VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
INFORMATIKOS KATEDRA

Valdas Diči ̄unas Gintaras Skersys

DISKREČIOJI MATEMATIKA

Mokymo priemonė

Vilnius – 2003

Įvadas

Išvertus iš lotynų kalbos žodis “diskretus” (*discretus*, lot.) reiškia “atskirtas”. (Nemaišykite su iš pranc ̄uzų kalbos kilusiu žodžiu “diskretiškas”, kuris reiškia “laikantis paslaptį”.) Taigi, *diskrečioji matematika* nagrinėja objektus kurie yra tam tikra prasme atskirti vienas nuo kito, pavieniai. Tuo diskrečioji matematika yra priešpastatoma tolydžiajai matematikai, kurios nagrinėjami objektai yra tolyd ̄us, vientisi, ir kur didžiausią vaidmenį vaidina ribos ir tolydumo sąvokos. Paprasčiausi diskret ̄us objektai yra baigtinės ir skaičiosios aibės, pavyzdžiui, aibė $\{0, 1\}$, nat ̄uraliųjų skaičių aibė $\mathbb{N}$ ir racionaliųjų skaičių aibė $\mathbb{Q}$. Tolydžių objektų pavyzdžiai yra realiųjų skaičių aibė $\mathbb{R}$ bei kompleksinių skaičių aibė $\mathbb{C}$.

Remiantis tokiu nagrinėjamų objektų skirstymu, diskrečiai matematikai plačiąja šios sąvokos suvokimo prasme galima b ̄utų priskirti gerą pusę matematikos: algebrą, skaičių teoriją, tikimybių teoriją, matematinę logiką ir t.t. Kadangi minėti mokslai išsivystė į savarankiškas matematikos šakas, tai tradiciškai diskrečiajai matematikai priskiriamos matematikos sritys nagrinėjančios *baigtinius objektus*, jų skaičiavimą, sąryšius tarp jų bei paprasčiausias operacijas su jais:

- Kombinatorika;
- Grafų teorija;
- B ̄ulio funkcijų ir schemų teorija;
- Kodavimo teorija;
- Algoritmų teorija.

Pirmieji du aukščiau išvardinti dalykai jums bus dėstomi kituose semestruose, todėl šį semestrą mes nagrinėsime tris likusias sritis, t.y., *B ̄ulio funkcijas bei schemas*, *kodavimo teoriją* ir *algoritmų teoriją*. Algoritmų teorija priklauso diskrečiai matematikai todėl, kad intuityvia prasme algoritmas yra baigtinis instrukcijų (komandų) rinkinys, leidžiantis žingsnis po žingsnio taikant šias instrukcijas konstruktyviems objektams išspręsti norimą uždavinį. Taigi, tiek algoritmas, tiek jo vykdymas, tiek objektai, kuriais gali manipuliuoti algoritmas, paprastai yra diskret ̄us.

Be trijų minėtų sričių mes trumpai panagrinėsime *matematinės logikos* pradmenis, kadangi matematinė logika nagrinėja formalius samprotavimus bei įrodymus, jų teisingumą, automatinio

(algoritminio) įrodymo galimybes, o visa tai reikalinga tiek matematikoje, tiek informatikoje. Matematinė logika jums padės suprasti ir kai kurių jums dėstomų disciplinų logiką, o kartais ir pačių dėstytojų logiką.

Jūs turbūt jau žinote, kad šiuolaikiniai kompiuteriai operuoja dvejetainiais kodais. Dvejetainiu pavidalu galime koduoti tiek skaičius, tiek ir įvairius kitokius objektus. Be to, kompiuterio atmintyje gali būti saugomi tik baigtinio ilgio kodai. Kadangi realieji skaičiai yra begalinės periodinės arba neperiodinės trupmenos, tai jie visada yra apvalinami iki artimiausios baigtinės dvejetainės trupmenos. Taigi, informatikos, kaip kompiuterių ir algoritmų mokslo, objektai taip pat yra diskretūs. Jei norėsite suprasti šiuolaikinių kompiuterių architektūrą, programinę įrangą, komunikacijos sistemas, skaitmeninį signalų apdorojimą, informacijos teoriją, neuroninius tinklus, valdymo sistemas ir t.t., jūs turėsite išmokti bent truputį, o gal ir daugiau, diskrečiosios matematikos. Daugelis šiuolaikinės ekonomikos modelių taip pat yra diskretūs. Taigi, diskrečioji matematika yra daugelio matematikos, informatikos, ekonomikos sričių pagrindas, o tuo pačiu ir daugelio šiandienos progresyvių technologijų pagrindas!

