

Vilniaus universitetas  
Matematikos ir informatikos fakultetas  
Programų sistemų katedra

---

**Vytautas Čyras**

**INTELEKTUALIOS SISTEMOS**

**DIRBTINIS INTELEKTAS  
IR  
ŽINIŲ VAIZDAVIMAS**

<http://www.mif.vu.lt/~cyras/AI/konspektas-intelektualios-sistemos.pdf>

Vilnius  
2017

## Turinys

<b>Pirma dalis .....</b>	<b>6</b>
<b>1. Dirbtinio intelekto samprata .....</b>	<b>7</b>
1.1. Dirbtinio intelekto istorija .....	7
1.2. Dirbtinio intelekto klasifikacija ir tyrimo objektas .....	7
1.3. Žodžio „intelektualus“ samprata dirbtiniame intelekte .....	9
1.4. Hanojaus bokštai .....	10
1.5. Dirbtinio intelekto sampratos kitimas .....	13
1.6. Uždavinių sprendimo principai .....	14
<b>2. Dirbtinio intelekto sistema kaip produkcijų sistema.....</b>	<b>15</b>
2.1. Žirgo maršruto programos testavimas.....	19
<b>3. Valdymas su grįžimais ir procedūra BACKTRACK.....</b>	<b>21</b>
3.1. Predikato DEADEND pavyzdys .....	23
<b>4. Valdoviu išdėstymas šachmatų lentoje .....</b>	<b>25</b>
<b>5. Euristika .....</b>	<b>27</b>
5.1. Euristika valdovių uždavinyje .....	28
5.2. Dar viena euristika: valdovę statyti žirgo éjimu .....	31
<b>6. Patobulinta paieškos su grįžimais strategija. Procedūra BACKTRACK1 .....</b>	<b>39</b>
<b>7. Labirinto uždavinys – kelio paieška į gylį .....</b>	<b>41</b>
7.1. Paieškos algoritmai BACKTRACK ir BACKTRACK1 variantai .....	44
7.2. Produkcių algebrinės savybės .....	46
7.3. Testavimas.....	46
<b>8. Paieška į plotį labirinte – bangos algoritmas .....</b>	<b>49</b>
8.1. Testavimas.....	53
<b>9. Paieška į plotį grafe be kainų.....</b>	<b>55</b>
<b>10. Paieška grafe su kainomis .....</b>	<b>59</b>
<b>11. Paieškos į gylį algoritmas grafe .....</b>	<b>62</b>
11.1. Paieškos „i plotį“ skirtumas nuo „i gylį“ .....	65
11.2. Sprendéjas ir planuotojas .....	66
<b>12. Prefiksinė, infiksinė ir postfiksinė medžio apéjimo tvarka.....</b>	<b>67</b>
<b>13. Bendras paieškos grafe algoritmas GRAPHSEARCH .....</b>	<b>73</b>
<b>14. Skirtumai tarp BACKTRACK1 ir GRAPHSEARCH.....</b>	<b>74</b>
14.1. Paieška grafe .....	74
14.2. Pavyzdys paieškai labirinte su ciklais .....	75
14.3. Kontrpavyzdys, kai pranašesnis BACKTRACK1 .....	79
<b>15. Valdymo metodas „kopimas į kalną“ .....</b>	<b>81</b>
<b>16. Manheteno atstumas.....</b>	<b>85</b>
<b>17. Algoritmas A* .....</b>	<b>87</b>
17.1. A* pavyzdys su euristine funkcija h(n) Manheteno atstumu .....	87
17.2. A* pavyzdys keliu žemėlapje.....	91

<b>18. Tiesioginis ir atbulinis išvedimas produkcijų sistemoje.....</b>	<b>93</b>
18.1. Tiesioginio išvedimo algoritmas .....	93
18.2. Atbulinio išvedimo algoritmas .....	95
18.3. Programų sintezės elementai .....	96
18.4. Perteklinės produkcijos tiesioginiame išvedime .....	98
18.5. Perteklinės produkcijos atbuliniame išvedime .....	99
18.6. Tiesioginio išvedimo sudėtingumas .....	100
18.7. Testai tiesioginiam išvedimui .....	100
18.8. Testai atbuliniams išvedimui .....	103
<b>19. Rezoliucijų metodas.....</b>	<b>105</b>
19.1. Atbulinis ir tiesioginis įrodymai su rezoliucijos taisykle .....	108
19.2. Pavyzdys: trys produkcinės taisyklos .....	110
19.3. Pavyzdys: rezoliucija teoremos įrodyme .....	112
19.4. Logikos vaidmuo samprotavime .....	114
<b>20. Ekspertinės sistemos.....</b>	<b>116</b>
<b>21. Internetinė parduotuvė .....</b>	<b>121</b>
<b>22. Tiuringo testas.....</b>	<b>129</b>
<b>23. Neįmanomumas pasiekti keletą tikslų .....</b>	<b>131</b>
23.1. Nusikaltelio nubaudimo pavyzdys .....	131
23.2. Veiksmų įvertinimas pagal skatinamą ir žeminamą tikslų santykį .....	133
23.3. Planas argumentavimui už viešuosius darbus cs.....	133
23.4. Argumentavimas už įkalinimą $pr^+ >^1 pr^-$ .....	134
23.5. Argumentavimas už baudą $fi^+$ .....	135
23.6. Argumentavimas už viešuosius darbus $cs^+ >^2 cs^-$ .....	136
23.7. Antrasis žingsnis: viešieji darbai nurungia įkalinimą, $cs^+ >^3 pr^+$ .....	137
23.8. Trečiasis žingsnis: viešieji darbai nurungia baudą, $cs^+ >^4 fi^+$ .....	138
23.9. Gėrimas ar bandelė? .....	139
<b>24. Intensionalas, ekstensionalas ir ontologija .....</b>	<b>141</b>
24.1. Ženklai.....	142
24.2. Kas yra konceptualizacija?.....	145
24.3. Kas yra formaliai išreikštinė specifikacija? .....	149
24.4. Skirtingi modeliai tai pačiai specifikacijai .....	153
<b>25. Dirbtinio intelekto egzamino bilietai.....</b>	<b>156</b>
<b>Antra dalis.....</b>	<b>157</b>
<b>26. Žinių vaizdavimas kaip dirbtinio intelekto šaka .....</b>	<b>157</b>
26.1. Duomenys, informacija ir žinios .....	159
26.2. Žmogiškasis pažinimas ir žiniomis grindžiamos sistemos.....	161
26.3. Žinių vaizdavimas ir žinių vadyba .....	163
26.4. Žinių vaizdavimo būdų apžvalga .....	164
<b>27. Programos deklaratyvus vaizdavimas jėjimu–išėjimu.....</b>	<b>166</b>
27.1. Procedūrinis žinių vaizdavimas.....	166

27.2. Deklaratyvus žinių vaizdavimas .....	167
27.3. Įėties–išeities vaizdavimo pavyzdys .....	170
<b>28. Semantinis tinklas.....</b>	<b>172</b>
<b>29. Bendrinė sąvoka ir individuali sąvoka.....</b>	<b>176</b>
29.1. Ryšio is-a interpretacijos.....	177
29.2. Ryšio instance-of interpretacijos .....	182
<b>30. Klasifikavimo vaizdavimas .....</b>	<b>185</b>
30.1. Algebrinis požiūris į klasifikavimą .....	186
30.2. Santykio sąvoka .....	187
30.3. Tiesinės erdvės faktorizavimas pagal tiesinį poerdvį.....	190
<b>31. Semantiniai tinklai pagal Russell ir Norvig .....</b>	<b>193</b>
31.1. Paveldėjimas .....	194
31.2. Santykis ir ryšys .....	195
31.3. Atvirkštinis santykis .....	196
<b>32. Konceptiniai modeliai duomenų bazėse .....</b>	<b>201</b>
<b>33. Žinių vizualizavimas.....</b>	<b>207</b>
<b>34. Teisės informatika .....</b>	<b>209</b>
<b>35. Literatūra .....</b>	<b>212</b>

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/AI/konspektas-intelektualios-sistemos.pdf>>.

## Pratarmė

Šis vadovėlis parengtas kaip kursų „Dirbtinis intelektas“, „Žinių vaizdavimas“ ir „Intelektualios sistemos“ paskaitų konspektas. *Dirbtinis intelektas* (DI) yra tarpšakinė disciplina, kurioje susipynusios informatikos, matematikos, technikos mokslų, filosofijos ir kitų mokslų gijos. Pasaulyje yra pripažintų, bet skirtinį požiūrių į DI. Šiame vadovėlyje DI pristatomas kaip informatikos plačiaja prasme (*computer science, computing*) šaka, o *Žinių vaizdavimas* (ŽV) kaip DI sudėtinė dalis.

Kiekvienas aukščiau minėtas mokslas turi specifinę dalyką. Analogiškai ir DI bei ŽV turi joms būdingą *dvasią (spirit)*, kuri ir pristatoma. Mokslo dalykas siejamas su prasme, koncepcija, idėja, t. y. tai kas išeina už teksto. Čia kyla problema, kaip tą prasmę pavaizduoti. Aš iškeliu visų pirma žmoniškajį pažinimą, o tik paskui intelektualizuotų sistemų kūrimą. Bendra šaknis žodžiuose „žinios“ bei „pažinimas“ ir apibūdina tokią nuostatą.

DI dėstau dvasia, kuria parašyta Nilso Nilsono knyga „Dirbtinio intelekto principai“ (1982). Kurse tiesiogiai remiamasi tik pirmaisiais trimis minėtos knygos skyriais. Toliau paminėčiau George Luger'io (2005) knygą.

Dėstoma jau klasika tapusi medžiaga. Vadovaujamasi Nilsono požiūriu – dirbtinis intelektas kaip produkcių sistema. *Dirbtinio intelekto sistema* yra suprantama kaip trejetas: *globali duomenų bazė, produkcių aibė ir valdymo sistema*. Kurse pagrindinis dėmesys yra skiriamas uždavinių sprendimui naudojant paiešką (*problem solving by search*). Atkreipiame dėmesį, kad DI kontekste anglų kalbos *problem* ir *problem solving* versime „uždavinys“ ir „uždavinių sprendimas“ (kas atitinka rusų kalboje yra prigijusius vertimus „задача“ ir „решение задач“).

Terminas „dirbtinis intelektas“ siejamas su žodžiu „intelektualus“. DI dalykas apima žmogaus kuriamas sistemas, ir todėl geriau tiktų žodis „intelektualizuotos, nes sistemos nemastą; mąstymą modeliuoja programuotojai. Intelektualias sistemas suprantame kaip programų sistemas, kuriose naudojami DI principai. Tokiu būdu į DI žiūrime programavimo ir programų sistemų inžinerijos požiūriais.

Šiame vadovėlyje vadovaujamasi matematiniu metodu, tačiau atsižvelgiama į praktiką socialiniuose ir humanitariniuose moksluose. Siekiama pateikti fundamentinę medžiagą. Tekste nemažai cituojama – vadovaujamasi humanitarinių ir socialinių mokslų tradicija.

Dirbtinio intelekto dvasią autorius supranta kaip dirbtinio intelekto principų visumą. Kuo skiriasi metodas nuo principio? Metodas taikomas esant apibrėžtai sąlygų sudėčiai. Principais vadovaujamasi, kai situacija nėra aiški. Jeigu uždavinui išspręsti nėra žinomas metodas, tai tenka vadovautis principais.

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

Rengiant vadovėli talkino nemažai studentų, kurie klausėsi paskaitų ir konspektavo. Valdas Birbilas, Antanas Vilimas, Indrė Lukauskaitė ir Marek Meškevič buvo pirmieji, užrašę paskaitų turinį failuose. Vėliau daug prisidėjo Edgaras Abromaitis. Talkino Vera Afanasjeva, Dalius Masiliūnas, Laimis Vaikus, Rimantas Žakauskas, Eglė Saulėnaitė, Aivaras Ruzveltas, Vaidas Bielskis, Dainius Kunigauskas, Jonas Rudžianskas ir kiti. Vadovėlis nebūtų sukurtas, jeigu ne Justo Arasimavičiaus kūrybinis darbas konspektuojant ir perkeliant į failą. Atsakomybė už klaidas tenka autorui V. Čyrui.

## Pirma dalis

## 1. Dirbtinio intelekto samprata

Dirbtinis intelektas gali būti apibūdinamas kaip mokslas apie kūrimą tokį kompiuterizuotų sistemų, kurios atlieka užduotis, išprastai reikalaujančias žmogiškojo intelekto.<sup>1</sup> Pradėsime termino kilmės istorija ir tėsime aiškindami sampratą.

### 1.1. Dirbtinio intelekto istorija

2006 metais buvo minima termino „dirbtinis intelektas“ (*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 konferencijoje JAV Dartmouth'o mieste, 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 programos įtakojo pažangą kitose srityse, pvz., informacijos paiešką internete, žmogiškojo supratimo automatizavimo ribų supratimą.

Dirbtinis intelektas neatsiejamas nuo fakto, kad dešimtmetyje nuo 1940-ųjų buvo sukurtas elektroninis kompiuteris ir kitame dešimtmetyje nuo 1950-ųjų jau rašomas programos žmogaus gebėjimams priskiriama veiklai. Tradiciškai DI siejamas su Alano Tiuringo (*Alan Turing*) iškeltu klausimu „ar gali mašina mąstyti?“ (Turing 1950; Tiuringas 1961). Todėl Tiuringas yra laikomas vienu iš DI pradininkų. Išairių autorių DI apibrėžtys akcentuoja skirtingus požiūrius į DI ir užduočių motyvus.

Kadangi dirbtinis intelektas yra siejamas su matematine logika, o matematinė logika siejama su filosofijos šaka logika, kurios pradininkai yra senovės Graikijos filosofai Sokratas, Platonas, Aristotelis ir kt., tai mes DI galėtume sieti su antika. Žodis logika kilęs iš senosios graikų kalbos (gr. *logos* – žodis, kalba, protas, samprotavimas).

Dirbtinis intelektas yra informatikos (*computer science*) šaka ir DI dalykas yra kūrimas tokį mašinų, kurių elgseną žmonės laiko protinga. „Protingų“ mašinų kūrimas domino žmones nuo senų laikų.

### 1.2. Dirbtinio intelekto klasifikacija ir tyrimo objektas

Dirbtinis intelektas yra informatikos šaka. Žemiau pateikiama DI vieta JAV priimtoje ACM (*Association for Computing Machinery*) informatikos plėčiaja prasme (kompiuterijos, *computing*) šakų klasifikacijoje ACM Computing Classification System (žr. <http://www.acm.org/about/class/1998/> ir <http://www.acm.org/about/class/2012/>). Informatikos šakos yra šios:

Computing – informatika

- A. General Literature – bendra literatūra
- B. Hardware – techninė įranga
- C. Computer Systems Organization – kompiuterinių sistemų sandara
- C'. Networks – tinklai (išterpta 2012 m.)
- D. Software and its Engineering – programinė įranga

<sup>1</sup> *Artificial intelligence* – the science of designing computer systems to perform tasks that would normally require human intelligence (Sowa 2000, p. XI).

- F. Theory of Computation – skaičiavimo teorija (programavimas matematinėmis mašinomis)
- G. Mathematics of Computing – kompiuterijos matematika
- H. Information Systems – informacinių sistemų
- H'. Security and Privacy – sauga ir privatumas (iterpta 2012 m.)
- H''. Human-Centered Computing (iterpta 2012 m.)
- I. Computing Methodologies – kompiuterijos metodikos
- J. Applied Computing – kompiuteriniai taikymai
- K. Social and Professional Topics – socialinės ir profesinės temos

Dirbtinis intelektas yra šakoje I. Computing Methodologies:

- I. Computing Methodologies – informatikos metodikos
  - 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 kompiuterinė rega
  - I.5 PATTERN RECOGNITION – šablonų atpažinimas
  - ...

Dirbtinis intelektas yra skaidomas į šakas, tarp kurių yra Žinių vaizdavimas:

- I.2 ARTIFICIAL INTELLIGENCE – dirbtinis intelektas (toliau pagal 2012 m.)**
  - Natural language processing – natūralios kalbos apdorojimas
    - Information extraction; Machine translation; Discourse, dialogue and pragmatics; Natural language generation; Speech recognition; Lexical semantics; Phonology / morphology; Language resources
  - **Knowledge representation and reasoning – žinių vaizdavimas ir samprotavimas**
    - Description logics; Semantic networks; Nonmonotonic, default reasoning and belief revision; Probabilistic reasoning; Vagueness and fuzzy logic; Causal reasoning and diagnostics; Temporal reasoning; Cognitive robotics; Ontology engineering; Logic programming and answer set programming; Spatial and physical reasoning; Reasoning about belief and knowledge
  - Planning and scheduling – planavimas ir tvarkaraščių sudarymas
    - Planning for deterministic actions; Planning under uncertainty; Multi-agent planning; Planning with abstraction and generalization; Robotic planning
  - Search methodologies – paieškos metodikos
    - Heuristic function construction; Discrete space search; Continuous space search; Randomized search; Game tree search; Abstraction and micro-operators; Search with partial observations
  - Control methods – valdymo metodai
    - Robotic planning; Computational control theory; Motion path planning
  - Philosophical/theoretical foundations of artificial intelligence – Filosofiniai ir teoriniai dirbtinio intelekto pagrindai
    - Cognitive science; Theory of mind

- Distributed artificial intelligence – išskirstytas dirbtinis intelektas
  - Multi-agent systems; Intelligent agents; Mobile agents; Cooperation and coordination
- Computer vision
  - Computer vision tasks
    - Biometrics; Scene understanding; Activity recognition and understanding; Video summarization; Visual content-based indexing and retrieval; Visual inspection; Vision for robotics; Scene anomaly detection
  - Image and video acquisition
    - Camera calibration; Epipolar geometry; Computational photography; Hyperspectral imaging; Motion capture; 3D imaging; Active vision
  - Computer vision representations
    - Image representations; Shape representations; Appearance and texture representations; Hierarchical representations
  - Computer vision problems
    - Interest point and salient region detections; Image segmentation; Video segmentation; Shape inference; Object detection; Object recognition; Object identification; Tracking; Reconstruction; Matching

Galima DI apibūdinti jo tikslu: „Dirbtinio intelekto tikslas yra pateikti intelektualaus elgesio kompiuterinius modelius, svarbiausia, sveiko proto samprotavimo.“<sup>2</sup>

### **1.3. Žodžio „intelektualus“ samprata dirbtiniame intelekte**

Mes laikomės požiūrio, kad intelektualus gali būti tik žmogus, o ne kompiuteris. Programų sistema, mašina ar kitas žmogaus sukurtas artefaktas yra ne intelektualus, bet „intelektualus“ (kabutėse), kitais žodžiais, intelektualizuotas. Pavyzdžiui, intelektualus yra žmogus, bet „intelektuali“ yra programų sistema, „intelektualus“ kompiuteris ir „intelektuali“ mašina. Tačiau DI literatūroje kalbant apie „intelektualias“ sistemas yra priimta kabutes praleisti.

Trumpai kalbant, DI nagrinėja, kaip kompiuterius intelektualizuoti. Dirbtinio intelekto disciplina nagrinėja klausimus: kas yra laikomas „intelektuali“? ką reiškia būti „dirbtinai intelektuali“ (*artificially intelligent*)? DI siekia sukurti ir suprasti intelektualias esybes. Ką reiškia intelektualus? Intelektualais yra laikomi šie gebėjimai: 1) suvokti (*perceive*); 2) suprasti (*understand*); 3) prognozuoti (*predict*); ir 4) manipuliuoti – atliliki veiksma (*manipulate*).

Intelekto požymis yra gebėjimas spręsti blogai suformuluotus uždavinius. DI tūliai, kaip sukurti mašinas, kurios gali matyti, judėti ir veikti. Taigi DI tyrimo objektas siejamas su kompiuteriais, tiksliau, kaip kompiuteris „mąsto“, kitais žodžiais, kaip programuoti, kad kompiuteris imituotų mąstymą.

Alanas Tiuringas (1950) suformulavo klausimą „ar gali mašina mąstyti?“ (*can machine think?*). Jis suformulavo testą, kuris vėliau pavadintas Tiuringo testu. Testas trumpai skamba taip. Žmogus A terminalu su klaviatūra bendrauja su dviem agentais B ir C, kurių vienas yra žmogus, o kitas mašina, bet A nežino, kuris yra kuris. Žmogaus A tikslas yra nustatyti, kuris

<sup>2</sup> „The aim of artificial intelligence is to provide a computational model of intelligent behavior, most importantly, commonsense reasoning“; žr. J. Pearl knygą „Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference“, 1988, p. 14.

agentas yra žmogus ir kuris mašina. Jeigu A nepavyksta atskirti, tai laikoma, kad mašina „mąsto“ (praeina Tiuringo testą).

Pavyzdžiu, Alice – tai programa, kuri Tiuringo testo atveju atstotų mašiną. Ji prieinama internete (žr. <http://www.alicebot.org/>).

Viena iš DI šakų yra *kognityvistika* (*cognitive science*). Kognityvistikos tyrimo objektas yra žmogus, tiksliau, kaip mąsto žmogus. Į kognityvistiką mūsų medžiagoje nesigilinama, nes gilinamasi į kitą DI paradigmą – kaip užprogramamuoti kompiuterį „mąstyti“.

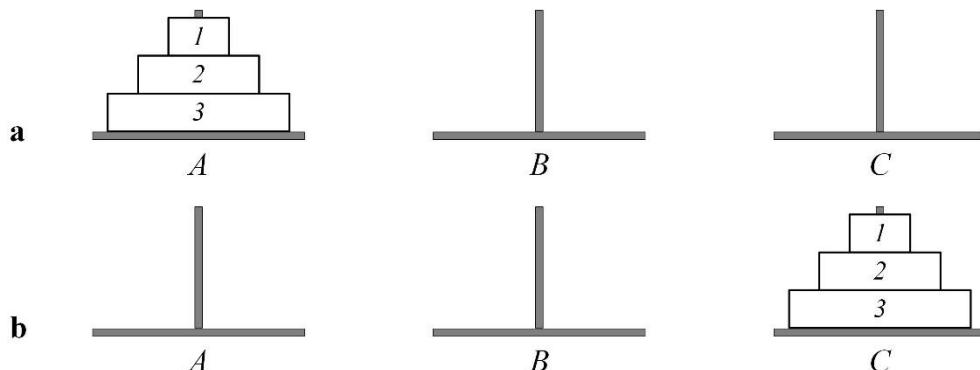
Reziumuojame: tik žmogui yra būdingas žmogiškasis intelektas. O kompiuteris pasižymi dirbtiniu intelektu. Mechaninių skaičiavimo mašinelių sukūrimas XIX a. ir aritmometrai (pvz. finansų apskaitoje XX a. plačiai naudotas „Feliks“) pademonstravo, kad sugebėjimas skaičiuoti nėra išskirtinis žmogiškojo intelekto požymis. Žmogui pavyko automatizuoti aritmetinius veiksmus. Kalkulatorius šiuo metu nėra laikomas DI artefaktu. Prieš 40-50 metų DI varomoji jėga buvo natūralios kalbos apdorojimas. Natūralios kalbos mašininio apdorojimo ir supratimo kriterijus yra loginis išvedimas (*entailment*).

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

#### 1.4. Hanojaus bokštai

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

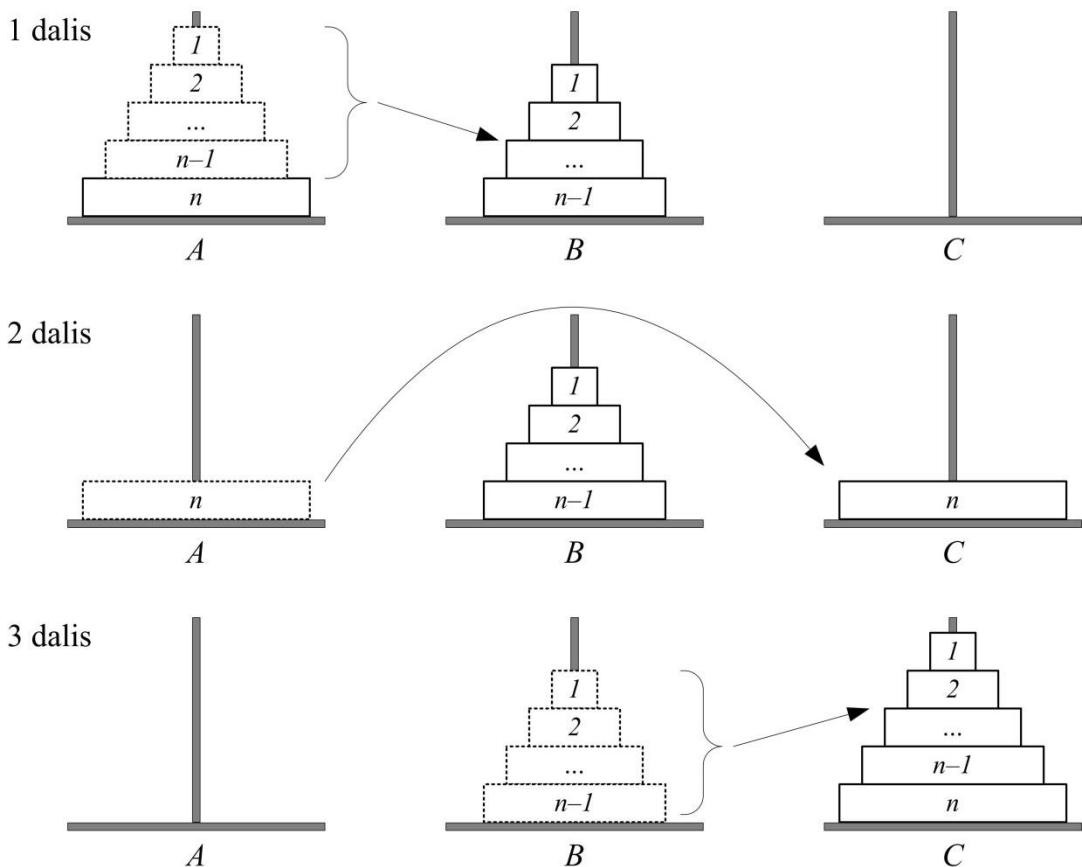
Uždavinio apibendrinimą žr. [http://en.wikipedia.org/wiki/Towers\\_of\\_Hanoi](http://en.wikipedia.org/wiki/Towers_of_Hanoi).



1.1 pav. a) Hanojaus bokštų uždavinio pradinė būsena  $A=(3,2,1)$ ,  $B=()$ ,  $C=()$ .  
b) Terminalinė būsena  $A=()$ ,  $B=()$ ,  $C=(3,2,1)$

Rasti diskų perkėlimo seką yra intelekto reikalaujantis uždavinys. Intelektas glūdi algoritme, kuris yra rekursinis ir yra trijų dalių seką:

- 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  $n=3$ :

Pradinė būsena  $A=(3,2,1)$ ,  $B=()$ ,  $C=()$

1. Diską 1 nuo A perkelti ant C.  $A=(3,2)$ ,  $B=()$ ,  $C=(1)$ .
2. Diską 2 nuo A perkelti ant B.  $A=(3)$ ,  $B=(2)$ ,  $C=(1)$ .
3. Diską 1 nuo C perkelti ant B.  $A=(3)$ ,  $B=(2,1)$ ,  $C=()$ .
4. Diską 3 nuo A perkelti ant C.  $A=()$ ,  $B=(2,1)$ ,  $C=(3)$ .
5. Diską 1 nuo B perkelti ant A.  $A=(1)$ ,  $B=(2)$ ,  $C=(3)$ .
6. Diską 2 nuo B perkelti ant C.  $A=(1)$ ,  $B=()$ ,  $C=(3,2)$ .
7. Diską 1 nuo A perkelti ant C.  $A=()$ ,  $B=()$ ,  $C=(3,2,1)$ .

Šios sekos radimas yra ir žmogiškajam intelektui, ir kompiuteriui įveikiamas uždavinys. 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.}

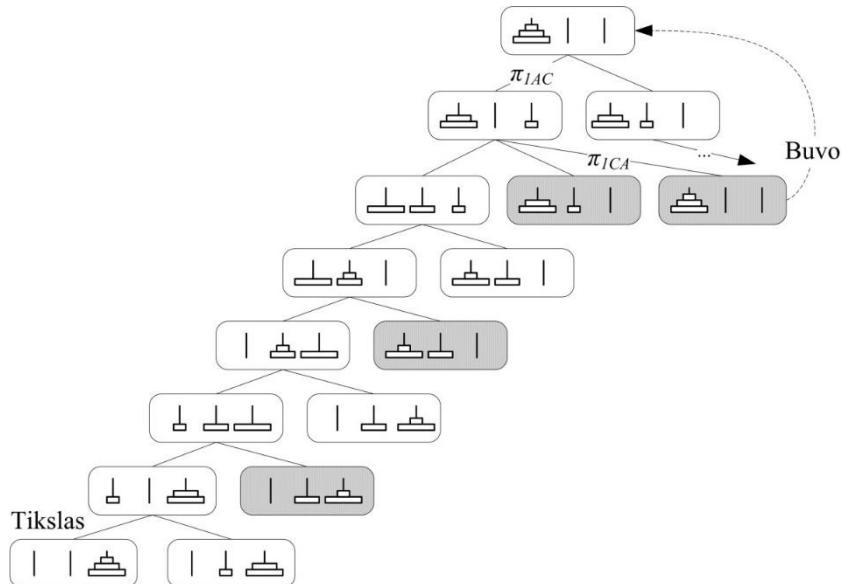
```

```

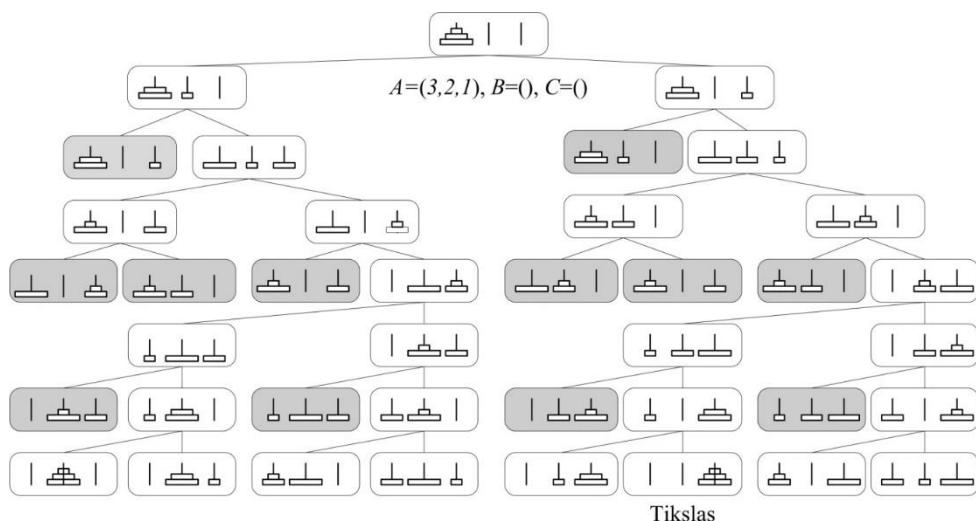
writeln('Diską ', n, ' nuo ', x, ' perkelti ant ', z); {2.}
hb(y, x, z, n-1); {3. Perkelti n-1 diską ant tikslo. }
end
end

```

Šios procedūros iškvietimas su  $n=3$ :  $hb('A', 'B', 'C', 3)$ . Bendru atveju perkėlimų skaičius yra eksponentinis –  $2^{n-1}$ . Tokia elegantiška rekursinė procedūra parašoma nedaugeliui DI uždavinių. Toliau aptariami paieškos algoritmai būsenų perėjimo grafuose (*state transition graphs*). 1.3 pav. pateiktas GRAPHSEARCH\_I\_GYLĮ algoritmo paieškos medis Hanojaus bokštų uždaviniui, kai  $n=3$ . GRAPHSEARCH nagrinėjamas toliau.



1.3 pav. Algoritmo GRAPHSEARCH\_I\_GYLĮ paieškos medis Hanojaus bokštų uždaviniui  $n=3$ . Pilkai pažymėtos būsenų viršūnės, kurios jau yra sąraše ATIDARYTA arba UŽDARYTA.



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

**1 pratimas. Iteracinis Hanojaus bokštų algoritmas.** Iteracinis Hanojaus bokštų algoritmas nėra taip žinomas kaip nagrinėtas rekursinis; žr. pvz., (Brachman, Levesque 2004, p. 133–134). Bokštai sustatomi ratu: A, B ir C. Disko perkėlimo taisyklės:

1. Jeigu  $n$  lyginis, tai perkelti ant artimiausio galimo bokšto **pagal** laikrodžio rodyklę.

Jeigu  $n$  nelyginis, tai perkelti ant artimiausio galimo bokšto **prieš** laikrodžio rodyklę.

2. Draudžiama tą patį diską kelti du kartus iš eilės.

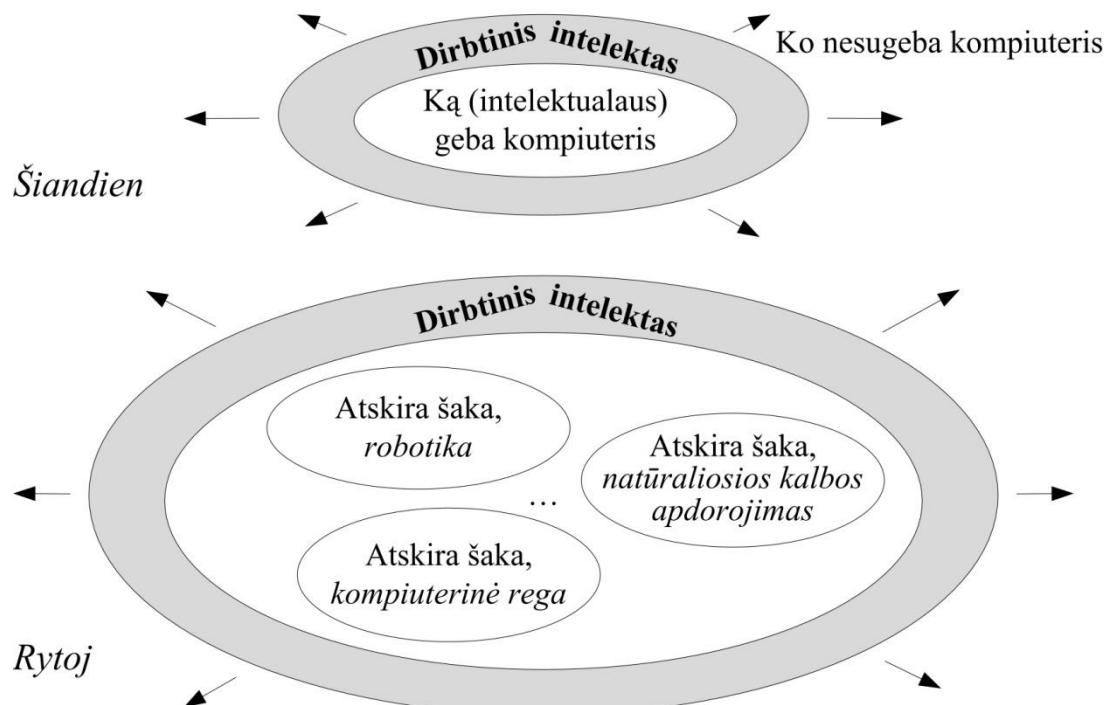
Parašykite programą.

**2 pratimas. Apie Hanojaus bokštų uždavinio sudėtingumą.** Tegu vienas éjimas trunka 1 sekundę. Kiek metų truks uždavinio sprendimas, kai  $n=64$ ?

**3 pratimas. Bezdžionės ir banano uždavinys.** Kambaryje yra bezdžionė, kėdė, lazda ir prie lubų prikabintas bananas. Bezdžionė gali numušti bananą tik pristūmusi kėdę po bananu, paémusi lazdą, užlipusi and kėdės ir pakelta ranka mosuodama lazda. Parašykite bezdžionės veiksmus paimti bananą ([https://en.wikipedia.org/wiki/Monkey\\_and\\_banana\\_problem](https://en.wikipedia.org/wiki/Monkey_and_banana_problem)).

## 1.5. Dirbtinio intelekto sampratos kitimas

Iškirtinos dvi DI savybės. Pirma, DI nagrinėja informatikos pažinimo ribas: ką tokio (intelektualaus) kompiuteris gali ir ko negali. Antra, dirbtinio intelekto samprata laikui bégant kinta. Kas šiandien priskiriamas DI, rytoj gali būti nebepriskiriamas DI (1.5 pav.). Šitaip formuoja atskiros šakos, pavyzdžiu, robotika, kompiuterinė rega ir kt.



1.5 pav. Kas šiandien priskiriamas DI, ateityje gali būti nebepriskiriamas DI.  
Šitaip formuoja atskiros šakos

## 1.6. Uždavinių sprendimo principai

Kompiuterizuojamų uždavinių išankstinis klasifikavimas turi prasmę. Žmogui susidūrus su nauju uždaviniu, kuris priskirtas tam tikrai klasei, sufleruoja sprendimo būdą. Uždavinio pavyzdys yra naftos būvimo prognozė (yra naftos ar nėra) pagal seismogramą (trimati žemės gelmių vaizdą, gautą iš prietaisų).

Uždavinių klasifikacija, kuri priimta kitose ne dirbtino intelekto dalykinėse srityse:

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

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

Nors aukščiau vadinami metodais, mes juos laikysime uždavinių sprendimo principais. Juos apima DI principai. Kuo metodas skiriasi nuo principio? Metodas taikomas tiksliai apibrėžtoje situacijoje. Jeigu nėra žinomas metodas, tai vadovaujamasi 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. Pavyzdžiui, principas yra „vyras suteikia pirmenybę moteriai“. Bet normos gali įtvirtinti ir specifines išlygas ar net prieštarauti viena kitai, pvz.:

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

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

Metodas = schema + procedūros

Metodika = metodas + rekomendacijos

Šios formulės nors ir supaprastina termino prasmę, bet atspindi esmę. Yra ir kitų termino „metodas“ apibūdinimų, pvz., metodas = notacija + proceso modelis. Pavyzdys yra UML + RUP, t. y. kompanijos Rational modeliavimo kalba Unified Modelling Language + procesas (Hesse, Verrijn-Stuart 2001, p. 1).

## 2. Dirbtinio intelekto sistema kaip produkcių sistema

Šiame skyriuje vadovaujamės (Nilsson 1982) 1.1 poskyriu.

Daugumoje dirbtinio intelekto sistemų galima išskirti tris esminius komponentus: duomenis, operacijas ir jų valdymą. Kitaip tariant, tokiose sistemoje galima išskirti tam tikrą duomenų struktūrą, kurią vadinsime *globalia duomenų baze*, bei su ja atliekamus iš anksto veiksmus apibrėžtus veiksmus. 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. Juo remiantis dalykinės srities žinios gali būti klasifikuojamos pagal tai, kur yra vaizduojamos. *Deklaratyviosiomis žiniomis* 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. Priklausomai nuo dalykinės srities, ji gali būti tiek įprasta sveikujių skaičių matrica, tiek sudėtinga duomenų bazių valdymo sistema. Produkciai iš produkcių aibės yra taikoma GDB ir po pritaikymo GDB yra pervedama iš vienos būsenos į kitą. Produkciai gali būti taikomi tik tada, jeigu GDB būsena tenkina tos produkcijos pradinę sąlygą. Kurią būtent produkcią iš produkcių aibės parinkti ir taikyti, sprendžia valdymo sistema. Po kiekvienos produkcijos taikymo, valdymo sistema patikrina, ar GDB būsena yra terminalinė. Jeigu GDB tenkina terminalinės būsenos sąlygą, tai algoritmas baigia darbą. Jeigu GDB nėra terminalinė, o valdymo sistema nebegali taikyti nei vienos produkcijos, tai laikoma, kad uždavinio sprendinys neegzistuoja. Produkcių sistemos algoritmas:

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

Procedūros rezultatas yra produkcių seka  $\langle \pi_{i1}, \pi_{i2}, \dots, \pi_{in} \rangle$ , kuri pradinę būseną pvereda į terminalinę. Produkcių sistemos elementais yra pradinė GDB būsena (visada viena) ir terminalinė GDB būsena (gali būti keletas). Pastarųjų aibė gali būti apibrėžiama predikatu, kurio reikšmė yra true jeigu būsena tenkina terminalinę sąlygą, ir false priešingu atveju.

Procedūra aukščiau yra nedeterminuota. Operatorius **select** išrenka produkciją nedeterminuotai. Išrinkimo algoritmas nėra nusakomas ir tinkamai pasirinkimas, pavyzdžiu, spėjimas. Visos produkcijos turi vienodą prioritetą ir nekviečia viena kitos. Išrenkama nebūtinai pirmoji produkcija iš aibės visų galimų taikyti produkcių. Operatoriuje

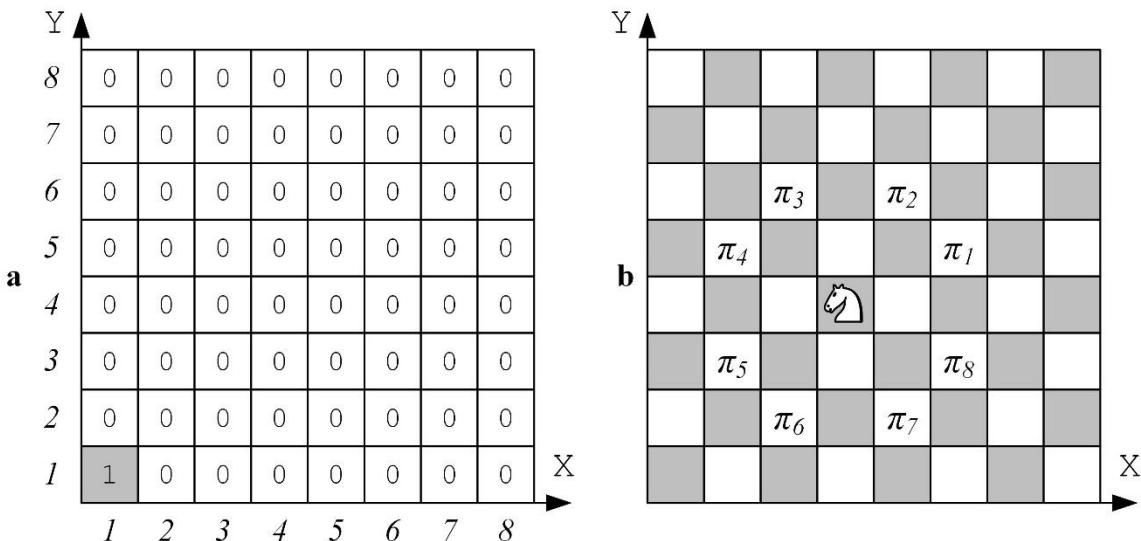
**select** ir glūdi intelektas – ką pasirinkti? Tokiu būdu procedūra PRODUCTION apibrėžia visą *determinuotą* algoritmų šeimą. PRODUCTION vykdo paiešką „i gylį“.

Procedūra PRODUCTION yra suprantama kaip valdymo sistemos *paradigma (paradigm)*. Ši žodži aiškina anglų kalbos žodynu Oxford English Dictionary (<http://dictionary.oed.com/>):

### paradigma (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 linksniuojuojamųjų kalbų gramatikose: šablonas arba lentelė, rodanti visas konkretaus daiktavardžio, veiksmažodžio ar būdvardžio linksnių formas, naudojama kaip modelis kitų žodžių asmenavimui ar linksniavimui. ...
4. Koncepcinis ar metodinis modelis, pagrindžiantis mokslo arba disciplinos teorijas ar panaudojimą tam tikru laiku; (taigi) bendrai priimta pasaulėžiūra (*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 nagrinėsime žirgo maršruto uždavinį: žirgu apeiti visą šachmatų lentą, langelis aplankant tik po vieną kartą. Plačiau apie šį uždavinį žr. [https://en.wikipedia.org/wiki/Knight%27s\\_tour](https://en.wikipedia.org/wiki/Knight%27s_tour). Tegu pradinė žirgo padėtis – langelis [1,1]. Šiame uždavinyje GDB vaizduojama dvimačiu  $8 \times 8$  sveikujų skaičių masyvu. Pradinėje būsenoje visi elementai turi reikšmę 0, o langeliui [1,1], priskiriama reikšmė 1 (2.1 a pav.). Produkcijų aibė  $\{\pi_1, \pi_2, \pi_3, \pi_4, \pi_5, \pi_6, \pi_7, \pi_8\}$  – visi 8 galimi žirgo éjimai (2.1 b pav.).



2.1 pav. a) Pradinė žirgo padėtis yra langelis [1,1]. b) Galimi žirgo éjimai – tai 8 produkcijos  $\{\pi_1, \pi_2, \pi_3, \pi_4, \pi_5, \pi_6, \pi_7, \pi_8\}$

Pritaikius produkciją GDB pervedama iš vienos būsenos į kitą. Naują žirgo poziciją rodo éjimo numeris langelyje, į kurį žengiama. Šiame uždavinyje terminalinė būsena yra išreiškiama teiginiu „visi masyvo langeliai yra užpildyti skaičiais nuo 1 iki 64, išdėstytais žirgo éjimais. Jeigu nepritaikoma jokia produkcija ir masyve yra laisvų langelių, tai apeiti nepavyko.“

Toliau pateikiama programa Paskalio kalba pagal tekstą iš V. Dagienės, G. Grigo ir K. Augučio knygos (1986). Uždavinio esmė nesikeičia, kai lentos dydis yra ne  $8 \times 8$ , o  $N \times N$ .

langelių. Paprasčiausias atvejis  $N=5$ . Įsitikinkite, kad  $4 \times 4$  lentoje kelias neegzistuoja. Kelias  $5 \times 5$  lentoje parodytas 2.2 a pav. Pirmoji aklavietė 21-me langelyje; žr. 2.2 b pav. Toliau nežengiamė, nes nepavyksta pritaikyti jokios produkcijos. Žirgi tenka grįžti ir bandyti kitus variantus. 2.2 b pav. parodyta, kad tik pirmieji septyni éjimai sutampa su sèkmingu keliu 2.3 a pav. Aštuntasis éjimas į tikslą neveda.

2.2 pav. a) Sèkmingas apéjimas  $5 \times 5$  lentoje.  
b) Išskirti pirmieji 7 éjimai, sutampantys su sèkmingu apéjimu. Aštuntasis éjimas neveda į tikslą

	Y	1	2	3	4	5	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		

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

```

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);
{Iéjimo parametrai L - éjimo numeris; X, Y - paskutiné žirgo padétis.}
{Iséjimo parametras (t.y. rezultatas) YRA.}
var
  K    : integer; {Produkcijos eilés numeris.}
  U, V : integer; {Nauja žirgo padétis.}
begin
  K := 0;
  repeat {Perrenkamos produkcijos iš 8 produkcijų aibės.}
    K := K + 1;
    U := X + CX[K]; V := Y + CY[K];
    {Patikrinama, ar duomenų bazė tenkina produkcijos taikymo sąlygą.}
    if LENTA[U, V] < 0 then
      begin
        LENTA[U, V] := L;
        if L = 1 then YRA := true;
      end;
  until K = 8;
end;

```

```

if (U >= 1) and (U <= N) and (V >= 1) and (V <= N)
then {Neišeinama už lentos krašto.}
    {Patikrinama, ar lanelis laisvas, t. y. ar žirgas tame nebuvo.}
    if LENTA[U, V] = 0
    then
        begin
            {Nauja žirgo pozicija pažymima éjimo numeriu.}
            LENTA[U, V] := L;
            {Patikrinama, ar lenta apeita.}
            if L < NN
            then
                begin {Jeigu lenta neapeita, tai bandoma daryti éjimą.}
                EITI(L+1, U, V, YRA);
                {Jei buvo pasirinktas nevedantis i sékmę éjimas...}
                {tai gríztama atgal, t. y. lanelis atlaisvinamas.}
                if not YRA then LENTA[U, V] := 0;
                end
            else YRA := true; {Kai L=NN}
        end;
    until YRA or (K = 8); {Sékmé arba perrinktos visos 8 produkcijos.}
end; {EITI}

begin {Pagrindiné programa (main program)}
    {1. Paruošiama globali duomenų bazé ir užpildoma produkcijų aibé.}
    INICIALIZUOTI; YRA := false;

    {2. Pažymima pradiné žirgo padétis: lanelis [1,1].}
    LENTA [1, 1] := 1;

    {3. Kviečiama procedūra sprendinio paieškai. Daryti antrą éjimą, }
    { stovint langelyje X=1 ir Y=1, ir gauti atsakymą YRA. }
    EITI(2, 1, 1, YRA);

    {4. Jeigu kelias 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('Kelias neegzistuoja.');
    end.

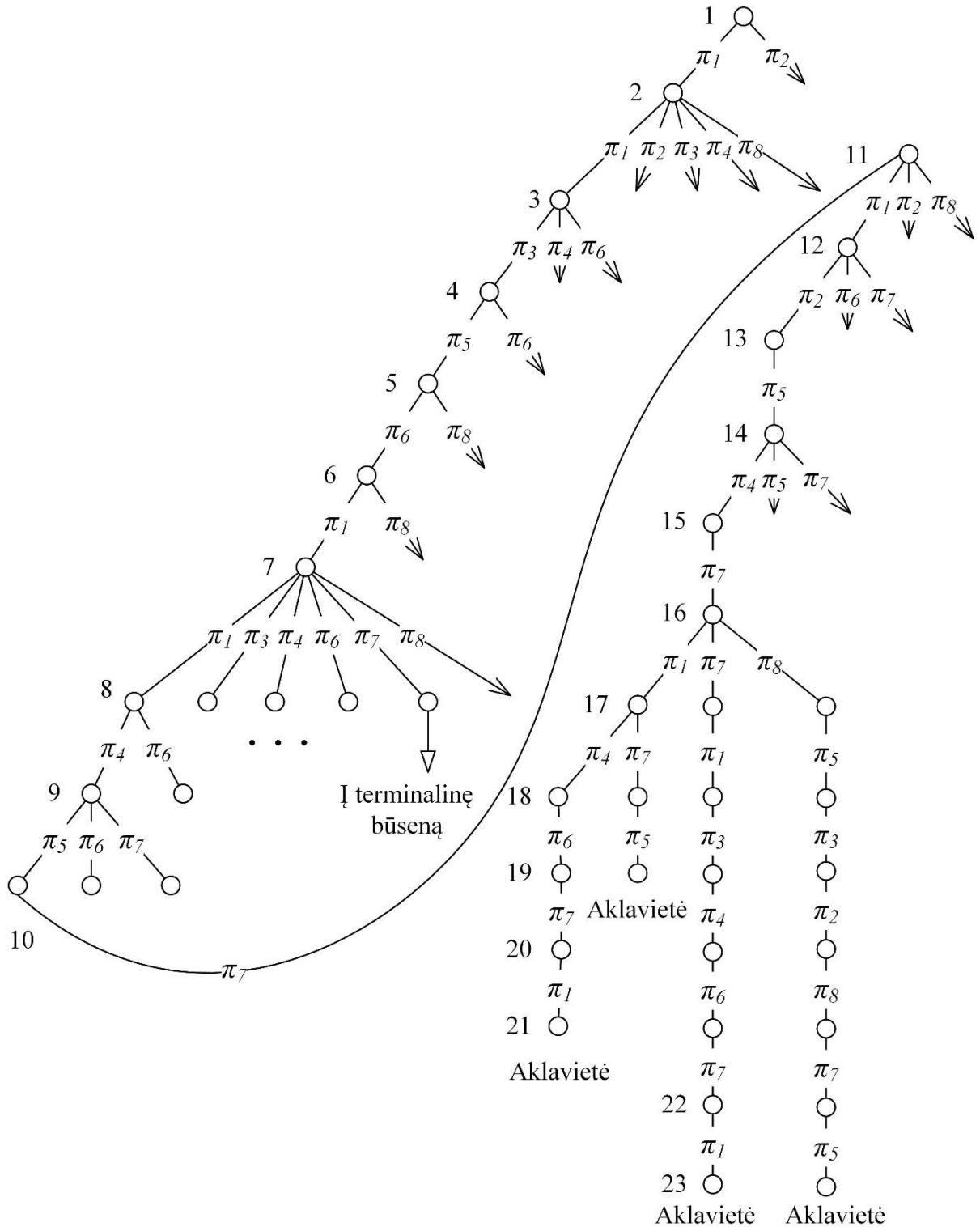
```

Atsakymas yra kelias iš 24 éjimų  $\langle \pi_{i1}, \pi_{i2}, \pi_{i3}, \dots, \pi_{i24} \rangle$ , kur  $\pi_{ij} \in \{\pi_1, \pi_2, \pi_3, \pi_4, \pi_5, \pi_6, \pi_7, \pi_8\}$ , t. y.  $i, j \in \{1, 2, \dots, 8\}$ . Žirgo uždavinys turi keletą sprendinių. Pavyzdžiui, yra simetriški pagrindinei įstrižainei apéjimai. Paaiškinsime, kas užprogramuota produkcijų sistemos trejetu:

1. LENTA, X ir Y – tai globali duomenų bazé;
2. CX ir CY – tai produkcijų aibé;
3. Paieškos į gylį su grížimas algoritmas EITI – tai valdymo sistema.

Žiniomis grindžiamų sistemų architektūroje pabrëžiami šie architektūros elementai:

1. Samprotavimas (*reasoning*). Tai valdymas (protavimas, mästymas). Išvedimo mašiną galima įsigyti. Ji nepriklauso nuo dalykinės srities;
2. Žinių bazé (*knowledge base*). Ją sudaro GDB (globali duomenų bazé) ir produkcijų aibé. Žinių bazés negalima įsigyti, nes ji priklauso nuo dalykinės srities. „Dievas yra detalėse, o ne teoremore“ – primena DI autoritetai.



2.3 pav. Paieškos medis iki pirmosios aklavietės. Rodomi ėjimus  $5 \times 5$  lentoje iki pasiekiamą pirmojo aklavietė, kai žirgas negali eiti toliau ir grįžta. Terminalinė lento būsena parodyta 2.2 b pav.

## 2.1. Žirgo maršruto programos testavimas

Papildykite programą, kad spausdintų paieškos medį 2.5 pav. kaip vykdymo protokolą:

1 DALIS. Duomenys

- 1) Lenta 5x5.
- 2) Pradinė žirgo padėtis X=1, Y=1. L=1.

2 DALIS. Vykdymas

- 1) R1. U=3, V=2. L=2. Laisva. LENTA[3,2]:=2.
- 2) -R1. U=5, V=3. L=3. Laisva. LENTA[5,3]:=3.
- 3) --R1. U=7, V=4. L=4. Už krašto.
- 4) --R2. U=6, V=5. L=4. Už krašto.
- 5) --R3. U=4, V=5. L=4. Laisva. LENTA[4,5]:=4.
- 6) ---R1. U=6, V=6. L=5. Už krašto.
- 7) ---R2. U=5, V=7. L=5. Už krašto.
- 8) ---R3. U=3, V=7. L=5. Už krašto.
- 9) ---R4. U=2, V=6. L=5. Už krašto.
- 10) ---R5. U=2, V=4. L=5. Laisva. LENTA[2,4]:=5.
- 11) ----R1. U=4, V=5. L=6. Siūlas.

Ir taip toliau iki aklavietės L=21 ir grįzimo vienu žingsniu atgal

- ...) -----R1. U=4, V=2. L=21. Laisva. LENTA[4,2]:=21.
- ...) -----R1. U=6, V=3. L=22. Už krašto.
- ...) -----R2. U=5, V=4. L=22. Siūlas.
- ...) -----R3. U=3, V=4. L=22. Siūlas.
- ...) -----R4. U=2, V=3. L=22. Siūlas.
- ...) -----R5. U=2, V=1. L=22. Siūlas.
- ...) -----R6. U=3, V=0. L=22. Už krašto.
- ...) -----R7. U=5, V=0. L=22. Už krašto.
- ...) -----R8. U=6, V=1. L=22. Už krašto. Backtrack.
- ...) -----R2. U=3, V=3. L=21. Siūlas.
- ...) -----R3. U=1, V=3. L=21. Siūlas.
- ...) -----R4. U=0, V=2. L=21. Už krašto.
- ...) -----R5. U=0, V=0. L=21. Už krašto.
- ...) -----R6. U=1, V=-1. L=21. Už krašto.
- ...) -----R7. U=3, V=-1. L=21. Už krašto.
- ...) -----R8. U=4, V=0. L=21. Už krašto. Backtrack.
- ...) -----R8. U=3, V=2. L=20. Siūlas. Backtrack
- ...) -----R7. U=3, V=3. L=19. Siūlas.
- ...) -----R8. U=4, V=4. L=19. Siūlas. Backtrack.
- ...) -----R5. U=2, V=3. L=18. Siūlas.
- ...) -----R6. U=3, V=2. L=18. Siūlas.
- ...) -----R7. U=5, V=2. L=18. Laisva. LENTA[5,2]:=18.
- ...) -----R1. U=7, V=3. L=19. Už krašto.

Ir taip toliau.

- 70611) -----R1. U=5, V=5. L=23. Laisva. LENTA[5,5]:=23.
- 70612) -----R1. U=7, V=6. L=24. Už krašto.
- 70613) -----R2. U=6, V=7. L=24. Už krašto.
- 70614) -----R3. U=4, V=7. L=24. Už krašto.
- 70615) -----R4. U=3, V=6. L=24. Už krašto.
- 70616) -----R5. U=3, V=4. L=24. Siūlas.
- 70617) -----R6. U=4, V=3. L=24. Laisva. LENTA[4,3]:=24.
- 70618) -----R1. U=6, V=4. L=25. Už krašto.
- 70619) -----R2. U=5, V=5. L=25. Siūlas.
- 70620) -----R3. U=3, V=5. L=25. Siūlas.
- 70621) -----R4. U=2, V=4. L=25. Siūlas.
- 70622) -----R5. U=2, V=2. L=25. Siūlas.
- 70623) -----R6. U=3, V=1. L=25. Siūlas.
- 70624) -----R7. U=5, V=1. L=25. Laisva. LENTA[5,1]:=25.

3 DALIS. Rezultatai

- 1) Apéjimas rastas
- 2) Apéjimas pseudografika

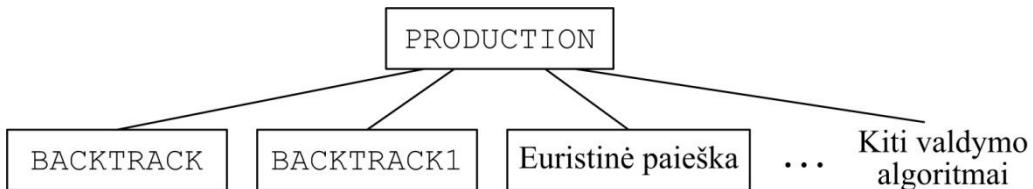
Y, V ^  
5 | 21 16 11 4 23  
4 | 10 5 22 17 12  
3 | 15 20 7 24 3  
2 | 6 9 2 13 18  
1 | 1 14 19 8 25  
-----> X, U

1 2 3 4 5

### 3. Valdymas su grįžimais ir procedūra BACKTRACK

Terminas „valdymas su grįžimais“ žymi vieną iš keleto valdymo sistemos algoritmų, tiksliau, vieną 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. Valdymo su grįžimais požymis yra suprantamas kaip DI principas.

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



3.1 pav. Procedūra BACKTRACK valdymo algoritmų kontekste

Valdymą su grįžimais paaiškinsime valdovų 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 valdovė 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  $\pi_k$  – statyti einamają valdovę į  $k$ -tajį stulpelį. Pastaruoju atveju vadovaujamasi 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 sustatymas yra 8 produkcijų seka  $\langle \pi_{k1}, \pi_{k2}, \dots, \pi_{k8} \rangle$ , 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 grįžimais* algoritmas, kuris sutrumpintai vadinamas „valdymu su grįžimais“ (*backtracking*).

Valdymo su grįžimais 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 grįžimais vaizduoja procedūra BACKTRACK (Nilsson 1982, 2.1 poskyris) žemaiu.

Procedūra BACKTRACK kaip rezultatą grąžina produkcijų sąrašą.

```
recursive procedure BACKTRACK (DATA) {return list of rules}
{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 produkcių sąrašą, žymimą NIL, ir sėkmingai baigia darbą.

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

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

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

**5 žingsnis.** Iš produkcių aibės (sarašo) RULES yra paimama pirmoji produkcia. 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 (*LISP Processing*). Pavyzdžiui, FIRST( $\langle \pi_1, \pi_2, \pi_3, \pi_4, \pi_8 \rangle$ ) =  $\pi_1$ .

**6 žingsnis.** Iš produkcių aibės RULES yra pašalinama pirmoji produkcia. 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.** Produkcia R globalių 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 nesékmės požymis FAIL, tai reiškia, kad 5 žingsnyje paimta produkcia R neveda į terminalinę būseną. Tokiu atveju grįztama į 4 žingsnį. Bus imama kita produkcia.

**10 žingsnis.** Produkcių 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 bazę yra pervedama iš pradinės būsenos į terminalinę būseną. Pavyzdžiui,  $\pi_{j3}(\pi_{j2}(\pi_{j1}(DATA)))$  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 algebros tipo aibė. Argumentų aibė (dar vadinama *apibrėžimo sritimi*) nėra svarbi.

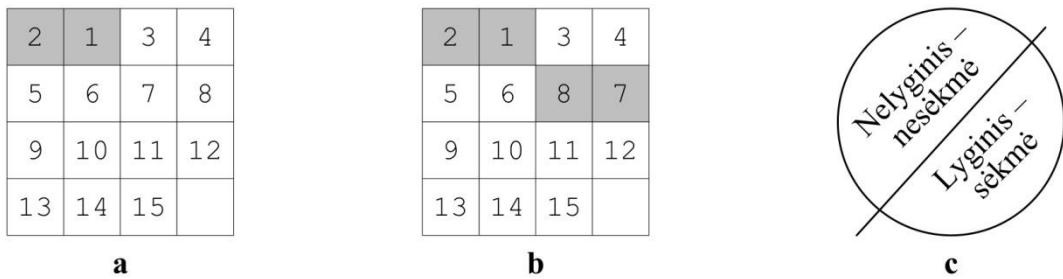
Predikatą DEADEND, kuris naudojamas procedūros BACKTRACK 2 žingsnyje, iliustruojame pavyzdžiu. Paimkime „15 kauliukų“ žaidimą (15-puzzle). Žaidimo esmė yra tokia: kauliukai sudedami į  $4 \times 4$  dėžutę atsitiktine tvarka. Viena vieta lieka laisva. Kauliukas gali būti stumiamas keturiomis kryptimis: 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 lavelio stūmimas, bet priešinga kryptimi. Tokiu būdu yra keturios produkcijos: tuščias lavelis gali būti stumiamas kairėn, aukštyn, dešinėn arba žemyn.

Pradinė būsena realiai 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 lyginant su terminaline būseną.



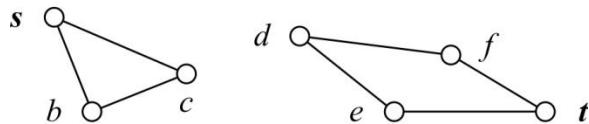
3.3 pav. a) Pradinė būsena 15 kauliukų žaidime, kai kauliukai 1 ir 2 sukeisti vietomis. Perstatu skaičius yra 1. Kadangi tai nelyginis skaičius, tai ši būsena nėra sėkminga. b) Pradinė būsena su 2 perstatomis, kai kauliukai 1 ir 2 bei 7 ir 8 yra sukeisti vietomis. Kadangi tai lyginis skaičius, tai ši būsena sėkminga. Tokia būsena sėkminga. c) Pradinį būsenų aibės suskaidymas į nesėkmingas (su nelyginiu perstatu skaičiumi) ir sėkmingas (su lyginiu)

Esminė 15 kauliukų uždavinio savybė yra tokia. Jeigu perstatu skaičius pradinėje būsenoje yra **nelyginis**, tai tokia būsena niekada nepasitaiko kelyje į terminalinę būseną. Jeigu perstatu

skaičius pradinėje būsenoje yra **lyginis**, tai tokia būsena yra sėkminga (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 (*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 tokią gilią 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. Pavyzdžiuui, 3.4 pav. pateiktame grafe keliautojas, vien tik pažvelgęs į GDB gali pasakyti, kad iš *s* į *t* nukeliauti neįmanoma, kadangi šios viršūnės priklauso nesusijusiems pografiams.



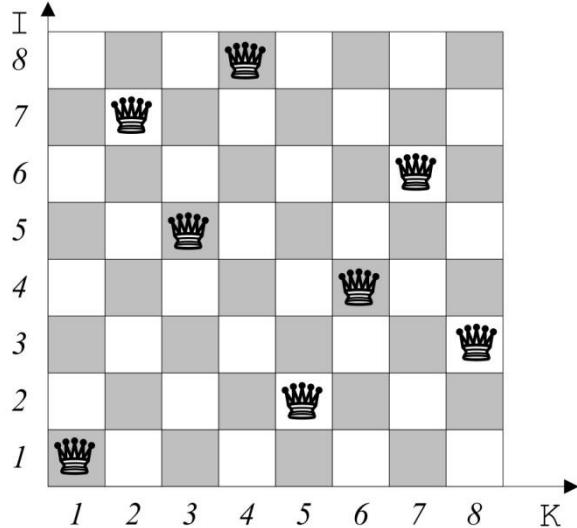
3.4 pav. Kelias grafe iš *s* į *t* neegzistuoja, nes šios viršūnės priklauso nesusijusiems pografiams. Galima užprogramuoti: DEADEND (DATA) =**true**, jeigu pradinė *s* ir terminalinė *t* yra nesusijusiose pografiuose

### **Pratimai**

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

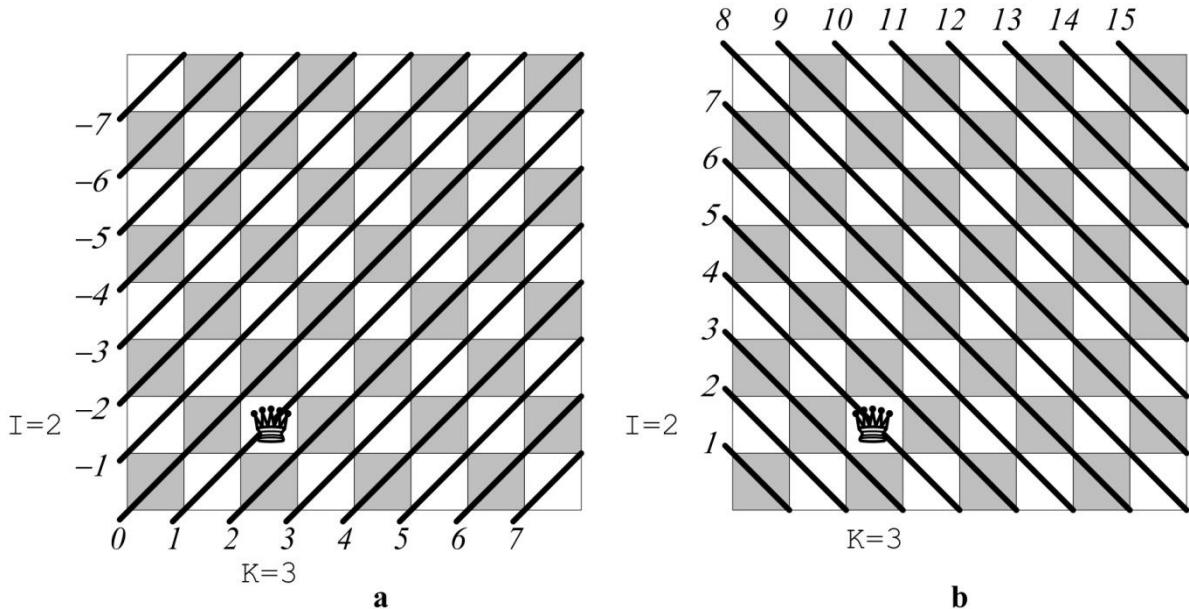
#### 4. Valdovių išdėstymas šachmatų lentoje

Uždavinys yra aštuonias valdoves šachmatų lentoje 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 numeris. Vienas išdėstymų yra 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, kad nekirstų viena kitos

Spredinys koduoojamas skaičių nuo 1 iki 8 perstatą. Galimas pilno perrinkimo (*brute force*) algoritmas – 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 lentoje perstatą yra  $N!$ . Kai  $N$  artėja į begalybę, tai faktorialas yra aproksimuojamas eksponentine funkcija:  $N! = 2^{\alpha N}$ , kur  $\alpha$  kažkoks realus skaičius, priklausantis nuo  $N$ .



4.2 pav. a) Kylančios įstrižainės nuo  $-7$  iki  $7$ ; b) besileidžiančios nuo  $1$  iki  $15$

Taikysime procedūrą BACKTRACK. Pirmają valdovę statysime į pirmąjį eilutę, antrają valdovę – į antrają eilutę ir taip toliau iki aštuntosios. Produkciai  $\pi_k$  vaizduoja eilinės valdovės

statymą į  $k$ -tajį stulpelį. Pavyzdžiu, išdėstymą 4.1 pav. atitinka kelias iš pradinės būsenos, kai lenta tuščia:  $\text{PATH} = \langle \pi_1, \pi_5, \pi_8, \pi_6, \pi_3, \pi_7, \pi_2, \pi_4 \rangle$ . Kelio  $I$ -toji produkcia yra  $\pi_{X[I]}$ . Kitais žodžiais,  $\text{PATH}[I] = \pi_{X[I]}$ , kur  $I = 1, 2, \dots, 8$ .

Toliau pateikiamas programa Paskalio kalba pagal knygą (Dagienė, Grigas, Augutis, 1986); žr. taip pat vikipedijoje ([http://en.wikipedia.org/wiki/Eight\\_queens\\_puzzle](http://en.wikipedia.org/wiki/Eight_queens_puzzle)). Uždavinio ir programos esmė nesikeičia, kai lentos dydis yra ne  $8 \times 8$ , o  $N \times N$  langelių.

Aiškinama kylančių ir besileidžiančių įstrižainių sąvokomis. Tikrinama, ar valdovės nekerta per įstrižainę. 4.2 a pav. rodo kylančias įstrižaines, sunumeruotas nuo -7 iki 7 (bendru atveju nuo  $-(N-1)$  iki  $(N-1)$ ). O 4.2 b pav. rodo besileidžiančias įstrižaines, sunumeruotas nuo 1 iki 15 (bendru atveju nuo 1 iki  $2 \cdot N - 1$ ). Jeigu kintamieji  $I=2$  ir  $K=3$ , tai vertikalė yra  $K=3$ , kylančioji įstrižainė  $K-I = 3-2 = 1$ , ir besileidžiančioji  $I+K-1 = 2+3-1 = 4$ .

```

program VALDOVÉS;
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 įstrižainės}
  LEID   : array [1..N2M1] of boolean;     {Besileidžiančios įstrižainės}
  YRA    : boolean;          I, J : integer; BANDSK : longint;
procedure TRY (I : integer; var YRA : boolean);
{Išėjimas 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 įstrižainės 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 keliai 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 {Pagrindiné 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('Bandymu skaičius: ', BANDSK);
      end
    else writeln('Spredinys neegzistuoja');
end.

```

## 5. Euristika

Praeitame skyriuje buvo aprašomas pilnas produkcijų perrinkimas. Siekiant jo išvengti galima pasiūlyti tokią taisykłę valdovei statyti: pasirinkti tą langelį, kuriame valdovė kerta mažiausiai per ilgesnį iš dviejų įstrižainių.

*Euristika* – tai praktikos padiktuota pasirinkimo taisykla (*rule of thumb*), skirta rasti sprendinį be varginančios paieškos (paprastai tai perrinkimas). Būdvardis **euristinis** (*heuristic*) apibūdina uždavinio sprendimo būdą, kuris pagerina sprendimo vidutinę charakteristiką, bet nebūtinai blogiausio atvejo charakteristiką (Russell, Norvig 2003, p. 94). Euristikos terminas kildinamas iš graikiško žodžio „heuriskein“ (atrasti). Legenda byloja, kad senovės graikų filosofas Archimedes sušuko „eureka“ (atrada), kai suvokė apie jégą, kuri išstumia kietą kūną, panardintą į skystį. Dabar ši jégą vadinama *Archimedeo jéga*.

Anglų kalboje „heuristic“ vartojamas kaip būdvardis ir daiktavardis. Žodžio „euristika“ reikšmė gali būti aiškinima remiantis anglų kalbos žodynu *Oxford English Dictionary* (žr. <http://www.oed.com/>):

### **euristinis** (*heuristic*)

- a. Skirtas išsiaiškinti arba atrasti (Serving to find out or discover) ...
- c. *kompiuteriai* ... Su „euristinio“ programavimo procedūra kompiuteris atlieka paiešką tarp galimų sprendinių kiekvienoje programos stadioje, įvertina šiai stadijai „gerą“ sprendimą ir pereina prie kitos stadijos. Iš esmės euristinis programavimas yra panašus į uždavinių sprendimą bandymų ir klaidų būdais, kuriais mes naudojamės kasdienybėje (T. W. McRae 1964).

### **euristika** (*heuristic*)

- b. Euristinis procesas arba metodas mėginant uždavinio sprendinį; taisykla arba informacijos vienetas naudojamas tokiam procese (A. Newell et al. 1957). ... Procesas, kuriuo galimas, bet negarantuojamas uždavinio išsprendimas. Trumpai sakant, „euristika“ yra „euristinis procesas“ sinonimas.

Dirbtinis intelektas ankstyvoje vystymosi fazėje dar buvo vadinamas *euristiniu programavimu* (*heuristic programming*), kaip alternatyva neinformuotoms (*brute force*) perrinkimo procedūroms, tokioms kaip paieška „i gylį“ arba „i plotį“.

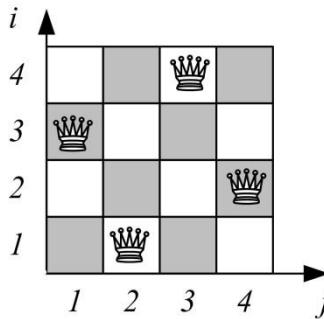
Euristika gali būti grindžiama ir ankstesniu patyrimu. Jeigu panašus uždavinys buvo sprestas ankčiau arba net nesėkmingai, tai patirtis gali padėti. Pavyzdžiu, vairuotojas atkreipia dėmesį į blogą automobilio darbą – trūkčioja variklis. Vairuotojas iš karto nesuvokia gedimo priežasties. Tačiau automobilį jis vairuoja daug metų ir prisimena panašią situaciją anksčiau. Tą kartą remontuotojas nustatė jungiamojo vamzdelio sandarumo pažeidimą, dėl kurio į variklį patekdavo oro, ir nurodė gedimo vietą. Taigi dabar vairuotojas sustojo ir rado, kad tas pats vamzdelis vėl nesandarus.

Euristika skirta sprendinio paieškai, bet be garantijos. Kadangi neatliekama gili analizė, tai gali būti siūlomas ir neteisingas sprendinys. Euristika siūlo priimtiną, o ne geriausią sprendinį, ir tai gali būti priimtina šiaisiai atvejais:

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

## 5.1. Euristika valdovių uždavinyje

Euristika toliau iliustruojama valdovių išdėstymo uždaviniu (žr. 4 skyrių). Euristika sumažins perrinkimą. Paprastumo dėlei paimkime  $4 \times 4$  lentą. Joje valdoves išdėstyti įmanoma:



5.1 pav. Vienas iš galimų išdėstymų  $4 \times 4$  lentoje

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

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

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

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

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

$$\text{diag}(1,1)=\max(4,1)=4$$

Analogiškai apskaičiuojame ilgius įstrižainių, einančių per kitus pirmosios eilutės langelius:

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

Remdamiesi šiais ilgiais sutvarkome:

$$\text{diag}(1,2) \leq \text{diag}(1,3) \leq \text{diag}(1,1) \leq \text{diag}(1,4)$$

Taigi vadovaudamiesi euristika sutvarkome produkcijas valdovės statymui pirmojoje eilutėje:

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

Analogiškai apskaičiuojame ilgius įstrižainių, einančių per antrosios eilutės langelius:

$$\begin{aligned} \text{diag}(2,1) &= \max(3,1)=3; \text{diag}(2,2)=\max(4,3)=4; \text{diag}(2,3)=\max(3,4)=4; \\ \text{diag}(2,4) &= \max(2,3)=3 \end{aligned}$$

Vadovaudamiesi euristika sutvarkome produkcijas valdovės statymui antrojoje eilutėje:

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

Analogiškai apskaičiuojame ilgius įstrižainių, einančių per trečiosios eilutės langelius:

$$\begin{aligned} \text{diag}(3,1) &= \max(2,3)=3; \text{diag}(3,2)=\max(3,4)=4; \text{diag}(3,3)=\max(4,3)=4 \\ \text{diag}(3,4) &= \max(3,2)=3 \end{aligned}$$

Vadovaudamiesi euristika sutvarkome produkcijas valdovės statymui trečiojoje eilutėje:

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

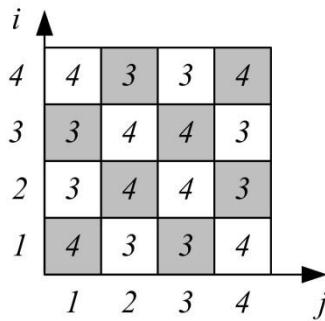
Analogiškai apskaičiuojame ilgius įstrižainių, einančių per ketvirtosios eilutės langelius:

$$\begin{aligned} diag(4,1) &= \max(1,4) = 4, \\ diag(4,2) &= \max(2,3) = 3, \\ diag(4,3) &= \max(3,2) = 3 \\ diag(4,4) &= \max(4,1) = 4 \end{aligned}$$

Vadovaudamiesi euristika sutvarkome produkcijas valdovės statymui ketvirtojoje eilutė:

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

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  $4 \times 4$  lentos langeliams

Reziumuojame visų keturių eilucių produkcijų sutvarkymą:

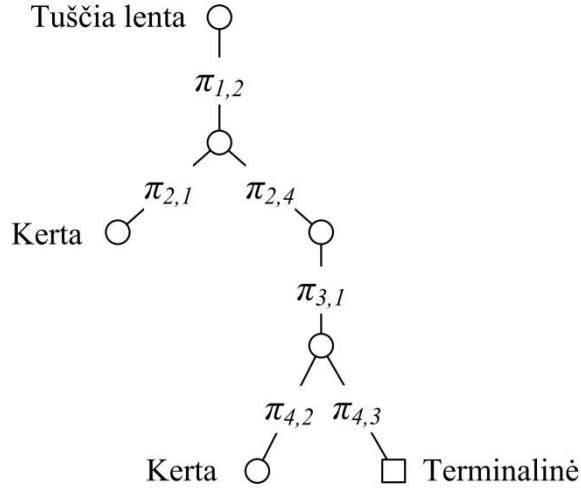
$$\begin{aligned} \pi_{1,2} \leq \pi_{1,3} \leq \pi_{1,1} \leq \pi_{1,4} & \quad - \text{pirma eilutė} \\ \pi_{2,1} \leq \pi_{2,4} \leq \pi_{2,2} \leq \pi_{2,3} & \quad - \text{antra eilutė} \\ \pi_{3,1} \leq \pi_{3,4} \leq \pi_{3,2} \leq \pi_{3,3} & \quad - \text{trečia eilutė} \\ \pi_{4,2} \leq \pi_{4,3} \leq \pi_{4,1} \leq \pi_{4,4} & \quad - \text{ketvirta eilutė} \end{aligned}$$

### 5.3 lentelė. Produkcijų sutvarkymas pagal euristiką valdovėms $4 \times 4$ lentoje

Remiantis šiuo sutvarkymu paieška vyksta kaip parodyta 5.4 pav.

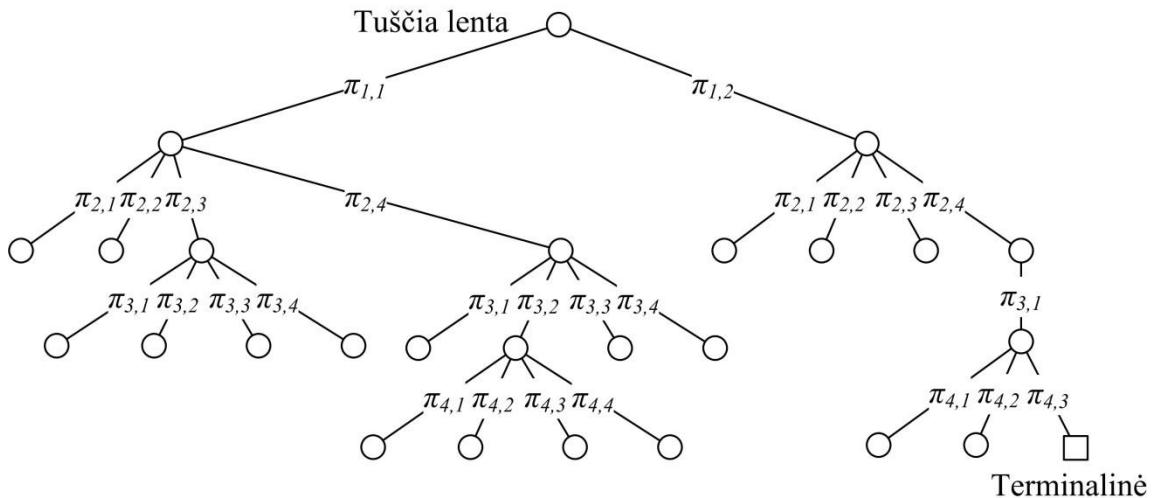
Algoritmo sudėtingumo matas yra bandymų skaičius. Medžio atveju tai briaunų skaičius.

Paaškinsime medž 5.4 pav. ir valdovių statymą vadovaujantis euristiniu produkcijų sutvarkymu. Pirmoji valdovė statoma į langelį  $[i=1, j=2]$ ), nes  $\pi_{1,2}$  geriausia pirmoje eilutėje 5.3 lentelėje. Analogiškai antrają valdovę bandome statyti į langelį  $[2,1]$ , nes  $\pi_{2,1}$  pradeda antrą eilutę. 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 antros eilutės produkcija. Trečioji valdovė statoma į langelį  $[3,1]$ , nes  $\pi_{3,1}$  geriausia trečioje eilutė. Ketvirtą valdovę bandome statyti į langelį  $[4,2]$ , nes  $\pi_{4,2}$  pradeda ketvirtą eilutę. Tačiau čia ją kertą pirmoji ir trečioji valdovės. Todėl ketvirtą valdovę statome į langelį  $[4,3]$ , nes  $\pi_{4,3}$  yra sekanti pagal sutvarkymą ketvirtose eilutėje. Čia terminalinė būsena, nes valdovės nekerta viena kitos. Bandymų skaičius, kaip matyti 5.4 pav., yra 6.



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

Be euristikos, t. y. sutvarkius produkcijas eilutėje  $i$  jų didėjimo tvarka ( $\pi_{i,1} \leq \pi_{i,2} \leq \pi_{i,3} \leq \pi_{i,4}$ ), tai paieškos medis būtų kaip parodyta 5.5 pav.



5.5 pav. Paieškos medis valdovų išdėstymui  $4 \times 4$  lentoje be jokios euristikos.  
Briaunų skaičius – bandymų skaičius – yra 26

Pirmaoji valdovė statoma į langelį ( $i=1, j=1$ ) 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 efektyvumas matuojamas briaunų skaičiumi paieškos medyje. Palyginę paieškos medžius 5.4 pav. ir 5.5 pav. matome, kad euristinė paieška yra geresnė, nes paieškos medis 5.4 pav. turi 6 briaunas, o 5.5 pav. – 26 briaunas.

