).

Kaip gauti Has-sister(John, ?) logaritminį sudėtingumą? Tam būtina sistemai iš anksto pateikti euristinę informaciją apie dalykinę sritį, kad santykis Has-sister yra atvirkštinis santykiui Sister-of (žr. 34.11 pav.):

(26) Inverse-of(Has-sister, Sister-of).



34.11 pav. Teiginys Inverse-of(Has-sister, Sister-of), kad santykis Has-sister yra atvirkštinis santykiui Sister-of suteikia galimybę gauti užklausos Has-sister(John, ?) logaritminį, o ne tiesinį sudėtingumą

Predikate (26) Inverse-of yra transliuojamas į formulę (25), kuri formaliai pasako, ką reiškia atvirkštinis. Tokia euristinė informacija savo esme yra ryšys tarp santykų Has-sister ir Sister-of kaip sąvoką. Predikatas (26) yra „sintaksinis cukrus“ (angl. syntax sugar) atžvilgiu materialaus teiginio (25).

Transliuojant predikatą Inverse-of loginio išvedimo programa papildoma lentele (žr 34.12 lentelę), kuri yra atvirkštinė 34.9 lentelei.

Male_person	Female_person
John	Mary
John	Ada
John	Lena
Peter	Mary
Peter	Ada
Peter	Lena
Boris	Nataly
Steve	Nataly

34.12 lentelė. Santykis Has-sister. Atkreipiama dėmesys kad ši lentelė skiriasi nuo 34.9 lentelės ne tik stulpelių apsukimu, bet ir kita įrašų tvarka, nes raktu tapo Male_person

Kaip matome, 34.12 lentelė skiriasi nuo 34.9 lentelės tik stulpelių apsukimu. Nauju raktu čia tapo Male_person. Ši apvertimo procedūra (25) yra brangi $O(N \cdot \ln(N))$. Tačiau vieną kartą atlikus brangią procedūrą, toliau užklausos Has-sister(John, ?) sudėtingumas pagerinamas nuo tiesinio $O(N)$ iki logaritminio $O(\ln(N))$.

Reziumuojama, kad išvedimo naudojantis atvirkštiniu ryšiu privalumai išryškėja kai:

1. ryšys yra reifikuotas, t.y. paverstas objektu, kaip (25);
2. ryšiui priskirta tam tikra procedūra (pvz. santykio lentelės apsukimas), t.y. atliktas procedūrinis priskyrimas.

Jeigu parašoma ši brangi procedūra, kuri realizuoja apsukimą, tai vėliau turima galimybė predikatu *Inverse-of* deklaruoti atvirkštinius santykius.

34.2 Apibrėžimas. Procedūrinis priskyrimas (angl. *procedural attachment*) – tai procedūra priskiriama ryšiui ir naudojama atsakymui į užklausą arba teiginio išvedimui, kuris yra efektyvesnis negu bendro loginio išvedimo algoritmo atveju.

Procedūrinis priskyrimas ir yra bruožas, skiriantis semantinį tinklą nuo logikos (čia turima galvoje teiginių logika arba pirmos eilės predikatų logika).

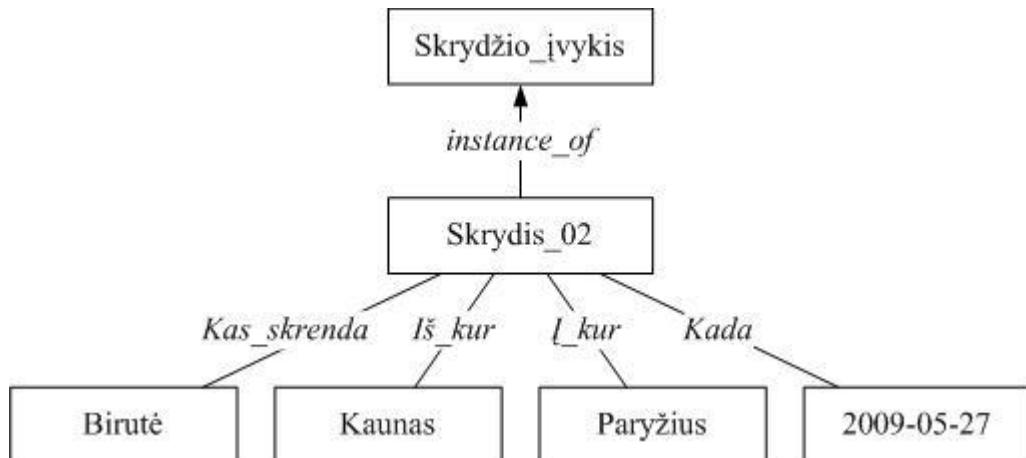
Reifikavimas yra deklaravaus vaizdavimo forma. Procedūrinis priskyrimas yra procedūrinio vaizdavimo forma. Kaip minėta anksčiau, procedūrinio vaizdavimo forma paprastai nepriskiriama žinių vaizdavimui.

Dar vienas pavyzdys demonstruoja santykio vaizdavimą semantiniu tinklu. Imamas santykis

Skrydis(Kas_skrenda, Iš_kur, Į_kur, Kada)

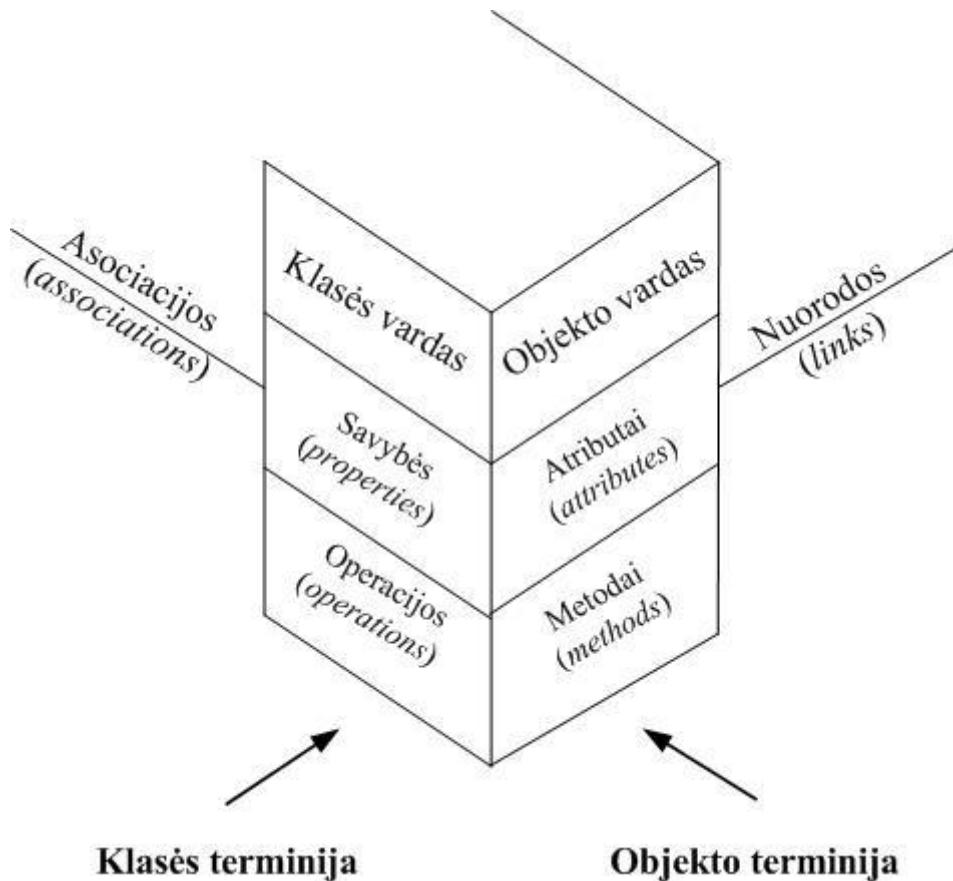
Teiginys

Skrydis(Birutė, Kaunas, Paryžius, 2009-05-27)
yra vaizduojamas semantiniu tinklu (žr. 34.13 pav.).



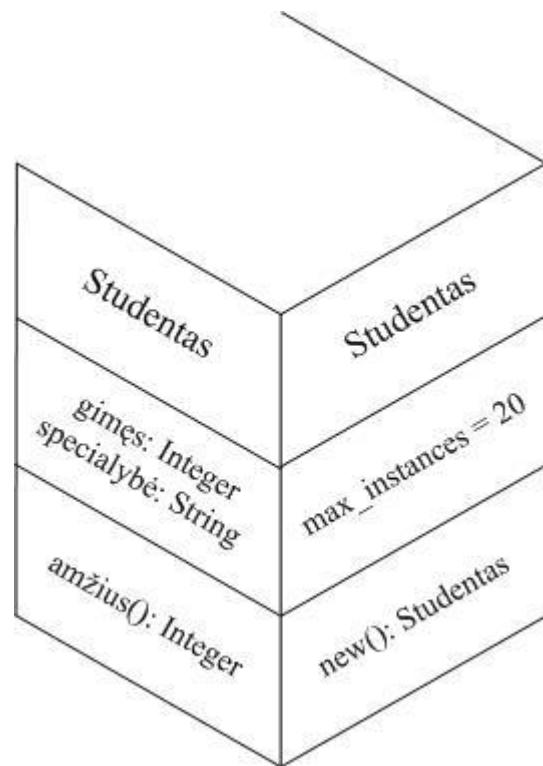
34.13 pav. Teiginio Skrydis(Birutė, Kaunas, Paryžius, 2009-05-27) vaizdavimas semantiniu tinklu

Semantinių tinklų terminologija susijusi su objektinio programavimo terminais. Apjungiantis klasį ir objektų terminologiją požiūris į klases ir objektus yra pateiktas 34.14 pav. Toks požiūris į klasę-objektą yra vadinamas klabjektu (angl. *clabject, class and object*) [Atkinson 1997]. Pastarasis yra suprantamas kaip metamodelis.



34.14 pav. Klabjektas (angl. *clabject, class and object*) – tai objektinio programavimo požiūris į klasses ir objektus. Pastarasis yra suprantamas kaip metamodelis

Klabjekto pavyzdys pateikiamas 34.15 pav. Klabjektas turi savybes, operacijas, asociacijas, atributus, metodus ir nuorodas.

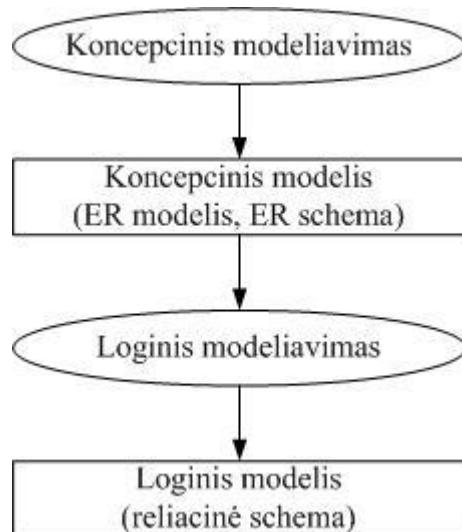


34.15 pav. Klabjektas Studentas

35. Koncepciniai modeliai duomenų bazėse

Šis skyrius dėstomas remiantis Romo Barono 2002 m. išleistos mokymo priemonės ir atnaujinto vadovėlio [Baronas 2005] 4 skyriumi „Semantinis modeliavimas“. Pastarajame yra aptariamas duomenų bazių (DB) esybių-ryšių (ER) modelis (angl. *entity-relationship model*), kurį 1976 m. pasiūlė Piteris Pin-Šan Čenas (*Peter Pin-Shan Chen*).

Pirmasis duomenų bazės projektavimo etapas yra koncepcinis (arba sampratos) modeliavimas. Jo rezultatas yra koncepcinis modelis (ER modelis, ER schema). Kitas etapas yra loginis modeliavimas. Pastarojo etapo rezultatas yra loginis modelis (reliacinė schema) (žr. 35.1 pav.).



35.1 pav. Duomenų bazės projektavimo etapai

Koncepcinis modelis kuriamas analizės stadijos metu.

Toliau imamas duomenų bazės pavyzdys iš [Baronas 2005] 2 skyriaus.

Duomenų bazę *Darbai* sudaro trys lentelės: *Vykdytojai*, *Projektai*, *Vykdymas*. Šios lentelės užpildytos tokiais duomenimis:

Nr	Pavardė	Kvalifikacija	Kategorija	Išsilavinimas
1	Jonaitis	Informatikas	2	VU
2	Petraitis	Statistika	3	VU
3	Gražulytė	Inžinierius	1	NULL
4	Onaitytė	Vadybininkas	5	VDU
5	Antanaitis	Informatikas	3	VU

35.2 lentelė. DB *Darbai* lentelė *Vydytojai*

Nr	Pavadinimas	Svarba	Pradžia	Trukmė
1	Studentų apskaita	Maža	2001.01.01	12
2	Buhalterinė apskaita	Vidutinė	2001.03.01	10
3	WWW svetainė	Didelė	2001.06.01	2

35.3 lentelė. DB *Darbai* lentelė *Projektai*

Projeketas	Vykdytojas	Statusas	Trukmė
1	1	Programuotojas	30
1	2	Dokumentuotojas	100
1	3	Testuotojas	100
1	4	Vadovas	100
2	1	Programuotojas	300
2	2	Analitikas	250
2	4	Vadovas	100
3	1	Programuotojas	250
3	2	Vadovas	400
3	3	Dizaineris	150

35.4 lentelė. DB *Darbai* lentelė *Vykdymas*

Lentelės eilutės suprantamos kaip santykio kortežai. Lentele užduodamas santykis yra Dekarto sandaugos poaibis, pavyzdžiu:

$$\text{Vykdytojai} \subset \text{integer} \times \text{string} \times \text{string} \times \text{integer} \times \text{string}$$

Čia lentelėje *Vykdytojai* stulpelis *Nr* yra *integer* tipo, *Pavardė* – *string*, *Kvalifikacija* – *string*, *Kategorija* – *integer*, *Išsilavinimas* – *string*. Stulpelių pavadinimai, pvz., *Pavardė*, *Kvalifikacija*, *Kategorija*, *Išsilavinimas* atitinka esybės iš ER modelio atributus. Dekarto sandaugos elementas yra suprantamas ir vaizduojamas kaip kortežas (e_1, e_2, e_3, e_4, e_5), kur e_i yra i -tojo atributo reikšmė, pavyzdžiu:

$$(1, 'Jonaitis', 'Informatikas', 2, 'VU') \in \text{Vykdytojai}$$

Analogiškai:

$$\text{Vykdytojai} \subset \text{Nr} \times \text{Pavardė} \times \text{Kvalifikacija} \times \text{Kategorija} \times \text{Išsilavinimas}$$

Čia $\text{Nr} \subset \text{integer}$, *Kvalifikacija* $\subset \text{string}$, *Pavardė* $\subset \text{string}$, *Kategorija* $\subset \text{integer}$, *Išsilavinimas* $\subset \text{string}$.