Rekomenduojama literatūra

1. K.H. Rosen, *Discrete Mathematics and Its Applications*, 5-as leid., McGraw-Hill, Boston, 2003.

2. S.V. Jablonskij, *Vvedenije v Diskretnuju Matematiku*, 2-as leid., Nauka, Maskva, 1986 (rusų k.).

3. S. Norgėla, *Matematinės Logikos Įvadas*, VU leidykla, Vilnius, 1985.

4. R. Grigutis, *Baigtinės Algebrinės Struktūros*, VU leidykla, Vilnius, 1998.

5. V. Stakėnas, *Informacijos Kodavimas*, VU leidykla, Vilnius, 1996.

1 skyrius

Aibės, funkcijos ir sąryšiai

1.1 Aibių algebra

Paprasčiausias diskretus objektas yra *baigtinė aibė*, todėl šiame skyrelyje priminsime kai kurias aibių teorijos sąvokas.

Aibė ir jos *elementas* yra pirminės matematikos sąvokos (kaip *skaičius, taškas* ir pan.). Intuityviai aibę laikome skirtingų objektų rinkiniu, o pačius objektus vadiname tos aibės elementais. Tai, kad elementas a priklauso aibei A, žymime $a \in A$. Jei b nepriklauso A, rašome $b \notin A$. Tuščią rinkinį vadiname *tuščia aibe* ir žymime $\emptyset$. Aibė B yra aibės A *poaibis* ($B \subseteq A$), jei $B = \emptyset$, arba kiekvienas aibės B elementas priklauso ir aibei A. Jei $B \subseteq A$ ir $B \neq A$, poaibį B vadiname *tikru* ir žymime $B \subset A$. Aibės A poaibių aibę žymime $\mathcal{P}(A)$. Baigtinę aibę, turinčią $n \geq 0$ elementų, vadiname *n-aibe*, o bet kurį jos poaibį, turintį m elementų, vadiname *m-poaibiu* ($0 \leq m \leq n$). Baigtinės aibės elementų skaičius dar vadinamas jos galia. Aibės A galia žymima $|A|$.

Baigtinė aibė paprastai apibrėžiama, *išvardijant* jos elementus: $A = \{a_1, \ldots, a_n\}$. Kitas aibės apibrėžimo būdas — nurodoma *savybė* S, kuria pasižymi tos aibės elementai ir nepasižymi visi kiti objektai. Tokiu atveju naudojamas užrašas $A = \{x\colon S(x)\}$. Pavyzdžiui, $\mathbb{Z}_3 = \{x\colon x$ yra sveikasis skaičius, kuris dalinasi iš $3\}$. Trečias aibės apibrėžimo būdas yra *induktyvus* arba *konstruktyvus*. Paprastai induktyvus apibrėžimas būna sudarytas iš trijų skirsnių. Pirmajame skirsnyje mes nurodome vieną ar keletą objektų, kurie priklauso apibrėžiamai aibei. Antrajame skirsnyje pateikiame būdą kaip iš jau turimų aibės elementų gauti naujus jos elementus. Pagaliau trečias skirsnis teigia, kad objektas priklauso apibrėžiamai aibei tada ir tik tada, kai tai išplaukia iš pirmojo ir antrojo skirsnių.