$8 \times 8$  lentoje nėra taip paprasta. Produkcių sutvarkymas parodytas 5.6 pav., kur langeliams priskirti euristiniai įvertinimai:  $diag(1,1)=8$  ir t.t.

<i>i</i>	1	2	3	4	5	6	7	8
8	8	7	6	5	5	6	7	8
7	7	8	7	6	6	7	8	7
6	6	7	8	7	7	8	7	6
5	5	6	7	8	8	7	6	5
4	5	6	7	8	8	7	6	5
3	6	7	8	7	7	8	7	6
2	7	8	7	6	6	7	8	7
1	8	7	6	5	5	6	7	8

5.6 pav.  $8 \times 8$  lentoje funkcija  $diag(i,j)$  – ilgis ilgesnės iš dviejų įstrižainių, einančių per langelį  $[i,j]$

$8 \times 8$  lentoje pirmoji valdovė statoma į  $[1,4]$ . Antroji valdovė bandoma statyti antroje eilutėje į ketvirtą stulpelį, tačiau čia ją kerta. Tada bandoma į penktą – ir vėl kerta. Tada į pirmą – jis nekertamas. Ir t.t. Deja, septintoje eilutėje nepasiseka, tenka grįžti ir ieškoti toliau.

## 5.2. Dar viena euristika: valdovę statyti žirgo ējimu

Toliau nagrinėjama euristika „žirgo ējimas“.  $N \times N$  lentoje valdovė bandoma statyti šachmatų žirgo ējimu: du langeliai į kairę ir vienas į viršų prieš tai pastatytos valdovės atžvilgiu (5.7 pav.).  $8 \times 8$  lentoje pasiseka net keturi tokie ējimai.

<i>i</i>	1	2	3	4	5	6	7	8
8								
7								
6								
5								
4							👑	
3					👑			
2			👑					
1	👑							

5.7 pav. Valdovės statymas žirgo ējimu: du langeliai į kairę ir vienas į viršų.  $8 \times 8$  lentoje pasiseka padaryti net keturis tokius ejimus iš eilės

Dar pastebime, kad valdovių išsidėstymas žirgo éjimu pasitaiko ir atsakyme su procedūra BACKTRACK bei programa VALDOVÈS; žr. 4.1 pav. septintoje ir aštuntoje eilutëse. Begalinéje lentoje valdoves galima statyti žirgo éjimu. Mums gi aktualu tik baigtinës lentos, kai reikia pasirinkti pasiekus lento kraštą.

Šią euristiką 2005 m. pradéjo tirti Justas Arasimavičius bûdamas studentu, o pratësè Edgaras Abromaitis 2008 m. Manome, kad euristinis perrinkimas yra tiesinis, kai  $N$  auga, su išimtimis  $N=6 \cdot n+2$  and  $N=6 \cdot n+3$ , kai yra grëžimai atsiranda. Tai matyti lenteléje poskyrio pabaigoje. Tokio gero rezultato nesitikëjome.

Žirgo éjimų euristiką sudaro trys produkcijos:  $\pi_1$ ,  $\pi_z$  ir  $\pi_k$ .  $\pi_1$  yra pačios pirmosios valdovës statymo produkcija.  $N \times N$  lentoje pirmoji valdovë statoma į pirmos eilutës antrą langelį ( $[i=1, j=2]$ ) arba ketvirtą ( $[i=1, j=4]$ ). I ketvirtą langelį statome, jeigu  $N=9, 15, 21, \dots$ , t. y.  $N=6n+3$ , kur  $n = 1, 2, 3, \dots$ . Kitais atvejais statome į antrą langelį:

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

Šios pozicijos pasirinktos neatsitiktinai, tai tyrimo rezultatas pastebéjus mažesnį perrinkimą (žr. 5.23 lentelę). Tai euristinës žinios.

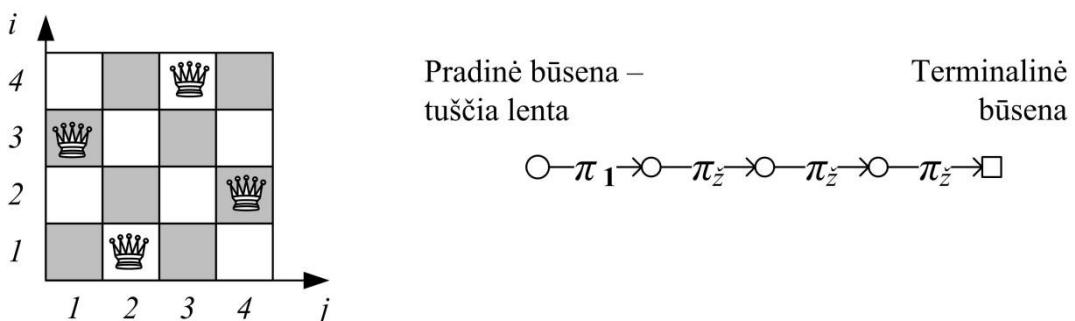
$\pi_z$  – esminë euristikos produkcija – žirgo éjimas. Tegu paskutiné valdovë pastatyta langelyje  $[i, j]$ . Tada kitą valdovę statome į  $[i+1, j+2]$ , jeigu toks lanelis priklauso lentai, ir į langelį pirmame stulpelyje  $[i+1, 1]$ , jeigu nepriklause (t. y.  $j+2 > N$ ):

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

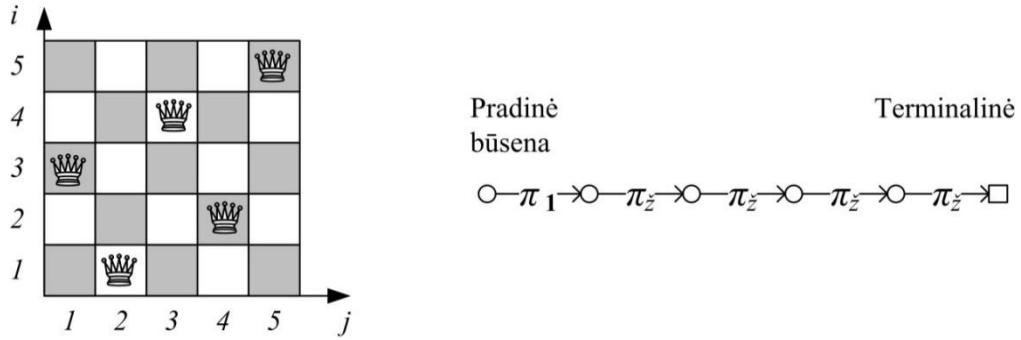
$\pi_k$  – produkcija, kuri taikoma, kai valdovës negalima pastatyti pagal  $\pi_z$  (valdovë yra kertama).  $\pi_k$  rezultatas yra kitas lanelis greta. Jeigu esame pozicijoje  $[i, j]$ , tai kitas lanelis yra  $[i, j+1]$ , jeigu toks lanelis priklauso lentai, arba pirmame stulpelyje  $[i, 1]$ , jeigu nepriklause. Sutrumpintai galima užrašyti, kad statoma į  $[i, (j \bmod N) + 1]$ . Ši produkcija taikoma, kol perrenkami visi vienos eilutës langeliai, taigi ne daugiau kaip  $N$  kartų.

$$\pi_k = \text{eilinę valdovę statyti į } [i, j+1], \text{ bet cikliškai, t. y. permetant į } [i, j=1], \text{ jeigu } j+1 > N \text{ (t. y. išeinama už krašto).}$$

Galima sakyti, kad eilinę valdovę statyti į langelį  $[i, (j \bmod N) + 1]$ , kur  $[i, j]$  yra ankstesnis bandymas toje pačioje eilutëje.



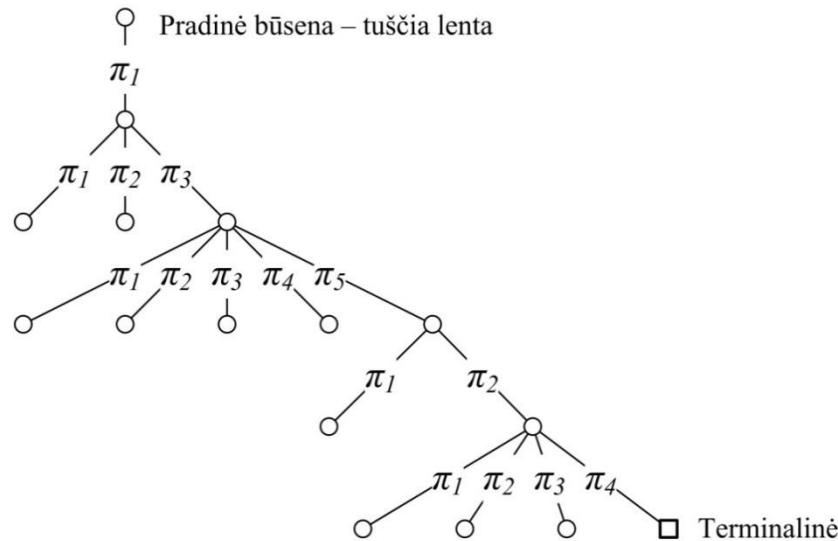
5.8 pav. Valdovių išdėstymas ir paieškos medis  $4 \times 4$  lentoje žirgo éjimu



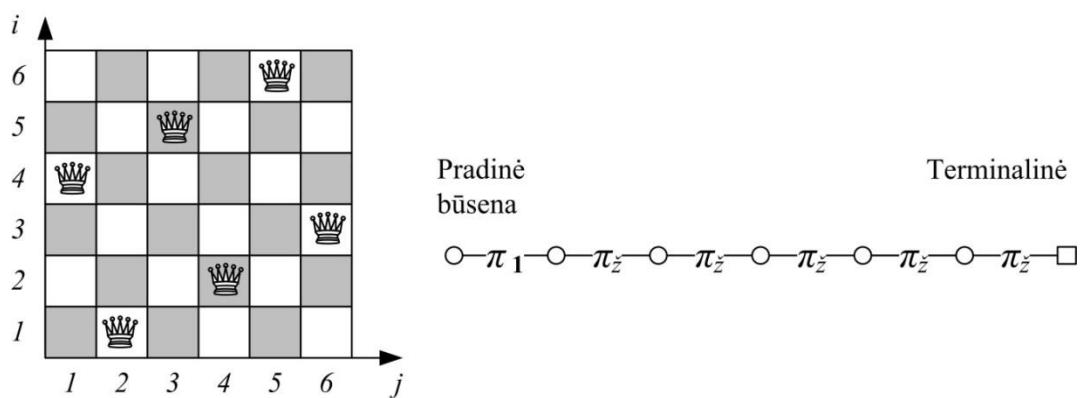
5.9 pav. Valdovių išdėstymas ir paieškos medis  $5 \times 5$  lentoje

Toliau pateikiami paieškos medžiai lentoms nuo  $N=4$  (5.8 pav.) iki  $N=9$ . Perrinkimo nėra, kai  $N=4, 5, 6$  ir  $7$ . Todėl gali kilti mintis, kad žirgo éjimo euristika veikia kaip algoritmas – be jokio perrinkimo. Tačiau perrinkimas atsiranda, kai  $N=8$  (5.13 pav.).

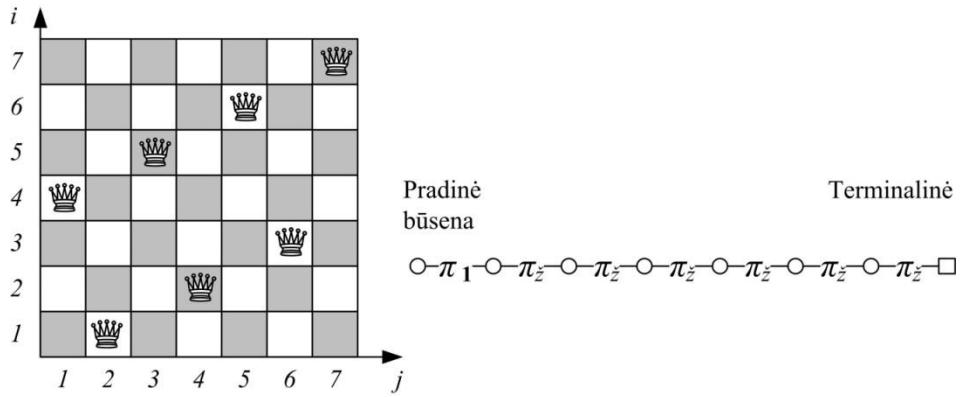
Palyginimui 5.10 pav. pateikiamas paieškos medis pilnam perrinkimui be jokios euristikos (*brute force*)  $5 \times 5$  lentoje.



5.10 pav. Paieškos medis pilnam perrinkimui be jokios euristikos (*brute force*)  
 $5 \times 5$  lentoje

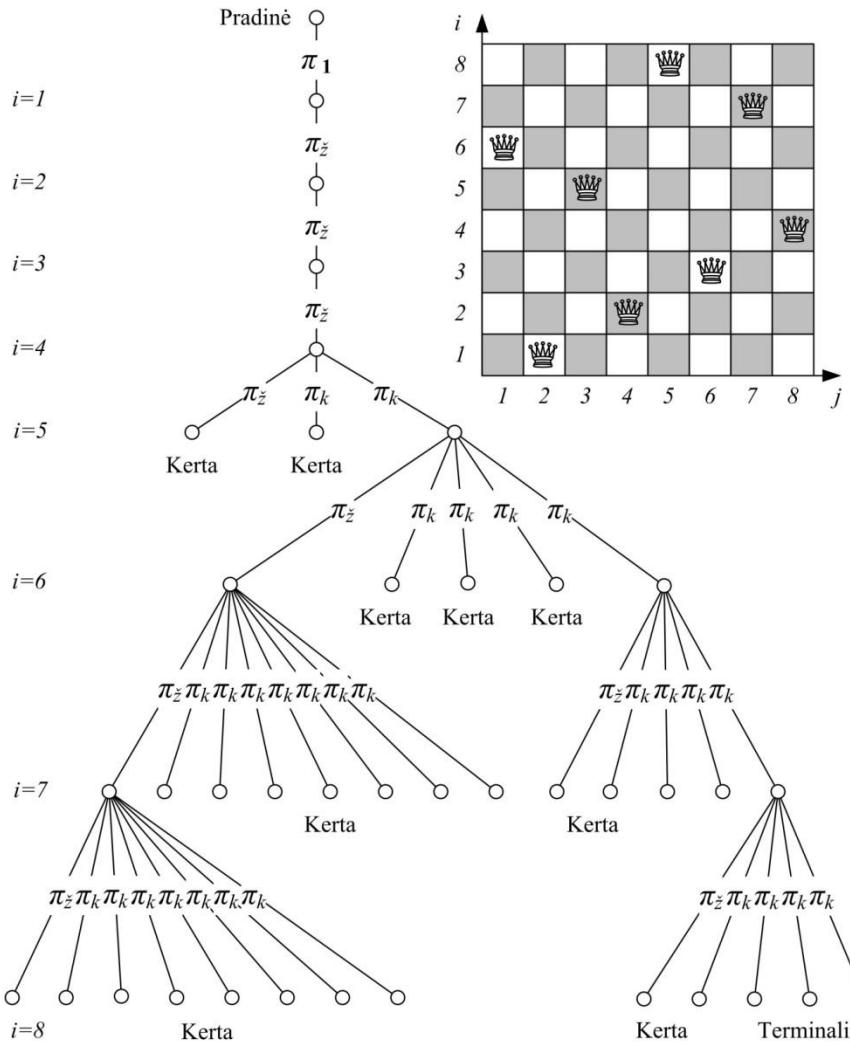


5.11 pav. Valdovių išdėstymas ir paieškos medis  $6 \times 6$  lentoje žirgo éjimu



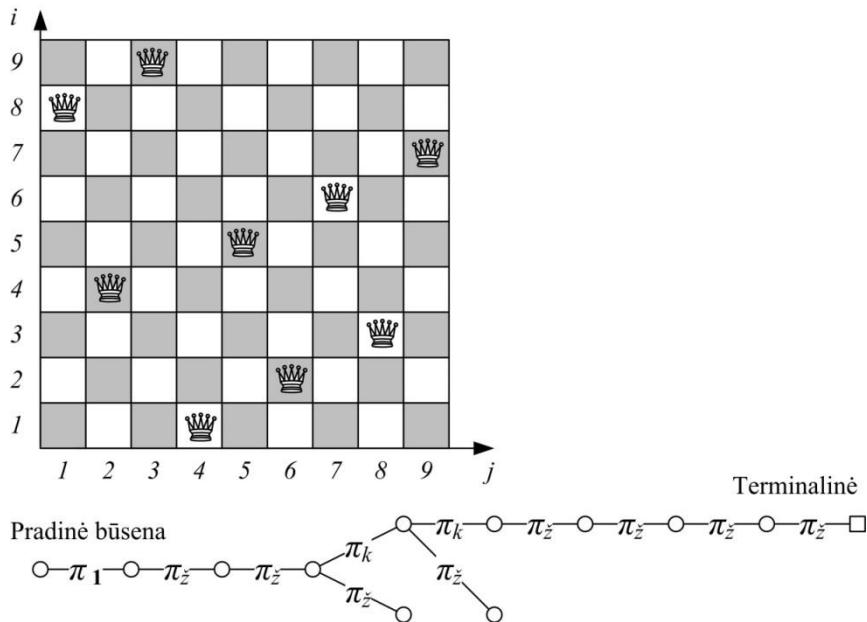
5.12 pav. Valdovių išdėstymas ir paieškos medis  $7\times 7$  lentoje žirgo ėjimu

Išnagrinėkime perrinkimą  $8\times 8$  lentoje (5.13 pav.). Pirmieji keturi ėjimai be perrinkimo, o nuo penkto žirgo ėjimas  $\pi_z$  derinamas su  $\pi_k$ , t. y su perrinkimu. Paieškoje 38 variantai (žr. 5.15 lentelės eilutę  $N=8$ ).



5.13 pav. Valdovių išdėstymas ir paieškos medis  $8\times 8$  lentoje žirgo ėjimu. Perrenkami 38 variantai. Tai mažiau negu 204 variantai su trumpiausios įstrižainės euristika ir 876 be euristikos

Paimkime  $9 \times 9$  lentą. Šios  $N = 9$  tenkina savybę  $N = 6n+3$ . Su  $\pi_1$  pasirenkamas ketvirtas pirmoje eilutėje langelis (žr. 5.14 pav.). Toliau  $\pi_z$  derinama su  $\pi_k$ . Paieškos medyje tik 2 šakos veda į aklavietę – perrinkimas mažas.



5.14 pav. Valdovių išdėstymas ir paieškos medis  $9 \times 9$  lentoje žirgo ėjimu.  
Paieškoje tik 2 grįžimai

Algoritmo gerumas vertinamas perrinkimų skaičiumi (jis lygus briaunų skaičiui paieškos medyje). 5.15 lentelėje pateikiamas perrinkimų skaičius trims algoritmams: 1) be euristikos, 2) trumpiausios įstrižainės euristika ir 3) žirgo ėjimo euristika.

Lentos dydis, $N$	Perrinkimai – briaunų skaičius paieškos medyje		
	Be euristikos (brute force)	Trumpiausios įstrižainės euristika	Žirgo ėjimo 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

### 5.15 lentelė. Perrinkimų skaičius trims algoritmams

5.15 lentelė rodo, kad euristika dažniausiai sutrumpina paiešką, bet ne visada. Tokią išimtį iliustruoja 9 eilutė, kur trumpiausios įstrižainės euristika yra blogiau negu be euristikos.

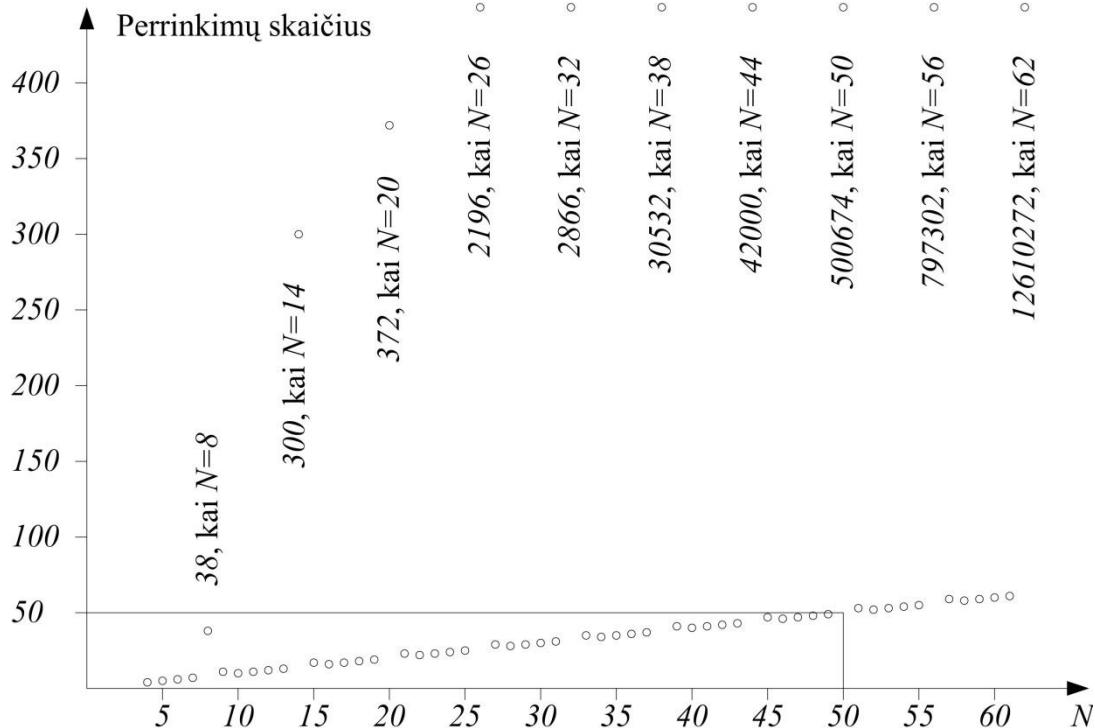
**Žirgo ėjimo euristikos tiesiskumas.** Tai galimybė perrinkti be grįžimų. Žirgo ėjimo euristika duoda rezultatą be grįžimų įspūdingai didelei lentų  $N \times N$  atvejų, būtent  $N \in \{4, 5, 6, 7, 10, 11, 12, 13, 16, 17, 18, 19, 22, 23, 24, 25\}$  ir kt.). Jose paieškos medis turi lygai  $N$  briaunų, kitais žodžiais, neturi atsišakojimų į aklavietes – neturi grįžimų. Perrinkimų skaičius yra tiesinis, bet su išimtimis. Išimtys, kai  $N \in \{8, 9, 14, 15, 20, 21, 26, 27\}$  ir kt.).

5.16 pav. pateikiamas perrinkimų skaičiaus priklausomybės nuo  $N$  grafikas.

Įrodyti hipotezę, kad visiems  $N$  perrinkimų skaičius yra tiesinis, nepavyks. Čia priminsime tokį sąmoji. Kaip klaidingai gali būti įrodinėjimas teiginys, kad „visi nelyginiai skaičiai yra pirminiai“. Vienetas yra pirminis pagal apibrėžtį; 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.

Studentai suformulavo hipotezę apie atvejus, kai yra arba nėra perrinkimo (žr. žemiau). Iš viso yra šeši  $N$  atvejai pagal dalybos iš šešių liekaną. Pabandykite įrodyti arba paneigtin.

$$N = \left\{ \begin{array}{ll} 6 \cdot n & - \text{ nėra perrinkimo} \\ 6 \cdot n + 1 & - \text{ nėra perrinkimo} \\ 6 \cdot n + 2 & - \text{ yra perrinkimas} \\ 6 \cdot n + 3 & - \text{ yra perrinkimas} \\ 6 \cdot n + 4 & - \text{ nėra perrinkimo} \\ 6 \cdot n + 5 & - \text{ nėra perrinkimo} \end{array} \right.$$



5.16 pav. Perrinkimų skaičiaus priklausomybė nuo lentos dydžio  $N$  žirgo ėjimo euristikai. Priklausomybė yra tiesinė daugeliui  $N$ . Išimtys {8, 9, 14, 15, 20, 21, 26, 27 ir kt.}

Papildomai pateikiama programa valdoviu išdėstymui 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; {Kyylančios istrizainės}
  LEID   : array [1..N2M1] of boolean;          {Besileidžiančios istrizainės}
  YRA    : boolean;
  I, J   : integer;
  BANDSK: longint;

procedure TRY (I, J: integer; var YRA: boolean);
{Iėjimas 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 istrizainės 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
              begin
                if VERT[I + 1] and KYL[I - J + 1] and LEID[I + J] then
                  begin
                    X[J + 1] := I + 1;
                    VERT[I + 1] := false; KYL[I - J + 1] := false; LEID[I + J] := false;
                    if J + 2 < N then
                      TRY( 2, J + 2, YRA)
                    else
                      begin
                        if YRA then
                          begin
                            for I := 1 to N do
                              write(X[I], ' ');
                            writeln;
                          end
                        end
                      end
                  end
                end
              end
          end
      end
  until K = J;
end;

```

```
    else
        TRY(I + 2, J + 1, YRA);
    if not YRA then
        begin {Nerastas keliai 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  $5 \times 5$  lentoje trumpiausios įstrižainės euristika. Palyginkite su paieškos medžiu pilnam perrinkimui 5.13 pav.
2. Išdėstykite 8 rikius, kad būtų kertamas kiekvienas langelis. Parašykite programą. Pasiūlykite euristiką. Pasiūlykite išdėstymo algoritmą be perrinkimo.
3. Patobulinkite žirgo ėjimo euristiką valdovių išdėstymui.
4. Nubraižykite paieškos medį valdovių išdėstymui žirgo ėjimo euristika  $21 \times 21$  lentoje.

## 6. Patobulinta paieškos su grīžimais strategija. Procedūra BACKTRACK1

Procedūra BACKTRACK turi esminių trūkumų. Pirmasis – BACKTRACK gali užsiciklini, t. y. nesustoti. Priežastis – BACKTRACK nesaugo nesēkmings būsenų.

Antrasis trūkumas – paieška gali nueiti pernelyg gylyn. Tada BACKTRACK sukurs tiek daug duomenų bazės būsenų, kad neužteks kompiuterio atminties arba bus viršytas tam tikras laiko limitas. Todėl yra tikslingo 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 nepriklauso.

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

**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 baigiamā ir grąžinamas nesēkmės požymis FAIL. Predikatu DEADEND yra programuojamos tokios žinios, kai programuotojas iš anksto žino, kad tam tikra globalios duomenų bazės būsena niekada nepasitaiko kelyje į terminalinę būseną.

**5 žingsnis.** Tikrinama, ar rekursijos gylis, t. y. sąrašo DATALIST elementų skaičius neviršija iš anksto apibrėžto gylio BOUND. Jeigu ši riba pasiekta, tai procedūra BACKTRACK1 grąžina nesėkmės požymį FAIL ir sustoja. Pavyzdžiui, LENGTH(<a,b,c>)=3, LENGTH(<a>)=1, LENGTH(<>)=0. Kintamojo BOUND paskirtis yra 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 nesėkmės 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. Pvz., TAIL(<a,b,c>)=<b,c>.

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

**11 žingsnis.** Prie globalios duomenų bazės būsenų sąrašo DATALIST prijungiant 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 nesėkmės požymis FAIL, tai reiškia, kad 8 žingsnyje paimta produkcija R neveda į sėkmę. Tenka grįžti į 7 žingsnį. Imti kitą produkciją.

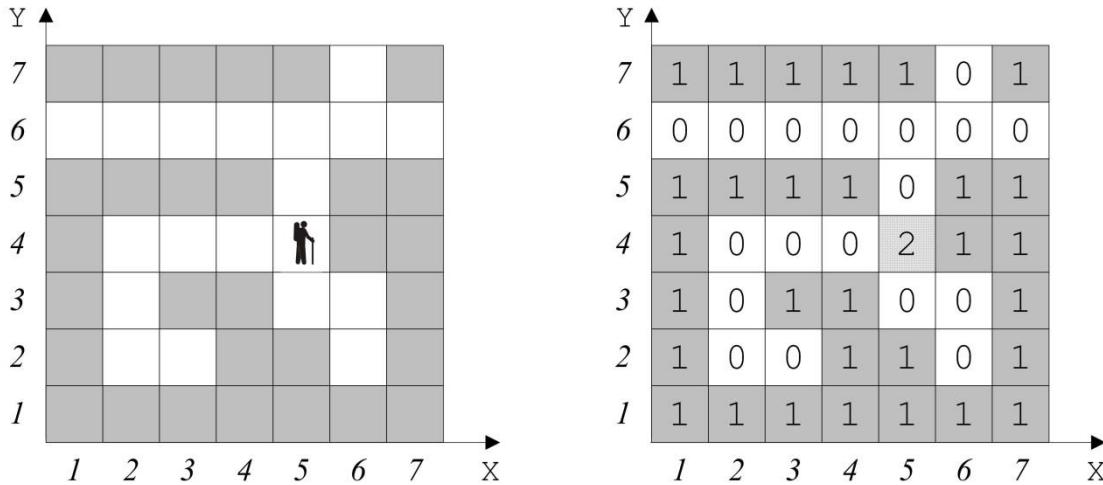
**14 žingsnis.** Kaip procedūros rezultatas yra grąžinamas produkcijų sąrašas. Funkcija CONS prijungia elementą R prie sąrašo PATH.

Procedūra BACKTRACK1 atlieka paiešką į gylį. Pavyzdžiai kituose skyriuose.

## 7. Labirinto uždavinys – kelio paieška į gylį

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

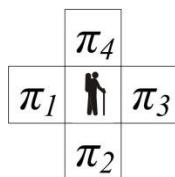
Paprasčiausiu atveju labirintas gali būti suprantamas kaip dvimatių lenta, kurios langeliai arba užpildyti, arba laisvi. Pavyzdys yra 7.1 pav.



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 langelis, t. y. „siena“, vaizduojamas sveikuojamis skaičiumi 1, laisvas langelis 0, o langelis kuriame pradinėje būsenoje stovi agentas 2, t. y.  $LAB[5,4] = 2$

Labirintą pavaizduokime dvimačiu sveikujų skaičių masyvu. Užpildytas langelis, t. y. „siena“, vaizduojamas skaičiumi 1, o laisvas langelis – 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 langelio keturios produkcijos  $\{\pi_1, \pi_2, \pi_3, \pi_4\}$  yra surūšiuotos šia tvarka:  $\pi_1$  – „i vakarus“,  $\pi_2$  – „i pietus“,  $\pi_3$  – „i rytus“ ir  $\pi_4$  – „i šiaurę“ (žr. 7.2 pav.).

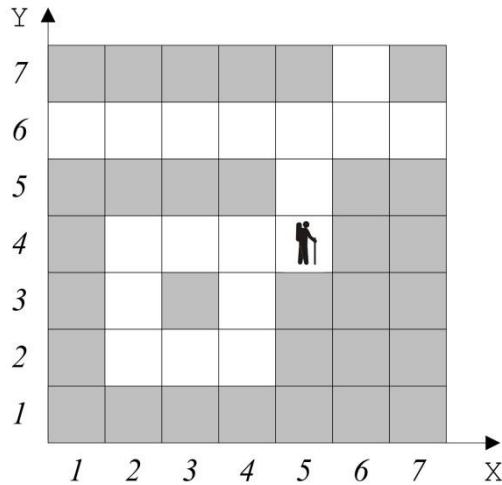


7.2 pav. Keturi agento éjimai – tai keturios produkcijos: „i vakarus“, „i pietus“, „i rytus“ ir „i šiaurę“

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

Procedūra BACKTRACK1 niekada neužsiciklina, o BACKTRACK gali ir užsiciklinti. Kai labirinte yra „salų“, BACKTRACK gali užsiciklinti (bet gali ir neužsiciklinti – tai priklauso

nuo produkcijų perrinkimo tvarkos). Pavyzdžiu, BACKTRACK užsiciklina labirinte 7.3 pav. Norint neužsiciklinti reikia įsiminti ankstesnes būsenas. Galima sakyti, kad reikia intelekto, kuris suprantamas kaip gebėjimas įsiminti einamąjį kelią.



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

Toliau pateikiama programa kelio iš labirinto paieškai valdymo su grįžimais algoritmu BACKTRACK1, t. y. į gylį. Ji spausdina kelią labirinte iš pradinio langelio 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 keliais 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; {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('Keliais egzistuoja'); {Belieka atspausdinti kelią}
else writeln('Keliais neegzistuoja'); {Keliais 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 langelį.

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 BACKTRACK\_SU\_ATMINTIMI.

V2) LAB[U,V] := 0 ir yra LAB[U,V]:=L aukščiau. Tai BACKTRACK1. 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. Paieškos algoritmai BACKTRACK ir BACKTRACK1 variantai

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

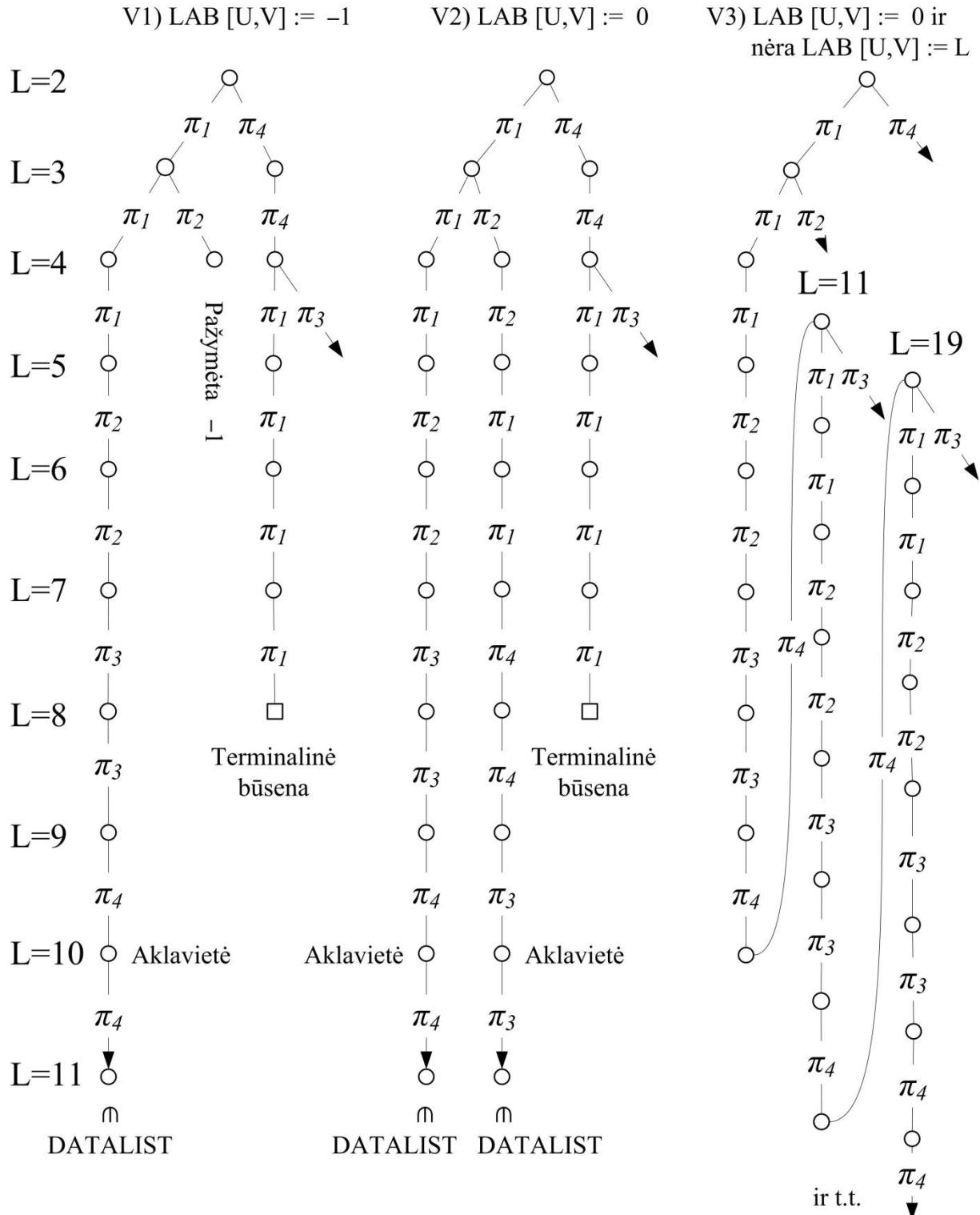
BACKTRACK < BACKTRACK1 < BACKTRACK\_SU\_ATMINTIMI

Čia „ $\leq$ “ žymi sutvarkymą pagal laiką. BACKTRACK yra prasčiausias – trunka daugiausiai laiko. Už jį geresnis yra BACKTRACK1, o už pastarajį geresnis BACKTRACK\_SU\_ATMINTIMI, kuris nagrinėjamas 13 ir 14 skyriuose. BACKTRACK1 dar įvardinamas sinonimu BACKTRACK su DATALIST kaip steku. Sutvarkymas „geresnis“ yra suprantamas kaip paieškos medžio briaunų skaičius. Tai vykdymo laiko kriterijus.

Šiuos tris variantus iliustravome kelio paieška labirinte 7.3 pav. BACKTRACK1 kaupia būsenas sąraše DATALIST, o BACKTRACK neturi tokio sąrašo. Abiejų procedūrų rekursyvumas salygoja steko formavimą kompiuterio atmintyje. Sąrašas DATALIST atlieka dar vieno steko – išreikštinio – vaidmenį. Jame kaupiamos tik einamósios šakos būsenos. BACKTRACK\_SU\_ATMINTIMI atmintyje pasižymédamas -1 kaupia ne tik einamosios šakos, bet ir visas aplankytas GDB būsenas. BACKTRACK atveju paieškos medis yra neišreikštinis. BACKTRACK1 atveju išreikštinis yra ne visas paieškos medis, o tik einamasis kelias nuo pradinės viršūnės; šis kelias kaupiamas sąraše DATALIST.

DATALIST kaip steką apibūdina siūlo alegorija. Eidamas gilyn keliautojas išvynioja siūlų kamuoliuką, o grįždamas susivynioja (žr. senovės graikų mitą apie Ariadnės siūlą, <https://lt.wikipedia.org/wiki/Ariadn%C4%97>). BACKTRACK\_SU\_ATMINTIMI langelio žymėjimą „-1“ galima palyginti su žymėjimu iškasant duobutę arba padedant akmenuką.

Paieškos į gylį (rekursinė) ir į plotį (ne rekursinė, o ciklinė) algoritmus žr. Vikipedijoje, [https://en.wikipedia.org/wiki/Depth-first\\_search](https://en.wikipedia.org/wiki/Depth-first_search), [https://en.wikipedia.org/wiki/Breadth-first\\_search](https://en.wikipedia.org/wiki/Breadth-first_search), [https://en.wikipedia.org/wiki/Graph\\_traversal](https://en.wikipedia.org/wiki/Graph_traversal).



7.4 pav. Trys paieškos medžių variantai rasti kelią labirinte 7.3 pav. V1 variantas)  $LAB[U,V] := -1$  programuojamas BACKTRACK\_SU\_ATMINTIMI. V2 variantas)  $LAB[U,V] := 0$  ir  $LAB[U,V] := L$  programuojamas BACKTRACK1, t. y. su DATALIST kaip steku. Aplink salą keliaujama antrą kartą, bet nėra užsiciklinimo. V3 variantas)  $LAB[U,V] := 0$  ir nėra  $LAB[U,V] := L$ . Tai klasikinis BACKTRACK. Programa užsiciklina

## 7.2. Produkcių algebrinės savybės

Toliau pateikiamos dvi produkcių sistemos algebrinės savybės:

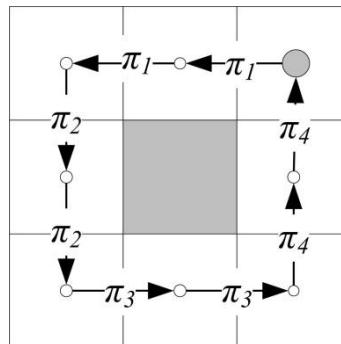
**1 savybė.** Produkcia gali būti suprantama kaip atvaizdavimas. Šio atvaizdavimo *apibrėžimo sritis* ir *kitimo sritis* yra ta pati aibė, būtent, globalios duomenų bazės būsenų aibė. Bet kuri produkcia  $\pi$  perveda GDB iš vienos būsenos į kitą:  $\pi(\text{DATA}) = \text{RDATA}$ .

**2 savybė.** Produkcijos  $\pi$  *atvirkštinė* produkcia yra žymima  $\pi^{-1}$ . Tokiu būdu, pagal apibrėzimą  $\pi^{-1} \circ \pi = I$ . Čia „ $\circ$ “ žymi atvaizdavimų *superpoziciją*, o  $I$  žymi vadinamą *vienetinį atvaizdavimą*. 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 grįztama į tą pačią būseną:

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

Šis ciklas yra pateiktas 7.4 pav. V3 variante.



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

## 7.3. Testavimas

Paieškos į gylį programą testuojame su dviem duomenų failais: 1) labirintu 7.3 pav. ir 2) susikurtu  $20 \times 15$  labirintu, kuriame kelias viršytų 100 viršunių ir būtų daug aklaviečių.

Testavimo protokole trys dalys: pradiniai duomenys, vykdymo žingsniai ir rezultatai. Labirinto 7.3 pav. atveju protokolas žemiau.

1 DALIS. Duomenys  
1.1. Labirintas

Y, V		> X, U						
7	1	1	1	1	1	0	1	
6	0	0	0	0	0	0	0	
5	1	1	1	1	0	1	1	
4	1	0	0	0	0	1	1	
3	1	0	1	0	1	1	1	
2	1	0	0	0	1	1	1	
1	1	1	1	1	1	1	1	
-----> X, U								
1 2 3 4 5 6 7								

1.2. Pradinė padėtis X=5, Y=4. L=2.

- 2 DALIS. Vykdymas
- 1) R1. U=4, V=4. Laisva. L:=L+1=3. LAB[4,4]:=3.
  - 2) -R1. U=3, V=4. Laisva. L:=L+1=4. LAB[3,4]:=4.
  - 3) --R1. U=2, V=4. Laisva. L:=L+1=5. LAB[2,4]:=5.
  - 4) ---R1. U=1, V=4. Siena.
  - 5) ---R2. U=2, V=3. Laisva. L:=L+1=6. LAB[2,3]:=6.
  - 6) ----R1. U=1, V=3. Siena.
  - 7) ----R2. U=2, V=2. Laisva. L:=L+1=7. LAB[2,2]:=7.
  - 8) -----R1. U=1, V=2. Siena.
  - 9) -----R2. U=2, V=1. Siena.
  - 10) -----R3. U=3, V=2. Laisva. L:=L+1=8. LAB[3,2]:=8.
  - 11) -----R1. U=2, V=2. Siūlas.
  - 12) -----R2. U=3, V=1. Siena.
  - 13) -----R3. U=4, V=2. Laisva. L:=L+1=9. LAB[4,2]:=9.
  - 14) -----R1. U=3, V=2. Siena.
  - 15) -----R2. U=4, V=1. Siena.
  - 16) -----R3. U=5, V=2. Siena.
  - 17) -----R4. U=4, V=3. Laisva. L:=L+1=10. LAB[4,3]:=10.
  - 18) -----R1. U=3, V=3. Siena.
  - 19) -----R2. U=4, V=2. Siūlas.
  - 20) -----R3. U=5, V=3. Siena.
  - 21) -----R4. U=4, V=4. Siūlas.  
-----Backtrack iš X=4, Y=3, L=10. LAB[4,3]:=-1. L:=L-1=9.  
-----Backtrack from X=4, Y=2, L=9. LAB[4,2]:=-1. L:=L-1=8.
  - 22) -----R4. U=3, V=2. Siūlas.  
-----Backtrack iš X=3, Y=2, L=8. LAB[3,2]:=-1. L:=L-1=7.
  - 23) -----R4. U=3, V=3. Siena.  
-----Backtrack iš X=2, Y=2, L=7. LAB[2,2]:=-1. L:=L-1=6.
  - 24) -----R3. U=3, V=3. Siena.
  - 25) -----R4. U=2, V=4. Siūlas.  
-----Backtrack iš X=2, Y=3, L=6. LAB[2,3]:=-1. L:=L-1=5.
  - 26) ---R3. U=3, V=4. Siūlas.
  - 27) ---R4. U=2, V=4. Siena.  
---Backtrack iš X=2, Y=4, L=5. LAB[2,4]:=-1. L:=L-1=4.
  - 28) --R2. U=3, V=3. Siena.
  - 29) --R3. U=4, V=4. Siūlas.
  - 30) --R4. U=3, V=5. Siena.  
--Backtrack iš X=3, Y=4, L=4. LAB[3,4]:=-1. L:=L-1=3.
  - 31) -R2. U=4, V=3. Siūlas.
  - 32) -R3. U=5, V=4. Siūlas.
  - 33) -R4. U=4, V=5. Siena.  
-Backtrack iš X=4, Y=4, L=3. LAB[4,4]:=-1. L:=L-1=2.
  - 34) R2. U=5, V=3. Siena.
  - 35) R3. U=6, V=4. Siena.
  - 36) R4. U=5, V=5. Laisva. L:=L+1=3. LAB[5,5]:=3.
  - 37) -R1. U=4, V=5. Siena.
  - 38) -R2. U=5, V=4. Siūlas.
  - 39) -R3. U=6, V=5. Siena.
  - 40) -R4. U=5, V=6. Laisva. L:=L+1=4. LAB[5,6]:=4.
  - 41) --R1. U=4, V=6. Laisva. L:=L+1=5. LAB[4,6]:=5.
  - 42) ---R1. U=3, V=6. Laisva. L:=L+1=6. LAB[3,6]:=6.
  - 43) ----R1. U=2, V=6. Laisva. L:=L+1=7. LAB[2,6]:=7.
  - 44) -----R1. U=1, V=6. Laisva. L:=L+1=8. LAB[1,6]:=8. Terminal.

3 DALIS. Rezultatai

3.1. Kelias rastas.

3.2. Kelias grafiškai

Y, V	1	1	1	1	1	0	1
7	1	1	1	1	1	0	1
6	8	7	6	5	4	0	0
5	1	1	1	1	3	1	1
4	1	-1	-1	-1	2	1	1
3	1	-1	1	-1	1	1	1
2	1	0	-1	-1	1	1	1
1	1	1	1	1	1	1	1

-----> X, U

1    2    3    4    5    6    7

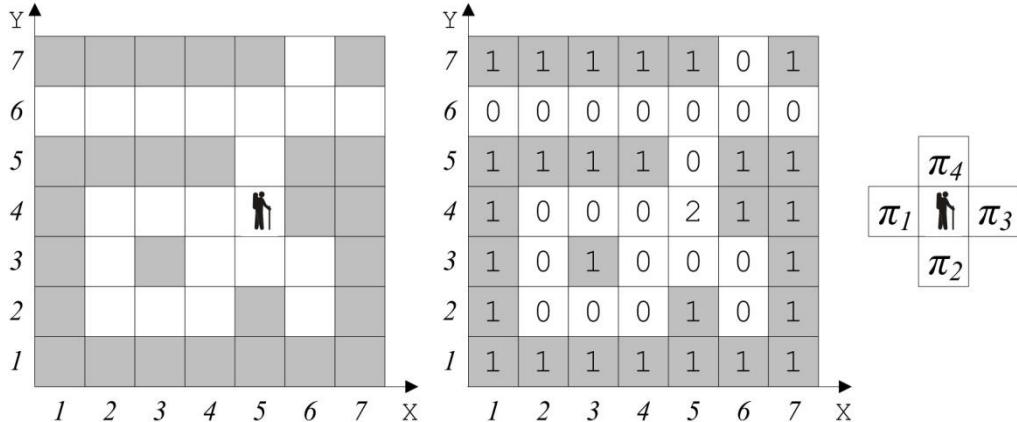
3.3. Kelias taisyklėmis: R4, R4, R1, R1, R1, R1.

3.4. Kelias viršūnėmis: [X=5,Y=4], [X=5,Y=5], [X=5,Y=6], [X=4,Y=6], [X=3,Y=6], [X=2,Y=6], [X=1,Y=6].

## 8. Paieška į plotį labirinte – bangos algoritmas

Dirbtinio intelekto produkcijų sistemoje be paieškos „i gylį“ yra ir paieškos „i plotį“ būdas. Jis dar vadinamas **bangos algoritmu**. Ji iliustruojame kelio paieška labirinte. Šis algoritmas pasižymi tuo, kad randamas trumpiausias kelias iš visų galimų.

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

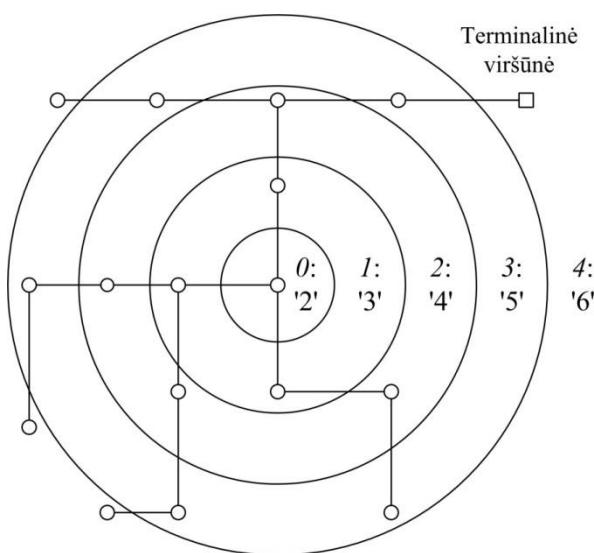


8.1 pav. Labirintas ir jo pavaizdavimas masyvu  $\text{LAB} [X, Y]$ . Agentas stovi langelyje  $X=5$ ,  $Y=4$ . Keturi agento éjimai  $\{\pi_1, \pi_2, \pi_3, \pi_4\}$

Kelio iš labirinto paieškai liekame šiuos veiksmus – cikliškai „skleidžiame bangą“:

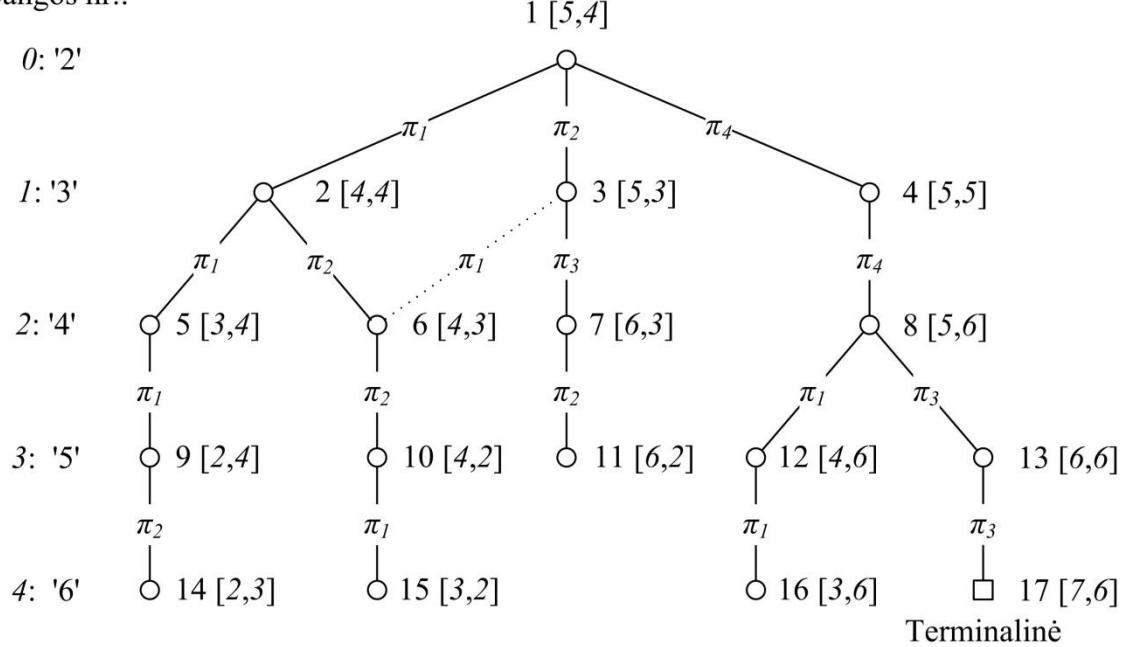
1) *Skleidžiame bangą*. Imame visų atidarytų lanelių vaikus, kuriuose dar nebuvome. Vaikais vadiname tuos gretimus (incidentinius) lanelius (grafe tai viršūnės), į kuriuos galima patekti iš einamojo lanelio, pritaikius tinkamas produkcijas. Gautus vaikus pažymime kaip „atidarytus“. Laneli, iš kurio jie buvo pasiekti, „uždarome“, t. y., pažymime „uždarytu“.

2) *Ar pasiekta terminalinė viršūnė?* Tikriname, ar yra toks atidarytas lanelis, kuris tenkina terminalinės būsenos sąlygą. Jeigu taip, tai kelias rastas. Belieka surinkti jį grįžtant atgal. Jeigu ne, tai einame į 1 žingsnį.



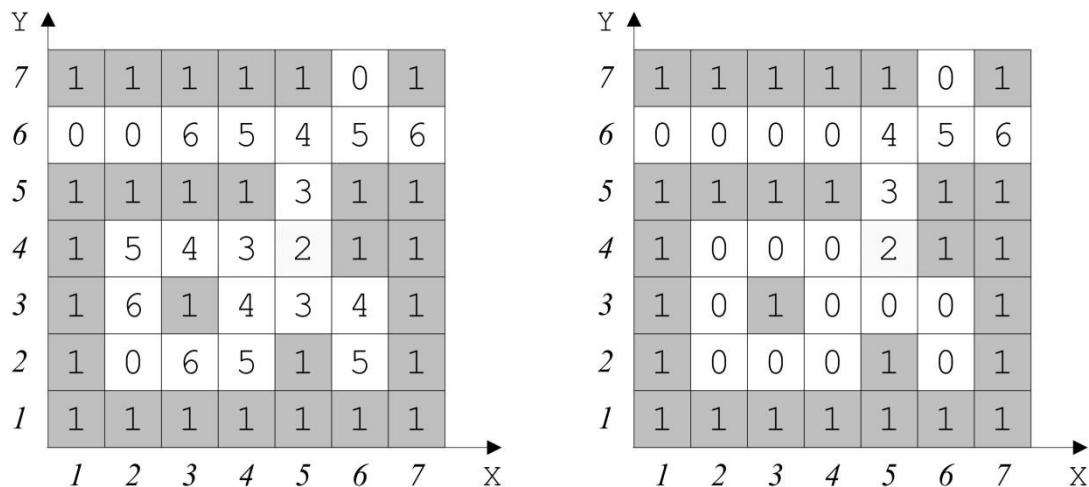
8.2 pav. Paieškos medis. Atskira banga – tai aibė viršūnių, esančių tame pačiam paieškos medžio gilyje

Bangos nr.:



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

Atidaromi langeliai yra žymimi iteracijos numeriu, t. y. bangos nr. plius 2. Kartodami iteracijas pasiekiamė būseną 8.4 a pav. Joje vienas langelis pažymėtas 6 ( $X=7, Y=6$ ) yra ant krašto. Tokiu būdu išėjimas pasiekta. Surinkę rastą kelią atgal, t. y. eidami langeliais nuo rasto išėjimo atgal numeriu mažėjimo tvarka, gauname masyvo LAB būseną 8.4 b pav.



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

Toliau yra pateikiama programa 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 keliais 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 := 5;
  repeat {Perrenkamos 4 producijos: 4, 3, 2, 1. Ieškomas toks langelis }
    {LABKOP[UU,VV], kurio reikšmė 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 langeli 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. Priskiriama 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 baigtį darba}
  begin YRA := true; U := X; V := Y;
  end;

```

```

if (X > 1) and (X < M) and (Y > 1) and (Y < N)
then
repeat {Ciklas per viršūnes}
    X := FX[UZD]; Y := FY[UZD]; {Uždaromos viršūnės 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 kvesti ATGAL(U,V)}
            else begin {Nauja viršūnė 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 terminalinių langelių paieška baigiamas. Atsakymas (kelias, planas) – tai produkcių sąrašas  $\langle \pi_4, \pi_4, \pi_3, \pi_3 \rangle$ . Kelią galima užrašyti ir langelių sąrašu  $\langle [5,4], [5,5], [5,6], [6,6], [7,6] \rangle$ . Taigi kelio užrašymas yra dualus. Žemėlapyje kelias paprastai nusakomas miestų (viršūnių) sąrašu, pvz.,  $\langle$  Druskininkai, Vilnius, Kaunas, Klaipėda  $\rangle$ . Galima ir autostradų (briaunų) numeriais  $\langle A4, A1, A1 \rangle$ .

Programoje atidaromų viršūnių koordinatės X ir Y talpinamos į pabaigą masyvą FX ir FY, vadinamą *frontu*. Frontas apjungia ir uždarytų, ir atidarytų viršūnių sąrašus. 8.4 lentelėje pateikiamas bangos algoritmo protokolas.

I	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	UZD = 13	NAUJA := 18
Banga	'2'	'3'	'3'	'3'	'4'	'4'	'4'	'4'	'5'	'5'	'5'	'5'	'5'	'6'	'6'	'6'	'6'	UZD = 13	NAUJA := 18
FX	5	4	5	5	3	4	6	5	2	4	6	4	6	2	3	6	7	UZD = 12	NAUJA := 17
FY	4	4	3	5	4	3	3	6	4	2	2	6	6	3	2	6	6	UZD = 10	NAUJA := 16
	UZD := 1	UZD = 1	NAUJA := 4	NAUJA := 5	UZD = 1	NAUJA := 6	NAUJA := 7	NAUJA := 8	NAUJA := 9	NAUJA := 10	NAUJA := 11	NAUJA := 12	NAUJA := 13	NAUJA := 14	NAUJA := 15	NAUJA := 16	NAUJA := 17	UZD = 9	NAUJA := 15
																		UZD = 8	NAUJA := 14
																		UZD = 7	NAUJA := 13
																		UZD = 6	NAUJA := 12
																		UZD = 5	NAUJA := 11
																		UZD = 4	NAUJA := 10
																		UZD = 3	NAUJA := 9
																		UZD = 2	NAUJA := 8
																		UZD = 1	NAUJA := 7
																		UZD = 0	NAUJA := 6

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

## 8.1. Testavimas

Paieškos į plotį programą testuojame su dviem duomenų failais: 1) labirintu 8.1 pav. ir 2) susikurtu  $20 \times 15$  labirintu, kurio fronte būtų virš 100 viršūnių ir daug aklaviečių.

Testavimo protokole trys dalys: pradiniai duomenys, vykdymo žingsniai ir rezultatai. Labirinto 8.1 pav. atveju protokolas nuo antros dalies tokis:

1 DALIS. Duomenys  
1.1. Labirintas

Y, V	1	1	1	1	1	0	1
7	1	1	1	1	1	0	1
6	0	0	0	0	0	0	0
5	1	1	1	1	0	1	1
4	1	0	0	0	2	1	1
3	1	0	1	0	0	0	1
2	1	0	0	0	1	0	1
1	1	1	1	1	1	1	1

-----> X, U

1 2 3 4 5 6 7

1.2. Pradinė padėtis X=5, Y=4. L=2.

2 DALIS. Vykdymas

BANGA 0, žymė L="2". Pradinė padėtis X=5, Y=4, NAUJA=1

BANGA 1, žymė L="3"

Uždaroma UZD=1, X=5, Y=4.

- R1. X=4, Y=4. Laisva. NAUJA=2.
- R2. X=5, Y=3. Laisva. NAUJA=3.
- R3. X=6, Y=4. Siena.
- R4. X=5, Y=5. Laisva. NAUJA=4.

BANGA 2, žymė L ="4"

Uždaroma UZD=2, X=4, Y=4.

- R1. X=3, Y=4. Laisva. NAUJA=5.
- R2. X=4, Y=3. Laisva. NAUJA=6.
- R3. X=5, Y=4. UŽDARYTA arba ATIDARYTA.
- R4. X=4, Y=5. Siena.

Uždaroma UZD=3, X=5, Y=3.

- R1. X=4, Y =3. UŽDARYTA arba ATIDARYTA.
- R2. X=5, Y =2. Siena.
- R3. X=6, Y =3. Laisva. NAUJA=7.
- R4. X=5, Y =4. UŽDARYTA arba ATIDARYTA.

Uždaroma UZD =4, X=5, Y=5.

- R1. X=4, Y=5. Siena.
- R2. X=5, Y=4. UŽDARYTA arba ATIDARYTA.
- R3. X=6, Y=5. Siena.
- R4. X=5, Y=6. Laisva. NAUJA=8.

BANGA 3, žymė L="5"

Uždaroma UZD=5, X=3, Y=4.

- R1. X=2, Y=4. Laisva. NAUJA=9.
- R2. X=3, Y=3. Siena.
- R3. X=4, Y=4. UŽDARYTA arba ATIDARYTA.
- R4. X=3, Y=5. Siena.

Uždaroma UZD=6, X=4, Y=3.

- R1. X=3, Y=3. Siena.
- R2. X=4, Y=2. Laisva. NAUJA=10.
- R3. X=5, Y=3. UŽDARYTA arba ATIDARYTA.
- R4. X=4, Y=4. UŽDARYTA arba ATIDARYTA.

Uždaroma UZD=7, X=6, Y=3.

- R1. X=5, Y=3. UŽDARYTA arba ATIDARYTA.
- R2. X=6, Y=2. Laisva. NAUJA=11.
- R3. X=7, Y=3. Siena.
- R4. X=6, Y=4. Siena.

Uždaroma UZD=8, X=5, Y=6.

- R1. X=4, Y=6. Laisva. NAUJA=12.
- R2. X=5, Y=5. UŽDARYTA arba ATIDARYTA.
- R3. X=6, Y=6. Laisva. NAUJA=13.
- R4. X=5, Y=7. Siena.

BANGA 4, žymė L="6"

Uždaroma UZD=9, X=2, Y=4.

- R1. X=1, Y=4. Siena.
- R2. X=2, Y=3. Laisva. NAUJA=14.
- R3. X=3, Y=4. UŽDARYTA arba ATIDARYTA.
- R4. X=2, Y=5. Siena.

Uždaroma UZD=10, X=4, Y=2.

- R1. X=3, Y=2. Laisva. NAUJA=15.
- R2. X=4, Y=1. Siena.
- R3. X=5, Y=2. Siena.
- R4. X=4, Y=3. UŽDARYTA arba ATIDARYTA.

Uždaroma UZD=11, X=6, Y=2.

- R1. X=5, Y=2. Siena.
- R2. X=6, Y=1. Siena.
- R3. X=7, Y=2. Siena.
- R4. X=6, Y=3. UŽDARYTA arba ATIDARYTA.

Uždaroma UZD=12, X=4, Y=6.

- R1. X=3, Y=6. Laisva. NAUJA=16.
- R2. X=4, Y=5. Siena.
- R3. X=5, Y=6. UŽDARYTA arba ATIDARYTA.
- R4. X=4, Y=7. Siena.

Uždaroma UZD=13, X=6, Y=6.

- R1. X=5, Y=6. UŽDARYTA arba ATIDARYTA.
- R2. X=6, Y=5. Siena.
- R3. X=7, Y=6. Laisva. NAUJA=17. Terminalinė.

### 3 DALIS. Rezultatai

#### 3.1. Kelias rastas.

#### 3.2. Kelias grafiškai

LABCOPY

Y, V	1	1	1	1	1	0	1
7	1	1	1	1	1	0	1
6	0	0	6	5	4	5	6
5	1	1	1	1	3	1	1
4	1	5	4	3	2	1	1
3	1	6	1	4	3	4	1
2	1	0	6	5	1	5	1
1	1	1	1	1	1	1	1

-----> X, U

1    2    3    4    5    6    7

#### 3.3. Kelias taisyklémis: R4, R4, R3, R3.

#### 3.4. Kelias viršūnémis: [X=5,Y=4], [X=5,Y=5], [X=5,Y=6], [X=6,Y=6], [X=7,Y=6].

## 9. Paieška į plotį grafe be kainų

Pateiksime algoritmą rasti kelią grafe, kurio briaunos neturi kainos. Turime grafą, pradinę viršūnę ir vieną terminalinę viršūnę. Paieška į plotį randamas trumpiausias keliai – mažiausiai briaunų. Algoritmas pateikiamas pagal E. Hunt knygos (1978) 10.1.1 poskyrį.

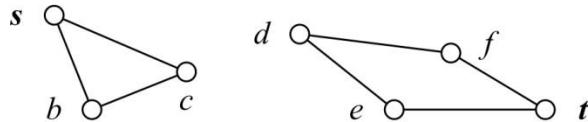
### Paieškos į plotį algoritmas grafe be kainų

ĮĖJIMAS: 1) grafas be kainų  $G$ , 2) pradinė viršūnė  $s$ , 3) terminalinė viršūnė  $t$ .

IŠĖJIMAS: trumpiausias kelias iš  $s$  į  $t$ .

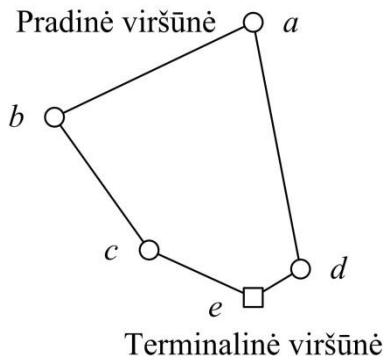
Sąrašai ATIDARYTA ir UŽDARYTA yra tušti.

1. Patalpinti pradinę viršūnę  $s$  į sąrašą ATIDARYTA.
2. Jeigu ATIDARYTA tuščias, tai kelias neegzistuoja. Grąžinti nesėkmės požymį ir baigtī. Taip atsitinka, kai grafas *nesusijęs*, t. y. tame yra nepersikertantys viršūnių poaibiai tokie, kad pradinė viršūnė yra viename poaibyje, o terminalinė kitame, ir šių poaibių viršūnių nejungia jokia briauna (pavyzdys 9.1 pav.).
3. Uždaryti **pirmąją** viršūnę  $n$  iš sąrašo ATIDARYTA, t. y. perkelti iš ATIDARYTA į UŽDARYTA. Jeigu  $n$  yra terminalinė, tai kelias rastas. Surinkti kelią atgal ir baigtī.
4. Paimti viršūnės  $n$  incidentinių viršūnių (gretimų) aibę  $S(n)$ . Iš ATIDARYTA **pabaigą** patalpinti tas viršūnes iš  $S(n)$ , kurių nėra nei ATIDARYTA, nei UŽDARYTA. Formaliai ATIDARYTA := ATIDARYTA  $\cup$   $S(n)$  / (ATIDARYTA  $\cup$  UŽDARYTA)
5. Pereiti prie 2 žingsnio.

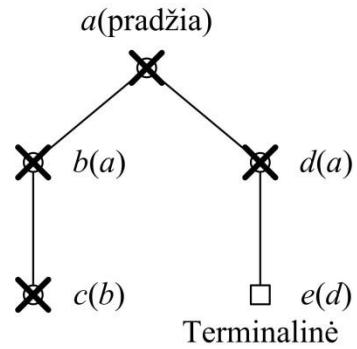


9.1 pav. Nesusijusio grafo pavyzdys. Kelias iš  $s$  į  $t$  neegzistuoja

Algoritmas veikia ir, kai terminalinių viršūnių yra keletas. Jos gali būti nusakomas tam tikru predikatu  $TERM(viršūnė)$ . Algoritmas tikrina, ar uždaroma viršūnė tenkina šį predikatą.



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



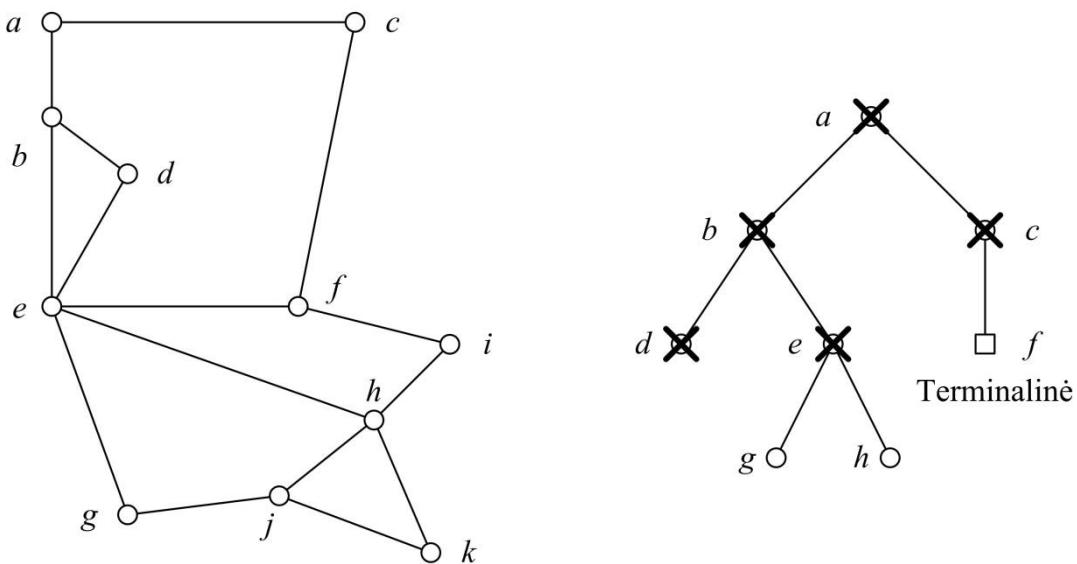
9.3 pav. Paieškos medis keliui iš  $a$  į  $e$  grafe, parodytame 9.2 pav.

Algoritmą iliustruojame grafu, pateiktu 9.2 pav. Ieškomas keliai iš pradinės viršūnės  $a$  į terminalinę viršūnę  $e$ . Paieškos medis parodytas 9.3 pav. Sąrašų ATIDARYTA ir UŽDARYTA būsenos algoritmo žingsniuose parodytos 9.4 lentelėje:

	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 nenagrinėjami

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

Kitas pavyzdys: ieškomas kelias iš  $a$  į  $f$  grafe 9.5 pav. Paieškos medis parodytas 9.6 pav., o ATIDARYTA ir UŽDARYTA būsenos – 9.7 lentelėje



9.5 pav. Grafas be kainų. Pradinė viršūnė  $a$ , terminalinė  $f$

9.6 pav. Paieškos medis keliui iš  $a$  į  $f$  grafe 9.5 pav.

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

9.7 lentelė. Sąrašai ATIDARYTA ir UŽDARYTA keliui 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. Kelias surenkamas einant nuorodomis atgal ir gaunama viršūnių seka  $\langle a(pradžia), c(a), f(c) \rangle$ . Jos ilgis yra dvi briaunos. Kelią galima vaizduoti ir viršūnėmis  $\langle a,c,f \rangle$ , ir briaunomis  $\langle (a,c), (c,f) \rangle$ . Iš vieno pavaizdavimo yra gaunamas kitas.

Paieškos medis yra pateikiamas 9.6 pav. Terminalinė viršūnė  $f$  yra antroje bangoje. Algoritmas pradeda atidarinėti ir trečiąjį bangą. Čia pastebime, kad skyriuje pateiktą klasikinį algoritmą galima patobulinti, kad jis sustotų, kai **atidaroma** terminalinė viršūnė. Tuo tarpu klasikinis algoritmas sustoja kai **uždaros** terminalinė viršūnė. Patobulinimas sutaupytu vieną bangą. Algoritmų teorijoje sudėtingumas tokis pats, ar atidaroma  $N$  ar  $N + 1$  banga. Šis patobulinimas tinkas, kai visos briaunos turi lygias kainas „vienetas“ ir netinka, kai grafo briaunos turi skirtinges kainas.

Dar vienas pavyzdys: ieškomas kelias nuo  $a$  iki  $h$  grafe 9.8 pav.

Paieškos į plotį algoritmą galima apibūdinti šitaip. Iš pradžių paimame pradinę viršūnę ir „pakratome“ grafą lyg laikydami už „pakarpas“. Briaunos išsitempią lyg guminės, ir viršūnės nusistovi pagal bangas. Kiekviena banga yra atskirame lygmenyje. Tada einame per bangas: pirmają, antrają, trečiąją, ir t.t. Suradę terminalinę viršūnę, surenkame kelią į viršų.

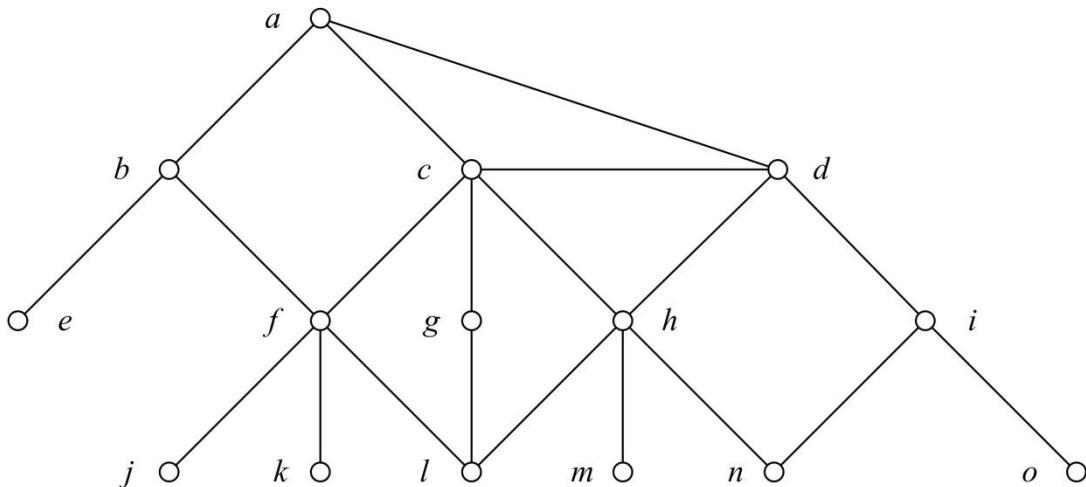
Grafo 9.8 pav. viršūnių aibė yra  $V = \{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o\}$  ir briaunų aibė

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

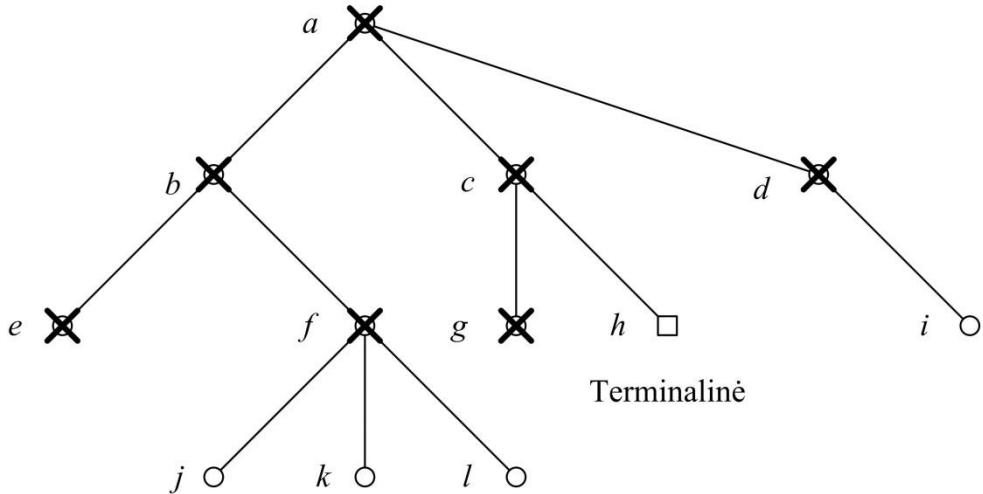
Grafas yra neorientuotas, t. y. briauna  $(a,b) \in E$  tada ir tik tada, kai  $(b,a) \in E$ . Kaip žinia, grafas  $G$  yra formalizuojamas kaip pora  $G = \langle V, E \rangle$ , sudaryta iš viršūnių aibės  $V$  ir briaunų aibės  $E$ , kur  $E \subset V \times V$ . Briaunų aibė yra viršūnių aibės Dekarto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

Grafaip yra dviejų tipų: *išreikštiniai* ir *neišreikštiniai*. Išreikštinis grafas vaizduojamas pateikiant aibes  $V$  ir  $E$  kaip sąrašus. Išreikštinį grafą yra įprasta vaizduoti grafiškai.

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



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



9.9 pav. Paieškos medis grafe 9.8 pav. iš  $a$  į  $h$

Atidaromas ir uždaromos viršūnes vaizduojame vienam fronte:

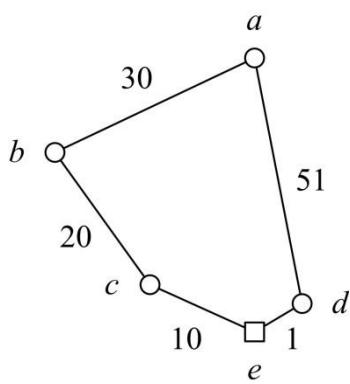
$a, b, e, d, e, f, g, h, i, j, k, l$

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

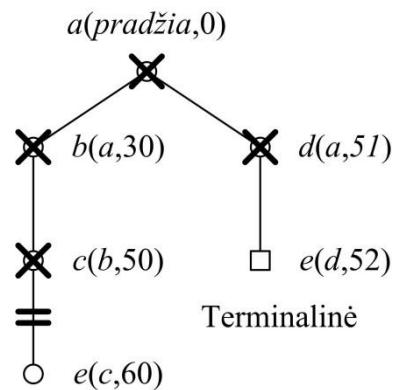
## 10. Paieška grafe su kainomis

Algoritmas pateikiamas vadovaujantis (Hunt 1978, 10.1.2 sk.). Tai variacija olandų informatiko Edsger Dijkstra 1956 m. sukurtam algoritmui, skirtam rasti trumpiausius kelius nuo vienos viršūnės iki kitų svoriniame grafe su neneigiamais svoriais; žr. [http://en.wikipedia.org/wiki/Dijkstra%27s\\_algorithm](http://en.wikipedia.org/wiki/Dijkstra%27s_algorithm). Kelio kaina tarp dviejų viršūnių yra kraštinių kainų suma tame kelyje. Paieška iliustruojama grafu 10.1 pav. Paieškos medis parodytas 10.2 pav.

Kai kiekvienos briaunos kaina yra vienetas, tai uždavinyse ekvivalentus rasti kelią su mažiausiai briaunų, kuris išnagrinėtas prieštame skyriuje.



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.

**ĮJIMAS:** 1) grafas  $G$ , kurio briaunos turi neneigiamas kainas; 2) pradinė viršūnė  $s$ ; 3) terminalinė viršūnė  $t$ .

**IŠĖJIMAS:** trumpiausias kelias nuo  $s$  iki  $t$ .

1. Patalpinti pradinę viršūnę  $s$  į sąrašą ATIDARYTA.
2. Jeigu sąrašas ATIDARYTA tuščias, tai kelias neegzistuoja. Grąžinti nesékmés požymį ir baigti. Tokia situacija, kai grafas nesusijęs.
3. Uždaryti tokią viršūnę  $n$ , kurios kelio nuo pradinės viršūnės kaina yra **mažiausia**: perkelti  $n$  iš ATIDARYTA į UŽDARYTA. Tokiu būdu sąrašas ATIDARYTA rūšiuojamas. Jeigu  $n$  terminalinė, tai kelias rastas. Surinkti kelią atgal ir baigti.
4. Paimti viršūnės  $n$  incidentinių viršūnių aibę  $S(n)$ . Kiekvienai viršūnei  $n^*$  iš  $S(n)$ , kurios nėra sąraše UŽDARYTA, apskaičiuoti naują kelio kainą. Formaliai,  $\forall n^* \in S(n)/UŽDARYTA$  priskirti  $KAINA(s, n^*) := KAINA(s, n) + BRIAUNA(n, n^*)$ . Iš ATIDARYTA ištraukti tas  $S(n)$  viršūnes, kurių ATIDARYTA dar nėra. Formaliai, jeigu  $n^* \notin ATIDARYTA$ , tai  $ATIDARYTA := ATIDARYTA \cup n^*$ . Jeigu  $n^* \in S(n)$  jau priklauso ATIDARYTA, tai palyginti seno ir naujo kelio kainas. Jeigu naujas kelias yra geresnis, tai ištraukti  $n^*$  iš ATIDARYTA su geresne kaina ir nauja nuoroda.
5. Pereiti prie 2 žingsnio.

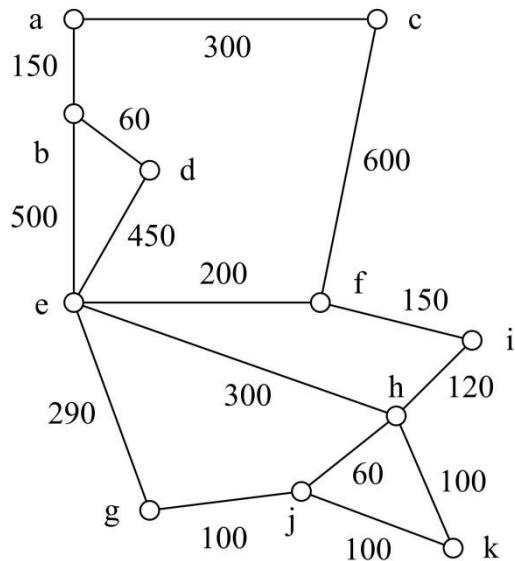
Žemiau pateikiamos ATIDARYTA ir UŽDARYTA būsenos algoritmo žingsniuose.

	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 <b>pasirenkamas</b>
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 keliais  $\langle a(pradžia, 0), d(a, 51), e(d, 52) \rangle$

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 esantį kelio per  $c$  įvertinimą  $e(c, 60)$ , nes  $52 < 60$ . Todėl pasirenkama  $e(d, 52)$  – su geresniu kelio įvertinimu. Algoritmo darbas baigiamas 6 žingsnyje, kai į sąrašą UŽDARYTA patalpinama terminalinė  $e$ . Rastas keliais  $\langle a(pradžia, 0), d(a, 51), e(d, 52) \rangle$  arba trumpai tiesiog  $\langle a, d, e \rangle$ .

Toliau pateikiamas kitas pavyzdys. Grafas parodytas 10.4 pav.