Žemiau pateikiama užklausa SQL kalba, kuria išrenkami lentelės *Vykdytojai* tos atributų *Pavardė* ir *Kategorija* reikšmes, kurios priklauso kortežams, turintiems atributo *Kvalifikacija* reikšmę '*Informatikas*':

SELECT Pavardė, Kategorija FROM Vykdytojai WHERE Kvalifikacija = 'Informatikas'

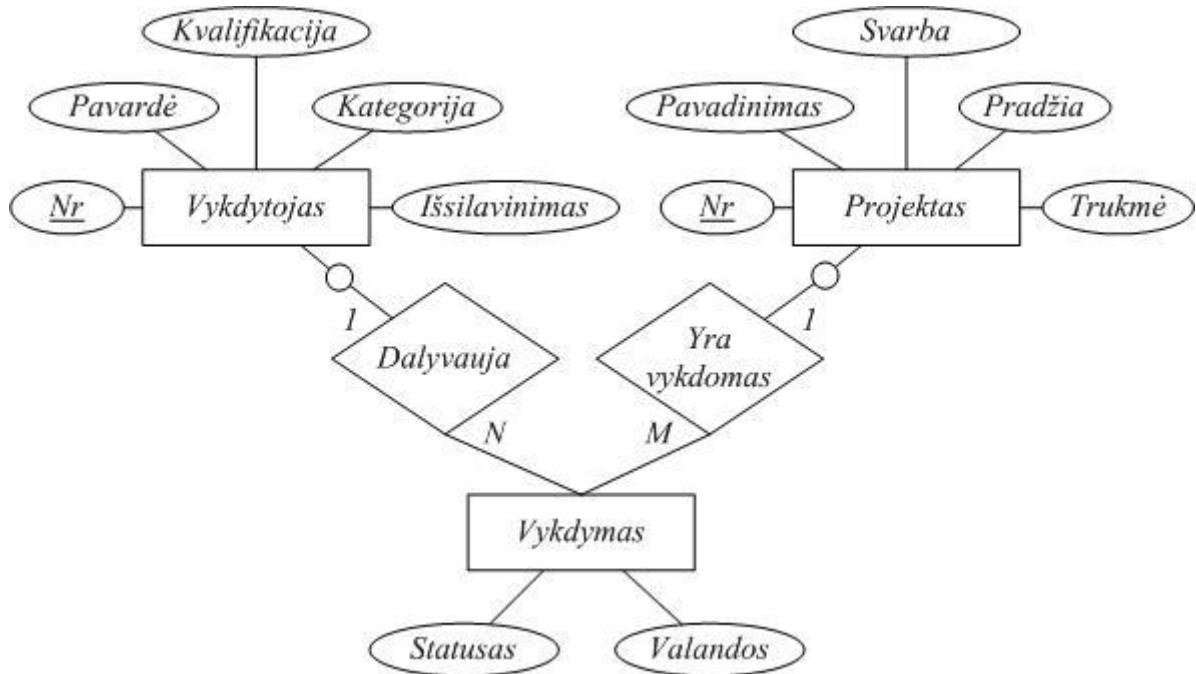
Šios užklausos vykdymo rezultatas yra 35.5 lentelė.

Pavardė	Kategorija
Jonaitis	2
Antanaitis	3

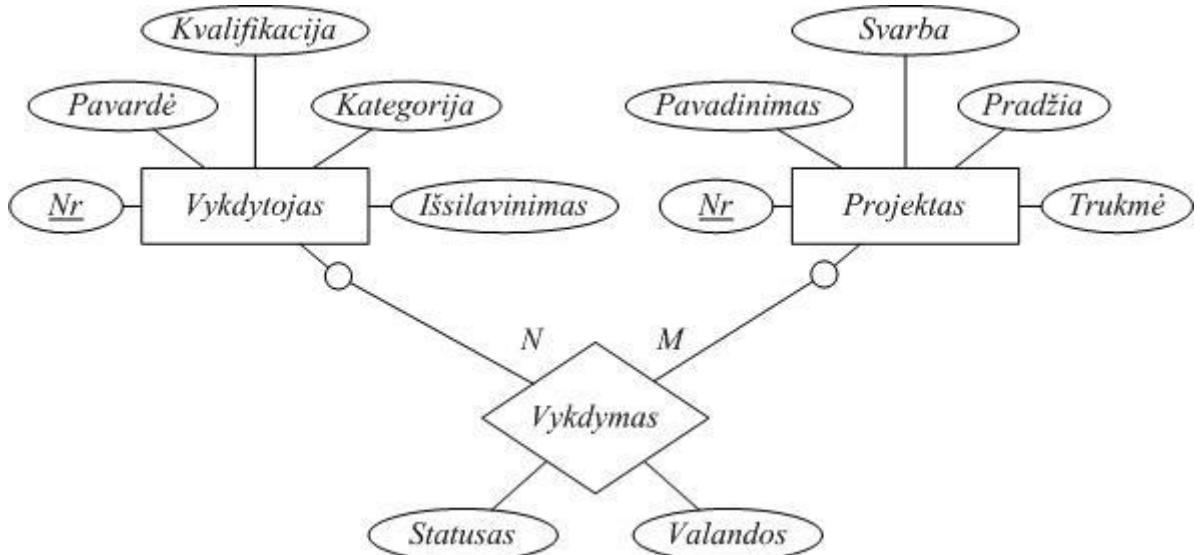
35.5 lentelė. Užklausos rezultatas yra lentelė – įmonės informatikai

Nagrinėjama įmonės veikla, kai darbuotojai vykdo projektus, kiekvienas turi tam tikrą vaidmenį ir skiria projektui tam tikrą laiką. Išskiriama dvi esybės: *Vykdytojas* ir *Projektas*. Kiekvienos iš šių esybių atributų rinkinys priklauso nuo įmonės veiklos parametru. Tegu apsiribojama lentelėse pateiktais *Vykdytojo* atributais *Nr*, *Pavardė*, *Kvalifikacija*, *Kategorija*, *Išsilavinimas* ir *Projekto* atributais *Nr*, *Pavadinimas*, *Svarba*, *Pradžia*, *Trukmė*. Čia atributo pabraukimas žymi kad atributas įeina į lentelės pirminį raktą. Darbuotojų veikla gali būti modeliuojama keliais būdais [Baronas 2005, 4.4 poskyris, p. 78 ir 2002 m. leidimo, 4.5 poskyris].

1. I vykdytojų veiklą projekte galima žiūrėti kaip į atskirą esybę *Vykdymas* su atributais *Statusas* ir *Valandos*, susiejant šią esybę tiek su esybe *Vykdytojas*, tiek ir su *Projektas*. Kadangi vienas vykdytojas gali dalyvauti keliuose projektuose, tai tarp esybių *Vykdytojas* ir *Vykdymas* egzistuoja *1:N* tipo ryšys *Dalyvauja*. Tarus, kad vykdytojas gali nedalyvauti nė viename projekte, *Vykdytojas* yra nebūtinas šio ryšio dalyvis. Pavyzdžiui, toks vykdytojas yra Antanaitis, nes jis nedalyvauja nei viename projekte. Panašiai tarp esybių *Projektas* ir *Vykdymas* taip pat yra *1:N* ryšys ir jis yra neprivalomas iš esybių *Projektas* pusės. Taip sudaryta ER schema pateikta 35.6 pav.
2. I vykdytojų dalyvavimą projektuose galima žiūrėti kaip į *N:M* tipo ryši *Vykdymas* tarp esybių *Vykdytojas* ir *Projektas*. Abu šio ryšio dalyviai yra neprivalomi. Šis ryšys turi du atributus – *Statusas* ir *Valandos*. Taip sudaryta ER schema pateikta 35.7 pav.



35.6 pav. Duomenų bazės *Darbai* ER schema, kai *Vykdymas* modeliuojamas kaip esybė



35.7 pav. Duomenų bazės *Darbai* ER schema, kai *Vykdymas* modeliuojamas kaip ryšys *N:M*

Toliau aptariami ER schemas elementai.

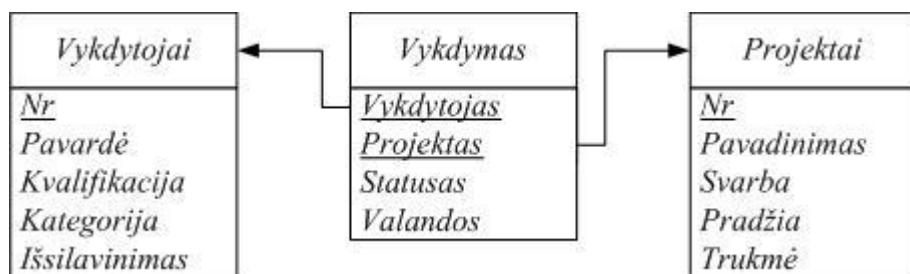
Esybės. Kiekviena esybė vaizduojama atskiru stačiakampiu. Jo viduje rašomas esybės vardu.

Atributai. Kiekvienas atributas ER schemae vaizduojamas atskiru ovalu. Su atitinkamos esybės stačiakampiu atributas yra sujungiamas ištisine linija. Ovalo viduje rašomas atributo vardu. Sudėtinio atributo sudedamosios dalys vaizduojamos ovalais, sujungtais ištisine linija su sudėtinio atributo ovalu. Raktinių atributų vardai pabraukiami. Daugiaareikšmiai atributai jungiami su esybe dviguba linija.

Ryšiai. Kiekvienas ryšys vaizduojamas atskiru rombu su ryšio vardu viduje. Rombas apvedamas dviguba linija, jeigu ryšys yra tarp silpnosios esybės ir pagrindinės esybės. Ryši vaizduojantis rombas sujungiamas ištisinėmis linijomis su visais ryšio dalyviais. Kiekviena tokia linija pažymima „1“, „N“ ar „M“ ryšio tipui pažymeti. Linija, jungianti ryšio rombą su silpnaja esybe, yra dviguba.

Potipiai ir virštipiai. Jeigu esybės X_1, X_2, \dots, X_n yra esybės Y potipiai, tai iš kiekvienos esybės X_i ($i=1, \dots, n$) stačiakampio brėžiama linija į grafinį elementą, vadinamą deskriptoriumi (jį vaizduosime trikampiu), o iš jo į esybės Y stačiakampį su rodykle linijos gale.

Duomenų bazės *Darbai* reliacinė schema, gauta iš ER schemas pateikiama 34.8 pav.



35.8 pav. Duomenų bazės *Darbai* reliacinė schema, gauta iš ER schemas 34.6 pav.

Iš ER schemas esybių *Vykdytojas* ir *Projektas* (žr. 35.6 pav.) reliacinėje schemaje 34.8 pav. gaunamos lentelės *Vykdytojai* ir *Projektai* (daugiskaita):

Vykdytojai(Nr, Pavardė, Kvalifikacija, Kategorija, Išsilavinimas)
Projektai(Nr, Pavadinimas, Svarba, Pradžia, Trukmė)

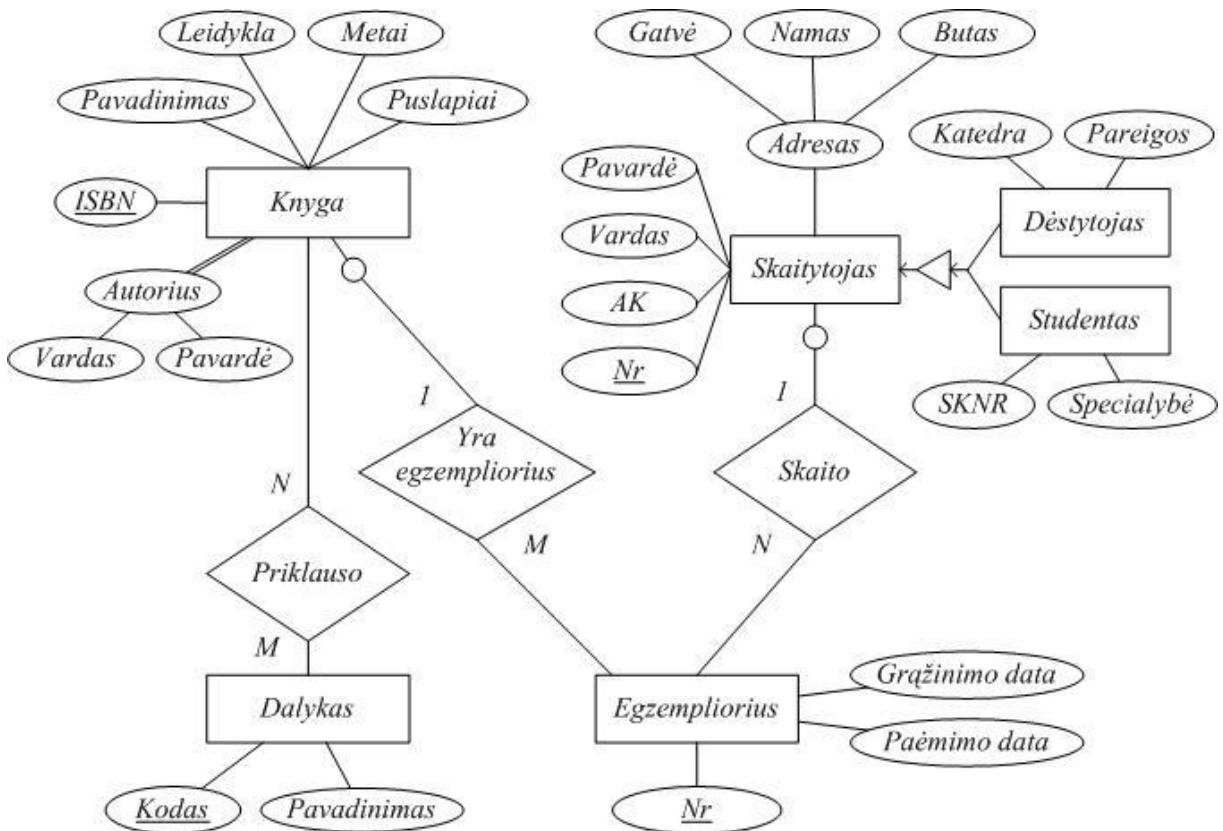
Abiejų ER schemų atvejais, t.y. ir esybei *Vykdymas* (žr. 35.6 pav.), ir N:M tipo ryšiui *Vykdymas* (žr. 34.7 pav.), reliacinėje schemaje (žr. 35.8 pav.) sudaroma atskira lentelė *Vykdymas*. Į šią lentelę įtraukiamais ryšio atributai, ir jo dalyvių raktai, kurie tampa lentelės išoriniai raktai.

Vykdymas(Projektas, Vykdytojas, Statusas, Valandos)

Čia yra du išoriniai raktai: *Projektas* nukreipia į *Projektai*, *Vykdytojas* nukreipia į *Vykdytojai*.

ER schema naudojama ankstyvosiose projektavimo stadijose (analizėje) o reliacinė schema – vėlesnėse (detaliajame projektavime). ER schemas transformavimo į reliacinių modelių taisyklės pateikiamos [ibid., p. 74-77].

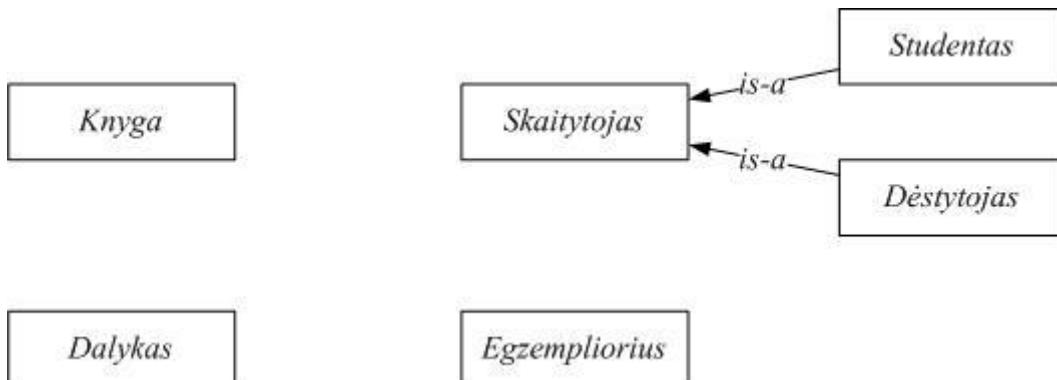
Toliau pateikiamas kitas pavyzdys. Tai duomenų bazės *Biblioteka* ER schema iš [ibid., p. 74] (žr. 35.9 pav.).



35.9 pav. Duomenų bazės *Biblioteka* ER schema iš [Baronas 2005, p. 74]

Skirtinguose lygmenyse (konceptinio modelio, ER schemas ir reliacinės schemas) terminologija skiriasi. Konceptinis modelis yra analizės lygmenyje, o reliacinė schema – projektavimo lygmenyje.

Toliau demonstruojama, kaip kuriamas bibliotekos konceptinis modelis. Konceptiniame modelyje iš pradžių išskiriamos keturios esybės: *Knyga*, *Skaitytojas*, *Dalykas* ir *Egzempliorius*. Toliau *Skaitytojas* skaidomas į dar dvi esybes: *Dėstytojas* ir *Studentas*. Tokiu būdu tarp pastarųjų ir *Skaitytojas* yra *is-a* ryšys (žr. 35.10 pav.).