Pavyzdys 1.1.1. Visais trimis išvardintais būdais apibrėšime vieną ir tą pačią lyginių sveikųjų skaičių aibę $\mathbb{Z}_2$:

- $\mathbb{Z}_2 = \{\ldots, -4, -2, 0, 2, 4, 6, \ldots\}$;
- $\mathbb{Z}_2 = \{x\colon x$ yra lyginis sveikasis skaičius$\}$;

- 1. $0 \in \mathbb{Z}_2$;
  2. Jei x priklauso $\mathbb{Z}_2$, tai $x+2$ ir $x-2$ taip pat priklauso $\mathbb{Z}_2$;
  3. Bet kuris objektas priklauso aibei $\mathbb{Z}_2$ tada ir tik tada, kai tai išplaukia iš 1 ir 2 šio apibrėžimo skirsnių.

Dabar apibrėšime pagrindines aibių operacijas (naudodami antrą aibių apibrėžimo būdą):

- aibių A ir B *sąjunga* $A \cup B = \{x\colon x \in A \text{ arba } x \in B\}$;
- aibių A ir B *sankirta* $A \cap B = \{x\colon x \in A \text{ ir } x \in B\}$;
- aibių A ir B *skirtumas* $A \setminus B = \{x\colon x \in A \text{ ir } x \notin B\}$.

Kai visos nagrinėjamos aibės yra poaibiai kokios nors didesnės (*universalios*) aibės U, dažnai apibrėžiama dar viena aibių operacija:

- aibės $A \subseteq U$ *papildinys* (iki aibės U) $\overline{A} = U \setminus A = \{x\colon x \notin A\}$.

Aišku, kad aibė U turi būti nurodyta arba nuspėjama iš konteksto.

Pavyzdys 1.1.2. Nagrinėsime natūrinių skaičių aibės $\mathbb{N} = \{0, 1, 2, \ldots\}$ poaibius, t.y., $U = \mathbb{N}$. Apibrėžę

$$\mathbb{N}_2 = \{x\colon x \text{ dalinasi iš } 2\} \quad \text{ir} \quad \mathbb{N}_3 = \{x\colon x \text{ dalinasi iš } 3\},$$

gauname

$$\begin{aligned}
\mathbb{N}_2 \cup \mathbb{N}_3 &= \{x\colon x \text{ dalinasi iš 2 arba 3}\} = \{0, 2, 3, 4, 6, 8, 9, \ldots\};\\
\mathbb{N}_2 \cap \mathbb{N}_3 &= \{x\colon x \text{ dalinasi iš 6}\} = \{0, 6, 12, \ldots\};\\
\mathbb{N}_2 \setminus \mathbb{N}_3 &= \{x\colon x \text{ dalinasi iš 2, bet nesidalina iš 3}\} = \{2, 4, 8, 10, 14, \ldots\};\\
\overline{\mathbb{N}_2} &= \{x\colon x \text{ nesidalina iš 2}\} = \{1, 3, 5, \ldots\}.
\end{aligned}$$

Aibių operacijos $\cup, \cap$ ir $\setminus$ atitinka aritmetines operacijas $+, \times$ ir $-$. Todėl aibių operacijos tenkina kelis dėsnius, kuriuos jūs jau žinote iš aritmetikos:

(1) $A \circ B = B \circ A$ (*komutatyvumas*; čia vietoje $\circ$ abiejose lygybės pusėse galime įstatyti $\cup$ arba $\cap$);

(2) $A \circ (B \circ C) = (A \circ B) \circ C$ (*asociatyvumas*);

(3) $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$ (*distributyvumas*);

(4) (a) $A \cup \emptyset = A$;

(b) $A \cap \emptyset = \emptyset$ (tuščia aibė $\emptyset$ atitinka skaičių 0 aritmetikoje);

(5) $\overline{\overline{A}} = A$.