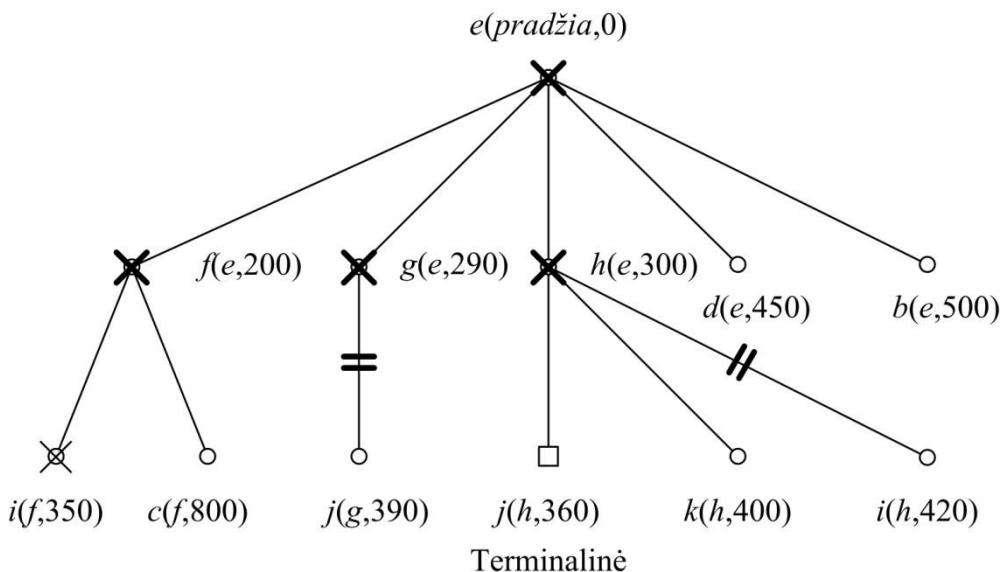
10.4 pav. Grafas, kurio briaunos turi kinas. Ieškomas keliais iš  $e$  į  $j$

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 senąjį. Naujas $j$ įvertinimas $j(h, 360)$ yra geresnis negu senas $j(g, 390)$ , todėl ir <b>pasirenkamas</b>
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  $\langle e(pradžia, 0), h(e, 300), j(h, 360) \rangle$

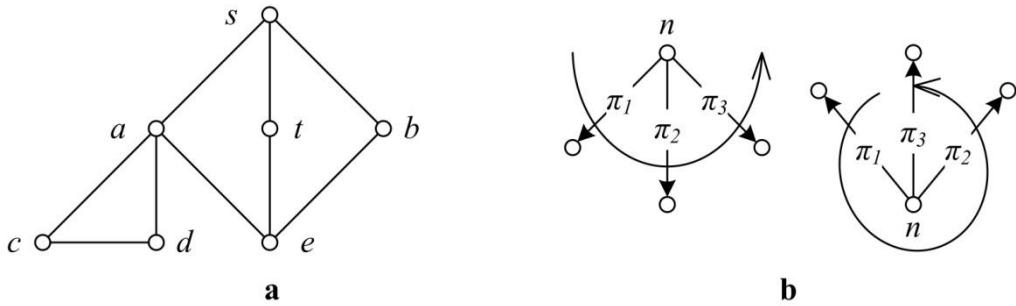
Paieškos medis atitinkantis aukščiau nagrinėtą uždavinį pateiktas 10.6 pav.



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

## 11. Paieškos į gylį algoritmas grafe

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. „dvylirktos valandos“ produkcija yra paskutinė (11.1 b pav.).



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

Paieškos į plotį algoritmas randa kelią  $\langle s,t \rangle$ . Šis kelias bus randamas uždarant pirmąjā 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. Uždarės viršūnė  $t$ , paieškos į plotį algoritmas baigia darbą.

Paieškos į gylį (*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 (*FIFO – First In First Out*) principu. O paieškos į gylį atveju – į jo **pradžią**, t. y. steko (*LIFO – Last In First Out*) principu.

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

**ĮĖJIMAS:** 1) grafas be kainų; 2) pradinė viršūnė  $s$ ; 3) terminalinė viršūnė  $t$ .

**IŠĖJIMAS:** kelias nuo  $s$  iki  $t$ .

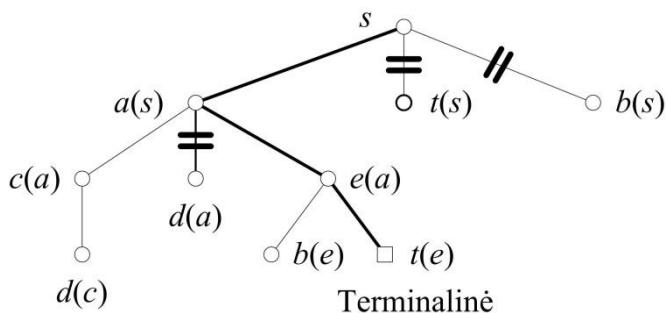
1. Pradinę viršūnę patalpinti į sąrašą ATIDARYTA.
2. Jeigu ATIDARYTA tuščias, tai kelias neegzistuoja. Grąžinti nesékmés požymį ir baigt. Baigt darbą, grąžinant nesékmés požymį. Taip atsitinka, kai grafas nesusijęs.
3. Uždaryti **pirmąjā** viršūnę  $n$  iš sąrašo ATIDARYTA, t. y. perkelti iš ATIDARYTA į UŽDARYTA. Jeigu  $n$  yra terminalinė, tai kelias rastas. Surinkti kelią atgal ir baigt.
4. Paimti viršūnės  $n$  incidentinių viršūnių aibę  $S(n)$ . Į ATIDARYTA **pradžią** patalpinti tas viršūnes iš  $S(n)$ , kurios neuždarytos, t. y.  $S(n)/UŽDARYTA$ . O tas viršūnes, kurios kartojasi, t. y.  $S(n) \cap$  ATIDARYTA, iš ATIDARYTA pabaigos pašalinti. Šitaip pakartotinai atidaromoms viršūnėms yra keičiamos nuorodos pagal principą „i gylį“ (nuorodos rodo, iš kur viršūnės buvo pasiektos).
5. 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  $s$  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. Perkeliamą 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 $S(a) = \{c,d,e,s\}$ . $s$ yra UŽDARYTA, todėl nenagrinnėjama. Tos viršūnės, kurios nepriklauso UŽDARYTA, t. y. $\{c,d,e\}$ , patalpinamos į ATIDARYTA <b>pradžią</b>	
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 nenagrinnėjama. $d$ patalpinama į ATIDARYTA <b>pradžią</b> , o iš pabaigos pašalinama. Tokiu būdu pakeičiamos 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. Perkeliamą į UŽDARYTA. Jos incidentinės $S(a) = \{c,a\}$ yra UŽDARYTA, todėl nenagrinnėjamos	
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 nenagrinnėjama. $b$ ir $t$ patalpinamos į ATIDARYTA <b>pradžią</b> , 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 nenagrinnėjamos	

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 plotį“ skirtumas nuo „i gylį“

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

	ATIDARYTA	UŽDARYTA ir frontas	Komentaras	Paieškos medis „i plotį“
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 neprieklauso nei ATIDARYTA, nei UŽDARYTA, t. y. $c, d$ ir $e$ , patalpinamos į ATIDARYTA <b>pabaiga</b>	
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 plotį“ 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 plotį 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* (*problem-solver*). Sinonimas yra *sprendžiantis agentas* (*solving agent*). Iprasta kad sprendėjo uždavinio prigimtis yra spręsti ji vieną kartą. Pavyzdžiu, ištrūkti iš labirinto. Sprendėjas neturi prieš akis labirinto žemėlapio. Ištrūkti iš šalto labirinto yra jo gyvybės ar mirties klausimas.

Paieška į plotį paprastai naudojasi *planuotojas* (*planner*). Sinonimas yra *planuojantis agentas* (*planning agent*). Racionalu, kad planuotojas prieš akis turi žemėlapį. Pavyzdžiu, planuotojas sėdi patogiamame krėslėje šiltame kambaryje ir ieško trumpiausio maršruto kroviniui 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. Pavyzdžiu, grafe rastu trumpiausiu keliu vežėjui yra prasminga daug kartų gabenti krovinių 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.

Paiešką į gylį galima palyginti su pasivaikščiojimu neturint tikslų rasti trumpiausią kelią. O paieškos į plotį tikslas ne tik vaikščioti, bet ir rasti trumpiausią kelią.

Galima iškelti klausimą, kuris algoritmas geresnis? Toks klausimo kėlimas nekorekiškas, nes sprendėjas ir planuotojas paprastai naudojami esant skirtingoms aplinkybėms. Keliant palyginimo klausimą turi būti apibrėžtas kriterijus. Paieška į plotį naudojama ieškant trumpiausio kelio. Paieška į gylį siejama su intelekto sąvoka. Laikoma, kad žmogaus trumpalaikė atmintis veikia paieškos į gylį principu. Tokiu būdu vienoje situacijoje gali geriau tiki vienas algoritmas, pavyzdžiu, kai iš anksto nežinomas dalykinės srities būsenų grafas, kitoje – kitas, kai žinomas.

Iprasta, kad sprendėjas ir planuotojas naudojami skirtingose situacijose:

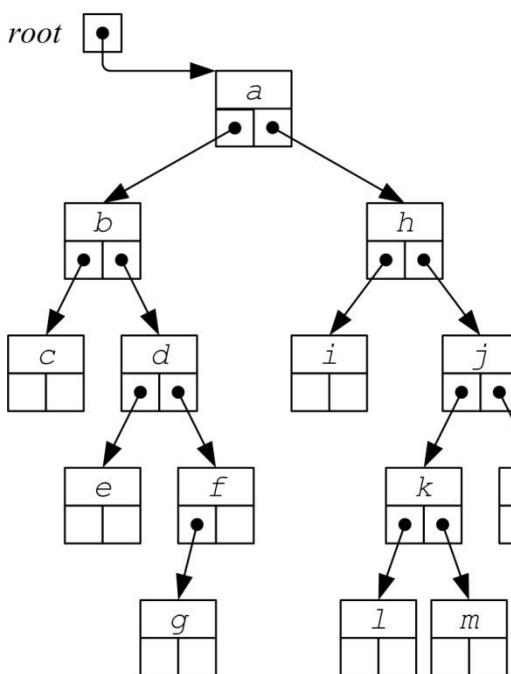
	Sprendėjas	Planuotojas
1	Neturi žemėlapio	Turi žemėlapį
2	Randa nebūtinai trumpiausią kelią	Gali rasti trumpiausią kelią
3	Paieška į gylį	Paprastai paieška į plotį
4	Keliu pasinaudoja vieną kartą	Kelias naudojamas daug kartų

## 12. Prefiksinė, infiksinė ir postfiksinė medžio apėjimo tvarka

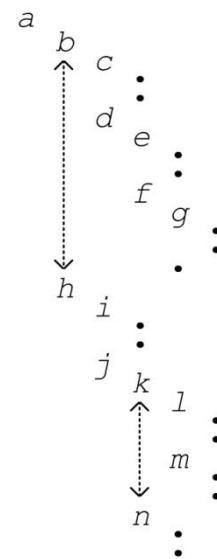
Iki šiol aptarėme medžio apėjimo tvarką „i gylį“ ir „i plotį“. Yra ir kitokių visuotinai priimtų būdų. Jeigu medyje yra  $N$  viršūnių, tai apėjimo būdų yra tiek, kiek yra viršūnių perstatų, t. y.  $N$  faktorialas  $N! = N \cdot (N-1) \cdot (N-2) \cdot \dots \cdot 1$ . Čia apėjimo būdas suprantamas 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 koduoja simbolų eilutė  
 $abc\dots de\dots fg\dots hi\dots jkl\dots mn\dots$



12.2. pav. Medis, izomorfiškas  
12.1 pav., bet pavaizduotas knygos turinio metafora

**Prefiksinė dvejetainio medžio apėjimo tvarka** (*prefix order*) – medžio viršūnės vardinamos taip: viršūnė, kairysis pomedis, dešinysis pomedis. Mes vaizduojame 100, kur 1 žymi vardinių (t. y. informacijos, siejamos su viršūne apdorojimą), o 0 – pomedžio apėjimą. Ši tvarka yra ta pati, kuria viršūnės apeinamos paieškoje i gylį dvejetainiame medyje.

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

**Postfiksinė dvejetainio medžio apėjimo tvarka** (*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 kaip K. Jensen ir N. Virtos knygos 12.1 poskyryje; žr. (Jensen, Wirth 1982, p. 65–67). Primename, kad N. Virtas yra žinomas kaip Paskalio programavimo kalbos autorius. Jo elegantiškos procedūros skirtos iliustruoti ir rekursiją, ir medžio apėjimus.

Medis 12.1 pav. yra suformuojamas, kai įvedama tokia simbolių eilutė:

*abc..de...fg...hi..jkl...m...n...*

Ši eilutė aiškinama pasitelkiant knygos turinio metaforą. Eilutė koduoja, kaip knygos struktūrinės dalys – skyriai ir poskyriai – eina turinyje su atitraukimais; žr. 12.2 pav.

```
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	– infišinė tvarka, procedūra inorder
Cegfdbilmknjha	– postfiksinė tvarka, procedūra postorder.

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

*abcdefghijklmn*

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

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

*abhcdijefknglm*

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

**Infišinė aritmetinės išraiškos užrašymo forma** – tai mums įprasta aritmetinės išraiškos užrašymo forma. Paprastai dar naudojami skliaustai kaip metasimboliai, nurodant veiksmų tvarką su įdėtinėmis išraiškomis, pavyzdžiui:

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

Šios išraiškos sintaksinis medis parodytas 12.3 pav.

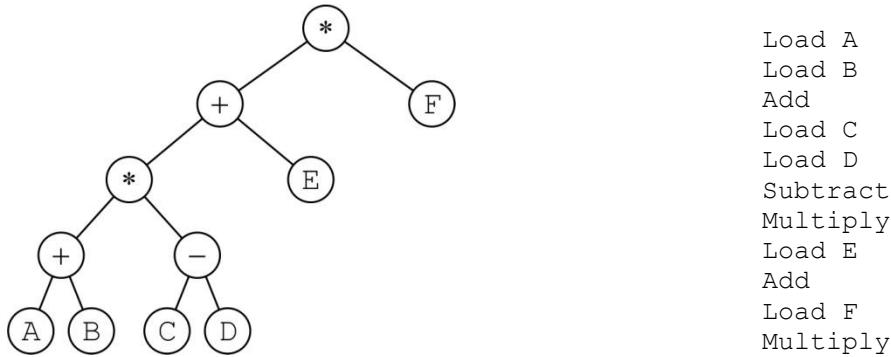
**Prefiksinė aritmetinės išraiškos užrašymo forma** užrašoma rekursyviai – operacijos ženklas, pirmasis operandas ir antrasis operandas:

\*+\*+AB-CDEF

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

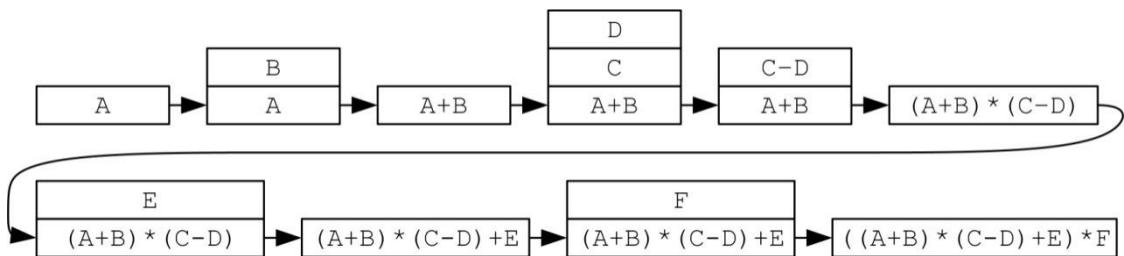
AB+CD-\*E+F\*

Iliustruojame išraiškos komiliavimą į stekinės mašinos tarpinę kalbą. Pavyzdžiui,  $( (A+B) * (C-D) +E ) *F$  yra komiliuojama į stekinės mašinos komandų seką 12.3 pav.



12.3 pav. Sintaksinis medis aritmetinei išraiškai  $((A+B)*(C-D)+E)*F$  ir stekinės mašinos komandų seka

Toliau 12.4 pav. parodytas stekinės mašinos atminties – steko – kitimas laiko taktais, kai vykdomas mašininis kodas 12.3 pav. Rodyklė vaizduoja perėjimą iš steko būsenos laiko momentu  $t$  į būseną  $t+1$ .

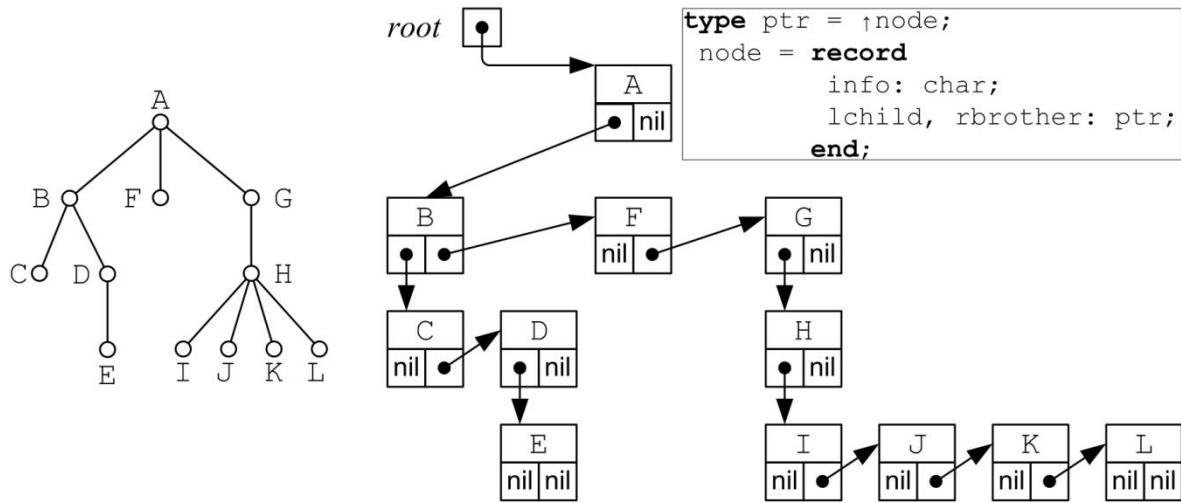


12.4 pav. Steko turinio kitimas laike išraiškai  $((A+B)*(C-D)+E)*F$

### Pratimai

1. Užrašykite išraiškas  $A*B+C/D$ ,  $A*B+C-D$  ir  $A+B*C-(D+E)*F-B$  prefiksine, infiksine bei postfiksine formomis. Užrašykite stekinės mašinos kodą pastarosioms išraiškoms ir pavaizduokite steko kitimą.

3. Parašykite procedūras darbui su bendro pavidalo medžiais: a) medžio įvedimui, b) apėjimui prefiksine tvarka ir c) apėjimui postfiksine tvarka. Pavyzdžiu, simbolių eilutė ABC.DE...F.GHI.J.K.L koduoja medį, parodytą 12.5 pav. Galimos ir kitos medžių užrašymo kalbos, pvz., A(B(CD(E))FG(H(IJKL))), t. y. su skliaustais žymėti hierarchiją.



12.5 pav. Bendro pavidalo medis ABC.DE...F.GHI.J.K.L ir jo vaizdavimas kompiuterio atmintyje

Pseudokodas medžio kodavimo kalbai įvesti yra tokis.

ĮĖJIMAS: *graph\_string* – simbolių eilutė koduoti medži.  
IŠĖJIMAS: *node* – nuoroda į medžio šakninę viršūnę.

```

function construct_tree(graph_string) : node
  current_node := nil
  for all character ∈ graph_string
    if character = dot_character
      new_node := node(character)
      set_parent(new_node, current_node)
      set_children(new_node, nil)
      if current_node ≠ nil
        add_child(current_node, new_node)
      end if
      current_node := new_node
    else
      current_node := get_parent(current_node)
    end if
  end for
  while get_parent(current_node) ≠ nil do
    current_node := get_parent(current_node)
  end while
  return current_node
end function

```

Testuojama su medžiu 12.5 pav. Testo įvestis: ABC.DE...F.GHI.J.K.L . Toliau pateikiama testo išvestis.

1 DALIS. Duomenys

Medis užrašytas simbolių eilute: "ABC.DE...F.GHI.J.K.L "

2 DALIS. Vykdymas

- 0) . Skaitoma 'A'. Itraukiamas vaikas A iš šaknies vaikus. Gilyн.
- 1) A. Skaitoma 'B'. Itraukiamas vaikas B iš A vaikus. Gilyн.
- 2) -B. Skaitoma 'C'. Itraukiamas vaikas C iš B vaikus. Gilyн.
- 3) --C. Skaitoma '.'. Viršūnė C neturi vaikų. Grižtame.

- B. Skaitoma 'D'. Įtraukiamas vaikas D i B vaikus. Gilyn.
- 4) --D. Skaitoma 'E'. Įtraukiamas vaikas E i D vaikus. Gilyn.
- 5) ---E. Skaitoma '..'. Viršūnė E neturi vaikų. Grīztame.
- D. Skaitoma '..'. Viršūnė D neturi vaikų. Grīztame.
- B. Skaitoma '..'. Viršūnė B neturi vaikų. Grīztame.
- A. Skaitoma 'F'. Įtraukiamas vaikas F i A vaikus. Gilyn.
- 6) -F. Skaitoma '..'. Viršūnė F neturi vaikų. Grīztame.
- A. Skaitoma 'G'. Įtraukiamas vaikas G i A vaikus. Gilyn.
- 7) -G. Skaitoma 'H'. Įtraukiamas vaikas H i G vaikus. Gilyn.
- 8) --H. Skaitoma 'I'. Įtraukiamas vaikas I i H vaikus. Gilyn.
- 9) ---I. Skaitoma '..'. Viršūnė I neturi vaikų. Grīztame.
- H. Skaitoma 'J'. Įtraukiamas vaikas J i H vaikus. Gilyn.
- 10) ---J. Skaitoma '..'. Viršūnė J neturi vaikų. Grīztame.
- H. Skaitoma 'K'. Įtraukiamas vaikas K i H vaikus. Gilyn.
- 11) ---K. Skaitoma '..'. Viršūnė K neturi vaikų. Grīztame.
- H. Skaitoma 'L'. Įtraukiamas vaikas L i H vaikus. Gilyn.
- 12) ---L. Skaitoma pabaigos požymis ' '. Medis sukonstruotas.

### 3 DALIS. Rezultatai

Medis:

- 1) A
- 2) -B
- 3) --C
- 4) --D
- 5) ---E
- 6) -F
- 7) -G
- 8) --H
- 9) ---I
- 10) ---J
- 11) ---K
- 12) ---L

### Kalbos medžiui koduoti gramatika Backus-Naur'o forma

```
<medis> ::= <viršūnė> | <viršūnė> <šakos>
<šakos> ::= <šaka> | <šakos> <šaka>
<šaka> ::= <lapas> | <išsišakojimas>
<lapas> ::= <viršūnė> <taškas>
<išsišakojimas> ::= <viršūnė> <šakos> <taškas>
<viršūnė> ::= 'A' .. 'Z'
<taškas> ::= '.'
```

**Kontekstinė taisyklė.** Tokia gramatika gamina simbolių eilutes, kurių pabaigoje yra taškai. Šie taškai eilutės pabaigoje prasmingos informacijos apie medį neduoda. Todėl jie praleidžiami, kad eilutė baigtusi viršūnės vardu.

**Eilutės validumo kriterijus.** Simbolių eilutė validi tada ir tik tada, kai ją skaitant nuo pradžios po vieną simbolį niekada neiškils situacija, kad perskaitytų taškų kiekis sutaps su perskaitytų viršūnių kiekiu einamuoju momentu.

Dvejetainis medis iš 12.1 pav. koduojamas šitaip:

*abc.de.fg....hi.jkl.m..n*

Pastebime, kad pastaroji simbolių eilutė skiriasi nuo eilutės *abc..de...fg...hi...jkl..m..n..* iš (Jensen, Wirth 1982, p. 66).

## 13. Bendras paieškos grafe algoritmas GRAPHSEARCH

Ankstesnėse temose išnagrinėjome algoritmus „i gylį“ ir „i plotį“. Apjunkime šias paieškas i 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 kaulukai, kryžiukai-nuliukai, šaškės, sachmatai ir kt. būsenų medžiai.

ĮĖJIMAS: 1) grafas; 2) pradinė viršūnė  $s$ ; 3) terminalinė viršūnė (jų gali būti keletas).

IŠĖJIMAS: kelias nuo pradinės viršūnės  $s$  iki terminalinės (nebūtinai trumpiausias).

Sarašai ATIDARYTA ir UŽDARYTA yra tušti.

1. Patalpinti pradinę viršūnę  $s$  į sarašą ATIDARYTA ir sukurti paieškos medį  $T$ , susidedantį iš vienos viršūnės  $s$ .
2. Jeigu ATIDARYTA tuščias, tai kelias neegzistuoja. Grąžinti nesėkmės požymį ir baigti.
3. Uždaryti **pirmają** viršūnę  $n$  iš ATIDARYTA: perkelti ją į UŽDARYTA.
4. Jeigu  $n$  yra terminalinė, tai sėkmė. Sekant nuorodomis atgal nuo  $n$  iki  $s$  surinkti kelią ir baigti.
5. Išskleisti  $n$ , t. y. paimti  $n$  incidentinių viršūnių aibę  $S(n)$ . Tas  $n^* \in S(n)$ , kurių nėra nei ATIDARYTA, nei UŽDARYTA, t. y.  $n^* \in (S(n) / (\text{ATIDARYTA} \cup \text{UŽDARYTA}))$ , patalpinti į ATIDARYTA ir į  $T$  formuojant nuorodas  $n^*(n)$  atgal. Formaliai  $\text{ATIDARYTA} := \text{ATIDARYTA} \cup S(n) / (\text{ATIDARYTA} \cup \text{UŽDARYTA})$
6. Kiekvienai viršūnei  $n^* \in S(n)$ , kuri priklausė ATIDARYTA (senajai būsenai, žr. dešiniąją priskyrimo pusę aukšciau), t. y.  $n^* \in (S(n) \cap \text{ATIDARYTA})$ , nuspresti, ar perorientuoti nuorodą. Nuspresti būtina nes yra senas kelias ir naujas kelias. Likusių  $n^*$ , t. y.  $n^* \in (S(n) \cap \text{UŽDARYTA})$ , nenagrinėti.
7. Sutvarkyti ATIDARYTA pagal kokią nors schemą arba euristiką. Pavyzdžiu,
  - a) pagal kainą;
  - b) „i gylį“, t. y.  $S(n) / \text{UŽDARYTA}$  patalpinti į ATIDARYTA **pradžią**, o besidubliuojančias viršūnes iš ATIDARYTA pabaigos pašalinti, kad nesikartotų. Taip sutvarkoma **steko** principu, t. y. LIFO (Last In First Out).
  - c) „i plotį“, t. y.  $S(n) / \text{UŽDARYTA}$  patalpinti į naujojo ATIDARYTA **pabaigą**, o besidubliuojančias viršūnes iš ATIDARYTA **pradžios** pašalinti. Taip sutvarkoma **eilės** principu, t. y. FIFO (First In First Out).
8. Pereiti prie 2 žingsnio.

GRAPHSEARCH algoritmas pateiktas (Nilsson 1982, 2.2.2 sk.)

## 14. Skirtumai tarp BACKTRACK1 ir GRAPHSEARCH

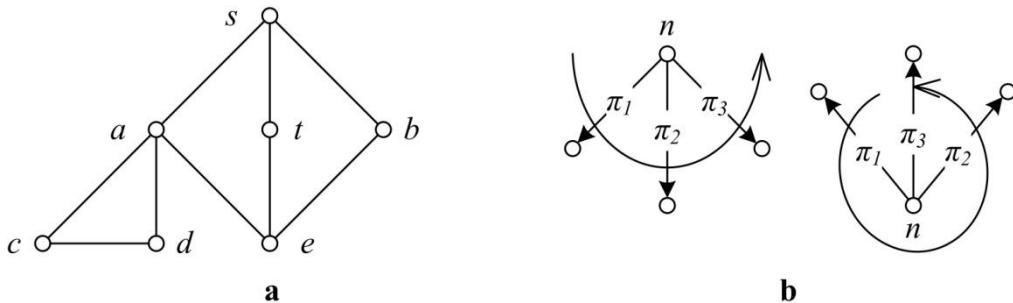
Pavyzdžiais pademonstruosime skirtumą tarp algoritmų BACKTRACK1 ir GRAPHSEARCH\_I\_GYLJ. Kaip minėta anksčiau, BACKTRACK1 įvardinamas sinonimu BACKTRACK\_SU\_STEKU. Čia steko vaidmenį atlieka procedūros parametras DATALIST. Jo tipas yra sąrašas, bet iš rekursijos yra grįztama į ankstesnę sąrašo būseną. Tokiu būdu DATALIST „kvėpuoja“ kaip stekas. Algoritmų skirtumas apibūdinamas dviem teiginiais.

1. Procedūra BACKTRACK1 grindžiama savokomis a) produkcijos ir b) globalios duomenų bazės būsenos, o GRAPHSEARCH\_I\_GYLJ – a) atidarytos viršūnės ir b) uždarytos viršūnės.
2. Procedūra BACKTRACK1 aplink „salas“ eina *keletą* kartų, nes sąraše DATALIST kaip steke saugo tik einamajį kelią. Procedūra GRAPHSEARCH\_I\_GYLJ aplink salą eina tik *vieną* kartą; už laiką ji moka atmintimi.

Skirtingų kriterijų, pvz., laiko ir atminties – pusiausvyra yra informatikos principas.

### 14.1. Paieška grafe

Algoritmų skirtumus nagrinėsime grafe, jau pateiktame 11.1 a pav. Tegu ir produkcių sutvarkymas toks pats. Procedūros BACKTRACK1 paieškos medis parodytas 14.2 a pav.



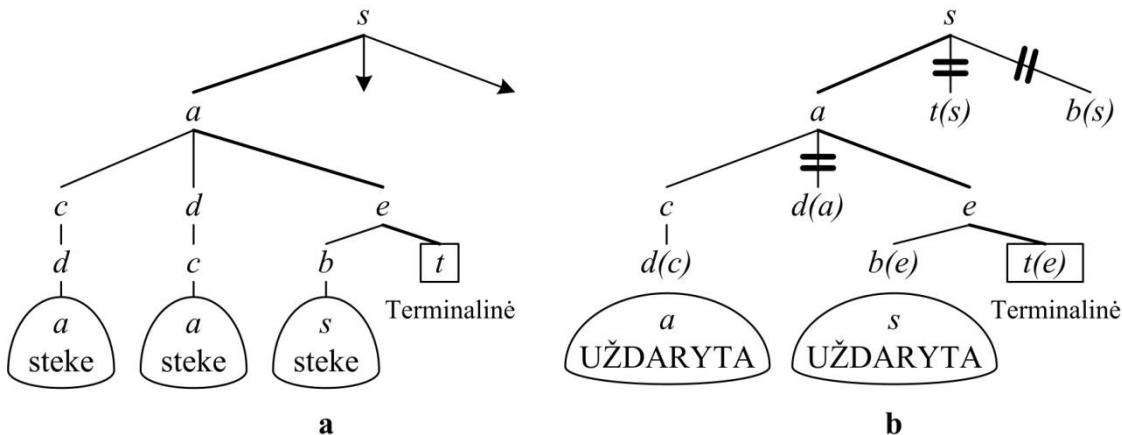
14.1 pav. a) Grafas, kurio briaunos neturi kainų. b) Produkcių sutvarkymas „prieš laikrodžio rodykle“ – „dvylitka valanda“ imama paskutinę

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

Atidarytų ir uždarytų viršunių frontas GRAPHSEARCH\_I\_GYLJ baigus darbą:

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

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



14.2 pav. Paieškos medžiai: a) BACKTRACK1,  
b) GRAPHSEARCH\_I\_GYLJ. Randamas tas pats kelias  $\langle s,a,e,t \rangle$

Nukirtimų lentelėje trys nukirtimai – 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 briaunų: 11 talpinimų į steką, iš jų 3 tikrinimai, ar talpinama viršūnė jau yra steke. GRAPHSEARCH\_I\_GYLJ naudoja daugiau atminties: paieškos medyje (ir fronte) yra 10 viršūnių (tiesa, 3 dubliuojasi). Tuo tarpu BACKTRACK1 steko gylis yra 5.

## 14.2. Pavyzdys paieškai labirinte su ciklais

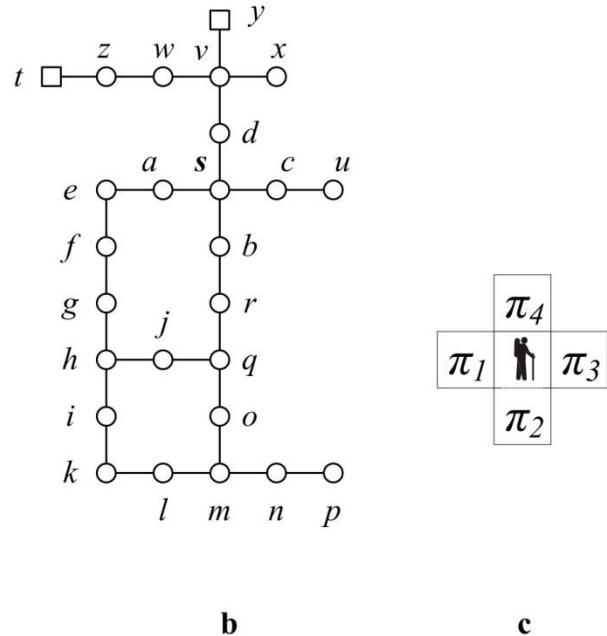
Pateiksime sudėtingesnį pavyzdį. Labirintas gali būti pavaizduotas kaip grafas, kuriame viršūnė atitinka langelį, o briauna tarp viršūnių dedama, kai langeliai yra gretimi. Paieška demonstruojama keliu nuo pradinės viršūnės  $s$  iki krašto labirinte parodytame 14.3 a pav. Šiame labirinto yra du išėjimai – terminalinės viršūnės  $t$  ir  $y$ . Remiantis produkcių sutvarkymu 14.3 c pav. randamas išėjimas per  $t$ , nes keturios produkcijos yra surūšiuotos šia tvarka: „i vakarus“, „i pietus“, „i rytus“ ir „i šiaurę“.

BACKTRACK1 paieškos medis algoritmui baigus darbą yra parodytas 14.4 pav. Šis paieškos medis yra neišreikštinis. Stekui „susitraukiant“, kelią yra „užmirštamas“.

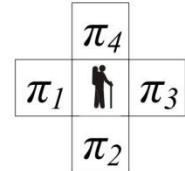
Abu algoritmai nagrinėjamame grafe randa tą patį kelią  $\langle s,d,v,w,z,t \rangle$ . GRAPHSEARCH\_I\_GYLJ paieškos medis algoritmui baigus darbą yra pateiktas 14.5 pav. Pomedžiai žemyn nuo  $j(h)$  ir  $b(s)$  nėra nagrinėjami, nes į sąrašą ATIDARYTA vėliau (t. y. pagal principą „i gylį“) patalpinamos atitinkamai viršūnės  $j(q)$  ir  $b(r)$ . Šių pomedžių nukirtimas ir sudaro esminį GRAPHSEARCH\_I\_GYLJ skirtumą nuo BACKTRACK1.

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

**a**

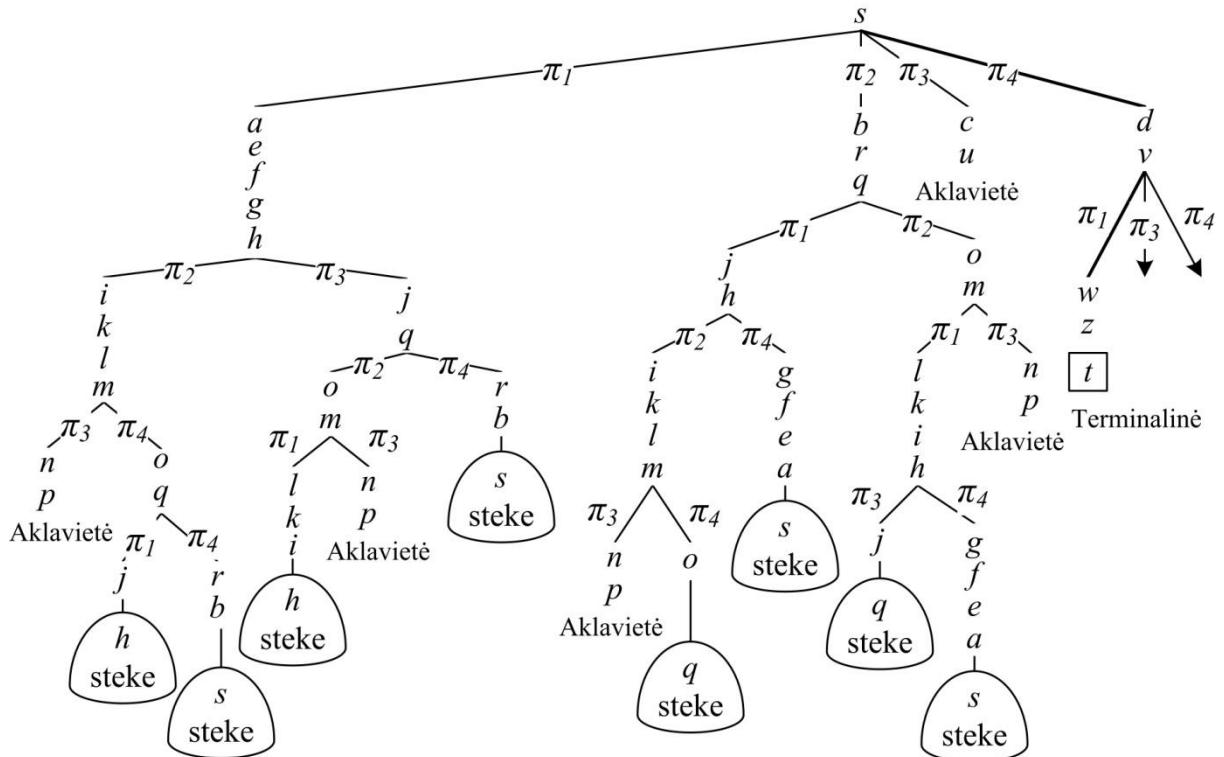


**b**



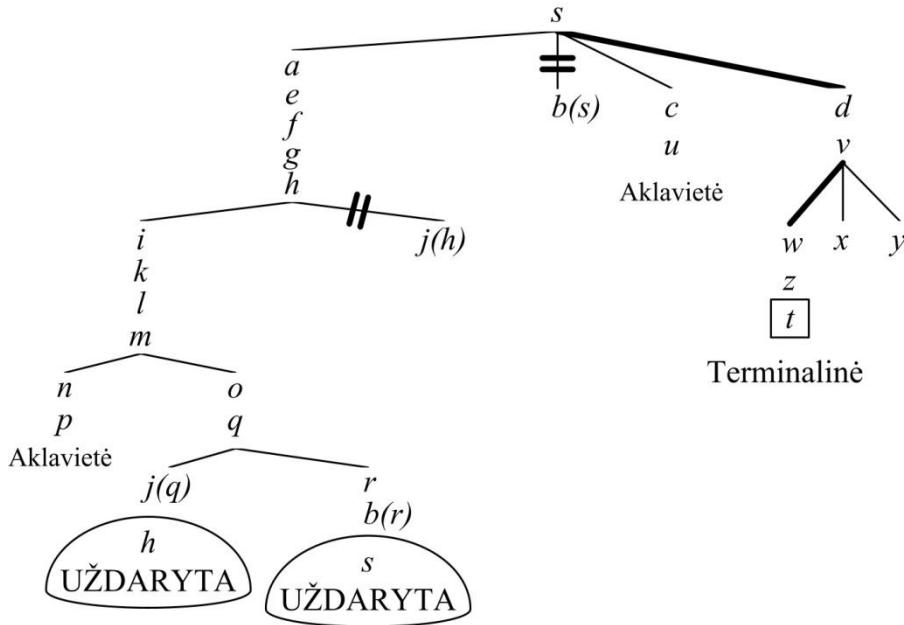
**c**

14.3 pav. a) Labirintas kelio iš *s* paieškai. Yra du išėjimai – *t* ir *y*. b) Labirinto pavaizdavimas grafu, kuriame viršūnė atitinka langelį, o briauna – langelių gretumumo santykį. c) briaunų sutvarkymas



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 bražomas pagal į frontą talpinamų viršūnių nuorodas.



14.5 pav. GRAPHSEARCH\_I\_GYL<sub>I</sub> paieškos medis. Jis išreikštinis

Toliau lentelėje pateikiamas GRAPHSEARCH\_I\_GYL<sub>I</sub> 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 $j(q), r(q) \in S(q)$ . Reikia pastebėti, kad $j$ jau yra ATIDARYTA kaip $j(h)$ . Pagal principą „i gylį“ yra perorientuojama $j$ nuoroda: <b>pasirenkama</b> 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 pastebėti, kad $b$ jau yra ATIDARYTA kaip $b(s)$ . Pagal principą „i gylį“ yra perorientuojama $j$ nuoroda: <b>pasirenkama</b> 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	$s(start) \{a(s) b(s) e(s) d(s)\} e(a) f(e) g(f)$ $h(g) \{i(h) j(h)\} k(i) l(k) m(l) \{n(m) o(m)\}$ $p(n) q(o) \{j(q) r(q)\} b(r) u(e) v(d) \{w(v)\}$ $x(v) y(v)\} z(w) t(z)$	Uždaroma $z(w)$ ir atidaromas jos vaikas $t \in S(z)$
25	$s(start) \{a(s) b(s) e(s) d(s)\} e(a) f(e) g(f)$ $h(g) \{i(h) j(h)\} k(i) l(k) m(l) \{n(m) o(m)\}$ $p(n) q(o) \{j(q) r(q)\} b(r) u(e) v(d) \{w(v)\}$ $x(v) y(v)\} z(w) t(z)$	Uždaroma $t(z)$ ir pastebima, kad ji yra terminalinė. Surenkamas kelias: $t$ pasiekti 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

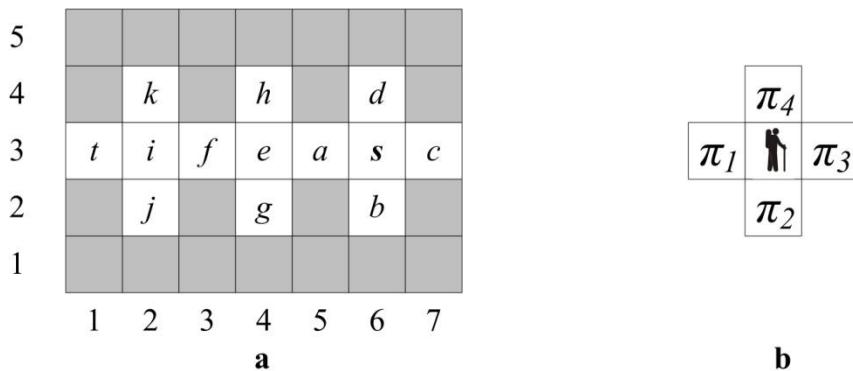
14.6 lentelė. Algoritmo GRAPHSEARCH\_I\_GYL iteracijos

Nukirtimų lentelė – nuorodos keistos du kartus:

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

### 14.3. Kontrpavyzdys, kai pranašesnis BACKTRACK1

Ankstesčiau pavyzdžiai buvo pademonstruotas skirtumas tarp algoritmų BACKTRACK1 ir GRAPHSEARCH\_I\_GYL. Pastarasis pranašesnis ta prasme, kad jo paieškos medis (ir tokiu būdu vykdymo laikas) yra mažesnis. Tačiau tai pasiekiamā 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 kainos yra tokios:

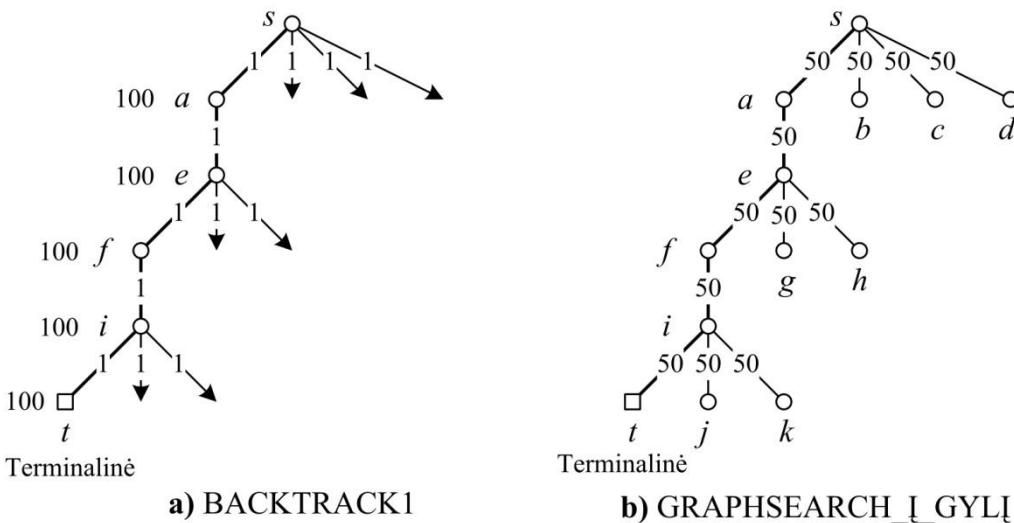
- 1 – už **pasižiūrėjimą**, t. y. produkcijos patalpinimą į sąrašą RULES su funkcija APPRULES (žr. 2 skyrių);
- 100 – už **patalpinimą** į steką. Tai kaina už „siūlą“ ar „grūdo“ padėjimą langelyje.

Tegu GRAPHSEARCH\_I\_GYLĮ kainos yra tokios:

- 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\_I\_GYLĮ. BACKTRACK1 paieškos medžio kaina yra  $5 \cdot 100 + 3 \cdot 1 + 2 \cdot 1 + 2 \cdot 1 = 512$  (14.8 a pav.), o GRAPHSEARCH\_I\_GYLĮ –  $12 \cdot 50 = 600$  (14.8 b pav.).

Šiame pavyzdyste procedūrai GRAPHSEARCH\_I\_GYLĮ, kurios įkainis 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“ (1·100). GRAPHSEARCH\_I\_GYLĮ iš kartoj atidaro  $n$  viršūnių ( $n \cdot 50$ ). Pasižiūrėjimas yra pigus, bet „grūdas“ yra brangus; jo kaina daug didesnė už pasižiūrėjimą.



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

### Pratimas

Kuris iš GRAPHSEARCH\_I\_GYLJ ir BACKTRACK1 duoda geresnį rezultatą pagal laiką ir atmintį kelio paieškai iš *s* iki krašto labirinte žemiau? Nubraižykite paieškos medžius su kainomis: pažiūrėjimas – 1, „grūdo“ padėjimas – 100 ir viršūnės atidarymas – 50.

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

## 15. Valdymo metodas „kopimas į kalną“

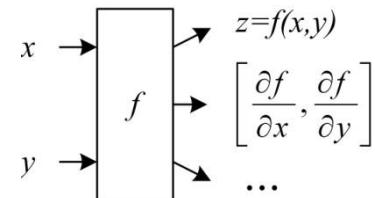
„Kopimo į kalną“ esmė: iš *lokaliros informacijos* daryti prielaidas apie *globalią informaciją* – apie sprendinį. Dirbtiniame intelekte jis nejprastas dėl šių priežasčių: 1) lokalaus maksimumo problemos ir 2) plató. Šis metodas ateina iš matematikos.

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

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 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 stačiakampėje srityje  $x \in [a_1, b_1]$ ,  $y \in [a_2, b_2]$ . Agentas kopia didžiausio statumo 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 taške, kuriame stovi(15.1 pav.).

15.1 pav. Sprendžiantis agentas vadovaujasi lokalia informacija  $f(x,y)$  savybes taške, kuriame stovi.

Priklasomai nuo agento intelekto be  $f$  reikšmės jis gali apskaičiuoti didžiausio gradiento kryptį ir kitą lokalią informaciją, pvz., antrają išvestinę



Gradiento vektorius pasirenkamas eiti stačiausia kryptimi aukštyn. Toks vektorius yra vienas iš  $360^\circ$  agento horizonto – liečiamosios plokštumos tame funkcijos  $f(x,y)$  taške. Tokiu būdu agentas turi pasirinkti, kuriuo azimutu nuo  $0^\circ$  iki  $360^\circ$  eiti (15.2 pav.).

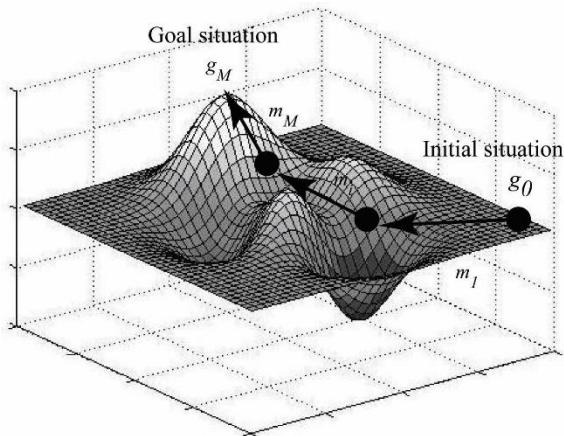
Darydamas žingsnius  $\langle [x_0, y_0], [x_1, y_1], [x_2, y_2], \dots \rangle$  didžiausio gradiento kryptimis agentas gali pasiekti ir *lokalų* maksimumą, t. y. užkopti į žemesnę viršūnę, negu absolutus maksimumas. Tokiu būdu gradientinis metodas garantuoja absolutaus maksimumo pasiekimą tik „gražioms“ funkcijoms – tolydžioms, turinčioms tolydžias išvestines, turinčioms tik vieną maksimumą apibrėžimo srityje ir kt.

Be lokalaus maksimumo, agentas susiduria su kitu pavojumi – plató. Atsidūrės ant plokštikalnės agentas nežino, kuriuo azimutu eiti, nes yra „aklas“, bet gali „paliesti“ reljefą.

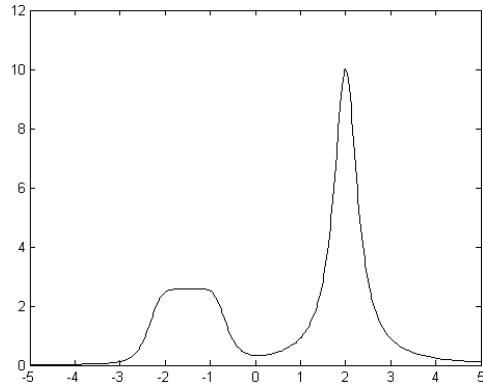
Vieno argumento atveju,  $z=f(x)$ , agento pasirinkimas daug paprastesnis. Jis renkasi tarp dviejų krypčių –  $x$  didėjimo ar mažėjimo (15.3 pav.).

Gradientiniai metodai yra taikomi „geroms“ funkcijoms. Reikalavimai „geroms“ funkcijoms būtų šie: 1) funkcija tolydi, 2) turi tolydžią išvestinę kiekviename taške iš intervalo  $[a,b]$ , 3) išvestinė turi vienodą ženklą (arba neteigiamo, arba neneigiamo) intervale ir pan. Tokiomis savybėmis paprastai pasižymi funkcijos, konstruojamos iš elementarių funkcijų tokiu kaip polinomas, eksponentė, trigonometrinės funkcijos ir kt.

Dirbtinio intelekto sistemose yra išskiriama dvi pagrindinės valdymo strategijos: negrijtamoji (*irrevocable*) ir grīztamoji (*tentative*). Taikant negrijtamąją strategiją, produkcija yra pasirenkama ir pritaikoma negrijtamai, t. y. be galimybės grīžti į ankstesnę būseną. „Kopimas į kalną“ yra viena iš negrijtamujų strategijų. Ir priešingai, „valdymas su grīžimais“ BACKTRACK yra grīztamosios strategijos pavyzdys.



15.2 pav. Kopimas į kalną  $\langle [x_0, y_0], [x_1, y_1], [x_2, y_2], \dots \rangle$  kaip funkcijos  $z=f(x, y)$  maksimizavimas



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

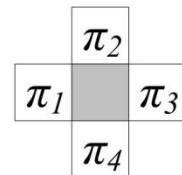
„Kopimą į kalną“ iliustruojame paieška 8 kauliukų galvosūkyje (8-puzzle). Apibrėžkime funkciją  $f(n)$  nuo globalios duomenų bazės būsenos  $n$  tokiu būdu. Tegu  $f(n) = -w(n)$ , kur  $w(n)$  yra skaičius kauliukų, kurie ne savo vietoje lyginant su terminaline būseną. Pavyzdžiui, funkcijos reikšmė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

2	8	3
1	6	4
7		5

Pradinė

1	2	3
8		4
7	6	5

Terminalinė



Produkcijos

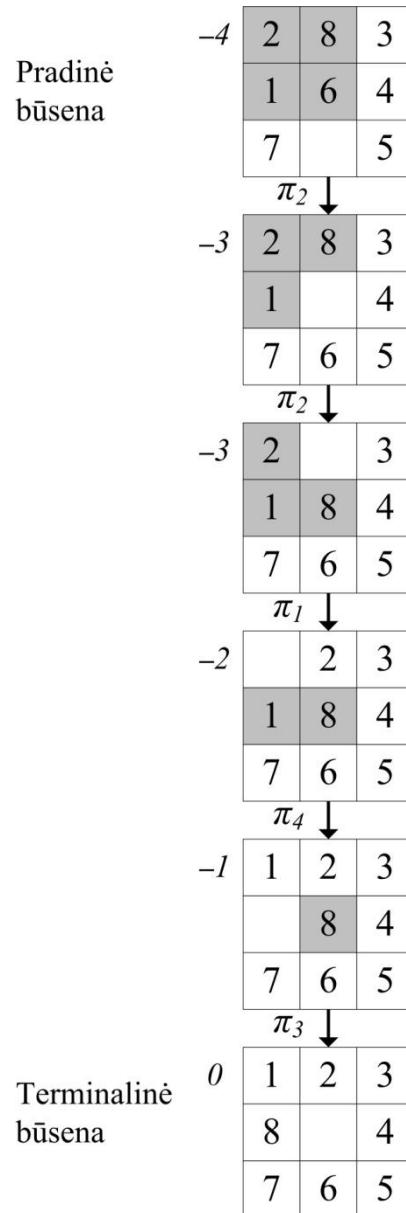
15.4 pav. Pradinės būsenos pavyzdys, terminalinė būsena ir 4 produkcijos (tuščio lango ėjimai  $\pi_1, \pi_2, \pi_3, \pi_4$ ) 8 kauliukų galvosūkyje

Pradinėje būsenoje funkcijos reikšmė yra blogiausia. Pereinant į kitas būsenas siekiama padidinti  $f$  reikšmę, t. y. ją maksimizuoti ir pasiekti terminalinę būseną  $f(n) = 0$ . Einama pastumiant kurį nors kauliuką į tuščią langelį. Toks kauliuko stūmimas yra suprantamas kaip tuščio langelio stūmimas priešinga kryptimi. Tegu keturios produkcijos  $\pi_1, \pi_2, \pi_3, \pi_4$  yra sutvarkytos, kaip parodyta 15.4 pav.

15.5 pav. rodo  $f$  kilimą iš pradinės būsenos iki terminalinės, kurioje  $f(n) = 0$ . Funkcijos  $f$  reikšmė yra nurodyta prie kiekvienos būsenos. 15.6 pav. rodo paieškos į plotį medį „kopimo į kalną“ metode, kai atsižvelgiama į viršūnės gylį rūšiuojant viršūnes sąraše ATIDARYTA. Šiuo atveju  $f(n) = w(n) + d(n)$ , kur  $d(n)$  yra būsenos gylis. Ši  $f$  minimizuojama. Kuo būsena giliau, tuo ji blogesnė. Paieškos medis ir frontas su uždaromomis viršūnėmis pagrindžia antrosios produkcijos pasirinkimą iš  $\pi_1$  ir  $\pi_2$  gylje  $d = 2$ .

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

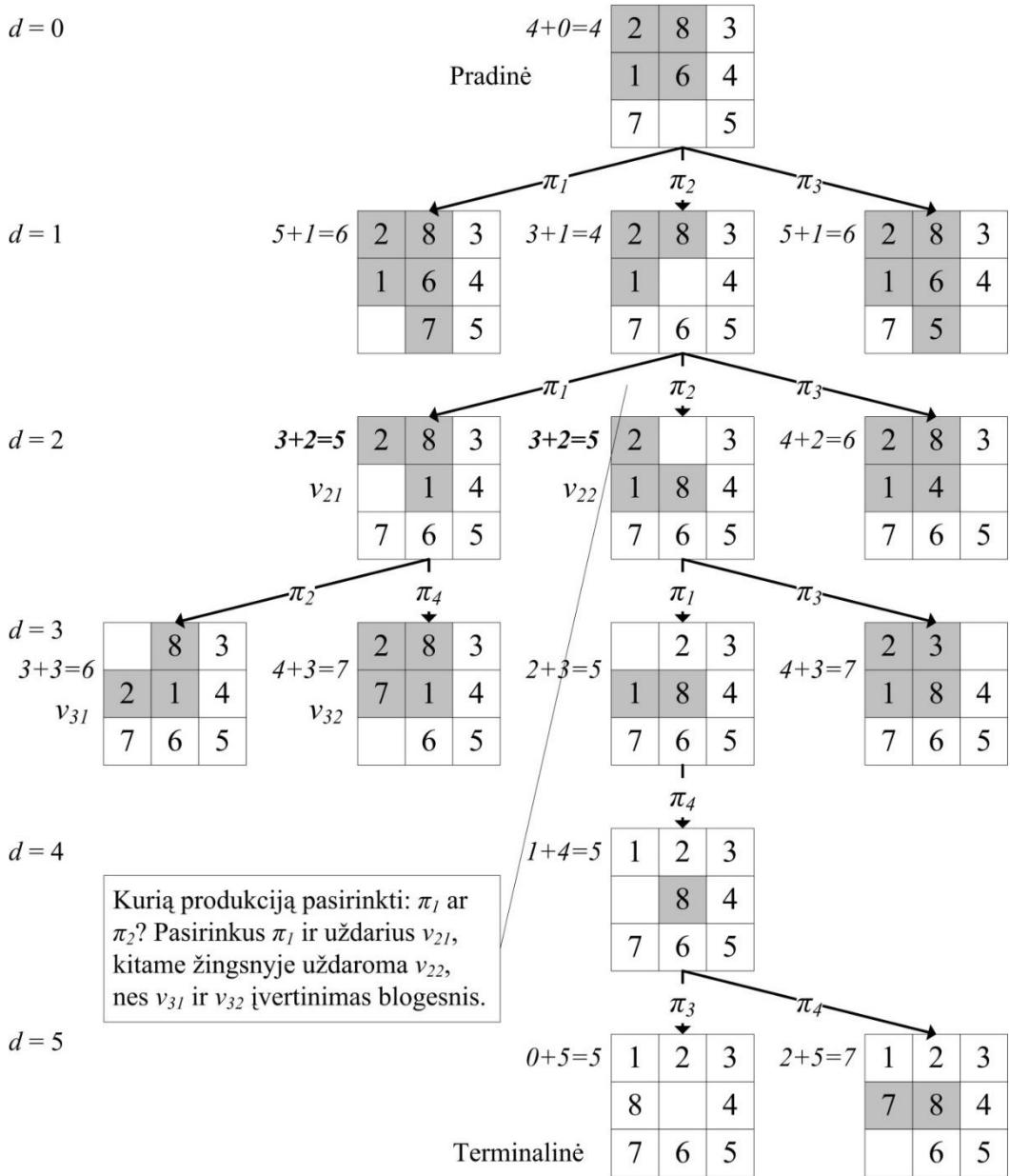
Lokalaus maksimumo situacija pateikta 15.7 pav. Kiekviename produkcijos perveda į būseną, kurioje funkcijos reikšmė yra blogesnė negu pradinėje.



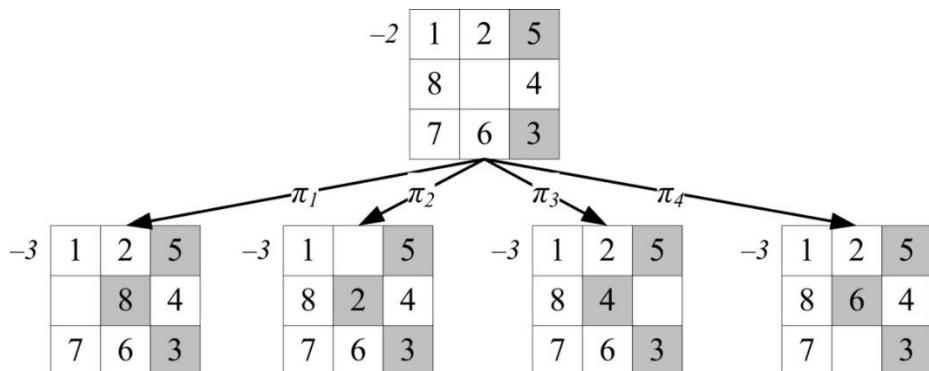
15.5 pav. Funkcijos  $f(n) = -w(n)$ , kur  $w(n)$  yra skaičius kauliukų ne savo vietoje, maksimizavimas 8 kauliukų žaidime

Funkcijos  $f$  lokalias informacijos esmė: tai funkcijos savybės viename taške, pagal kurias sprendžiama apie sprendinio globalias savybes. Tai euristinė funkcija. Tokios funkcijos atradimas gali būti suprantamas kaip intelektualus gebėjimas.

Funkcija  $f(n) = w(n) + d(n)$ , kur  $d(n)$  yra būsenos gylis paieškos medyje, yra pateikiama (Nilsson 1982, 2.4.1 sk.) kaip pavyzdys euristinės funkcijos, skirtos būsenos įvertinimui procedūroje GRAPHSEARCH. Paėmus  $f(n)=d(n)$  yra gaunama paieška į plotį.



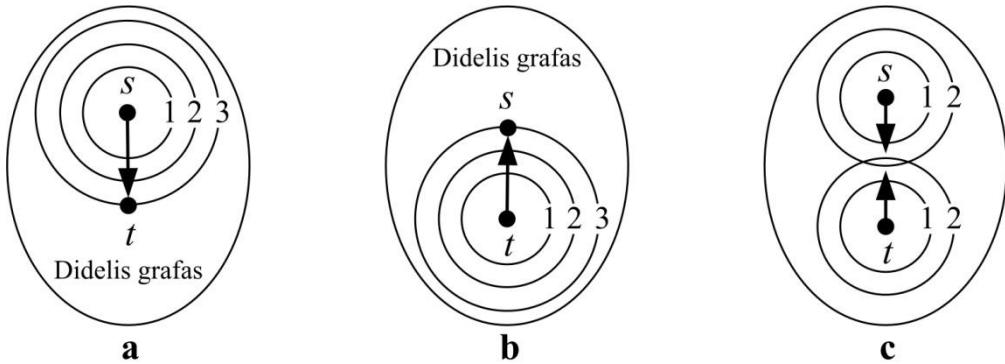
15.6 pav. Funkcija  $f(n) = w(n)+d(n)$ , kur  $w(n)$  yra kauliukų skaičius ne savo vietoje, o  $d(n)$  – būsenos gylis. Funkcija  $f$  yra minimizuojama



15.7 pav. Pradinė būsena yra lokaliame maksimume, nes kiekviena produkcija perveda į blogesnę būseną

## 16. Manheteno atstumas

Tegu turime didelį grafą ir paiešką į plotį keliui iš pradinės viršūnės  $s$  į terminalinę  $t$ . Pirmaja banga atidarome viršūnes, nutolusias per vieną briauną nuo  $s$ , antraja – viršūnes per dvi briaunas nuo  $s$ , ir t. t. (16.1 a pav.). Kai atidaroma  $t$ , tai kelias rastas; belieka ji surinkti.



16.1 pav. Kelio paieška į plotį grafe skleidžiant bangas: a) iš  $s$  iki  $t$ ; b) iš  $t$  iki  $s$ ; c) lygiagrečiai dvi bangos – iš  $s$  ir iš  $t$ . Skaičiai – bangų numeriai

Galime pradėti ir kita kryptimi, nuo terminalinės  $t$  (16.1 b pav.). Kyla klausimas – kaip skleisti bangas geriau? Atsakymas: skleiskime dvi bangas lygiagrečiai – iš  $s$  ir iš  $t$  (16.1 c pav.). Kai bangos „susitinka“, tai kelias rastas. Šitas būdas yra geresnis, nes mažiau viršūnių fronte (atidaromų kartu su uždaromomis).

**Irodymas.** Tegu kelio ilgis yra  $N$  briaunų. Skleidžiant bangą nuo  $s$ , viršūnių skaičius fronte bus  $O(N^\alpha)$ , kur paprastai  $\alpha > 2$ . Skleidžiant nuo  $t$  iki  $s$ , įvertinimas yra toks pat –  $O(N^\alpha)$ . Jeigu skleidžiamos dvi bangos (iš  $s$  ir iš  $t$ ) tai bangos susitiks greičiau: po  $N/2$  žingsnių, o ne po  $N$ . Todėl viršūnių skaičius fronte yra mažesnis (tiesa, polinomiškai, o ne eksponentiškai):  $O(N^\alpha) > O((N/2)^\alpha) + O((N/2)^\alpha)$ . Pavyzdžiui, 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  $N^3 > (N/2)^3 + (N/2)^3 = N^3/8 + N^3/8 = N^3/4$ .

Dviejų bangų skleidimą iliustruojame grafe, kurio viršūnes yra dvimatės erdvės taškai su sveikomis koordinatėmis (Manheteno grafas). Šitaip gatvių tinklo briaunos vaizduoja gatves, o viršūnės – sankryžas. Ieškosime kelio tarp  $s$  ir  $t$ . Pradedame nuo  $s$ . Pirmaja banga pasiekiamiame 4 viršūnes, antraja 8 ir t. t. kol pasiekiamiame  $t$  (16.2 pav.).

Viršūnių skaičius kiekvienoje bangoje (fronte 145 viršūnės) parodytas 16.3 a lentelėje:

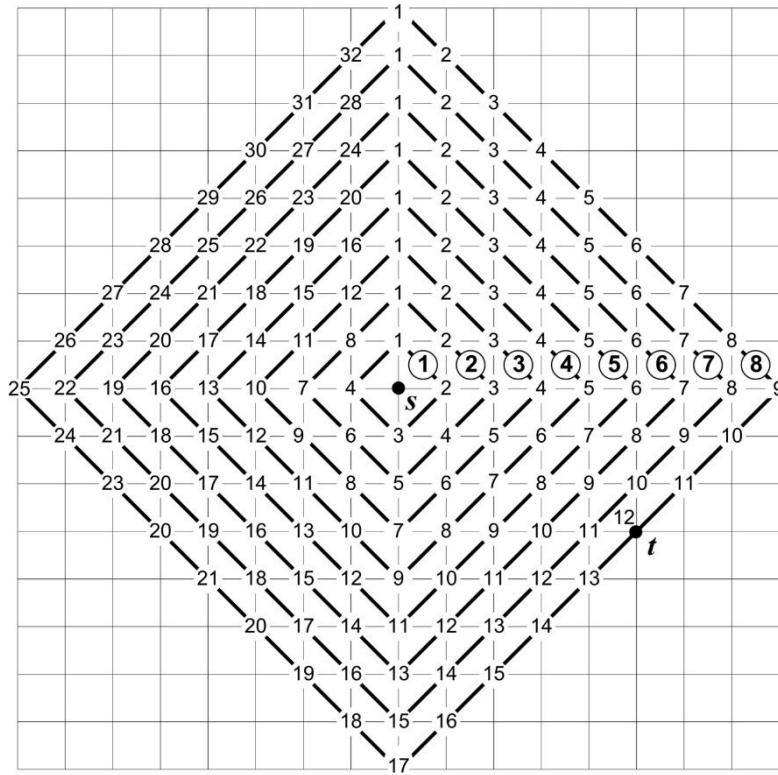
Bangos nr.	Viršūnių skaičius
0	1 + 0
1	4
2	8
3	12
4	16
5	20
6	24
7	28
8	32
Iš viso	145

a

Bangos nr.	Viršūnių skaičius
0	1 + 1 + 0 + 0
1	4 + 4
2	8 + 8
3	12 + 12
4	16 + 16
Iš viso	82

b

16.3 lentelė. Viršūnių skaičius, kai skleidžiama: a) viena banga; b) dvi bangos



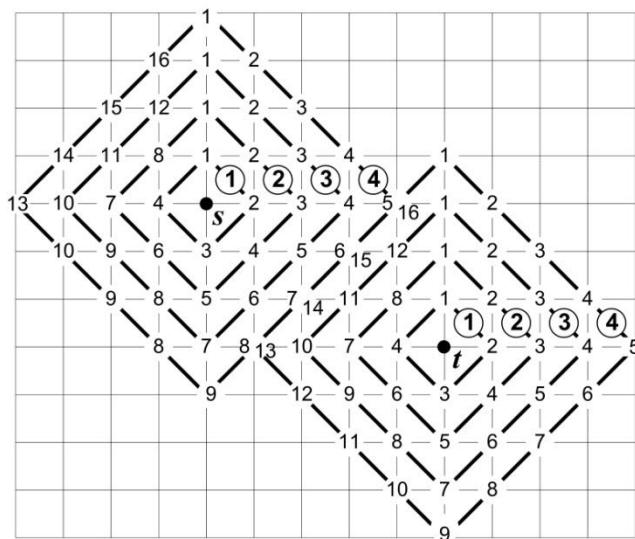
16.2 pav. Iš  $s$  į  $t$  reikia 8 bangų. Kelyje 5 produkcijos „i rytus“ ir 3 „i pietus“

Kelio ilgis tarp  $s$  ir  $t$  yra  $5 + 3 = 8$  briaunos, todėl prireiks 8 bangų. Pirmoje bangoje 4 viršūnės, antroje 8 ir t.t. aritmetine progresija; bangoje  $N$  yra  $4 \cdot N$  viršūnių. Todėl pagal aritmetinės progresijos formulę po  $N$  bangų iš viso bus atidaryta tiek viršūnių:

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

$N$ -taja banga apribotame kvadrate yra  $2 \cdot (N^2 + N) + 1$  viršūnių (pridedama ir pradinė). Todėl 8 briaunų keliui rasti atidaryti daugiausiai 145 viršūnes.

Dabar pradékime paiešką dviem bangomis. Grafas ir bangos yra parodytos 16.4 pav. Viršūnių skaičius kiekvienoje bangoje (iš viso 82 viršūnės) parodytas 16.3 b lentelėje.



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

## 17. Algoritmas A\*

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

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

Čia  $g(n)$  yra trumpiausias kelias nuo pradinės viršūnės iki  $n$ , o  $h(n)$  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 (ir sprendėjas, ir planuotojas) nemato viso žemėlapio ir vadovaujasi euristine funkcija  $h(n)$ .

Algoritmą A\* N. Nilsonas su bendrautoriais publikavo 1968 m, o vėliau pateikė vadovėlyje (Nilsson 1982); žr. taip pat kituose DI vadovėliuose, pvz., (Russell, Norvig 2003; Luger 2005) ir Vikipedijoje ([http://en.wikipedia.org/wiki/A\\*\\_search\\_algorithm](http://en.wikipedia.org/wiki/A*_search_algorithm)). Algoritmas iš pradžių įvardijamas A, o vėliau dėl tam tikro optimalumo pavadinamas A\*.

Dirbtiniame intelekte skiriama dvi paieškos strategijos: informuota paieška ir neinformuota paieška. *Neinformuota paieška* – kai algoritmu nėra žinoma globali informacija apie sprendinį, t. y. nei žingsnių skaičius, nei kelio įvertinimas nuo einamosios būsenos iki terminalinės. Pavyzdžiai: paieška „i gylį“ bei „i plotį“ ir grafe be kainų ir su kainomis.

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

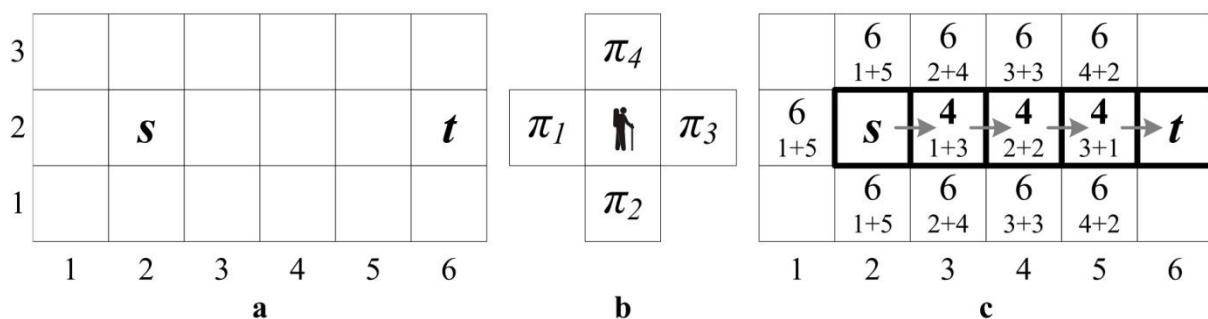
Informuota paieška yra efektyvesnė. Paieškos medyje yra mažiau viršūnių, bet efektyvumas pasiekiamas euristinio įvertinimo kaina.

### 17.1. A\* pavyzdys su euristine funkcija $h(n)$ Manheteno atstumu

Langelių lentoje agentas keliauja iš pradinio langelio  $s$  į terminalinį  $t$  (17.1 pav.). Lentoje gali būti kliūčių. 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 įėjimai (produkcijos)  $\{\pi_1, \pi_2, \pi_3, \pi_4\}$  yra surūšiuoti kaip parodyta 17.1 b pav. Manheteno atstumas  $d(n_1, n_2)$  tarp langelių  $n_1=(x_1, y_1)$  ir  $n_2=(x_2, y_2)$  yra apibrėžiamas šitaip:

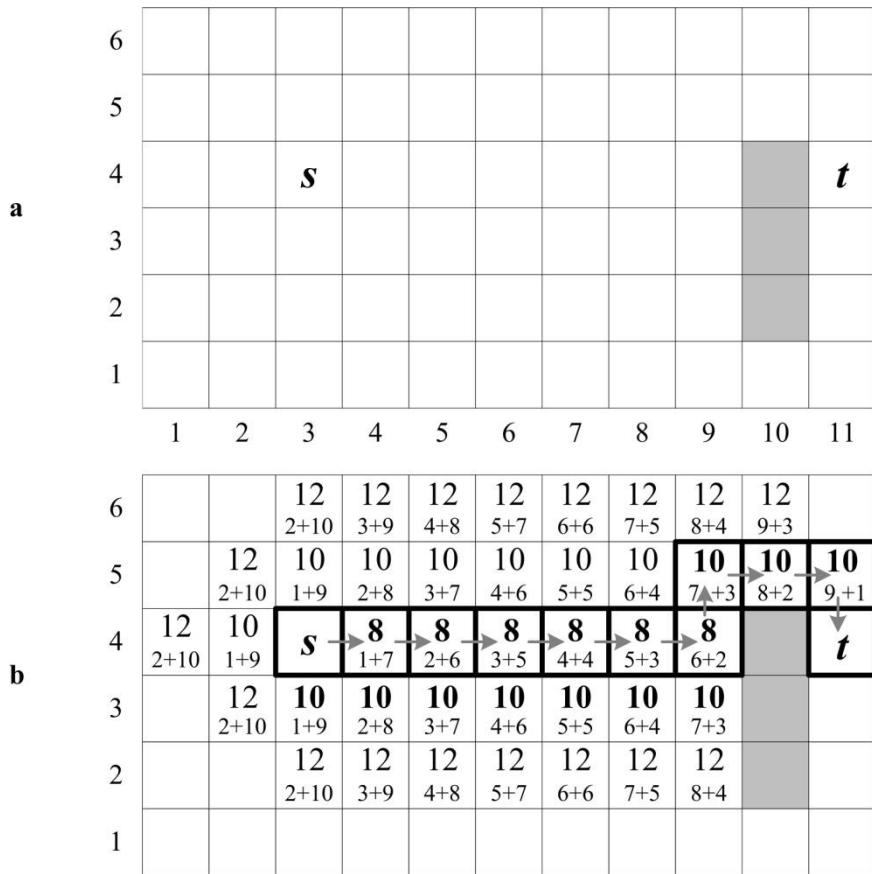
$$d(n_1, n_2) = |x_2 - x_1| + |y_2 - y_1|$$

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



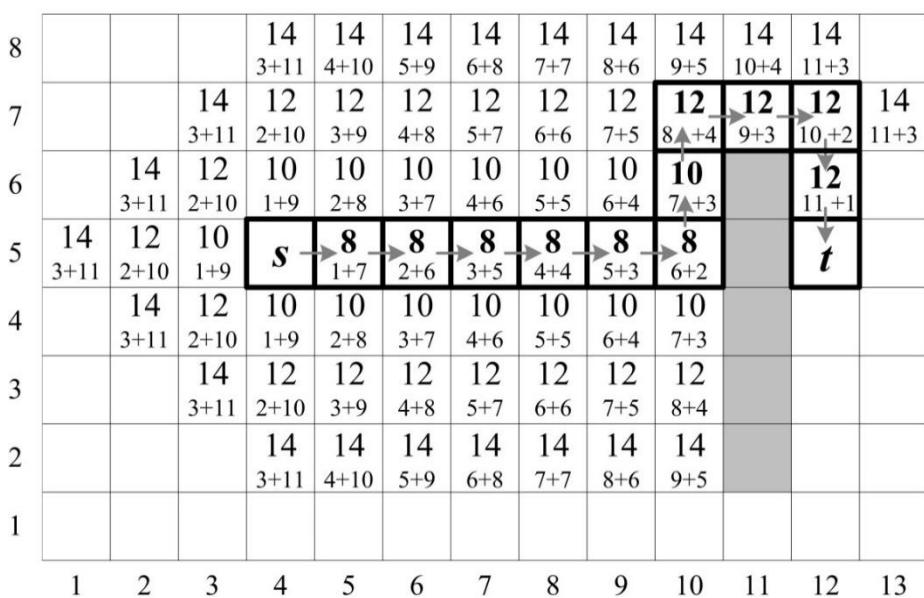
17.1 pav. a) Agentas keliauja iš  $s$  į  $t$ . b) Keturi įėjimai. c) Uždaromos ir atidaromos viršūnės (frontas) bei rastas kelias iš  $s$  į  $t$

Toliau pavyzdys, kai lentoje yra kliūčių (langelių, kur negalima žengti) (žr. 17.2 pav.).



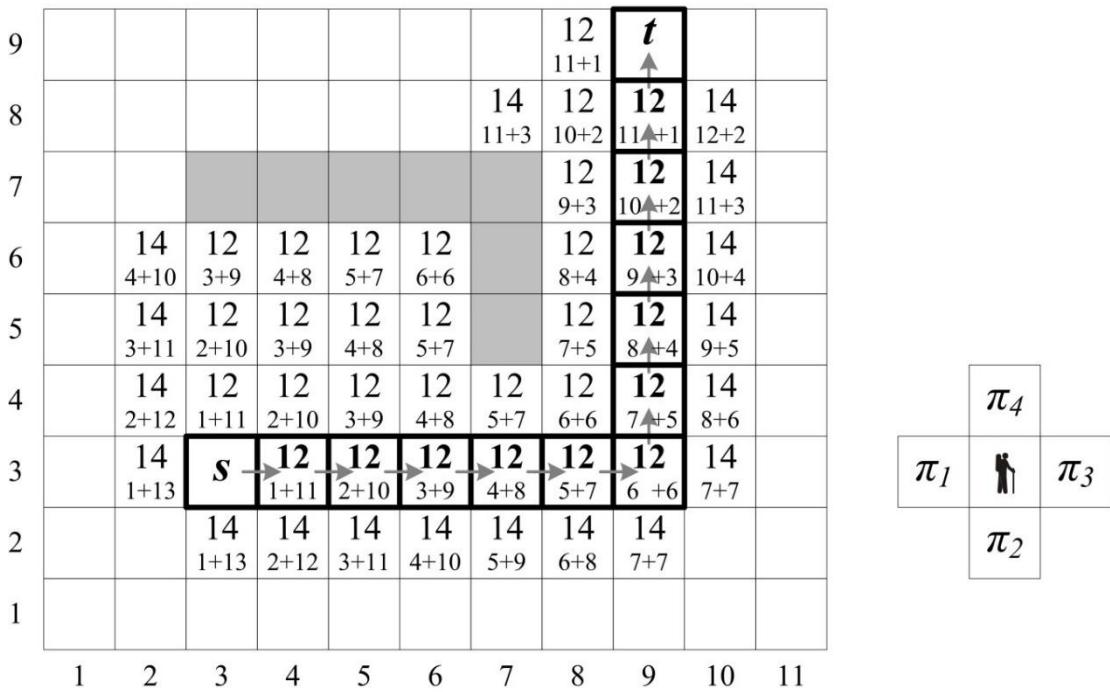
17.2 pav. a) Agentas keliauja iš *s* į *t*. Paryškintos kliūtys.  
b) Uždaromos ir atidaromos viršūnės (frontas) bei rastas kelias

Toliau 17.3 pav. pateikiamas pavyzdys su ilgesne kliūtimi.



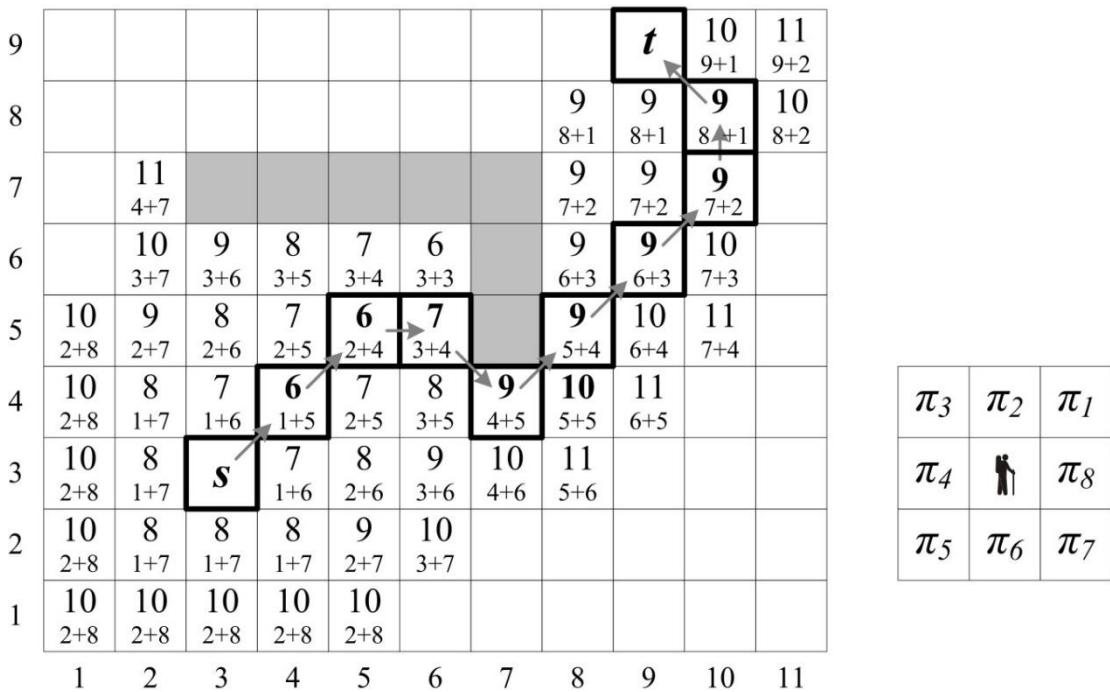
17.3 pav. Uždaromos ir atidaromos viršūnės bei rastas kelias iš *s* į *t*

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



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

Toliau 17.5 pav. pavyzdys, kai ieškomos kelias tokiai pat labirinte kaip aukščiau 17.4 pav., bet agentas turi 8 produkcijas, t. y. gali eiti ir įstrižainėmis. Rekomenduojame Vikipedijoje pasižiūrėti panašų dar didesnį pavyzdį, kuriame frontas animuojamas skirtingais laiko momentais; žr. [http://en.wikipedia.org/wiki/A\\*\\_search\\_algorithm](http://en.wikipedia.org/wiki/A*_search_algorithm).



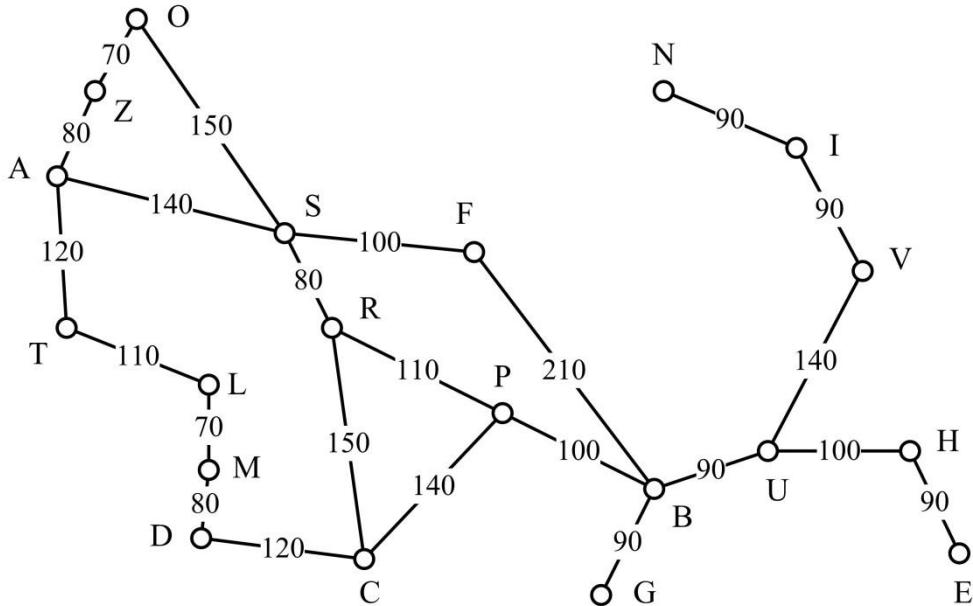
17.5 pav. Kelias iš  $s$  į  $t$ , kai yra aštuonios produkcijos – einama ir įstrižainėmis

Kaip matyti iš visų pavyzdžių 17.1–17.5 pav., banga turi „elipsės“ nuo  $s$  link  $t$  formą. Šitaip banga skleidžiama efektyviau, t. y. fronte yra žymiai mažiau viršūnių, negu skleidžiant apskritimais. Koncentriniai apskritimai gaunami, kai  $f(n)=g(n)$ , t. y. neatsižvelgiant į euristinę funkciją  $h(n)$  – terminalinio lanelio padėtį. Apskritimai transformuojami į kvadratus, kai imamas Manheteno, o ne Euklido atstumas („oru“) (žr. ankstesniame skyriuje 16.2 pav.).

Algoritmas A\* yra optimalus, kai euristinė funkcija  $h(n)$  yra optimistinė (Russell, Norvig 2003). Euristika vadinama *optimistinė*, jeigu jos duodamas atstumo ivertis yra ne didesnis negu tikras atstumas. Agento kambaryje 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, 4 sk., p. 94–98). Ieškomas kelias iš viršūnės (miesto) A į B grafe (kelių žemėlapyje) 17.6 pav.



17.6 pav. Rumunijos kelių žemėlapis. Ieškomas kelias iš miesto A į B; žr. (Russell, Norvig 2003, p. 63, 3.2 pav.)

Euristinės funkcijos  $h(n)$  reikšmės pateikiamos 17.7 lentelėje. Tai Euklido atstumas (tiesia linija, „oru“) nuo  $n$  iki B.

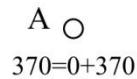
Miestas	$h(n)$
A	370
B	0
C	160
D	240
E	160
F	180
G	80
H	150
I	230
L	240

Miestas	$h(n)$
M	240
N	230
O	380
P	100
R	190
S	250
T	330
U	80
V	200
Z	370

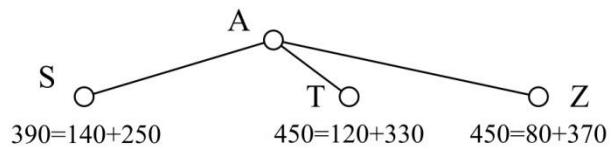
17.7 lentelė. Euristinės funkcijos  $h(n)$  reikšmės. Tai atstumas tiesia linija nuo miesto  $n$  iki galinio miesto B

Algoritmo A\* paieškos medžiai skirtingais žingsniais pateikiami 17.8 pav. Funkcija  $g(n)$  apskaičiuojama pagal tikrus atstumus tarp miestų 17.6 pav.

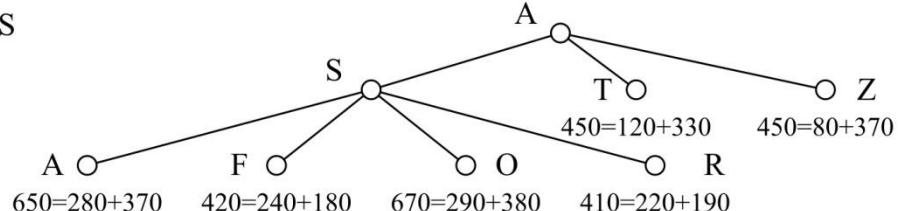
a) Pradinė būsena



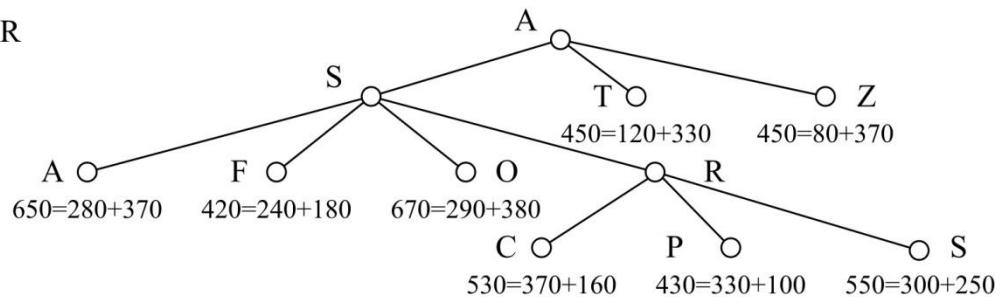
b) Išskleidus, t. y. uždarius A



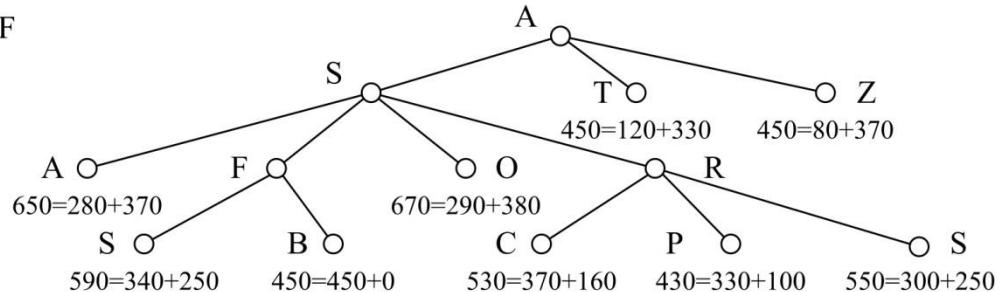
c) Uždarius S



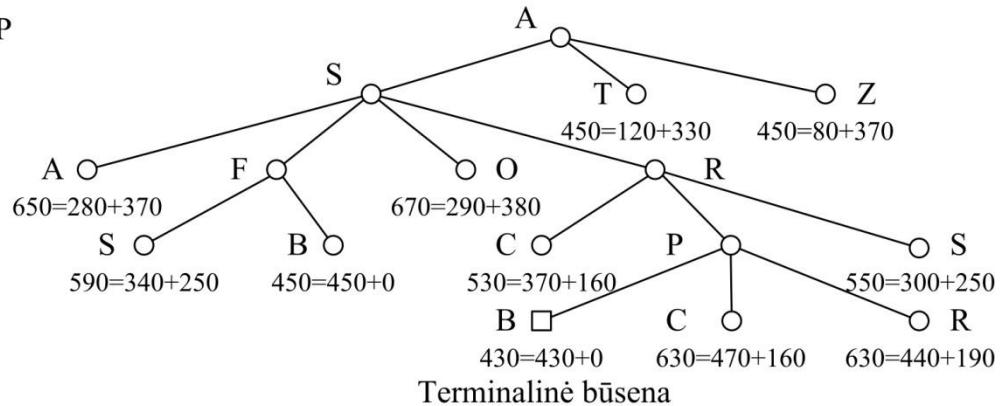
d) Uždarius R



e) Uždarius F



f) 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 produkcinių taisyklių atveju; žr. taip pat (Negnevitsky 2011). Paimkime produkcijas:

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

Produkcijos semantika apibrėžiama šitaip. Jeigu globalios duomenų bazės būsenoje produkcijos visi propoziciniai (loginiai) kintamieji iš kairiosios pusės ( $\pi_1$  atveju, tai F ir B) turi reikšmes *true* tai produkciją galima taikyti. Pritaikius produkciją, nauja GDB būsena papildoma propoziciniu kintamuoju iš dešiniosios pusės ( $\pi_1$  atveju, tai Z). Pasakymas „yra turimas objektas A“ reiškia, kad dvejetainio kintamojo A reikšmė yra *true*. Kitaip tariant, turimas faktas A.

Bendru atveju, produkcijų aibė yra sudaryta iš  $N$  produkcijų. Kairiąjį produkcijos pusę sudaro propozicinių kintamųjų vardai iš tam tikros vardu aibės, dar vadinamos *alfabetu*. Dešiniąjį pusę sudaro vienas vardas. Jeigu dešiniąjį pusę sudarytų keli propoziciniai kintamieji, pavyzdžiui,  $\pi_j: A, B, C \rightarrow J, K$ , tai produkciją būtų gali būti perrašyta į dvi:

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

Šitaip perrašant gaunamas ir kiek kitoks modeliavimas. Kablelis produkcijoje yra suprantamas kaip konjunkcijos operacija  $\&$ , pvz.,  $\pi_j: A \& B \& C \rightarrow J \& K$ . O dvi produkcijos suprantamos kaip disjunkcija  $\vee$ , nes tampa galima taikyti vieną, kitą arba abi.

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

Uždavinys yra formuluojamas taip: rasti produkcijų seką, kuri pradinę GDB būseną perveda į terminalinę būseną. Terminalinė būsena yra bet kokia būsena, kuriai priklauso tikslø kintamasis (šiame pavyzdyste Z).

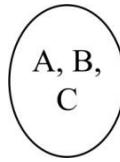
Produkcijų sekai gauti aptariami du algoritmai: tiesioginis išvedimas (*forward chaining*) ir atbulinis išvedimas (*backward chaining*). Tiesioginis išvedimas juda nuo faktų prie tikslø, o atbulinis išvedimas – nuo tikslø prie faktų.

Šiuos abu algoritmus web servisais realizavo Julius Valkauskas. Serverinė dalis yra <http://juliuschainingexample.appspot.com/>. Klientinė dalis yra <http://juliuschainingwsclient.herokuapp.com/>. Į pastarąjį copy-paste būdu kopijuojamas serverinės dalių rezultatas JSON formatu.

### 18.1. Tiesioginio išvedimo algoritmas

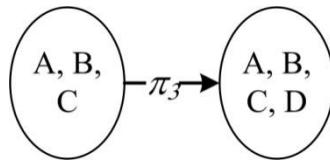
Pradedama nuo pradinės GDB būsenos, t. y. nuo faktų. Produkcijos perrenkamos iteracijomis. Kiekvienoje iteracijoje produkcijos perrenkamos iš eilės. Jeigu paimtą produkciją galima taikyti, tai ji yra taikoma ir pažymima, kad daugiau nebūtų taikoma kitose iteracijose; valdymas perduodamas kitai iteracijai. Priešingu atveju imama kita produkcija. Jeigu produkcijos išsemtos, tai nesėkmė – ieškoma produkcijų seka neegzistuoja.

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



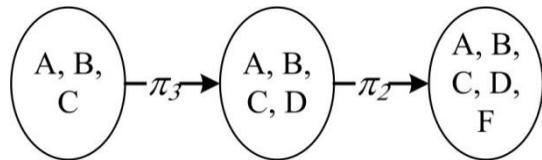
18.1 pav. Pradinė globalios duomenų bazės būsena  $\{A, B, C\}$

**1 iteracija.** Esamoje būsenoje  $\{A, B, C\}$  produkcijos  $\pi_1$  negalima taikyti, kintamojo F nėra GDB būsenoje. Produkcijs  $\pi_2$  taip pat negalima taikyti, nes nors ir yra C, bet nėra D. Produkcijs  $\pi_3$  galima taikyti, nes A yra. Ji taikoma. Pereinama į naują GDB būseną  $\{A, B, C, D\}$ , kuri gaunama papildžius pradinę būseną kintamuoju D iš  $\pi_3$  dešiniosios pusės (18.2 pav.). Tai nėra terminalinė būsena, nes iš ją nejine tikslas Z, ir todėl pereinama prie kitos iteracijos.



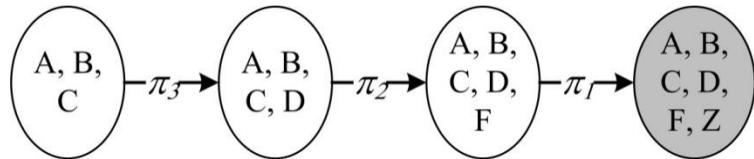
18.2 pav. Pritaikius  $\pi_3$  yra pereinama į naują GDB būseną  $\{A, B, C, D\}$  dešinėje

**2 iteracija.** Esamoje būsenoje  $\{A, B, C, D\}$  produkcijs  $\pi_1$  negalima taikyti, nes F nėra šioje būsenoje. Bet kitą produkcijs,  $\pi_2$  galima taikyti, nes yra ir C, ir D. Ji taikoma. Pereinama į būseną  $\{A, B, C, D, F\}$ , kuri gaunama papildžius einamają būseną kintamuoju F iš  $\pi_2$  dešiniosios pusės (18.3 pav.). Tai nėra terminalinė būsena, nes iš ją nejine tikslas Z, ir todėl pereinama prie kitos iteracijos.



18.3 pav. Pritaikius  $\pi_2$  pereinama į naują GDB būseną  $\{A, B, C, D, F\}$  dešinėje

**3 iteracija.** Esamoje būsenoje  $\{A, B, C, D, F\}$  produkcijs  $\pi_1$  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 kintamuoju Z iš  $\pi_1$  dešiniosios pusės (18.4 pav.). Tai terminalinė būsena. Algoritmas baigia darbą.

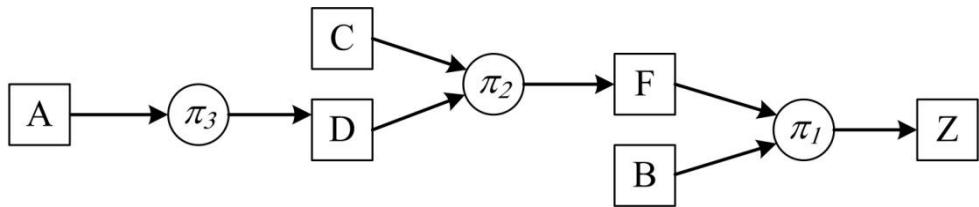


18.4 pav. Pritaikius  $\pi_1$  yra pereinama į naują GDB būseną  $\{A, B, C, D, F, Z\}$  dešinėje. Tai terminalinė būsena, nes iš ją įjina tikslas Z

Gautas rezultatas yra produkcijs seka, kuri vadinama *keliu arba planu*:

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

Produkcijų seka vaizduojama *dvidaliu grafu* (angl. *bipartite graph*, rus. *двуодольный граф*), parodytu 18.5 pav. Tai tokis grafas, kurio viršūnės yra dviejų tipų. Šiuo atveju tai viršūnės-stačiakampiai ir viršūnės-skrituliai. Viršūnė-stačiakampis pažymėta propozicinio kintamojo vardu. Viršūnė-skritulys vaizduoja produkciją ir pažymėta jos 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.

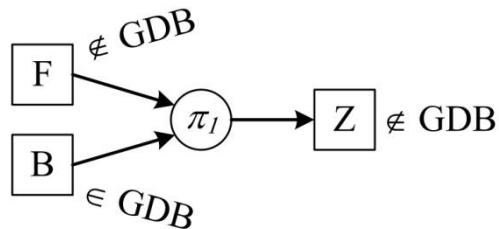


18.5 pav. Semantinis grafas, vaizduojantis tikslo kintamojo Z išvedimą iš  $\{A,B,C\}$  produkcijų sistemoje (18.1). Kelias yra seka  $\langle \pi_3, \pi_2, \pi_1 \rangle$

## 18.2. Atbulinio išvedimo algoritmas

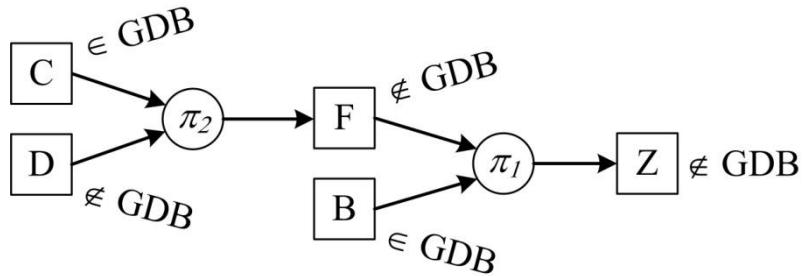
Skirtingai nuo tiesioginio išvedimo atbulinis išvedimas pradeda nuo tikslo (šiame pavyzdyste – nuo Z), o ne nuo faktų (šiame pavyzdyste  $\{A,B,C\}$ ). Atbulinio išvedimo algoritmas pateikiamas produkcijų sistemos (18.1) pavyzdžiu.

**1 iteracija.** Produkcijos (18.1) yra perrenkamos nuo pirmosios. Ieškoma tokios produkcijos, kurios dešinėje pusėje yra tikslas Z. Tokia produkcija yra  $\pi_1$ . Ji pažymima (pakeliant vėliavę *flag1* iš 0 į 1), kad nebūtų išrenkama kitose iteracijose. Imama išrinktos produkcijos  $\pi_1$  kairioji pusė  $\{F,B\}$ . Turime du naujus tikslus: F ir B. F nėra tarp faktų, o B yra (18.6 pav.). Todėl F padaromas nauju potiksliu. Toliau vyksta rekursyvus atbulinio išvedimo procedūros kvietimas. Šis kvietimas toliau aiškinamas kaip kita iteracija.



18.6 pav. Produkcijos  $\pi_1$  semantinis grafas. Jis vaizduoja Z išvedimą „atgal“, taikant  $\pi_1$ . Produkcijos jeitį tampa naujas tikslas. F nėra tarp faktų  $\{A,B,C\}$ , o B yra. Todėl F padaromas nauju potiksliu

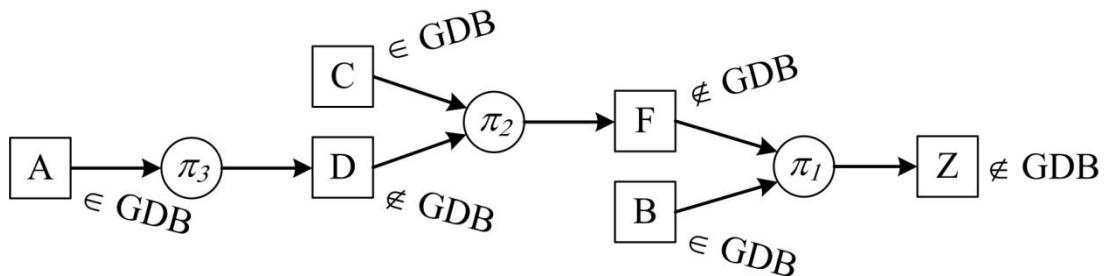
**2 iteracija.** Naujas tikslas – F. Produkcijos yra perrenkamos nuo pirmosios. Ieškoma tokios produkcijos, kurios dešinėje pusėje yra F. Tokia produkcija yra  $\pi_2$ . Ji pažymima, kad nebūtų išrenkama kitose iteracijose. Imama išrinktos produkcijos  $\pi_2$  kairioji pusė  $\{C,D\}$ . Kintamasis C priklauso faktams  $\{A,B,C\}$ , o D nepriklauso (18.7 pav.). Todėl D padaromas nauju potiksliu. Toliau vyksta rekursyvus kvietimas – kita iteracija.



18.7 pav. Semantinis grafas, vaizduojantis F išvedimą „atgal“, taikant  $\pi_2$ . Objektas C priklauso faktams {A,B,C}, o D nepriklauso. Todėl D padaromas nauju potiksliu

**3 iteracija.** Naujas tikslas – D. Produkcijos yra perrenkamos nuo pirmosios. Ieškoma tokios produkcijos, kurios dešinėje pusėje yra D. Tokia produkcija yra  $\pi_3$ . Šiame pavyzdyste ji vienintelė iš likusių nepažymėtų. Ji pažymima. Imama išrinktos produkcijos  $\pi_3$  kairioji pusė {A}. Kintamasis A priklauso faktams {A,B,C}, todėl išvedimas sėkmingai baigiamas. Grąžinamas kelias.

Gautas semantinis grafas parodytas 18.8 pav.



18.8 pav. Semantinis grafas, vaizduojantis tikslo Z išvedimą iš faktų {A,B,C} su produkcijų seka  $\langle \pi_3, \pi_2, \pi_1 \rangle$

### 18.3. Programų sintezės elementai

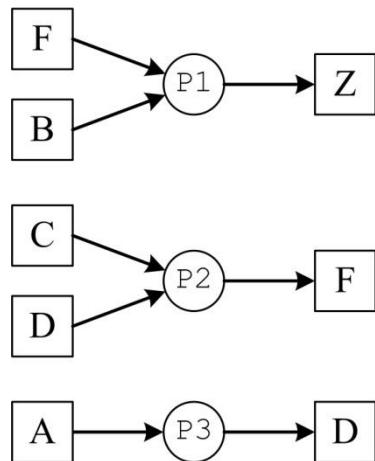
Ir tiesioginio, ir atbulinio išvedimo (18.1) pavidalo produkcijų sistemoje rezultatas yra produkcijų seka – planas. Ši seka vadinama planu todėl, kad ji planuojama vieną kartą, o vykdoma gali būti daug kartų.

Paimkime procedūras (18.3). Jų semantika yra pavaizduota produkcijomis (18.1). Čia programos semantika yra suprantama kaip pora (*jėjimas, išėjimas*). Ir jėjimas, ir išėjimas yra objektų vardų aibė. Objektus yra įprasta realizuoti kaip globalius kintamuosius algoritminėje kalboje, pvz., Pascal, Fortran, C. Algoritminėje kalboje objektai gali būti bet kokio tipo, o ne tik dvejetainiai kintamieji, įgaunantys reikšmes true arba false, kaip kad produkcijų sistemoje. Tiesa jų vardų aibė sutampa su propozicinių kintamųjų alfabetu produkcijų sistemoje. Šiame pavyzdyste alfabetas yra {A,B,C,D,E,F,Z}.

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

Procedūromis P1, P2, P3 yra užprogramuotos tam tikros *funkcinės priklausomybės* f1, f2, f3 tarp atitinkamų objektų. Tokio pavidalo procedūros vaizduoja procedūrines žinias apie tam tikrą dalykinę sritį. Programų sistemoje procedūras išprasta apjungti į biblioteką.

Procedūrų P1, P2, P3 semantika yra vaizduojama atitinkamai produkcijomis  $\pi_1$ ,  $\pi_2$  ir  $\pi_3$ . Grafiškai ir procedūrų semantika, ir jas atitinkančios produkcijos yra vaizduojamos to paties pavidalo semantiniai grafaus (18.9 pav.).



18.9 pav. Procedūrų P1, P2 ir P3 semantikos grafinis vaizdavimas semantiniai grafaus. Procedūros semantika yra suprantama kaip pora jėjimas-išėjimas ir jos vaizdavimas tekstu sutampa su produkcija

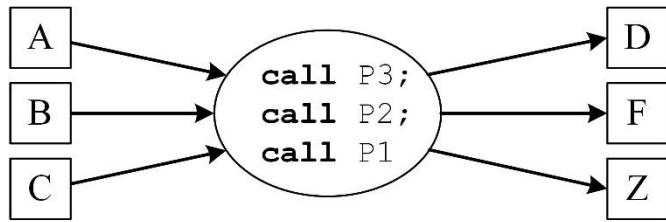
Produkcijas atitinka procedūros algoritminėje kalboje. Todėl šiame pavyzdyste ir tiesioginio, ir atbulinio išvedimo rezultatas yra produkcijų sekā  $\langle \pi_3, \pi_2, \pi_1 \rangle$ , kuri atitinka šitokią sintezuotą programą:

```

call P3;
call P2;
call P1
(18.4)

```

Sintezuotos programos forma yra nuosekli kompozicija. Kviečiamų procedūrų sekā atitinka produkcijų seką kelyje. Programos (18.4) semantika jėjimas-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 skirtinėmis reikšmėmis 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;

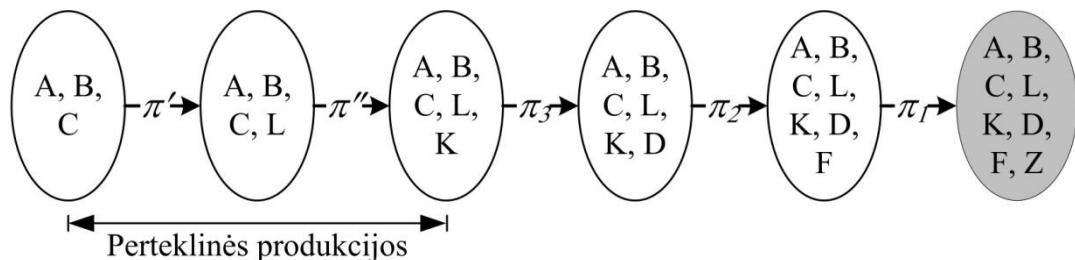
    writeln(Z);       { 3) Spausdinama Z reikšmė
end
  
```

#### 18.4. Perteklinės produkcijos tiesioginiame išvedime

Perteklinės produkcijos kelyje, kurį randa tiesioginio išvedimo algoritmas, yra iliustruojamos pavyzdžiai. Produkcijų sistema (18.1) papildoma dviem produkcijomis  $\pi'$  ir  $\pi''$ :

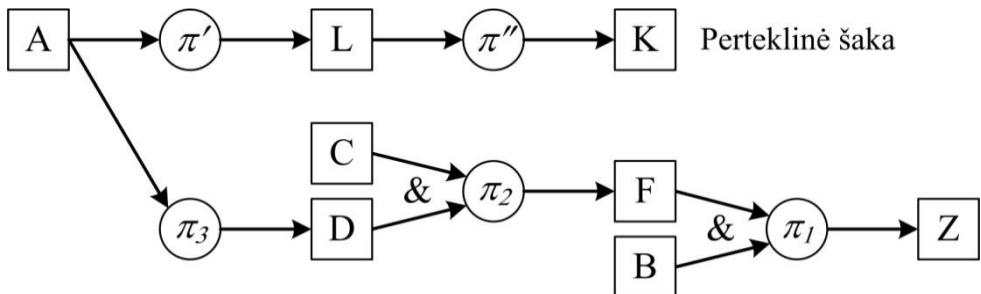
$$\begin{aligned}
 \pi': & A \rightarrow L \\
 \pi'': & L \rightarrow K \\
 \pi_1: & F, B \rightarrow Z \\
 \pi_2: & C, D \rightarrow F \\
 \pi_3: & A \rightarrow D
 \end{aligned}$$

Tegu turimų faktų aibė yra  $\{A, B, C\}$ , o tikslas –  $Z$ . Tiesioginis išvedimas randa produkcijų seką  $\langle \pi', \pi'', \pi_3, \pi_2, \pi_1 \rangle$ , kurioje dvi perteklinės produkcijos  $\pi'$  ir  $\pi''$ . Jų padariniai yra perteklinių objektų  $L$  ir  $K$  išvedimas (18.11 pav.).



18.11 pav. Tiesioginis išvedimas randa kelią  $\langle \pi', \pi'', \pi_3, \pi_2, \pi_1 \rangle$ , kuriamė perteklinės produkcijos  $\pi'$  ir  $\pi''$ . Yra išvedami pertekliniai  $L$  ir  $K$

Šiame pertekliškume „tragedijos“ kaip nesėkmės požymiu nėra. Tiesiog pailgėja ir išvedimo laikas, ir sintezuotos programos vykdymo laikas. Jeigu sintezuota programa yra vykdoma keletą kartų, tai kiekvieną kartą ji gamina perteklinius objektus L ir K. Semantinis grafas parodytas 18.12 pav.



18.12 pav. Tiesioginiame išvedime yra perteklinės produkcijos  $\pi'$  ir  $\pi''$

Perteklišumas tiesioginiame išvedime kyla dėl paieškos į plotį. Pasiekus tikslą paieška stabdoma ir grąžinamas visas paieškos medis nuoseklaus kelio pavidalu. Šio kelio išvalymui nuo perteklinių produkcijų būtų antras etapas. Tam reikalinga procedūra ATGAL, kurios mūsų algoritme nėra. Jos kvietimas būtų analogiškas ATGAL iškvietimui paieškos į plotį grafe programoje LABIRINTAS\_I\_PLOTI 8 skyriuje.

Perteklišumo prigimtis yra tokia. Analitikas, sudarinėdamas dalykinės srities modelį, žinias užrašo produkcijų pavidalu. Analitikas neturi išankstinės nuostatos, kad bus taikomas tiesioginis išvedimas. Produkcijų aibė gali būti didelė. Iš anksto nėra žinoma, kurios produkcijos bus naudojamos tam tikram faktui išvesti, o kurios ne. Kalbėdami apie išvedimo algoritmą mes nagrinėjame išvedimo metodus, o ne dalykines sritis. Kaip pradinis duomenis gavęs tam tikrus faktus, algoritmas į kelią gali įtraukti ir perteklinės produkcijas. Produkcijų perteklišumas kelyje atsiranda dėl išvedimo *monotonijos* matematinės logikos prasme.

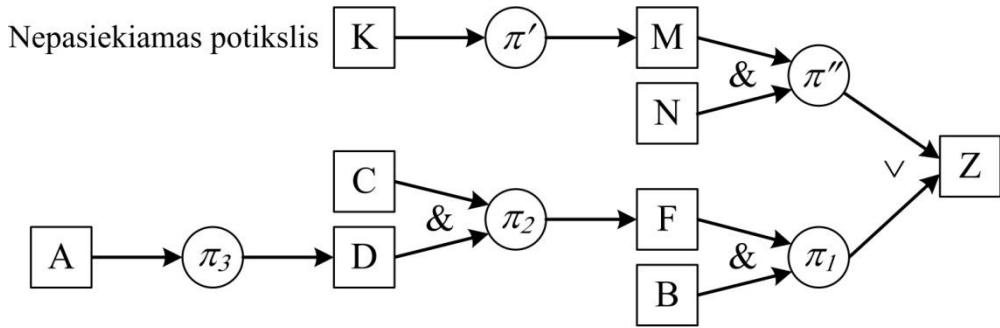
## 18.5. Perteklinės produkcijos atbuliniame išvedime

Perteklinės produkcijos atbuliniame išvedime yra kitokios prigimties. Perteklinės šakos atsiranda dėl nepasiekiamų tikslų ir grįzimo atgal (*backtrack*). Nepasiekiami tikslai yra analogija aklavietėms kelio paieškoje labirinte į gyli.

Toliau pavyzdyste prie produkcijų sistemos (18.1) yra pridėtos dvi produkcijos  $\pi'$  ir  $\pi''$ :

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

Faktų aibė  $\{A, B, C\}$  ir tikslas Z yra tie patys. Perteklinės produkcijos  $\pi'$  ir  $\pi''$  nagrinėjamos išvedime – iš pradžių  $\pi''$  o paskui gylyn  $\pi'$ . Tačiau jos į kelią  $\langle \pi_3, \pi_2, \pi_I \rangle$  nėra įtraukiama. Dėl jų yra iškeliami nepasiekiami tikslai K ir N. Iš pradžių  $\pi''$  iškelia potikslius M ir N, o paskui  $\pi'$  iškelia potikslį K. Kadangi K nepasiekiamas, tai nėra prasmės išvedinėti N; žr 18.13 pav. Konjunkcijoje  $p_1 \& \dots \& p_n$  aptikus pirmajį neišvedamą tikslą, likusių išvedinėti nėra prasmės.



18.13 pav. Atbuliniame išvedime perteklinės produkcijos  $\pi'$  ir  $\pi''$ . Jos į kelią  $\langle \pi_3, \pi_2, \pi_1 \rangle$  nėra įtraukiamos. Potikslis K nepasiekiamas

## 18.6. Tiesioginio išvedimo sudėtingumas

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

**Įrodymas.** 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 produkcija yra pažymima (pakeliant vėliavę *flag1*), kad nebūtų taikoma kitose iteracijose. Trečiojoje iteracijoje atliekama  $N-2$  veiksmų. Ir taip toliau. Paskutinė kelio produkcija yra pasirenkama  $N$ -tosios iteracijos metu, kai po  $N-1$ -osios iteracijos lieka nepažymėta viena produkcija. Tokiu būdu bendras atliekamų veiksmų skaičius (blogiausiu atveju) yra:

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

Labai gero laipsnio sudėtingumas – kvadratinis, o ne eksponentinis – yra todėl, kad faktai suprantami kaip propoziciniai kintamieji, o ne kaip predikatų vardai. Šitaip nekyla predikatų parametrų sutapatinimo problema, žinoma, pavyzdžiui, Prologo algoritminėje kalboje, kur išvedimo sudėtingumas eksponentinis.

## 18.7. Testai tiesioginiams išvedimui

### 1. Faktas konsekvente

Taisyklės lentelėje.

Faktai A, B, C.

Tikslas Z.

Kelias R1, R2, R7, R4, R6, R5.

Duomenų failas:

R1:	A	$\rightarrow$	L
R2:	L	$\rightarrow$	K
R3:	D	$\rightarrow$	A
R4:	D	$\rightarrow$	M
R5:	F, B	$\rightarrow$	Z
R6:	C, D	$\rightarrow$	F
R7	A	$\rightarrow$	D

Studentas Vardenis Pavardenis, studijų programa, kursas, grupė. Data  
1 testas. Paskaitos pavyzdys

1) Taisyklės

L A // R1: A → L. Komentarai dėl dubliavimo.  
K L // R2: L → K  
A D // R3: D → A  
M D // R4: D → M  
Z F B // R5: F, B → Z  
F C D // R6: C, D → F  
D A // R7: A → D

2) Faktai

A B C

3) Tikslas

Z

Protokolas:

1 DALIS. Duomenys

1) Taisyklės

R1: A → L  
R2: L → K  
R3: D → M  
R4: D → M  
R5: F, B → Z  
R6: C, D → F  
R7: A → D

2) Faktai

A, B, C

3) Tikslas

Z

2 DALIS. Vykdymas

1 ITERACIJA

R1:A->L taikome. Pakeliame flag1. Faktai A, B, C ir L.

2 ITERACIJA

R1:A->L praleidžiame, nes pakelta flag1 (taisyklė panaudota).  
R2:L->K taikome. Pakeliame flag1. Faktai A, B, C ir L, K.

3 ITERACIJA

R1:A->L praleidžiame, nes pakelta flag1.  
R2:L->K praleidžiame, nes pakelta flag1.  
R3:D->A netaikome, nes trūsta D.  
R4:D->M netaikome, nes trūsta D.  
R5:F,B->Z netaikome, nes trūksta F.  
R6:C,D->F netaikome, nes trūksta D.  
R7:A->D taikome. Pakeliame flag1. Faktai A, B, C ir L, K, D.

4 ITERACIJA

R1:A->L praleidžiame, nes pakelta flag1.  
R2:L->K praleidžiame, nes pakelta flag1.  
R3:D->A netaikome, nes konsekventas faktuose. Pakeliame flag2.  
R4:D->M taikome, pakeliame flag1. Faktai A, B, C ir L, K, D, M.

5 ITERACIJA

R1:A->L praleidžiame, nes pakelta flag1.

R2:L->K praleidžiame, nes pakelta flag1.  
R3:D->A praleidžiame, nes pakelta flag2.  
R4:D->M praleidžiame, nes pakelta flag1.  
R5:F,B->Z netaikome, nes trūksta F.  
R6:C,D->F taikome. Pakeliame flag1. Faktai A, B, C ir L, K, D, M, F.

#### 6 ITERACIJA

R1:A->L praleidžiame, nes pakelta flag1.  
R2:L->K praleidžiame, nes pakelta flag1.  
R3:D->A praleidžiame, nes pakelta flag2.  
R4:D->M praleidžiame, nes pakelta flag1.  
R5:F,B->Z taikome. Pakeliame flag1. Faktai A, B, C ir L, K, D, M, F, Z.  
Tikslas gautas.

#### 3 DALIS. Rezultatai

- 1) Tikslas Z išvestas.
- 2) Kelias: R1, R2, R7, R4, R6, R5.

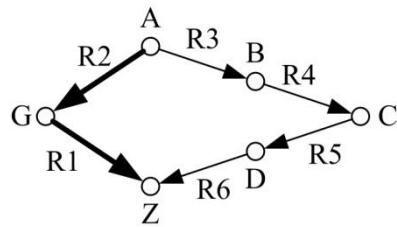
#### 2. Čyras vs. Negnevitsky; Čyras laimi

Taisykłės lentelėje.

Faktas A.

Tikslas Z.

Kelias R2, R1.



R1:	$G \rightarrow Z$
R2:	$A \rightarrow G$
R3:	$A \rightarrow B$
R4:	$B \rightarrow C$
R5:	$C \rightarrow D$
R6:	$D \rightarrow Z$

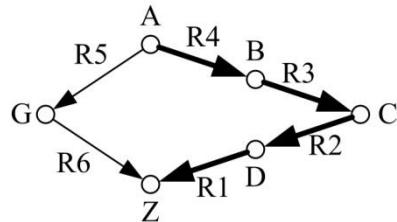
#### 3. Čyras vs. Negnevitsky; Negnevitsky laimi, bet su atliekama taisykla.

Taisykłės lentelėje.

Faktas A.

Tikslas Z.

Kelias R4, R3, R2, R1.



R1:	$D \rightarrow Z$
R2:	$C \rightarrow D$
R3:	$B \rightarrow C$
R4:	$A \rightarrow B$
R5:	$A \rightarrow G$
R6:	$G \rightarrow Z$

#### 4. Tikslas tarp faktų

- 3 DALIS. Rezultatai  
Tikslas A tarp faktų. Kelias tuščias.

#### 5. Kelias neegzistuoja

Taisykłės R1: A → B; R2: C → Z. Faktas A. Tikslas Z.

#### 6. Negnevitsky pavyzdys

Pavyzdys su 5 taisyklemis iš (Negnevitsky 2011) skyriaus apie tiesioginį išvedimą.

## 18.8. Testai atbuliniams išvedimui

### 1. Užmirštama šaka

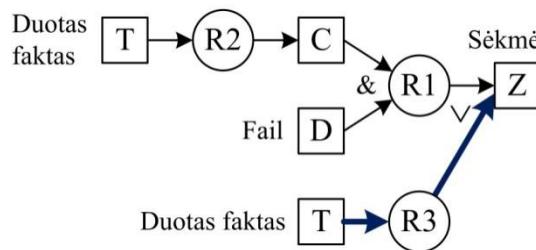
Taisyklys R1: C, D → Z; R2: T → C; R3: T → Z. Faktas T. Tikslas Z. Kelias R3. Svarbu pastebėti, kad kelias ne R2, R3 ir kad faktų aibė pabaigoje ne T, C, Z, bet T, Z. Taip yra dėl valdymo su grįžimais (*backtrack*) užmirštant šaką nuo Z gilyn į C.

2 DALIS. Vykdymas

- 1) Tikslas Z. Randame R1:C,D->Z. Nauji tikslai C, D.
- 2) -Tikslas C. Randame R2:T->C. Nauji tikslai T.
- 3) --Tikslas T. Faktas (duotas), nes faktai T. Grįžtame, sékmė.
- 4) -Tikslas C. Faktas (dabar gautas). Faktai T ir C.
- 5) -Tikslas D. Néra taisyklių jo išvedimui. Grįžtame, FAIL.
- 6) Tikslas Z. Randame R3:T->Z. Nauji tikslai T.
- 7) -Tikslas T. Faktas (duotas), nes faktai T. Grįžtame, sékmė.
- 8) Tikslas Z. Faktas (dabar gautas). Faktai T ir Z. Sékmė.

3 DALIS. Rezultatai

- 1) Tikslas Z pasiektas.
- 2) Kelias: R3.



### 2. Devynios produkcijos D, C

Taisyklys lentelėje.

Faktas T.

Tikslas Z.

Kelias R6, R5, R4, R3, R1.

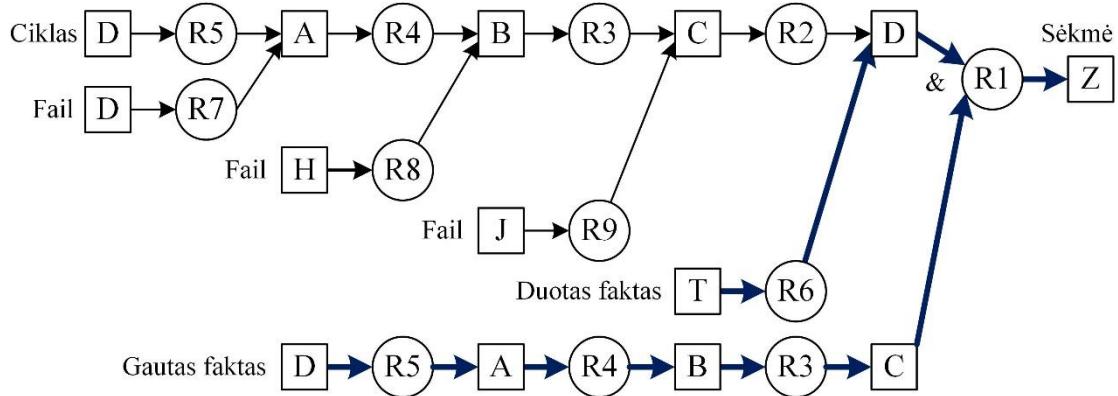
2 DALIS. Vykdymas

- 1) Tikslas Z. Randame R1:D,C->Z. Nauji tikslai D, C.
- 2) -Tikslas D. Randame R2:C->D. Nauji tikslai C.
- 3) --Tikslas C. Randame R3:B->C. Nauji tikslai B.
- 4) ---Tikslas B. Randame R4:A->B. Nauji tikslai A.
- 5) ----Tikslas A. Randame R5:D->A. Nauji tikslai D.
- 6) -----Tikslas D. Ciklas. Grįžtame, FAIL.
- 7) -----Tikslas A. Randame R7:G->A. Nauji tikslai G.
- 8) -----Tikslas G. Néra taisyklių jo išvedimui. Grįžtame, FAIL.
- 9) -----Tikslas A. Néra daugiau taisyklių jo išvedimui. Grįžtame, FAIL.
- 10) ---Tikslas B. Randame R8:H->B. Nauji tikslai H.
- 11) ---Tikslas H. Néra taisyklių jo išvedimui. Grįžtame, FAIL.
- 12) ---Tikslas B. Néra daugiau taisyklių jo išvedimui. Grįžtame, FAIL.
- 13) --Tikslas C. Randame R9:J->C. Nauji tikslai J.
- 14) --Tikslas J. Néra taisyklių jo išvedimui. Grįžtame, FAIL.
- 15) --Tikslas C. Néra daugiau taisyklių jo išvedimui. Grįžtame, FAIL.
- 16) -Tikslas D. Randame R6:T->D. Nauji tikslai T.
- 17) --Tikslas T. Faktas (duotas), nes faktai T. Grįžtame, sékmė.
- 18) -Tikslas D. Faktas (dabar gautas). Faktai T ir D.
- 19) -Tikslas C. Randame R3:B->C. Nauji tikslai B.
- 20) --Tikslas B. Randame R4:A->B. Nauji tikslai A.
- 21) --Tikslas A. Randame R5:D->A. Nauji tikslai D.
- 22) ----Tikslas D. Faktas (buvo gautas), nes faktai T ir D. Grįžtame, sékmė.
- 23) --Tikslas A. Faktas (dabar gautas). Faktai T ir D, A. Grįžtame, sékmė.
- 24) --Tikslas B. Faktas (dabar gautas). Faktai T ir D, A, B. Grįžtame, sékmė.
- 25) .Tikslas C. Faktas (dabar gautas). Faktai T ir D, A, B, C. Grįžtame, sékmė.
- 26) Tikslas Z. Faktas (dabar gautas). Faktai T ir D, A, B, C, Z. Sékmė.

R1:	D, C → Z
R2:	C → D
R3:	B → C
R4:	A → B
R5:	D → A
R6:	T → D
R7	G → A
R8	H → B
R9	J → C

### 3 DALIS. Rezultatai

- 1) Tikslas Z pasiektas.
- 2) Kelias: R6, R5, R4, R3, R1.



### 3. Devynios produkcijos C, D

Kaip aukščiau, bet R1: C, D → Z. Randamas tas pats kELIAS.

### 4. Ciklas ir praleistas potikslis

Taisykles R1: A → Z; R2: B → A; R3: A, C → B; R4: T → B; R5: T → C. Faktas T. Tikslas Z. Kelias R4, R2, R1.

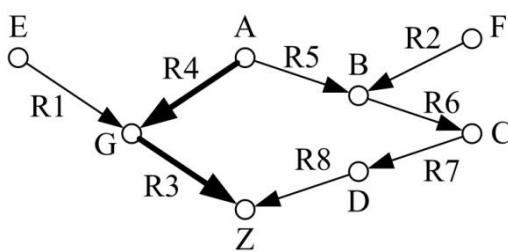
### 5. Grafas su trumpu keliu

Taisykles lentelėje.

Faktas A.

Tikslas Z.

KELIAS R4, R3.



R1:	E → G
R2:	F → B
R3:	G → Z
R4:	A → G
R5:	A → B
R6:	B → C
R7:	C → D
R8:	D → Z

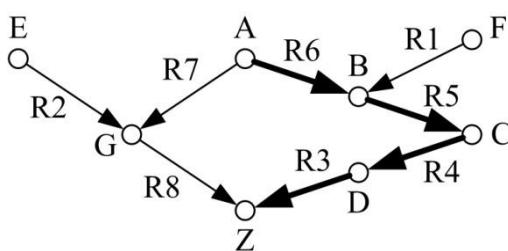
### 6. Grafas su ilgu keliu

Taisykles lentelėje.

Faktas A.

Tikslas Z.

KELIAS R6, R5, R4, R3.



R1:	F → B
R2:	E → G
R3:	D → Z
R4:	C → D
R5:	B → C
R6:	A → B
R7:	A → G
R8:	G → Z

### 7. Trys alternatyvos tikslui

Taisykles R1: A → Z; R2: B → Z; R3: C → Z. Faktas C. Tikslas Z. KELIAS R3.

### 8. Nepasiekiamas tikslas

Taisykles R1: C, D → Z; R2: C, E → Y. Faktai C, D. Tikslas Y. KELIAS neegzistuoja.

### 9. Tikslas tarp faktų – tuščias kELIAS

#### 3 DALIS. Rezultatai

Tikslas A tarp faktų. Tuščias kELIAS.

### 10. Negnevitsky pavyzdys

Pavyzdys su 5 taisyklemis iš (Negnevitsky 2011) skyriaus apie atbulinių išvedimą.

## 19. Rezoliucijų metodas

Žodis „rezoliucija“ žymi ir išvedimo taisykę, ir išvedimo metodą, kurie aiškinami toliau. Žr. taip pat [https://en.wikipedia.org/wiki/Resolution\\_\(logic\)](https://en.wikipedia.org/wiki/Resolution_(logic)). Išvedimas rezoliucijų metodu demonstruojamas paprastu pavyzdžiu; žr. (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:

$$\text{Taisyklė R1: } \text{Prof}(x, y) \ \& \ \text{Stud}(z, w) \ \& \ \neg \text{Lygu}(x, z) \Rightarrow \text{Egz}(y, w) \quad (19.3)$$

Pastaroji formulė (19.3) suprantama kaip produkcinė taisyklė 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  $P \Rightarrow Q$  pašalinama, keičiant ją disjunkcija su  $Q$  paneigimu:  $\neg P \vee Q$ . Ši pakeitimą galima suprasti ir kaip *perrašymo taisyklę* (*term rewriting rule*):

$$\frac{P \Rightarrow Q}{\neg P \vee Q} \quad \text{Implikacijos pašalinimo taisyklė} \quad (19.4)$$

Čia virš brūkšnio yra prielaida (t. y. kas perrašoma), o po brūkšniu išvada (iš ką perrašoma).

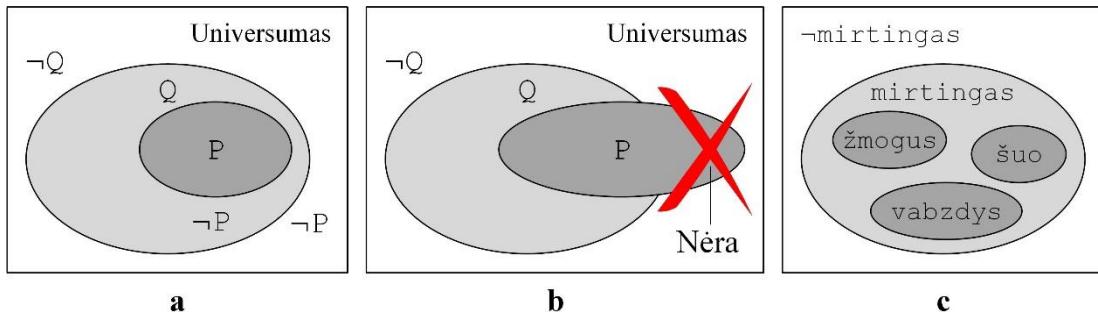
Iš matematinės logikos žinoma, kad ši perrašymo taisyklė yra *validi* (*sound*). Kitais žodžiais, korektiška semantiškai, logiška, nes viršutinė ir apatinė formulė yra ekvivalentios:  $P \Rightarrow Q \equiv \neg P \vee Q$ . Šių formulų teisingumo lentelės sutampa (nes `true`  $\vee$  `true` = `true`, `true`  $\vee$  `false` = `true`, ir tik `false`  $\vee$  `false` = `false`):

$P$	$Q$	$\neg P$	$\neg P \vee Q$	$P \Rightarrow Q$
true	true	false	true	true
true	false	false	false	false
false	true	true	true	true
false	false	true	true	true

Implikacijos pašalinimo taisyklė (19.4) paaiškinama šiuo pavyzdžiu:

$$\frac{\text{žmogus} \Rightarrow \text{mirtinges}}{\neg \text{žmogus} \vee \text{mirtinges}}$$

Kai  $P \Rightarrow Q$ , tai  $P$  būsenų aibė (kurioje  $P = \text{true}$ ) universume yra poaibis  $Q$  būsenų aibės (kurioje  $Q = \text{true}$ ), t. y.  $būsenos(P) \subset būsenos(Q)$ ; žr. 19.1 a pav. Kitais žodžiais,  $P$  būsenų aibė neišeina už  $Q$  ribų (19.1 b pav.). Atkreipiame dėmesį, kad aibų įdėjimas rodo priešinga kryptimi negu implikacijos rodyklė. Veno diagrama 19.1 c pav. iliustruoja tris implikacijas žmogus  $\Rightarrow$  mirtingas, vabzdys  $\Rightarrow$  mirtingas ir šuo  $\Rightarrow$  mirtingas.



19.1 pav. a) Implikacijos  $P \Rightarrow Q$  vaizdavimas Veno diagrama. b) Nepersidengia  $P$  ir  $\neg Q$ , t. y. „išsikišimo“ negali būti. c) Trys implikacijos

Implikacijos pašalinimo taisykla yra apibendrinama šitaip:

$$\frac{P_1 \& P_2 \& \dots \& P_n \Rightarrow Q}{\neg P_1 \vee \neg P_2 \vee \dots \vee \neg P_n \vee Q} \quad \text{Implikacijos pašalinimas (apibendrintas)} \quad (19.5)$$

Kadangi  $\neg\neg P \equiv P$ , tai turime *dvigubo neigimo pašalinimo taisykla*:

$$\frac{\neg\neg P}{P} \quad \text{Dvigubo neigimo pašalinimas} \quad (19.6)$$

Toliau produkcija (19.3) perrašoma pagal implikacijos pašalinimo taisykla (19.5):

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

Pastarojoje formulėje pašalinamas dvigubas neigimas:

$$R1: \neg \text{Prof}(x, y) \vee \neg \text{Stud}(z, w) \vee \text{Lygu}(x, z) \vee \text{Egz}(y, w) \quad (19.7)$$

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

Toliau parodysime, kaip įrodoma rezoliucijos taisyklos pagalba. Rezoliucijos taisykla pateikiama toliau (19.8), o ją aiškinti pradedame nuo išvedimo taisyklos *modus ponens* (lotyniškai – teigimo būdas):,,1) Tegu F. 2) Tegu jei F, tai G. 3) Taigi G“.

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

Dar rašoma horizontaliai, abi prielaidas skiriant kableliu (tvarka nėra svarbi):

$$\frac{P, \quad P \Rightarrow Q}{Q} \qquad \textit{modus ponens}$$

Pavyzdžiui, „1) Žmogus. 2) Jei žmogus, tai mirtingas. 3) Taigi mirtingas.“ Kitas pavyzdys, bet užrašytas predikatų logika: „Sokratus yra žmogus. Jei  $x$  yra žmogus, tai mirtingas  $x$ . Taigi Sokratus mirtingas.“ (Predikatai yra teiginiai su kintamaisiais; predikatas – tai funkcija, kurios reikšmė true arba false.) Išvedama keičiant kintamąjį  $x$  konstanta Sokratus:

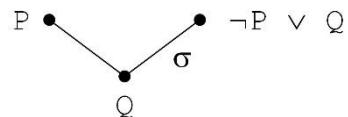
$$\frac{\begin{array}{l} 1) \text{ žmogus (Sokratus)} \\ 2) \forall x \text{ žmogus } (x) \Rightarrow \text{mirtingas } (x) \\ \hline 3) \text{ mirtingas (Sokratus)} \end{array}}{\{ \text{Sokratus}/x \}}$$

*Rezoliucijos taisyklė* paprasčiausia forma užrašoma šitaip (grafiškai parodyta 19.2 pav.):

$$\frac{P, \quad \neg P \vee Q}{Q} \quad \sigma \qquad \textbf{Rezoliucijos taisyklė (paprasčiausia forma)} \qquad (19.8)$$

Aiškinima šitaip. Tegu turime dvi formules  $P$  ir  $\neg P \vee Q$ . Pagal rezoliucijos taisyklę iš jų išvedame  $Q$ . Pastaroji formulė, išvada, vadinama *rezolvente*. Keitinys pažymėtas  $\sigma$ . Formulės turi bendrą pavidalą  $\forall x_1 \forall x_2 \dots \forall x_n G(x_1, x_2, \dots, x_n)$ ; plačiau žr., pvz., (Norgėla 2007, p. 26, 92, 158–163; Nilsson 1998, p. 253–268). Teiginių (o ne predikatų) logikos atveju taisyklė (19.8) vadinama *atkirtos taisykle* (Norgėla 2007, p. 109). Tai išvedimo taisyklė *disjunktų dedukcinėje sistemoje*. Lotyniškai *deductio* reiškia išvedimą. Teiginių logikoje keitinio  $\sigma$  nėra, nes nėra individinių kintamųjų  $x_i$ .

19.2 pav. Grafiškai vaizduojama rezoliucijos taisyklė (dar vadinama atkirtos taisykle)



Užrašymo forma (19.8) analogiška *modus ponens*. Prisiminkime, kad aukščiau parodėme formulų  $P \Rightarrow Q$  ir  $\neg P \vee Q$  yra ekvivalentumų; žr. nuo (19.4). Rezoliucijos taisyklė taikoma, kai į vieną formulę įeina teigiamas disjunktas  $P$ , o į kitą – neigiamas disjunktas  $\neg Q$ . Rezolventėje jų nelieka.

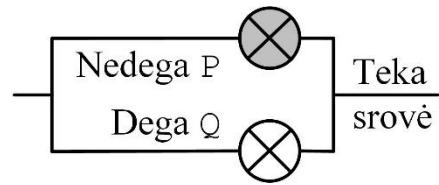
Bendru atveju disjunktai  $P$  ir  $\neg P$  gali turėti kintamuosius, pvz.,  $\neg P(x)$ . Todėl prieš taikant rezoliucijos taisyklę atliekami keitiniai, pvz.,  $\{ A/x \}$ , kad  $P(A)$  ir  $\neg P(x)$  susitapatintų.

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

$$\frac{\neg P, \quad P \vee Q}{Q} \qquad (19.9)$$

Ir *modus ponens* taisyklė, ir rezoliucijos taisyklė iliustruojamos 19.3 pav. Paimkime dviejų lempučių  $P$  ir  $Q$  lygiagreatus sujungimo grandinę. Grandinė modeliuojama formulė  $P \vee Q$ . Tegu  $P$  lemputė perdegusi (t. y. nedega,  $\neg P = \text{true}$ ), bet srovė grandine teka,  $P \vee Q = \text{true}$ . Iš to daroma išvada, kad lemputė  $Q$  dega,  $Q = \text{true}$ , rašoma  $Q$ .

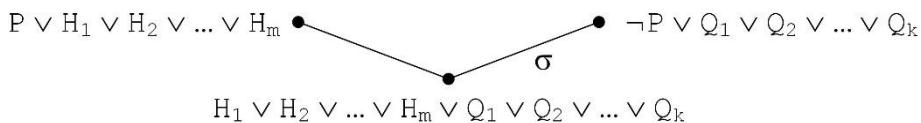
19.3 pav. Lygiagreti lempučių  $P$  ir  $Q$  grandinė užrašoma disjunkcija  $P \vee Q$ . Teiginys „lemputė  $P$  perdegusi“ užrašomas  $\neg P$ . Iš šių dviejų formulų išplaukia  $Q$ , t. y. lemputė  $Q$  dega



Bendru atveju rezoliucijos taisyklė užrašoma šitaip, kur  $n \geq 0$ :

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

Rezolventė  $H \vee Q$  gaunama iš dviejų pradinių formulų pašalinant teigiamą ir neigiamą disjunktus. Iš  $P$ ,  $H$  ir  $Q$  įeinantys kintamieji  $x_i$ , jeigu tokiu yra, keičiami išraiškomis  $A_i$ . Bendra rezoliucijos taisyklės forma parodyta 9.4 pav.



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

## 19.1. Atbulinis ir tiesioginis įrodymai su rezoliucijos taisykle

Yra dvi įrodymo kryptys kaip du išvedimo būdai: *atbulinis įrodymas* (nuo tikslo link duomenų) ir *tiesioginis įrodymas* (nuo duomenų link tikslo).

**Atbulinis įrodymas.** Šiame pavyzdje tikslas yra įrodyti teoremą – teiginį „Žakas egzaminuoja Mari“,  $Egz(\text{Žakas}, \text{Mari})$ . Įrodoma prieštaros būdu – daroma priešinga prielaida. Tegu profesorius Žakas negali egzaminuoti studentės Mari. Tai užrašoma kaip tikslo paneigimas:

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

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

$$\frac{\neg Egz(\text{Žakas}, \text{Mari}), \neg Prof(x, \text{Žakas}) \vee \neg Stud(z, \text{Mari}) \vee Lygu(x, z) \vee Egz(\text{Žakas}, \text{Mari})}{\neg Prof(x, \text{Žakas}) \vee \neg Stud(z, \text{Mari}) \vee Lygu(x, z)}$$

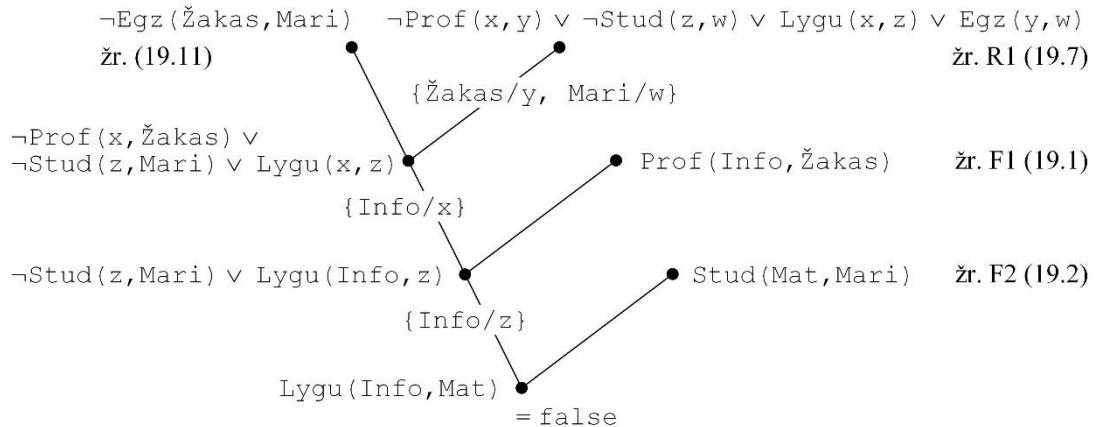
Toliau imama pastaroji rezolventė bei faktas F1 (19.1), ir taikoma rezoliucijos taisykla (19.8). Keitinys  $\{\text{Info}/x\}$ .

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

Toliau imama pastaroji rezolventė bei faktas F2 (19.2) ir taikoma rezoliucijos taisykla (19.8). Keitinys  $\{\text{Mat}/z\}$ .

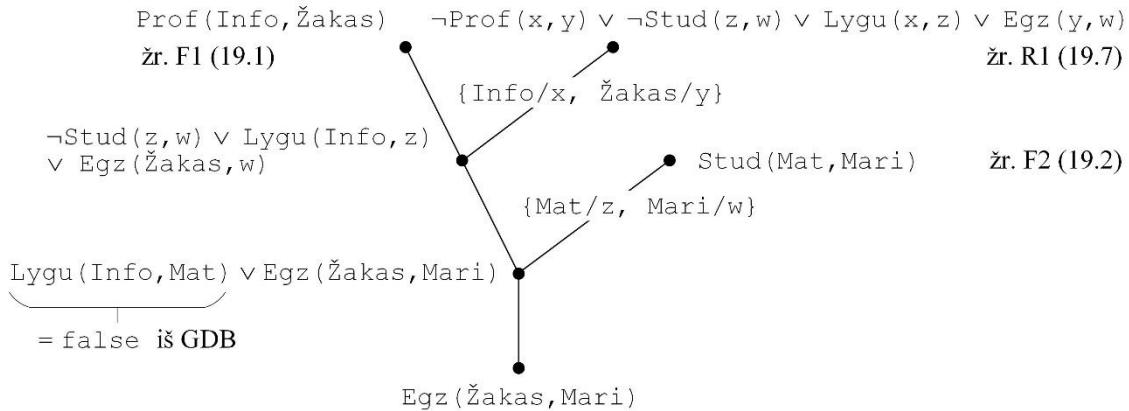
$$\frac{\neg \text{Stud}(\text{Mat}, \text{Mari}) \vee \text{Lygu}(\text{Info}, \text{Mat}), \quad \text{Stud}(\text{Mat}, \text{Mari})}{\text{Lygu}(\text{Info}, \text{Mat})}$$

Kadangi informatikos ir matematikos fakultetai yra skirtingi fakultetai, formaliai  $\text{Lygu}(\text{Info}, \text{Mat}) = \text{false}$ , tai gauta rezolventė yra  $\text{false}$ . Gaunama prieštara. Kadangi iš tikslo paneigimo išvedama prieštara, tai tikslas Egz(Žakas, Mari) yra teisingas. Irodymo medis parodytas 19.5 pav. **Irodymo pabaiga**.



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

**Tiesioginis irodymas – nuo faktų prie tikslo.** Duomenų bazėje yra du faktai – F1 (19.1) bei F2 (19.2) – ir viena produkcija R1 (19.7). Irodyti  $\text{Egz}(\text{Žakas}, \text{Mari})$ .



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

Imamas faktas F1  $\text{Prof}(\text{Info}, \text{Žakas})$  ir ieškoma produkcija, su kuria galima sutapatinti. Imama R1 (19.7) ir taikoma rezoliucijos taisykla (19.8) su keitiniu  $\{ \text{Info}/x, \text{Žakas}/y \}$ :

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

Toliau imamas faktas F2 Stud (Mat, Mari) ir pastaroji rezolventė. Taikoma rezoliucijos taisyklė (19.8) su keitiniu {Mat/z, 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  $\text{Egz}(\text{Žakas}, \text{Mari})$ . Q.E.D. (lotyniškai *quod erat demonstrandum*) – ką ir reikėjo įrodyti. Įrodymo medžiagą žr. 19.6 pav. **Įrodymo pabaiga**.

Pridėkime dar vieną faktą: Irma yra informatikos fakulteto studentė. Formaliai, F3:  $\text{Stud}(\text{Info}, \text{Irma})$ . Tačiau su taisykle R1 neįmanoma išvesti, kad profesorius Žakas gali egzaminuoti studentę Irmą.

Bendru atveju galima įrodinėti ir nuo duomenų, ir nuo tikslų. Įrodymo esmė, kokia tvarka parinkti faktus ir taisykles, kad sutapatinti kintamuosius rezoliucijos taisyklėje. Intelektas glūdi sekoje. Logika yra tik priemonė plano sudarytojui arba įrodymo programai (*prover*).

## 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
$\pi_1$ :	$F, B \rightarrow Z$	$F \& B \Rightarrow Z$	$\neg F \vee \neg B \vee Z$
$\pi_2$ :	$C, D \rightarrow F$	$C \& D \Rightarrow F$	$\neg C \vee \neg D \vee F$
$\pi_3$ :	$A \rightarrow D$	$A \Rightarrow D$	$\neg A \vee D$

Be šių trijų produkcijų dar yra trys faktai: F1: A, F2: B ir F3: C. Čia faktai suprantami kaip propoziciniai kintamieji:  $A=\text{true}$ ,  $B=\text{true}$  ir  $C=\text{true}$ . Įrodyti: Z.

**Atbulinis įrodymas.** Pradedama nuo tikslų paneigimo:  $\neg Z$ . Ieškoma produkcija tapatinti. Tai  $\pi_1$ . Toliau taikoma rezoliucijos taisyklė (19.8):

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

Gautai rezolventei  $\neg F \vee \neg B$  ieškoma kita produkcija, su kuria galima tapatinti. Tai  $\pi_2$ . Toliau taikoma rezoliucijos taisyklė (19.10):

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

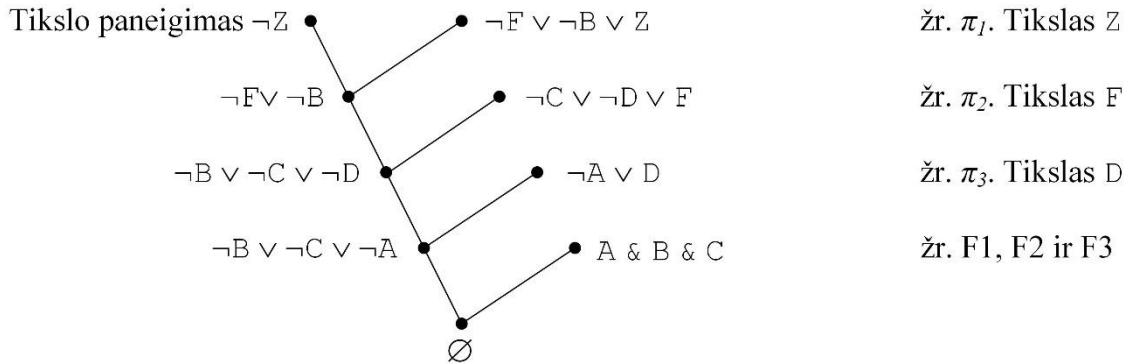
Gautai rezolventei  $\neg B \vee \neg C \vee \neg D$  ieškoma kita produkcija, su kuria galima tapatinti. Tai  $\pi_3$ . Toliau taikoma rezoliucijos taisyklė (19.10):

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

Rezolventę viršuje galima perrašyti, nes  $\neg B \vee \neg C \vee \neg A \equiv \neg(B \wedge A \wedge C)$ . Kadangi globalioje duomenų bazėje yra faktai A, B ir C, tai  $A \wedge B \wedge C = \text{true}$ . Todėl taikome rezoliucijos taisykłę:

$$\frac{\neg(B \wedge A \wedge C), A \wedge B \wedge C}{\emptyset}$$

Gauname tuščią disjunktą. Jis reiškia prieštarą. Tokiu būdu iš tiksllo paneigimo  $\neg Z$  gavome prieštarą. Todėl tikslas Z teisingas. Įrodymo medži žr. 19.7 pav. **Atbulinio įrodymo pabaiga**.



19.7 pav. Teoremos Z atbulinio išvedimo medis: nuo tiksllo paneigimo iki prieštaros (tuščio disjunkto)

Intelektas pasireiškia sutapatinimo sekoje  $\pi_1, \pi_2, \pi_3$ . Skaitant pastarąjį seką nuo galo yra gaunamas kelias  $\langle \pi_3, \pi_2, \pi_1 \rangle$ .

**Tiesioginis įrodymas.** Pradedama nuo faktų. Imamas faktas F1, t. y. A. Ieškoma produkcija, kurią bus galima pritaikyti pagal rezoliucijos taisykłę. Tokia yra  $\pi_3$ , t. y.  $\neg A \vee D$ . Taikoma rezoliucijos taisykla (19.8):

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

Rezolventei D ieškoma produkcija pritaikyti. Randama  $\pi_2$ :  $\neg C \vee \neg D \vee F$ . Taikoma rezoliucijos taisykla (19.8):

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

Rezolventei  $\neg C \vee F$  parenkamas faktas F3: C. Taikoma rezoliucijos taisykla (19.8):

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

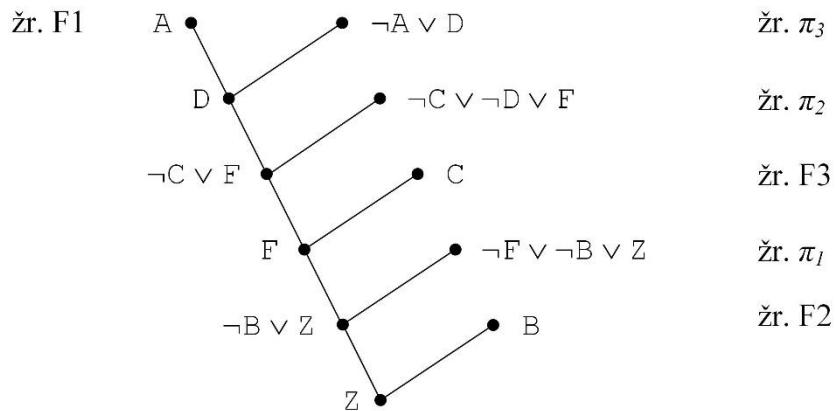
Rezolventei F ieškoma produkcija pritaikyti. Randama  $\pi_1$ :  $\neg F \vee \neg B \vee Z$ . Taikoma rezoliucijos taisykla (19.8):

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

Rezolventei  $\neg B \vee Z$  parenkamas faktas F2: B. Taikoma rezoliucijos taisyklė (19.8):

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

Gavome Z, ką ir reikėjo įrodyti. Įrodymo medži žr. 19.8 pav. **Tiesioginio įrodymo pabaiga.**



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

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

Šis skyrelis iliustruoja išvendimą teiginių logikoje. Todėl jo teorinis pagrindas yra disjunktū dedukcinė sistema; žr. skyrelį (Norgėla 2007, p. 109–115) su bendresniais pavyzdžiais.

### 19.3. Pavyzdys: rezoliucija teoremos įrodyme

Pateikiamas pavyzdys iš Nilsson (1998, 16.5, p. 260–261) poskyrio „Using Resolution to Prove Theorems“. Tegu robotas žino, kad kiekvienas paketas 27 kambarje yra mažesnis už bet kurį paketą 28 kambarje. Tai užrašoma formule predikatų kalba:

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

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

$$(3) P(A). \text{ Tai faktas.}$$

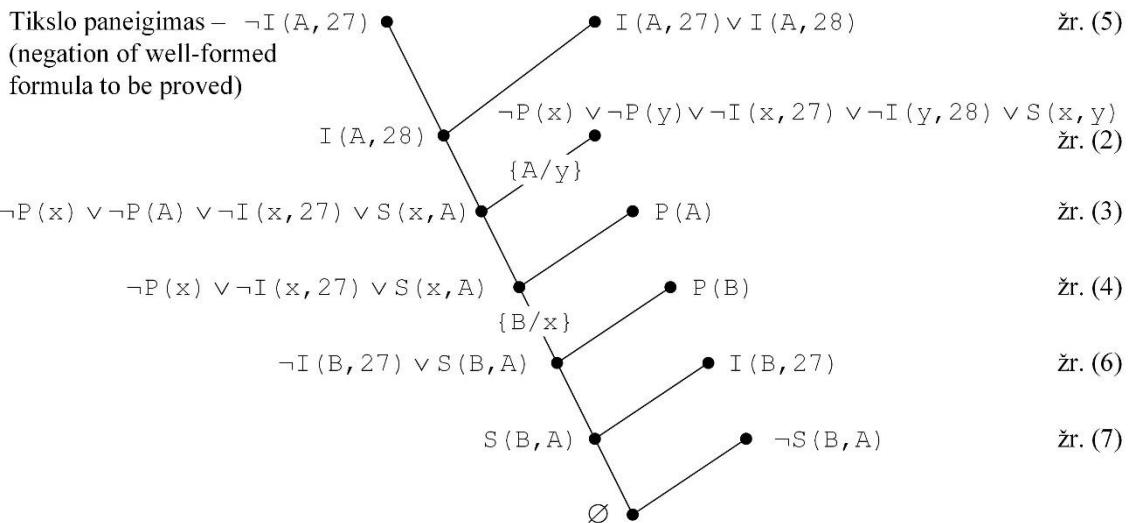
$$(4) P(B). \text{ Tai faktas.}$$

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

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

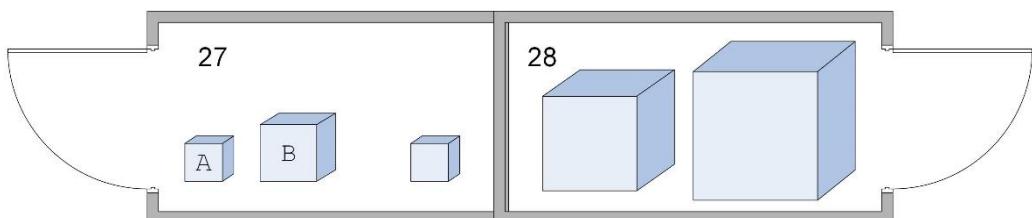
(7)  $\neg S(B, A)$ . Tai faktas, kurį žino robotas.

Robotas gali įrodyti, kad paketas A yra 27 kambarje. Teorema (tikslas) yra  $I(A, 27)$ . Rezoliucijos taisyklė naudojama kiekviename įrodymo žingsnyje. Pradedama nuo teoremos paneigimo: „Tegu  $\neg I(A, 27)$ “ (19.9 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.



19.9 pav. Teoremos įrodymo medis (*proof tree*); žr. (Nilsson 1998, p. 261)

Galimas pasaulis parodytas 19.10 pav.



19.10 pav. Galimas pasaulis, apie kurį žino robotas

## 19.4. Logikos vaidmuo samprotavime

Naujos žinios išvesti svarbus loginis, dedukcinis išvedimas (Brachman, Levesque 2004).  
Pavyzdžiui, tegu teisingi du teiginiai:

1. Pacientas Jonas yra alergiškas medikamentui M;
2. Kas alergiškas medikamentui M, tas alergiškas ir medikamentui M';

Iš jų 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:  $\text{alerg}(\text{Jonas}, \text{M}')$ . 1 žingsnis:

$$\frac{\forall x (\text{alerg}(x, \text{M}) \Rightarrow \text{alerg}(x, \text{M}'))}{\text{alerg}(\text{Jonas}, \text{M}) \Rightarrow \text{alerg}(\text{Jonas}, \text{M}')} \quad x \text{ keičiamas konstanta Jonas}$$

2 žingsnis. Taikoma įrodymo taisyklė *modus ponens* (lot. teigimo būdas):

$$\frac{\begin{array}{c} \text{alerg}(\text{Jonas}, \text{M}) \\ \text{alerg}(\text{Jonas}, \text{M}) \Rightarrow \text{alerg}(\text{Jonas}, \text{M}') \\ \hline \text{alerg}(\text{Jonas}, \text{M}') \end{array}}{\text{Q}} \quad \frac{\text{P}}{\text{P} \Rightarrow \text{Q}} \quad \text{Q}$$

Tokiu būdu išvedama, kad Jonas yra alergiškas medikamentui M', todėl Jonui draudžiama skirti M'. Matematinėje logikoje nagrinėjama tik įrodymo struktūra, o ne turinys. Toliau kitas išvedimo pavyzdys su ta pačia struktūra. 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. 1 žingsnis

$$\frac{\forall x (\text{žmogus}(x) \Rightarrow \text{mirtingas}(x))}{\text{žmogus}(\text{Sokratas}) \Rightarrow \text{mirtingas}(\text{Sokratas})} \quad x \text{ keičiamas į Sokratas}$$

Pastaba: Kintamojo x keitimas konstanta A yra suprantamas kaip išvedimo taisyklė, vadinama *universalia instanciacija* (*universal instantiation*; žr., pvz., Klenk 2011). Išraiška  $\{A/x\}$  vadinama *keitiniu*. Rašoma:

$$\frac{\forall x P(x)}{P(A)} \quad \{A/x\}$$

2 žingsnis

$$\frac{\begin{array}{c} \text{žmogus}(\text{Sokratas}) \\ \text{žmogus}(\text{Sokratas}) \Rightarrow \text{mirtingas}(\text{Sokratas}) \\ \hline \text{mirtingas}(\text{Sokratas}) \end{array}}{\text{Taikoma modus ponens}}$$

Tokiu būdu išvesta, kad Sokratas 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:  $\neg \text{mirtingas}(\text{Dzeusas})$ . 1 žingsnis

$$\frac{\forall x (\text{dievas}(x) \Rightarrow \neg \text{mirtingas}(x))}{\text{dievas}(\text{Dzeusas}) \Rightarrow \neg \text{mirtingas}(\text{Dzeusas})} \quad x \text{ keičiamas į Dzeusas}$$

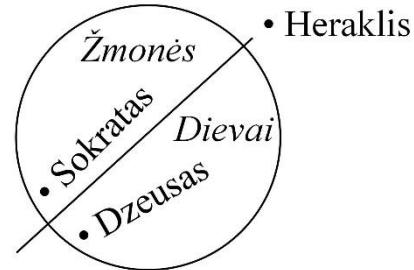
2 žingsnis

$$\begin{aligned} &\text{dievas}(\text{Dzeusas}) \\ &\underline{\text{dievas}(\text{Dzeusas}) \Rightarrow \neg \text{mirtingas}(\text{Dzeusas})} \quad \text{Taikoma } modus\ ponens \\ &\neg \text{mirtingas}(\text{Dzeusas}) \end{aligned}$$

Tokiu būdu išvesta, kad Dzeusas yra nemirtingas. Teiginys „Sokratus yra nemirtingas“ nėra išvedamas, nes trūksta teiginio „Sokratus yra dievas“ pradinių teiginių aibėje.

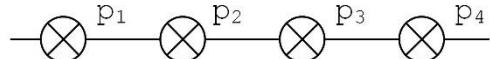
Remiantis požymiu mirtingas ar nemirtingas galima klasifikuoti būtybes į dvi aibes: žmonės ir dievai (19.11 pav.). Tačiau pagal tokią klasifikaciją yra problematiškas žinių modeliavimo klausimas – kuriai klasei priskirti pusdievius, pavyzdžiui, Heraklį.

19.11 pav. Aibių Žmonės ir Dievai sajunga yra predikato mirtingas apibrėžimo sritis



Logika nėra vaistas savaimė – automatiškai ji nerodo uždavinio sprendimo kelio. Tegu nedega keturių lempučių girlianda (19.12 pav.). Kuri lemputė ar net kelios yra perdegusios? Logikos kalba  $p_1 \& p_2 \& p_3 \& p_4 = f$ . Čia  $p_i = f$  žymi, kad lemputė  $p_i$  perdegusi. Kadangi girlianda nedega, tai išplaukia, kad egzistuoja toks  $i$ , jog  $p_i = f$ . Formaliai,  $\exists i p_i = f$ . Bet neišku, kuris  $i$ . Gali būti keletas tokų  $i$ . Gali būti  $p_1 = f$ , arba  $p_2 = f$ , arba  $p_3 = f$ , arba  $p_4 = f$ , arba  $p_1 = f \& p_2 = f$ , arba  $p_1 = f \& p_3 = f$  ir t. t., nes gali būti perdegusios kelios lemputės.

19.12 pav. Girlianda nedega. Kuri lemputė(-s) perdegusi?



Logika (gr. λόγος, logos – žodis, reikšmė) – filosofijos mokslo šaka, tirianti priimtinus samprotavimo būdus; plačiaja prasme – taisyklingas mastymas, samprotavimų eiga, sveikas protas, vidinis dėsningumas. Šnekamojoje kalboje logika dažniausiai vadinamas samprotavimų analizavimas. Tradiciškai logika buvo mokoma kaip filosofijos dalis, bet jau du šimtmečius logika studijuojama ir kaip matematikos, o paskutiniai dešimtmečiai – kaip kompiuterių mokslo dalis (<https://lt.wikipedia.org/wiki/Logika>).

**Pratimas.** Duota produkcija Mama (X, Y) & Tétis (Y, Z)  $\Rightarrow$  Senelė (X, Z) ir du faktai: „Adelė yra Broniaus mama“: Mama (Adelė, Bronius), „Bronius yra Dainiaus tétis“: Tétis (Bronius, Dainius). Įrodykite tiesioginiu ir atbuliniu išvedimu teigini: „Adelė yra Dainiaus senelė“: Senelė (Adelė, Dainius).

## 20. Ekspertinės sistemos

Šis skyrius grindžiamas atbulinio išvedimo produkcijų sistemoje (*backward chaining*) pavyzdžiu iš Sawyer ir Foster (1986) vadovėlio.

Literatūroje pateikiamos įvairio *ekspertinės sistemos* (ES) (*expert system*) apibrėžtys. 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 kaip diagnozė, prognozė, konfigūravimas, sprendimų priemimas, planavimas ir kt., kurių sprendimas paprastai reikalauja žmoniškojo intelekto.