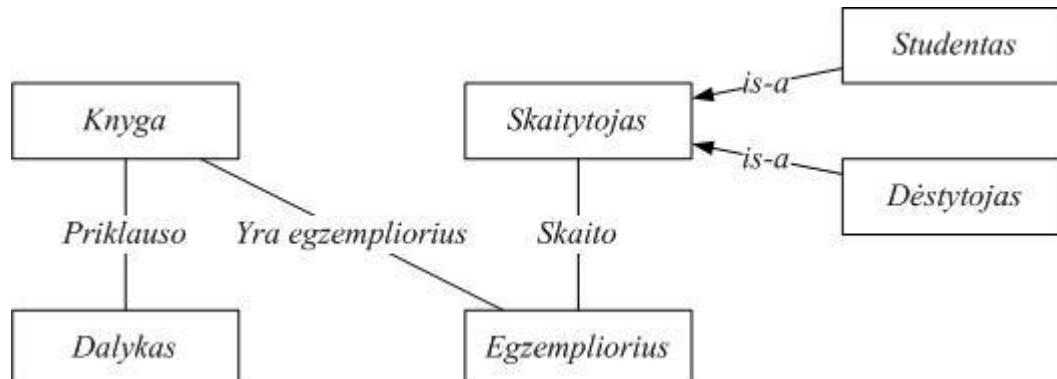
35.10 pav. *is-a* ryšys tarp esybių *Skaitytojas*, *Studentas* ir *Dėstytojas*

ER schemas lygmenyje yra keturios pagrindinės esybės ir du potipiai:

$$\begin{aligned} \text{Pagrindinės_esybės} &= \{\text{Knyga}, \text{Dalykas}, \text{Egzempliorius}, \text{Skaitytojas}\} \\ \text{Potipiai} &= \{\text{Dėstytojas}, \text{Studentas}\} \end{aligned}$$

Ir pagrindinė esybė, ir potipis yra esybės. Taigi į skaitytoją žiūrima kaip į aibę, o į studentą ir dėstytoją, kaip į poaibius. Tačiau kalbant apie konceptinį modelį, nesakoma, ar tai aibė, ar kas kita, o tiesiog kalbama apie sąvokos vardą. Vėliau tikslinama, kur yra aibė arba predikatas.

Toliau pridedamos trys briaunos, žymintos ryšius: *Skaito*, *Yra egzempliorius* ir *Priklauso*. Primenama, kad ryšys bus transformuojamas į lentelę. O lentelė yra žemesnio lygmens – reliacinės schemas – sąvoka.



35.11 pav. Konceptinis modelis, kai 34.10 pav. papildomas trimis ryšiais

Toliau belieka pridėti atributus bei ryšių kardinalumus ir prieiti prie ER schemas 35.9 pav.

36. Žinių vizualizavimas

Kaip buvo minėta žinių vaizdavimas gali būti aptariamas žinių vadybos (angl. *knowledge management*) kaip savarankiškos disciplinos kontekste. Žinių vadybos kontekste gali būti aptariamas ir *žinių vizualizavimas* (angl. *knowledge visualization*).

Žinių vizualizavimas (angl. *knowledge visualization*) skiriasi nuo žinių vaizdavimo (angl. *knowledge representation*). Žinių vizualizavimas skiriamais žmogui, kitais žodžiais žinių vizualizavimo rezultatas skirtas žmogui. Tuo tarpu žinių vaizdavimo rezultatas skirtas kompiuteriui. Žinių valdymo enciklopedijoje yra rašoma ir apie žinių vizualizavimą [Eppler, Burkhard 2006], ir apie žinių vaizdavimą [Zarri 2006].

Toliau vadovaujamasi [Eppler, Burkhard 2006]. Žinių vizualizavimo objektas yra vizualiniai pavaizdavimai (angl. *visual representation*) skirti žinių *kūrimui* (angl. *creation*) ir *perdavimui* (angl. *transfer*) tarp mažiausiai dviejų žmonių. Kaip matyti iš apibrėžimo, žinių vizualizavimo subjektas yra žmogus. Čia yra skirtumas nuo žinių vaizdavimo subjekto – kompiuterio.

Vizualaus supratimo (angl. *visual cognition and perception*) studijose daroma išvada, kad vizualizavimas, ženkliai padidina žmogaus gebėjimus mąstyti ir komunikuoti.

Žinių vizualizavimo schemą (angl. *framework*) sudaro trys perspektyvos (žr 36.1 lentelę).

Žinių tipas (kas?)	Vizualizavimo tikslas, paskirtis (kodėl?)	Vizualizavimo formatas (kaip?)
Žinoti-ką	Perdavimas (išaiškinimas, išgavimas, socializacija)	Euristiniai eskizai
Žinoti-kaip	Kūrimas (atradimas, kombinavimas)	Koncepcinės diagramos
Žinoti-kodėl	Mokymasis (gavimas, internalizavimas)	Vizualios metaforos
Žinoti-kur	Kodifikavimas (dokumentavimas, eksternalizavimas)	Žinių animacijos
Žinoti-kas	Radimas (pvz. ekspertų, dokumentų, grupių)	Žinių žemėlapiai
	Vertinimas (ivertinimas, reitingavimas)	Sričių struktūros

36.1 lentelė. Trys žinių vaizdavimo schemas perspektyvos

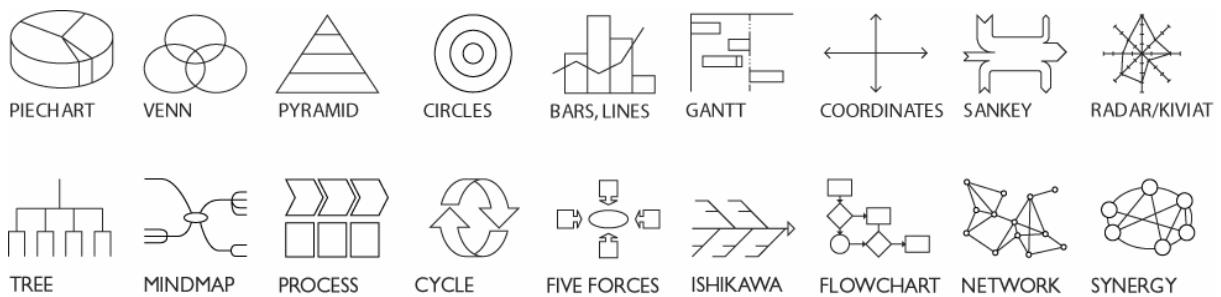
Šios perspektyvos atsako į tris kertinius klausimus apie vizualizuojamas žinias:

1. Kokios žinios yra vizualizuojamos (objektas)?
2. Kodėl žinios yra vizualizuojamos (tikslas, paskirtis)?
3. Kaip žinios yra vaizduojamos (metodas)?

Žinių vaizdavimo kontekste svarbi trečioji perspektyva. Verta atskirai aptarti kiekvienu vizualizavimo formatą. Dažniausiai naudojamų koncepcinių diagramų tipai, pagal [Eppler, Burkhard 2006] (žr. 36.2 pav.):

1. pyrago diagrama (angl. *pie chart*);
2. Veno (angl. *Venn*) diagrama;
3. piramidė;

4. skrituliai;
5. stulpeliai, kreivės;
6. Ganto (angl. *Gantt*) diagrama;
7. koordinačių ašys;
8. Sankei (angl. *Sankey*) diagrama. Pagal Airių kapitoną *Matthew H. P. R Sankey*, panaudojusį 1898;
9. radaras;
10. medis;
11. minčių žemėlapis;
12. procesas;
13. ciklas;
14. penkios jėgos;
15. Išikava (angl. *Ishikawa*) diagrama;
16. struktūrinė schema;
17. tinklas;
18. sinergija;



36.2 pav. Dažniausiai naudojamų koncepcinių diagramų tipai: pyrago diagrama, Veno diagrama, piramidė, skrituliai, stulpeliai-kreivės, Ganto diagrama, koordinačių ašys, Sankei diagrama, radaras, medis, minčių žemėlapis, procesas, ciklas, penkios jėgos, Išikava diagrama, struktūrinė schema, tinklas ir sinergija [Eppler, Burkhard 2006, 554 p]

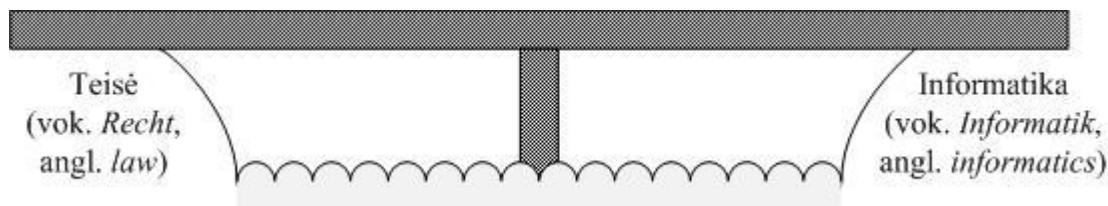
Žinių vizualizavimas svarbus žinių perdavimui. Žinios perduodamos įvairiais lygmenimis: tarp individų, grupių ir organizacijų. Žinių perdavimui svarbu, kad žinios turi būti atkuriamas priėmėjo sąmonėje.

37. Teisės informatika

Teisės informatika (angl. *legal informatics*, vok. *Rechtsinformatik*) suprantama kaip informatikos taikymas teisės dalykinėje srityje. Tai tarpšakinė disciplina informatikos ir teisės sankirtoje. Skaitomame kurse į teisės informatiką žiūrima labiau iš informatikos pusės, t. y. kaip į skaičiavimo modelius, pvz. skaičiavimo modelius teisiniame argumentavime (angl. *computational models of argumentation*). Teisės informatika skiriasi nuo informacinių technologijų teisės, kuri dar vadinama informatikos teise. Pastaroji yra teisės šaka. Informatikos teisė reglamentuoja teisinius žmonių santykius, tokius kaip 1) elektroninis parašas, 2) sutarčių sudarymas telekomunikacinėmis priemonėmis pvz. internetu 3) asmens duomenų apsauga ir kt.

Teisės dalykinė sritis yra nevienalytė, nes teisė visuomenėje atlieka skirtinges funkcijas: reguliacinę, informacinię, prevencinę, legislatyvinę ir kt. Šios funkcijos gali būti klasifikuojamos ir kitaip. Todėl ir informacinių sistemų turi skirtingą paskirtį ir atlieka skirtinges užduotis. Pavyzdžiui, teismo administravimo informacinių sistemos turi skirtingą paskirtį negu įstatymų leidybos informacinių sistemos, pvz., <http://www.lrs.lt>.

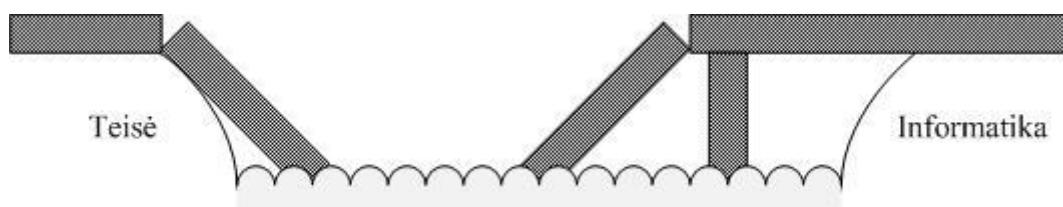
Remiantis Šteinmiulerio ir Garstkos 1970, 1976 ir 1993 m. apibrėžimais [Schweighofer 1999, p. 4], teisės informatika yra teorija apie ryšį tarp elektroninio duomenų apdorojimo ir teisės, o taip pat apie su tuo susijusias prielaidas ir pasekmes. Tokiu būdu teisės informatika suprantama kaip tilto tiesimas tarp teisės ir informatikos. Ši tilto vizualinė metafora pateikta 37.1 pav.



37.1 pav. Teisės informatika vaizduojama tilto metafora tarp teisės ir informatikos

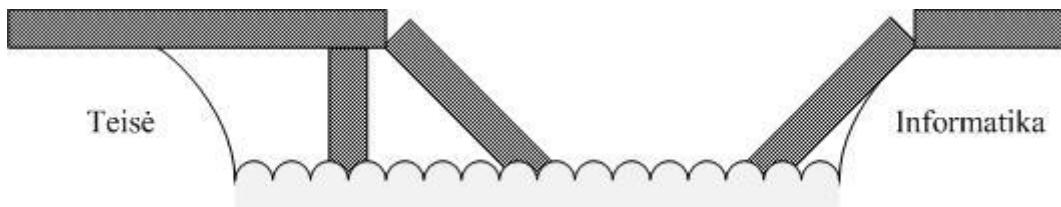
Teisės informatika labiau aktuali mąstytojams teoretikams, negu praktikams. Informatikos praktikai užsiima informaciniemis technologijomis. Teisės praktikai – teisėjai, advokatai, prokurorai, notarai ir tiesiog teisininkai praktikai – atlieka savo funkcijas.

Teisės ir informatikos specialistų bendradarbiavimas turi atitikti jų kompetenciją. Kitais žodžiais, tiltas tarp teisės ir informatikos turi būti statomas subalansuotai. Kai informatikai iš teisininkų tikisi per daug, tai teisininkai neišlaiko ir tiltas griūna. Pavyzdžiui, tiltas neišlaikys, kai informatikai tikėsis, kad teisininkai formalizuos daugiau informatikos, o ne teisinių žinių reikalaujančius darbus. Ši vizualinė metafora pateikta 37.2 pav.



37.2 pav. Tiltas tarp teisės ir informatikos turi būti statomas subalansuotai.
Jeigu informatikai teisininkams užkrauna per daug, tai pastarieji
neišlaiko, ir tiltas griūna

Analogiškai, kai teisininkai per daug tikisi iš informatikų, tai pastarieji taip pat neišlaiko ir tiltas griūna. Pavyzdžiui, tiltas neišlaiko, kai informatikai kurdami matematinius modelius susiduria su teisiinių žinių trūkumu arba nesugeba ižvelgti teisinio samprotavimo specifikos. Ši vizualinė metafora pateikta 37.3 pav.

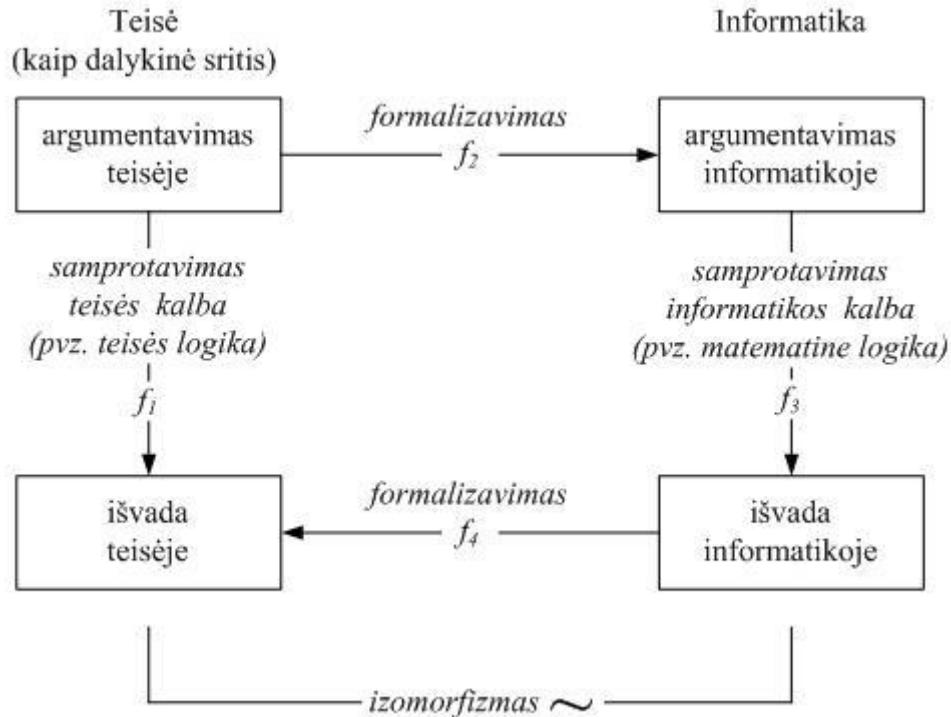


37.3 pav. Kai informatikai patys sau per daug užsikrauna arba teisininkai iš jų per daug tikisi, tai informatikai neišlaiko, ir tiltas griūna

Šiame skyriuje teisės dalykinė sritis, žodžiai „teisė“ ir „teisininkas“ gali būti pakeisti kita dalykinė sritimi. Kai informatika taikoma tam tikroje dalykinėje srityje, tai taikymo vizualizavimui gali būti naudojama tilto metafora. Informatika taikoma, pavyzdžiui, kai yra kuriamą informacinę sistemą tam tikroje dalykinėje srityje.