Aibių visumą, kurioje apibrėžtos sąjungos, sankirtos ir papildinio operacijos, vadina *aibių algebra*. Aibių algebroje galioja ir kiti dėsniai, neturintys analogų aritmetikoje:

(6) (a) $A \cup A = A$;
(b) $A \cap A = A$ (*idempotentumas*);

(7) $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$ (*antras distributyvumo dėsnis*);

(8) (a) $\overline{A \cup B} = \overline{A} \cap \overline{B}$;
(b) $\overline{A \cap B} = \overline{A} \cup \overline{B}$ (*de Morgano dėsniai*);

Pastebėkime, kad dėsnius (6)(b), (7) ir (8)(b) galima gauti atitinkamai iš dėsnių (6)(a), (3) ir (8)(a), pakeitus juose ženklus $\cup$ į $\cap$ ir atvirkščiai. Pasirodo, aibių algebroje galioja dualumo principas:

Dualumo principas. *Jei turime teisingą aibių lygybę, sudarytą iš universalios aibės U poaibių ir operacijų $\cup, \cap$ bei $^-$ (papildinio), tai pakeitę abiejose lygybės pusėse ženklus $\cup$ į $\cap$, $\cap$ į $\cup$, $\emptyset$ į U ir U į $\emptyset$, vėl gausime teisingą aibių lygybę.*

Pavyzdžiui, iš (4) galime gauti teisingas lygybes

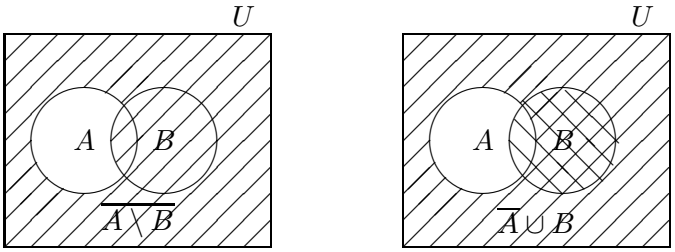
(9) (a) $A \cap U = A$;
(b) $A \cup U = U$.

Norint įrodyti, kad dvi aibės yra lygios ($C = D$), pakanka įrodyti, kad kiekviena iš jų yra kitos poaibis ($C \subseteq D$ ir $D \subseteq C$). Įrodykime, pavyzdžiui, distributyvumo dėsnį (7).

Įrodymas. Jei $A \cup (B \cap C) = \emptyset$, tai aišku, kad $\emptyset \subseteq (A \cup B) \cap (A \cup C)$. Tarkime, $x \in A \cup (B \cap C)$ — bet kuris šios aibės elementas. Tada x priklauso bent vienai iš aibių A ir $B \cap C$. Antruoju atveju gauname $x \in B$ ir $x \in C$. Vadinasi, x priklauso bent vienai iš aibių A, B ir tuo pačiu metu priklauso bent vienai iš aibių A, C. Gavome, kad $x \in (A \cup B) \cap (A \cup C)$. Kadangi tai teisinga bet kuriam x, tai $A \cup (B \cap C) \subseteq (A \cup B) \cap (A \cup C)$. Atvirkštinė įdėtis $(A \cup B) \cap (A \cup C) \subseteq A \cup (B \cap C)$ įrodoma, kartojant aukščiau pateiktus samprotavimus atvirkščia tvarka. Formaliai mūsų įrodymą galime užrašyti taip:

$$\begin{aligned} x \in A \cup (B \cap C) &\iff x \in A \quad \text{arba} \quad x \in B \cap C \\ &\iff (x \in A \text{ arba } x \in B) \quad \text{ir} \quad (x \in A \text{ arba } x \in C) \\ &\iff x \in A \cup B \quad \text{ir} \quad x \in A \cup C \\ &\iff x \in (A \cup B) \cap (A \cup C). \end{aligned}$$

□



Pav. 1.1: Veno diagramos.

Nesudėtingas aibių algebros formules patogu patikrinti grafiškai *Veno diagramų* (dar vadinamų *Oilerio skrituliais*) pagalba. Universalią aibę vaizduojame stačiakampiu, o jos poaibius — susikertančiais skrituliais. Pavyzdžiui, aibių lygybę $\overline{A \setminus B} = \overline{A} \cup B$ demonstruoja diagramos pavaizduotos paveikslėlyje 1. Kairiojoje diagramoje užštrichuota aibė $\overline{A \setminus B}$, o dešiniojoje diagramoje aibę $\overline{A} \cup B$ atitinka ta sritis, kuri užštrichuota bent vienu būdu.

Norėdami apibrėžti paskutinę aibių operaciją, pateikiame dar keletą sąvokų.