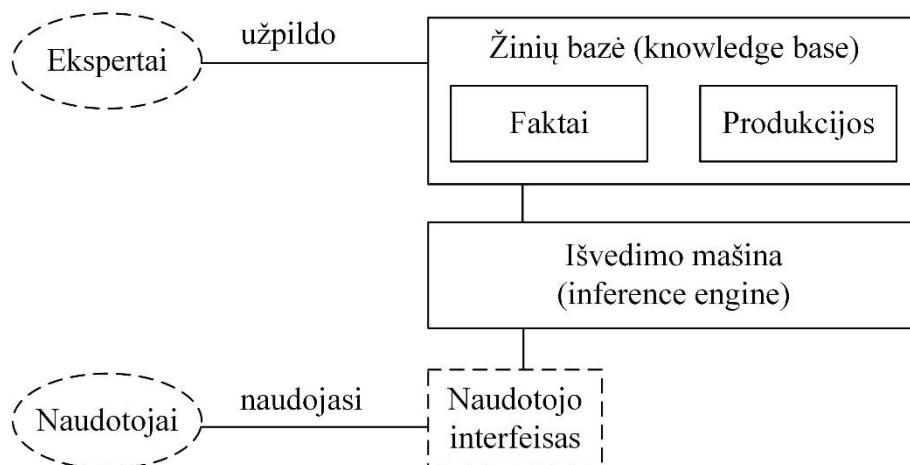
Terminas „ekspertinė sistema“ išpopuliarėjo 1980–1990 m. Jis paplito iš dalies dėl rinkodaros priežasčių. Šiuo metu populiарesnis sinonimas – *žiniomis grindžiama sistema* (*knowledge-based system*). Pastarasis terminas prigijo su antros kartos ekspertinėmis sistemomis, kuriose visiškai buvo atskirta *žinių bazė* nuo *išvedimo mašinos*.

Terminas „*expert system shell*“ atitinka neužpildytą žiniomis ekspertinę sistemą, o „*expert system*“ – užpildytą. Tarp šių kraštutinių gali būti 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 sistemai 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 nagrinėjamų uždaviniių aibę arba juos spręsti detaliau, turėtų pakakti papildyti žinių bazę naujomis žiniomis.

Bendroji ekspertinės sistemos architektūra yra parodyta 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 sistemose naudotojai taip pat gali būti ir ekspertais, kurie užpildo ES žinių bazę *faktais* ir *produkcinėmis taisykliemis*. 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 bazę – DB lenteles. Atskyrimu siekiama, kad žinių bazę ir išvedimo mašiną galima būtų keisti nepriklausomai nuo viena kitos. Tačiau tai tėra siekiamybė, realiose sistemose nepavyksta visiškai atskirti šių dviejų architektūros dalių. Čia galima palyginti su sunkumais, kurie kyla naudojant vieno gamintojo DBVS su kito gamintojo DB lentelėmis.

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

Toliau aprašoma demonstracinė ekspertinė sistema iš (Sawyer, Foster 1986). Pavyzdys iliustruoja žmogaus gyvenimo trukmės prognozę. Faktų bazę sudaro 20.2 pav. pateikti faktai. Sistemos naudotojas pats pasirenka variantus, kurie paryškinti. Objekto (pvz., **amžius**) reikšmė yra simbolinių eilutė.

1) amžius=	2) lytis=	3) svoris=
1. '25_ir_mažiau' 2. ' <b>25-55</b> ' 3. '55_ir_daugiau'	1. 'vyras' 2. ' <b>moteris</b> '	1. ' <b>55_ir_mažiau</b> ' 2. '55-85' 3. '85_ir_daugiau'
4) sudėjimas=	5) cholesterolis=	6) druska=
1. ' <b>smulkus</b> ' 2. 'stambus'	1. ' <b>norma</b> ' 2. daug'	1. ' <b>norma</b> ' 2. 'daug'
7) rūko=	8) charakteris=	9) alkoholis=
1. 'taip' 2. ' <b>ne</b> '	1. ' <b>agresyvus</b> ' 2. 'švelnus'	1. 'nevartoja' 2. ' <b>vidutiniškai</b> ' 3. 'daug'

20.2 pav. Demonstracinės ekspertinės sistemos faktai. Naudotojo pasirinkimai paryškinti

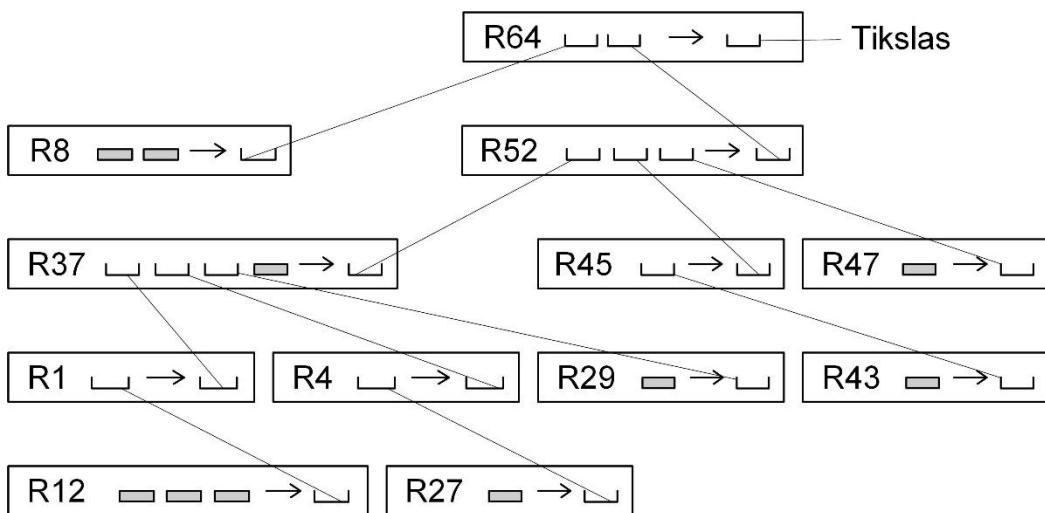
Žinių inžinierius (*knowledge engineer*), sudarantis žinių bazę ir užduodantis reikšmių režius, turi išmanysti dalykinę sritį. Toliau 20.3 lentelėje pavaizduotos produkcijos kurios yra žinių bazėje ir naudojamos gyvenimo trukmės išvedimo medyje. Pavyzdyje iš (Sawyer, Foster 1986) yra apie 100 produkcijų.

R1	Jeigu santykinis svoris normalus, tai bendras įvertinimas „taip“. <b>if</b> santykinis_svoris='normalus' <b>then</b> įvertinimas='taip'
R4	Jeigu širdies sustojimo rizika yra vidutinė, tai širdies pavojus žemas. <b>if</b> širdies_sustojimo_rizika='vidutinė' <b>then</b> širdies_pavojus='žemas'
R8	Jeigu amžius 25-55 metai ir lytis moteris, tai bendra gyvenimo trukmė 67 metai. <b>if</b> amžius='25-55' & lytis='moteris' <b>then</b> bendra_gyvenimo_trukmė='67'
R12	Jeigu svoris ne didesnis nei 55 kilogramai, sudėjimas smulkus ir lytis moteris, tai santykinis svoris yra normalus. <b>if</b> svoris='55_ir_mažiau' & sudėjimas='smulkus' & lytis='moteris' <b>then</b> santykinis_svoris='normalus'
R27	Jeigu cholesterolio kiekis kraujuje normos ribose, tai širdies sustojimo rizika vidutinė. <b>if</b> cholesterolis='norma' <b>then</b> širdies_sustojimo_rizika='vidutinė'

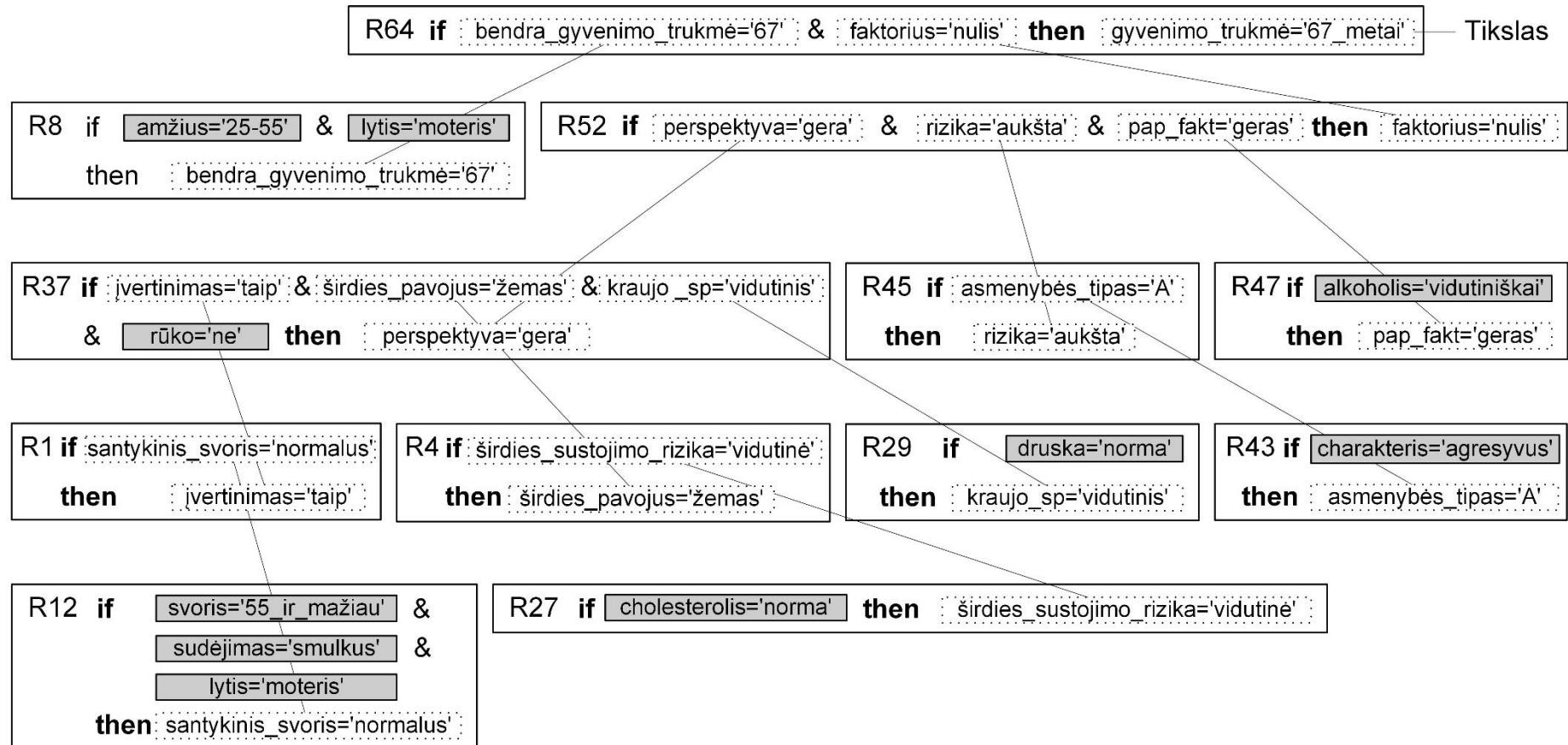
R29	Jeigu druskos suvartojimas yra normos ribose, tai kraujo spaudimas vidutinis. <b>if druska='norma' then kraujo_sp='vidutinis'</b>
R37	Jeigu bendras įvertinimas yra „taip“, širdies pavojus yra žemas, kraujo spaudimas yra vidutinis ir nerūko, tai perspektyva gera. <b>if įvertinimas='taip' &amp; širdies_pavojus='žemas' &amp; kraujo_sp='vidutinis' &amp; rūko='ne' then perspektyva='gera'</b>
R43	Jeigu charakteris agresyvus, tai asmenybės tipas A. <b>if charakteris='agresyvus' then asmenybės_tipas='A'</b>
R45	Jeigu asmenybės tipas A, tai rizikos lygis yra aukštas. <b>if asmenybės_tipas='A' then rizika='aukšta'</b>
R47	Jeigu alkoholio vartoja vidutiniškai, tai papildomas faktorius yra geras. <b>if alkoholis='vidutiniškai' then pap_fakt='geras'</b>
R52	Jeigu perspektyva yra gera, rizikos lygis yra aukštas ir papildomas faktorius yra geras, tai faktorius yra nulis. <b>if perspektyva='gera' &amp; rizika='aukšta' &amp; pap_fakt='geras' then faktorius='nulis'</b>
R64	Jeigu bendra gyvenimo trukmė yra 67 metai ir faktorius yra nulis, tai prognozuojama gyvenimo trukmė yra 67 metai. <b>if bendra_gyvenimo_trukmė='67' &amp; faktorius='nulis' then gyvenimo_trukmė='67_metai'</b>

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 parodytas 20.5 pav. Loginis išvedimas atliekamas atbulinio išvedimo algoritmu (*backward chaining*). Tikslo objektas yra gyvenimo\_trukmė.



20.4 pav. Supaprastintas išvedimo medis, vaizduojantis išvedimą 20.5 pav. Pilki stačiakampiai žymi faktus



20.5 pav. Paieškos medis demonstracinėje ekspertinėje sistemoje. Pilki stačiakampiai žymi pradinius faktus. Jie toliau nėra išvedinėjami – tai lapai. Medžio šaknyje yra tikslas **gyvenimo\_trukmė='67\_metal'**. Tai prognozuojama gyvenimo trukmė smulkaus sudėjimo agresyviai nerūkančiai 25-55 metų moteriai, saikingai vartojančiai druską, sveriančią 55 kilogramus ar mažiau ir normaliu cholesterolio kiekiu kraujuje.

Demonstracinė ekspertinė sistema ieško jai pateikto teiginio kaip tikslų įrodymo. Jeigu faktū 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 su taisyklėmis, pavaizduotomis teiginį logika (žr. 18 skyrių). Šio algoritmo sudėtingumas yra polinominis. Tai mažesnis sudėtingumas negu eksponentinis, kurį naudoja Prolog kalbos interpretatorius. Polinominj sudėtingumą salygoja tai, kad produkcijose yra  $o_i = \text{reikšmė}_i$  pavidalo faktai, o ne predikatai, pvz.,  $P_i(x,y)$ . Kaip minėta, produkcijos yra tokio pavidalo:

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

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 pavidalą:

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

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

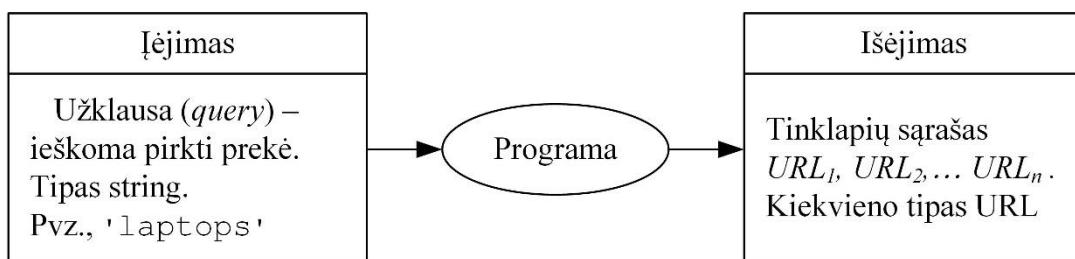
Demonstracinių ekspertinės sistemos produkcijų aibė yra realizuota sąrašu, kurio elementas yra produkcija. Savo ruožtu šios produkcijos kairioji pusė yra porų  $o_i=r_i$  sąrašas. Faktų bazė taip pat vaizduojama porų  $o_i=r_i$  sąrašu.

Naudotojo interfeiso realizacijos ypatybė yra tai, kad objekto duomenų struktūroje saugomas ir klausimas, kuris yra simbolų eilutė. Šį 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 (deep)* žinių vaizdavimas. Informacinėje sistemoje naudojamas *plokščias (flat)* duomenų vaizdavimas, pvz., lentelės duomenų bazių valdymo sistemoje.

## 21. Internetinė parduotuvė

Vadovaujamės (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 jėjimo ir išėjimo. Programos jėjimas yra užklausa apie ieškomą pirkti prekę. Jos tipas yra simbolų eilutė, string. Programos išėjimas yra sąrašas  $URL_1$ ,  $URL_2$ , ...,  $URL_n$  tinklapiu, kuriuose parduodama ieškoma prekė (21.1 pav.). Pavyzdžiui, užklausiamu 'laptops' ir gaunami tinklapiai, kuriuose parduodami nešiojami kompiuteriai.



21.1 pav. Internetinės parduotuvės programos jėjimas ir išėjimas

Programa suprantama kaip kompiuterinis agentas, kuris naršo po internetą ir ieško vartotojo nurodytos prekės (21.2 pav.). Apie kompiuterinių agentų klasifikavimą plačiau žr., pavyzdžiui, (Russell, Norvig 2003) 2 skyrių „Intelektualūs agentai“.



21.2 pav. Kompiuterinio agento jėjimas ir išėjimas

Žmogus skiriasi nuo kompiuterinio agento:

- žmogus mato tinklapio vaizdą, sieja '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 rodytų prekių kategorijas ne žodžiais, o piktogramomis, tai tik žmogus sugebėtų atskirti, kurioje kategorijoje gali būti ieškomos 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 (*completely automated public Turing test to tell computers and humans apart*). Žmogus slaptažodį atpažsta, o kompiuteris ne.

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

Žiūrėdamas į tinklapį 21.3 pav. žmogus sugeba padaryti išvadą, kad Jame siūloma pirkti nešiojamus kompiuterius. Programai tokią išvadą padaryti sunkiau, bet įmanoma. Tam būtina susieti simbolų eilutę 'Computers' su nešiojamais kompiuteriais (tegu užklausa 'laptops').

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

## 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, kurį žmogus suvokia kaip internetinę parduotuvę.  
Apačioje HTML tekstas. Mes kursime parduotuvę adresu  
<http://www.gen-store.com> ir pavadinsime GenStore

### 1 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 siūloma pirkti. Pavyzdžiu, tinklapis yra nerelevantinis, jeigu tame kalbama apie baldus arba apžvelgiami nešiojamieji kompiuteriai. Reikalavimas tinklapui būti relevantišku pasiūlymu išreiškiamas predikatu:

$$\text{RelevantOffer}(\text{page}, \text{url}, \text{query}) \Leftrightarrow \\ \text{Relevant}(\text{page}, \text{url}, \text{query}) \And \text{Offer}(\text{page})$$

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

### 2 etapas. Predikatas **Offer**

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

$$\begin{aligned} \text{Offer}(\text{page}) &\Leftrightarrow (\text{InTag}('a', \text{str}, \text{page}) \vee \\ &\quad \text{InTag}('form', \text{str}, \text{page})) \\ &\quad \& (\text{In}('buy', \text{str}) \vee \text{In}('price', \text{str})) \\ \text{InTag}(\text{tag}, \text{str}, \text{page}) &\Leftrightarrow \text{In}('<' + \text{tag} + \text{str} + '</' + \text{tag} + '>', \text{page}) \\ \text{In}(\text{sub}, \text{str}) &\Leftrightarrow \exists i \text{ str}[i:i+\text{Length}(\text{sub})-1] = \text{sub} \end{aligned}$$

Čia predikatas **In** (*sub, str*) išreiškia, kad simbolių eilutė *sub* įeina į simbolių eilutę *str*, o simbolis „+“ žymi eilučių konkatenacijos operaciją. Pavyzdžiui:

```
In('KAD', 'ABRAKADABRA') = true  
In('laptop', 'ABRAKADABRA') = false  
'ABC' + 'DEFG' = 'ABCDEFG'
```

Šiame pavyzdyme pasiūlymas pirkti užkoduotas labai supaprastintai – žodžiais 'buy' ir 'price'. Pirma, 'buy' arba 'price' sutinkamas žymenyje ir, antra, tas žymuo sutinkamas tinklapyje *page*. Ši specifikacija nėra labai detali. Pavyzdžiui, ar turi prasmę didžiosios raidės eilutėse 'Buy' ir 'Price', asmenavimas, daugiskaita ir pan.? Rimta detalizacija turi apimti kuo daugiau žinių apie sąvoką **Offer**, įskaitant ir 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 kuriamą *GenStore*, tenkina predikatą **OnlineStores**, nes yra internetinės prekyvietės su žinomais interneto adresais.

Kuriama elektroninė parduotuvė turi žinias apie tam tikras prekyvietes. Pavyzdžiui:

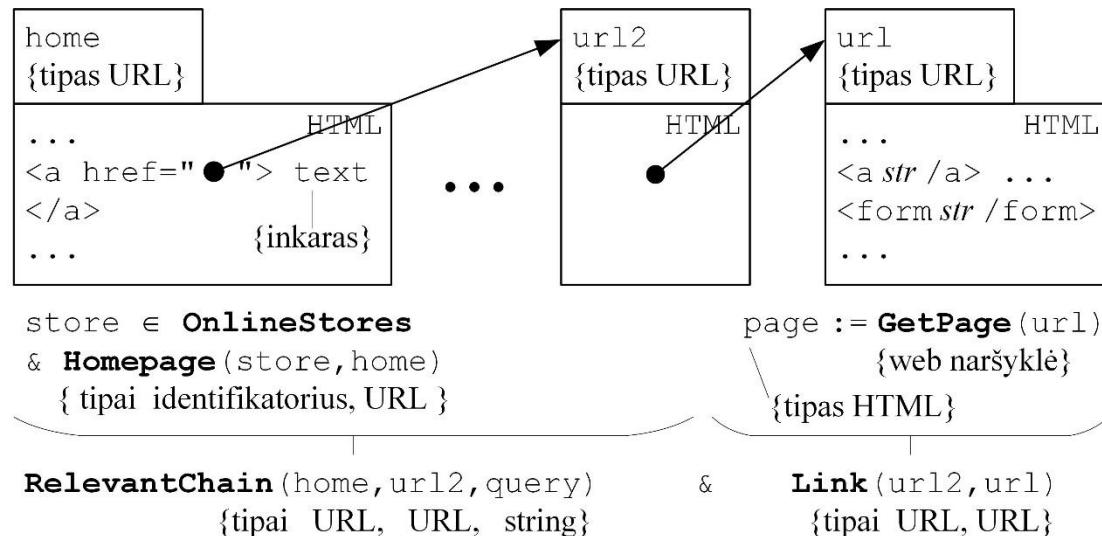
```
Amazon ∈ OnlineStores  
    & Homepage(Amazon, 'http://www.amazon.com')  
Ebay ∈ OnlineStores  
    & Homepage(Ebay, 'http://www.ebay.com')  
GenStore ∈ OnlineStores  
    & Homepage(GenStore, 'http://www.gen-store.com')
```

Čia vardai Amazon, Ebay, GenStore žymi individualias sąvokas. Jų tipas yra „identifikatorius“. Šiais trimis faktais pasakoma, kad pasaulyje yra tinklapių, kurie yra prekyvietės.

### 4 etapas. Predikatas **Relevant**

Kuriamos parduotuvės idėja parodyta 21.4 pav. Stačiakampyje kairėje *text* tarp HTML žymenų „a“ yra vadinamas *inkaru* (*anchor*). Inkarai matomi kompiuterio ekrane, pvz., „Computers“, „Books“ (21.3 pav.). Toliau detalizuojamas predikatas **Relevant** (21.4 pav.):

```
Relevant(page, url, query) ⇔ ∃ store ∃ home (  
    store ∈ OnlineStores & Homepage(store, home)  
    & ∃ url2 RelevantChain(home, url2, query)  
    & Link(url2, url) & page = GetPage(url) )  
(21.1)
```



21.4 pav. Aiškinama kuriamos parduotuvės specifikacija. Pavyzdžiui,  
home = 'http://www.gen-store.com' ir store = GenStore

Kuriamo agento – programos GenStore – specifikacijos (21.4 pav.) esmė 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 tipai yra URL. Predikatu **Link** išreiškiama, kad tinklapyje from\_url yra nuoroda į to\_url tinklapį (21.4 pav.).

Tokiu būdu titulinis tinklapis yra home ir tikslo tinklapis – url. Priskyrimas page := **GetPage**(url) žymi, kad page bus priskirtas HTML tekstas esantis tinklapyje adresu url. Šis priskyrimą galima palyginti su naršyklės iškvietimu.

Predikatu **Homepage**(store, home) išreiškiama, kad prekyvietė vardu store turi titulinį tinklapį home.

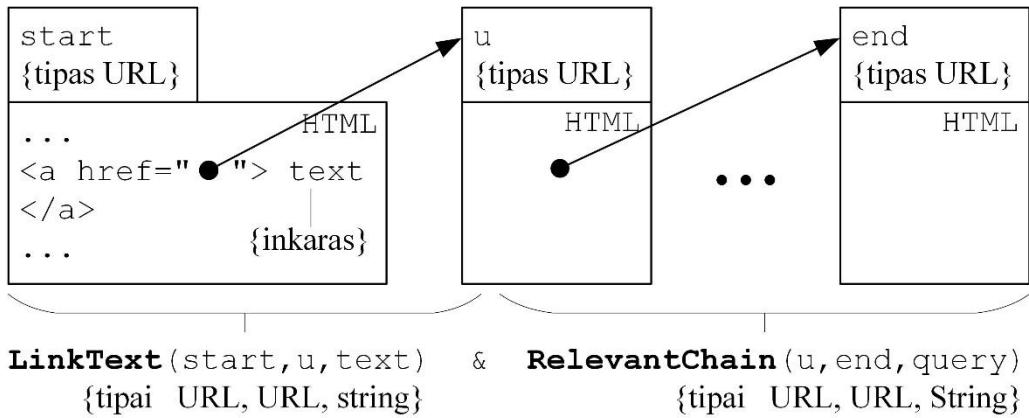
Taigi, perrenkami visi relevantiniai tinklapiai url, pasiekiami einant nuorodomis iš turimų prekyviečių store ∈ **OnlineStores** titulinio tinklapio home. Predikatas **offer** iš šių relevantinių tinklapių atsijoja tik tuos, kuriuose siūloma pirkti.

### 5 etapas. Predikatas **RelevantChain**

Detalizuojamas predikatas **RelevantChain**(home, url2, query); žr. (21.1) ir 21.5 pav. Iš tinklapio start, kuriame esame, galima pasiekti tinklapį u tik tada, kai start žymė (inkaras, anchor, t. y. tekstas tarp HTML žymenų „a“) text yra relevantinis užklausai query. Ir taip toliau rekursyviai. Grandine relevantinių nuorodų, perrenkami tinklapiai end.

$$\begin{aligned}
 \mathbf{RelevantChain}(\text{start}, \text{end}, \text{query}) &\Leftrightarrow (\text{start} = \text{end}) \\
 \vee (\exists u \exists \text{text} \mathbf{LinkText}(\text{start}, u, \text{text}) & \quad (21.2) \\
 &\quad \& \mathbf{RelevantCategoryName}(\text{query}, \text{text}) \\
 &\quad \& \mathbf{RelevantChain}(u, \text{end}, \text{query}) )
 \end{aligned}$$

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' (21.3 pav.). Toliau rekursyviai ieškomi tinklapiai `end` iš `http://www.gen-store.com/compu`.

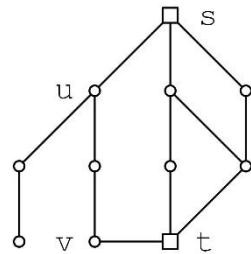
Verta priminti, kad specifikacijos rekursyviais predikatais yra naudojamos ir teorinėje informatikoje, ir dirbtiniame intelekte. Pavyzdžiui, grafe tiesioginės paieškos (*forward chaining*) nuo pradinės viršūnės `s` iki terminalinės `t` (21.6 pav.) specifikacija yra tokia:

$$\text{ForwardChaining}(s, t) \Leftrightarrow \text{Link}(s, u) \& \text{ForwardChaining}(u, t)$$

Atbulinės paieškos (*backward chaining*) specifikacija:

$$\text{BackwardChaining}(s, t) \Leftrightarrow \text{BackwardChaining}(s, v) \& \text{Link}(v, t)$$

21.6 pav. Paieška grafe nuo `s` iki `t` gali būti atliekama ir tiesioginės paieškos, ir atbulinės paieškos algoritmais



### 6 etapas. Žodžių priskyrimas kategorijoms

Iš pradžių aptarsime žodžių, kurie gali įeiti į užklausą, kategorizavimo problemą.

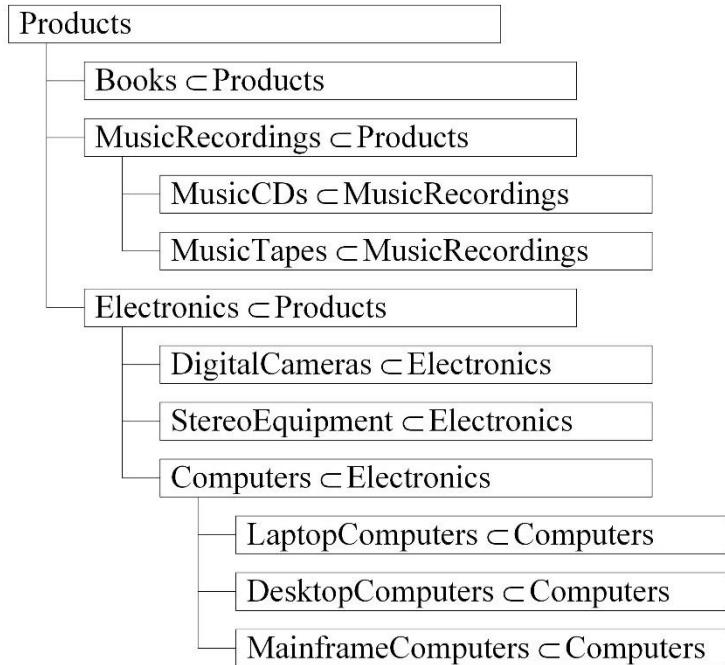
Galima pageidauti, kad užklausos forma būtų neformalizuota, pavyzdžiui `query='Man reikalingi laptopai'`. Žmogus gali užduoti užklausą ir kitaip žodžiais, nes nežino kategorijų, kurias išreiškia inkaras `text`. Be to, vienas žodis gali būti priskiriamas kelioms kategorijoms. Pavyzdžiui, „lapiukas“ kategorizuojant 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 reikšmė 'Computers', o query – 'Man reikalingi laptopai'.

Kategorizavimui naudojami du medžiai: 1) produktų kategorijų hierarchija ir 2) žodžių priskyrimo kategorijoms medis. Produktų kategorijų *is-a* medžio pavyzdys yra 21.7 pav.

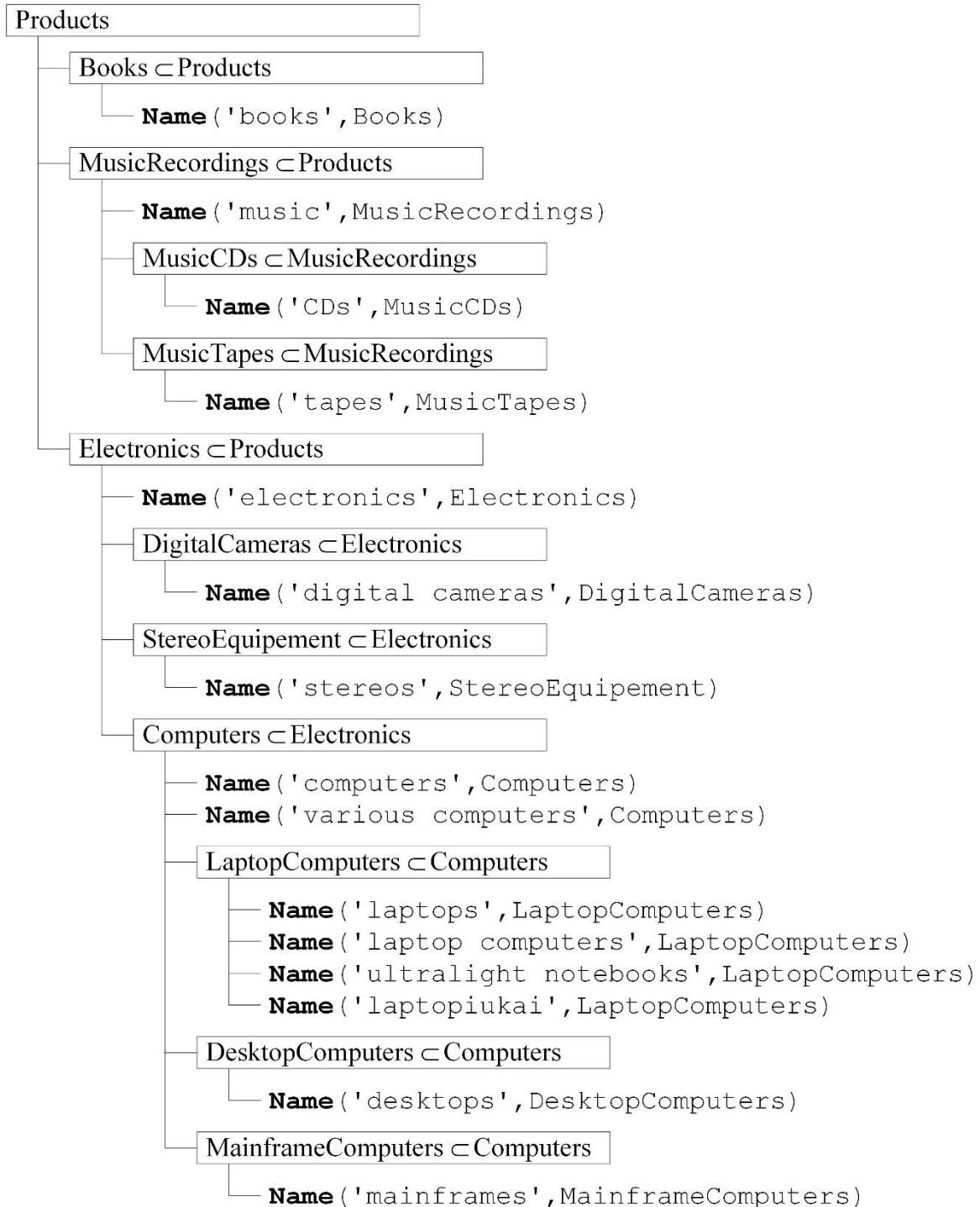


21.7 pav. Produktų kategorijų hierarchija. Medžio briaunos žymi ryšį *is-a*

Žodžiai, kurie jeina į užklausą query, yra priskiriami kategorijoms (21.8 pav). Natūralioje kalboje žodžių yra daug – dešimtys tūkstančių. Kategorijų taip pat gali būti daug. Pavyzdžiui, tūkstančiai kategorijų yra muitinės tarifų lentelėse, rengiamose duomenų baze ir knyga. Kategorijas gali įvardinti ir žodžių junginiai, pvz., 'ultralight notebooks'. Pavyzdžiui, žodis „book“, atitinka sąvoką (kategoriją) Books. Tai išreiškiama predikatu **Name** ('book', Books). Žargono arba mažybiniai žodžiai taip pat priskiriami kategorijoms, pvz., **Name** ('laptopiukai', LaptopComputers). Išvystytą programą turėtų suprasti linksnius ir žodžių darybą.

Vienas žodis gali būti priskiriamas kelioms kategorijoms, pvz., 'lapiukas'. Tokiu būdu atkreipiama 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).



21.8 pav. Žodžių priskyrimas kategorijoms

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:

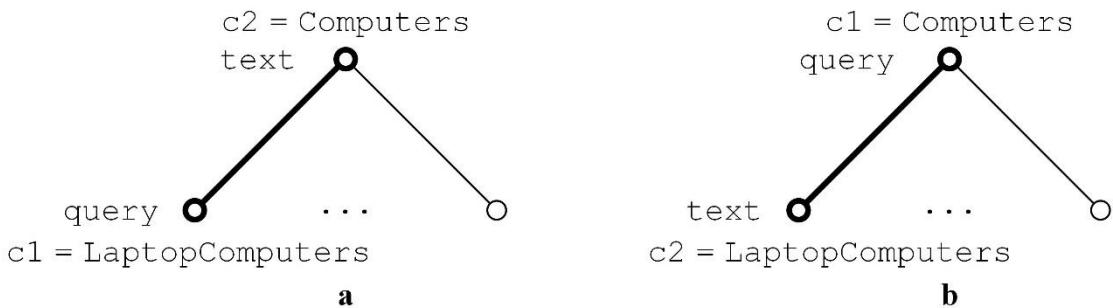
1. text ir query įvardina tą pačią kategoriją, pvz., text='laptop computers' ir 'laptops';
2. text įvardina bendresnę kategoriją, pvz., text='computers' (21.9 pav.);

3. text įvardina labiau specifinę kategoriją, pvz., `text='ultralight notebooks'` (21.10 pav.).

Predikatas apibrėžiamas šitaip:

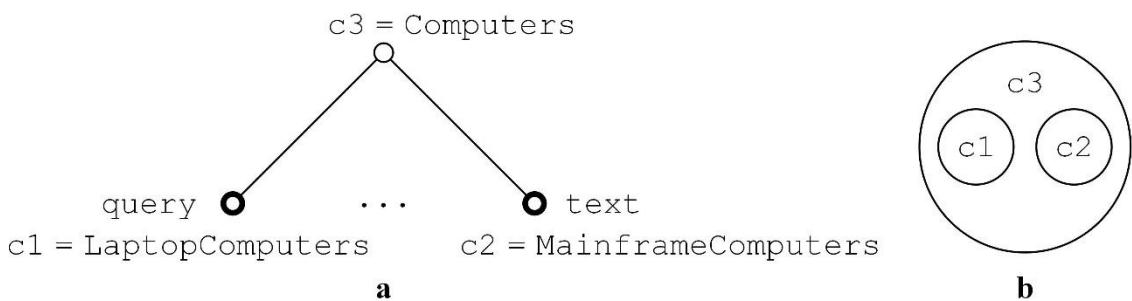
**RelevantCategoryName**(query, text)  $\Leftrightarrow$   
 $\exists c_1 \exists c_2 \text{ Name(query, } c_1) \& \text{ Name(text, } c_2) \& (c_1=c_2 \vee c_1 \subset c_2 \vee c_1 \supset c_2)$

Atvejis  $c_1 \subset c_2$  parodytas 21.9 a pav., o atvejis  $c_1 \supset c_2$  21.9 b pav.



21.9 pav. Du atvejai: a)  $c_1 \subset c_2$  ir b)  $c_1 \supset c_2$

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  $c_3 = \text{Computers}$  (21.10 a pav.). Kategorijos  $c_1$  ir  $c_2$  kaip aibės yra aibės  $c_3$  poaibiai, bet nesikerta (21.10 b pav.).



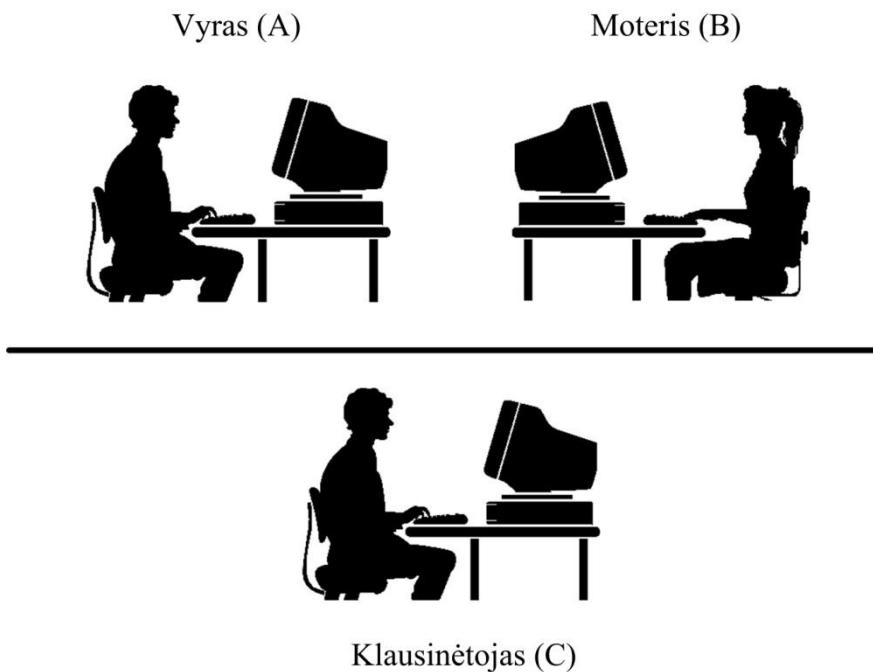
21.10 pav. a) Predikatas netenkinamas, kai nėra nei  $c_1=c_2$ , nei  $c_1 \subset c_2$ , nei  $c_1 \supset c_2$ . b) Kategorijos  $c_1$  ir  $c_2$  nepalyginamos

Užklausa gali būti, pavyzdžiui, „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 vadoveliuose (Luger 2005), ir interne (žr. [http://en.wikipedia.org/wiki/Turing\\_test](http://en.wikipedia.org/wiki/Turing_test)).

Alanas Tiuringas [Turing 1950] aptaria galimybę atsakyti į klausimą „Ar gali mašinos mąstyti?“. Jis nurodo, kad iškeltas klausimas yra dviprasmiškas ir bando ji konkretizuoti. Tiuringas suformuluoją taip vadinamą *imitacijų žaidimą* (*imitation game*). Žaidime dalyvauja vyras (A), moteris (B) ir klausinėtojas (C). Pastarojo lytis nėra svarbi. Klausinėtojas neturi tiesioginio kontakto su pašnekovais (tuomet imitacinis žaidimas netektu prasmės). Jis yra atskirame kambaryje (22.1 pav.). Klausinėtojo tikslas yra nustatyti, kuris iš dviejų žaidėjų yra vyros ir kuris moteris. Tuo tarpu kitų žaidėjų tikslas – įtikinti klausinėtoją, kad jis arba ji yra moteris. A ir B neprivalo sakytis tiesos – jie gali meluoti. Taigi A tikslas yra apkvailinti klausinėtoją, o B – įtikinti klausinėtoją.



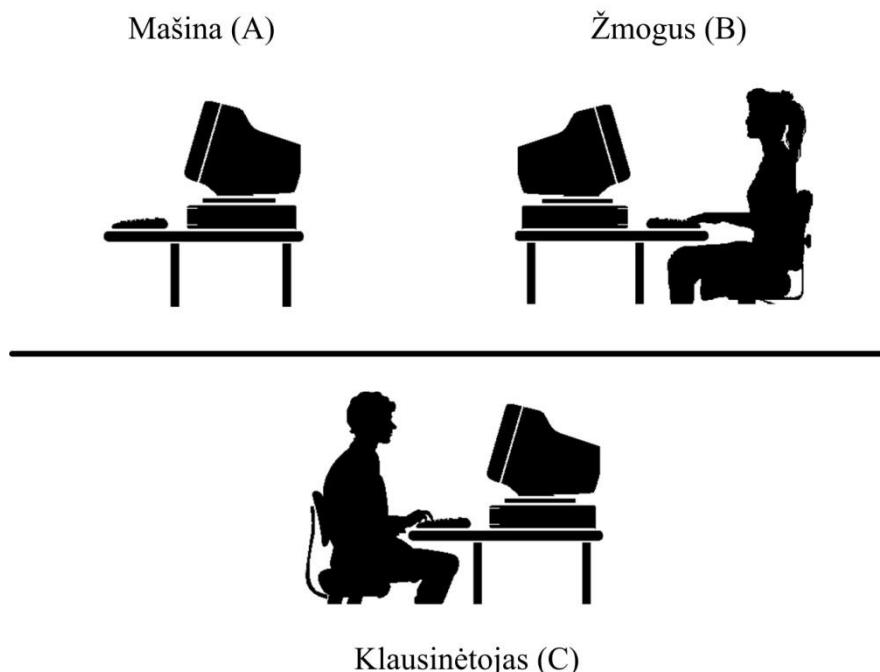
22.1 pav. Imitacinis žaidimas. Žaidime dalyvauja vyras (A), moteris (B) ir klausinėtojas (C). Klausinėtojo tikslas yra nustatyti, kuris A ir B yra vyros ir kuris moteris. Ir A, ir B siekia įtikinti klausinėtoją, kad jie yra moteris. Abu gali meluoti. Vyro A tikslas yra apkvailinti klausinėtoją

Klausimai užduodami ir atsakymai gaunami raštu arba ryšiais, pvz., „teletaip“ (šio meto sąlygomis – kompiuteriu). Taigi, klausinėtojas klausimus užduoda ir atsakymus gauna raštu. Klausimai gali būti užduodami bet kokie, pavyzdžiu: „Koks jūsų plaukų ilgis“.

Anot Tiuringo, klausimą „Ar gali mašinos mąstyti?“ galima perfrazuoti į jam ekvivalentų: „Kas atsitiks, jeigu vietoje A žais mašina? Ar klausinėtojas klys taip pat dažnai, kaip ir tada, kai žaidžia tik žmonės?“ (Tiuringas 1961). Vėliau Tiuringas 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 imitacijų žaidimą (o žaidėjas B bus žmogus)?“.

Pastebėsime, kad naujoje formuluočėje nebedalyvauja moteris, tačiau žaidėjų tikslai tie patys (22.2 pav.). Taigi Tiuringo testo paskirtis yra ne išsiaiškinti, ar konkreti mašina A gali imituoti moterį B imitaciniame žaidime, bet išsiaiškinti, ar mašina apskritai gali imituoti žmogų. Daugumoje vėlesnių Tiuringo testo formuluočių lytis yra ignoruojama ir laikoma, kad žaidime dalyvauja mašina A, žmogus B ir klausinėtojas C. Klausinėtojo tikslas yra nustatyti, kuris iš A ir B yra žmogus ir kuris mašina.



22.2 pav. Tiuringo testas yra suprantamas kaip imitacinis žaidimas. Klausinėtojas siekia išsiaiškinti, kuris iš A ir B yra žmogus ir kuris mašina. Mašina A siekia apkailinti klausinėtoją, kad ji žmogus

Esminis akcentas čia yra *imitacija*. Viena vertus, imitacino žaidimo prigimtis yra apgaulė: vyrui žaidime leidžiama sakyti viską, siekiant klausinėtoją padaryti klaidingas išvadas. Tuo tarpu moters tikslas yra padėti klausinėtojui išsiaiškinti tiesą. Žaidime, kuriame dalyvauja mašina (vietoje A) ir žmogus, tikslai tie patys. Mašina įtikinėja klausinėtoją, kad ji yra žmogus.

Dirbtinio intelekto filosofai Tiuringo testo formuluočę bandė perfrazuoti. Pavyzdžiu, Džonas Searle (žr. [http://en.wikipedia.org/wiki/Chinese\\_room](http://en.wikipedia.org/wiki/Chinese_room)) pasiūlė minties eksperimentą, kuris vadinas *kinų kambario argumentas* (*Chinese Room argument*). Žr. taip pat mokomajį vienos minutės filmuką <https://www.youtube.com/watch?v=TryOC83PH1g>, kurį sukūrė Atvirasis universitetas Anglioje (The Open University).

Žmogaus pašnekesį imituojančią programą Eliza dar 1976 m. pademonstravo J. Weizenbaum; žr. <http://en.wikipedia.org/wiki/ELIZA>. Vėlesnius pavyzdžius programas-robotus (*chat bot*) galima išbandyti internete, pvz., <http://nlp-addiction.com/eliza>.

Tiuringo klausimas „Ar gali mašinos mąstyti?“ panašus, pavyzdžiu, į klausimą „Kas pirmesnis, višta ar kiaušinis?“ tuo, kad tai filosofiniai klausimai. Jais keliamos filosofinės problemos. Jie neskirti atsakyti „taip“ ar „ne“ – tai ne logikos teiginiai.

## 23. Neįmanomumas pasiekti keletą tikslų

Šiame skyriuje nagrinėjamas klausimas „Kaip pasirinkti geriausią veiksmą siekiant keleto nesuderinamų tikslų?“ Mūsų pradiniai duomenys yra:

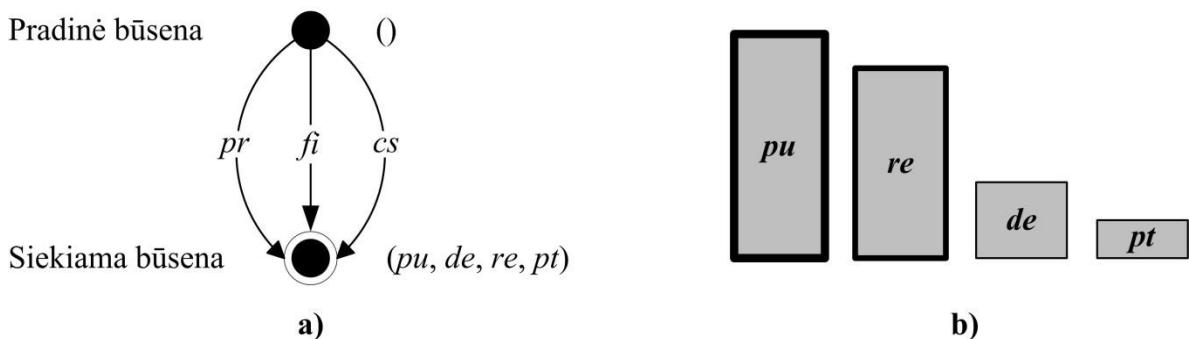
1. galimi veiksmai, tarp kurių reikia rinktis,
2. tikslai, kuriuos reikia pasiekti,
3. priežastiniai ryšiai tarp veiksmų ir tikslų. Gali būti vaizduojami produkcinėmis taisyklėmis,
4. tikslų svarba eilės tvarka.

Pateiksime pavyzdį, kuris modeliuoja argumentavimą teisėje; žr. (Bench-Capon, Prakken 2006).

### 23.1. Nusikaltėlio nubaudimo pavyzdys

**Pavyzdys.** Teisėjas turi nubausti (*punish – pu*) nusikaltėlį. Jis gali skirti vieną iš trijų bausmių: laisvės atėmimą (*imprisonment – pr*), baudą (*fine – fi*) arba viešieji darbai (*community service – cs*). Be nubaudimo *pu* yra dar trys tikslai: atgrasymas (*deterring the general public – de*), perauklėjimas (*rehabilitating the offender – re*) and ir visuomenės apsauga nuo nusikaltimų (*protecting society from crime – pt*). □

Taigi nubaudimas *pu* yra svarbiausias tikslas, o bausmės rūšis *pr*, *fi* arba *cs* priklauso nuo kitų tikslų svarbos. Teisėjo tikslų aibę sudaro keturi elementai, t. y.  $G = \{pu, de, re, pt\}$  (23.1 a pav.). Mes iliustruojame ir kokie galėtų būti aritmetiniai tikslų svoriai (23.1 b pav.).



23.1 pav. a) Nusikaltėlio nubaudimo uždavinys. b) Tikslų svorių pavyzdys

Reziumuojame, kas duota. Galimi trys veiksmai, t.y. bausmių rūšys:

1. laisvės atėmimas (*pr*)
2. bauda (*fi*)
3. viešieji (*cs*)

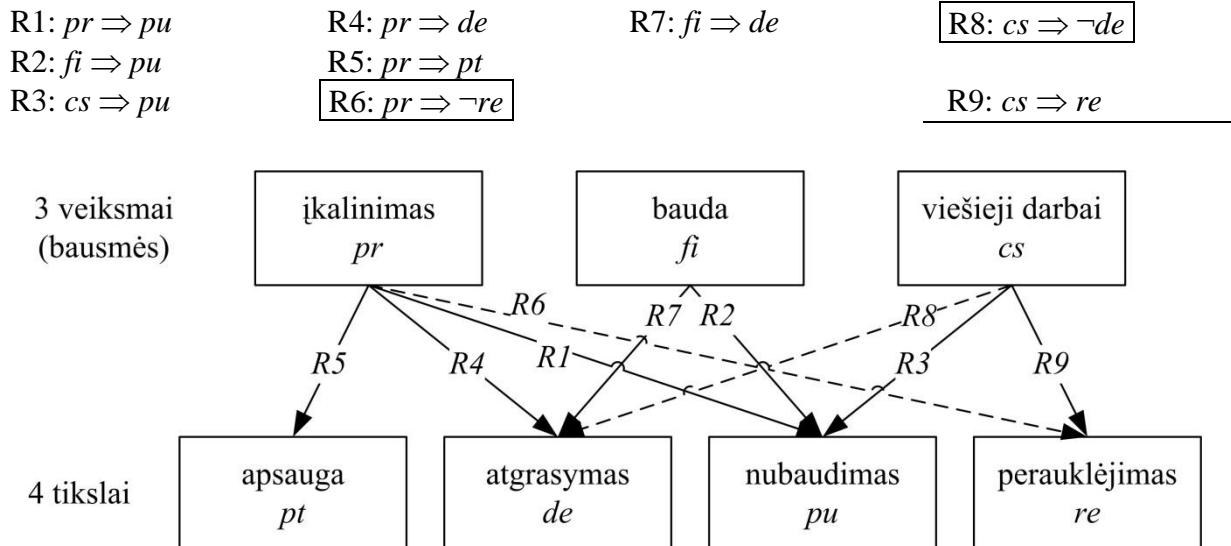
Siekiamā keturių tikslų:

1. nubaudimas (*pu*). Tai svarbiausias tikslas
2. atgrasymas (*de*)
3. perauklėjimas (*re*)
4. apsauga (*pt*)

Toliau apibrėžiama, kaip kiekvienas atskiras veiksmas sąlygoja kiekvieną tikslą:

1. Laisvės atėmimas,  $pr$ , skatina (*promotes*) atgrasymą,  $de$  (tai taisyklė R4:  $pr \Rightarrow de$ ) ir apsaugą,  $pt$  (R5:  $pr \Rightarrow pt$ ), bet žemina, t.y. turi neigiamą poveikį, (*demotes*) perauklijimui,  $re$  (R6:  $pr \Rightarrow \neg re$ ).
2. Bauda,  $fi$ , skatina atgrasymą,  $de$  (R7:  $fi \Rightarrow de$ ), bet neturi poveikio apsaugai ir perauklijimui.
3. Viešieji darbai,  $cs$ , skatina perauklijimą,  $re$  (tai taisyklė R9:  $cs \Rightarrow re$ ), bet žemina atgrasymą,  $de$  (R8:  $cs \Rightarrow \neg de$ ).

Reziumuojame taisykles (23.2 pav.):

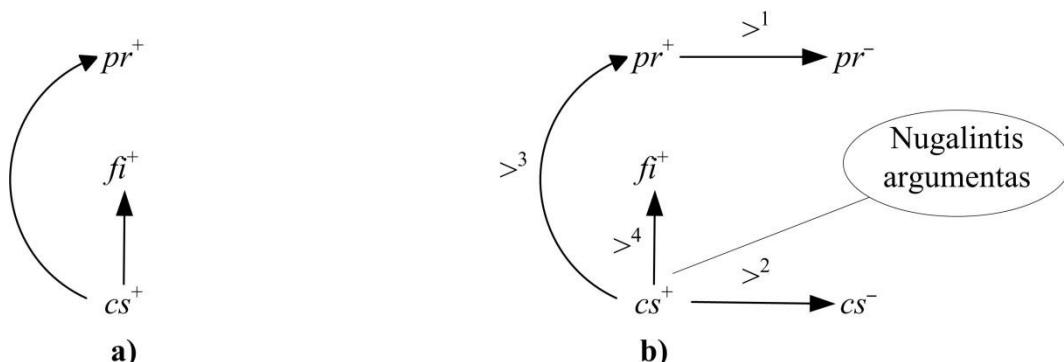


23.2 pav. Priežastiniai ryšiai tarp veiksmų ir tikslų. Punktyrinės briaunos  $R6$  ir  $R8$  vaizduoja neigiamą poveikį

Pastebėkime, kad visi keturi tikslai negali gūti pasiekiami. Kiekvienu veiksmu pasiekiami tik keletas tikslų. Be to, tiek  $pr$ , tiek  $cs$  išsaukia ir neigiamą poveikį (t.y. tikslo neigimą).

Teisėjo pasirinkimas iš trijų bausmių gali būti suvestas į klausimą: kuris veiksmas nubausti nusikaltėli yra geriausias?

Toliau argumentuojamas kiekvienas iš trijų veiksmų. Užbėgdami į priekį pasakysime, kad bus nuspręsta pasirinkti viešuosius darbus,  $cs$ . Argumentai už viešuosius darbus, žymimi  $cs^+$ , nurungs argumentus už įkalinimą  $pr^+$  ir argumentus už baudą  $fi^+$ . Šie nurungimai pavaizduoti dviem briaunomis argumentų nurungimo grafe 23.3 a pav. Briauna iš nurungėjo į nurungtajį.



23.3 pav. a) Argumentai už viešuosius darbus  $cs^+$  nurungia argumentus už įkalinimą  $pr^+$  ir baudą  $fi^+$ . b) Nurungimo grafas (defeat graph)

## 23.2. Veiksmų įvertinimas pagal skatinamą ir žeminamą tikslų santykį

Įvertinkime kiekvieną veiksmą; žr. 23.4 lentelę. Čia  $v$  yra įvertinimo funkcija, kurios reikšmė yra pora (teigiamų tikslų aibė, neigiamų tikslų aibė). Kiekvienas tikslas gali būti įvertintas vienodu svoriu 1 ir veiksmai įvertinti santykiu skatinamą, t.y. teigiamų tikslų skaičius prieš žeminamą, t.y. neigiamų tikslų skaičių.

Ivertinimas	(skatina, žemina)	{pu, de, pt, re}	Santykis	Skirtumas
$v(pr^+) =$	({pu, de, pt}, {re} )	(1, 1, 1, -1)	3:1	2
$v(fī^+) =$	({pu, de}, {Ø} )	(1, 1, 0, 0)	2:0	2
$v(cs^+) =$	({pu, re}, {de} )	(1, -1, 0, 1)	2:1	1

23.4 lentelė. Veiksmų  $pr$ ,  $fī$  ir  $cs$  įvertinimai

Geriausias santykis yra įkalinimo  $pr$ : 3:1, nes skirtumas tarp teigiamų tikslų ir neigiamų tikslų yra  $3-1 = 2$ . Tokiu pačiu skirtumu pasižymi ir  $fī$ : 2:0, nes  $2-0 = 2$ . O  $cs$  santykis 2:1 yra blogesnis, nes  $2-1 = 1$ , kas yra blogiau už 2. Tokiu būdu reziumuojame, kad  $cs$  būtų blogiausias, o ne geriausias pasirinkimas, bet tai tuo atveju, jeigu būtų vadovaujamas prielaida, kad visi tikslai lygūs pagal svarbumą. O mes pradėjome nuo prielaidos, kad egzistuoja svarbiausio tikslo savoka, kitais žodžiais, tikslai yra sutvarkyti ir  $pu$  yra svarbiausias.

## 23.3. Planas argumentavimui už viešuosius darbus cs

Pateiksime planą samprotavimui, kad  $cs$  yra geriausias veiksmas.

**1 žingsnis:**  $>^1$  ir  $>^2$ . Parodysime, kad argumentas už kiekvienos bausmės skyrimą nurungia argumentą tos bausmės neskirti. Kitaip tariant, kaltintojo argumentas nurungia gynėjo argumentą. Taigi parodysime, pirma, kad argumentas už įkalinimo skyrimą nurungia argumentą už įkalinimo neskyrimą,  $pr^+ >^1 pr^-$ , ir antra, kad argumentas už viešujų darbų skyrimą nurungia argumentą neskirti viešujų darbų,  $cs^+ >^2 cs^-$ ; žr. 23.3 b pav.

Irodoma samprotavimu, kad bausmės neskyrimu nepasiekiamas tikslas nubaudimas  $pu$ .

Argumento neskirti baudos  $fī$  nėra, nes bauda neturi neigiamo poveikio tikslams. Taigi šiame žingsnyje parodysime:

1.  $pr^+ >^1 pr^-$ . Tai nurungimas  $>^1$
2.  $cs^+ >^2 cs^-$ . Tai nurungimas  $>^2$

**2 žingsnis:**  $>^3$ . Bus pasirinktas antras pagal svarbą tikslas – perauklėjimas  $re$ . Tai pavyzdžio dalykinėje srityje – baudžiamojos teisenos – priimtas pasirinkimas. Tai yra už formaliosios logikos ribų išeinantis (ekstraloginis) pasirinkimas (*extralogical choice*).

Iš šio pasirinkimo išplaukia, kad perauklėjimas nusveria atgrasymą ir apsaugą kartu sudėjus, kitaip sakant,  $re >^3 de + pt$ . O iš to išplauks, kad viešieji darbai, kurių pasekmė perauklėjimas, nurungs įkalinimą,  $cs^+ >^3 pr^+$ .

**3 žingsnis:**  $>^4$ . Bus pasirinkta, kad perauklėjimas nors ir neužtikrinant atgrasymo nusveria atgrasymą, kitaip sakant,  $re - de >^4 de$ . Iš šio ekstraloginio pasirinkimo išplauks, kad viešieji darbai nurungs baudą,  $cs^+ >^4 fī^+$ .

Taigi  $cs^+$  bus nugalintis argumentas.

### 23.4. Argumentavimas už įkalinimą $pr^+ >^1 pr^-$

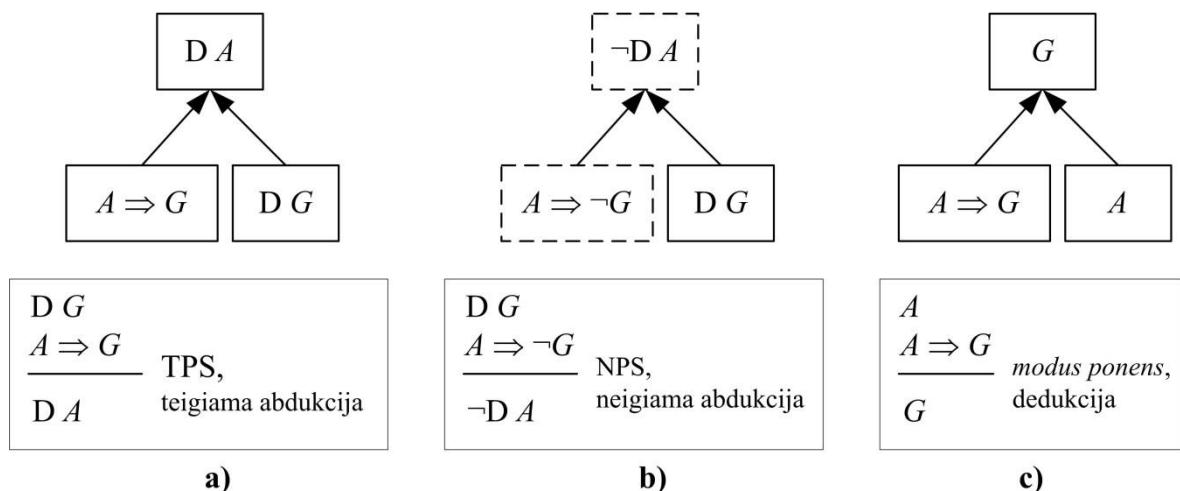
Įrodysime, kad argumentas skirti įkalinimą  $pr^+ >^1$  nurungia argumentą įkalinimo neskirtį  $pr^-$ . Įrodoma samprotavimu, kad įkalinimo neskyrimu  $pr^-$  nepasiekiamas  $pu$ , kuris ir yra pagrindinis tikslas – nubaudimas.

Tikslus užrašome  $G = \{ D pu, D de, D re, D pt \}$ , kur  $D$  žymi modalinės logikos modalinių operatorių ir  $D G$  yra skaitoma „siekти tikslą  $G$ “.

Samprotavimui naudojama išvedimo taisyklė, vadinama *teigiamu praktiniu silogizmu*:

1. Agentas  $P$  siekia tikslą  $G$ .
2. Jeigu  $P$  atliks veiksmą  $A$ , tai tikslas  $G$  bus pasiektas.
3. Todėl  $P$  turėtų atlikti veiksmą  $A$ .

Logikoje *teigiamas praktinis silogizmas* užrašomas kaip *teigiamos abdukcijos* taisyklė (23.5 a pav.). Tai atbulinio samprotavimo (abdukcijos) taisyklė. Mes ją naudojome aiškindami atbulinio išvedimo su produkcinėmis taisyklėmis (*backward chaining*) algoritmą. Ši taisyklė turi šaknis senovės Graikijos filosofo Aristotelio darbuose. Grafiškai ji parodyta 23.5 a pav.



23.5 pav. a) Teigiamas praktinis silogizmas, b) neigiamas praktinis silogizmas ir  
c) dedukcija

Kita argumentavime naudojama išvedimo taisyklė yra *neigiamas praktinis silogizmas* (23.5 b pav.):

1. Agentas  $P$  siekia tikslą  $G$ .
2. Jeigu  $P$  atliks veiksmą  $A$ , tai tikslas  $G$  nebus pasiektas.
3. Todėl  $P$  neturėtų atlikti veiksmą  $A$ .

Teigiamu praktiniu silogizmu vadovaujasi kaltintojas, o neigiamu praktiniu silogizmu – gynėjas. Kaltintojas argumentuoja veiksmo skyrimą  $D A$ , o gynėjas argumentuoja veiksmo neskyrimą  $\neg D A$ .

Yra svarbu paminėti, kad praktinio silogizmo taisyklėmis išvesti teiginiai gali būti nurungti. Tai sudaro skirtumą nuo samprotavimo vadovaujantis dedukcijos (nuo bendro prie atskiro) taisykle *modus monens*, kuri parodyta 23.5 c pav. Su nurungimo samprata mes jau buvome susidūrę atbulinio išvedimo algoritme, nors termino „nurungimas“ ir nenaudojome. Nurungimą atbuliniame išvedime iliustravo perėjimas prie aukščiau steke esančio tikslø, kai nepavykdavo išvesti žemesnio tikslø.

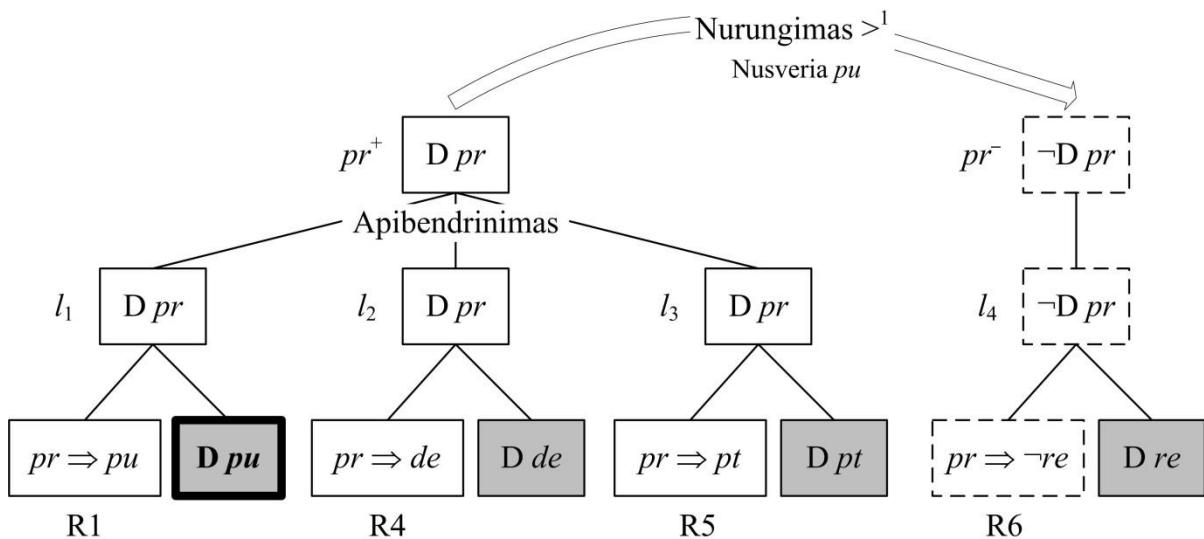
Kaltintojas gali pateikti net tris argumentus skirti įkalnimą *pr*. Pirma, kadangi *pr* skatina nubaudimą *pu* (žr. R1) ir siekiama šio tikslas *pu*, tai siekiama *pr*. Antra, kadangi *pr* skatina atgrasymą *de* (žr. R4) ir siekiama šio tikslas *de*, tai siekiama *pr*. Trečia, kadangi *pr* skatina apsaugą *pt* (žr. R5) ir siekiama tikslas *pt*, tai siekiama *pr*. Suteikime šiemis argumentams žymes *l<sub>1</sub>*, *l<sub>2</sub>* ir *l<sub>3</sub>*:

$$\frac{pr \Rightarrow pu, D\ pu}{D\ pr} \quad (l_1)$$

$$\frac{pr \Rightarrow de, D\ de}{D\ pr} \quad (l_2)$$

$$\frac{pr \Rightarrow pt, D\ pt}{D\ pr} \quad (l_3)$$

Šie trys argumentai *l<sub>1</sub>*, *l<sub>2</sub>* ir *l<sub>3</sub>* apjungiami į vieną argumentą *pr<sup>+</sup>*, vadinamą *apibendrinimu* (*accrual*); žr. 23.6 pav. kaireje.



23.6 pav. Argumentas skirti įkalnimą *pr<sup>+</sup>* ><sup>1</sup> nurungia argumentą įkalinimo neskirti *pr<sup>-</sup>*

Gynėjas gali pateikti tik vieną argumentą *pr<sup>-</sup>* neskirti įkalinimo. Kadangi *pr* žemina perauklėjimą *re* (žr. R1) ir siekiama tikslas *re*, tai siekiama neskirti *pr*. Pastarojo argumento žymė *l<sub>3</sub>* ir jis imamas kaip *pr<sup>-</sup>* (23.6 pav. dešinėje).

$$\frac{pr \Rightarrow \neg re, D\ re}{\neg D\ pr} \quad (l_4)$$

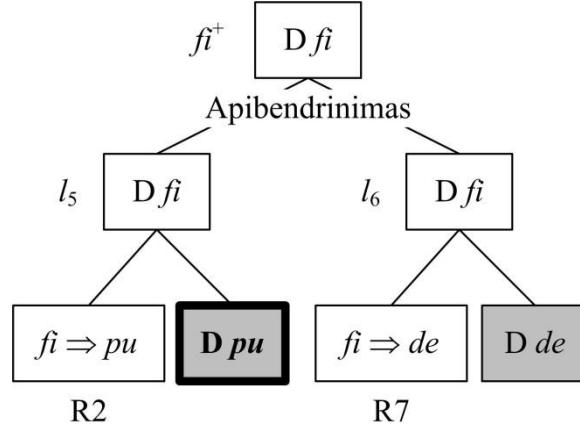
Argumentas už įkalnimą nurungia argumentą už įkalinimo neskyrimą, *pr<sup>+</sup>* > *pr<sup>-</sup>*, nes neskiriant *pr* yra nepasiekiamas pagrindinis tikslas *pu* (kuris 23.6 pav. išryškintas). Neigiami tikslai vaizduojami punktyriniuose stačiakampiuose.

### 23.5. Argumentavimas už baudą *fi<sup>+</sup>*

Kaltintojas gali pateikti du argumentus skirti baudą *fi*. Pirma, kadangi *fi* skatina nubaudimą *pu* (žr. R2) ir siekiama *pu*, tai siekiama *fi*. Antra, kadangi *fi* skatina atgrasymą *de* (žr. R7) ir siekiama *de*, tai siekiama *fi*. Suteikime šiemis argumentams žymes *l<sub>5</sub>* ir *l<sub>6</sub>*:

$$\frac{f\bar{i} \Rightarrow pu, D\ pu}{D\ f\bar{i}} \quad (l_5) \qquad \frac{f\bar{i} \Rightarrow de, D\ de}{D\ f\bar{i}} \quad (l_6)$$

Šie du argumentai  $l_5$  ir  $l_6$  apjungiami į vieną apibendrintą argumentą  $f\bar{i}^+$  (23.7 pav.).



23.7 pav. Apibendrintas argumentas  $f\bar{i}^+$  skirti baudą

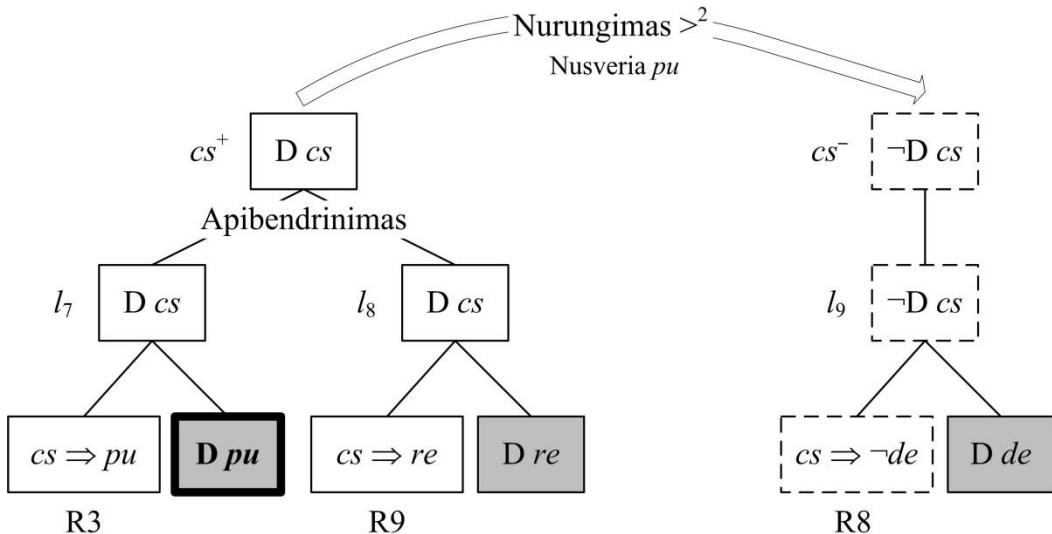
Gynėjas argumento  $f\bar{i}^-$  neturi, nes  $f\bar{i}$  neturi neigiamo poveikio tikslams. Todėl nėra argumento, kurį galėtų  $f\bar{i}^+$  nurungti.

### 23.6. Argumentavimas už viešuosius darbus $cs^+ >^2 cs^-$

Kaltintojas gali pateikti tris argumentus skirti viešuosius darbus  $cs$ . Pirma, kadangi  $cs$  skatina nubaudimą  $pu$  (žr. R3) ir siekiama  $pu$ , tai siekiama  $cs$ . Antra, kadangi  $cs$  skatina perauklėjimą  $re$  (žr. R9) ir siekiama  $re$ , tai siekiama  $cs$ . Suteikime šiems argumentams žymes  $l_7$  ir  $l_8$ :

$$\frac{cs \Rightarrow pu, D\ pu}{D\ cs} \quad (l_5) \qquad \frac{cs \Rightarrow re, D\ re}{D\ cs} \quad (l_6)$$

Šie du argumentai  $l_5$  ir  $l_6$  gali būti apjungti į vieną apibendrintą argumentą  $cs^+$ ; žr. 23.6 pav. kairėje.



23.8 pav. Argumentas skirti viešuosius darbus  $cs^+$  nurungia argumentą jų neskirti  $cs^-$

Gynėjas gali pateikti vieną argumentą  $cs^-$  neskirti viešujų darbų. Kadangi  $cs$  žemina atgrasymą  $de$  (žr. R8) ir siekiama tikslą  $de$ , tai siekiama neskirti  $cs$ . Pastarojo argumento žymė  $l_9$  ir jis imamas kaip  $cs^-$  (23.8 pav. dešinėje).

$$\frac{cs \Rightarrow \neg de, D de}{\neg D cs} \quad (l_9)$$

Argumentas už viešuosius darbus nurungia argumentą už jų neskyrimą,  $cs^+ > cs^-$ , nes neskiriant  $cs$  yra nepasiekiamas pagrindinis tikslas  $pu$  (kuris 23.8 pav. išryškintas).

### 23.7. Antrasis žingsnis: viešieji darbai nurungia įkalnimą, $cs^+ >^3 pr^+$

Trijuose prieš tai einančiuose poskyriuose buvo pateiktas pirmasis argumentavimo žingsnis, susidedantis iš dviejų nurungimų  $pr^+ >^1 pr^-$  ir  $cs^+ >^2 cs^-$ . Šiame poskyryje argumentuoja, kad viešieji darbai nurungia įkalnimą,  $cs^+ >^3 pr^+$  (23.3 b pav. ir 23.10 pav.).

Primename veiksmų  $pr$  ir  $f_i$  įvertinimus, kurie buvo 23.4 lentelėje:

Ivertinimas	(skatina, žemina)	Santykis
$v(pr^+) =$	$(\{pu, de, pt\}, \{re\})$	3:1
$v(cs^+) =$	$(\{pu, re\}, \{de\})$	2:1

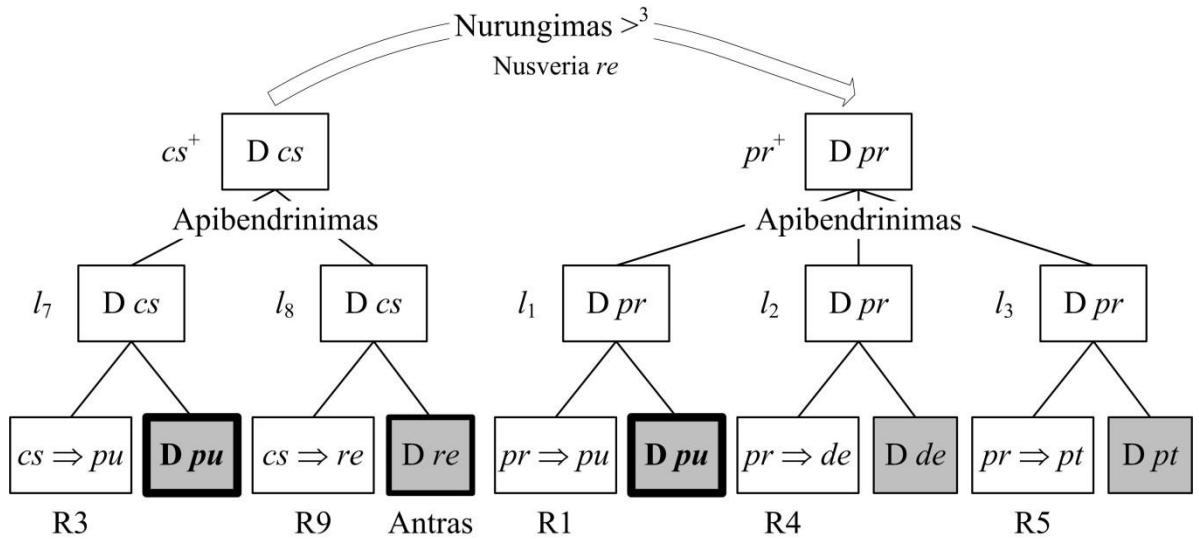
23.9 lentelė. Veiksmų  $pr$  ir  $cs$  įvertinimai

**Pirmasis sprendimas.** Toliau teisėjas nusprendžia, kad perauklėjimas  $re$  yra antras pagal svarbumą tikslas.

Tai ekstraloginis sprendimas. Jis išplaukia ne iš formaliosios logikos, o vertybinių vertinimo dalykinėje srityje. Teisėjas galėtų nuspresti ir kitaip, pavyzdžiui, baudos  $f_i^+$  naudai.

Iš šio sprendimo išplaukia, kad  $re$  (kurią skatina  $cs$ ) nusveria, kai rungiasi argumentas  $cs^+$  prieš  $pr^+$  (23.10 pav.). Galima pasakyti, kad  $re$  nusveria atgrasymą  $de$  ir apsaugą  $pt$  (kuriuos skatina  $pr$ ) kartu sudėjus, t.y.  $re >^3 de + pt$  (žr. pabrukimus 23.9 lentelėje). Subtiliau būtų

pasakyti, kad *re* be *de* (kurie yra  $cs^+$ ) nusveria *de* ir *pt* kartu ir be *re* (kurie yra  $pr^+$ ), t.y.  $re-de >^3 de+pt-re$ . Taigi reziumuojame, kad  $cs^+$  nurungia  $pr^+$ .



23.10 pav. Argumentas skirti viešuosius darbus nurungia argumentą skirti įkalinimą:  $cs^+ >^3 pr^+$

### 23.8. Trečiasis žingsnis: viešieji darbai nurungia baudą, $cs^+ >^4 fi^+$

Argumentuosime, kad viešieji darbai nurungia baudą,  $cs^+ >^4 fi^+$  (23.3 b pav. ir 23.12 pav.). Primename veiksmų *pr* ir *fi* įvertinimus, kurie buvo 23.4 lentelėje:

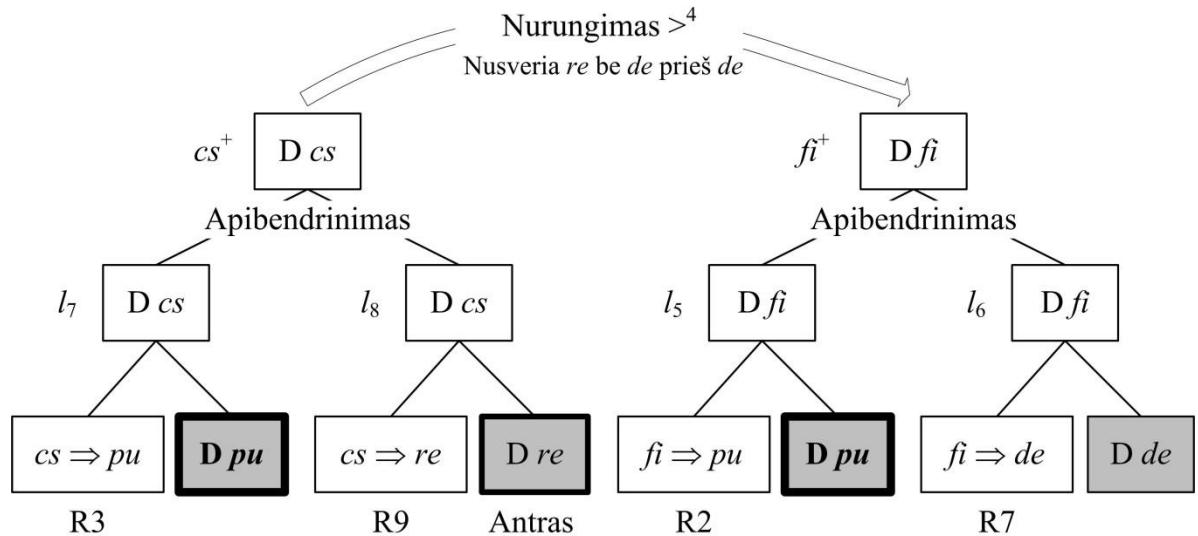
Ivertinimas	(skatina, žemina)	Santykis
$v(fi^+) = (\{pu, \underline{de}\}, \emptyset)$	2:0	
$v(cs^+) = (\{pu, \underline{re}\}, \{\underline{de}\})$	2:1	

23.11 lentelė. Veiksmų *pr* ir *fi* įvertinimai

**Antrasis sprendimas.** Toliau teisėjas nusprendžia, kad perauklėjimas *re* nors ir be atgrasymo *de* nusveria atgrasymą *de*. Čia vadovaujamasi, kad *re* yra antras pagal svarbumą tikslas.

Tai ekstraloginis sprendimas. Jis išplaukia ne iš formaliosios logikos, o iš vertybiniio vertinimo.

Iš šio sprendimo išplaukia, kad *re* (kurią skatina *cs*) nusveria, kai rungiasi argumentas *fi*<sup>+</sup> prieš *fi*<sup>+</sup> (23.12 pav.). Galima pasakyti, kad *re* nors ir be *de* nusveria *de* (kurią skatina *fi*), t.y.  $re-de >^4 de$  (žr. pabraukimus 23.11 lentelėje). Taigi reziumuojame, kad  $cs^+$  nurungia *fi*<sup>+</sup>. Tokiu būdu nugali argumentas  $cs^+$  skirti viešuosius darbus, ką ir reikėjo pagrįsti.