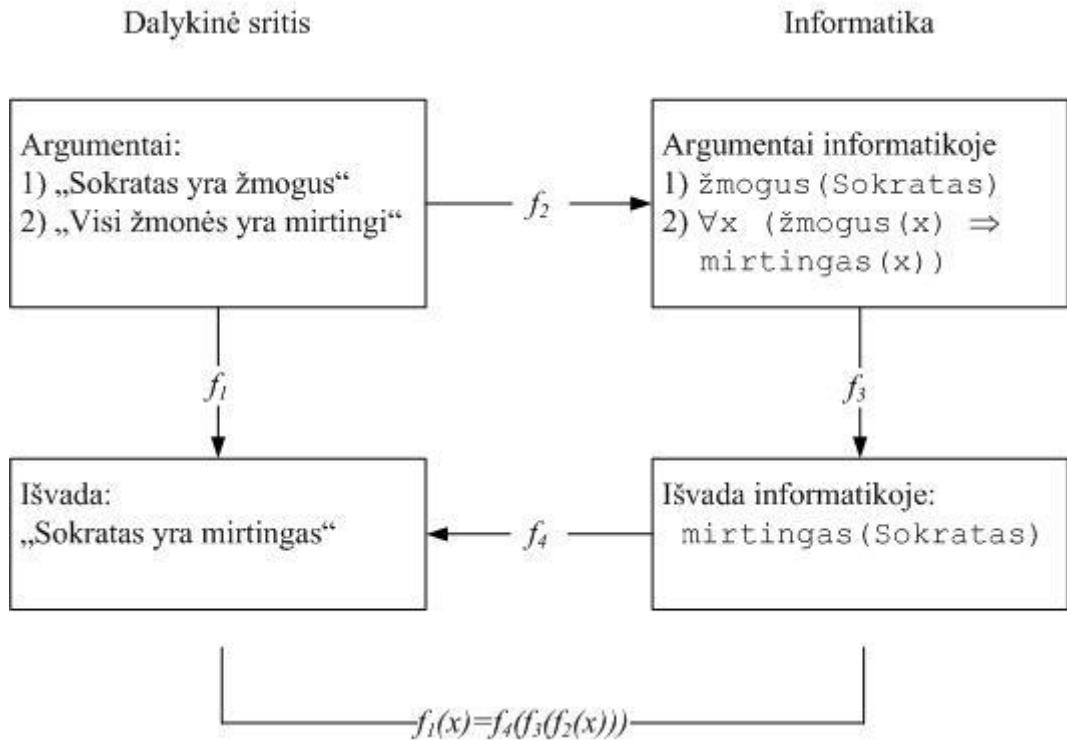
Žinių vizualizavimas (angl. *knowledge visualization*) skiriasi nuo žinių vaizdavimo (angl. *knowledge representation*). Žinių vizualizavimas skiriamas žmogui, kitais žodžiais žinių vizualizavimo rezultatas skirtas žmogui. Tuo tarpu žinių vaizdavimo rezultatas skirtas kompiuteriui. Žinių valdymo enciklopedijoje yra rašoma ir apie žinių vizualizavimą [Eppler, Burkhard 2006], ir apie žinių vaizdavimą [Zarri 2006].

Samprotaujant dalykinės srities kalba, pvz. teisės, turi būti gaunama tokia pati išvada, kaip ir samprotaujant formaliai kalba, pvz. informatikos, matematikos. Tegu atvaizdavimas f_1 žymi samprotavimą dalykinės srities kalba, pavyzdžiui teisės logika. Atvaizdavimas f_2 žymi atvaizdavimą iš dalykinės kalbos į formalią kalbą, pavyzdžiui informatiką, matematinę logiką, matematiką. Atvaizdavimas f_3 žymi samprotavimą formaliai kalba. Atvaizdavimas f_4 žymi perėjimą iš formalios kalbos į dalykinę kalbą. Tada, diagrama 37.4 pav. privalo komutuoti, t.y. $f_1 = f_4 \circ f_3 \circ f_2$, kur \circ žymi atvaizdavimų kompoziciją. Kitais žodžiais $\forall x f_1(x) = f_4(f_3(f_2(x)))$.



37.4 pav. Diagrama privalo komutuoti: $f_1 = f_4 \circ f_3 \circ f_2$, kur \circ žymi atvaizdavimų kompoziciją. Kitais žodžiais $\forall x f_1(x) = f_4(f_3(f_2(x)))$. Samprotaujant dalykinės srities kalba, pvz. teisės, turi būti gaunama tokia pati išvada, kaip ir samprotaujant formalia kalba, pvz. informatikos, matematikos

Diagramos pavyzdys pateikiamas 37.5 pav., kur iliustruojamas išvados „Sokratus yra mirtingas“ išvėdimas iš prielaidų 1) „Sokratus yra žmogus“ ir 2) „Visi žmonės yra mirtingi“.



37.5 pav. Iliustruojamas išvados „Sokratus yra mirtingas“ išvedimas iš prielaidų 1) „Sokratus yra žmogus“ ir 2) „Visi žmonės yra mirtingi“

38. Argumentavimas teisėje

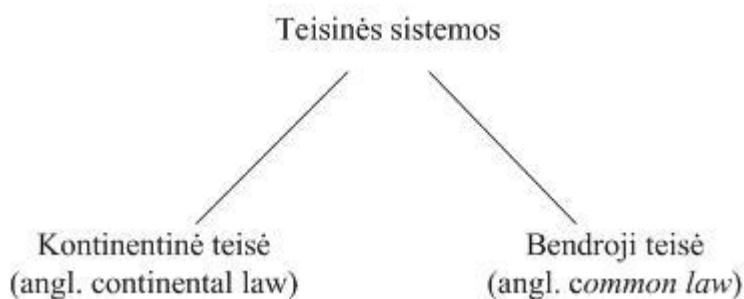
Šis skyrius parengtas remiantis [Bench-Capon 2002]. Jame kalbama apie argumentavimo teisėje formalizavimą.

Pristatomos trys bylos iš amerikiečių teisės vadovėlio. Kiekviena šių bylų yra precedentas, kuriuo gali būti remiamasi kitose bylose. Byloje yra dvi pusės: ieškovas ir atsakovas.

Toliau pateikiami teisės teorijos elementai. Supaptastintai, teisinės sistemos gali būti suskirstytos į kontinentinę teisę (angl. *continental law*) ir bendrąjį teisę (angl. *common law*) (žr. 38.1 pav.).

Precedentai turi svarbų vaidmenį argumentuojant bendrojoje teisėje. Todėl pastaroji dar vadinama precedentine teise arba anglo-saksų teise. Bendroji teisė yra įsigalėjusi Jungtinėje Karalystėje, JAV ir eilėje kitų šalių.

Kontinentinėje teisėje pagrindinis teisės šaltinis yra įstatymas, o ne precedentas. Kontinentinė teisė dar vadinama statutine teise. Kontinentinės teisinė sistema yra įsigalėjusi Lietuvoje, Vokietijoje ir kitose žemyninės Europos šalyse.



38.1 pav. Kontinentinė teisė ir bendroji teisė kaip teisinių sistemų pavyzdžiai

Ginčo teisena ir kontinentinėje ir bendrojoje teisėje vyksta pagal tokį supaprastintą modelį, susidedantį iš trijų žingsnių. Pirma, ieškovas argumentuoja savo ieškinį. Jis pateikia precedentus, kurie pagrindžia jo argumentus. Antra, atsakovas atsikerta. Pastarasis remiasi precedentais, kurie pagrindžia jo argumentus. Trečia, ieškovas atsikerta į atsakovo atsikirtimą. Tokiu būdu, trys žingsniai yra tokie:

1. ieškovas argumentuoja ieškinį;
2. atsakovas atsikerta;
3. ieškovas atsikerta į atsakovo atsikirtimą.

Analogiškus tris žingsnius daro ir atsakovas. Ši pora po tris žingsnius supaprastintai:

1. ieškovas remiasi savo precedentu → atsakovas atsikerta → ieškovas atsikerta į atsikirtimą;
2. atsakovas remiasi savo precedentu → ieškovas atsikerta → atsakovas atsikerta į atsikirtimą.

Toliau teismas, remdamasis ieškovo ir atsakovo argumentais priima sprendimą.

38.1. Trys bylos

Toliau pristatomos trys minėtos bylos iš [Bench-Capon 2002].

Pierson vs. Post (1805 m.) „lapė atviroje vietovėje“. Piersonas medžiojo lapę niekieno žemėje (atviroje visiems). Lapė nuo jo bėga ir ją pagauna Postas. Piersonas paduoda Postą į teismą. Teismo sprendimu bylą laimėjo atsakovas Postas.

Gali atrodyti, kad teisus Piersonas. Tačiau teisėje yra tokia norma: kas užvaldo laukinį gyvūną, to ir nuosavybė.

Keeble vs. Hickerlingill (1707 m.) „antys nuosavoje kūdroje“. Keeblas savo sklype kūdroje jauku viliodavo antis, jas nušaudavo ir pardavinėdavo. Antys buvo jo pragyvenimo šaltinis. Jo kaimynas Hickeringillas iš pavydo šaudydamas į orą gasdindavo antis, kad šios nesileistų į Keeblo kūdrą. Keeblas padavė savo kaimyną į teismą. Teismo sprendimu bylą laimėjo ieškovas Keeblas.

Young vs. Hitchens (1844 m.) „žuvys jūroje“. Žvejai verslininkai Youngas ir Hitchensas gaudė žuvis atviroje jūroje. Youngas pastatė tinklą (apie 1 km. ilgio, pusapskritimo formos) ir palaipsniui ji siaurino. Kai atstumas tarp tinklo kraštų tapo keletas metrų, Hitchensas priešais pastatė nedidelį tinklą ir sugavo visas žuvis. Youngas padavė konkurentą į teismą. Teismo sprendimu bylą laimėjo atsakovas Hitchensas.

Toliau analizuojamas teismo sprendimo trečiojoje byloje pagrindimas precedentais. Išskiriami penki faktoriai. Pro-ieškovo faktoriai:

- A (LIVELIHOOD) – ieškovas siekė pragyvenimo šaltinio;
- B (OWNLAND) – ieškovas buvo savo valdoje.

Pro-atsakovo faktoriai:

- C (NOTCAUGHT) – ieškovas neužvaldė gyvūno;
- D (COMPETE) – atsakovas siekė pragyvenimo šaltinio;
- E (OPEN) – žemė buvo niekieno.

Formalizuojami argumentai, įtakoju sprendimą (žr. 38.2 lentelę).

Metai	Byla. <i>Ieškovas vs. atsakovas. Pabrauktas laimėtojas</i>	A	B	C	D	E	Aplinkybės formaliai
		V2	V3	V1 V3	V2	V3	
		LIVELIHOOD Pragyvenimas	OWNLAND Nuosava žemė	NOTCAUGHT Nepagavo	COMPETE Konkurentai	OPEN Atvira žemė	
Pro-ieškovo faktoriai		Pro-atsakovo faktoriai					

1805	<i>Pierson vs. Post.</i> („lapė atviroje vietovėje“)	-	-	NOTCAUGHT	-	OPEN	C&E
1707	<i>Keeble vs. Hickeringill.</i> („antys nuosavoje kūdroje“)	LIVELIHOOD	OWNLAND	NOTCAUGHT	-	-	A&B&C
1844	<i>Young vs. Hitchens.</i> („žuvys jūroje“)	LIVELIHOOD	-	NOTCAUGHT	COMPETE	OPEN	A&C&D&E

38.2 lentelė. Trijų bylų aplinkybių formalizavimas penkiais faktoriais

Faktoriai A ir B remia ieškovą ir vadinami pro-ieškovo faktoriais. Faktoriai C, D ir E remia atsakovą ir analogiškai vadinami pro-atsakovo faktoriais.

Gali būti bandoma formalizuoti tik keturiais faktoriais. Faktorius OPEN gali būti interpretuojamas kaip priklausomas nuo OWNLAND. Jeigu žemė yra nuosava, tai ji nebegali būti atvira. Tokiu būdu, jeigu yra OWNLAND, tai nėra OPEN. Toliau vis dėl to paliekami penki faktoriai.

Informatikas galėtų siūlyti faktoriams suteikti vienodus svorius. Tada pro-ieškovo ir pro-atsakovo faktorių kiekiei atskirose bylose įgautų santykio pavidalą, pateiktą 38.3 lentelėje.

	Byla	Santykis
1.	<i>Pierson vs. Post</i>	0:2
2.	<i>Keeble vs. Hickeringill</i>	2:1
3.	<i>Young vs. Hitchens</i>	1:3

38.3 lentelė. Pro-ieškovo ir pro-atsakovo faktorių kiekių santykis atskirose bylose

Informatikui šių santykių stipresnioji pusė gali būti pagrindimu, kodėl pirmoje byloje laimėjo atsakovas Postas, antroje ieškovas Keeblas ir trečioje – atsakovas Hitchensas. Tačiau teismas atskiriems faktoriams neteikia lygaus svorio. Teismo pareiga yra pasverti visas bylos aplinkybes ir priimti teisingą sprendimą. Svērimo ir teisingumo samprata sudaro teisės disciplinos esmę. Šių dviejų savokų formalizavimas yra labai sunkus.

38.2. *Svorių suteikimas vektoriaus koordinatėms*

Svorių suteikimo faktoriams problema iliustruojama anekdotu apie Kocmaną.

Ant laivo denio išeina asmuo su bocmano švarku ir civilio kelnėmis. Jūrininkas jo klausia: – Sakykite, ar jūs locmanas? Šis atsako: – Ne. – Ar jūs bocmanas? Šis vėl atsako: – Ne. – Tai kas jūs? Nepažįstamas atsako: – Aš esu Kocmanas.

Esminis klausimas tokis: Kocmano apranga nusveria bocmano ar civilio pusėn?

Tegu aprangos sąvoka formalizuojama kaip susidedanti iš švarko ir kelnų. Gaunamos dvi koordinatės: švarkas ir kelnės. Kiekvienoje atidedamos dvi reikšmės: bocmano 0, civilio 1. Bocmano apranga vaizduojama vektoriumi $u_b=(0,0)$, t. y. bocmano švarkas = 0 ir bocmano kelnės = 0. Civilio apranga vaizduojama vektoriumi $u_c=(1,1)$, t. y. civilio švarkas = 1 ir civilio kelnės = 1. Asmuo su bocmano švarku ir civilio kelnėmis vaizduojamas vektoriu $u_k=(0,1)$, t. y. bocmano švarkas = 0 ir civilio kelnės = 1.