Bet kokių objektų (t.y., nebūtinai skirtingų) rinkinį vadiname *multiaibe* (vietoje žodžio multiaibė taip pat yra vartojamas terminas *šeima*). Baigtinę multiaibę, turinčią m elementų, vadiname *m-multiaibe*. Sutvarkytą multiaibę vadiname *seka*. Sekoje turi reikšmę ne tik pats objektas, bet ir jo vieta. Sukeitę du skirtingus objektus vietomis, gauname kitą seką. Baigtines sekas iš m objektų dar vadina *m-vektoriais*, o begalines sekas — tiesiog sekomis. m-vektoriaus $\alpha = (\alpha_1, \ldots, \alpha_m)$ i-ąjį elementą α_i vadina vektoriaus α *i-aja koordinate*.

Pavyzdys 1.1.3. $A = \{0, 1\}$ — 2-aibė; $\mathcal{P}(A) = \{\emptyset, \{0\}, \{1\}, A\}$; $\mathcal{A} = \{0, 1, 1, 1\}$ — šeima; $\alpha = (0, 1, 0, 0)$ ir $\beta = (1, 0, 0, 0)$ — du skirtingi 4-vektoriai; $\gamma^\infty = 00110011\ldots$ — begalinė periodinė seka.

Netuščių aibių $A_1, A_2, \ldots, A_m$ *Dekarto sandauga* vadiname aibę

$$A_1 \times \cdots \times A_m = \{\alpha = (\alpha_1, \ldots, \alpha_m) \colon\ \alpha_i \in A_i, i = 1, \ldots, m\}.$$

Jei $A_1 = A_2 = \cdots = A_m = A$, tai gautą aibę žymėsime A^m ir vadinsime aibės A *m-uoju Dekarto laipsniu*, o jos elementus — *m-vektoriais aibėje A*. Vektorius aibėje $\{0, 1\}$ vadiname *dvejetainiais*. Taigi, α ir β iš pavyzdžio 1.3 — dvejetainiai vektoriai ir $\alpha, \beta \in \{0, 1\}^4$. Praplėtę aibės Dekarto laipsnio apibrėžimą, simboliu A^∞ žymėsime aibę begalinių sekų, sudarytų iš aibės A elementų. Jei γ^∞ — seka iš pavyzdžio 1.2, tai $\gamma^\infty \in \{0, 1\}^\infty$.

Pavyzdys 1.1.4. Tegu $V = \{a, b, \ldots, h\}$ (vertikalės), $H = \{1, 2, \ldots, 8\}$ (horizontalės). Tada $L = V \times H = \{a1, a2, \ldots, h8\}$ — šachmatų lentos laukelių aibė (vietoje vektoriaus $(a, 1)$ čia mes naudojame šachmatininkams įprastą žymėjimą $a1$; vėliau mes taip pat dažnai praleisime kablelius ir skliaustus vektorių žymėjime, pvz. $\alpha = 0111$ ir pan.)

Uždaviniai

1. Duotas aibes užrašykite pavidalu $\{x\colon S(x)\}$. Pateikite taip pat induktyvius aibių A ir B apibrėžimus.

   (a) $A = \{1, 3, 5, 7, \dots\}$;

   (b) $B = \{1, 6, 11, 16, \dots, 96\}$;

   (c) $C = \{2, 3, 5, 7, 11, 13, 17, 19, 23, 29, \dots, 997\}$.

2. Duotos aibės $A = \{\text{Jonas}, \text{Petras}\}$ ir $B = \{\emptyset, \{\emptyset\}\}$. Raskite jų poaibių aibes $\mathcal{P}(A)$ ir $\mathcal{P}(B)$.

3. Duotos aibės $A = \{0, 1, 2\}$ ir $B = \{1, 2, 3\}$. Raskite aibes $A \cup B$, $A \cap B$, $A \setminus B$, $B \setminus A$, $\overline{A}$, $\overline{B}$, $\overline{A} \cup \overline{B}$, $\overline{A} \cap \overline{B}$, $\overline{A} \setminus \overline{B}$, $\overline{B} \setminus \overline{A}$ ir $(A \cap \overline{B}) \cap (\overline{A} \cup B)$.