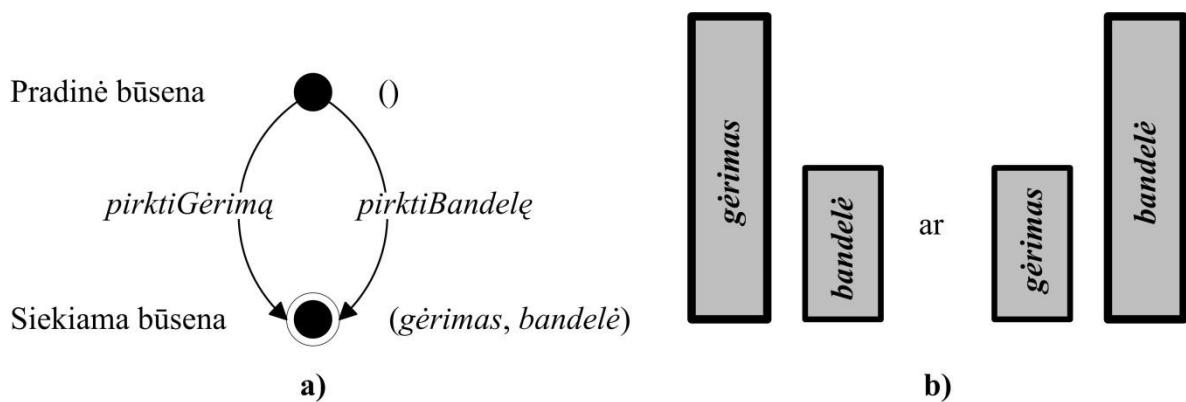
23.12 pav. Argumentas skirti viešuosius darbus nurungia argumentą skirti baudą:  $cs^+ >^4 fi^+$

### 23.9. Gérimas ar bandelė?

Neįmanomumą pasiekti du tikslus su ribotais resursais iliustruoja tokis pavyzdys. Tegu yra turima viena moneta. Ir gérimo buteliukas, ir bandelė kainuoja po vieną monetą. Tikslas – nusipirkti ir gérimą, ir bandelę.

Aišku, kad pasiekti tikslų būseną *gérimas & bandelė* su viena moneta yra neįmanoma (žr. 23.13 a pav.). Du atskirius veiksmus *pirktiGérimq* ir *pirktiBandelę* mes modeliuojame dviem produkcinėmis taisyklėmis:

- R1: *moneta* → *gérimas & ¬moneta*  
 R2: *moneta* → *bandelė & ¬moneta*



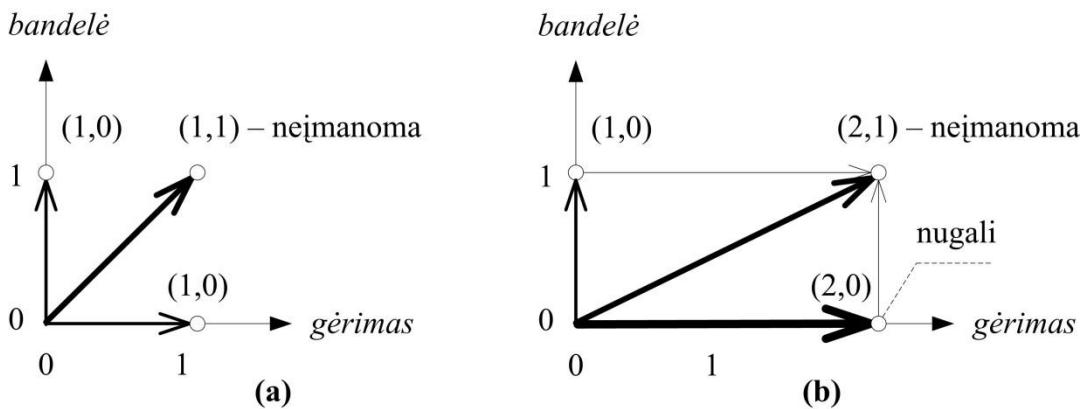
23.13 pav. a) Negali būti pasiekti du tikslai – ir *gérimas*, ir *bandelė*. b) Tikslams gali būti suteikti skirtini svoriai

Taisyklių R1 ir R2 sintaksė lyginant su taisyklėmis tiesioginio ir atbulinio išvedimo algoritmuose yra išplėsta. Konsekventas yra keleto termų konjunkcija, kur termas gali būti ir su neigimo ženklu. Dėl toks taisyklių kalbos praplėtimo keletas tikslų pasiekimo problema tampa sudėtingesnė.

Iš pradinio fakto  $\{moneta\}$  yra neįmanoma pasiekti būseną kurioje pasiekiami abu tikslai, t.y.  $gérimas \& bandelé$ , kitaip sakant, faktų aibė  $\{gérimas, bandelé\}$ , t.y. būsena ( $gérimas = 1$ ,  $bandelé = 1$ ). Pirmuoju veiksmu –  $pirktiGérimq$  – pasiekama būsena  $gérimas \& \neg moneta$ , kitaip sakant, faktų aibė  $\{gérimas\}$ . Antruoju veiksmu  $\neg pirktiBandelq$  – pasiekama būsena  $\{bandelé\}$ .

Ką rinktis pirkėjui? Pasirinkimą galima paaiškinti ir geometriškai kaip Manheteno ar Euklido atstumą nuo nepasiekiamos dviejų tikslų būsenos iki pasiekiamos vieno tiksllo būsenos.

Jeigu tiek gérimas, tiek bandelė turi lygius svorius – po 1, tai argumentai  $pirktiGérimq$  ir  $pirktiBandelq$  nenusveria vienas kito. Pasirinkus  $pirkGérimq$  yra pasiekama būsena ( $gérimas = 1$ ,  $bandelé = 0$ ), kuri vaizduojama vektoriumi  $(1,0)$ ; žr. 23.14 a pav. Pasirinkus  $pirkBandelq$  yra pasiekama būsena ( $gérimas = 0$ ,  $bandelé = 1$ ), kuri vaizduojama vektoriumi  $(0,1)$ . Dviejų tikslų būsena ( $gérimas = 1$ ,  $bandelé = 1$ ) vaizduojama vektoriumi  $(1,1)$ . Pasirinkimu  $pirkGérimq$  atstumas (Manheteno) nuo tiksllo būsenos  $(1,1)$  iki gérimo  $(1,0)$  yra  $\|(1,1) - (1,0)\| = \|(0,1)\| = 1$ . Pasirinkimu  $pirkBandelq$  atstumas nuo tiksllo būsenos  $(1,1)$  iki bandelės  $(0,1)$  yra  $\|(1,1) - (0,1)\| = \|(1,0)\| = 1$  ir šie atstumai lygūs. Taigi pasirinkimai nenusveria vienas kito.



23.14 pav. a)  $gérimas$  ir  $bandelé$  lygiaverčiai. b)  $gérimas$  nusveria  $bandelé$

Tačiau ištroškės pirkėjas rinksis gérimą. Tai modeliuojama tikslų sutvarkymu  $gérimas > bandelé$ . Pavyzdžiui, gérimui suteikiamas svoris 2, o bandelei 1. Tokiu atveju argumentas  $pirktiGérimq$  nusveria argumentą  $pirktiBandelq$ .

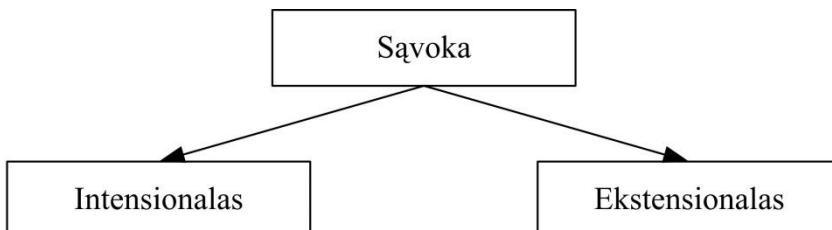
Išalkęs pirkėjas rinksis bandelę. Tai modeliuojama tikslų sutvarkymu  $gérimas < bandelé$ . Pavyzdžiui, gérimui suteikiamas svoris 1, o bandelei 2. Tada argumentas  $pirktiGérimq$  nusileidžia argumentui  $pirktiBandelq$ .

Ištroskusio pirkėjo pasirinkimą galima paaiškinti ir geometriškai. Pasirinkimu  $pirkGérimq$  atstumas nuo tiksllo būsenos  $(2,1)$  iki gérimo  $(2,0)$  yra  $\|(2,1) - (2,0)\| = \|(0,1)\| = 1$ . Pasirinkimu  $pirkBandelq$  atstumas nuo tiksllo būsenos  $(2,1)$  iki bandelės  $(0,1)$  yra  $\|(2,1) - (0,1)\| = \|(2,0)\| = 2$ . Didesnis atstumas iki tiksllo yra blogesnis. Todėl nugali argumentas  $pirkGérimq$ .

## 24. Intensionalas, ekstensionalas ir ontologija

Šiame skyriuje yra apibrėžama kas yra ekstensionalinė reliacinė struktūra, pasaulis, intensionalinis santykis, intensionalinė reliacinė struktūra, ekstensionalinė pirmos eilės struktūra (kalbos modelis), intensionalinė pirmos eilės struktūra (ontologinis įspareigojimas), numatomi modeliai ir ontologija.

Sąvoka gali būti aprašoma tiek savo *intensionalu*, tiek *intensionalu*. Sąvoka vaizduojama ženklu (*sign*). Intensionalas, ekstensionalas ir ženklas sudaro sąvokos triadą (24.1 pav.).



- Intensionalinė apibrėžtis (savybių aibė)
- Tipas
- Tipo lygmuo
- Language
- Ekstensionalinė apibrėžtis (egzempliorius arba egzempliorių aibė)
- Klasė
- Egzemplorių lygmuo
- Parlance

**24.1 pav.** Sąvokos triada

**Intensionalo pavyzdys.** Žr. *intensional definition* [http://en.wikipedia.org/wiki/Intensional\\_definition](http://en.wikipedia.org/wiki/Intensional_definition): „For example, an intensional definition of bachelor is 'unmarried man'. Being an unmarried man is an essential property of something referred to as a bachelor. It is a necessary condition: one cannot be a bachelor without being an unmarried man. It is also a sufficient condition: any unmarried man is a bachelor (Cook 2009).

This is the opposite approach to the extensional definition, which defines by listing everything that falls under that definition — an extensional definition of bachelor would be a listing of all the unmarried men in the world (Cook 2009). ... Similarly, an intensional definition of a game, such as chess, would be the rules of the game; any game played by those rules must be a game of chess, and any game properly called a game of chess must have been played by those rules.“

**Intensionalių savybių identiškumo pavyzdys.** Intensionalinė savybė  $P_1$  vadinama *identiška* intensionalinei savybei  $P_2$  jeigu 1) kiekvienas objektas tenkinantis  $P_1$  tenkina ir  $P_2$  ir 2) kiekvienas objektas netenkinantis  $P_1$  netenkina ir  $P_2$ .

Tačiau su ekstensionalinėmis savybėmis nėra taip paprasta. Pavyzdžiui, savybė 'būti\_lygiakraščiu\_trikampiu' pripažystama identiška savybei 'būti\_lygiakampiu\_trikampiu' priklausomai nuo mūsų žinių apie geometriją. Žiūrint į lygiakraščio trikampio egzempliorių (t.y. ekstensionalinėje aibėje) galima pastebėti faktą, kad jis yra lygiakampus ir atvirkščiai. Tačiau stebėjimo fakto nepakanka išvadai, kad dvi ekstensionalinės aibės – lygiakraščių trikampių ir lygiakampių – sutampa. Tik iš geometrijos žinių, tiksliau, teoremos, kad trikampis yra lygiakraštis tada ir tik tada, kai jis yra lygiakampus, daroma išvada, kad lygiakraščių trikampių ir lygiakampių trikampių aibės sutampa. O tai yra intensionalių savybių  $P_1$  ir  $P_2$  identiškumo įrodymas. Todėl kalbant apie sąvokas yra nagrinėjamos intensionalinės savybės.

**Kitas pavyzdys.** Stebint faktą, kad skaičių aibėje {1, 3, 5, 7, 11, 13, 17, 19, 23, 29} visi elementai yra pirminiai, dar negalima daryti išvados, kad jeigu skaičius nelyginis, tai jis pirminis. Pvz., nelyginiai {9, 15, 21, 25, 27 ...} nėra pirminiai.

Rezume: iš ekstensionalinės savybės stebėjimo fakto neišplaukia intensionalinė savybė. Kitais žodžiais, iš fakto neišplaukia taisyklė. Iš esamybės neišplaukia būtinybė.

## 24.1. Ženklai

Ženklui (*sign*) sąvokai žymėti paprastai yra žodis ar keletas žodžių, pvz., 'povandeninis laivas'). Taigi paprastai ženklai yra tekstiniai, sudaryti iš raidžių. I ženklą gali įeiti skaičiai ir kitokios literos, jeigu komunikuoojantys agentai (žmonės ar kompiuteriai) taip susitaria. Pavyzdžiu, 'Audi A4' žymi automobilių gamintoją ir markę, o 'ABC 123' konkretaus automobilio numerį.

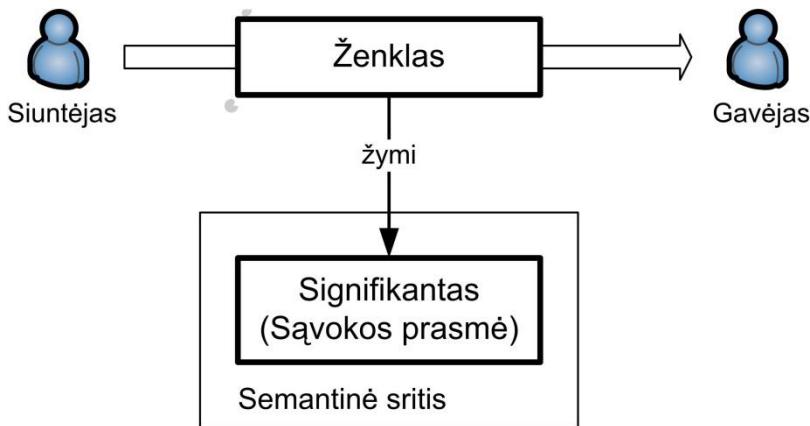
Ženklai gali būti ir grafiniai (24.2 pav.), pvz., kelio ženklai. Ženklų prasmę (semantiką) aprašyti žodžiais gali būti neelementaru, nes apima pareigas bei teises.



24.2 pav. Grafinių ženklų pavyzdžiai

Ženklai gali būti išreiškiami gestais (pvz., policininko gestai), garsu (pvz., automobilio ar laivo signalai), šviesomis (pvz., šviesoforo signalai), konkliudentiniai veiksmai (pvz., pasisveikinimas, įspėjamasis šūvis, pastūmimas). Tokiu būdu žeklus žmogus gali suvokti skirtingais pojūčiais, pvz., rega, klausa, prisilietimu, fiziniu poveikiu.

Semiotikoje ženklas (*signifier, signifikatorius*) rodo į signifikantą (*significant*), t.y. ženklo prasmę (*meaning*) (24.2 pav.). Semiotika yra mokslas apie ženklus ir jų aiškinimą (interpretavimą).



**24.3 pav.** Ženklas rodo į sąvokos prasmę (kitaip tariant, idėją, mintį)

Gali būti neįvardintų sąvokų, t.y. sąvokų, kurioms vardas (t.y. ženklas) nėra suteiktas. Pvz., „Braudamasis tamsoje rūke pro mišką aš priartėjau prie šviesaus objekto, kuris buvo aukštai virš šakos, subraižiusios man veidą. Kas tai galėjo būti? Ménulio apšviesta šiukslė, kurią niekadėjas užmovė aukštai ant šakos, vėjo atnešto baliono liekana ar dar kas nors?“

**Terminija.** Pagrindiniai koncepcinių modelių elementai yra *sąvokos* ir *ryšiai*. Yra autorių, kurie koncepciniame modeliavime vartoja kitus terminus: sąvoka – konceptas, intensionalas – intencija, ekstensionalas – ekstensija, ženklas – simbolis, santykis (ekstensionalo prasme) (*relation*) – sąryšys arba ryšis; intensionalinis santykis (*relationship*) – ryšys arba sąryšis. Kadangi kalbant apie apie santykį galima turėti omenyje tiek ekstensionalą, tiek intensionalą, tai reikalingi du žodžiai. O pasirinkti galima iš trijų žodžių: santykis, ryšys ir sąryšis. Koncepcinio modeliavimo vadovelyje (Paradauskas, Nemuraitė 2008) dera terminai konceptas, intencija, ekstencija ir ryšys, nes iš konteksto aišku, ar apie ryšio intensionalą ar ekstensionalą yra kalbama.

**Sąvokos: bendrinės ir individualios.** Ženklas gali žymeti bendrinę sąvoką, pvz., „imperatorius“ arba individualią, pvz., „imperatorius Napoleonas“.

**Filosofinė problema: kas pirmesnis – sąvoka (idėja, sąmonė) ar materija.** Projektuojant informacines sistemas vartojamos intensionalinės sąvokų apibrėžtys, pvz., „prekės užsakymas“, „sutartis“. Jas apibendrinantis tipas yra „ivykis“ (event). Iwykis yra perejimas iš sistemos vienos būsenos į kitą.

Sąvoka yra abstrakcija, idėja. Pavyzdžiai yra matematinės sąvokos. Matematikoje sąvoka „apskritimas“ apibrėžiama kaip geometrinė vieta taškų plokštumoje, vienodai nutolusių nuo taško, vadinamo centru. Sąvokos gyvuoja žmonių sąmonėje; be sąmonės sąvokų nėra.

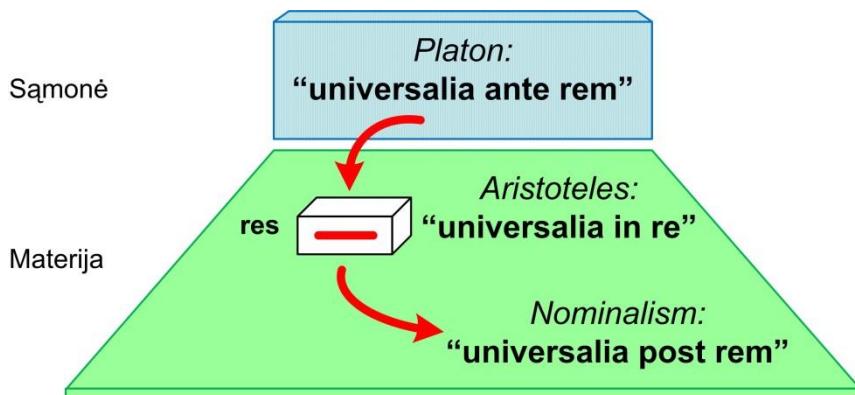
Filosofijoje yra iškeliamas klausimas: kas pirmesnis – sąvoka (t.y. idėja) ar materialus daiktas, kurį ši sąvoka žymi? Šis klausimas yra įvardijamas kaip „sąmonės ir materijos problema: kas pirmesnis?“. Klausimą galima perforfuluoti: ar idėjos egzistuoja objektyviai?

Minėta problema dar gali būti įvardija kaip *universalijų problema*: ar universalijos (kategorijos) egzistuoja objektyviai, ir, jeigu egzistuoja, tai kas jos yra ([http://en.wikipedia.org/wiki/Problem\\_of\\_universals](http://en.wikipedia.org/wiki/Problem_of_universals)). Universalija yra savybė, kuri yra bendra keletui esybių, pavyzdžiui, „būti raudonos spalvos“ (*redness*), „būti puoduku“ (*cupness*), „būti žmogumi“ (*homo sapiens*, pagal mokslinę klasifikaciją gyvūnų karalystės, chordinių tipo, žinduolių klasės, primatų būrio, hominidų šeimos, genties *homo* egzemplioriumi, <http://lt.wikipedia.org/wiki/%C5%BDmogus>). Manoma, kad esminis požymis, skiriantis žmogų nuo kitų gyvūnų, yra abstraktus mąstymas. O tai salygoja kalbą kaip ženklu sistemą sąvokoms žymeti.

Be profesinio filosofijos išsilavinimo sunku perprasti minėtos problemos esmę. Ją trumpai išdėstyti nėra paprasta. Pavyzdžiu, ar egzistuoja apskritimas (kaip sąvoka, kaip idėja) objektyviai? Oponentas gali atsakyti, kad neegzistuoja, nes kiekvienas materializuotas apskritimas (pvz., lėkštutė, skriestuvu ant poperiaus nubrėžtas apskritimas, apskritimas kompiuterio ekrane) yra tik priartėjimas ir netiksliai vaizduoja apskritimo idėją. Padidinus brėžinį galima įžiūrėti nelygumus. Tarp medžagos atomų yra tarpai.

Grubiai galima išskirti tris požiūrius (24.4 pav.):

- „universalija yra anksčiau už daiktą“, lotyniškai *universalia ante rem*. Taip teigė Platonas. Jis teigė, kad universalios yra objektyvios ir egzistuoja nepriklausomai nuo atskirybių (particulars). Taigi čia postuluojama, kad idėja yra pirmesnė už materiją. Tai idealizmo pozicija: idėja yra prieš materiją. Idealistai tokie kaip Kantas ir Hegelis, postulavo, kad universalios yra realios, bet egzistuoja tik žmonių sąmonėje. Pavyzdžiu, spredžiant ar du puodukai pasižymi apvalumo savybe vadovaujamasi ne nuo sąmonės nepriklausoma savybe „apvalumas“, bet sąmonėje glūdinčiu supratimu, kas yra apvalumas.
- „universalija yra daikte“ (t.y. abu egzistuoja vienu metu), *universalia in re*. Taip teigė Aristotelis. Jis teigė, kad universalios yra realios esybės ir egzistuoja kartu su atskirybe, daiktu, kuris pasižymi šia savybe. Tai realizmo, tiksliau Aristoteliškojo realizmo pozicija.
- „universalija yra po daikto“ *universalia post rem*. Tai nominalizmo pozicija (nuo lotyniško *nomen* (vardas)).



**24.4 pav.** Skirtingos pozicijos filosofijoje. Ar savybės egzistuoja objektyviai?  
Kas pirmesnis: idėja ar materija?

Vadovaujantis materialistine filosofija materija yra pirminė prieš sąmonę. Tačiau pastarąsias nuostatas reikia suprasti filosofiškai. Kaip kontrpavyzdį pateiksime buitišką pavyzdį, kai idėja gimsta sąmonėje anksčiau, negu ją materializuojantis daiktas. Architektu galvoje pirma gimsta būsimo pastato idėja, o tik vėliau jis tą idėją materializuojant brėžiniuose ir makete iš poperiaus arba kompiuteryje. Dar vėliau pastatą stato statybininkai.

**Subsumcija (subsumption) – egzempliorius tapatinimas su intencionalu.** Tai siauraja prasme. Gali būti kalbama ir apie vienos sąvokos tapatinimą kitai sąvokai. Pavyzdžiu, ar žurnalinis staliukas gali būti tapatinamas su staloo sąvoka? Ar taburetė šalia mano lovos gali būti tapatinama su stalu? Kiek taburetėje stališkumo savybės? Aš ją naudoju ir kaip taburetę (atsisesti) ir kaip stalą (pasideti žadintuvą). Tapatinimo rezultatas gali būti 'taip' ar 'ne', 1 ar 0. Šitaip vaizduojama dviejų reikšmių logikoje. Dar atsakymas gali būti 'labiau tapatu nei netapatu'. Taip yra diskrečioje keleto reikšmių skalėje arba tolydinėje skalėje, kur naudojami tokie modeliavimo būdai kaip *blanki aibė* (*fuzzy set*) ir *tikėtinumo funkcija*, kurios kitimo sritis

yra realių skaičių intervalas  $[0,1]$ . Reziumuojame: subsumcijos rezultatas priklauso nuo konteksto bei tapatinančiojo agento ir yra subjektyvus.

## 24.2. Kas yra konceptualizacija?

Toliau šiame skyriuje vadovaujamasi (Guarino et al. 2009) straipsniu. Aiškinsime apibrėžtį, kad ontologija yra formalū, išreikštinė sutartos konceptualizacijos specifikacija (*an ontology is a formal, explicit specification of a shared conceptualization*). Pradėsime nuo savybos konceptualizacijos aiškinimo.

**24.1 apibrėžtis. Ekstensionalinė reliacinė struktūra** yra pora  $S = (D, \mathbf{R})$ , kur

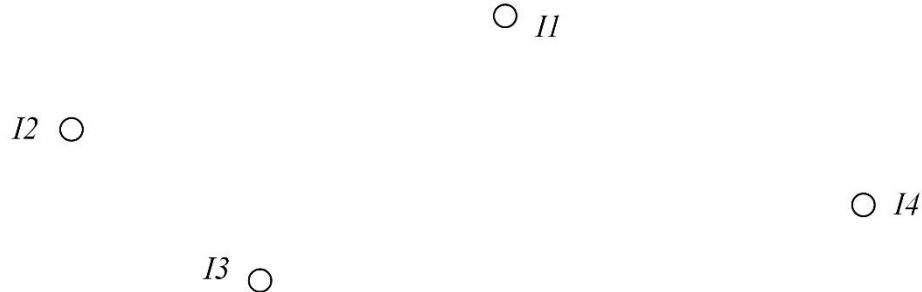
- $D$  yra aibė, vadinama diskurso universumu (*universe of discourse*)
- $\mathbf{R}$  yra santykių virš  $D$  aibė. (Guarino et al. 2009, 2.1 apibrėžtis)

□

Kiekvienas  $R$  elementas yra ekstensionalinis santykis, t.y. matematinis santykis, dekartinės sandaugos poaibis.

**24.2 pavyzdys.** Kalbėsime apie įmonės personalo informacinię sistemą. (Let us consider human resources management in a large software company with 50.000 people, each one identified by a number (e.g., the social security number, or a similar code) preceded by letter  $I$ . Let us assume that our universe of discourse  $D$  contains all these people, and that we are only interested in relations involving people. Our  $\mathbf{R}$  will contain some unary relations, such as *Person*, *Manager*, and *Researcher* as well as the binary relations *reports-to* and *cooperates-with*.<sup>3</sup>) (Guarino et al. 2009, 2.1 pavyzdys).

Tegu yra 4 žmonės, t.y. aibė  $D$  turi 4 elementus:  $D = \{I1, I2, I3, I4\}$  (24.5 pav).



**24.5 pav.** Diskurso universumas yra aibė  $D = \{I1, I2, I3, I4\}$ . Elementai žymi 4 žmones. Kalbama apie įmonės personalo informacinię sistemą

Yra 5 santykiai (5 lentelės):

$$\begin{aligned} \mathbf{R} = & \{ \{ (I1), (I2), (I3), (I4) \}, \\ & \{ (I1) \} \\ & \{ (I2), (I3) \} \\ & \{ (I2, I1), (I3, I1) \} \\ & \{ (I2, I3) \} \} \end{aligned}$$

Įvardinkime kiekvieną šiu 5 santykiumi:

$\mathbf{R} = \{ \text{Person}, \text{Manager}, \text{Researcher}, \text{reports-to}, \text{cooperates-with} \}$ , kur

<sup>3</sup> Vardas ir pavardė gali būti priskiriami santykiais, pvz., *firstname(I4, 'Daniel')* and *lastname(I4, 'Oberle')*.

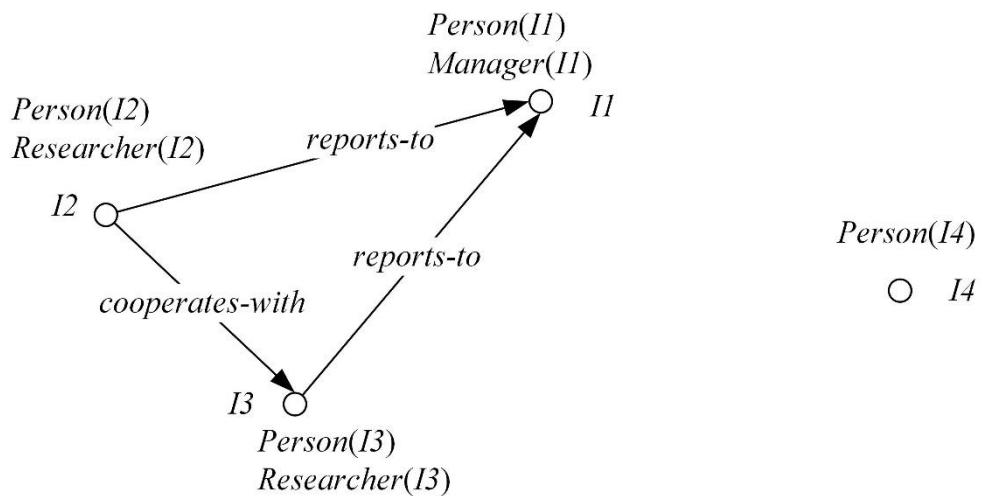
<i>Person</i>	= { (II), (I2), (I3), (I4) }	- vienvietis santykis
<i>Manager</i>	= { (II) }	- vienvietis santykis
<i>Researcher</i>	= { (I2), (I3) }	- vienvietis santykis
<i>reports-to</i>	= { (I2, II), (I3, II) }	- dvivietis santykis
<i>cooperates-with</i>	= { (I2, I3) }	- dvivietis santykis

Šie 5 santykiai gali būti pavaizduoti grafiškai 5 lentelėmis (24.6 pav.):

$\mathbf{R} = \{$	<table border="1" style="display: inline-table; vertical-align: middle;"> <tr><td>I1</td></tr> <tr><td>I2</td></tr> <tr><td>I3</td></tr> <tr><td>I4</td></tr> </table>	I1	I2	I3	I4	<table border="1" style="display: inline-table; vertical-align: middle;"> <tr><td colspan="2">II</td></tr> </table>	II		<table border="1" style="display: inline-table; vertical-align: middle;"> <tr><td colspan="2">I2</td></tr> <tr><td colspan="2">I3</td></tr> </table>	I2		I3		<table border="1" style="display: inline-table; vertical-align: middle;"> <tr><td colspan="2">I2</td></tr> <tr><td>I3</td><td>I1</td></tr> </table>	I2		I3	I1	<table border="1" style="display: inline-table; vertical-align: middle;"> <tr><td colspan="2">I2</td></tr> <tr><td>I3</td><td>I3</td></tr> </table>	I2		I3	I3	$\}$
I1																								
I2																								
I3																								
I4																								
II																								
I2																								
I3																								
I2																								
I3	I1																							
I2																								
I3	I3																							
	<i>Person</i>	<i>Manager</i>	<i>Researcher</i>	<i>reports-to</i>	<i>cooperates-with</i>																			

**24.6 pav.** 5 santykiai pavaizduoti 5 lentelėmis

Ekstensionalinė reliacinė struktūra iš 24.2 pavyzdžio pavaizduota grafiškai (24.7 pav.):

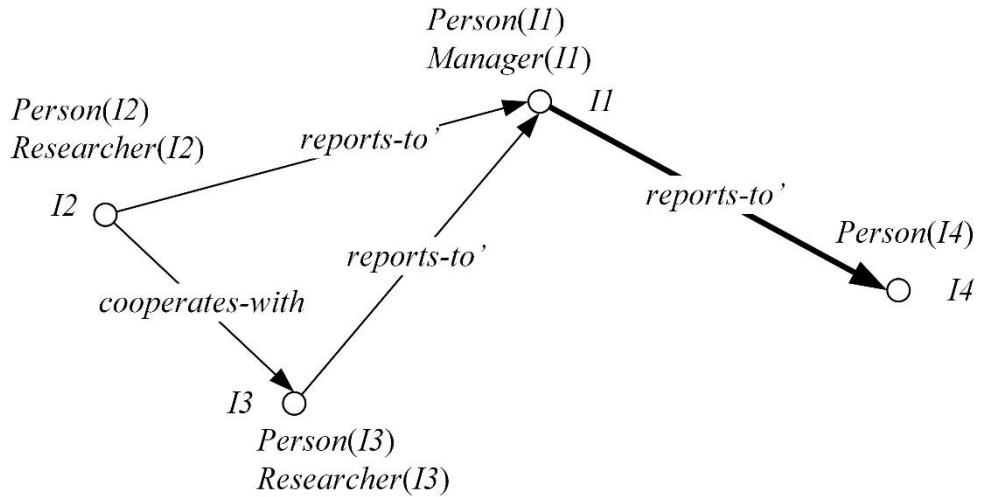


**24.7 pav.** Ekstensionalinės reliacinės struktūros iš 24.2 pavyzdžio grafinis vaizdavimas. Atitinka tam tikrą pasaulio ir informacinių sistemų būseną

**24.3 pavyzdys** (žr. Guarino et al. 2009, 2.2 pavyzdj). Papildykime ekstensionalinę reliacinę struktūrą iš 24.2 pavyzdžio vienu elementu santykyje *reports-to*:

- $D' = D$
- $\mathbf{R}' = \{ \text{Person}, \text{Manager}, \text{Researcher}, \text{reports-to}', \text{cooperates-with} \}$ , kur  $\text{reports-to}' = \text{reports-to} \cup \{(II, I4)\}$

Tokiu būdu gauname kitą ekstensionalinę rečiacinę struktūrą  $(D', \mathbf{R}') \neq (D, \mathbf{R})$  nors  $\text{reports-to}'$  lentelė papildyta tik viena eilute, t.y. tik viena briauna grafe 24.7 pav.

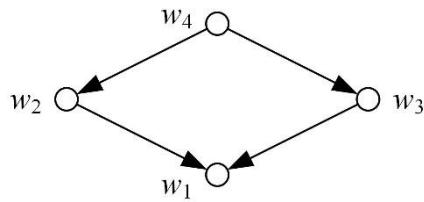


**24.8 pav.** Ekstensionalinė rečiacinė struktūra  $(D', \mathbf{R}')$  iš 24.3 pavyzdžio. Santykis  $\text{reports-to}'$  papildytas vienu elementu ( $II, I4$ ), t.y. briauna grafe. Tai atitinka kitą pasaulio ir informacinės sistemos būseną

**24.4 apibrėžtis.** Pasaulis yra sutvarkyta aibė  $W = \{w_1, w_2, w_3 \dots\}$ . □

Pasaulio  $W$  elementai vadinami pasaulio *būsenomis*. Pasaulis yra *pilnai sutvarkyta* aibė. Tarp jos elementų yra *pilno sutvarkymo* (*total ordering*) santykis. Modeliuojama, kad laikas yra sutvarkytas. Pilnas sutvarkymas reiškia, kad bet kurioms skirtingoms būsenoms  $w_i$  ir  $w_j$  yra teisinga  $w_i < w_j$  arba  $w_i > w_j$ . Šitaip pasaulio sąvokos pagalba yra modeliuojamos informacinės sistemos būsenos kintant laikui.

*Dalinai sutvarkyta* aibės (*partially ordered set*) pavyzdys parodytas (24.9 pav.). Tai aibė  $W = \{w_1, w_2, w_3, w_4\}$ . Joje apibrėžtas toks sutvarkymo santykis:  $\{w_1 < w_2, w_1 < w_3, w_2 < w_4, w_3 < w_4\}$ . Kaip matyti, tarp elementų  $w_2$  ir  $w_3$  santykio nėra, t.y. nėra nei  $w_2 < w_3$ , nei  $w_3 < w_2$ . Tokie elementai vadinami *nepalyginamais*.



**24.9 pav.** Dalinai sutvarkyta aibė  $W = \{w_1, w_2, w_3, w_4\}$

**24.5 apibrėžtis. Intensionalinis santykis** (dar vadinama konceptinis santykis ar ryšys)  $\rho^{(n)}$  virš  $(D, W)$  yra atvaizdavimas  $W \rightarrow \text{powerset}(D^n)$ . Kitais žodžiais,  $\rho^{(n)}: w_i \rightarrow \rho^{(n)}(w_i)$ , t.y.  $\rho^{(n)}$  atvaizduoja pasaulio būseną  $w_i$  į dekartinės sandaugos  $D^n$  poaibį  $\rho^{(n)}(w_i)$  – į kortežų iš  $D^n$  aibę, kitaip sakant, į ekstensionalinę reliacinę struktūrą. (Guarino et al. 2009, 2.3 apibrėžtis) □

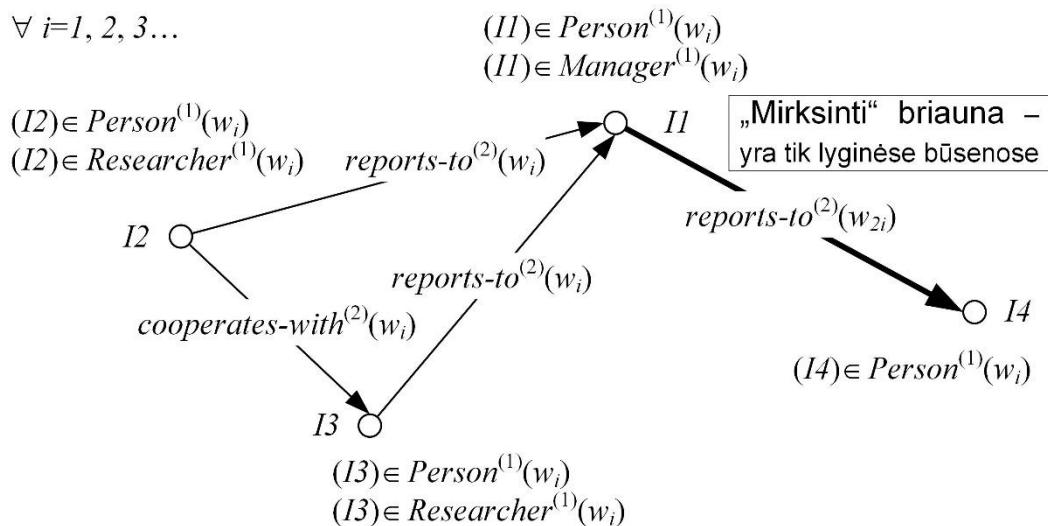
Čia  $\text{powerset}(A)$  žymi aibės  $A$  poaibį aibę. Dar žymima  $2^A$ . Pavyzdžiu, tegu  $A = \{a, b\}$ . Tada  $\text{powerset}(A) = 2^A = \{\{\}, \{a\}, \{b\}, \{a, b\}\}$ .

**24.6 apibrėžtis. Intensionalinė reliacinė struktūra** (dar vadinama konceptualizacija) yra trejetas  $\mathbf{C} = (D, W, \mathcal{R})$ , kur

- $D$  – diskurso universumas
- $W$  – pasaulis
- $\mathcal{R}$  – aibė intensionalinių santykių virš ( $D, W$ ). (Guarino et al. 2009, 2.4 apibr.)  $\square$

**24.7 pavyzdys.** Tai intensionalinė reliacinė struktūra, su kuria yra suderinami 24.2 ir 24.3 pavyzdžiai. Kitais žodžiais, ekstensionalinės reliacinės struktūros 24.2 ir 24.3 pavyzdžiuose atitinka dvi būsenas tokios intensionalinės reliacinės struktūros **C** (24.10 pav) (Guarino et al. 2009, 2.3 pavyzdys):

- $D = \{I1, I2, I3, I4\}$
- $W = \{w_1, w_2, w_3 \dots\}$
- $\mathcal{R} = \{\text{Person}^{(1)}, \text{Manager}^{(1)}, \text{Researcher}^{(1)}, \text{reports-to}^{(2)}, \text{cooperates-with}^{(2)}\}$
- $\forall i=1,2,3\dots \quad \text{Person}^{(1)}(w_i) = \{(I1), (I2), (I3), (I4)\}$
- $\forall i=1,2,3\dots \quad \text{Manager}^{(1)}(w_i) = \{(I1)\}$
- $\forall i=1,2,3\dots \quad \text{Researcher}^{(1)}(w_i) = \{(I2), (I3)\}$
- $\forall i=1,2,3\dots \quad \text{reports-to}^{(2)}(w_{2i-1}) = \{(I2, I1), (I3, I1)\} - 2$  briaunos nelyginėse būsenose
- $\forall i=1,2,3\dots \quad \text{reports-to}^{(2)}(w_{2i}) = \{(I2, I1), (I3, I1), (I1, I4)\} - 3$  briaunos lyginėse būsenose
- $\forall i=1,2,3\dots \quad \text{cooperates-with}^{(2)}(w_i) = \{(I2, I3)\}$



**24.10 pav.** Intensionalinės reliacinės struktūros iš 24.7 pavyzdžio grafinis vaizdavimas. Intensionalinių santykiai  $\text{Person}^{(1)}$ ,  $\text{Manager}^{(1)}$ ,  $\text{Researcher}^{(1)}$  ir  $\text{cooperates-with}^{(2)}$  nekinta, o  $\text{reports-to}^{(2)}$  kinta – lyginėse būsenose atsiranda elementas ( $I1, I4$ )

### 24.3. Kas yra formaliai išreikštinė specifikacija?

Intensionalinei reliacinei struktūrai specifikuoti reikalinga kalba **L**. Tegu tai pirmosios eilės kalba (*first-order language*), t.y. predikatų kalba.

Predikatų kalbos *žodynas* (t.y. terminalinių simbolių aibė) susideda iš dviejų aibių: 1) konstantų simbolių (*constant symbols*, t.y. ženklu, vardu, kuriuos vaizduosime simbolių eilutėmis, *String* tipo), pvz., '*I1*', '*I2*', '*I3*', '*I4*', ir 2) predikatų simbolių (*predicate symbols*), pvz., '*Person*', '*Manager*', '*Researcher*', '*reports-to*', '*cooperates-with*'. Formaliai:

$$\mathbf{V} = \text{konstantos} \cup \text{predikatai}$$

Pirmosios eilės kalbos skiriasi nuo antrosios eilės kalbų tuo, kad predikatų argumentai gali būti išraiškos, kurias sudaro konstantos ir kintamieji, bet ne predikatų simboliai. Kitais žodžiais, lentelių elementai negali būti kitų lentelių vardu. Pavyzdžiuui, pirmosios eilės kalbos sakiniai yra '*reports-to*('I2', 'I1') = '*true*' ir  $\forall x,y \text{ } \text{'reports-to'}(x,y) \Rightarrow \text{'Person'}(x) \& \text{'Person'}(y)$ , bet nėra '*reports-to*('Person', 'Person').

### 24.8 apibrėžtis (ekstensionalinė pirmosios eilės struktūra $M = (S, I)$ ). Tegu

- **L** – pirmosios eilės kalba su žodynu **V**
- $S = (D, \mathbf{R})$  – ekstensionalinė reliacinė struktūra

**Ekstensionalinė pirmosios eilės struktūra** (dar vadinama **kalbos L modelis**) yra pora  $M = (S, I)$ , kur  $I$  (vadinama **ekstensionalinė interpretacijos funkcija**) yra atvaizdavimas  $I: \mathbf{V} \rightarrow D \cup \mathbf{R}$ , tokis, kad konstantų simboliai atvaizduojami į aibę  $D$  ir predikatų simboliai į  $\mathbf{R}$ . (Guarino et al. 2009, 3.1 apibrėžtis)  $\square$

### 24.9 apibrėžtis (intensionalinė pirmos eilės struktūra $K = (C, I)$ ). Tegu

- **L** – pirmosios eilės kalba su žodynu **V**
- $C = (D, W, \mathcal{R})$  – intensionalinė reliacinė struktūra, t.y. konceptualizacija

**Intensionalinė pirmos eilės struktūra** (dar vadinama **ontologinis įsipareigojimas**) kalbai **L** yra pora  $K = (C, I)$ , kur  $I$  (vadinama **intensionalinė interpretacijos funkcija**) yra atvaizdavimas  $I: \mathbf{V} \rightarrow D \cup \mathcal{R}$ , tokis kad konstantų simboliai atvaizduojami į aibę  $D$  ir predikatų simboliai į aibę  $\mathcal{R}$ . (Guarino et al. 2009, 3.2 apibrėžtis)  $\square$

**24.10 pavyzdys.** Prisiminkime einamąjį pavyzdį – ekstensionalinę reliacinę struktūrą iš 24.2 pavyzdžio. Tada intensionalinė pirmosios eilės struktūra yra tokia:

$$\begin{aligned} \mathbf{V} &= \{'I1', 'I2', 'I3', 'I4'\} \cup \\ &\quad \{'Person', 'Manager', 'Researcher', 'reports-to', 'cooperates-with'\} \end{aligned}$$

$$\begin{aligned} I: 'I1' &\rightarrow I1, \quad 'I2' \rightarrow I2, \quad 'I3' \rightarrow I3, \quad 'I4' \rightarrow I4, \\ 'Person' &\rightarrow Person^{(1)}, 'Manager' \rightarrow Manager^{(1)}, 'Researcher' \rightarrow Researcher^{(1)}, \\ 'reports-to' &\rightarrow reports-to^{(2)}, 'cooperates-with' \rightarrow cooperates-with^{(2)} \end{aligned}$$

### 24.11 apibrėžtis (numatomi modeliai $I_K(L)$ ). Tegu

- $C = (D, W, \mathcal{R})$  – intensionalinė reliacinė struktūra (konceptualizacija)
- **L** – pirmosios eilės kalba su žodynu **V**
- $K = (C, I)$  – intensionalinė pirmosios eilės struktūra (ontologinis įsipareigojimas)

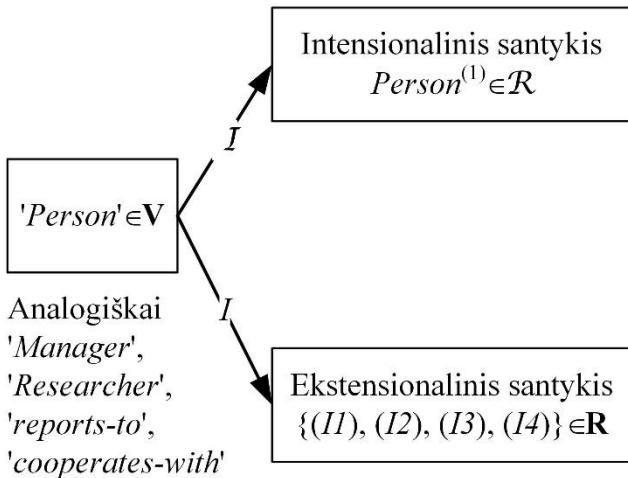
Modelis  $M = (S, I)$ , kur  $S = (D, \mathbf{R})$ , yra vadintamas numatomu kalbos **L** modeliu pagal **K** tada ir tik tada, kai

1. Kiekvienai konstantai  $c \in \mathbf{V}$  turime  $I(c) = J(c)$
2. Egzistuoja toks pasaulis  $w \in W$ , kad kiekvienam predikatiniam simbolui  $v \in \mathbf{V}$  egzistuoja intensionalinis santykis  $\rho^{(n)} \in \mathcal{R}$ , toks kad  $J(v) = \rho^{(n)}$  ir  $I(v) = \rho^{(n)}(w)$

**Numatomi modeliai  $\mathbf{I}_K(\mathbf{L})$**  yra aibė visų numatomų kalbos **L** modelių pagal **K**.  
(Guarino et al. 2009, 3.3 apibrėžtis)  $\square$

Pavyzdžiu, paimkime 24.7 pavyzdį. Pasaulio būsenai  $w_1$  turime (24.11 pav.):

$$\begin{aligned} I('Person') &= \{ (II), (I2), (I3), (I4) \} = Person^{(1)}(w_1) \\ I('reports-to') &= \{ (I2, II), (I3, II) \} = reports-to^{(2)}(w_1) \text{ ir t.t.} \end{aligned}$$



Intensionalinė pirmosios eilės struktūra  
(t.y. ontologinis įsipareigojimas)

$$[\mathbf{K} = (\mathbf{C}, I)]$$

$I : \mathbf{V} \rightarrow D \cup \mathcal{R}$  ir  $\mathbf{C} = (D, W, \mathcal{R})$   
intensionalinė reliacinė struktūra (t.y.  
konceptualizacija); žr. 24.9 apibrėžtį

Ekstensionalinė pirmosios eilės struktūra  
(t.y. kalbos **L** modelis)  $[\mathbf{M} = (S, I)]$ , kur

$I : \mathbf{V} \rightarrow D \cup \mathcal{R}$  ir  $S = (D, \mathcal{R})$   
ekstensionalinė reliacinė struktūra;  
žr. 24.8 apibrėžtį

**24.11 pav.** Predikatinis simbolis 'Person' turi ir ekstensionalinę interpretaciją (per ekstensionalinę pirmosios eilės struktūrą, t.y. modelį), ir intensionalinę interpretaciją (per intensionalinę pirmosios eilės struktūrą, t.y. per ontologinį įsipareigojimą) (Guarino et al. 2009, 3 pav.)

#### 24.12 apibrėžtis (ontologija **OK**). Tegu

- $\mathbf{C} = (D, W, \mathcal{R})$  – intensionalinė reliacinė struktūra (konceptualizacija)
- **L** – pirmos eilės kalba su žodynu **V**
- $\mathbf{K} = (\mathbf{C}, I)$  – intensionalinė pirmosios eilės struktūra (ontologinis įsipareigojimas)

Ontologija **OK** intensionalinei reliacinei struktūrai **C**, žodynui **V** ir ontologiniams įsipareigojimams **K** yra loginė teorija, sudaryta iš kalbos **L** tokios formulų aibės, kuri kiek galima geriau specifikuoja numatomų modelių aibę  $\mathbf{I}_K(\mathbf{L})$ . (Guarino et al. 2009, 3.4 apibrėžtis)  $\square$

**24.13 pavyzdys.** Ontologija yra logikos formulų aibė. Toliau ontologijos nuo  $O_1$  to  $O_5$  specifikuos personalo dalykinę sritį, apie kurią kalbama 24.2 ir 24.7 pavyzdžiuose. (Guarino et al. 2009, 3.2 pavyzdys)

- Klasifikavimas. Sąvokos *Researcher* ir *Manager* yra *Person* poaibiai. Kitais žodžiais, jos yra su paskutiniuoją sąvoka ryšyje *is-a*:

$$\begin{aligned} O_1 = \{ & 'Researcher'(x) \Rightarrow 'Person'(x), \\ & 'Manager'(x) \Rightarrow 'Person'(x) \} \end{aligned}$$

- Apibrėžimo ir kitimo sritys. Pridedame prie  $O_1$  dar keletą formulų:

$$O_2 = O_1 \cup \{ \begin{aligned} & \text{'cooperates-with'}(x, y) \Rightarrow \text{'Person'}(x) \ \& \ \text{'Person'}(y) \\ & \text{'reports-to'}(x, y) \qquad \qquad \Rightarrow \text{'Person'}(x) \ \& \ \text{'Person'}(y) \end{aligned} \}$$

- Simetrija:

$$O_3 = O_2 \cup \{ \text{'cooperates-with'}(x, y) \Leftrightarrow \text{'cooperates-with'}(y, x) \}$$

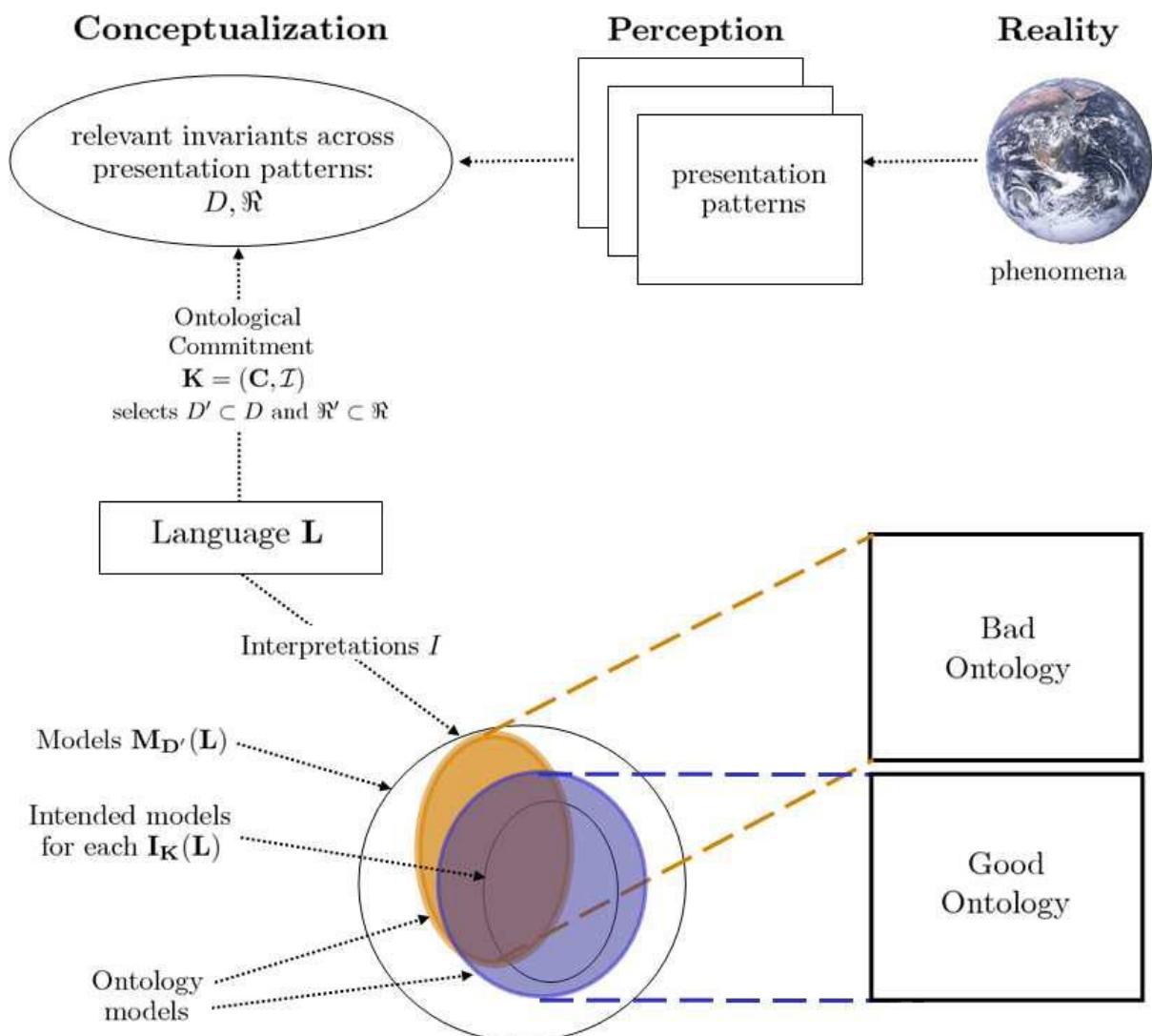
- Tranzityvumas:

$$O_4 = O_3 \cup \{ \text{'reports-to'}(x, z) \Leftarrow \text{'reports-to'}(x, y) \ \& \ \text{'reports-to'}(y, z) \}$$

- Nepersikirtimas. Nėra tokio *Person*, kuris būtų ir *Researcher*, ir *Manager*:

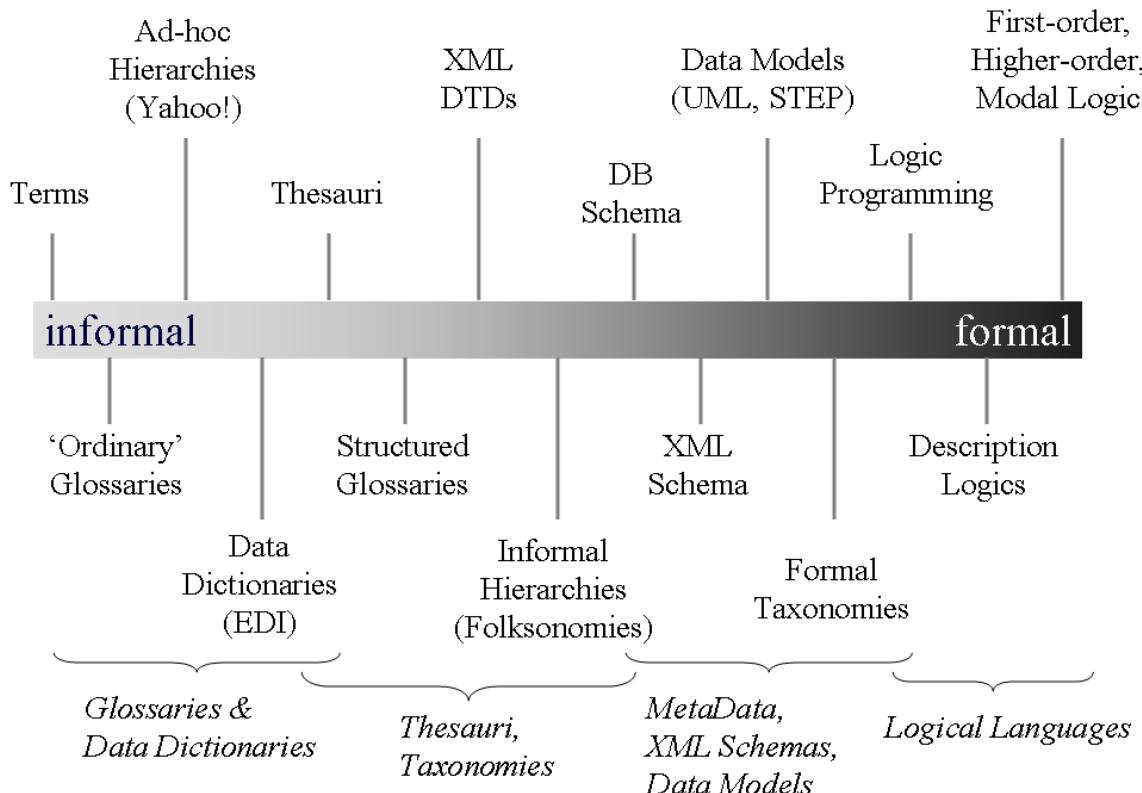
$$O_5 = O_4 \cup \{ \text{'Manager'}(x) \Rightarrow \neg \text{'Researcher'}(x) \}$$

Kuo daugiau formulų ontologijoje, tuo daugiau apribojimų ir tuo mažesnė numatomų modelių  $I_K(L)$  aibė (24.12 pav.). Kitaip sakant, jeigu  $D' \subset D$  ir  $R' \subset R$ , tai  $M_{D'}(L) \supset M_D(L)$ .



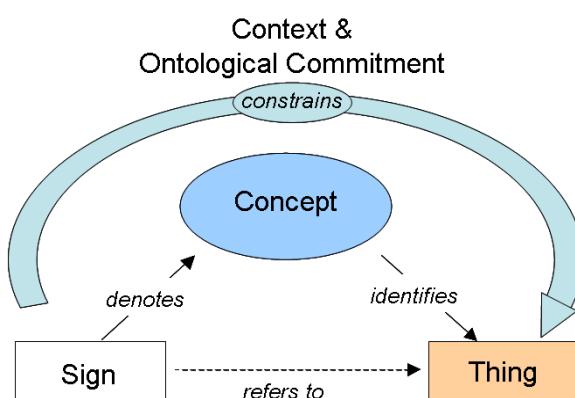
**24.12 pav.** Ryšiai tarp reišinių realybėje, jų suvokimo įvairiais laiko momentais, jų abstrakčios konceptualizacijos, kalbos konceptualizacijai specifikuoti, numatomų modelių ir ontologijos (Guarino et al. 2009, 2 pav.)

Ontologijoms užrašyti yra naudojamos įvairios kalbos. Jos gali būti mažiau formalios negu predikatų kalba. Įvairovė apima spektrą nuo silpnai formalizuotų žmogiškųjų kalbų iki labiau formalų kalbų, tame tarpe algoritminių kalbų (24.13 pav.). Paminėsime deskriptyvią kalbą OWL (*Web Ontology Language*, <http://www.w3.org/2001/sw/wiki/OWL>). Ji skirta ontologijoms semantiniame saityne (*semantic Web*). Ji grindžiama *deskriptyviaja logika*. „Deskriptyviosios logikos yra žinių vaizdavimo kalbų šeima, kuri gali būti panaudota taikomosios srities struktūrizuotų ir formaliai apibrėžtų žinių aprašymui“ (Norgėla, Sakalauskaitė 2007, p. 98).



**24.13 pav.** Kalbų L įvairovė. Formaliomis logikomis grindžiamos kalbos tinkamai formaliomis išreikštinėmis specifikacijomis, kitaip sakant, ontologijoms užrašyti (Guarino et al. 2009, 4 pav.)

Semiotinis trikampis, kuris parodo ryšius tarp ženklo ir jo prasmės, yra 24.14 pav.



**24.14 pav.** Semiotinis trikampis (Guarino et al. 2009, 6 pav.)

## 24.4. Skirtingi modeliai tai pačiai specifikacijai

Pateiksime pavyzdžius, kai skirtingi modeliai tenkina tą pačią formulų aibę. Formulės yra logikos kalba **L**.

**24.11 pavyzdys** (Loeckx et al. 1996, example 2.6, p. 29). Tegu **L** yra kalba, kurios alfabetą **V** sudaro vienas konstantinis simbolis '*0*' ir vienas funkcinis simbolis '*Succ*'. Pastarasis yra vardas vieno argumento operacijos '*Succ*':  $\_ \rightarrow \_$ . Tai sintaksinės esybės, nes joms kol kas nepriskirtos jokios semantinės esybės.

$$V = \{ '0' \} \cup \{ 'Succ' \}$$

Kalbos **L** termo pavyzdys yra '*Succ*('Succ'(x)). Kalbos **L** bazinių termų (t. y. termų be laisvų kintamujų) aibė yra tokia:

$$\{ '0', 'Succ'('0'), 'Succ'('Succ('0'))), 'Succ'('Succ('Succ('0')))), \dots \}$$

Termai apibrežiami induktyviai. Konstantinis simbolis yra termas. Kintamasis yra termas. Jeigu '*f*' yra  $k$ -vietis funkcinis simbolis,  $k \geq 1$ , ir  $t_1, \dots, t_k$  yra termai, tai '*f*'( $t_1, \dots, t_k$ ) yra termas. Termas gali būti vaizduojamas medžiu. Išsišakojimai yra viršūnėse, kurios žymi  $k$ -viečius funkcinius simbolius su  $k \geq 2$ .

Tegu formulų aibė ontologijoje yra tuščia, t. y.  $O = \emptyset$ . Pateiksime tris kalbos **L** modelius.

**Modelis Nat.** Klasikiniu **L** modeliu yra natūrinių skaičių aibė  $D = Nat = \{0, 1, 2, 3, \dots\}$ . Funkcinis simbolis '*Succ*' („sekantis“, *successor*) yra interpretuojamas kaip funkcija, kuri prie argumento prideda *I*:

$$I_{Nat} ('Succ') (n) = n + 1$$

Interpretacija  $I_{Nat}$  atvaizduoja termus (sintaksines esybes) į semantines esybes – natūrinius skaičius:

$$I_{Nat} : \begin{aligned} '0' &\rightarrow 0, \quad 'Succ'('0') \rightarrow 1, \quad 'Succ'('Succ('0')) \rightarrow 2, \quad 'Succ'('Succ('Succ('0'))) \rightarrow 3, \dots \\ 'Succ' &\rightarrow \_ + 1 \end{aligned}$$

**Modelis Bool.** Modeliu yra loginių reikšmių aibė  $D = Bool = \{ true, false \}$ . Konstantinis simbolis '*0*' interpretuojamas *false*, '*Succ*' – funkcija, kuri „apverčia“ argumentą:

$$I_{Bool} ('Succ') (x) = \begin{cases} false & \text{if } x = true \\ true & \text{if } x = false \end{cases}$$

Interpretacija  $I_{Bool}$  atvaizduoja:

$$I_{Bool} : \begin{aligned} '0' &\rightarrow false, \quad 'Succ'('0') \rightarrow true, \quad 'Succ'('Succ('0')) \rightarrow false, \\ &\quad \quad \quad 'Succ'('Succ('Succ('0'))) \rightarrow true, \dots \end{aligned}$$

**Modelis C.** Modeliu yra aibė iš vieno elemento  $D = D_C = \{\#\}$ . Konstantinis simbolis '*0*' is yra interpretuojamas *#*, '*Succ*' – funkcija, kuri atvaizduoja *#* į *#*:

$$I_C ('Succ') (\#) = \#$$

Interpretacija  $I_C$  atvaizduoja:

$$\mathcal{I}_C : '0' \rightarrow \#, \quad 'Succ('0') \rightarrow \#, \quad 'Succ('Succ('0')) \rightarrow \#, \quad 'Succ('Succ('Succ('0'))) \rightarrow \#, \dots$$

**24.12 pavyzdys** (Loeckx et al. 1996, example 5.24 iv, p. 87). ). Tegu **L** yra kalba, kurios alfabetą **V** sudaro vienas konstantinis simbolis ' $0$ ' ir vienas funkcinis simbolis '+':

$$V = \{'0'\} \cup \{+'\}$$

Tegu ontologiją sudaro keturių formuliu aibė  $O$ :

$$\begin{aligned} O = \{ & \quad x +' (y +' z) = (x +' y) +' z, \\ & \quad x +' '0' = x, \\ & \quad '0' +' x = x, \\ \} \quad \forall x \exists y (x +' y = '0' \& y +' x = '0') \} \end{aligned} \quad (24.1)$$

Pirmoji formuluojas asociatyvumą. Antroji ir trečioji formuluojas neutralaus elemento ' $0$ ' savybes sudėties operacijoje '+'. Ketvirtoji formuluojas atvirkštinę elementų egzistavimą. Ontologijos formulės suprantamos kaip aksiomos, kurios specifikuojas modeli.

Modeliu yra bet kuris abstrakčiosios algebro grupė. Taip yra todėl, kad ontologija (24.1) formuluojas grupės aksiomos. Toliau pateikiame du modeliai su begaline ir baigtine grupėmis.

**Modelis Z.** Klasikinė begalinės grupė yra sveikujų skaičių aibė  $D = Z = \{\dots, -3, -2, -1, 0, 1, 2, 3, \dots\}$ . Čia operacija '+' yra interpretuojama kaip sveikujų skaičių sudėties funkcija:

$$\mathcal{I}_Z(+) (n,m) = n + m$$

Atvirkštinio elemento  $y$  egzistavimą ketvirtoje (24.1) formulėje galima pakeisti Skolemo terminu ' $Inv'(x)$ . Toks pakeitimas vadinamas skolemizavimu. Interpretacija  $\mathcal{I}_Z$  atvaizduoja ' $Inv'$  į vienvietę funkciją „minus“ nuo argumento  $n$ :

$$\mathcal{I}_Z(Inv) (n) = -n$$

Sveikieji skaičiai tenkina visas keturias formules ontologijoje  $O$  (24.1). Visiems  $x, y \in Z$  yra tiesa, kad  $x + (y + z) = (x + y) + z$ ,  $x + 0 = x$ ,  $0 + x = x$ , o  $x$  atvirkštinis elementas yra  $-x$ . Be to  $-(-x) = x$ .

**Model  $Z_n$ .** Baigtiniu modeliu yra liekanų moduliui  $n$  grupė  $D = Z_n = \{0, 1, 2, \dots, n-1\}$ , kur  $n$  yra fiksotas natūrinis skaičius. Čia operacija operation '+' interpretuojama kaip sudėties moduliui  $n$ :

$$\mathcal{I}_{Z_n}(+) (l,m) = l + m \text{ modulo } n \quad (\text{or simply } l \oplus_n m)$$

Kai  $n=3$ , tai turime  $Z_3 = \{0, 1, 2\}$  ir

$$\begin{aligned} 0 + 0 \text{ modulo } 3 &= 0 & (\text{or } 0 \oplus_3 0 = 0) \\ 0 + 1 \text{ modulo } 3 &= 1 & (\text{or } 0 \oplus_3 1 = 1) \\ 0 + 2 \text{ modulo } 3 &= 2 & (\text{or } 0 \oplus_3 2 = 2) \\ 1 + 2 \text{ modulo } 3 &= 0 & (\text{or } 1 \oplus_3 2 = 0) \\ 2 + 2 \text{ modulo } 3 &= 1 & (\text{or } 2 \oplus_3 2 = 1) \end{aligned}$$

Kai  $n=2$ , tai  $Z_2 = \{0, 1\}$ . Rašoma  $0 \oplus 0 = 0$ ,  $0 \oplus 1 = 1$ ,  $1 \oplus 1 = 0$ .

Liekanų aritmetikoje du sveikieji skaičiai yra sudedami ir ši suma dalijama iš skaičiaus  $n$ , kuris vadinamas moduliui, ir rezultas yra liekana. Sveikieji skaičiai nuo  $0$  to  $n-1$  sudaro taip

vadinamą liekanų grupę sudėties moduliu  $n$  atžvilgiu. Elemento  $a$  atvirkštinis yra  $n - a$ , o  $0$  yra neutralus elementas  $0$ . Iliustracija yra laikrodžio ciferblatas. Tegu dabar  $9$  valanda. Kiek valandų rodys laikrodis už  $4$  valandų? Atsakymas:  $1$  valanda. Aiškinama, kad  $9 + 4$  moduliu  $12$  yra lygu  $1$ . Rašoma  $9 + 4 \text{ modulo } 12 = 1$ . Liekanų moduliu  $n$  grupė žymima  $Z_n$ ; žr. [https://en.wikipedia.org/wiki/Group\\_\(mathematics\)](https://en.wikipedia.org/wiki/Group_(mathematics)).

**24.13 pavyzdys** (Loeckx et al. 1996, example 5.24 iv, p. 87). Tegu **L** yra kalba, kurios alfabetą **V** sudaro vienas konstantinis simbolis ' $0$ ' ir du funkciniai simboliai '+' and 'Succ':

$$V = \{'0'\} \cup \{ '+', 'Succ'\}$$

Tegu ontologiją  $O$  sudaro dvi formulės:

$$\begin{aligned} O = \{ & x ' +' '0' = x, \\ & 'Succ'(x ' +' y) = x ' +' 'Succ'(y) \} \end{aligned} \quad (24.2)$$

Pirmai formuluojama neutralaus elemento savybę. Antrai formuluojama skliaustelių perkėlimą operacijoje 'Succ'.

Modeliu yra natūrinių skaičių aibė  $D = \{0, 1, 2, 3, \dots\}$ . Funkcinius simbolius 'Succ' yra interpretuojamas kaip funkcija, kuri prie argumento prideda  $1$ . Here the successor function is interpreted as adding  $1$  to its argument. Operacija '+' yra interpretuojama kaip sveikujų skaičių sudėtis +. Interpretacija  $I$  atvaizduoja:

$$\begin{aligned} I: & '0' \rightarrow 0, \quad 'Succ'('0') \rightarrow 1, \quad 'Succ'('Succ('0')) \rightarrow 2, \quad 'Succ'('Succ('Succ('0'))) \rightarrow 3, \dots \\ & '+' \rightarrow +, \\ & 'Succ' \rightarrow \_ + I \end{aligned}$$

Paaškinsime aukšciau esančią paskutinę eilutę ' $Succ' \rightarrow \_ + 1$ '. Funkcinius simbolius 'Succ' atvaizduojamas į vienvietę funkciją, kuri prie argumento prideda  $1$ . Natūriniai skaičiai tenkina ontologiją  $O$  (24.2), nes visiems  $x, y \in D$  yra tiesa, kad  $x + 0 = x$  ir  $(x + y) + 1 = x + (y + 1)$ .

Kitu tos pačios kalbos **L** modeliu yra lyginių skaičių aibė  $D = \{0, 2, 4, 6, \dots\}$ . Operacija 'Succ' yra interpretuojama kaip funkcija, kuri prideda  $2$  prie argumento. Interpretacija  $I$  atvaizduoja:

$$\begin{aligned} I: & '0' \rightarrow 0, \quad 'Succ'('0') \rightarrow 2, \quad 'Succ'('Succ('0')) \rightarrow 4, \quad 'Succ'('Succ('Succ('0'))) \rightarrow 6, \dots \\ & '+' \rightarrow +, \\ & 'Succ' \rightarrow \_ + 2 \end{aligned}$$

Aukšciau paskutinė eilutė ' $Succ' \rightarrow \_ + 2$ ' žymi, kad funkcinis simbolis 'Succ' yra atvaizduojamas į vienvietę funkciją, kuri prideda  $2$  prie argumento. Kadangi lyginių skaičių aibė yra izomorfiška natūrinių skaičių aibei, tai lyginiai skaičiai kaip modelis nėra nagrinėjami.

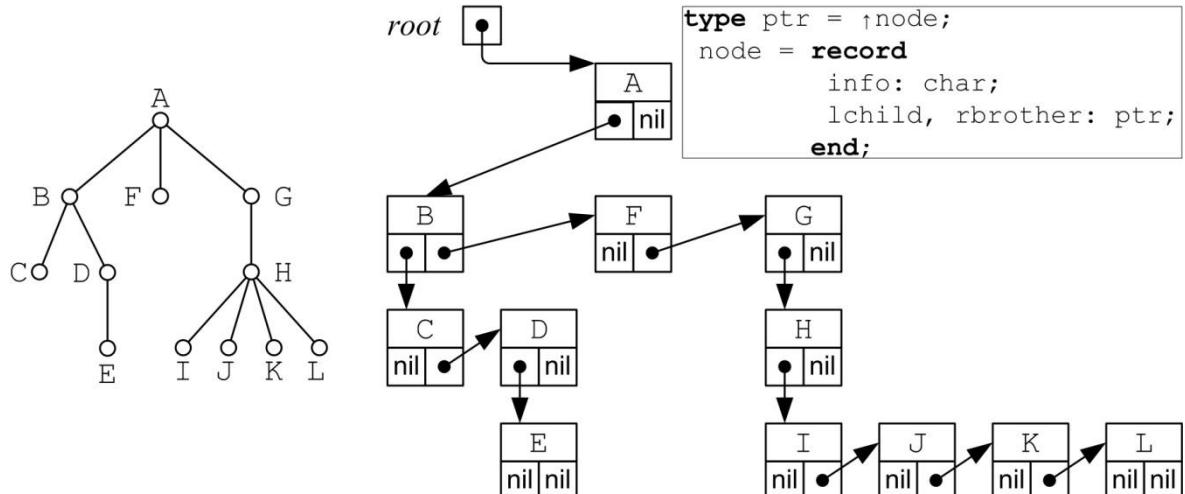
## 25. Dirbtinio intelekto egzamino bilietai

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, užsicklinimo 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 ABC.DE...F.GHI.J.K.L arba 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 briaunas 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 taisykę.

12. Ekspertinė sistema kaip dirbtinio intelekto sistema; jos architektūra, demonstracinis 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.

16. Ekstensionalinė reliacinė struktūra, pasaulis, intensionalinis santykis, intensionalinė reliacinė struktūra, ekstensionalinė pirmos eilės struktūra (kalbos modelis), intensionalinė pirmos eilės struktūra (ontologinis įsipareigojimas), numatomai modeliai ir ontologija.

17. Skirtingi modeliai tai pačiai specifikacijai.

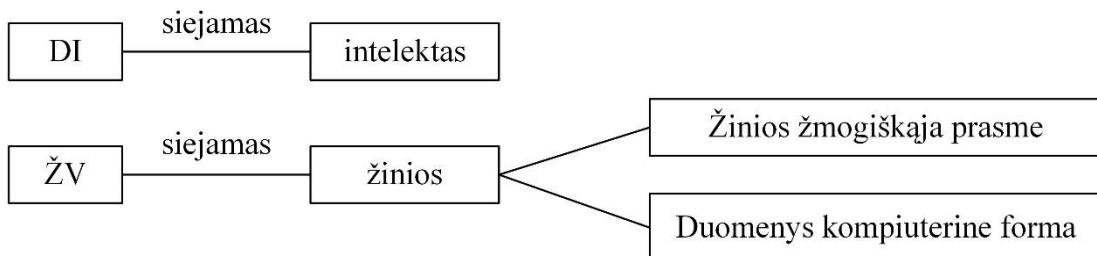
## Antra dalis

### 26. Žinių vaizdavimas kaip dirbtinio intelekto šaka

Šiame vadovėlyje vadovaujamas požiūriu, kad *žinių vaizdavimas* (*knowledge representation*) toliau ŽV, yra dirbtinio intelekto šaka. Egzistuoja ir kitokie požiūriai.

Žinių vaizdavimo ir samprotavimo žinyne (van Harmelen et. al 2008) pateikiamas tokis apibūdinimas. „*Žinių vaizdavimas ir samprotavimas yra dirbtinio intelekto iššukių centre: suprasti intelekto ir pažinimo prigimtį tokiu mastu, kad kompiuteriai demonstruotų žmogui būdingus gebėjimus*“. Pastebime, kad čia akcentuojamas pažinimas, o ne dirbtinio žmogaus sukūrimas.

Sąvokos „žinių vaizdavimas“ turinys atskleidžiamas žodžiu „žinios“ ir „vaizdavimas“ gramatiniu aiškinimu (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 paplitusiouose vadoveliuose, pvz., (Brachman, Levesque 2004; Russell, Norvig 2003; Stefik 1995).



26.1 pav. DI yra siejamas su žodžiu „intelektas“, o ŽV – su „žinios“

Sąvoka žinios (*knowledge*) gali būti apibrėžiama įvairiai, atsižvelgiant į aiškinimą įvairiuose moksluose, pvz., filosofijoje, jos šakoje kognityvistikoje (pažinimo teorijoje), sociologijoje, istorijoje ir informatikoje.

Mūsų nuostata, kad žinių vaizdavimas – tai žinių vertimas duomenimis. Sąvoka duomenys (*data*) informatikoje nėra mistinė, o turi visuotinai priimtą turinį.

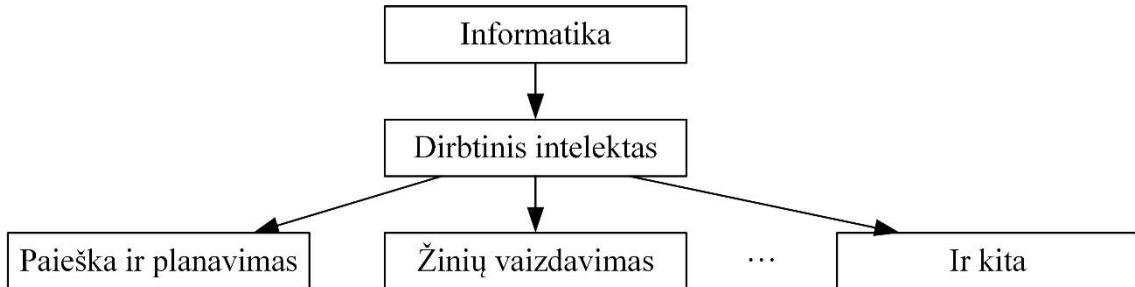
DI sampratą apsprendžia riba tarp žmogiškojo intelekto (*human intelligence*) ir mašininio intelekto (*machine intelligence*). DI sampratai yra būdinga, kad ji kinta. Šios ribos plėtimasis sąlygoja ir sąvokos *intelektuali sistema* (*intelligent system*) sampratą. DI riba plečiasi, užgriebdama vis naujas 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 vystymasis sąlygoja kad kiekvienam dešimtmeciui yra būdinga tam tikra DI samprata. Yra išskiriamos tam tikros *programavimo eros*. Analogiskai yra su DI ir ŽV. Todėl pateisinama, kad skirtingi autoriai akcentuoja skirtingus požiūrius į DI ir ŽV.

Vadovaujantis (Nilsson 1982) *dirbtinio intelekto sistema* yra suprantama kaip *produkčijų sistema*, kuri, savo ruožtu, yra trejetas: *globali duomenų bazė*, *produkčijų aibė* ir *valdymo sistema*. Remiantis tokiu modeliu dalykinės srities žinios klasifikuojamos pagal tai, kur yra vaizduojamos. *Deklaratyviųsiomis 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.



26.2 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* (*data mining*). Pastaroji yra siejama ir su DI, o ne tik su duomenų bazių valdymo sistemomis ar statistika.

Žinių vaizdavimo būdai (t.y. kryptys, paradigmos) skirstomos į dideles šakas:

1. **procedūrinis** (*procedural*) žinių vaizdavimas. Algoritmai ir programos yra šio vaizdavimo pavyzdžiai (pavaizdavimas plačiąja prasme);
2. **deklaratyvus** (*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*. Operatorius *O(A)* žymi „privalo būti *A*“ (*it is obligatory that A*), o *P(A)* – „gali būti *A*“ (*it is permitted that A*).
  - 4) Nurungimo logika (*defeasibility logic*)
2. **procedūrinis** (siaurajā prasme)
  - 1) Produkcių sistemos (pvz.  $A, B, C \rightarrow D, E$ )
3. **tinklinis** (network representation schema)
  - 1) semantiniai tinklai (*semantic networks*). Semantinis tinklas – tai grafas, kurio viršūnėms ir briaunoms suteikiama tam tikra prasmė.
  - 2) koncepciniai grafai (*conceptual graphs*). Koncepcinis grafas – tai grafas, kuris atspindi dalykinės srities **sąvokas**.
4. **struktūrinis** (structured knowledge representation)
  - 1) freimai (*frames*)
  - 2) objektai (*objects*)

Supaprastintai skirtumą tarp procedūrinio ir deklaratyvaus žinių vaizdavimo galima apibūdinti 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.

## 26.1. 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 vaizdavime esminė sąvoka yra išvedimas (*entailment*). Kitokios duomenų, informacijos ir žinių prasmės yra nagrinėjamo komunikacijos moksle (*communication science*).

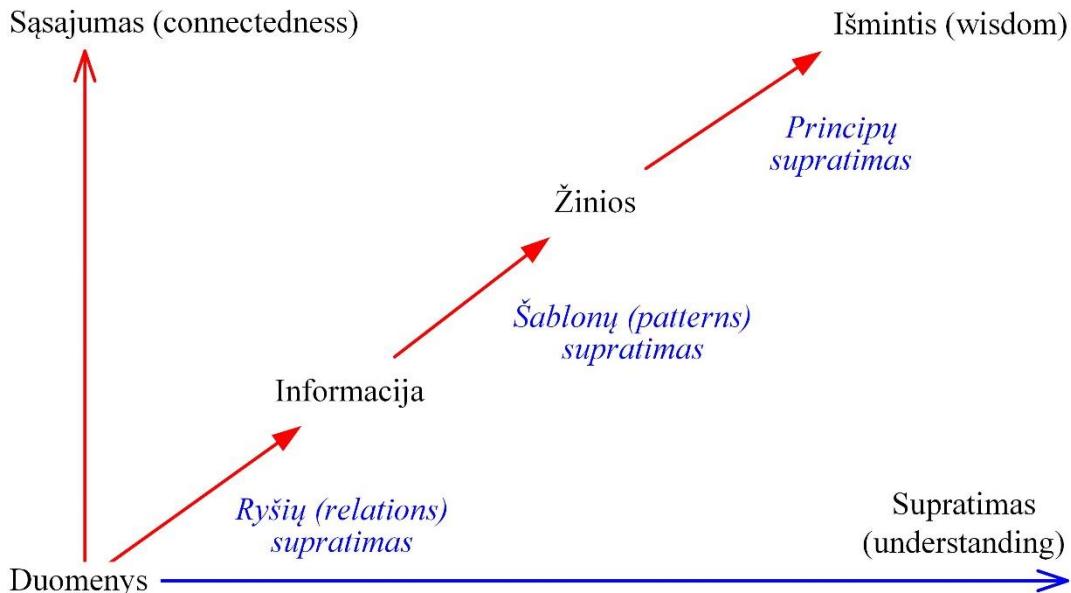
Apibūdindami sąvokas „duomenys“, „informacija“, „žinios“ ir „išmintis“ sistemų teorijos specialistai (Bellinger et al. 2004) pasitelkia pragmatines veiklas. Jie pradeda cituodami sistemų teorijos specialistą ir organizacijų kaitos profesorių Russell Ackoff (1985), kuris išskiria penkias kategorijas:

1. *duomenys*. Tai simboliai;
2. *informacija*. Tai duomenys, kurie yra apdorojami ir yra naudingi.; atsako į klausimus „kas“, „ką“, „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 praetimi – kas yra arba buvo žinoma. Tik penktoji – išmintis – yra siejama su ateitimi, nes apima įžvalgumą ir sumanymą. Šie apibūdinimai išplečiami iki modelio 26.3 pav. (Bellinger et al. 2004):

1. *duomenys* yra neapdoroti. Pavyzdžiui, kompiuterinės skaičiuoklės lentelė (*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 interpolavimu 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 išmokimo 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 ekstrapoliavimu, o taip pat determinuotumo bei tikimybės nebuvinu. 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. Kompiuteriai niekada neturės išminties. Išmintis būdinga tik žmogui. Išminčiai būtina siela, kuri glūdi ne tik prote, bet ir širdyje. Siela kompiuteriai neturės.