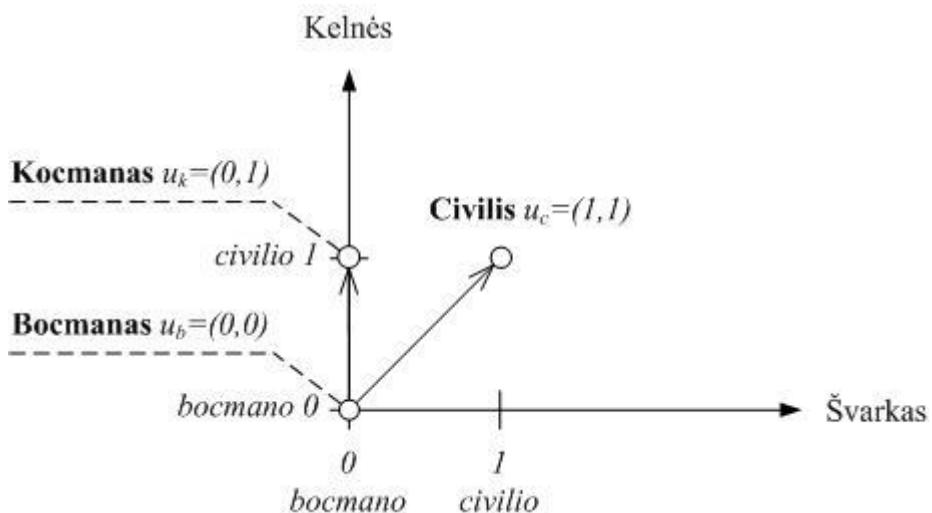
Kocmano vektorius u_k vienodai nutolęs tiek nuo bocmano vektoriaus u_b , tiek nuo civilio vektoriaus u_c (žr. 38.4 pav.):

$$\|u_k - u_b\| = \|u_k - u_c\|$$

Irodymas:

$$\begin{aligned}\|u_k - u_b\| &= \|(0,1)-(0,0)\| = \|(0,1)\| = 1 \\ \|u_k - u_c\| &= \|(0,1)-(1,1)\| = \|(-1,0)\| = 1\end{aligned}$$

Irodymo pabaiga.

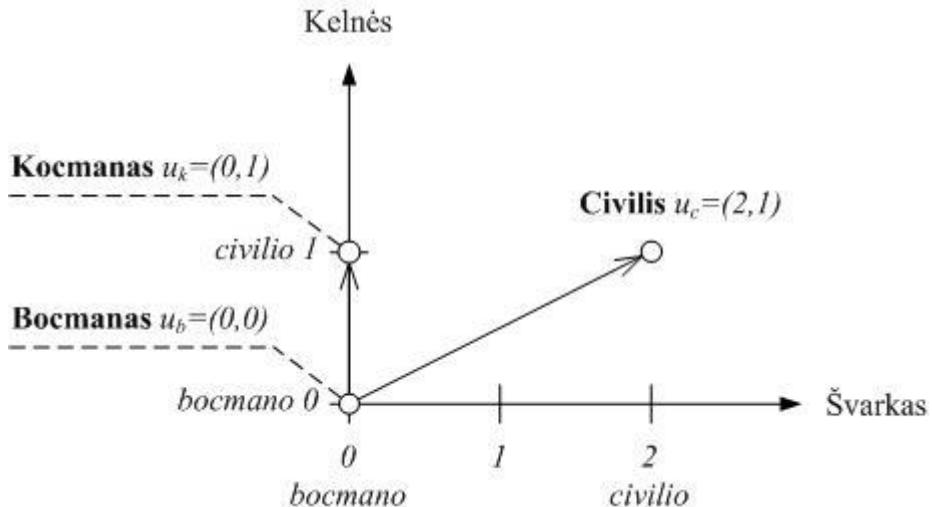


38.4 pav. Bocmano, civilio ir Kocmano vektoriai, kai švarkas ir kelnės turi vienodą svorį

Tegu švarkui suteikiamas svoris 2. Tada bocmano vektorius $v_b=(0,0)$, civilio vektorius $v_c=(2,1)$ ir Kocmano vektorius $v_k=(0,1)$. Tokiu atveju Kocmano vektorius yra arčiau bocmano, negu civilio (žr. 38.5 pav.):

$$\begin{aligned}\|v_k - v_b\| &= \|(0,1)-(0,0)\| = \|(0,1)\| = 1 \\ \|v_k - v_c\| &= \|(0,1)-(2,1)\| = \|(-2,0)\| = 2\end{aligned}$$

Kocmano ir bocmano skirtumas yra vektorius $(0,1)$, kurio ilgis 1. Kocmano ir civilio skirtumas yra vektorius $(-2,0)$, kurio ilgis 2.



38.5 pav. Bocmano, civilio ir Kocmano vektoriai, kai švarkui suteikiamas svoris 2, o kelnėms 1

Švarkui ir kelnėms gali būti suteikiami svoriai c_1 ir c_2 . Kocmano u_k atstumas iki bocmano u_b yra apibrėžiamas kaip šių vektorių skirtumo norma. O norma imama kaip suma atitinkamų koordinacijų $u_{k,i}$ - $u_{b,i}$ absolютinio dydžio, padauginto iš svorio c_i :

$$\|u_k - u_b\| = c_1 \cdot |u_{k,1} - u_{b,1}| + c_2 \cdot |u_{k,2} - u_{b,2}|$$

38.3. Ar penki faktoriai lygiaverčiai?

Pavyzdyme apie tris precedentus faktoriai gali būti formalizuojami ne dviem reikšmėmis *true* ir *false*, bet n reikšmių. Pavyzdžiui, faktoriui NOTCAUGHT galima išskirti užvaldymo gyvūnu laipsnius: *pagavo*, *mirtinai sužeidė*, *sužeidė*, *persekojo*, *pamatė*. Teismas kitaip vertintų aplinkybes, jeigu Postas būtų pagavęs Piersono stipriai sužestą lapę.

Lietuvos respublikos civilinio kodekso 4.59 straipsnis „Laukiniai gyvūnai“ sako: „laisvėje esantys laukiniai gyvūnai, kurie laikantis įstatymų buvo pagauti arba nušauti, tampa juos pagavusiojo arba nušovusiojo nuosavybe, jeigu įstatymų nenustatyta kitaip“. Šis straipsnis gali būti supaprastintas formalizuojamas implikacija:

$$\text{Pagavo_ar_nušovę(Asmuo, Gyvūnas)} \Rightarrow \text{Nuosavybė(Asmuo, Gyvūnas)}$$

Čia neatsižvelgta į įstatymo straipsnyje įtvirtintos normos sudėties elementą „laikantis įstatymų“ bei „jeigu įstatymų nenustatyta kitaip“. Kitaip tariant, jeigu nėra išimčių. Taigi, čia yra taisyklė aiškiems atvejams. Ginčas kyla, kai nėra aišku.

Nagrinėjamame pavyzdyme apie tris bylas būtų galima remtis prielaida, kad faktoriai A, B, C, D ir E turi vienodus svorius, lygius 1. Tačiau samprotavimu apie Kocmaną yra pademonstruota kad svoriai turėtų būti skirtingi ir neaišku kokie.

Toliau analizuojama kaip trečiojoje byloje *Young vs. Hitchens* šalys galėtų remtis precedentais.

Youngas kaip jam palankiu argumentu remiasi *Keeble vs. Hickeringill* precedantu. Jis tvirtina, kad aplinkybės panašios: sutampa faktoriai LIVELIHOOD (A) ir NOTAUGHT (C) (žr. 38.2 lentelę). Hitchensas atsikerta, kad šioje byloje yra pro-atsakovo faktoriai COMPETE

(D) bei OPEN (E), ir nėra pro-ieškovo faktoriaus OWNLAND (B). Youngas neturi kuo atsikirsti į atsikirtimą. Tuomet Hitchensas remiasi *Pierson vs. Post* precedentu. Youngas atsikerta, kad turi pro-ieškovo faktorių LIVELIHOOD (A). Tuomet Hitchensas atsikerta į atsikirtimą, sakydamas, kad turi pro-atsakovo faktorių COMPETE (D). Kaip galima ižvelgti, į abu precedentus buvo atsikirsta. Todėl negalima jais remtis vienareikšmiškai.

Jeigu lyginti atsikirtimus faktorių skaičiumi, tai pirmuoju atveju Hitchensas atsikerta D bei E buvimu ir B nebuvinu. Youngas neturi ką pasakyti. Taigi 0:3 Hitchenso naudai. Antruoju atveju Youngas atsikerta A buvimu, o Hitchensas atsikerta į atsikirtimą faktoriaus D buvimu, taigi 1:1. Tačiau svoris neaiškus.

Kaip buvo minėta, teismas laimėtoju pripažino Hitchensa. Toliau bandoma tai formalizuoti.

Remiantis 38.2 lentelės pirmaja eilute *Pierson vs. Post* precedentas vaizduojamas vektorumi:

$$(\text{LIVELIHOOD}, \text{OWNLAND}, \text{NOTCAUGHT}, \text{OPEN}) \\ \text{arba } (\text{A=true}, \text{B=false}, \text{C=true}, \text{D=false}, \text{E=true}) \text{ ar tiesiog } v_1=(0,0,1,0,1)$$

Analogiškai *Keeble vs. Hickeringill* precedentas vaizduojamas:

$$(\text{LIVELIHOOD}, \text{OWNLAND}, \text{NOTCAUGHT}, \text{OPEN}) \\ \text{arba } (\text{A=true}, \text{B=true}, \text{C=true}, \text{D=false}, \text{E=false}) \text{ ar tiesiog } v_2=(1,1,1,0,0)$$

Taip pat *Young vs. Hitchens* precedentas vaizduojamas:

$$(\text{LIVELIHOOD}, \text{NOTCAUGHT}, \text{COMPETE}, \text{OPEN}) \\ \text{arba } (\text{A=true}, \text{B=false}, \text{C=true}, \text{D=true}, \text{E=true}) \text{ ar tiesiog } v_3=(1,0,1,1,1)$$

Young vs. Hitchens vektoriaus atstumas iki *Pierson vs. Post*, kuriuo rēmėsi Hitchensas yra:

$$v_3-v_1=(1,0,1,1,1)-(0,0,1,0,1)=(1,0,0,1,0) \quad (38.1)$$

Young vs. Hitchens vektoriaus atstumas iki *Keeble vs. Hickeringill*, kuriuo rēmėsi Youngas yra:

$$v_3-v_2=(1,0,1,1,1)-(1,1,1,0,0)=(0,-1,0,1,1) \quad (38.2)$$

Gali būti iškeltas klausimas: v_3 yra arčiau v_1 ar v_2 ? Žiūrint į (38.1) ir žinant kad vienetas pirmojoje pozicijoje žymi pro-ieškovo faktorių LIVELIHOOD, o vienetas ketvirtijoje pozicijoje žymi pro-atsakovo faktorių COMPETE, bet nežinant pastarujų faktorių svorių c_1 ir c_4 nėra kaip nustatyti vektoriaus v_3-v_1 ilgi. Nera kaip teigti kad v_3-v_1 ilgis yra 0.

Vektoriaus v_3-v_2 ilgis, žiūrint iš atsakovo Hitchenso pozicijos, kuris atsikerta ieškovui Youngui yra 3 (vadovaujantis prielaida, kad faktoriai turi lygius svorius). Čia Hitchensas atsikerta kad jo naudai veikia pirmojo faktoriaus (OWNLAND) nebuvinimas ir ketvirtijojo (COMPETE) bei penktijo (OPEN) buvimas, lyginant su v_2 . Tokiu būdu nėra pagrindo priskirti v_3 arčiau v_1 . Tačiau taip pat nėra pagrindo priskirti v_3 arčiau v_2 .

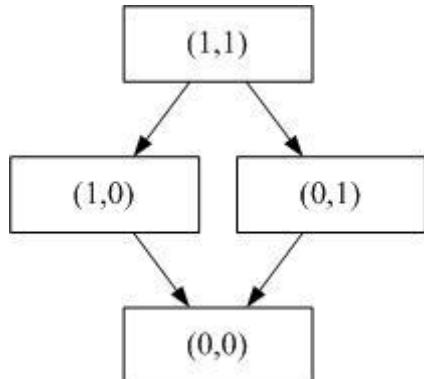
38.4. Formalizavimas remiamiantis faktoriais

Kaip buvo minėta, bendrojoje teisėje kiekviena šalis ieško jai palankių precedentų. Kiekvienas naujas teismo sprendimas gali tapti precedentu. Todėl precedentų daugėja. Tokiu būdu, bylos šaliai palankių precedentų paieška nėra triviali. Norint automatizuoti šių precedentų paiešką reikia formalizuoti paieškos kriterijus.

Precedentinio samprotavimo principas yra, kad panašios aplinkybės salygoja panašius sprendimus.

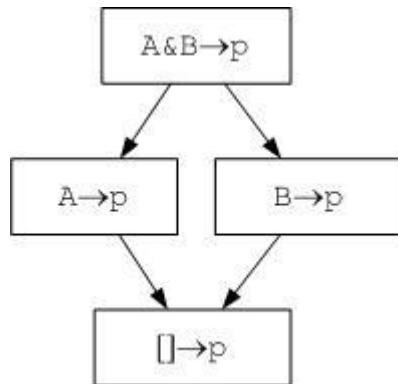
[Bench-Capon 2002] nagrinėja matematinę teoriją, pagrindžiančią teismų sprendimus aukščiau minėtose trijose bylose. Ši teorija vaizduojama matematine struktūra – gardele, o ne vektoriais v_1 , v_2 ir v_3 kurie įvesti aukščiau. Gardelės viršūnė atitinka faktorių rinkinių kaip ir vektoriaus atveju. Gardelė yra apibrėžiama kaip grafas, tarp kurio viršūnių įvestas dalinis sutvarkymas. Šis dalinis sutvarkymas formalizuoją faktorių rinkinių kaip vektorių sutvarkymą.

Toliau apsiribojama gardelėmis, kurių viršūnės žymimos dvejetainiais vektoriais. Pavyzdžiui, dvimačiai vektoriai yra $(0,0)$, $(0,1)$, $(1,0)$ ir $(1,1)$. Vektorius $(1,1)$ yra „geresnis“ negu $(1,0)$, formaliai, $(1,1) > (1,0)$. Taip pat $(1,1) > (0,1)$, $(1,0) > (0,0)$ ir $(0,1) > (0,0)$ (žr. 38.6 pav.).



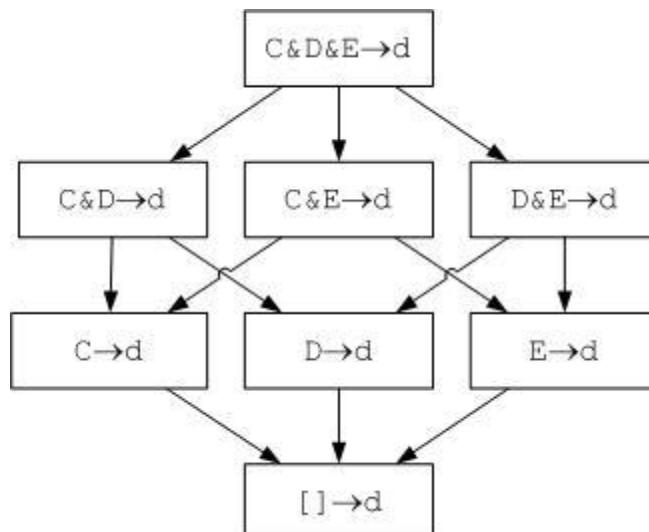
38.6 pav. Gardelė, kuri nustato dvimačių dvejetainių vektorių sutvarkymą:
 $(1,1) > (1,0)$, $(1,1) > (0,1)$, $(1,0) > (0,0)$ ir $(0,1) > (0,0)$

Analogiškai vaizduojama pro-ieškovo (angl. *plaintiff*, sutrumpintai p) faktorių A ir B gardelė (žr. 38.7 pav.). Gardelės aukščiausia viršūnė $A \& B \rightarrow p$ žymi tokį teiginį: jeigu byloje yra abu faktoriai A ir B, tai laimi ieškovas. Viršūnė $A \rightarrow p$, žymi teiginį: jeigu byloje yra faktorius A, tai laimi ieškovas. Analogiškai, viršūnė $B \rightarrow p$, žymi teiginį: jeigu byloje yra faktorius B, tai laimi ieškovas. Viršūnė $[] \rightarrow p$, žymi teiginį: net jeigu byloje nėra nei A, nei B tai laimi ieškovas.



38.7 pav. Gardelė kurioje pavaizduotas pro-ieškovo faktorių A ir B sutvarkymas. Čia p žymi ieškovo (angl. *plaintiff*) laimėjimą

Analogiškai dvimatei pro-ieškovo gardelei yra sudaroma trimatė pro-atsakovo (angl. *defendant*) gardelė (žr. 38.8 pav). Kiekviena gardelės viršūnė atitinka rinkinį iš trijų pro-atsakovo faktorių: C (NOTCAUGHT), D (COMPETE) ir E (OPEN).