4. Aibę $D = A \cap (\overline{B} \cup C)$ pavaizduokite Veno diagrama.

5. Įrodykite lygybę $(A \setminus B) \cup (B \setminus A) = (A \cup B) \setminus (A \cap B)$: (a) Veno diagramų pagalba; (b) analitiškai.

6. Supaprastinkite reiškinius:

   (a) $((A \cup B) \setminus (A \cup \emptyset)) \setminus B$;

   (b) $(A \cup ((C \setminus B) \cap \emptyset)) \cap (A \cup U)$.

7. Raskite visus trejetainius 2-vektorius, t.y. $\alpha \in \{0, 1, 2\}^2$.

1.2 Funkcijos ir sąryšiai

Atitiktimi tarp netuščių aibių A ir B vadiname bet kokį netuščią poaibį $F \subseteq A \times B$. Atitiktis F — *funkcinė*, jei kiekvienam ($\forall$) $a \in A$ egzistuoja ($\exists$) vienintelis vektorius $(a, b) \in F$. Jei F — funkcinė atitiktis, tai sakome, kad F apibrėžia *funkciją* $f : A \to B$ ir vektoriaus $(a, b) \in F$ koordinatę b žymime $f(a)$. Taigi, funkcija $f : A \to B$ yra atitiktis tarp A ir B, priskirianti kiekvienam aibės A elementui kokį nors aibės B elementą. Aibės B poaibį $f(A) = \{b\colon \exists a \in A$ toks, kad $b = f(a)\}$ vadiname aibės A *vaizdu*. Aibės A poaibį $f^{-1}(b) = \{a\colon f(a) = b\}$ vadiname elemento $b \in B$ *pirmvaizdžiu*. Kai aibės A ir B yra baigtinės, funkciją $f : A \to B$ taip pat vadiname *baigtine*. Jei $A = \{a_1, \dots, a_n\}$, baigtinę funkciją galime vaizduoti jos reikšmių lentele:

x	$f(x)$
a_1	$f(a_1)$
a_2	$f(a_2)$
$\vdots$	$\vdots$
a_n	$f(a_n)$

Funkcijos $f : A \to B$, pasižyminčios svarbiomis savybėmis, turi dar kitus pavadinimus:

- f — *injekcija*, jei $a_1 \neq a_2 \Rightarrow f(a_1) \neq f(a_2)\ \forall a_1, a_2 \in A$;
- f — *surjekcija*, jei $f(A) = B$;
- f — *bijekcija*, jei f — injekcija ir surjekcija kartu.

Bijekciją dar vadina abipus vienareikšmiu atvaizdavimu, kadangi šiuo atveju kiekvienam $b \in B$ egzistuoja vienintelis $a \in A$ toks, kad $f(a) = b$. Pažymėję tokį elementą $a = g(b)$, gauname funkciją $g : B \to A$, kurią vadina *atvirkštine* funkcijai f. Aišku, kad jei A ir B yra baigtinės aibės, o $f : A \to B$ — bijekcija, tai A ir B turi po tiek pat elementų.

Pavyzdys 1.2.1. Tegu

$$A = \{0, 1\}, \quad B = \{a, b, c\}, \quad F_1 = \{(0, a), (1, a)\}, \quad F_2 = \{(0, b), (1, a)\}.$$

1. Tarkime, F_i apibrėžia $f_i : A \to B$. Tada f_2 — injekcija, o f_1 — nei injekcija, nei surjekcija.
2. Tarkime, F_2 apibrėžia $f_2 : A \to B_2 = \{a, b\}$. Tada f_2 — bijekcija.
3. Tarkime, F_1 apibrėžia $f_1 : A \to B_1 = \{a\}$. Tada f_1 — surjekcija.