Bellinger et al. (2004) duomenis, informaciją, žinias ir išmintį vaizduoja semantinio grafo viršūnėmis, o supratimą – kaip briaunas (perėjimus) tarp šių viršūnių.



26.3 pav. Trys supratimo lygmenys kaip perėjimai tarp duomenų, informacijos, žinių ir išminties (Bellinger et al. 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 ląstelėje parašytas skaičius 7. Šiam skaičiui nesuteikiama jokia prasmė. Jeigu pasakytume, kad tai 7 kilometrai, 7 mylios, 7 ar 7 000 eurų – 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. Išmintis pasižymi ekstrapoliavimu. Pavyzdžiui:

*„Lauke lyja, nes yra toks reiškinys – lietus. Lietus kaip gamtos reiškinys apima supratimą apie garavimą, oro sroves, temperatūros pokyčius, garų kondensaciją, lietų ir kt.“*

Aukščiau pateikti modeliai labiau aktualūs komunikacijos moksluose (*library, information and communication sciences*). Pastaruosiuose esminė sąvoka yra informacija. O informatikoje svarbesnė yra duomenų sąvoka. Žinių vaizdavime svarbesnė yra išvedimo sąvoka.

Interpretacijos perėjime nuo duomenų prie informacijos pavyzdys toks. „Važiavau mieste 50, mane sustabdė kelių policininkas ir nubaudė už greičio viršijimą“. Tegu toks tekstas saugomas dokumentų valdymo sistemoje, kaip pažeidėjo parodymai. Jo interpretacija: 50 mylių per valandą. Tokia interpretacija yra ir tiketina 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.

**Ko nepasieks dirbtinis intelektas?** Nepasiekiamu užprogramuoti žmonių išmintį. Dirbtinis intelektas yra apie kompiuterių programavimą užduotims, kurios įprastai priskiriamos žmogaus protui. Sekoje duomenys-informacija-žinios-išmintis pasiseks užprogramuoti žinias, bet ne išmintį. Išmintis apima žmogaus gebėjimą atsižvelgti į VISAS aplinkybes. O kompiuteris negali atsižvelgti į visas aplinkybes; jam prieinamos tik užprogramuotos žinios, galbūt rastos tekstuose internete ir pan. Kompiuteris negalės pasverti papročiais, morale, istorija teise ir kitaip grindžiamų, galbūt prieštaringuų, sprendimų ir nuspręsti į ateitį. Todėl nebus užprogramuotas visažinantis teisingų sprendimų priemėjas, pvz., automatinis teisėjas. Nors tam tikrose srityse paprastais, bet ne sunkiais atvejais, nuspręsti gali ir kompiuteris. Pvz., jeigu viršijai greitį, tai mokėk baudą. Vairuotojas gali įrodinėti pareigūnui, kad viršijo greitį siekdamas išvengti didesnės žalos, pvz., vežę į ligoninę.

## 26.2. Žmogiškasis pažinimas ir žiniomis grindžiamos sistemos

Šiame vadovelyje žinių vaizdavimas siejamas su dviem sąvokomis. Tai:

1. *žmogiškasis pažinimas*. Visų pirma, tai žmogui būdingo smalsumo patenkinimas. Antra, pažinimas kaip mokslinė, projektinė veikla, tarp kurios rezultatų yra ataskaita (*report*) apie tam tikroje dalykinėje srityje atskleistus faktus ir dėsnius;
2. *žiniomis grindžiamos sistemos*.

Šios dvi sąvokos skirtos dviejų tipų siekiams. Į šias sąvokas galime žiūrėti kaip į 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 reikia šią sritį pažinti (26.4 pav.).



26.4 pav. Dvi paveikslė interpretacijos. Pirma, žmogiškasis pažinimas yra svaresnis motyvas negu žiniomis grindžiamų sistemos. Antra, žmogiškasis pažinimas yra būtinės kaip žinių išgavimo stadija kuriant žiniomis grindžiamą sistemą

**Žiniomis grindžiama sistema (ŽGS)** (*knowledge-based system*, KBS) – tai kompiuterinė sistema, kuri naudoja žinas. Šis terminas gali būti trumpinamas iki **žinių sistema** (*knowledge system*) (Stefik 1995). Toks trumpinimas priimtinas vienos knygos rėmuose. Tačiau bendru atveju ir pavartotas neišskiriant tam tikro konteksto jis gali būti suprastas kaip žinių iš tam tikros dalykinės srities, o ne jos apdorojimo būdą, 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 dalykinės srities informaciją. Tai sistematizuota informacija. Pavyzdžiu, Universalioji dešimtainė klasifikacija, UDK (универсальная десятичная классификация, УДК), gali būti priskiriama prie informacijos sistemų. UDK – tai sisteminė, visas žinių sritis aprépianti bibliotekinė-bibliografinė klasifikacija.

I žiniomis grindžiamas sistemas galima žiūrėti kaip į atskirą informacinių sistemų tipą:

**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** (*knowledge acquisition*) (Stefik 1995, p. 217). Žinių išgavimo veikloje būtina pažinti dalykinę sritį. Tiesa, **žinių inžinierius** neprivalo jos pažinti taip giliai kaip dalykinės srities **ekspertas**.

Žinios, esančios ŽGS, turi būti prieinamos tiems, kam jos yra reikalingos (Buchanan et al. 1990). Žiniomis grindžiamų sistemų paskirtis yra apibūdinama šitaip: kaip laiku ir tiksliai pateikti žinas tikrai užduočiai (*task*). Kitaip sakant, kaip profesinę patirtį tam tikroje srityje pateikti sprendimų priėmėjams, kada jiems reikia ir kur reikia. Terminą „ekspertinė sistema“ naudojamas dėl tokių priežasčių:

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 uždaviniių 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. Jis žymi dirbtinio intelekto taikymą uždaviniių sprendimo (*problem-solving*) ir sprendimų priėmimo (*decision-making*) užduotyse.

ŽGS architektūroje yra išskiriamos dvi sudėtinės dalys:

1. samprotavimo (*reasoning*) (arba užduoties sprendimo) procesas;
2. žinių bazė (*knowledge base*)

ŽGS skirta tik *tam tikro* tipo užduotims. Dirbtinio intelekto „jaunystėje“ mokslininkai tikėjosi, kad bus sukurtas bendras užduočių sprendėjas (*general problem solver*) (Ernst, Newell 1969). Deja tokio universalaus sprendėjo neįmanoma sukurti iš principo. Yra algoritmiškai neišprendžiamų uždaviniių, ir jie nagrinėjami matematinėje logikoje. Išskiriamos tokios tipinės (*generic*) DI užduotys: numatymas, valdymas, diagnozė, nurodymas (Valente 1995, p. 136).

### 26.3. Žinių vaizdavimas ir žinių vadyba

Kalbant apie žinias galima eiti dviem kryptimis: 1) žinių vaizdavimas ir 2) žinių vadyba (*knowledge management*) (26.5 pav.). Pirmoji priskiriama informatikai (*computer science*), antroji – informacinių technologijų, informacijos ir komunikacijos mokslui, organizacijų mokslo (*organisation science*) ir kitoms šakoms, ku svarbu didinti organizacijos vertę.



26.5 pav. Žinių vaizdavimo ir žinių vadybos vieta kalbant apie žinias

Duomenų apdorojime duomenų bazėse svarbu išrinkimas (*retrieval*). Žinių vaizdavimas ir dirbtinis intelektas esmė siejamas su implikavimu (*implication*), sąlygojimu (*entailment*, *logical entailment*), samprotavimu (*reasoning*). Samprotavimas (*reasoning*) priskirtinas žmogui ir sietinas su išvedimu (*inference*) dirbtiniame intelekte.

Išrinkime svarbus algoritmą sudėtingumas, kuris yra polinominis, paprastai logaritminis, tiesinis, kvadratinis ar kubinis. Išvedimas dirbtiniame intelekte siejamas su eksponentiniu sudėtingumu ir algoritmiskai neišsprendžiamais uždaviniais.

Nagrinėkime išrinkimą duomenų bazėse. Tegu ieškomas įrašas atlyginimų lentelėje (žr. 26.2 lentelę) pagal užklausą su žmogaus pavarde Atlyginimas ('Butkaitis')=?

Pavarde	Atlyginimas (Lt)
Adoraitis	1000
Aldukaitis	900
Bajoraitis	1200
Caraitis	1000
...	
Žabaitis	1100

26.2 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 (*hashing*) arba indekso failas.

Dirbtinio intelekto metodų sudėtingumas yra eksponentinis  $O(2^N)$ . Pavyzdžiui, kelio keliaujančio pirklio uždavinyje su  $N$  miestų. Duoti atstumai tarp miestų. Rasti trumpiausią kelią apeiti visus miestus. Visų kelių perrinkimas yra eksponentinis.

## 26.4. Žinių vaizdavimo būdų apžvalga

Deklaratyvaus žinių vaizdavimo būdus yra priimta klasifikuoti į keturias grupes (Sowa 2000; Brachman, Levesque 2004):

1. loginis žinių vaizdavimas;
2. procedūrinis žinių vaizdavimas (siauraja prasme);
3. tinklinis žinių vaizdavimas;
4. struktūrinis žinių vaizdavimas.

**Loginis žinių vaizdavimas.** Jo būdų pagrindas yra pirmos eilės logika:

1. teiginių logika; kitaip dar *teiginių skaičiavimas* (*proposition calculus*);
2. predikatų logika; kitaip dar *predikatų skaičiavimas* (*predicate calculus*).

**Procedūrinis žinių vaizdavimas.** Siauraja prasme yra suprantamas kaip produkcijų sistemos. 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.

**Struktūrinis žinių vaizdavimas.** Jis siejamas su freimo (*frame*) sąvoka, kurią pasiūlė Marvinas Minskis. Freimas – tai duomenų struktūra skirta pavaizduoti stereotipinę situaciją, pavyzdžiui, buvimo kambaryje suvokimui arba samprotavimams apie atėjančią draugo gimtadienį. Freimas vaizduojamas tinklu iš viršūnių ir ryšių. Struktūriniam žinių vaizdavimui gali būti priskiriami objektiniame programavime naudojami deklaratyvaus vaizdavimo būdai.

**Tinklinis žinių vaizdavimas** (*network representation schema, network representation*). Juo apimi tokios būdų grupės:

1. semantiniai tinklai (*semantic networks*);
2. koncepciniai grafai (*conceptual graphs*).

Abi šios sąvokos gali būti suprantamos labai neformaliai ir todėl yra giminingos. Yra autorių, kurie kalbėdami apie tą patį pasirenka arba pirmąjį, arba antrąjį terminą.

Semantinis tinklas yra grafas, kuriuo vaizduojamos klasifikavimo žinios apie objektus ir jų savybes. Viršūnėmis vaizduojamos sąvokos, o briaunomis ryšiai (*relationships*).

Išskiriami šių rūsių ryšiai, vadinami *universaliaisiais*:

1. *is-a*;
2. *instance-of*;
3. *part-of*.

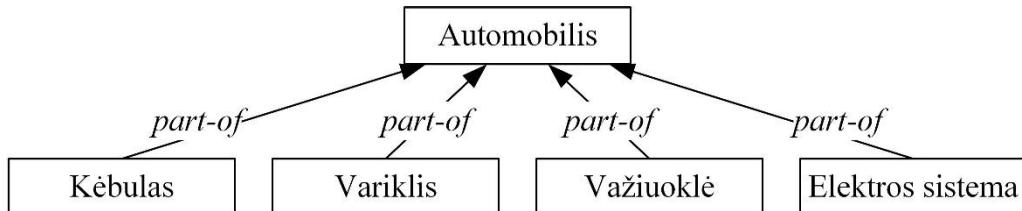
Tai bendrieji, nuo dalykinės srities nepriklausantys ryšiai.

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; Guarino et al. 2009). Konceptualizacija yra abstraktus, supaprastintas pasaulio vaizdas, kurį norima pavaizduoti tam tikru tikslu.

Informatikoje terminu *ontologija* buvo apibrėžta kaip tam tikros dalykinės srities sąvokų visumos specifikavimas išreikštiniu pavidalu (*explicit specification of a conceptualization*) (Gruber 1993). Šis apibrėžimas tiko statinei aplinkai taip vadinanamame kubelių pasaulyje robotikoje. Mes laikomės apibrėžimo, kad ontologija yra loginė teorija, tiksliu, loginė teorija

apie dalykinės srities konceptualizaciją. 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 pažsta pasaulyje)“. Informatikoje ontologija yra suprantama kaip loginė teorija apie dalykinę sričių.

Semantinis grafas 26.6 pav. vaizduoja teiginį, kad automobilio sudėtinės dalys yra kėbulas, variklis, važiuoklė ir elektrinė sistema

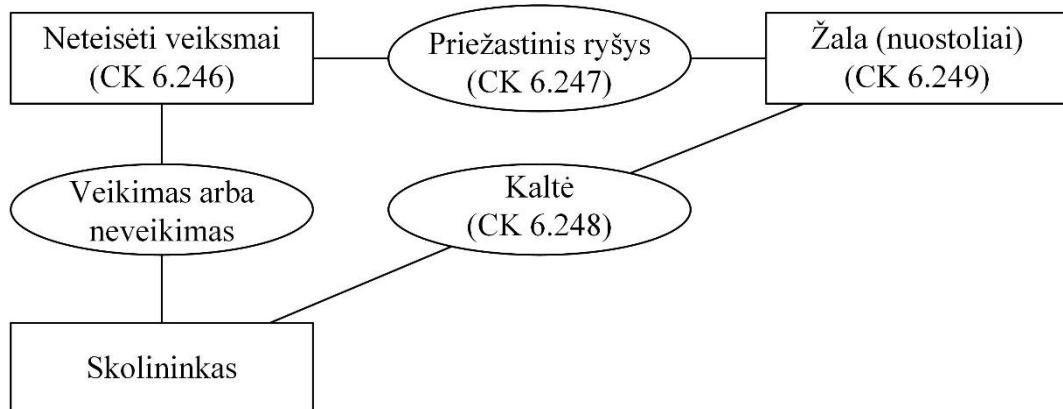


26.6 pav. Automobilių sudėtinės dalys

Semantinis grafas 26.7 pav. vaizduoja sąvokos „civilinė atsakomybė“ elementus. Jis sudarytas remiantis Lietuvos Respublikos civilinio kodekso 6.245 straipsniu. Pavaizduota dvidaliu grafas. Viršūnės 26.7 pav. vaizduoja sąvokas, bet nei viršūnėms, nei briaunoms griežta interpretacija nesuteikta.

Civilinio kodekso 6.245 straipsnis vadinasi „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 sąlyga“;
3. CK 6.247 str. „Priežastinis ryšys“;
4. CK 6.249 str. „Žala ir nuostoliai“ .



26.7 pav. Sąvokos „civilinė atsakomybė“ semantinis grafas, sudarytas remiantis Lietuvos Respublikos civilinio kodekso 6.245 straipsniu

## 27. Programos deklaratyvus vaizdavimas įėjimu–išėjimu

Kalbant apie procedūrinį ir deklaratyvų žinių vaizdavimą, algoritmas programoje siejamas su procedūriniu žinių vaizdavimu. Šis teiginys toliau iliustruojamas pavyzdžiu apie žinias kvadratinės lygties sprendime. Algoritmas yra viena iš žinių vaizdavimo formų.

### 27.1. Procedūrinis žinių vaizdavimas

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:

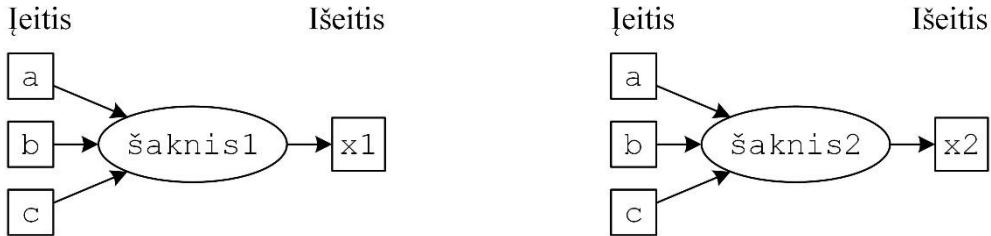
```
read(AA, BB, CC);
call saknis1(AA, BB, CC, XX1);
call saknis2(AA, BB, CC, XX2);
writeln(XX1, XX2)
```

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. Pirmoji ir antroji programa įkūnija skirtingas žinias apie sprendimą. Pirmuoju atveju sprendimo koncepcija yra paprastesnė, o antruoju – vykdymo laikas yra trumpesnis.

```
{ 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
```

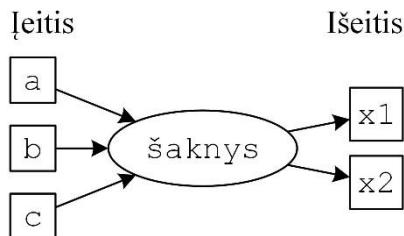
## 27.2. Deklaratyvus žinių vaizdavimas

Kvadratinės lygties šaknų apskaičiavimo procedūrų šaknis1 ir šaknis2 deklaratyvus pavaizdavimas, suprantamas kaip procedūros įeitis ir išeitis, yra pateiktas 27.1 pav. Tekstine forma  $\{a, b, c\} \xrightarrow{\text{šaknis1}} \{x_1\}$  ir  $\{a, b, c\} \xrightarrow{\text{šaknis2}} \{x_2\}$



27.1 pav. Procedūrų šaknis1 ir šaknis2 deklaratyvus pavaizdavimas.  
Vaizduojama procedūrų įeitis ir išeitis

Analogiškai procedūros šaknys deklaratyvus pavaizdavimas parodytas 27.2 pav. Tekstine forma  $\{a, b, c\} \xrightarrow{\text{šaknys}} \{x_1, x_2\}$ .



27.2 pav. Procedūros šaknys deklaratyvus pavaizdavimas

Deklaratyvus vaizdavimas atsako į klausimą „kas“, bet neatsako į klausimą „kaip“. Atsakymas į šį klausimą įkūnytas algoritmo realizacijoje.

Tegu *in* žymi procedūros įeitį (*input*), *out* – išeitį (*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 įeities ir išeities kintamujų (objektų iš tam tikro alfabeto), formaliai  $\langle in(P), out(P) \rangle$ , sutrumpintai  $\langle \bar{in} P, \bar{out} P \rangle$ . Ši pora dar vadinama procedūros P semantika įeities ir išeities terminais, ir žymima  $sem(P)$ . Tokia pora dar vadinama *funkcine priklausomybe* (*functional dependency*, *data dependency*) arba *skaičiavimo santykiai* (rus. *вычислительное отношение*), žr., pvz., (Tyugu 1984). Tokiu būdu:

$$sem(\text{šaknis1}) = in(\text{šaknis1}) \rightarrow out(\text{šaknis1}) = \{a, b, c\} \rightarrow \{x_1\} \quad (27.1)$$

$$sem(\text{šaknis2}) = in(\text{šaknis2}) \rightarrow out(\text{šaknis2}) = \{a, b, c\} \rightarrow \{x_2\} \quad (27.2)$$

$$sem(\text{šaknys}) = in(\text{šaknys}) \rightarrow out(\text{šaknys}) = \{a, b, c\} \rightarrow \{x_1, x_2\} \quad (27.3)$$

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 atskiro procedūros šaknys semantika sutampa:

$$sem(\text{šaknis1}; \text{šaknis2}) = sem(\text{šaknys})$$

Ir sekos šaknis1; šaknis2, ir atskiro procedūros šaknys sutampa tik semantika jeities– išeities terminais. Procedūra šaknys atlieka mažiau veiksmų negu seka šaknis1; šaknis2. Todėl skiriasi kitomis savybėmis.

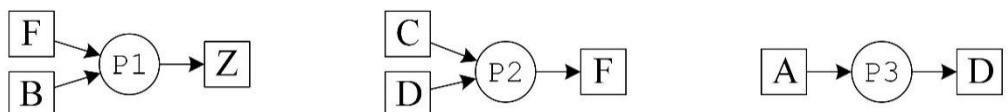
Toliau vėl prisimenama produkcių sistema (18.1), pateikta 18 skyriuje.

$$\begin{aligned} \pi_1: & F, B \rightarrow Z \\ \pi_2: & C, D \rightarrow F \\ \pi_3: & A \rightarrow D \end{aligned} \quad \text{žr. (18.1)}$$

Šios trys produkcijos siejamos su trimis procedūromis (18.3):

<b>procedure</b> P1; Z := f1(B, F)	žr. (18.3)
<b>procedure</b> P2; F := f2(C, D)	
<b>procedure</b> P3; D := f3(A)	

Procedūrų P1, P2 ir P3 semantika yra vaizduojama produkcijomis  $\pi_1$ ,  $\pi_2$  ir  $\pi_3$ . Grafiškai semantika yra vaizduojama semantiniai grafaus 27.3 pav. Procedūros P semantika  $sem(P)$  yra suprantama kaip jėjimo–išėjimo pora (jeitis, išeitis), formaliai  $(in(P), out(P))$ . Taip pat rašoma su rodykle  $in(P) \rightarrow out(P)$ .



27.3 pav. Procedūrų P1, P2 ir P3 semantiniai grafaus

Šiame pavyzdyste alfabetas yra visų išeities ir jeities objektų sąjunga {A, B, C, D, F, Z}. Procedūros P1 jeitis ir išeitis:

$$\begin{aligned} in(P1) &= \{F, B\} \\ out(P1) &= \{Z\} \end{aligned}$$

Analogiškai procedūros P2 jeitis ir išeitis:

$$\begin{aligned} in(P2) &= \{C, D\} \\ out(P2) &= \{F\} \end{aligned}$$

Bei procedūros  $P_3$  įėitis ir išeitis:

$$\begin{aligned}in(P_3) &= \{A\} \\out(P_3) &= \{D\}\end{aligned}$$

Keliamas klausimas: kokia yra procedūrų  $P_1$  ir  $P_2$  nuoseklios kompozicijos  $P_1; P_2$  (paprasčiau kalbant, sekos) įėitis ir išeitis? Tiksliau, kaip išreiškiama nuoseklios kompozicijos  $P_1; P_2$  semantika, jeigu žinoma  $P_1$  ir  $P_2$  semantika?

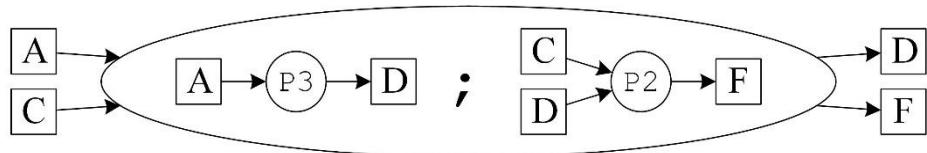
**27.1 apibrėžtis.** Procedūrų  $P_1$  ir  $P_2$  sekos  $P_1; P_2$  įėitis ir išeitis apibrėžiama šitaip:

$$in(P_1; P_2) = in P_1 \cup (in P_2 / out P_1) \quad (27.5)$$

$$out(P_1; P_2) = out P_1 \cup out P_2 \quad (27.6)$$

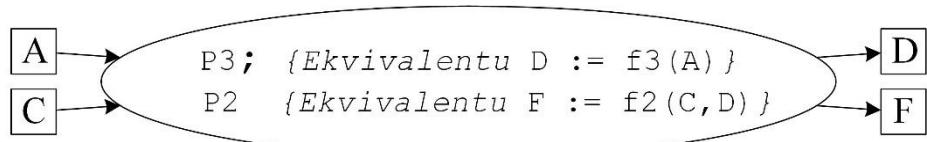
Paprasčiau paaiškinama išeities formulė. Dviejų programų sekos išeitis yra lygi šių programų išeicių sąjungai. Dviejų programų įėitis yra lygi sąjungai, kurią sudaro  $P_1$  įėitis ir ta  $P_2$  dalis, kuri nejineja į  $P_1$  išeitį.

Šias formules iliustruoja procedūrų  $P_3$  ir  $P_2$  sekos įėitis ir išeitis 27.4 pav.



27.4 pav. Procedūrų sekos  $P_3; P_2$  semantinis grafas

Sekos  $P_3; P_2$  semantika iš 27.4 pav. sutrumpintai pakartojama 27.5 pav.

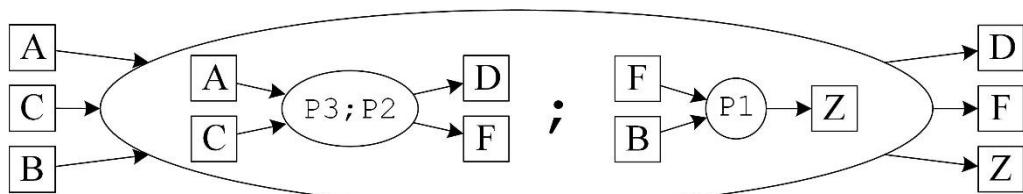


27.5 pav. Procedūrų sekos  $P_3; P_2$  semantika

Trijų procedūrų  $P_3$ ,  $P_2$  ir  $P_1$  nuosekli kompozicija yra apibrėžiama rekurentiškai, t. y. kaip nuosekli kompozicija pirmųjų dviejų, o paskui dar nuosekli kompozicija su trečiąja:

$$P_3; P_2; P_1 = (P_3; P_2); P_1 \quad (27.7)$$

Šią formulę iliustruoja procedūrų  $P_3$ ,  $P_2$  ir  $P_1$  sekos įėitis ir išeitis 27.6 pav.



27.6 pav. Trijų procedūrų sekos  $P_3; P_2; P_1$  išreiškimas per  $P_3; P_2$  ir  $P_1$  seką

Trijų procedūrų sekos  $P_3; P_2; P_1$  įeitis 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 (įeitis, išeitis). Bendru atveju trijų procedūrų sekos įeities 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 įeitis ir išeitis yra:

$$in(P_1; P_2; \dots; P_n) = \bigcup_{i=1}^n (in(P_i) / \bigcup_{j=1}^{i-1} out(P_j)) \quad (27.8)$$

$$out(P_1; P_2; \dots; P_n) = \bigcup_{i=1}^n out(P_i) \quad (27.9)$$

Įrodymas 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 sajungai. Sekos  $P_1; P_2; \dots; P_n$  įeitis (27.8) lygi sajungai, kur iš  $P_i$  įeities atimamos  $P_1, P_2, \dots, P_{i-1}$  išeitys.

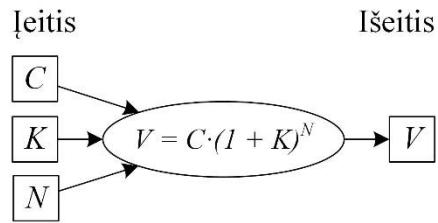
Nuoseklios kompozicijos esmė yra tame, kad anksčiau kviečiamos procedūros gamina rezultatus vėliau kviečiamoms. Kitais žodžiais, ankstesni sekos nariai „maitina“ vėlesnius.

### 27.3. Įeities–išeities vaizdavimo pavyzdys

Paimkime klasikinę kapitalo augimo formulę:

$$V = C \cdot (1 + K)^N$$

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 grafiškai pavaizduota semantiniu tinklu 27.7 pav.

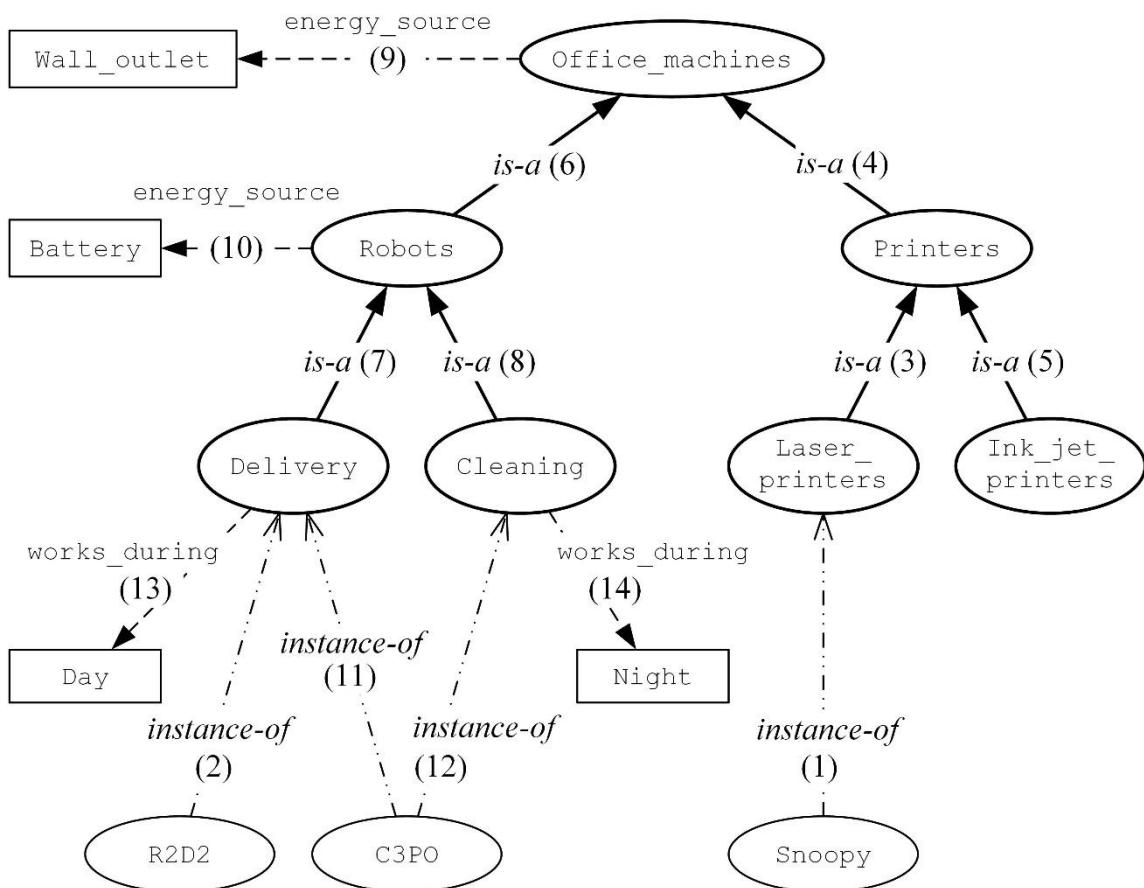


27.7 pav. Procedūros, kuri skaičiuoja kapitalo  $C$  vertę  $V$  po  $N$  metų esant palūkanoms  $K$ , įeities ir išeities pavaizdavimas

## 28. Semantinis tinklas

Semantinio tinklo sąvoką 1909 m. pasiūlė amerikiečių matematikas Charles Peirce. Nėra visuotinai priimto semantinių tinklų užrašymo būdo ir grafinio vaizdavimo kaip, pavyzdžiui, grafas. Semantinis tinklas apibrėžiamas kaip grafas, kuriuo viršūnės vaizduoja sąvokas, o briaunos – ryšius. Sinonimas – semantiniai grafai. Semantinių tinklų semantika nėra iki galio formalizuota. Tai suprantama kaip privalumas, o ne trūkumas.

Toliau remiamės pavyzdžiu iš (Nilsson 1998 p. 308–313) 18.3 poskyrio „Žinių vaizdavimas tinklais“. Jame pateikiama tokia biuro technikos klasifikacija (28.1 pav.).



Žymėjimai:

- > instance-of
- is-a
- > parametras, t. y., atributas, savybė, požymis, slotas

28.1 pav. Semantinis tinklas vaizduoja biuro technikos klasifikaciją (Nilsson 1998, p. 308–313)

Biuro technika `Office_machines` klasifikuojama į robotus `Robots` ir spaudsintuvus `Printers`. Robotai klasifikuojami į pristatymo `Delivery` ir valymo `Cleaning`. Spaudsintuvai 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`.

Ryšiai (žr. numerius 28.1 pav.) vaizduojami teiginiais:

- (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)  $\forall x [Cleaning_robot(x) \Rightarrow works_during(x) = Night]$

Semantiniame grafe yra naudojama daugiskaita, pvz., `Robots`, o logikos formulėse predikatų vardai vienaskaitoje, pvz., `Robot`. Tokiu būdu tarp sąvokų semantiniame grafe ir formulėse yra izomorfizmas.

Remiantis (1), (3), (4) ir (9) yra išvedama, kad Snoopy maitinamas iš rozetės:

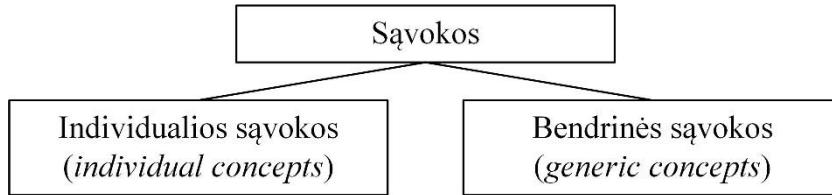
$$\text{Energy\_source}(\text{Snoopy}) = \text{Wall\_outlet}$$

Nevienareikšmiška išsiaiškinti, kada dirba robotas C3PO: dieną ar naktį? Formaliai `works_during(C3PO) = Day` ar `Night`. Remiantis (11) ir (13) C3PO dirba dieną, t. y. `works_during(C3PO) = Day`. Remiantis (12) ir (14) C3PO dirba naktį, t. y. `works_during(C3PO) = Night`. Šiame pavyzdyme žinių bazėje atributui `works_during` siūloma priskirti vieną reikšmę. Tai žinių bazių skirtumas nuo logikos. Logikas sakyta, kad formuliu (1)–(14) aibė prieštaringa. Žinių bazių specialistas pasiūlytų:

1. nustatyti reikšmę pagal nutylėjimą (*by default*);
2. prioritetą tarp (11) ir (12);
3. euristiką pasirinkimui tarp (11) ir (12);
4. modeliuoti atributą `works_during` kaip daugiareikšmį, t. y. tipo `set-of {Day, Night}`.

Semantiniai grafai formalizuojami grafi aprašymo būdais. Yra įprasta naudoti dvidalių grafi (*bipartite graph*). Primenama, kad grafas  $G$  yra pora: viršunių aibė  $V$  ir biaunų aibė  $E$ , formaliai –  $G = \langle V, E \rangle$ , kur  $E \subset V \times V$ . Kaip minėta, semantinio grafo viršūnės vaizduoja sąvokas, o

briaunoms yra suteikiama tam tikra prasmė. Sąvokos klasifikuojamos į individualias, pvz., Snoopy, ir bendrines, pvz. Robots.



28.2 pav. Sąvokos klasifikuojamos į individualias, pvz., Snoopy, ir bendrines, pvz., Robots.

Toliau nagrinėjamas semantinis grafas (28.3 pav.), sudarytas remiantis (Geldenhuys 1999, p. 13). Jis rodo pagrindinius pagrindinius teiginius apie žinių išgavimą ir sudarymą pedagogikoje. Šis semantinis grafas „koduoja“ viršūnėmis ir briaunomis tokius sakinius:

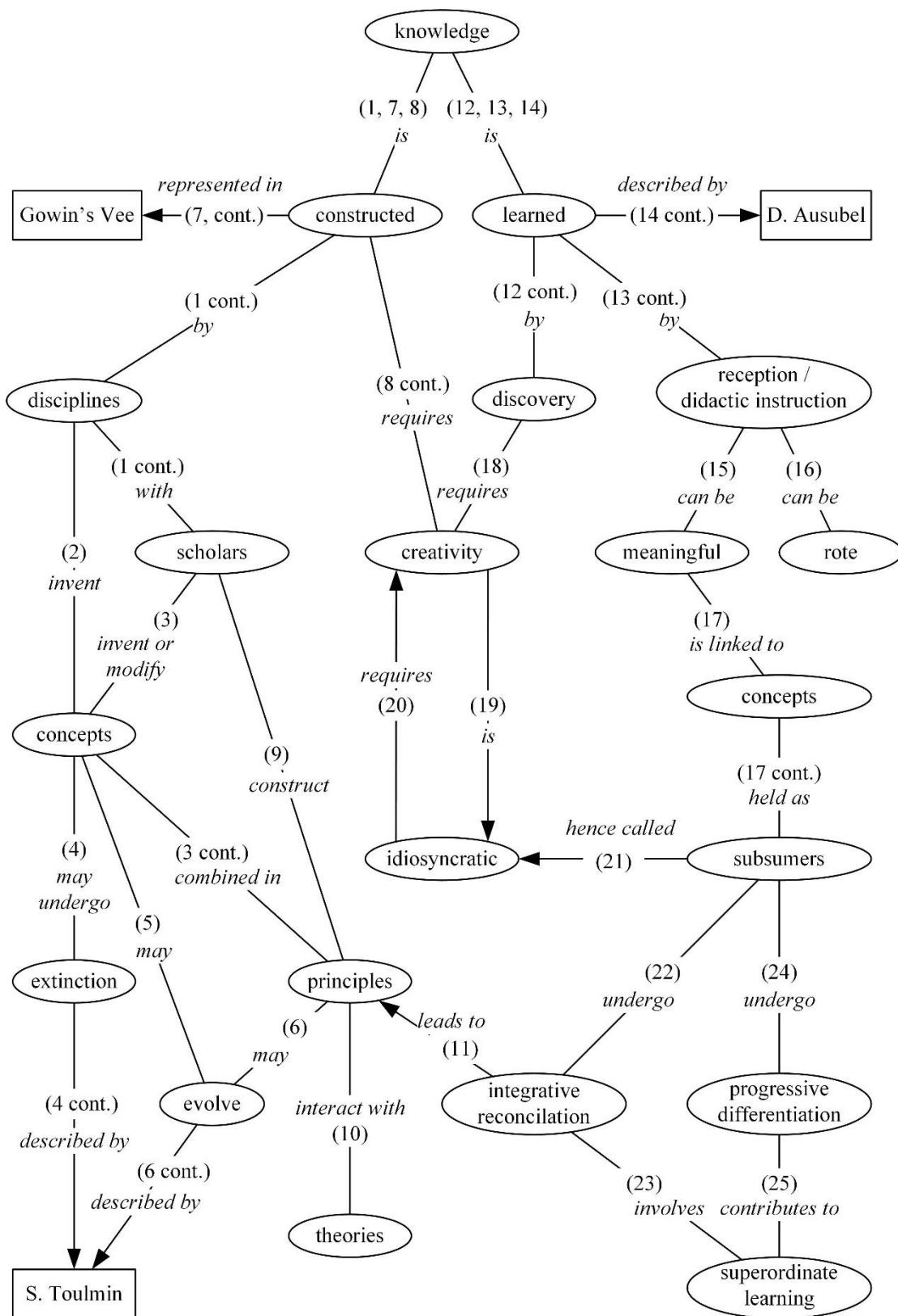
1. Knowledge is constructed by disciplines with scholars.
2. Disciplines invent concepts.
3. Scholars invent or modify concepts combined in principles.
4. Concepts may undergo extinction described by S. Toulmin<sup>4</sup>.
5. Concepts may evolve.
6. Principles may evolve – described by S. Toulmin.
7. Knowledge (is) constructed represented in Govins's Vee<sup>5</sup>.
8. Knowledge (is) constructed requires creativity.
9. Scholars construct principles.
10. Principles interact with theories.
11. Integrative reconciliation (suderinimas) leads to principles.
12. Knowledge is learned by discovery.
13. Knowledge is learned by reception/didactic instruction
14. Knowledge is learned described by D. Ausubel<sup>6</sup>.
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 (išskirtinė).
20. Idiosyncratic requires creativity.
21. Subsumers (priskiriantys asmenys) hence call idiosyncratic (išskirtinai).
22. Subsumers undergo integrative reconciliation.
23. Integrative reconciliation involves superordinate learning.
24. Subsumers undergo progressive differentiation.
25. Progressive differentiation contributes to superordinate learning.

Semantinis grafas 28.3. pav. yra iš pedagogikos srities ir vaizduoja kitokias prasmes, negu semantinis tinklas 28.1 pav.

<sup>4</sup> Stephen Toulmin, argumentavimo metodo autorius, žr. [https://en.wikipedia.org/wiki/Toulmin\\_method](https://en.wikipedia.org/wiki/Toulmin_method).

<sup>5</sup> Žinių įsisavinimo metodas pedagogikoje ir V formos diagrama, žr. J. Novak, D. B Gowin (1984), Learning how to learn. New York: Cambridge University Press.

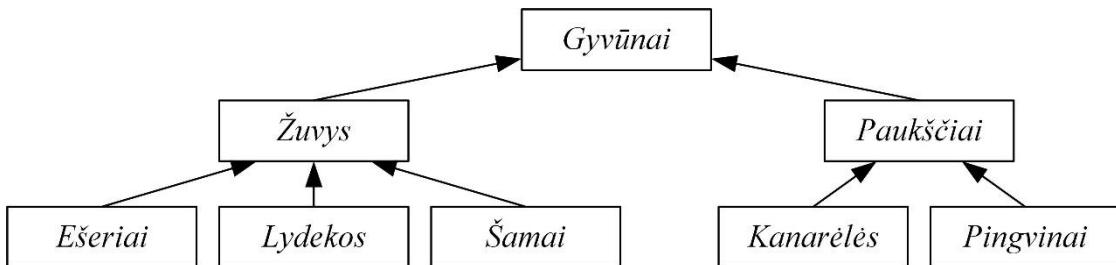
<sup>6</sup> David Ausubel, pedagogikos psichologijos autoritetas, [https://en.wikipedia.org/wiki/David\\_Ausubel](https://en.wikipedia.org/wiki/David_Ausubel).



28.3 pav. Semantinis tinklas pagal (Geldenhuys 1999). Koduojami sakiniai (teiginiai) apie žinių išgavimą ir sudarymą pedagogikoje

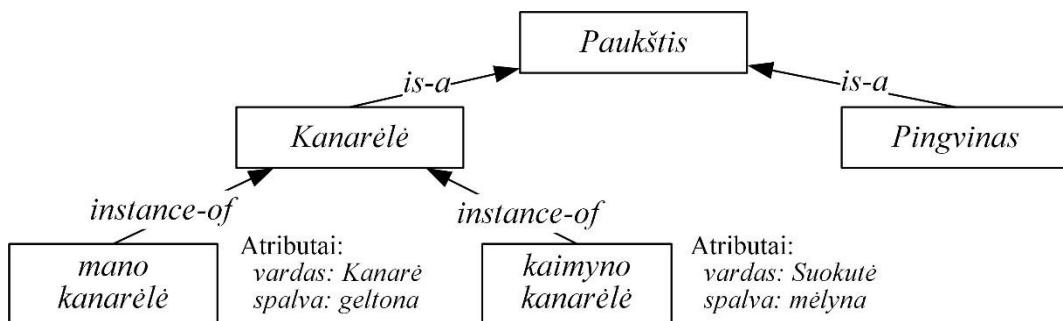
## 29. Bendrinė sąvoka ir individuali sąvoka

Tarp bendrinių sąvokų yra *is-a* yra ryšys (*relationship*). Tarp individualios sąvokos ir bendrinės yra *instance-of* ryšys. 29.1 pav. parodytas labai supaprastintu gyvūnų klasifikavimo pavyzdys. Šiame grafe pavaizduotos klasės ir jų paveldėjimo ryšiai.



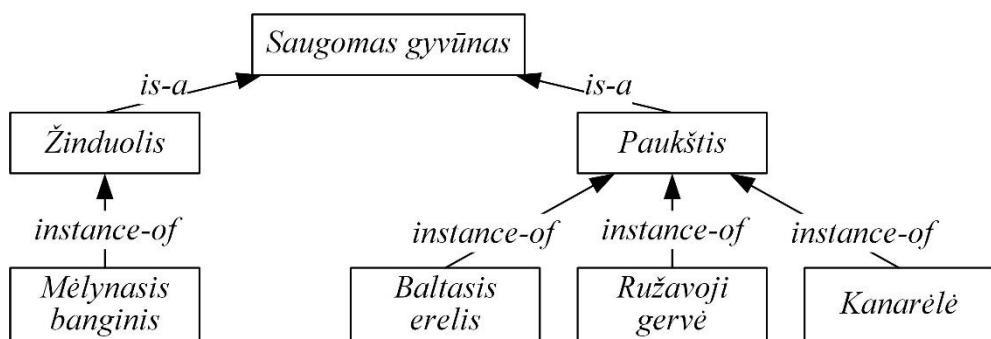
29.1 pav. Gyvūnų klasifikavimo pavyzdys

Klasės vardas paprastai rašomas vienaskaita, pavyzdžiu, *Gyvūnas*, *Žuvis*. Galima iškelti klausimą: kas yra kanarėlė – ar klasė, ar egzempliorius (29.2 pav.)? Pirmiausia yra priimta modeliuoti, kad kanarėlė yra klasė. Klasė *Kanarėlė* yra ryšyje *is-a* su klase *Paukštis*.



29.2 pav. *Kanarėlė* kaip bendrinė sąvoka, o *mano kanarėlė* – individuali

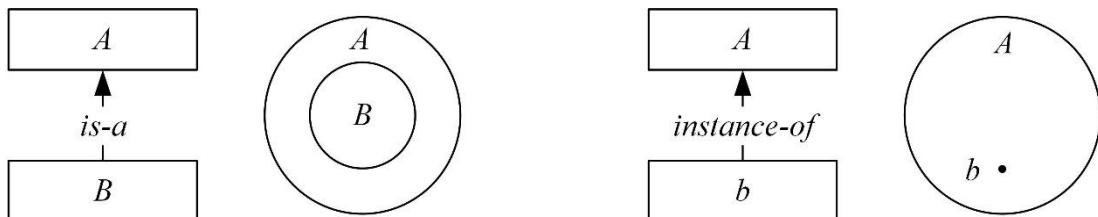
*Kanarėlė* gali žymėti ir individualią sąvoką ryšyje *instance-of* su klase *Paukštis* (29.3 pav.). Ji žymi visas kanarèles pasaulyje kaip nykstančią rūšį. Šiame pavyzdyme medžio lapai žymi individualias sąvokas – kiekviena saugoma rūsis kaip individuas. Kaip matyti, tas pats žodis viename kontekste gali žymėti bendrinę sąvoką, o kitame – individualią.



29.3 pav. *Kanarėlė* žymi individualią sąvoką – ryšyje *instance-of* su *Paukštis*

Buvimas ryšyje gali būti žymimas ir infikcine notacija  $B \text{ is-a } A$ , ir prefiksine  $\text{is-a}(B,A)$ . Abiem atvejais suprantama kaip predikatas – būti ar nebūti ryšyje.

Ryšiui  $B \text{ is-a } A$  suteikiama semantika poaibis–aibė,  $B \subset A$ . Ryšiui  $b \text{ instance-of } A$  suteikiama semantika aibės elementas–aibė,  $b \in A$  (29.4 pav.). Abiem atvejais žmogus gali ištarti vienodai: „ $B$  yra  $A$ “ (“ $B$  is  $A$ ”) ir „ $b$  yra  $A$ “ (“ $b$  is  $A$ ”), nors prasmė skirtina.

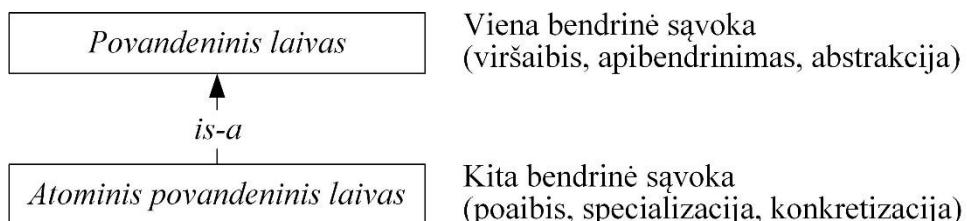


29.4 pav. Interpretacija:  $A$  bendrinė sąvoka,  $B$  bendrinė sąvoka ir  $b$  individuali sąvoka. Semantika aibų kalba:  $B \subset A$  ir  $b \in A$

## 29.1. Ryšio is-a interpretacijos

Ryšys  $\text{is-a}$  gali turėti tokias interpretacijas (Brachman 1988).

**1. Poaibis ir viršaibis** (*subset/superset*). Kai viršūnės vaizduoja aibes, tai briauna tarp viršūnių žymi poaibio santykį. Pavyzdžiu, teiginys „Atominis povandeninis laivas yra povandeninis laivas“ interpretuojamas  $\text{Atominis povandeninis laivas} \subset \text{Povandeninis laivas}$  (29.5 pav.). Kitaip tariant, „Kiekvienai esybei  $x$ , jeigu  $x$  priklauso atominiams povandeniniams laivams, tai  $x$  priklauso ir povandeniniams laivams“.



29.5 pav. Ryšys  $\text{is-a}$  kaip poaibis-viršaibis

**2. Apibendrinimas ir specializacija** (*generalization/specialization*). Apibendrinimas, kaip ryšys tarp predikatų reiškia implikaciją. Pavyzdžiu, tegu sąvoka *Povandeninis laivas* yra sąvokos *Atominis povandeninis laivas* apibendrinimas. Tada  $\text{is-a}$  reiškia „kiekvienai esybei  $x$ , jeigu *Atominis\_povandeninis\_laivas*( $x$ ), tai *Povandeninis\_laivas*( $x$ )“.

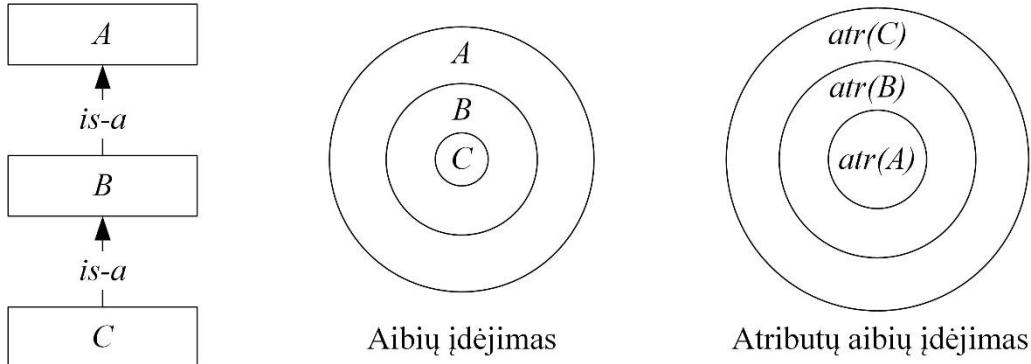
Apibendrinimas ir specializacija yra suprantami, kaip, pvz., objektiniame programavime. Sąvoka  $B$  yra sąvokos  $A$  *specializacija* ir  $A$  yra  $B$  *apibendrinimas*. Sąvokas modeliuojant predikatais yra teisinga  $\forall x B(x) \Rightarrow A(x)$ , kur  $\Rightarrow$  žymi implikaciją logikoje.

Kai sąvokos  $A$  ir  $B$  apibrėžiamos atributų rinkiniai, 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  $A$ , tačiau daugiau atributų:

$$B \subset A \tag{29.1}$$

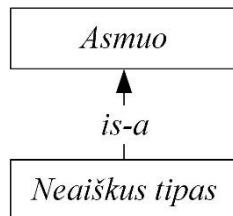
$$\text{atr}(B) \supset \text{atr}(A) \tag{29.2}$$

Čia *atr(savoka)* žymi atributų aibę. Pastebime, kad atributų įdėjimas (29.2) rodomas priešinga kryptimi, negu aibų įdėjimas (29.1): labiau specifinė savoka turi mažiau elementų, bet daugiau atributų (29.6 pav.). Pavyzdžiu, savoka A turi atributus E ir F, o savoka B be E ir F dar ir papildomą atributą G, kitaip tariant,  $E \& F \Rightarrow A$  ir  $E \& F \& G \Rightarrow B$ .



29.6 pav. Tipinė *is-a* interpretacija yra apibendrinimas (žiūrint iš apačios į viršų – rodyklės kryptimi) ir specializacija (žiūrint iš viršaus į apačią). Tai tipinis klasifikavimo ryšys

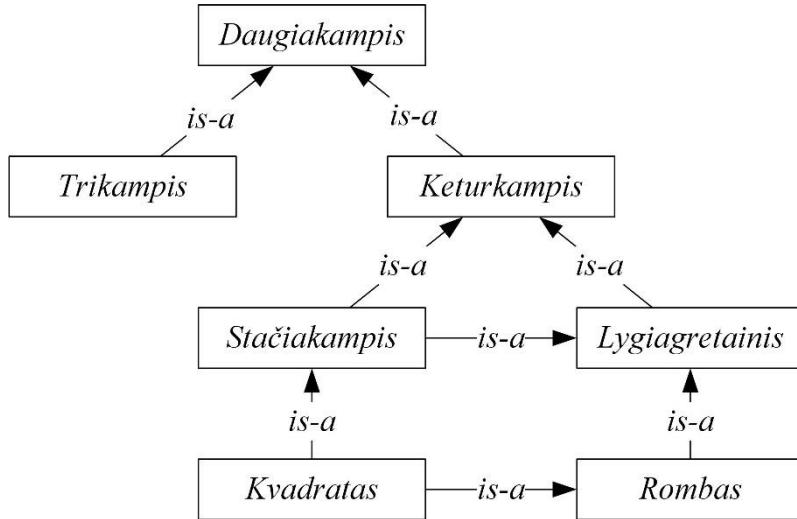
**3. AKO (a-kind-of).** Šią interpretaciją iliustruoja savoka „neaiškus tipas“ (*stranger*). Sakinys „Neaiškus tipas yra asmuo“ žymi ryšį *Neaiškus\_tipas is-a Asmuo* su interpretacija AKO (29.7 pav.). Šioje interpretacijoje savokai nesuteikiamas klasės vaidmuo, o įvardijama buitiškai: „neaiškus tipas“, bet paliekama galimybė paveldėti.



29.7 pav. Sakiniui „Neaiškus tipas yra asmuo“ suteikiamą interpretaciją AKO

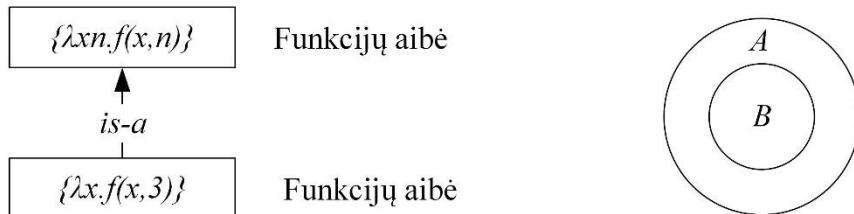
Pavyzdžiu, į degalinę užsukusį asmenį kasininkas iš pradžių gali priimti kaip neaišką tipą. Iš pradžių nėra žinomas jo apsilankymo tikslas: ar apsipirkti, ar apsižvalgyti, ar vogti. Jeigu išsirinkęs prekę, jis paduoda kreditinę kortelę, tai priimamas kaip pirkėjas.

**4. Koncepcinis įdėjimas (conceptual containment).** Pavyzdžiu, „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 parodyti 29.8 pav. Koncepcinis įdėjimas gaunamas žiūrint iš apačios į viršų ir koncepcinis apėmimas – žiūrint iš viršaus į apačią.



29.8 pav. Ryšiai *is-a* tarp geometrinių figūrų

Kitas pavyzdys yra  $\lambda$ -abstraksiacija, naudojama funkcinio programavimo teorijoje. Imkime dviejų argumentų funkcijas  $\{\lambda xn.f(x,n)\}$ . Šios funkcijų aibės specializacija yra vieno argumento funkcijų aibė  $\{\lambda x.f(x,3)\}$ , kuri gaunama sukonkretnius  $n=3$  (29.9 pav.).



29.10 pav. Ryšys *is-a* tarp dviejų aibių, kurių elementai yra funkcijos

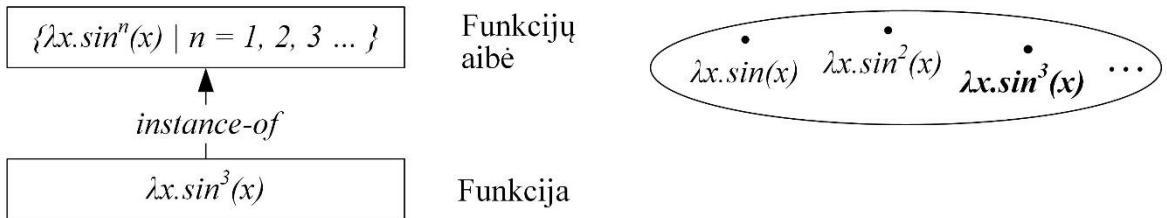
Toliau 29.10 pav. rodomas *instance-of* (o ne *is-a*) ryšys tarp vienos funkcijos  $\lambda x.\sin^3(x)$  ir funkcijų aibės (šeimos)  $\{\lambda x.\sin^n(x) / n=1, 2, 3 \dots\}$ . Beje, pastarają funkcijų aibę reikia skirti nuo vienos funkcijos  $\lambda n.\lambda x.\sin^n(x)$ , kuri priklauso nuo parametru  $n$ . Pastaroji funkcija atvaizduoja  $n$  į funkciją  $\lambda x.\sin^n(x)$ .

Funkcija turi apibrėžimo sritį ir kitimo sritį. Tai aibės. Funkcija (sinonimas „atvaizdavimas“) yra trejetas: apibrėžimo sritis, kitimo sritis, ir atvaizdavimo taisykla.

I fiksuočių dviejų argumentų  $x$  ir  $n$  funkciją  $f$  galima žiūrėti trejopai:

1. Tai dviejų argumentų funkcija  $\lambda xn.f(x,n)$ . 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  $n \rightarrow \lambda x.f(x,n)$ . Čia argumentui  $n$  priskiriama reikšmė  $\lambda x.f(x,n)$ , kuri savo ruožtu yra funkcija nuo  $x$ .

Skirtingai nuo 29.9 pav. su *is-a* ryšiu, 29.10 pav. rodo *instance-of* ryšį. 29.9 pav. vaizduoja *is-a*, nes  $f$  žymi bet kokią funkciją, kaip *Drambllys* žymi drambllio sąvoką. 29.10 pav. vaizduoja *instance-of*, nes  $f$  sukonkretninta.



29.10 pav. Ryšys *instance-of* tarp funkcijos  $\lambda x.sin^3(x)$  ir funkcijų aibės  $\{\lambda x.sin(x), \lambda x.sin^2(x), \lambda x.sin^3(x), \dots\}$

Dviejų argumentų funkcija  $\lambda xn.sin^n(x)$  programuojama kėlimo laipsniu operacija. Programuojant specializuotą funkciją  $\lambda x.sin^3(x)$  kėlimas laipsniu yra keičiamas sandauga  $\lambda x.sin(x).sin(x).sin(x)$ . Pavadinkime pastarąsias funkcijas FSIN ir FSIN3. Vadovaujantis trečiuoju požiūriu programuojama funkcija  $\lambda n.\lambda x.f(x,n)$ , žemiau pavadinta FEN.

```

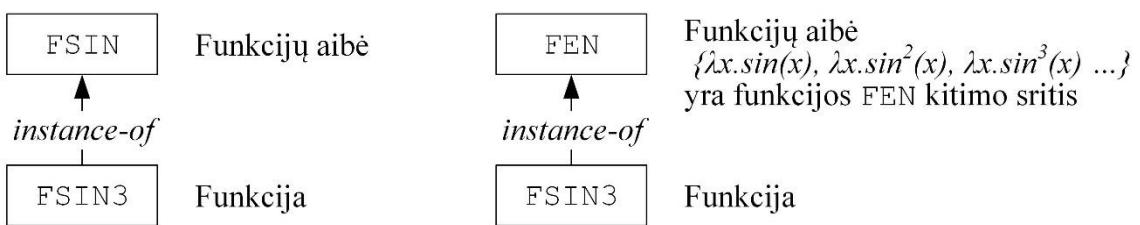
procedure FSIN(x: real; n: integer): real;
begin
    return sin(x)**n;
end

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 29.11 pav. kairėje. Tokiu modeliavimo būdu į FSIN žiūrima kaip į funkcijų šeimą nuo  $n$ , o ne kaip į funkciją nuo dviejų argumentų  $x$  ir  $n$ .



29.11 pav. Funkcija FSIN ryšyje *instance-of* ir su FSIN, ir su FEN

Į FSIN3 galima žiūrėti kaip į esančią *instance-of* ryšyje su FEN (29.11 pav. dešinėje). Čia vadovaujamasi trečiuoju požiūriu (žr. aukščiau ir 29.10 pav.).

Kitas pavyzdys iliustruoja ryšį tarp polinominį funkcijų. Tegu  $P$  yra funkcija  $n$ -tojo laipsnio polinomas  $\lambda xn.x^n+x^{n-1}+x^{n-2}+\dots+x^2+x+1$ :

```
procedure P(x: real; n: integer): real; { n ≥ 1 }
begin
  var rez := 1;
  for i := 1 to n do
    rez := rez * x + 1;
  return rez;
end
```

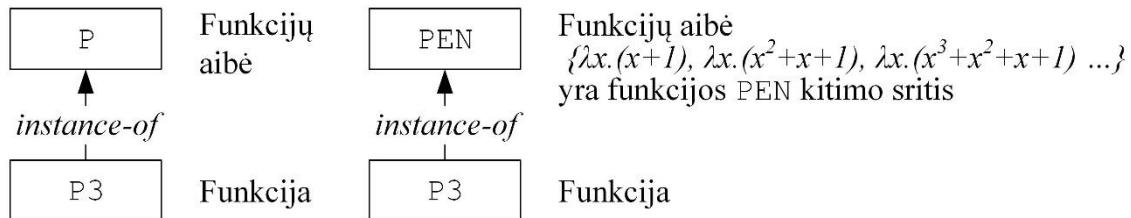
Tegu P3 yra P su įstatytu  $n=3$  ir programuojant be ciklo:

```
procedure P3(x: real): real;
begin
  return x*x*x + x*x + x + 1;
end
```

Tegu PEN atitinka FEN iš ankstesnio pavyzdžio.

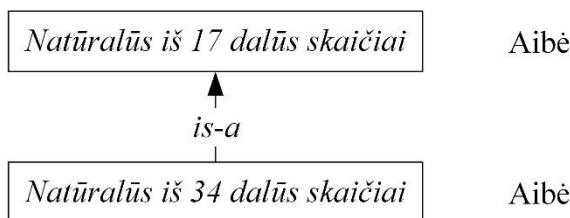
```
procedure PEN(n: integer): procedure (x: real) real;
begin
  var rez := 1;
  for i := 1 to n do
    rez := rez * x + 1;
  return rez;
end
```

Požiūris į P3 ir P kaip esančias *instance-of* ryšyje (o ne *is-a* ryšyje) yra modeliuojamas 29.12 pav. Šitaip į P žiūrima kaip į funkcijų šeimą nuo  $n$ , o ne kaip į funkciją nuo dviejų argumentų x ir n.



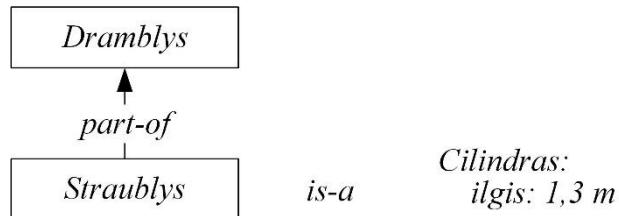
29.12 pav. Funkcija P3 ryšyje *instance-of*

Kitas 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 ...} (29.13 pav.). Skaičiai dalūs iš 17 nėra įvardinti atskira savoka kaip, pavyzdžiu, lyginiai ir nelyginiai.



29.13 pav. Ryšys *is-a* 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* (29.14 pav.). Tokia *is-a* interpretacija nėra klasifikavimo ryšys.



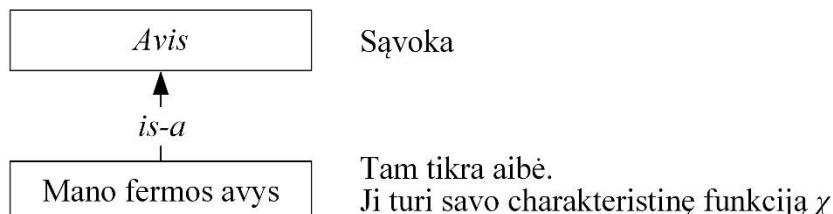
29.14 pav. Dramblio straublys yra (*is-a*) cilindras

**6. Aibė ir jos charakteristinė funkcija.** Tai nėra klasifikavimo ryšys. Tai, pavyzdžiui, ryšys tarp tam tikros dramblių aibės ir *Dramblis* kaip sąvokos. Šitaip susiejama šios aibės charakteristinė funkcija su sąvoka. Pavyzdžiui, tegu kalbama apie avis savo fermoje. Tada turimas omenyje ryšys tarp visų avių fermoje ir avies sąvokos (29.15 pav.). Turima omenyje, kad laikomos ne karvės. Atitinkama charakteristinė funkcija nusako, pavyzdžiui, kad kaimyno avis nr. 123 nepriklauso mano fermos avims:  $\chi_{\text{manoFermosAvys}}(\text{kaimynoAvis123}) = \text{false}$ .

Aibės *A* charakteristinė funkcija  $\chi$  (tariama chi) apibrėžiama šitaip:

$$\begin{aligned} \chi_A(x) &= \text{true}, \text{ jeigu elementas } x \text{ priklauso aibei } A \\ &\quad \text{false priešingu atveju} \end{aligned}$$

Pavyzdžiui:  $\chi_{\text{dalūsIš17}}(34) = \text{true}$ ,  $\chi_{\text{dalūsIš17}}(3) = \text{false}$ .

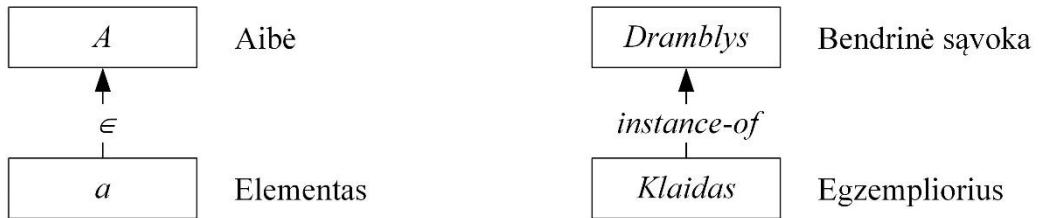


29.15 pav. *is-a* kaip ryšys tarp avių mano fermoje aibės ir avies sąvokos

## 29.2. Ryšio instance-of interpretacijos

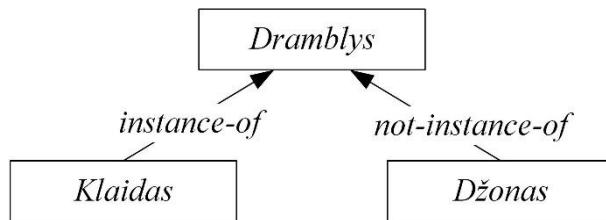
Ryšys *instance-of* gali turėti tokias interpretacijas.

**1. Elemento ir aibės ryšys;  $a \in A$**  (29.16 pav.). Pavyzdžiui, „Klaidas yra dramblis“ reiškia „Klaidas yra dramblių aibės elementas“.



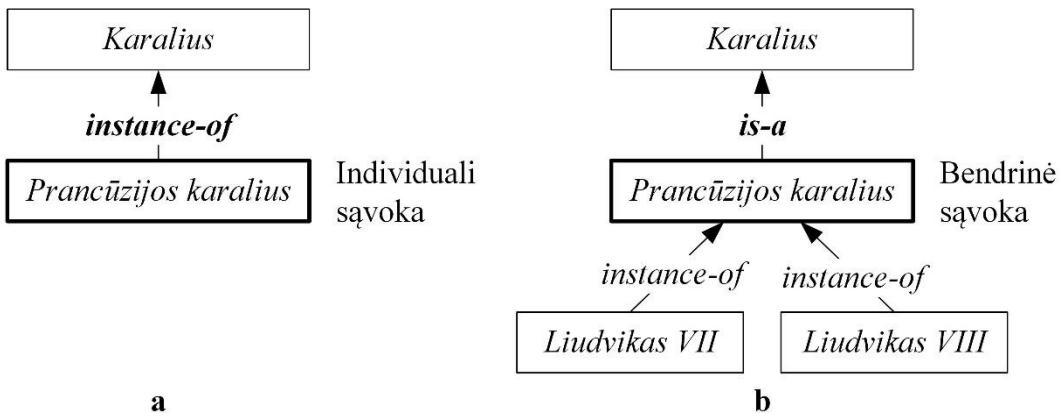
29.16 pav.  $a \in A$  kaip *instance-of* ryšys. Tai išreiškia „Klaidas yra dramblys“

**2. Predikatas.** Pavyzdžiui, „Klaidas yra dramblys“ implikuoja predikatą *Dramblys(Klaidas)=true*. „Džonas nėra dramblys“ reiškia *Dramblys(Džonas)=false* (29.17 pav.). „Klaidas nėra povandeninis laivas“ reiškia *Povandeninis\_laivas(Klaidas)=false*. Aptariamas predikatas atitinka aibės charakterinę funkciją  $\chi$ .



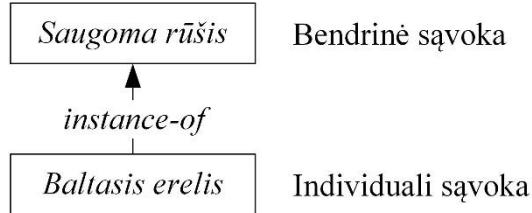
29.17 pav. „Klaidas yra dramblys“ reiškia *Dramblys(Klaidas)=true*. „Džonas nėra dramblys“ reiškia *Dramblys(Džonas)=false*

**3. Konceptinis įdėjimas (conceptual containment).** Pavyzdžiui, teiginys „Prancūzijos karalius yra karalius“ modeliuojamas *instance-of* ryšiu (29.18 a pav.). Tačiau modeliavimas priklauso nuo konteksto. Jeigu kalbama apie Liudviką VII ir Liudviką VIII, tai *Prancūzijos karalius* modeliuojamas kaip bendrinė sąvoka (29.18 b pav.).



29.18 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 sakinio „Baltasis erelis yra saugoma rūšis“ ištraukiamą individualią savoka *Baltasis erelis*, bendrinę savoką *Saugoma rūšis* ir abstrakciją reiškiantis ryšys *instance-of* (29.19 pav.).



29.19 pav. „Baltasis erelis yra saugoma rūšis“ interpretacija *instance-of*

## 30. Klasifikavimo vaizdavimas

Pademonšime klasifikavimą specifineje dalykinėje srityje – mokesčių teisėje. Vargu ar informatikas, kuriantis mokesčių administravimo informacinię sistemą gali perprasti visų mokesčių esmę. Mokesčių administravimo įstatymo (MAĮ) 13 straipsnis nustato mokesčių sąrašą Lietuvoje, kurį pateikiame nedaug sutrumpintą iliustruodami finansų reglamentavimą:

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. paveldimo turto mokestis;
12. privalomojo sveikatos draudimo įmokos;
13. įmokos į Garantinį fondą;
14. valstybės rinkliava;
15. loterijų ir azartinių lošimų mokestis;
16. mokesčiai už pramoninės nuosavybės objektų registravimą;
17. pelno mokestis (tik juridiniams asmenims);
18. valstybinio socialinio draudimo įmokos;
19. pertekliaus mokestis cukraus sektoriuje;
20. gamybos mokestis cukraus sektoriuje;
21. muitai;
22. atskaitymai nuo pajamų pagal Lietuvos Respublikos miškų įstatymą;
23. mokestis už valstybės turto naudojimą patikėjimo teise;
24. socialinis mokestis;
25. cukraus pramonės restruktūrizavimo laikinasis mokestis;
26. papildomos baltojo cukraus gamybos kvotos ir pridėtinės izogliukozės gamybos kvotos vienkartinio išsipirkimo mokestis.

Ar šie mokesčiai klasifikuojami į kokias nors nepersikertančias aibes? Mokesčių teisės knygoje 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 (užmokesčis) už valstybinę žemę bei valstybinio fondo vandens telkinius ir t. t.“ (Marcijonas, Sudavičius 2003, p. 18). Iš šios citatos išplaukia, kad mokesčiai negali būti suskaidyti į dvi nepersikertančias aibes, pirma, juridinių asmenų mokesčiai, ir, antra, fizinių asmenų mokesčiai. Persikirtimas įvardintas „bendri mokesčiai“.

Gaunamas toks aibės „mokesčiai“ suskaidymas į tris poaibius, kuriuos koduojame dviem būtais JA ir FA.

1. Juridinių asmenų mokesčiai. Koduojame:

`mokesčis.mokėtojo_tipas = {'juridinis_asmuo'}, t. y. JA:=1; FA=0`

2. Fizinių asmenų mokesčiai. Koduojame:

`mokesčis.mokėtojo_tipas = {'fizinis_asmuo'}, t. y. JA=0; FA=1`

3. Bendri mokesčiai. Koduojame:

`mokestis.moketojo_tipas = {'juridinis_asmuo', 'fizinis_asmuo'}`,  
 t. y. JA=1; FA=1, kur **type** `mokestis.moketojo_tipas` = **subset\_of**  
`{'juridinis_asmuo', 'fizinis_asmuo'}`. 2 bitai: JA ir FA.

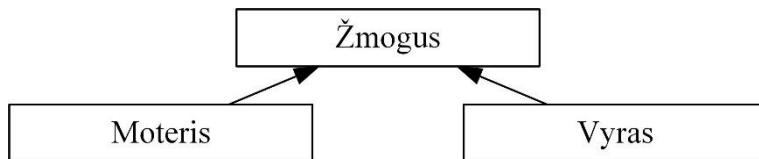
### 30.1. Algebrinis požiūris į klasifikavimą

Tegu turima aibė  $A = \{a_1, a_2, \dots\}$ .

**30.1 apibrėžtis.** Aibės  $A$  suskaidymu (*partition*)  $A = \{A_1, A_2, \dots, A_m\}$  yra vadinama aibės  $A$  poaibių aibė tokia, kad:

- 1) poaibiai nesikerta (*disjoint*):  $A_i \cap A_j = \emptyset$ , kai  $i \neq j$ ;
- 2) poaibių aibė yra išseansi (*exhaustive*):  $\forall a \in A, \exists i \in \{1, 2, \dots, m\}$  tokis, kad  $a \in A_i$ . T. y. kiekvienam aibės  $A$  elementui  $a$  egzistuoja tokis poaibis  $A_i \subset A$ , kad  $a$  yra jo elementas.

Paimkime kokią nors žmonių aibę  $\text{Žmogus}$  ir jos suskaidymą pagal lyti  $\text{Žmogus} = \{\text{Moteris}, \text{Vyras}\}$ . Turime 2 poaibius (klases), iš kuriuos žiūrėsime kaip į elementus (faktoraibės elementus). Skaidoma pagal atributą žmogus.lytis. Iš suskaidymo elementą *Moteris* (kaip aibės  $\text{Žmogus}$  poaibį) patenka tokie elementai  $x \in \text{Žmogus}$  su atributo lytis reikšme 'moteris', t. y.  $x.lytis = \text{'moteris'}$ . Iš suskaidymo elementą *Vyras* patenka elementai su atributo lytis reikšme 'vyras', t. y.  $x.lytis = \text{'vyras'}$  (30.1 pav.).



**30.1 pav.** Klasifikavimas kaip suskaidymas: aibė  $\text{Žmogus} = \{\text{Moteris}, \text{Vyras}\}$ , bet faktoraibė  $\text{Žmogus} = \{\text{Moteris}, \text{Vyras}\}$

Aibė į nepersikertančius poaibius skaidoma pagal kokį nors *ekvivalentumo* santykį.

**30.2 apibrėžtis.** 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  $x \in A$ .

**30.3 apibrėžtis.** Aibės  $A$  suskaidymas pagal joje apibrėžtą *ekvivalentumo* santykį  $\sim$  vadinamas faktoraibe ir žymimas  $A/\sim$ .

I vieną suskaidymo elementą (jis yra aibės  $A$  poaibis) patenka visi tarpusavyje *ekvivalentūs* aibės  $A$  elementai. Aibės faktorizavimas į faktoraibę yra vienas iš būdų paimti jos suskaidymą.

**30.4 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:

$$A/\sim = \{ \{0, 3, 6, 9, \dots\}, \{1, 4, 7, 10, \dots\}, \{2, 5, 8, 11, \dots\} \}$$

Pabrėžiame, kad suskaidymo elementai savo ruožtu yra aibės, t. y. aibės  $A$  poaibiai.

## 30.2. Santykio sąvoka

Santykis (*relation, отношение*) yra griežta matematinė sąvoka.

**30.5 apibrėžtis.** Binarinis santykis  $R$  tarp aibės  $A$  elementų yra dekartinės sandaugos  $A \times A$  poaibis, t. y.  $R \subset A \times A$ . Binarinio santykio elementai yra poros  $(a, b)$ , kur  $a, b \in A$ .

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 lentelė. Pavyzdžiu, 30.4 pavyzdje apibrėžtas ekvivalentumo santykis  $x \sim y$ , jeigu dalijant iš 3 jų liekanos yra lygios, t. y.  $(x \bmod 3) = (y \bmod 3)$ , užrašomas 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 parodyta 30.2 lentelėje. Joje trys stulpeliai vaizduoja natūraliųjų skaičių suskaidymą į tris poaibius:  $\{0, 3, 6, 9, \dots\}$ ,  $\{1, 4, 7, 10, \dots\}$  ir  $\{2, 5, 8, 11, \dots\}$ . Šiame pavyzdyme lentelė yra begalinė.

Aibė $\{0, 3, 6, 9, \dots\}$ , t. y. $(x \bmod 3) =$ $(y \bmod 3) = 0$	Aibė $\{1, 4, 7, 10, \dots\}$ t. y. $(x \bmod 3) =$ $(y \bmod 3) = 1$	Aibė $\{2, 5, 8, 11, \dots\}$ t. y. $(x \bmod 3) =$ $(y \bmod 3) = 2$
$0 \sim 0$	$1 \sim 1$	$2 \sim 2$
$0 \sim 3$	$1 \sim 4$	$2 \sim 5$
$0 \sim 6$	$1 \sim 7$	$2 \sim 8$
$0 \sim 9$	$1 \sim 10$	$2 \sim 11$
...	...	...
$3 \sim 0$	$4 \sim 1$	$5 \sim 2$
$3 \sim 3$	$4 \sim 4$	$5 \sim 5$
$3 \sim 6$	$4 \sim 7$	$5 \sim 8$
$3 \sim 9$	$4 \sim 10$	$5 \sim 11$
...	...	...
$6 \sim 0$	$7 \sim 1$	$8 \sim 2$
$6 \sim 3$	$7 \sim 4$	$8 \sim 5$
$6 \sim 6$	$7 \sim 7$	$8 \sim 8$
$6 \sim 9$	$7 \sim 10$	$8 \sim 11$
...	...	...
$9 \sim 0$	$10 \sim 1$	$11 \sim 2$
$9 \sim 3$	$10 \sim 4$	$11 \sim 5$
$9 \sim 6$	$10 \sim 7$	$11 \sim 8$
$9 \sim 9$	$10 \sim 10$	$11 \sim 11$
...	...	...

**30.2 lentelė.** Natūraliųjų skaičių aibės faktorizavimas pagal ekvivalentumo santykį  $\sim$ , kur  $x \sim y$ , jeigu dalijant iš 3 jų liekanos yra lygios, t. y.  $(x \bmod 3) = (y \bmod 3)$

Žodžiai „santykis“ ir „ryšys“ gali turėti nedidelę skirtingą prasmę priklausomai nuo konteksto, matematikos ir informatikos. Žodis „santykis“ paprastai suprantamas kaip

ekstensionalas, t. y. ekstensionalinis aibės uždavimas, o „ryšys“ – kaip intensionalas. Ryšiui tarp  $x$  ir  $y$  priskiriama ir kitų savybių, pvz.,  $1:n$ ,  $m:n$ , o ne tik buvimas santykyje. Elementai  $x$  ir  $y$  yra santykyje  $R$  tada ir tik tada, kai teisingas tam tikras predikatas  $R(x, y)$ .

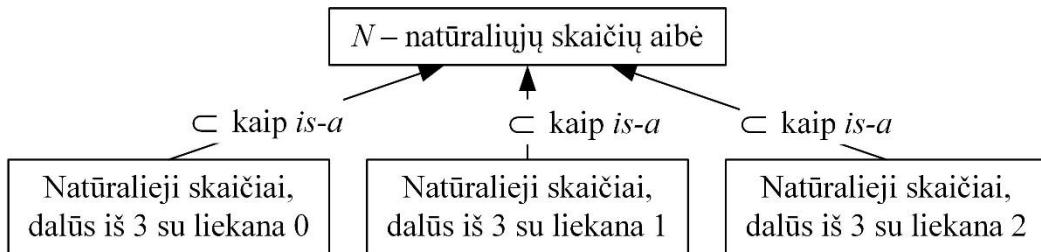
Įrodysime, kad 30.4 pavyzdyje apibrėžtas ekvivalentumo santykis  $x \sim y$ , yra ekvivalentumo santykis. Įrodyma pagal apibrėžtį. Pirma įrodomas tranzityvumas.

$x \sim y$ , kai  $(x \text{ mod } 3) = (y \text{ mod } 3)$ . 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  $(x \text{ mod } 3) = (z \text{ mod } 3)$ . Iš čia sekা  $x \sim z$ . Tai ir reikėjo įrodyti.

Toliau įrodomas refleksyvumas.  $x \sim x$ , nes  $(x \text{ mod } 3) = (x \text{ mod } 3)$ .

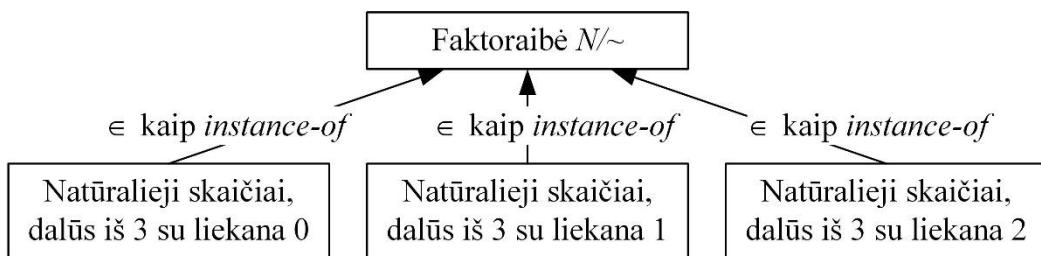
Kadangi ekvivalentumas yra santykis, tai jis suprantamas kaip tam tikras predikatas  $\text{equiv}(x, y)$ .  $\text{equiv}(x, y) = \text{true}$ , kai yra tenkinama kažkokia sąlyga. Pavyzdžiui,  $x$  dalinant iš trijų ir  $y$  dalinant iš trijų gaunama ta pati liekana.  $\text{equiv}(x, y) = \text{false}$ , kai ekvivalentumo sąlyga nėra patenkinta, pavyzdžiui, liekanos skirtingos.

Paaškinsime faktoraibės pavaizdavimą. Aibės  $N = \{0, 1, 2, 3, 4, \dots\}$ , suskaidymas pagal santykį  $\sim$  yra  $N/\sim = \{A_1, A_2, A_3\}$ . Čia  $A_1 = \{0, 3, 6, 9, \dots\}$ ,  $A_2 = \{1, 4, 7, 10, \dots\}$ ,  $A_3 = \{2, 5, 8, 11, \dots\}$ . Iš  $A_1$ ,  $A_2$  ir  $A_3$  galima žiūrėti kaip į poaibius:  $A_1 \subset N$ ,  $A_2 \subset N$ ,  $A_3 \subset N$  (30.3 pav.).



**30.3 pav.** Natūraliųjų skaičių aibės suskaidymas pagal ekvivalentumo santykį  $\sim$ , kur  $x \sim y$ , kai  $(x \text{ mod } 3) = (y \text{ mod } 3)$

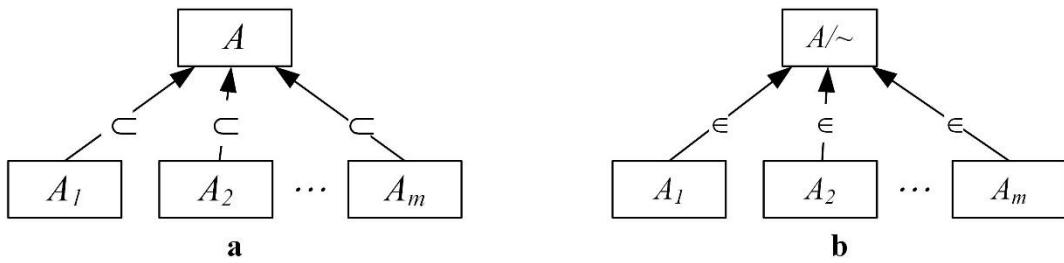
Savo ruožtu  $A_1$ ,  $A_2$  ir  $A_3$  yra faktoraibės  $N/\sim$  elementai, t. y.  $A_1 \in N/\sim$ ,  $A_2 \in N/\sim$ ,  $A_3 \in N/\sim$  (30.4 pav.).



**30.4 pav.** Natūraliųjų skaičių faktoraibės elementai

Natūraliųjų skaičių aibę galima suskaidyti pagal įvairius įvairius ekvivalentiškumo santykius, pavyzdžiui, pagal liekaną dalijant iš kokio nors pirminio skaičiaus  $p$ . Pavyzdžiui, kai  $p=5$ , tai faktoraibėje yra 5 elementai:  $A_1 = \{0, 5, 10, 15, \dots\}$ ,  $A_2 = \{1, 6, 11, 16, \dots\}$ ,  $A_3 = \{2, 7, 12, 17, \dots\}$ ,  $A_4 = \{3, 8, 13, 18, \dots\}$ ,  $A_5 = \{4, 9, 14, 19, \dots\}$ .

Bendru atveju faktoraibėje yra  $m$ ,  $m \geq 1$ , elementų:  $A/\sim = \{A_1, A_2, A_3, \dots, A_m\}$ . Čia iš  $A_i$  galima žiūrėti kaip į poaibį, aibės  $A$  poaibį  $A_i \subset A$  (30.5 a pav.) ir kaip į elementą, faktoraibės  $A/\sim$  elementą  $A_i \in A/\sim$  (30.5 b pav.).



**30.5 pav.** Du požiūriai į suskaidymą  $\{A_1, A_2, \dots, A_m\}$  ir grafiniai pavaizdavimai:

- a) į  $A_i$  žiūrima kaip į poaibį  $A_i \subset A$ , ir todėl viršunėje aibė  $A$ ,
- b) į  $A_i$  žiūrima kaip į faktoraibės elementą  $A_i \in A/\sim$ , ir todėl viršunėje  $A/\sim$

Santykis gali būti ne tik dvivietis, jis gali būti ir trivietis, ir daugia vietis.

**30.6 pavyzdys.** Santykis Skrydis (Kas\_skrenda, Iš\_kur, Į\_kur, Kada) gali būti nustatomas 30.6 lentele.

Kas_skrenda	Iš_kur	Į_kur	Kada
Adomas	Vilnius	Londonas	2009-03-14
Birutė	Kaunas	Paryžius	2009-05-27
...	...	...	...

**30.6 lentelė.** Keturvietį santykį Skrydis (Kas\_skrenda, Iš\_kur, Į\_kur, Kada) nustatanti lentelė

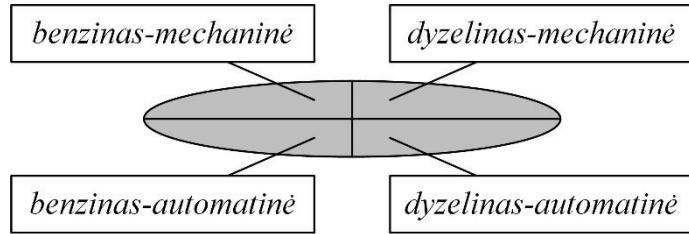
**30.7 teorema.** Jeigu objektų  $x$  ir  $y$  atributo  $atr$  reikšmės sutampa, t. y.  $x.atr = y.atr$ , tai objektais yra ekvivalentiškumo santykije.

Įrodoma pagal apibrėžimą. Įrodomas paliekamas skaitytojui kaip pratimas.

**30.8 pavyzdys.** Imamas spausdintuvų atributas spausdinimo\_tipas. Tegu jo reikšią aibė yra {'lazerinis', 'rašalinis', 'adatinis', 'kitoks'}. Apibrėžiamas 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.

**30.9 pavyzdys.** Automobilių aibė suskaidoma pagal atributą spalva. Kiek spalvų, tiek poaibų: raudoni, geltoni, žali ir t.t.

**30.10 pavyzdys.** Automobilių aibė suskaidoma pagal du atributus: kuras ir greičių\_dėžė. Tegu atributo kuras reikšmių aibė yra {'benzinas', 'dyzelinas'} ir atributo greičių\_dėžė – {'mechaninė', 'automatinė'}. Ekvivalentiškumo santykis gaunamas konjunkcijos operacijos tarp keleto atributų:  $x \sim y$  tada ir tik tada, kai  $(x.kuras = y.kuras) \& (x.greicių_dėžė = y.greicių_dėžė)$ . Šitaip automobilių aibė suskaidoma į keturis poaibius: benzinas-mechaninė, dyzelinas-mechaninė, benzinas-automatinė, dyzelinas-automatinė (30.7 pav).



**30.7 pav.** Automobilių aibės suskaidymas pagal ekvivalentiškumo santykį  $\sim$ , kur  $x \sim y$  jeigu  $(x.kuras = y.kuras) \quad \& \quad (x.greičių_dėžė = y.greičių_dėžė)$ . Iš viso keturi poaibiai

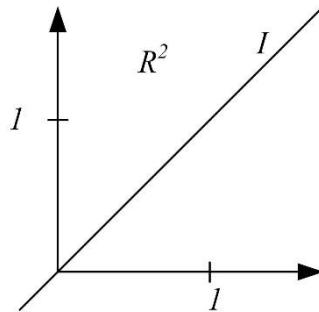
### 30.3. Tiesinės erdvės faktorizavimas pagal tiesinį poerdvį

Abstrakčiojoje algebroje yra nagrinėjamas tiesinės erdvės suskaidymas į faktoraibę pagal tiesinį poerdvį. Toks skaidymas vadinamas tiesinės erdvės *faktorizavimu*.

Žemiau nagrinėjamas pavyzdys, kai imama dvimatiė euklidinė erdvė  $R^2$  ir jos tiesinis poerdvis, žymimas  $I$ . *Tiesinis perdvis*  $I$  apibrėžiamas kaip  $R^2$  poaibis, kuriam galioja tiesiškumo savybės: jeigu  $x \in I$  ir  $y \in I$ , tai  $x+y \in I$ , ir  $t \cdot x \in I$ , kur  $t$  yra bet koks realus skaičius. Tiesinis poerdvis gali būti imamas tokio pavidalo:  $I = \{t \cdot v, \text{ kur } t \in R\}$ . Čia  $v$  yra vadinamas *baziniu vektoriumi*. Analogiškai apibrėžiamas tiesinis poerdvis  $n$ -matėje erdvėje  $R^n$ . Pavyzdžiui, trimatėje erdvėje dvimatis tiesinis poerdvis virš dviejų bazinių vektorių  $v_1$  ir  $v_2$  yra plokštuma:

$$I = \{ t_1 \cdot v_1 + t_2 \cdot v_2, \text{ kur } t_1 \text{ ir } t_2 \text{ yra realūs skaičiai, } t_1 \in R \text{ ir } t_2 \in R \}$$

Dvimatėje euklidinėje erdvėje  $R^2$  tiesinį poerdvį sudaro, pvz., 45 laipsnių kampu einanti tiesė. Tai aibė vektorių, turinčių pavidalą  $(t,t)$ , kur  $t$  yra realus skaičius:  $I = \{t \cdot (1,1), t \in R\}$  (30.8 pav.). Galima įrodyti, kad tokia aibė  $I$  yra tiesinis poerdvis. Reikia įrodyti, pirma, dviejų vektorių  $x$  ir  $y$  iš  $I$  suma  $x+y$  priklauso  $I$ , antra, vektoriaus  $x$  sandauga iš skaliaro  $k$  priklauso  $I$ .



**30.8 pav.** Tiesinis poaibis  $I = \{t \cdot (1,1), t \in R\}$  dvimatėje erdvėje  $R^2$

Įrodomas. Iš  $x \in I$  ir  $y \in I$ , seká 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.  $k \cdot x = k \cdot (t_1, t_1) = (k \cdot t_1, k \cdot t_1) \in I$ . Įrodomo pabaiga.

Tiesinėje erdvėje ekvivalentiškumo santykis  $\sim$  apibrėžiamas šitaip.

**30.11 apibrėžtis.** Vektoriai  $x$  ir  $y$  yra ekvivalentiški, rašoma  $x \sim y$ , tada ir tik tada kai jų skirtumas priklauso tiesiniams poerdiui:

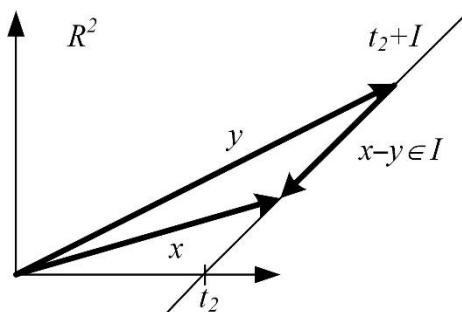
$$x \sim y \Leftrightarrow (x-y) \in I$$

Įrodymas, 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  $x \sim y$  seka ir  $x - y = (t_1, t_1)$ . Iš čia seka  $x = y + (t_1, t_1)$ . Iš  $y \sim z$  seka ir  $y - z = (t_2, t_2)$ . Iš čia seka  $y = z + (t_2, t_2)$ . 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 seka  $x - z = (t_2 + t_1, t_2 + t_1) \in I$ . 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  $\{(t_2 + t, t), t \in R\}$  (30.9 pav.). Kitais žodžiais, tai tiesė, pasvirusi  $45^\circ$  kampu ir nutolusi atstumu  $t_2$  nuo taško  $(0, 0)$ . Iš visų ši poaibį žiūrima kaip į atskirą elementą, dar vadinamą  $R^2/\sim$  tašku. Tokiu būdu į tiesę žiūrima kaip į faktoraibės elementą.

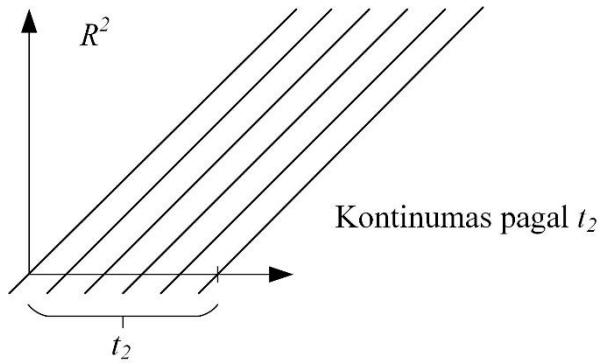


**30.9 pav.** Faktoraibės  $R^2/\sim$  elementas  $t_2 + I$  yra  $R^2$  poaibis, turintis pavidalą  $\{(t_2 + t, t), t \in R\}$ . Rodoma, kad vektorių  $x$  ir  $y$ , iš  $t_2 + I$  skirtumas priklauso  $I$ . Tokiu būdu pagal  $\sim$  apibrėžtį visi vektoriai iš  $t_2 + I$  yra ekvivalentūs

Kaip matome, į tiesę galima žiūrėti dvejopai. Pirma, kaip į taškų aibę –  $R^2$  poaibį, antra, faktoraibės  $R^2/\sim$  elementą.

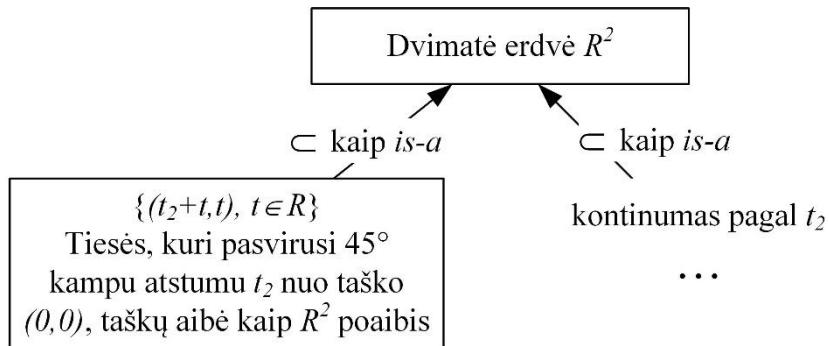
Iš visų dvimatę euklidinę erdvę  $R^2$  galima žiūrėti dvejopai. Pirma, kaip į suskaidymą į tieses, antra, kaip į faktoraibę  $R^2/\sim$ . Pirmuoju požiūriu į  $R^2$  žiūrima kaip į sąjungą tiesių, pasvirusiu  $45^\circ$  kampu. Kiekviena tiesė yra nutolusi savo atstumu  $t_2$  nuo taško  $(0, 0)$  (30.10 pav. ir 30.11 pav.):

$$R^2 = \bigcup_{t_2 \in (-\infty, +\infty)} \{(t_2 + t, t), t \in R\}$$



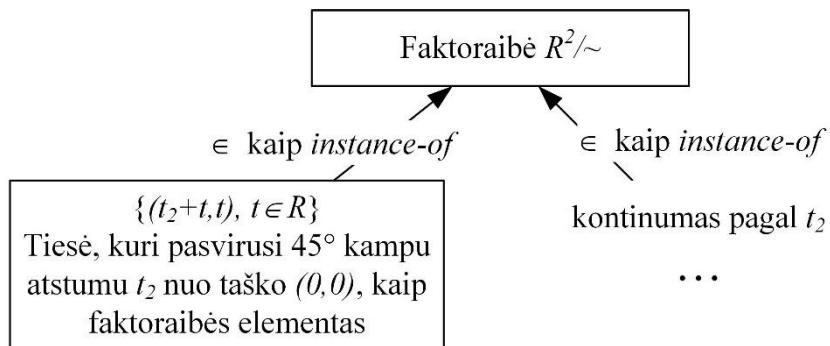
**30.10 pav.** Dvimatės euklidinės erdvės  $R^2$  išskaidymas į tieses, pasvirusias  $45^\circ$  kampu. Kiekviena tiesė nutolusi atstumu  $t_2$  nuo taško  $(0,0)$

Pirmasis požiūris į  $R^2$  kaip į tiesių sąjungą parodytas 30.11 pav.



**30.11 pav.** I dvimatę euklidinę erdvę  $R^2$  yra žiūrima kaip į sąjungą tiesių, kurių kiekvienai pasvirusi  $45^\circ$  kampu ir nutolusi atstumu  $t_2$  nuo taško  $(0,0)$

Antruoj požiūriu šios pasvirusios tiesės yra faktoraibės  $R^2/\sim$  elementai, t. y.  $t_2+I \in R^2/\sim$  kiekvienam  $t_2 \in R$  (30.12 pav.). Čia į tiesę žiūrima kaip į tašką – faktoraibės elementą.



**30.12 pav.** I  $R^2$  yra žiūrima kaip į faktoraibę  $R^2/\sim$ . Tokiu požiūriu atskira tiesė  $t_2+I$  yra faktoraibės  $R^2/\sim$  elementas  $t_2+I$ , t. y.  $t_2+I \in R^2/\sim$

## 31. Semantiniai tinklai pagal Russell ir Norvig

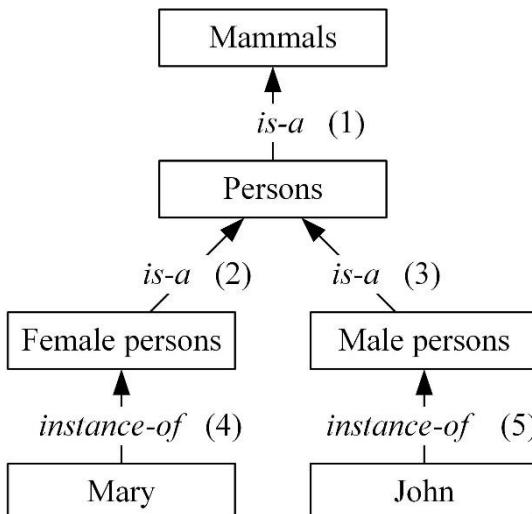
Semantinio tinklo sampratą pristatome aiškindami pavyzdį iš Russello ir Norvigo (2003) knygos<sup>10</sup> skyriaus „Žinių vaizdavimas“ (“Knowledge representation”), p. 350–352. Semantinio tinklo sąvoką pasiūlė 1909 metais amerikiečių matematikas ir mąstytojas Charles Peirce. Nuo tada diskutuojama, ar semantiniai tinklai yra tik logikos užrašymo forma, ar kažkas daugiau. Mes manome, kad semantiniai tinklai nėra vien tik logikos forma. Jeigu taip būtų, tai būtų susitarta dėl griežtos semantinių tinklų semantikos. O semantinių tinklų stiprybė yra tame, kad jiems nesuteikiama formaliai semantika. Interpretuojantis asmuo gali suteikti įvairias prasmes.

Kyla rizika, kad skirtingi asmenys skirtingai supras tą patį pavaizdavimą, bet tokia ir yra kaina. Kartais sąmoningai siekiama suteikti interpretavimo laisvę. Teisėje tai įvardijama kaip atviros prasmės (*open texture*) problema. Teisės mąstytojas Hartas kaip pavyzdį nagrinėja normą „transporto eismas parke draudžiamas“ (žr. Bench-Capon 2002). Keliami klausimai, kas priskiriama transporto priemonėms ir kokios išimtys įvažiavimui. Ar dviračiai, vaikų vežimėliai, riedučiai, žaisliniai automobiliai ir pan. yra transporto priemonės šios normos požiūriu? Ar gali į nelaimės vietą atvykti greitoji pagalba ar taksi išskirtinėmis aplinkybėmis?

Jeigu semantiniam grafui būtų suteikiamā vienintelė prasmė, tai būtų nukertama šaka ant kurios sėdima. Kiekviename reiškinyje galima įžvelgti ir teigiamą, ir neigiamą vertinimo kriterijų. Pavyzdžiui, automobilio gebėjimas pasiekti didesnį greitį, gali būti vertinamas teigiamai, bet tam būtinės didesnis kuro suvartojimas, kuris vertinamas neigiamai.

Semantinis tinklas iš (Russell, Norvig 2003, p. 350–352) užrašomas šiomis matematinės logikos formulėmis (31.1 pav.):

- (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.



**31.1 pav.** Semantinis tinklas iš (Russell, Norvig 2003, p. 350–352)

Šiuos teiginius galima užrašyti ir ne tik predikatų, bet ir aibų bei jų elementų forma (kaip padaryta (Russell, Norvig 2003), pavyzdžiu:

$$\forall x (x \in Persons \Rightarrow x \in Mammals)$$

Sąvokas 31.1 paveiksle žymi stačiakampiai, o Russelas ir Norvigas (2003) naudoja elipses ir sako, kad tai aibės (intensionalai).

Jeigu nagrinėjama dviejų laukų lentelė, tai visų potencialių įrašų aibė sudaro intensionalą. Tai laukų apibrėžimo sričių dekartinė sandauga. Ekstensionalas yra aibė įrašų, faktiškai esančių duomenų bazės egzemplioriuje tam tikru laiko momentu.

I grafinę vaizdavimo formą galima ijdėti daugiau prasmės negu aprašoma logikos formulėmis. Pavyzdžiui, gali būti skirtingai spalvinama ar paryškinama arba stačiakampiams suteikiamas skirtingas dydis.

### 31.1. Paveldėjimas

Savybė, kad žmogus turi dvi kojas, grafiškai pavaizduota 31.2 pav. Čia matyti, kad kategorija skirta ne tik aibei žymėti, bet ir suteikti savybes aibės elementams kaip objektams. Matematikoje aibės elementams nesuteikiamas savybės, pavyzdžiui, spalva, kvapas ir pan. O informatikoje objektiniame vaizdavime savybės suteikiamas.



**31.2 pav.** Kiekvienas asmuo turi dvi kojas:  $Persons.Legs = 2$ . Logikos kalba

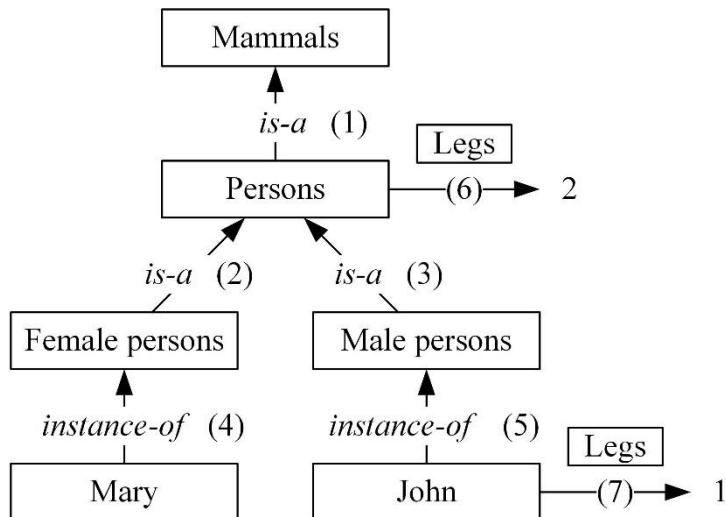
$$\forall x \text{ Person}(x) \Rightarrow \text{Legs}(x) = 2$$

Grafe 31.1 pav. kylant aukštyn ir atsižvelgiant į 31.2 pav. išvedama, kad individuali sąvoka *John* turės savybę  $John.Legs = 2$ . Analogiskas teiginys  $\text{Legs}(\text{John}) = 2$  išvedamas iš formulų (1)–(5) ir (6):

$$(6) \forall x \text{ Person}(x) \Rightarrow \text{Legs}(x) = 2$$

Toliau vaizduojama išimtis – *John* turi vieną koją,  $John.Legs = 1$  (31.3 pav.):

$$(7) \text{Legs}(\text{John}) = 1$$



**31.3 pav.** *John* turi vieną koją:  $John.\text{Legs} = 1$  arba  $\text{Legs}(\text{John}) = 1$

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.

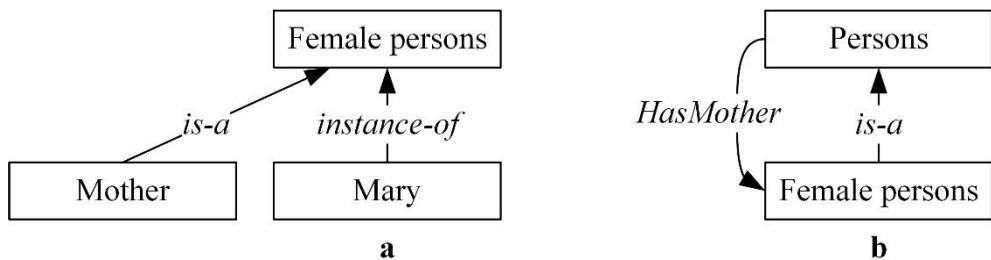
Esant išimtimis, pvz., Legs (John)=1, susitariama, kad labiau specifinė taisykla nugali bendrąją. Tai vadinama *metataisykle*. Tačiau nuo tokios susitarimų nukenčia išvedimo sudėtingumas. Išimtis apie vienakojį John gali būti užrašyta šitaip:

$$\forall x \ (x \in \text{Persons} \ \& \ x \neq \text{John}) \Rightarrow \text{Legs}(x) = 2$$

Tokia kalba sudėtinga dėl išimčių apdorojimo. Ji nepatogi duomenų bazių valdymo sistemose.

### 31.2. Santykis ir ryšys

Ar motinos sąvoką *Mother* galima modeliuoti kaip pavaizduota 31.4 a pav.?



- 31.4 pav. a) Kaip negali būti modeliuojama motinos sąvoka *Mother*.  
b) Santykis *HasMother* tarp asmens ir jo motinos

Deja, 31.4 a pav. rodo neteisingą modeliavimą. Motinos sudaro ne moterų poaibį, kaip modeliuotojui gali pasiroti 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 sąvokų *Persons* ir *Female persons* arba predikatu *HasMother* (*Person*, *Female person*) (31.4 b pav.).

Santykis suprantamas kaip lentelė. Žodžiu *HasMother* yra žymimas santykis kai kalbama apie Dekarto sandaugą ir ją atitinkančią 31.5 lentelę.

Asmuo	Asmens motina
<i>John</i>	<i>Catherine</i>
...	...

**31.5 lentelė.** Dvinaris santykis tarp asmens ir jo motinos – 2 stulpelių lentelė

Kad *Catherine* yra *John* motina vaizduojama dviem predikatais:

- (8) *Female\_person(Catherine)*
- (9) *HasMother(John, Catherine)*

Lentelėje yra galimybė asmeniui įrašyti dvi motinas, pavyzdžiui, *Catherine* ir *Riūta* kaip 31.6 lentelėje, kas prieštarauja realybei. Todėl ryšiams gali būti nustatomi reikalavimai kardinalumui: *1:1*, *1:N* ir *N:M*. Ryšio *HasMother* kardinalumas yra *1:1*.

Asmuo	Asmens motina
John	Catherine
John	Rūta
...	...

**31.6 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 ir apribojimas ryšio kardinalumui:

„*Asmens gimimo* įraše gali būti nurodyta tik viena motina arba nė vienos“

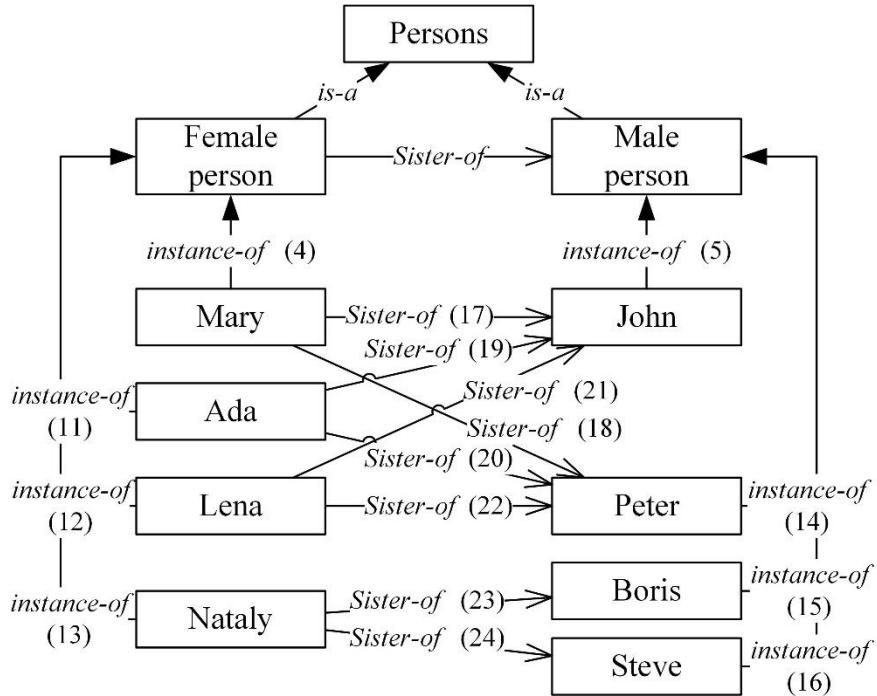
Pastaruoju teiginiu vaizduojamos žinios, kad vaiko gimimo įraše gali nebūti duomenų apie motiną. Taip civilinėje metrikacijoje sprendžiamas vaikų, kurių biologinė motina nėra žinoma, lauk „Motina“ užpildymas. Šis laukas gali būti paliekamas tuščias arba rodyti į moterį, kuri nėra biologinė motina.

### 31.3. 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* (31.7 pav.). 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.



**31.7 pav.** Trys seserys *Mary*, *Ada* ir *Lena* turi du brolius *John* ir *Peter*. *Nataly* turi du brolius *Boris* ir *Steve*

Santykis *Sister-of* nustatytas 31.8 lentele.

Female_person	Male_person
<i>Mary</i>	<i>John</i>
<i>Mary</i>	<i>Peter</i>
<i>Ada</i>	<i>John</i>
<i>Ada</i>	<i>Peter</i>
<i>Lena</i>	<i>John</i>
<i>Lena</i>	<i>Peter</i>
<i>Nataly</i>	<i>Boris</i>
<i>Nataly</i>	<i>Steve</i>

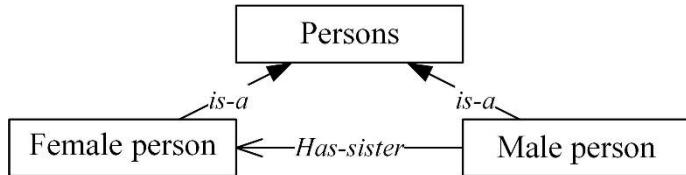
**31.8 lentelė.** Trys seserys *Mary*, *Ada* ir *Lena* turi du brolius *John* ir *Peter*. *Nataly* turi du brolius *Boris* ir *Steve*

**Atvirkštinis santykis.** Naudojimasis atvirkštiniai santykiai sutrumpina išvedimo (užklausos duomenų bazei) laiką. Iveskime ryšį *Has-sister* tarp savokų *Male\_person* ir *Female\_person*. Šis ryšys apibrėžiamas predikatu, kad *Has-sister* yra atvirkštinis *Sister-of* (31.9 pav.):

$$(25) \forall f \forall m \text{ Has-sister}(m, f) \Leftrightarrow \text{Sister-of}(f, m)$$

Pastarasis teiginys (25) reifikuoja ryšį *Has-sister*. Nuo predikato *Has-sister* pradedama žiūrėti į šį ryšį kaip į objektą (31.9 pav.).

**31.1 apibrėžtis. Reifikavimas** – tai predikato arba funkcijos pavertimas algoritminės kalbos objektu (Russell, Norvig 2003, p. 230).



31.9 pav. Ryšys *Has-sister* tarp *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 priskiriamas žmogiškiesiems gebėjimams.

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 santykyje *Sister-of* su *John*. Šios užklausos sudėtingumas yra tiesiškai priklausomas nuo lentelės *Sister-of* ilgio  $N$ . Perrenkami visi lentelės įrašai iš eilės ir ieškoma tokių įrašų, kurių stulpelio *Male\_person* reikšmė yra *John*. Randami trys tokie įrašai.

Ar galimas geresnis sudėtingumas? Klausimas performuluojamas, koks yra užklausos *Has-sister* (*John*, ?) sudėtingumas? Atsakymas: tiesinis pagal lentelės *Sister-of* ilgį  $N$ .

Įrodymas. Iš (25) gaunama:

*Has-sister* (*John*, ?)  $\Leftrightarrow$  *Sister-of* (? , *John*)

Įrodymo pabaiga.

Tuo tarpu užklausos „Kas yra *Ada* broliai?“, *Sister-of* (*Ada*, ?), atsakymas iš 3.8 lentelės yra du įrašai, *John* ir *Peter*. Užklausos sudėtingumas yra logaritminis,  $O(\ln(N))$ . Tai pasiekiamą reliacinę lentelę rūšiavimu pagal raktinį parametrą *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* (31.10 pav.):

(26) *Inverse-of* (*Has-sister*, *Sister-of*).



**31.10 pav.** Teiginys *Inverse-of* (*Has-sister*, *Sister-of*), kad santykis *Has-sister* yra atvirkštinis santykiui *Sister-of* salygoja užklausos *Has-sister* (*John*, ?) logaritminį, o ne tiesinį sudėtingumą

Predikate (26) *Inverse-of* yra transliuojamas į formulę (25), kuri pasako, ką reiškia žodis „atvirkštinis“. Tokia euristinė informacija savo esme yra ryšys tarp santykij *Has-sister* ir *Sister-of* kaip sąvoką. Predikatas (26) yra „sintaksinis cukrus“ (syntactic sugar) atžvilgiu materialaus teiginio (25). Predikato *Inverse-of* parametrai *Has-sister* ir *Sister-of* yra predikatų vardai. Tokiu būdu *Inverse-of* yra užrašytas antros eilės logikos kalba. Tačiau transliuojant (26) į (25) yra atsikratoma antros eilės logikos ir pereinama į pirmos eilės logiką. Po predikato *Inverse-of* sutransliavimo, loginio išvedimo programa kiekvienos užklausos metu (t. y. dinamiškai, *runtime* metu) naudojasi lentele 31.11, kuri atvirkštinė 31.8 lentelei.

<b>Male_person</b>	<b>Female_person</b>
<i>John</i>	<i>Mary</i>
<i>John</i>	<i>Ada</i>
<i>John</i>	<i>Lena</i>
<i>Peter</i>	<i>Mary</i>
<i>Peter</i>	<i>Ada</i>
<i>Peter</i>	<i>Lena</i>
<i>Boris</i>	<i>Nataly</i>
<i>Steve</i>	<i>Nataly</i>

31.11 lentelė. Santykis *Has-sister*. Lentelė skiriasi nuo 31.8 lentelės stulpelių apsukimu ir kita įrašų tvarka, nes raktu tapo *Male\_person*

Lentelė 31.11 skiriasi nuo 31.8 lentelės ne tik stulpelių apsukimu, bet ir kita įrašų tvarka, nes nauju raktu čia tapo *Male\_person*. Ši apsukimo procedūra (25) yra brangi. Jos sudėtingumą galima sumazinti nuo kvadratinio iki  $O(N \cdot \ln(N))$ . 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 – paverstas objektu; tai atliekama transliuojant (26) į (25);
2. ryšiui priskirta tam tikra procedūra (pvz., santykio lentelės apsukimas), t. y. atliktas procedūrinis priskyrimas.

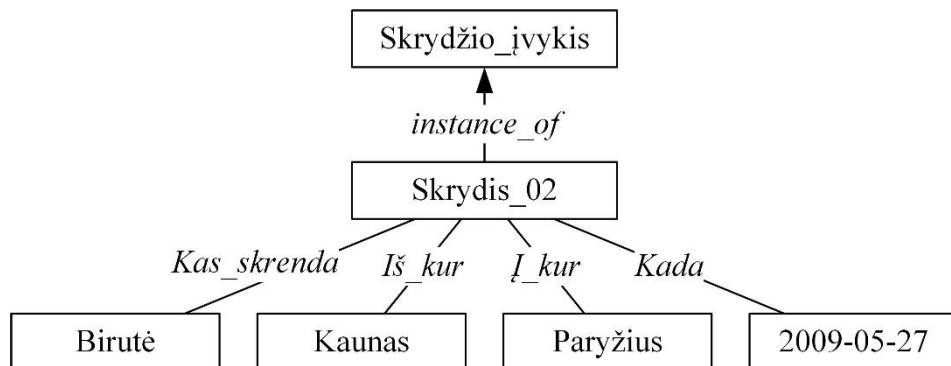
**31.2 apibrėžtis.** Procedūrinis priskyrimas (*procedural attachment*) – tai procedūra, priskiriama ryšiui ir naudojama atsakymui į užklausą arba teiginio išvedimui, kuris yra efektyvesnis negu bendro loginio išvedimo algoritmo atveju.

Procedūrinis priskyrimas ir yra bruožas, skiriantis semantinį tinklą nuo logikos (čia – teiginių logikos arba predikatų logikos). Reifikavimas yra deklaravaus vaizdavimo forma. Procedūrinis priskyrimas yra procedūrinio vaizdavimo forma. Kaip minėta anksčiau, procedūrinio vaizdavimo forma paprastai nepriskiriama žinių vaizdavimui.

Kitas pavyzdys rodo santykio vaizdavimą semantiniu tinklu. Imamas santykis

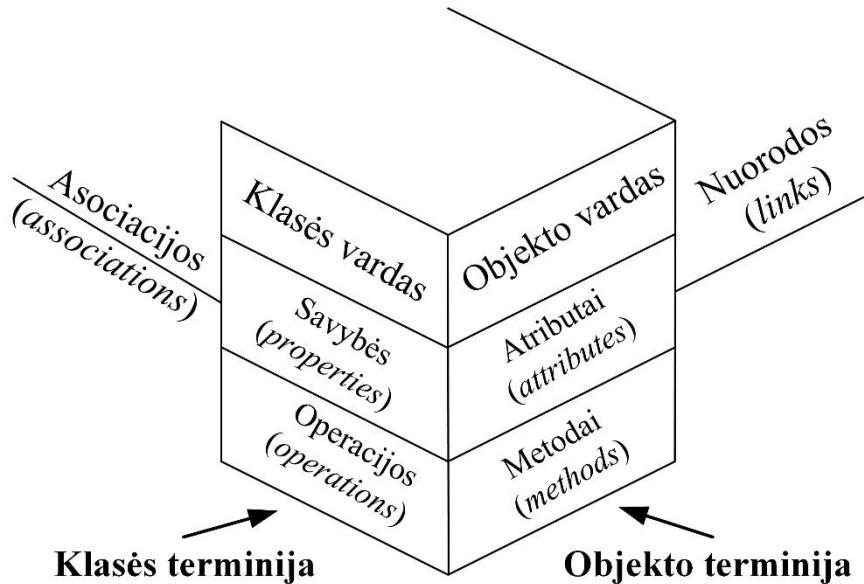
*Skrydis(Kas\_skrenda, Iš\_kur, I\_kur, Kada)*

Teiginys *Skrydis(Birutė, Kaunas, Paryžius, 2009-05-27)* atitinka vieną lentelės įrašą ir yra vaizduojamas semantiniu tinklu 31.12 pav.



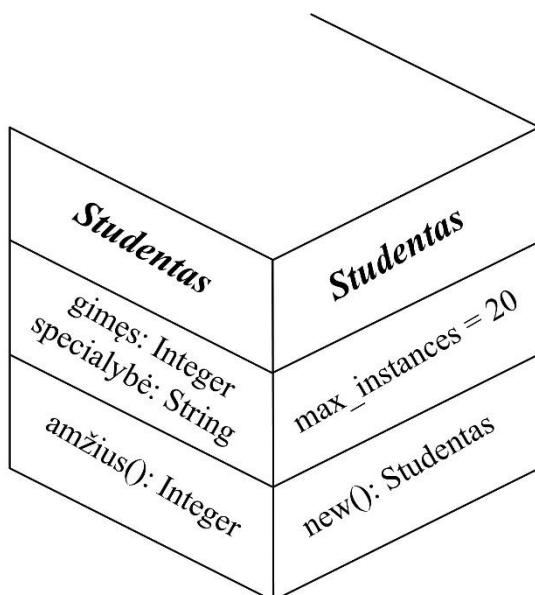
**31.12 pav.** Teiginio *Skrydis(Birutė, Kaunas, Paryžius, 2009-05-27)* vaizdavimas semantiniu tinklu

Semantinių tinklų terminai susiję su objektinio programavimo terminais. Yra neatskiriamos terminų poros: aibė ir jos elementas, egzempliorius; lentelė ir jos įrašas. Yra ryšių tipai *is-a* ir *instance-of*. Klasių ir objektų terminų apjungimas yra parodytas 31.13 pav. Toks požiūris į klasę-objektą yra vadinamas klabjektu (*clabject, class and object*), ir yra metamodelis (Atkinson 1997).



**31.13 pav.** Klabjektas (*clabject, class and object*) – tai objektinio programavimo požiūris į klases ir objektus kaip metamodelis

Klabjekto pavyzdys pateikiamas 31.14 pav. Klabjektas turi savybes, operacijas, asociacijas, atributus, metodus ir nuorodas.

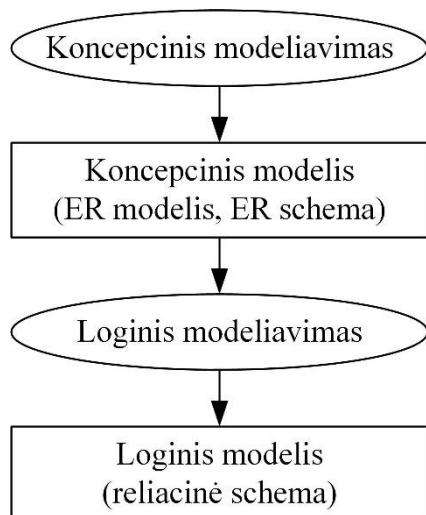


**31.14 pav.** Klabjektas *Studentas*

## 32. Koncepciniai modeliai duomenų bazėse

Šiame skyriuje vadovaujamasi Romo Barono 2002 m. išleistos mokymo priemonės ir atnaujinto vadovėlio (Baronas 2005) 4 skyriumi „Semantinis modeliavimas“. Aptariamas duomenų bazių (DB) esybių-ryšių (ER) modelis (*entity-relationship model*), kurį 1976 m. pasiūlė P. P. 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 – loginis modeliavimas. Jo rezultatas yra loginis modelis (reliacinė schema) (32.1 pav.).



32.1 pav. Duomenų bazės projektavimo etapai

Koncepcinis modelis kuriamas analizės stadijos metu. Toliau imamas duomenų bazės pavyzdys iš R. Barono (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	Statistikas	3	VU
3	Gražulytė	Inžinierius	1	NULL
4	Onaitytė	Vadybininkas	5	VDU
5	Antanaitis	Informatikas	3	VU

32.2 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

32.3 lentelė. DB *Darbai* lentelė *Projektai*

<i>Projektas</i>	<i>Vykdytojas</i>	<i>Statusas</i>	<i>Trukmė</i>
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

32.4 lentelė. DB *Darbai* lentelė *Vykdymas*

Lentelės eilutės suprantamos kaip santykio kortežai. Lentele užduodamas santykis yra Dekarto sandaugos poaibis, pavyzdžiu:

$$Vykdytojai \subset integer \times string \times string \times integer \times string$$

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:

$$(1, 'Jonaitis', 'Informatikas', 2, 'VU') \in Vykdytojai$$

Analogiškai:

$$Vykdytojai \subset Nr \times Pavard\acute{e} \times Kvalifikacija \times Kategorija \times I\ddot{ss}ilavinimas$$

Čia  $Nr \subset integer$ , *Kvalifikacija*  $\subset string$ , *Pavardė*  $\subset string$ , *Kategorija*  $\subset integer$ , *Išsilavinimas*  $\subset string$ .

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 32.5 lentelė.

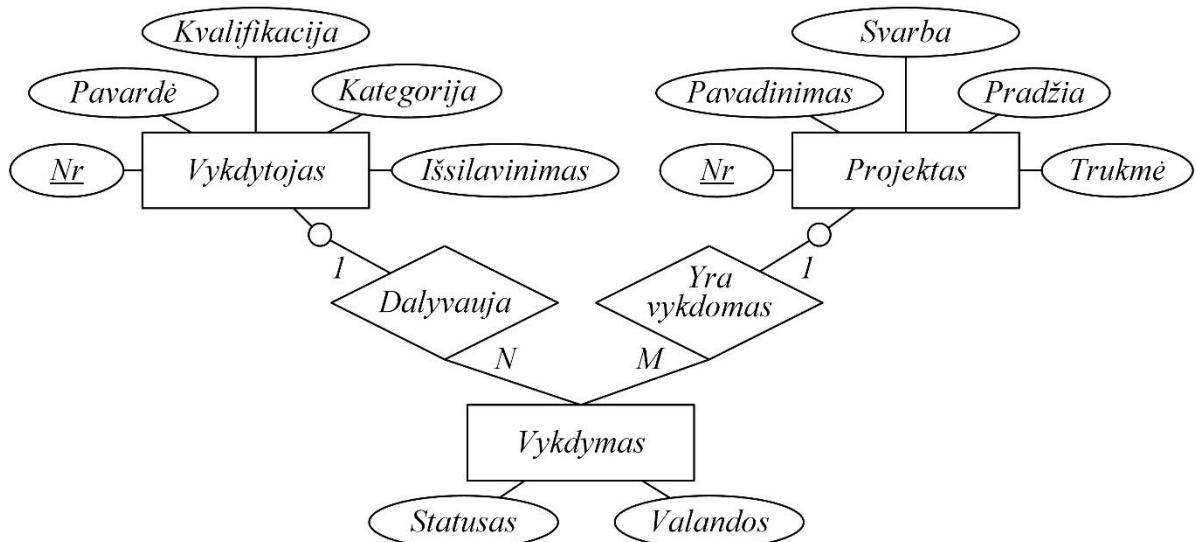
<i>Pavardė</i>	<i>Kategorija</i>
Jonaitis	2
Antanaitis	3

32.5 lentelė. Užklausos rezultatas yra lentelė – įmonės informatikai

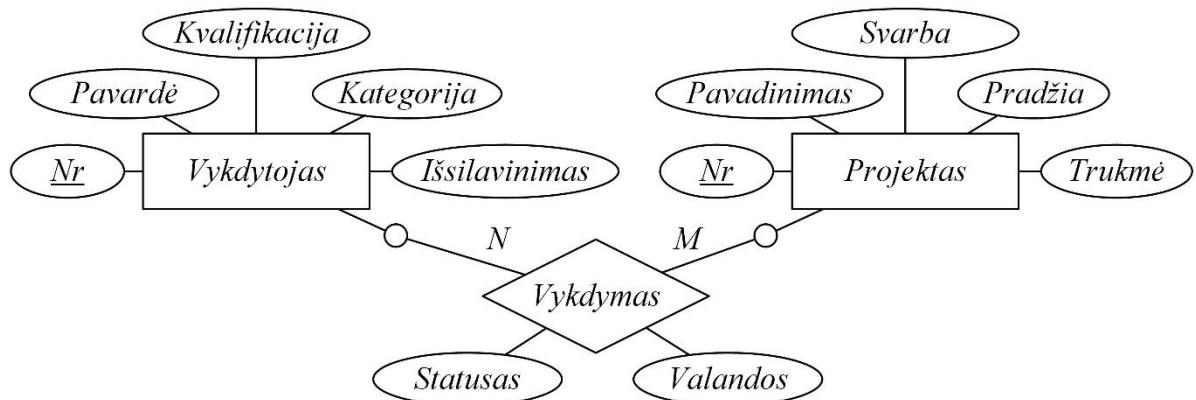
Nagrinėjama įmonės veikla, kai darbuotojai vykdo projektus, kiekvienas turi tam tikrą vaidmenį ir skiria projektui tam tikrą laiką. Išskiriame dvi esybės: *Vykdytojas* ir *Projektas*. Kiekvienos iš šių esybių atributų rinkinys priklauso nuo įmonės veiklos parametru. Tegu

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 sk., p. 78).

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 esybės *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 esybės *Projektas* pusės. Taip sudaryta ER schema pateikta 32.6 pav.
2. I vykdytojų dalyvavimą projektuose galima žiūrėti kaip  $N:M$  tipo ryšį *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 32.7 pav.



32.6 pav. Duomenų bazės *Darbai* ER schema, kai *Vykdymas* modeliuojamas kaip esybė



32.7 pav. Duomenų bazės *Darbai* ER schema, kai *Vykdymas* modeliuojamas kaip ryšys  $N:M$

Toliau aptariami ER schemos elementai.

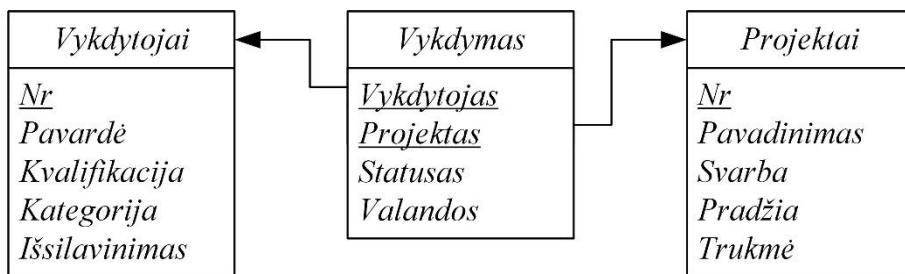
**Esybės.** Kiekviena vaizduojama atskiru stačiakampiu. Jo viduje rašomas esybės vardas.

**Atributai.** Kiekvienas atributas ER schemaje vaizduojamas atskiru ovalu. Su atitinkamos esybės stačiakampiu atributas yra sujungiamas ištisine linija. Ovalo viduje rašomas atributo vardas. Sudėtinio atributo sudedamosios dalys vaizduojamos ovalais, sujungtais ištisine linija su sudėtinio atributo ovalu. Raktinių atributų vardai pabraukiami. Daugiareikšmiai 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. Ryšį vaizduojantis rombas sujungiamas ištisinėmis linijomis su visais ryšio dalyviais. Kiekviename tokia linija pažymima „I“, „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 (32.9 pav. *Dėstytojas ir Studentas* yra *Skaitytojas* potipiai)

Duomenų bazės *Darbai* reliacinių schema, gauta iš ER schemas pateikiama 32.8 pav.



32.8 pav. Duomenų bazės *Darbai* reliacinė schema, gauta iš ER schemos 32.6 pav.

Iš ER schemas esybių *Vykdytojas* ir *Projektas* (32.6 pav.) reliacinėje schemaje 32.8 pav. gaunamos lentelės *Vykdytojai* ir *Projektai* (daugiskaita):

### Vykdytojai(Nr, Pavardė, Kvalifikacija, Kategorija, Išsilavinimas)

### *Projektai(Nr, Pavadinimas, Svarba, Pradžia, Trukmė)*

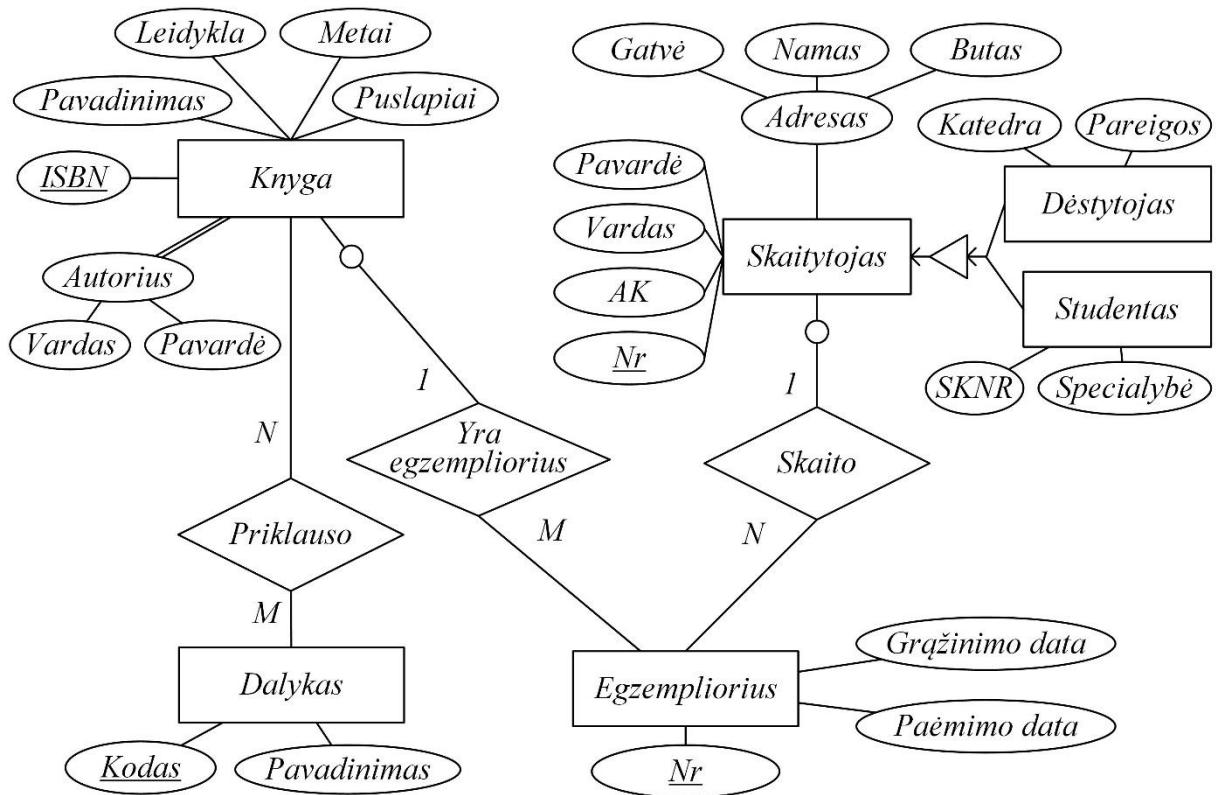
Abiejų ER schemų atvejais, t. y. ir esybei *Vykdymas* (32.6 pav.), ir N:M tipo ryšiui *Vykdymas* (32.7 pav.), reliacinėje schemaoje (32.8 pav.) sudaroma atskira lentelė *Vykdymas*. I ją įtraukiame ryšio atributai, ir jo dalyvių raktai, kurie tampa lentelės išoriniais raktais.

Vykdymas(*Projektas, Vykdytojas, Statusas, Valandos*)

Čia yra du išoriniai raktais: *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 reliacinį modelį taisyklės pateikiamos (Baronas 2005, p. 74–77).

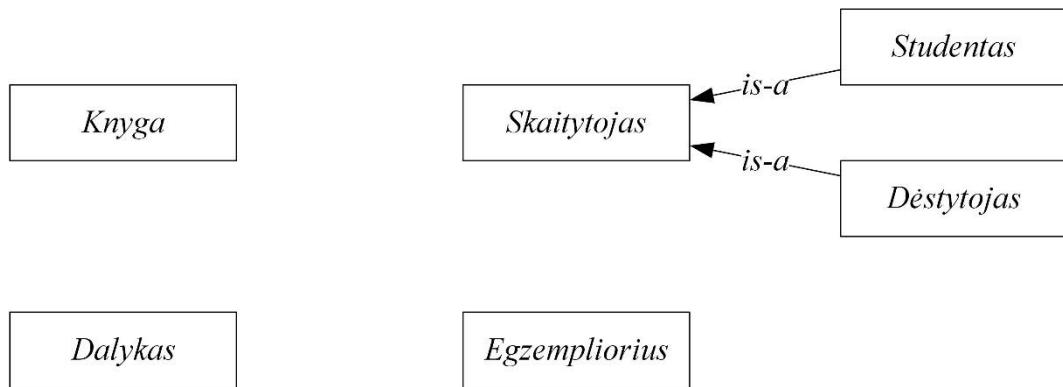
Toliau pateikiamas kitas pavyzdys. Tai duomenų bazės *Biblioteka* ER schema iš (Baronas 2005, p. 74) (32.9 pav.).



32.9 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 išskiriama keturios esybės: *Kniga*, *Skaitytojas*, *Dalykas* ir *Egzempliorius*. Toliau *Skaitytojas* skaidomas į dar dvi esybes: *Dėstytojas* ir *Studentas*. Tokiu būdu tarp pastarųjų ir *Skaitytojas* yra *is-a* ryšys (32.10 pav.).



32.10 pav. *is-a* ryšys tarp esybių *Skaitytojas*, *Studentas* ir *Dėstytojas*

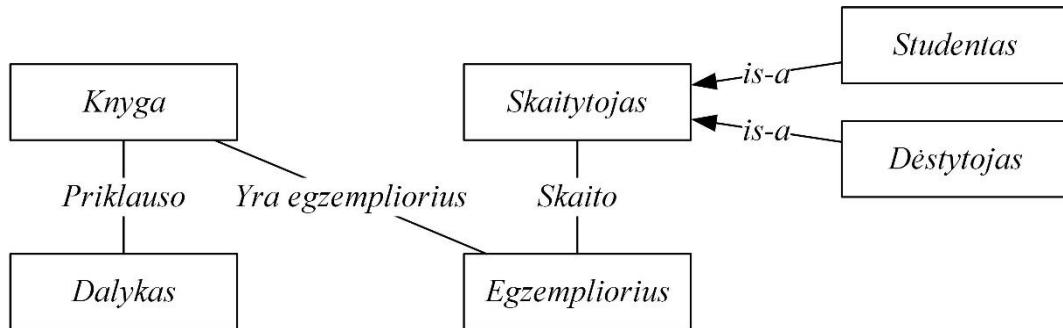
ER schemas lygmenyje yra keturios pagrindinės esybės ir du potipiai:

$$\text{Pagrindinės\_esybės} = \{\text{Kniga}, \text{Dalykas}, \text{Egzempliorius}, \text{Skaitytojas}\}$$

*Potipiai = {Dėstytojas, Studentas}*

Ir pagrindinė esybė, ir potipis yra esybės. Taigi į skaitytojų žiūrima kaip į aibę, o į studentą ir dėstytoją, kaip į poaibius. Tačiau kalbant apie koncepcinį 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, žyminčios ryšius: *Skaito*, *Yra egzempliorius* ir *Priklauso*. Primenama, kad ryšys bus transformuojamas į lentelę. O lentelė yra žemesnio lygmens – reliacinės schemas – sąvoka.



32.11 pav. Koncepcinis modelis, kai 32.10 pav. papildomas trimis rysiais

Belieka pridėti atributus bei ryšių kardinalumus ir prieiti prie ER schemas 32.9 pav.

### 33. Žinių vizualizavimas

Ir žinių vaizdavimas (*knowledge representation*), ir žinių vizualizavimas (*knowledge visualization*) gali būti aptariami ir žinių vadybos (*knowledge management*) kontekste. Žinių vizualizavimas skiriasi nuo žinių vaizdavimo. Žinių vizualizavimas skiriamas žmogui – žmogui skirtas vizualizavimo rezultatas. 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].

Žinių vizualizavimo objektas yra vizualiniai pavaizdavimai (*visual representation*), skirti žinių *kūrimui* (*creation*) ir *perdavimui* (*transfer*) tarp mažiausiai dviejų žmonių (Eppler, Burkhard 2006). Kaip matyti iš apibrėžties, žinių vizualizavimas skirtas žmogui. Vizualaus supratimo (*visual cognition and perception*) studijose daroma išvada, kad vizualizavimas, ženklai padidina žmogaus gebėjimus suvokti ir komunuoti.

Žinių vizualizavimo schemą (*framework*) sudaro trys *perspektyvos* (33.1 lentelė).

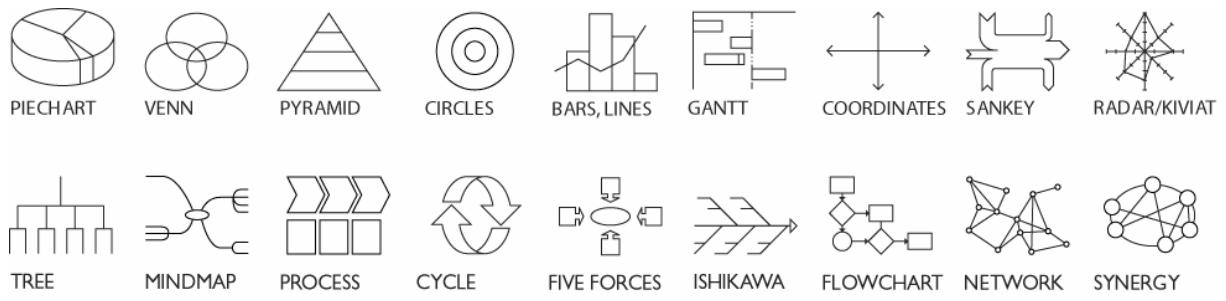
Žinių tipas (kas?)	Vizualizavimo tikslas, paskirtis (kodėl?)	Vizualizavimo formatas (kaip?)
Žinoti-ką	Perdavimas (išaiškinimas, išgavimas, socializacija)	Euristiniai eskizai
Žinoti-kaip	Kūrimas (atradimas, kombinavimas)	Koncepcinės diagramos
Žinoti-kodėl	Mokymasis (gavimas, internalizavimas)	Vizualios metaforos
Žinoti-kur	Kodifikavimas (dokumentavimas, eksternalizavimas)	Žinių animacijos
Žinoti-kas	Radimas (pvz. ekspertų, dokumentų, grupių) Vertinimas (įvertinimas, reitingavimas)	Žinių žemėlapiai Sričių struktūros

33.1 lentelė. Trys žinių vaizdavimo schemas perspektyvos

Šios perspektyvos atsako į tris kertinius klausimus apie vizualizuojamas žinielas:

1. **Kokios** žinielas vizualizuojamos (objektas)?
2. **Kodėl** žinielas vizualizuojamos (tikslas, paskirtis)?
3. **Kaip** žinielas vizualizuojamos (metodas)?

Žinių vaizdavimo kontekste svarbi trečioji perspektyva. Verta atskirai aptarti kiekvieną vizualizavimo formatą. Dažniausiai naudojamų koncepcinių diagramų tipai pagal (Eppler, Burkhard 2006): 1) pyrago diagrama (*pie chart*); 2) Veno (*Venn*) diagrama; 3) piramidė; 4) skrituliai; 5) stulpeliai, kreivės; 6) Ganto (*Gantt*) diagrama; 7) koordinacijų ašys; 8) Sankei (*Sankey*) diagrama (pagal airių kapitoną Matthew Sankey, 1898 m. panaudojusį ją); 9) radaras; 10) medis; 11) minčių žemėlapis; 12) procesas; 13) ciklas; 14) penkios jėgos; 15) Išikava (*Ishikawa*) diagrama; 16) struktūrinė schema; 17) tinklas; 18) synergija (37.2 pav.).



33.2 pav. Dažniausiai naudojamos koncepcinės diagramos: pyrago diagrama, Veno diagrama, piramidė, skrituliai, stulpeliai-kreivės, Ganto diagrama, koordinacių ašys, Sankei diagrama, radaras, medis, minčių žemėlapis, procesas, ciklas, penkios jėgos, Išikava diagrama, struktūrinė schema, tinklas ir sinergija (Eppler, Burkhard 2006, p. 554)

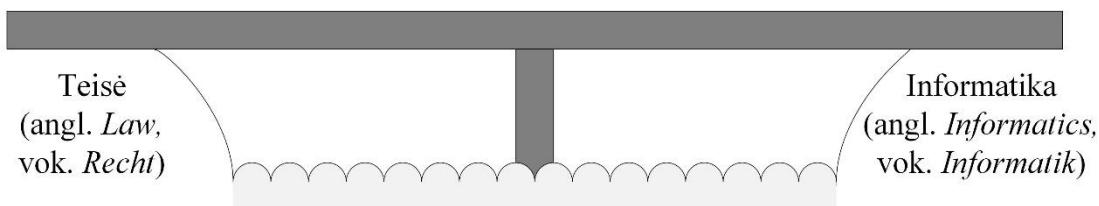
Žinių vizualizavimas svarbus žinių komunikavimui. Žinios komunikojamos įvairiais lygmenimis: tarp individų, grupių ir organizacijų. Svarbu, kad žinios būtų atkuriamas priėmėjo sąmonėje.

## 34. Teisės informatika

Teisės informatika (*legal informatics*, vok. *Rechtsinformatik*) suprantama kaip informatikos taikymas teisės dalykinėje srityje. Tai tarpšakinė disciplina informatikos ir teisės sankirtoje. Skaitomame kurse į teisės informatiką žiūrima visų pirma iš informatikos pusės, t. y. kaip į skaičiavimo modelius, pvz., modelius teisiniame argumentavime (*computational models of argumentation*). Teisės informatika skiriasi nuo informacinių technologijų teisės, kuri dar vadinama informatikos teise. Pastaroji yra teisės šaka ir 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, informacinę, prevencinę, legislatyvinę ir kt. Todėl ir informacinių sistemų turi skirtinę paskirtį – atliekamos skirtinges užduotys. Pavyzdžiuui, teismo administruavimo informacinių sistemų turi skirtinę paskirtį negu įstatymų leidybos informacinių sistemų, pvz., <http://www.lrs.lt>.

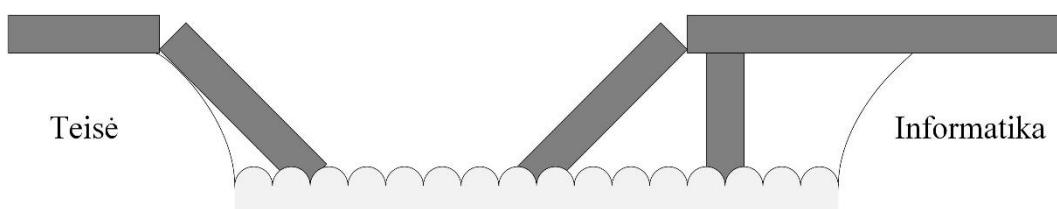
Teisės informatika yra teorija apie ryšį tarp elektroninio duomenų apdorojimo ir teisės bei apie su tuo susijusias priežiūras ir pasekmes (Schweighofer 1999, p. 4). Teisės informatika metaforiškai tiesiai tiltą tarp dviejų upės krantų, teisės ir informatikos (34.1 pav.).



34.1 pav. Teisės informatikos vizualizacija tilto metafora

Teisės informatika labiau aktuali mąstytojams teoretikams, negu praktikams. Informatikos praktikai užsiima informaciniemis technologijomis. Teisės praktikai – teisėjai, advokatai, prokurorai, notarai, juristai ir kt. – atlieka teisės, o ne informatikos, funkcijas.

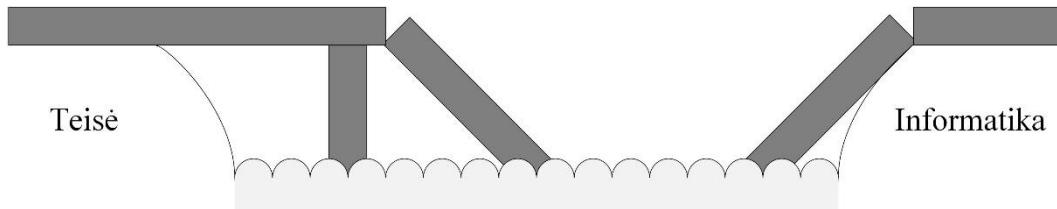
Teisės ir informatikos specialistų bendradarbiavimas turi atitikti jų kompetenciją. Kitais žodžiais, tiltas tarp teisės ir informatikos turi būti statomas subalansuotai. Kai informatikai iš teisininkų tikisi per daug, tai teisininkai neišlaiko, ir tiltas griūna. Pavyzdžiuui, tiltas neišlaikys, jeigu informatikai tikėsis, kad teisininkai formalizuos daugiau informatikos, o ne teisinių žinių reikalaujančius darbus (34.2 pav.).



34.2 pav. Tiltas tarp teisės ir informatikos turi būti statomas subalansuotai. Jeigu teisininkams užkraunama per daug, tai tiltas griūna

Analogiškai, jeigu teisininkai per daug tikisi iš informatikų, tai pastarieji gali neišlaiyti, ir tiltas griūna. Pavyzdžiuui, tiltas neišlaiko, kai informatikai kurdami matematinius modelius

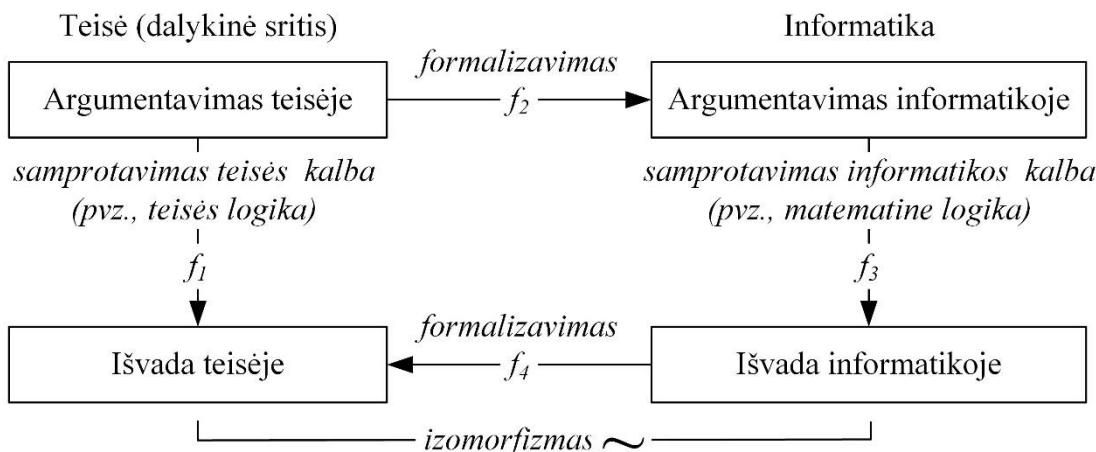
susiduria su teisinių žinių trūkumu arba nesugeba ižvelgti teisinio samprotavimo specifikos. Ši vizualinė metafora pateikta 34.3 pav.



34.3 pav. Kai informatikai užsikrauna per daug arba teisininkai iš jų per daug tikisi, tai informatikai neišlaiko, ir tiltas griūna

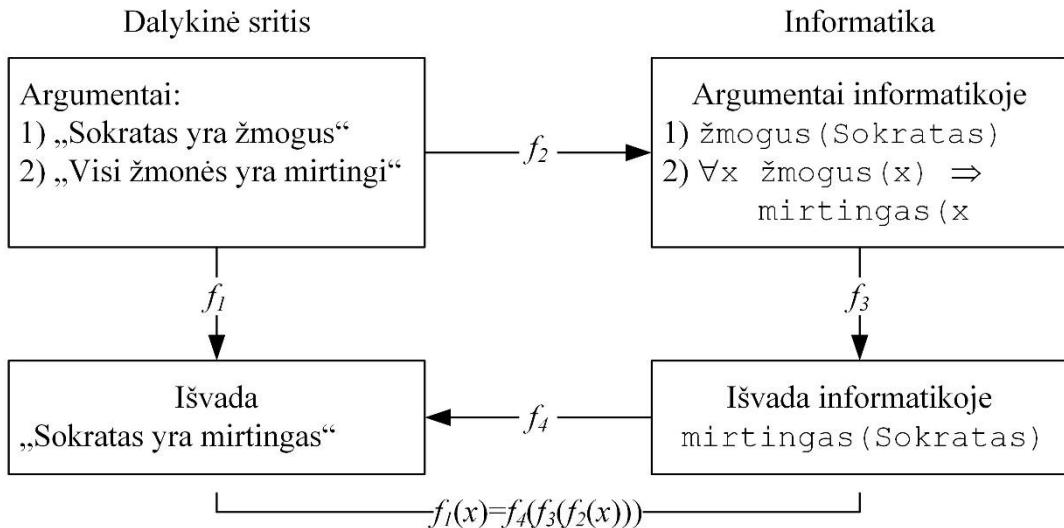
„Teisė“ bei „teisininkas“ gali būti pakeisti kita dalykine sritimi. Vizualizavime įprasta naudotis tilto metafora.

**Modelio atitikimas realybei.** Samprotaujant formalia kalba turi būti gaunama išvada, kurios interpretacija atitinka samprotavimą dalykinės srities kalba. Tegu atvaizdavimas  $f_1$  žymi samprotavimą dalykinės srities kalba, pvz., teisės logika. Atvaizdavimas  $f_2$  yra iš dalykinės kalbos į formalią kalbą, pavyzdžiu, matematinę logiką. Atvaizdavimas  $f_3$  žymi samprotavimą formalia kalba. Atvaizdavimas  $f_4$  žymi perėjimą iš formalios kalbos į dalykinę. Reikalavimas šių atvaizdavimų korektiškumui yra komutavimas diagramoje 34.4 pav., t. y.  $f_1 = f_4 \circ f_3 \circ f_2$ , kur  $\circ$  žymi atvaizdavimų kompoziciją. Kitais žodžiais,  $\forall x f_1(x) = f_4(f_3(f_2(x)))$ .



34.4 pav. Interpretavimo korektiškumas kaip reikalavimas diagramai komutuoti:  $f_1 = f_4 \circ f_3 \circ f_2$ , t. y.  $\forall x f_1(x) = f_4(f_3(f_2(x)))$ . Loginis išvedimas formalia kalba turi atitikti samprotavimą dalykine kalba

Diagrama 34.5 pav. iliustruoja išvadą „Sokratas yra mirtingas“ iš prielaidų 1) „Sokratas yra žmogus“ ir 2) „Visi žmonės yra mirtingi“.



34.5 pav. Iliustruojama išvada „Sokratus yra mirtingas“ iš prielaidų „Sokratus yra žmogus“ ir „Visi žmonės yra mirtingi“

## 35. Literatūra

### Pagrindinė literatūra

- [Brachman, Levesque 2004] Ronald Brachman, Hector Levesque. *Knowledge representation and reasoning*. The Morgan Kaufmann Series in Artificial Intelligence, 2004. 381 p. VU MIF: 004.8/Br-04.
- [Luger 2009] George Luger. *Artificial intelligence: structures and strategies for complex problem solving* (sixth ed.). Addison-Wesley, 2009. <http://www.cs.unm.edu/~luger/>. VU MIF: 004.8/Lu-59.
- [Negnevitsky 2011] Michael Negnevitsky. *Artificial intelligence: a guide to intelligent systems* (3rd ed.). Addison-Wesley, 2011. VU MIF: 004.8/Ne-44.
- [Russell, Norvig 2010] Stuart Russell, Peter Norvig. *Artificial intelligence: a modern approach* (third edition). Prentice Hall, 2010. <http://aima.cs.berkeley.edu>. VU MIF: 004.8/Ru-151.

### Papildoma literatūra

- [Norgėla 2007] S. Norgėla. *Logika ir dirbtinis intelektas*. – Vilnius: TEV, 2007. 256 p. MIF: 16/No66.
- [Nilsson 2010] Nils Nilsson. *The quest for artificial intelligence: a history of ideas and achievements*. Cambridge University Press. VU MIF: 004.8/Ni-133.
- [Nilsson 1998] Nils Nilsson. *Artificial intelligence: a new synthesis*. Morgan Kaufmann Publishers, 1998. 513 p. VU MIF: 004.8/Ni-133.
- [Nilsson 1982] Nils Nilsson. *Principles of artificial intelligence*. Springer, Berlin Heidelberg.
- [Sowa 2000] John F. Sowa. *Knowledge representation: logical philosophical, and computational foundations*. Brooks/Cole, Pacific Grove, 2000. 573 p.
- [Stefik 1995] Mark Stefik. *Introduction to knowledge systems*. Morgan Kaufmann Publishers, 1995. 871 p.

### Kita naudota arba cituota literatūra

- [Ackoff 1989] R. L. Ackoff. From data to wisdom. *Journal of Applied Systems Analysis*, vol. 16, p. 3–9, 1989.
- [Antoniou et al. 2012] G. Antoniou, P. Groth, F. van Harmelen, R. Hoekstra. *A semantic web primer* (third ed.). The MIT Press, 2012, 270 p. VU MIF 004.8/An-154.
- [Archer 1999] A. F. Archer. A Modern Treatment of the 15 Puzzle. *The American Mathematical Monthly*, vol. 106 p. 793–799, 1999. [www.cs.cmu.edu/afs/cs/academic/class/15859-f01/www/notes/15-puzzle.pdf](http://www.cs.cmu.edu/afs/cs/academic/class/15859-f01/www/notes/15-puzzle.pdf)
- [Atkinson 1997] C. Atkinson. Meta-modeling for distributed object environments. *EDOC'97 Proceedings*. IEEE Computer Society Press.
- [Baronas 2005] R. Baronas. Duomenų bazių valdymo sistemos. Vadovėlis aukštosioms mokykloms. Vilnius: TEV, 2005. 184 p. <http://www.mif.vu.lt/~baronas/dbvs/book/>.
- [Bellinger, Castro, Mills 2004] G. Bellinger, D. Castro, A. Mills. Data, information, knowledge and wisdom. [Interaktyvus, žiūrėta 2014-05-03] <http://www.systems-thinking.org/dikw/dikw.htm>.
- [Bench-Capon, Prakken 2006] T. J. M. Bench-Capon, H. Prakken. Justifying actions by accruing arguments. In: *Computational Models of Argument – Proceedings of COMMA 2006*, p. 247–258. IOS Press, 2006. [Interaktyvus, žiūrėta 2015-04-17] <http://www.cs.uu.nl/groups/IS/archive/henry/action.pdf>, skaidrės <http://www.cs.uu.nl/deon2006/tbc+hp.pdf>.

- [Bordini et al. 2007] R. Bordini, J. F. Hübner, M. Wooldridge. *Programming multi-agent systems in AgentSpeak using Jason*. John Wiley, 273 p. VU MIF: 004.42/Bo-271.
- [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, p. 111–117.
- [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.
- [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, p. 395–416, <http://arjournals.annualreviews.org/action/showJournals>.
- [Chen 1976] Peter Pin-Shan Chen. The entity-relationship model – towards a unified view of data. *ACM Transactions on Database Systems (TODS)*, 1(1), 9–36, 1976.
- [Cook 2009] R. T. Cook. Intensional definition. In *A Dictionary of Philosophical Logic*. Edinburgh: Edinburgh University Press, 2009, p. 155. [http://en.wikipedia.org/wiki/Intensional\\_definition](http://en.wikipedia.org/wiki/Intensional_definition).
- [Čaplinskas 1998] A. Čaplinskas. AI paradigms. *Journal of Intelligent Manufacturing*, 9(6): 493–502, 1998.
- [Dagienė, Grigas, Augutis 1986] V. Dagienė, G. Grigas, K. Augutis. *Šimtas programavimo uždavinių*. Kaunas: Šviesa, 1986. 223 p.
- [Eppler, Burkhard 2006] M. J. Eppler, R. A. Burkhard. *Knowledge visualization*. Encyclopedia of knowledge management, D. G. Schwartz (ed.). Idea Group Reference, Hershey, 2006. p. 551–559.
- [Ernst, Newell 1969] G. W. Ernst, A. Newell. GPS: A case study in generality and problem solving. New York: Academic Press.
- [Falkenberg et al. 1998] E. D. Falkenberg, W. Hesse, P. Lindgreen, B. E. Nilsson, J. L. H. Oei, C. Rolland, R. K. Stamper, F. J. M. Van Assche, A. A. Verrijn-Stuart, K. Voss. A framework of information system concepts, The FRISCO Report (Web edition), IFIP, the Netherlands, 1998. <http://www.mathematik.uni-marburg.de/~hesse/papers/fri-full.pdf>.
- [Geldenhuys 1999] Geldenhuys A.E., van Rooyen H. O., Stetter F. (1999). Knowledge representation and relation nets. Kluwer Academic Publishers, Boston, 279 p.
- [Gruber 1993] T. J. Gruber. A translation approach to portable ontologies. *Knowledge Acquisition*, 592: 199–220, 1993. <http://www-ksl.stanford.edu/kst/what-is-an-ontology.html>.
- [Guarino et al. 2009] N. Guarino, D. Oberle, S. Staab. What is an Ontology? In: *Handbook on Ontologies*, S. Staab, R. Studer (eds.) p. 1–17, Springer, 2009. [http://iaoa.org/isc2012/docs/Guarino2009\\_What\\_is\\_an\\_Ontology.pdf](http://iaoa.org/isc2012/docs/Guarino2009_What_is_an_Ontology.pdf).
- [Hesse, Verrijn-Stuart 2001] W. Hesse, A. Verrijn-Stuart. Towards a theory of information systems: the FRISCO approach. In: H. Jaakkola et al. (eds.) *Information Modelling and Knowledge Bases XII*, p. 81–91. IOS Press, Amsterdam, 2001.
- [Heesen et al. 1997]. C. Heesen, V. Homburg, M. Offereins. An agent view on law. *Artificial Intelligence and Law*, vol. 5, no. 4, p. 323–340 (1997).
- [Jensen, Wirth 1982]. K. Jensen, N. Wirth. *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.
- [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.
- [Loeckx et al. 1996] Loeckx, Jacques; Ehrich, Hans-Dieter; Wolf, Markus. *Specification of abstract data types*. Wiley-Teubner, Chichester, 1996.

- [McCarthy 1958] J. McCarthy. Programs with common sense. *Mechanisation of Thought Processes, Proc. Symp. Nat. Phys. Lab.*, vol. 1, p. 77–84. London: Her Majesty's Stationery Office. (Reprinted in SIP, p. 403–410).
- [Marcijonas, Sudavičius 2003] A. Marcijonas, B. Sudavičius. Mokesčių teisė. – Vilnius: Teisinės informacijos centras, 2003. – 288 p.
- [MAJ] Lietuvos Respublikos mokesčių administravimo įstatymas, 2004 m. balandžio 13 d. Nr. IX-2112, Žin., 2004, Nr. 63-2243.
- [Minsky 1975] M. Minsky. A Framework for Representing Knowledge. *The Psychology of Computer Vision*. P. 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.
- [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.
- [Norgėla, Sakalauskaitė 2007] S. Norgėla, J. Sakalauskaitė. Neklasikinės logikos informatikams. – Vilnius: TEV, 2007. – 132 p.
- [OED 2007] *Oxford English dictionary*. Oxford University Press, 2007. <http://dictionary.oed.com/>.
- [Paradauskas, Nemuraitė 2008] B. Paradauskas, L. Nemuraitė. Duomenų bazių semantiniai modeliai. – Kaunas: Kauno technologijos universitetas, 2008. – 338 p.
- [Plečkaitis 2004] R. Plečkaitis. *Logikos pagrindai*. – Vilnius: Tyto alba, 2004. – 438 p.
- [Sawyer, Foster 1986] B. Sawyer, D. Foster. Programming expert systems in Pascal. John-Wiley, 1986.
- [Schweighofer 1999] E. Schweighofer. *Legal knowledge representation. Automatic text analysis in public international and European law*. Kluwer Law International, The Hague, 1999.
- [Thayse et al. 1990] André Thayse, Pascal Gribomont, Georges Louis et al. *Approche logique de l'intelligence artificielle*. Dunot, Paris, 1988. Vertimas į rusų kalbą: А. Тейз А., Грибомон П., Луи Ж. и др. *Логический подход к искусственному интеллекту: от классической логики к логическому программированию*. Перевод с франц. – Москва: Мир, 1990. 432 с.
- [Tiuringas 1961] A. M. Tiuringas. Bendroji ir loginė automatu teorija. Vilnius, 1961.
- [Turing 1950] A. M. Turing. Computing machinery and intelligence. *Mind*, 1950, vol. 59, no. 236, p. 433–460. <http://mind.oxfordjournals.org/content/LIX/236/433>, <http://www.loebner.net/Prizef/TuringArticle.html>.
- [Tyugu 1984] Э. Х. Тыугу. Концептуальное программирование. – Москва: Наука, 1984. Vertimas į anglų kalbą: Е. Тыугу. *Knowledge-based programming*. Wokingham: Turing Institute with Addison-Wesley, 1987. 243 p.
- [Valente 1995] A. Valente. *Legal knowledge engineering: A modelling approach*. IOS Press, Amsterdam, 217 p.
- [van Harmelen et. al 2008] *Handbook of knowledge representation*. F. van Harmelen, V. Lifschitz, B. Porter (eds.). Elsevier, Amsterdam, 2008.
- [Wooldridge 2002] Michael Wooldridge. *An introduction to multiagent systems*. John Wiley, UK, 2002. 348 p.
- [Zarri 2006] G. P. Zarri. Knowledge representation. In: *Encyclopedia of knowledge management*, D. G. Schwartz (ed.). Idea Group Reference, Hershey, 2006. p. 467–477.