38.8 pav. Gardelė kurioje pavaizduotas pro-atsakovo faktorių C, D ir E sutvarkymas. Čia d žymi atsakovo (angl. *defendant*) laimėjimą

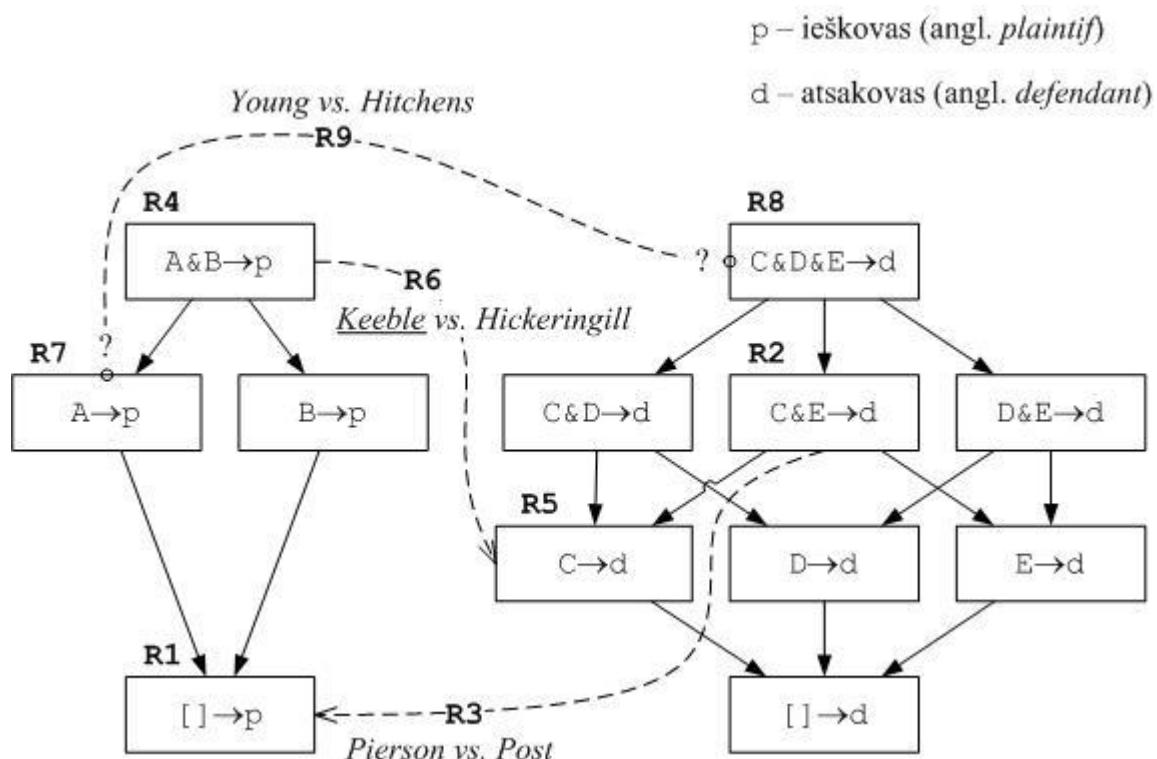
Analizuojant trijų bylų faktorių sudėtį ir atsižvelgiant į laimėtojus Postą bei Keeblą, išskiriamos taisyklės pateiktos 38.9 lentelėje.

<i>Pierson vs. Post</i>
R1: if [] then plaintiff
R2: if C&R then defendant
R3: R2 > R1
<i>Keeble vs. Hickeringill</i>
R4: if A&B then plaintiff
R5: if C then defendant
R6: R4 > R5
<i>Young vs. Hitchens. Kas laimi?</i>
R7: if A then plaintiff
R8: if C&D&E then defendant
R9: R7 < R8 ar R7 > R8?

38.9 lentelė. Trečiosios bylos formalizavimo esminis klausimas yra R9 taisyklė – kaip įvesti sutvarkymą tarp R7 ir R8? Kaip žinoma iš bylos, teismas nugalėtoju pripažino atsakovą Hitchensa

Taisykles R3, R6 ir R9 yra kitokio tipo, nei R1, R2, R4, R5, R7 ir R8. R3, R6 ir R9 yra taisykles, įtvirtinančios kitų taisyklių sutvarkymą. R3 ir R6 įtvirtina teismo sprendimus nugalėtojais pripažinti atitinkamai Postą ir Keeblą. R9 neišplaukia iš samprotavimo remiantis pirmaisiais dviem precedentais.

Gardelė, kurioje pavaizduotos taisykles R1 – R9 yra pateikta 38.10 pav.



38.10 pav. Pirmoji gardelė, vaizduojanti matematinę teoriją, kuri formalizuoją tris bylas. Ši gardelė vaizduoja taisykles 38.9 lentelėje

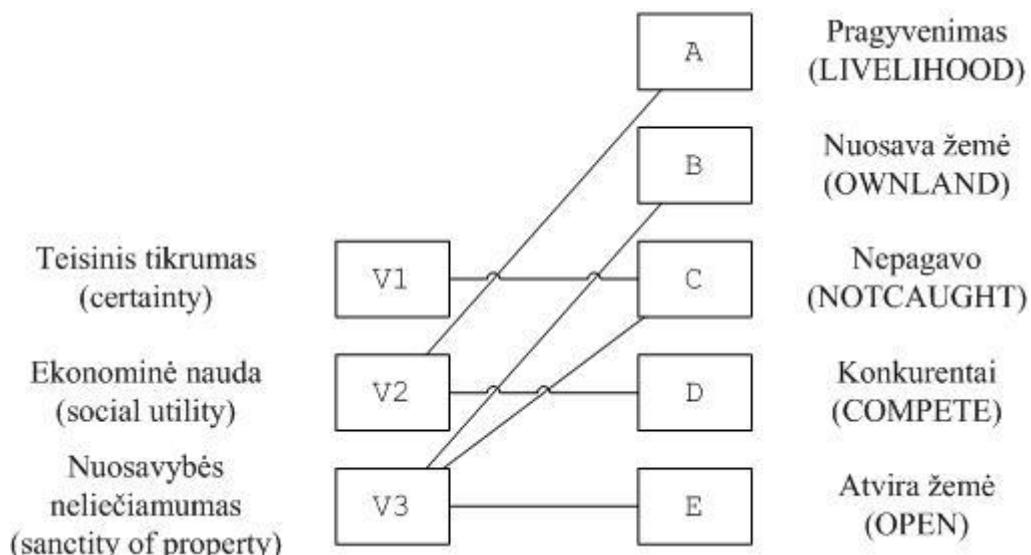
Primenama, kad faktorių kiekis negali lemti sprendimo. Teismas faktorius turi įvairiapusiškai įvertinti ir pasverti. Kad faktorių kiekis negali remti sprendimo iliustruoja pavyzdys su banknotais: Tegu vienas žmogus turi vieną banknotą, o kitas – 20 banknotų. Iš čia negalima pasakyti, kuris turi daugiau pinigų. Dar reikia žinoti banknotų nominalus.

38.5. Formalizavimas remiantis vertybėmis

Toliau nagrinėjama kita matematinė teorija, formalizuojanti tris bylas. Įvedama vertybės sąvoka. Vertybės bus siejamos su faktoriais ir suteiks jiems tam tikrą svorį. Trijų minėtų bylų teismo sprendimų motyvuose yra išskiriamos šios trys vertybės:

- V1 – teisinis tikrumas (angl. *certainty*). Trupinama tikrumas.
- V2 – ekonominė nauda. Ji suprantama kaip bendrojo vidaus produkto (BVP) didėjimas (angl. *promotion of Gross National Product, social utility*). Trumpinama nauda.
- V3 – nuosavybės neliečiamumas (angl. *sanctity of property*). Trumpinama nuosavybė.

Vertybės yra siejamos su faktoriais (žr. 38.11 pav.).



38.11 pav. Vertybų susiejimas su faktoriais

Teisinio tikrumo susiejimą su faktoriu NOTCAUGHT galima paaiškinti šitaip: „jeigu nepagavai, tai ir ne tavo“. Nauda yra suprantama kaip socialinė nauda visuomenei, BVP didinimas. Pavyzdžiu, *Keeble vs. Hickeringill* byloje, Keeblas didino BVP, o Hickeringilas ne.

Remiantis trimis bylomis, yra įvedamas dalinis sutvarkymas tarp V1, V2 ir V3. Toliau aiškinamas sutvarkymas. Kaip žinoma iš kombinatorikos, trijų vertybų sutvarkymų yra $3! = 6$. Iš pradžių pagrindžiama, kodėl $V2 > V1$. V3 vieta sutvarkyme nebus nagrinėjama. Gali būti tik $V2 > V1 > V3$ arba $V2 > V3 > V1$. Sutvarkymas $V3 > V2 > V1$ prieštarautų nusvērimui 38.20 pav., kuris aptariamas toliau.

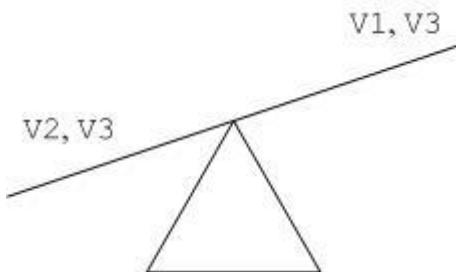
Pirmas žingsnis. Irodoma $V2 > V1$.

Nagrinėjamas antrasis precedentas – *Keeble vs. Hickerlingill* („antys nuosavoje kūdroje“) (žr. 38.12 lentelę).

Faktoriai. Skliausteliuose faktorių attinkančios vertybės				
Pro-ieškovo faktoriai		Pro-atsakovo faktoriai		
A (V2)	B (V3)	C (V1, V3)	D (V2)	E (V3)
LIVELIHOOD	OWNLAND	NOTCAUGHT	-	-

38.12 lentelė. *Keeble vs. Hickerlingill* byla. Ieškovas turėjo V2 ir V3, o atsakovas V1 ir V3. Teismo sprendimu laimėjo Keeblas

Vertybė V3 yra tiek ieškovo tiek atsakovo pusėje, todėl šie egzemplioriai, vienas kitą atsveria. Kadangi teismas laimėtoju pripažino Keeblą, tai išplaukia $V2 > V1$ (žr. žr. 38.13 pav.).



38.13 pav. *Keeble vs. Hickerlingill* byla. Ieškovas turėjo V2 ir V3, o atsakovas V1 ir V3. Kadangi teismo sprendimu laimėjo Keeblas, tai išplaukia, kad $V2 > V1$

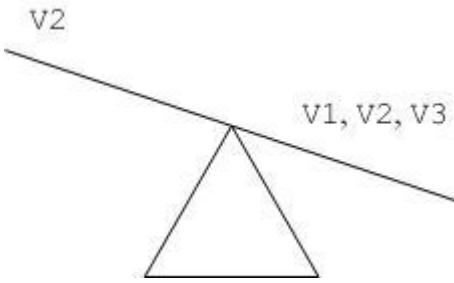
Antras žingsnis. $V3 + V3 = V3$

Trečiojo precedento *Young vs. Hitchens* („žuvys jūroje“) teismo sprendimą galima paaiškinti paaiškinimas remiantis vertybėmis (žr. 38.14 lentelę). Reikia remtis pirmuoju precedentu, *Pierson vs. Post* („lapė atviroje vietovėje“), kurį laimėjo Postas.

Faktoriai. Skliausteliuose faktorių attinkančios vertybės				
Pro-ieškovo faktoriai		Pro-atsakovo faktoriai		
A (V2)	B (V3)	C (V1, V3)	D (V2)	E (V3)
LIVELIHOOD	-	NOTCAUGHT	COMPETE	OPEN

38.14 lentelė. *Young vs. Hitchens* byla. Ieškovas turėjo V2, o atsakovas V1, V2 ir V3. Teismo sprendimu laimėjo Youngas

Vertybų svarstyklės *Young vs. Hitchens* pateiktos 38.15 pav.



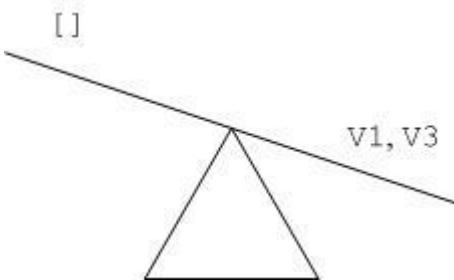
38.15 pav. Vertysiu svarstyklės *Young vs. Hitchens* („žuvys jūroje“) yra analogiškos *Pierson vs. Post* („lapė atviroje vietovėje“) svarstyklėms (žr. 38.17 pav.), kai pridedama V2 abiejose pusėse

V2 yra abiejose svarstyklų pusėse. Hitchensui nusveria V1 ir V3. Nėra kaip remtis *Keeble vs. Hickeringill* precedentu. Remiamasi *Pierson vs. Post* (žr. 38.16 lentelę).

Faktoriai. Skliausteliuose faktorių attinkančios vertybės				
Pro-ieškovo faktoriai		Pro-atsakovo faktoriai		
A (V2)	B (V3)	C (V1, V3)	D (V2)	E (V3)
-	-	NOTCAUGHT	-	OPEN

38.16 lentelė. *Pierson vs. Post* („lapė atviroje vietovėje“) byla. Ieškovas neturėjo vertysiu, o atsakovas turėjo V1 ir V3. Teismo sprendimu laimėjo Postas

Vertysiu svarstyklės *Pierson vs. Post* pateiktos 38.17 pav.



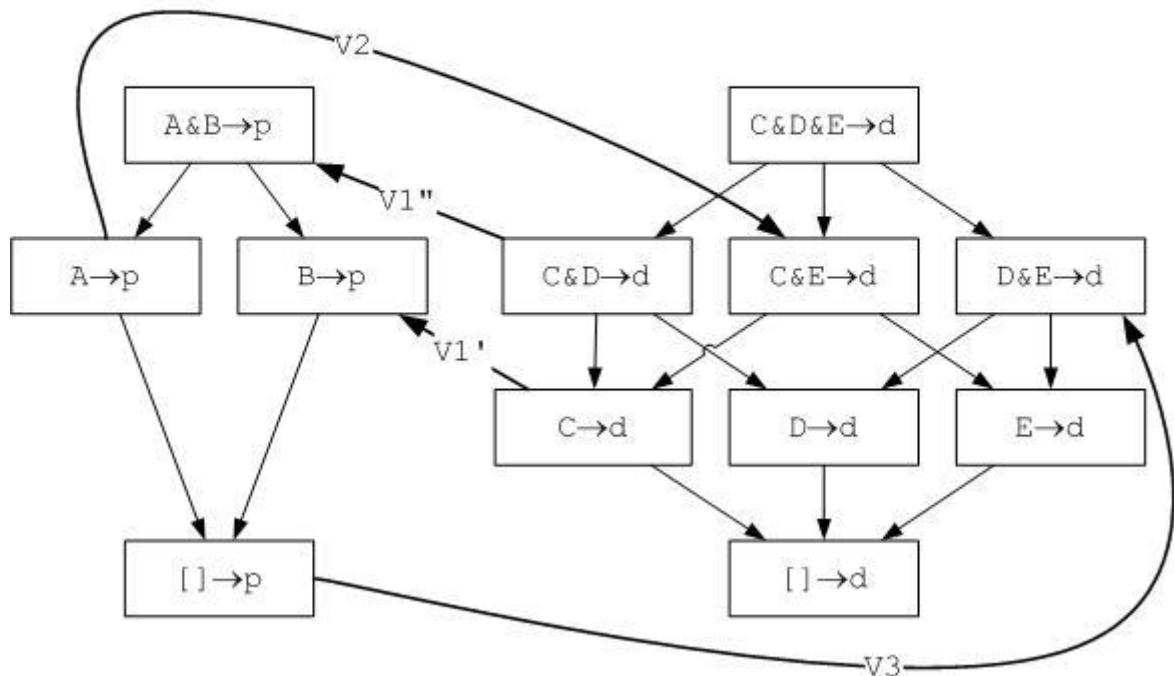
38.17 pav. Vertysiu svarstyklės *Pierson vs. Post* („lapė atviroje vietovėje“)

Teismo sprendimu *Pierson vs. Post* laimėjo Postas. Pagal vertibes $[] < V1, V3$ (žr. 13.17 pav.). Reikia pastebeti, kad V3 atsakovui ateina ir iš pro-atsakovo faktoriaus C ir iš E. O įskaitomos vieną kartą: $V3 + V3 = V3$.

Trečias žingsnis.

Modifikuojama gardelė, kurioje pridedamos briaunos, vaizduojančios vertysiu sutvarkymą (žr. 38.18 pav).

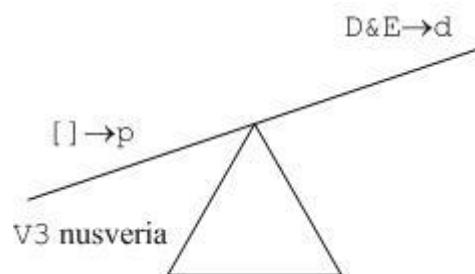
p – ieškovas (angl. *plaintiff*)
 d – atsakovas (angl. *defendant*)



38.18 pav. Antroji gardelė, formalizuojanti tris bylas. Faktorių sutvarkymas grindžiamas vertybėmis V1, V2 ir V3

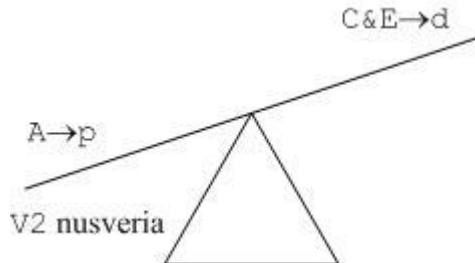
Toliau aptariamos briaunos, kurios įvardinamos vertybėmis V1, V2 ir V3.

1. Briauna V3 (nuosavybės neliečiamumas) iš $[] \rightarrow p$ į $D \& E \rightarrow d$. Nusveria $[] \rightarrow p > D \& E \rightarrow d$ (žr. 38.19 pav.). Primenama, kad D žymi COMPETE (konkurentai) ir E žymi OPEN (atviroje žemėje). Šia briauna tvirtinama, kad kai nėra faktoriaus C (NOTCAUGHT), tai atsakovas d neturi šansų laimėti, t.y. pralaimi. Kodėl? NOTCAUGHT nebuvinimas reiškia, kad atsakovas d neturi įrodymų, kad ieškovas p nepagavo gyvūno. Kitais žodžiais, „nėra nepagavo“ suprantamas kaip p pagavo. Kai ieškovas p tam tikru mastu užvaldo, nekyla ginčas dėl teisinio tikrumo V1. D reiškia, kad jie konkurentai. Toliau BVP neprisklauso nuo to, kuris iš verslininkų pagavo gyvūną. Taigi pagal naudos vertybę V2, jie lygūs. Ieškovas pranašesnis, nes jis tam tikru mastu užvaldė. Todėl ir jo nuosavybė V3. Nėra jokių prieštaravimų kitų vertibių atžvilgiu. Taigi, pranašumą turi ieškovas.



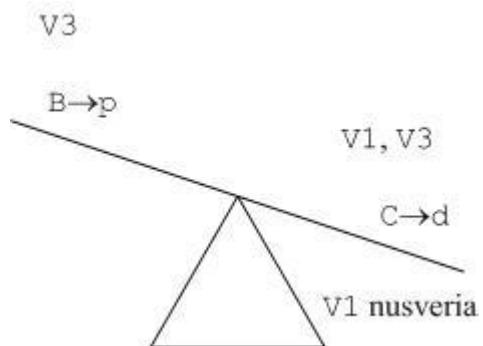
38.19 pav. Briauna V3 (nuosavybės neliečiamumas) $[] \rightarrow p > D \& E \rightarrow d$. Kai nėra C (NOTCAUGHT) tai atsakovas neturi šansų laimėti

2. Briauna V2 (ekominė nauda) iš $A \rightarrow p$ į $C \& E \rightarrow d$. Nusveria $A \rightarrow p > C \& E \rightarrow d$ (žr. 38.20 pav.). Jeigu ieškovas veikia dėl pragyvenimo šaltinio A (LIVELIHOOD), o atsakovas ne, t. y. neturi D (COMPETE), tai nusveria ekominė nauda V2. Tokiu būdu A nusveria bet ką, neturintį D. Čia atsakovas veikia tik dėl pramogos.



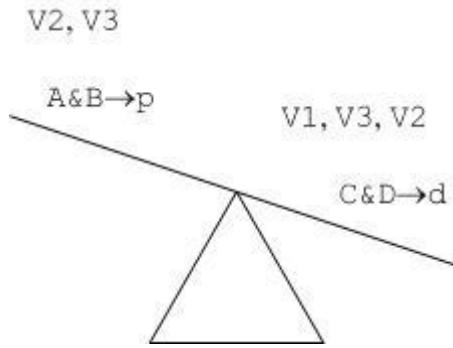
38.20 pav. Ekominė nauda V2 nusveria $A \rightarrow p > C \& E \rightarrow d$, jeigu ieškovas veikia dėl pragyvenimo šaltinio, o atsakovas tik dėl pramogos

3. Briauna v1' (teisinis tikrumas) iš $C \rightarrow d$ į $B \rightarrow p$. Nusveria $C \rightarrow d > B \rightarrow p$ (žr. 38.21 pav.). Jeigu ekominė nauda V2 neliečiamai, tai teisinis tikrumas V1 nulemia. Tokiu būdu C (NOTCAUGHT) laimi prieš bet ką, neturintį A (LIVELIHOOD). Čia primenama, kad A skatina ekominę naudą V2. Atsakovo d argumentas, kad ieškovas p nepagavo (C) nusveria ieškovo buvimą savo žemėje B (OWNLAND).



38.21 pav. Teisinis tikrumas V1 nusveria $C \rightarrow d > B \rightarrow p$, jeigu ekominė nauda V2 neliečiamai

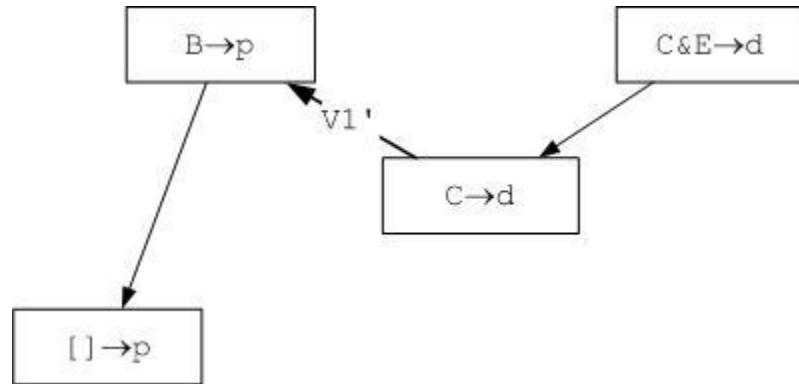
4. Briauna v1'' (teisinis tikrumas) iš $C \& D \rightarrow d$ į $A \& B \rightarrow p$. Nusveria $C \& D \rightarrow d > A \& B \rightarrow p$ (žr. 38.22 pav.). C (NOTCAUGHT) laimi prieš abu A (LIVELIHOOD) ir D (COMPETE). Taip yra, nes ekominė nauda V2 yra abiejose pusėse ir čia nusveria teisinis tikrumas V1.



38.22 pav. C laimi prieš abu A ir D. Nusveria teisinis tikrumas V1

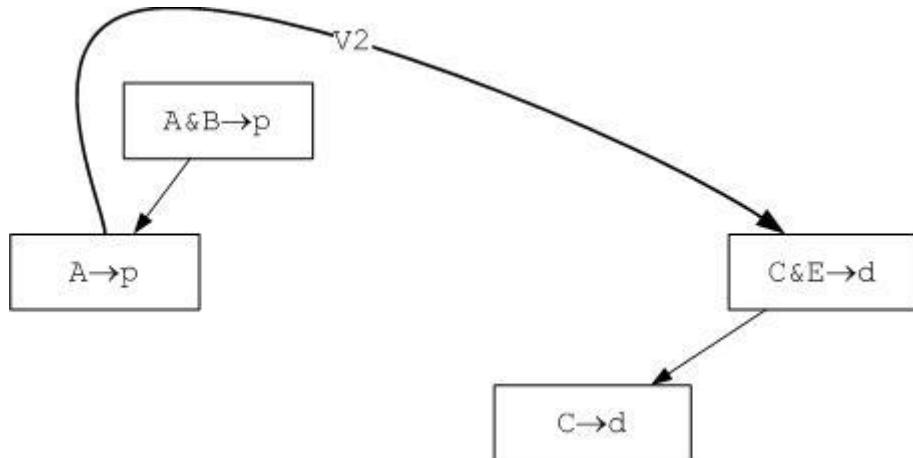
Ši antroji teorija (žr. 38.18 pav.) yra pilna ta prasme, kad jos pagalba išsprendžiamos visos bylos susidedančios iš penkių minėtų faktorių.

Pierson vs. Post išsprendžiama sutvarkymu $C \rightarrow d > [] \rightarrow p$. Faktorius E (OPEN) yra netgi nebūtinės atsakovo pergalei (žr. 38.23 pav.).



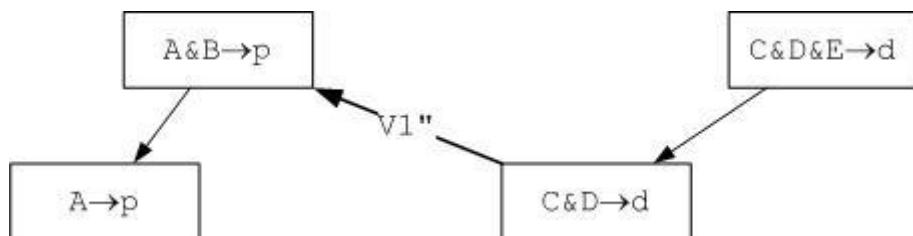
38.23 pav. Atsakovo Posto (d) pergalės *Pierson vs. Post* byloje pagrindimas antraja teorija, pateikta 38.18 pav.

Keeble vs. Hickeringill išsprendžiama sutvarkymu $A \rightarrow p > C \rightarrow d$. Nei B (OWNLAND) buvimas, nei E (OPEN) nebuvinimas nėra būtini ieškovo Keeblo pergalei (žr. 38.24 pav.).



38.24 pav. Ieškovo Keeblo (p) pergalės *Keeble vs. Hickeringill* byloje pagrindimas antraja teorija, pateikta 38.18 pav.

Young vs. Hitchens yra išsprendžiamą sutvarkymu $C \& D \rightarrow d > A \rightarrow p$. Čia E (OPEN) nėra relevantinis.



38.25 pav. Atsakovo Hitchenso (d) pergalės *Young vs. Hitchens* byloje pagrindimas antraja teorija, pateikta 38.18 pav.

Šioje antrojoje teorijoje (žr. 38.18 pav.) yra sumenkintas B (OWNLAND) ir E (OPEN) relevantiškumas su vertybe nuosavybės neliečiamumu V3.

Antrojoje teorijoje išvedamas sutvarkymas $B \rightarrow p > [] \rightarrow d$. Kitais žodžiais, savo žemėje esantis ieškovas, t.y. turintis B (OWNLAND), laimi prieš neturintį argumentų atsakovą. Tačiau vadovaujantis sveiku protu nuosava žemė dar nesuteikia teisės į laukinį gyvūną, užklydusi į šią žemę. Taigi, atsakovas besiremiantis C (NOTCAUGHT) laimi prieš besiriamiantį B ieškovą: $B \rightarrow p < C \rightarrow d$ (žr. 3. Briauna V1' 38.21 pav.).

38.6. Trečioji teorija, sukonstruota remiantis hipotetine byla

Tegu iškeliamą ketvirtąjį, hipotetinę bylą *Kūdros savininkas vs. Medžiotojas*. Asmuo savo žemėje turi ančių kūdrą. Ateina profesionalus medžiotojas, nušauna antis ir pateikia rinkai. Tegu kūdros savininkas iškelia bylą prieš medžiotoją. Bylos faktoriai yra B (OWNLAND), C (NOTCAUGHT) ir D (COMPETE) (žr. 38.26 lentelę).

Faktoriai. Skliausteliuose faktorius attinkančios vertybės				
Pro-ieškovo faktoriai		Pro-atsakovo faktoriai		
A (V2)	B (V3)	C (V1, V3)	D (V2)	E (V3)
-	OWNLAND	NOTCAUGHT	COMPETE	-

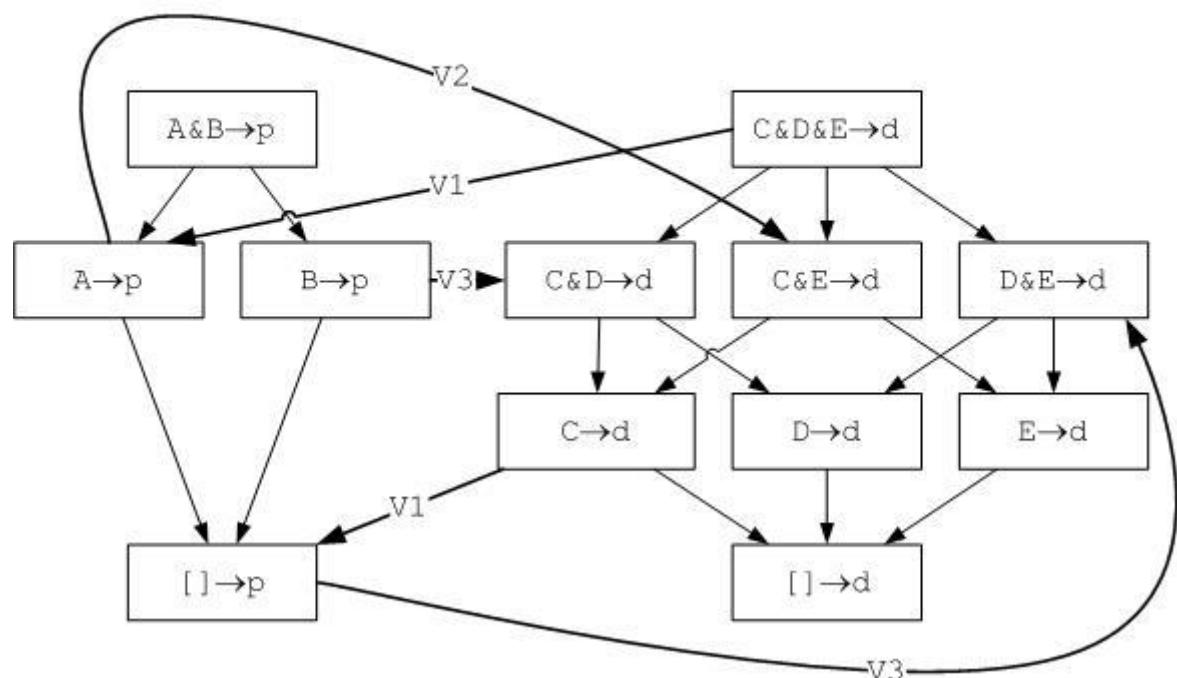
38.26 lentelė. Ketvirtoji, hipotetinė byla *Kūdros savininkas vs. Medžiotojas*

Remiantis antraja teorija, pateikta 38.18 pav., kūdros savininkas pralaimėtų: $B \rightarrow p < C \rightarrow d$ (žr. 3. Briauna V1' 38.21 pav.). Taip yra todėl, kad B ir E nejakoja vertybės V3 (nuosavybės neliečiamumas) kad nusvertų V2 (ekonominė nauda).

Kad teismas nuspręstų ieškovo kūdros savininko naudai, turi būti įvesta briauna $B \rightarrow p > C \& D \rightarrow d$. Gauta teorija pateikta 38.27 pav. Kaip bus matyti, vertybei V3 šitaip suteikiamas didžiausias svoris: $V3 > V2$. Ankčiau jau buvo parodyta $V2 > V1$.

p – ieškovas (angl. *plaintiff*)

d – atsakovas (angl. *defendant*)

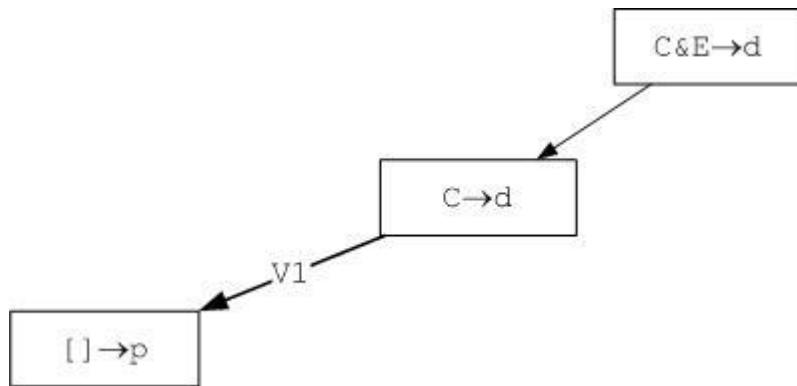


38.27 pav. Trečioji gardelė, formalizuojanti keturias bylas. Nuosavybės neliečiamumas yra labiausiai nusverianti vertybė: $V3 > V2 > V1$

Ir antroji ir trečioji teorijos yra pilnos ir nuoseklios (angl. *coherent*). Trečiosios skirtumas yra kad B skatina V3.

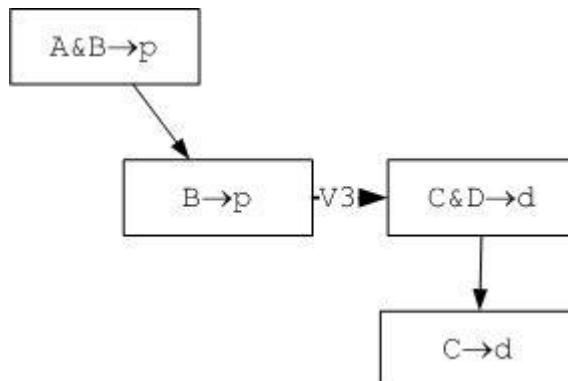
Toliau pateikiami trijų precedentų ir ketvirtojo hipotetinio precedento pagrindimai remiantis trečiaja teorija.

Pierson vs. Post išsprendžiama sutvarkymu $C \rightarrow d > [] \rightarrow p$ (žr. 38.28 pav.).



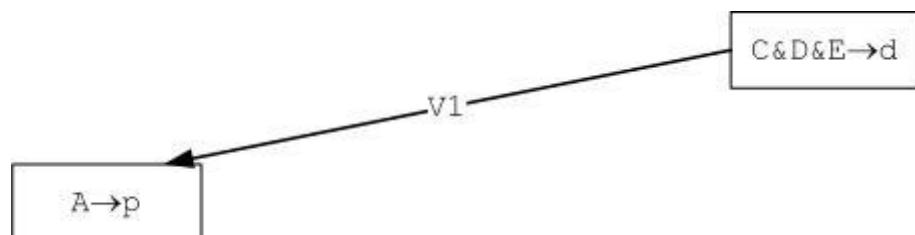
38.28 pav. Atsakovo Posto (d) pergalės *Pierson vs. Post* byloje pagrindimas trečiajā teorija, pateikta 38.27 pav.

Keeble vs. Hickerlingill yra išsprendžiamas sutvarkymu $B \rightarrow p > C \& D \rightarrow d$ (žr. 38.29 pav.). Reikia pastebėti kad Keeblo pergalę galima pagrįsti ir antrojoje teorijoje naudotu sutvarkymu $A \rightarrow p > C \rightarrow d$ (žr. 38.24 pav.).



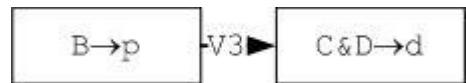
38.29 pav. Ieškovo Keeblo (p) pergalės *Keeble vs. Hickerlingill* byloje pagrindimas trečiajā teorija, pateikta 38.27 pav.

Young vs. Hitchens yra išsprendžiamas sutvarkymu $C \& D \& E \rightarrow d > A \rightarrow p$. (žr. 38.30 pav.).



38.30 pav. Atsakovo Hitchenso (d) pergalės *Young vs. Hitchens* byloje pagrindimas trečiajā teorija, pateikta 38.27 pav.

Hipotetinė ketvirtoji byla *Kūdros savininkas vs. Medžiotojas* yra išsprendžiamas sutvarkymu $B \rightarrow p > C \& D \rightarrow d$ (žr. 38.31 pav).



38.31 pav. Ieškovo kūdros savininko (p) pergalės *Kūdros savininkas* vs. *Medžiotojas* byloje pagrindimas trečiaja teorija, pateikta 38.27 pav.

Reziumuojant atkreipiamas dėmesys į elegantiškus triju matematinių teorijų pavaizdavimus grafas.

39. Literatūra

Pagrindinė literatūra

1. [Brachman, Levesque 2004] Ronald Brachman, Hector Levesque. *Knowledge representation and reasoning*. The Morgan Kaufmann Series in Artificial Intelligence, 2004. 381 p.
2. [Luger 2005] George Luger. *Artificial intelligence: structures and strategies for complex problem solving* (fifth ed.), Addison-Wesley, 2005. 928 p. <http://www.cs.unm.edu/~luger/>.
3. [Nilsson 1982] Nils Nilsson. *Principles of artificial intelligence*. Springer-Verlag, 1982. Vertimas į rusų k: Н. Нильсон. *Принципы искусственного интеллекта*: Перевод с англ. – Москва: Радио и связь, 1985. – 376 с.
4. [Russell, Norvig 2003] Stuart Russell, Peter Norvig. *Artificial intelligence: a modern approach*. Second edition, Prentice Hall, 2003. 1132 p. <http://aima.cs.berkeley.edu>.

Papildoma literatūra

5. [Baronas 2005] Romas BARONAS. Duomenų bazių valdymo sistemos. Vadovėlis aukštosioms mokykloms. Vilnius: TEV, 2005. 184 p. Prieiga per internetą <http://www.mif.vu.lt/~baronas/dbvs/book/>.
6. [Baniulis, Tamulynas 2003] Kazys Baniulis, Bronius Tamulynas. *Duomenų struktūros*. Kaunas: Technologija, 2003. 283 p.
7. [Konar 2000] Amit Konar. *Artificial intelligence and soft computing: behavioral and cognitive modeling of the human brain*. CRC Press, Florida, 2000. 787 p.
8. [MacCrimmon, Tillers 2002] The dynamics of judicial proof: computation, logic, and common sense, Marilyn MacCrimmon, Peter Tillers (ed.). Heidelberg; New York: Physica-Verlag, 2002. 494 p. (Studies in fuzziness and soft computing; Vol. 94)
9. [Nilsson 1998] Nils Nilsson. *Artificial Intelligence: a new synthesis*. Morgan Kaufmann Publishers, 1998. 513 p.
10. [Norgėla 2007] S. Norgėla. Logika ir dirbtinis intelektas. – Vilnius: TEV. 256 p.
11. [Sawyer, Foster 1986] B. Sawyer, D. Foster. Programming expert systems in Pascal. John-Wiley, 1986.
12. [Sowa 2000] John F. SOWA. *Knowledge representation: logical philosophical, and computational foundations*. Brooks/Cole, a division of Thomson Learning. 2000. 573 p.
13. [Stefik 1995] Mark Stefik. *Introduction to knowledge systems*. Morgan Kaufmann Publishers, 1995. 871 p.
14. [Thayse et al. 1990] A. Thayse, P. Gribomont, G. Louis et al. *Approche logique de l'intelligence artificielle*. 1988. Vertimas į rusų kalbą: А. Тейз А., Грибомон П., Луи Ж. и др. *Логический подход к искусственноому интеллекту: от классической логики к логическому программированию*. Перевод с франц. – Москва: Мир, 1990. 432 с.
15. [Waterman 1989] Donald A. Waterman. A guide to expert systems. Vertimas į rusų kalbą: Д. Уотермен. Руководство по экспертным системам. Перевод с англ. – Москва: Мир, 1989.
16. [Wooldridge 2002] Michael Wooldridge. *An introduction to multiagent systems*. John Wiley, UK, 2002. 348 p.

Kita naudota arba cituota literatūra

17. [Ackoff 1989] R. L. Ackoff. From data to wisdom. *Journal of Applied Systems Analysis*, vol. 16, pp. 3-9, 1989.
18. [Archer 1999] A. F. Archer. A Modern Treatment of the 15 Puzzle. *The American Mathematical Monthly*, vol. 106 pp. 793-799, 1999. www.cs.cmu.edu/afs/cs/academic/class/15859-f01/www/notes/15-puzzle.pdf
19. [Atkinson 1997] C. Atkinson. Meta-modeling for distributed object environments. *EDOC'97 Proceedings*. IEEE Computer Society Press.
20. [Bellinger, Castro, Mills 2004] Gene Bellinger, Durval Castro, Anthony Mills. Data, information, knowledge and wisdom. [Interaktyvus, žiūrėta 2006-11-05] <http://www.systems-thinking.org/dikw/dikw.htm>.
21. [Bench-Capon 2002] Bench-Capon T. J. M. (2002). The missing link revisited: the role of teleology in representing legal argument. *Artificial Intelligence and Law*, vol. 10, nos 1-3, pp. 76-94.
22. [Brachman 1988] R. J. Brachman. What IS-A is and isn't: an analysis of taxonomic links in semantic networks. *Principles of expert systems* (eds. A. Gupta, B. E. Prasad). IEEE Press, New York, 1988, pp. 111-117.
23. [Bubelis, Jakimenko 2004] R. Bubelis, V. Jakimenko. Logika. I dalis. Dvireikšmė teiginių logika, argumentacijos teorija: vadovėlis – Vilnius, Lietuvos teisės universiteto Leidybos centras, 2004. – 192 p.
24. [Buchanan et al. 1990] Bruce G. Buchanan, Daniel Bobrow, Randall Davis, John McDermott, Edward H. Shortliffe. Knowledge-based systems. *Annu. Rev. Comput. Sci.* 1990, vol. 4, pp. 395-416 <http://arjournals.annualreviews.org/action/showJournals>
25. [Chandrasekaran et al. 1992] B. Chandrasekaran, T. Johnson, J. Smith. A task-structure analysis for knowledge modeling. *Communications of the ACM*, 35(9): 124–137, 1992.
26. [Chen 1976] Peter Pin-Shan CHEN. The entity-relationship model – towards a unified view of data. *ACM Transactions on Database Systems (TODS)*, vol. 1, no. 1, pp. 9–36, 1976.
<http://portal.acm.org/citation.cfm?id=320434.320440&dl=portal&dl=ACM&idx=J777&part=periodical&WantType=periodical&title=ACM%20Transactions%20on%20Database%20Systems%20%28TODS%29&CFID=3314899&CFTOKEN=29141759>
27. [COMMA 2006] *Computational models of arguments*. Proceedings of COMMA 2006. P. E. Dune, T. Bench-Capon (eds). IOS Press, 2006.
28. [Čaplinskas 1998] A. Caplinskas. AI paradigms. *Journal of Intelligent Manufacturing*, 9(6): 493–502, 1998.
29. [Dagienė, Grigas, Augutis 1986] V. Dagienė, G. Grigas, K. Augutis. *Šimtas programavimo uždaviniai*. Kaunas: Šviesa, 1986. 223 p.
30. [Eppler, Burkhard 2006] M. J. Eppler, R. A. Burkhard. *Knowledge visualization*. Encyclopedia of knowledge management, D. G. Schwartz (ed). Idea Group Reference, Hershey, 2006. pp. 551–559.
31. [Ernst, Newell 1969] G. W. Ernst, A. Newell. GPS: A case study in generality and problem solving. New York: Academic Press.
32. [Filosofija WWW] Filosofijos žodynai (interaktyvus). [Žiūrėta 2006 m. spalio 29 d.]. Prieiga per internetą <http://filo.web1000.com/texts/zodynai/raides/i.htm>.
33. [Geldenhuys 1999] Geldenhuys A.E., van Rooyen H. O., Stetter F. (1999). Knowledge representation and relation nets. Kluwer Academic Publishers, Boston, 279 p.
34. [Gruber 1993] T. J. Gruber. A translation approach to portable ontologies. *Knowledge Acquisition*, 5(2): 1999-220, 1993. <http://www-ksl.stanford.edu/kst/what-is-an-ontology.html>.

35. [Hart, Nilsson, Raphael 1968] P. E. Hart, N. J. Nilsson, B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, vol. 4. no. 2, pp. 100-107, 1968.
36. [Heesen et al. 1997]. C. Heesen, V. Homburg, M. Offereins. An agent view on law. *Artificial Intelligence and Law*, vol. 5, no. 4, pp. 323-340 (1997).
37. [Jensen, Wirth 1982]. K. Jensen, N. Wirth. Pascal. *User manual and report*. Springer, 1978. Vertimas į rusų kalbą: Йенсен К., Вирт Н. Руководство для пользователя и описание языка. Перевод с англ. – Москва: Финансы и статистика. 1982. – 152 с.
38. [Klenk 2011] V. Klenk. *Kas yra simbolinė logika*. – Vilnius: Vilniaus universiteto leidykla. – 485 p. Versta iš Virginia Klenk, Understanding Symbolic Logic, 5th ed., 2008, Pearson Prentice Hall.
39. [McCarthy 1958] John McCarthy. Programs with common sense. *Mechanisation of Thought Processes, Proc. Symp. Nat. Phys. Lab.*, vol. 1, pp. 77-84. London: Her Majesty's Stationery Office. (Reprinted in SIP, p. 403-410).
40. [Marcijonas, Sudavičius 2003] A. Marcijonas, B. Sudavičius. Mokesčių teisė. – Vilnius: Teisinės informacijos centras, 2003. – 288 p.
41. [MAI] Lietuvos Respublikos mokesčių administravimo įstatymas, 2004 m. balandžio 13 d. Nr. IX-2112, Žin., 2004, Nr. 63-2243.
42. [Minsky 1975] M. Minsky. A Framework for Representing Knowledge. *The Psychology of Computer Vision*. R. H. Winston (ed.). Vertimas į rusų kalbą: М. Минский. Структура для представления знания. В сб. под ред. П. Уинстона *Психология машинного зрения*. Перевод с англ. – Москва: Мир. 1978. – с. 249-338.
43. [Norgėla, Sakalauskaitė 2007] S. Norgėla, J. Sakalauskaitė. Neklasikinės logikos informatikams. – Vilnius: TEV, 2007. – 132 p.
44. [OED 2007] *Oxford English dictionary*. Oxford University Press, 2007. <http://dictionary.oed.com/>
45. [Plečkaitis 2004] R. Plečkaitis. *Logikos pagrindai*. – Vilnius: Tyto alba, 2004. – 438 p.
46. [Schweighofer 1999] E. Schweighofer. *Legal knowledge representation. Automatic text analysis in public international and European law*. Kluwer Law International, The Hague, 1999.
47. [Turing 1950] A. M. Turing. Computing machinery and intelligence. *Mind*, 1950. Vol. 59. Nr. 236, p. 433–460.
48. [Tiuringas 1961] A. M. Tiuringas. Bendroji ir loginė automatių teorija. Vilnius, 1961.
49. [Tyugu 1984] Э. Х. Тыугу. Концептуальное программирование. – Москва: Наука, 1984. Vertimas į anglų kalbą: E. Tyugu. Knowledge-based programming. Wokingham: Turing Institute with Addison-Wesley, 1987. 243 p.
50. [Vaišvila 2004] A. Vaišvila. *Teisės teorija*. Mykolo Romerio universitetas. – 2-asis papildytas ir pataisytas leidimas. – Vilnius: Justitia, 2004. – 527 p.
51. [Valente 1995] A. Valente. Legal knowledge engineering: A modelling approach. IOS Press, Amsterdam, 217 p.
52. [van Harmelen et. al 2008] Handbook of knowledge representation. F. van Harmelen, V. Lifschitz, B. Porter (eds.). Elsevier, Amsterdam, 2008.
53. [Zarri 2006] G. P. Zarri. *Knowledge representation*. Encyclopedia of knowledge management, D. G. Schwartz (ed). Idea Group Reference, Hershey, 2006. pp. 467–477.