Jei A ir B — begalinės aibės ir egzistuoja bijekcija $f : A \to B$, tai A ir B vadiname vienodos galios aibėmis ir žymime $|A| = |B|$. Natūrinių skaičių aibės galia žymima "alef-nulinis" raide: $\mathcal{N}_0 = |\mathbb{N}|$ ("alef" yra pirmoji hebrajų abėcėlės raidė). Aibė A vadinama *skaičia*, jei $|A| = |\mathbb{N}|$, t.y. egzistuoja bijekcija $f : A \to \mathbb{N}$. Realiųjų skaičių aibės galia $|\mathbb{R}|$ vadinama *kontinuumu* ir žymima c raide (paprastai yra naudojama gotiška c raidė). Kadangi galima įrodyti, kad skaičiosios aibės poaibių aibės galia sutampa su realiųjų skaičių aibės galia, t.y. $|\mathcal{P}(\mathbb{N})| = |\mathbb{R}|$, tai kontinuumo galia dar žymima $2^{\mathcal{N}_0}$.

Pavyzdys 1.2.2. Sveikųjų skaičių aibė $\mathbb{Z}$ yra skaiti, nes funkcija

$$f(x) = \begin{cases} 2x, & x \geq 0, \\ -2x - 1, & x < 0 \end{cases}$$

yra bijekcija tarp $\mathbb{Z}$ ir $\mathbb{N}$. Funkcija f atvaizduoja neneigiamus sveikuosius skaičius į lyginius natūrinius skaičius, o neigiamus sveikuosius skaičius į nelyginius natūrinius skaičius. Iš šio pavyzdžio matome, kad begalinė aibė gali būti tos pačios galios su savo poaibiu, nes $\mathbb{N} \subset \mathbb{Z}$.

n-*viečiu* (n-nariuoju) *sąryšiu* aibėje A vadiname bet kokį poaibį $\sigma \subseteq A^n$. Pavyzdžiui, Pitagoro skaičių trejetai sudaro trivietį sąryšį $P = \{(x, y, z) \colon x^2 + y^2 = z^2,\ x, y, z \in \mathbb{Z}\}$ sveikųjų skaičių aibėje. Toliau nagrinėsime tik dviviečius (binariuosius) sąryšius, kuriuos vadinsime tiesiog sąryšiais. Pavyzdžiui, kiekvieną funkciją $f : A \to A$ atitinka sąryšis $\{(a, f(a)) \colon a \in A\}$ Jei $(a_i, a_j) \in \sigma$, tai sakome, kad a_i ir a_j *susieti sąryšiu* σ ir žymime $a_i \,\sigma\, a_j$. Sąryšį σ vadiname:

- *refleksyviu,* jei $a \,\sigma\, a \;\forall a \in A$;
- *simetriniu,* jei $a \,\sigma\, b \Rightarrow b \,\sigma\, a \;\forall a, b \in A$;
- *antisimetriniu,* jei $a \,\sigma\, b$ ir $b \,\sigma\, a \Rightarrow a = b \;\forall a, b \in A$;
- *tranzityviu,* jei $a \,\sigma\, b$ ir $b \,\sigma\, c \Rightarrow a \,\sigma\, c \;\forall a, b, c \in A$.

Refleksyvų, tranzityvų ir simetrinį sąryšį vadiname *ekvivalentumo* sąryšiu. Refleksyvų, tranzityvų ir antisimetrinį sąryšį vadiname *tvarkos* sąryšiu ir žymime $\leq$. Aibė A *sutvarkyta,* jei joje apibrėžtas koks nors tvarkos sąryšis. *Griežtos tvarkos* sąryšis $<$ sutvarkytoje aibėje apibrėžiamas taip: $a < b \Leftrightarrow a \leq b$ ir $a \neq b$. Sutvarkytą aibę vadiname *pilnai sutvarkyta,* jei $\forall a, b \in A \; a \leq b$ arba $b \leq a$. Sutvarkytos aibės grafiškai yra vaizduojamos *Hesės diagramomis*.

Pavyzdys 1.2.3. Tegu $A = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$ ir

$$\begin{aligned}
\sigma_1 &= \{(a, a)\colon a \in A\} \quad (A^2 \text{ “įstrižainė”}),\\
\sigma_2 &= \{(a, b)\colon a \leq b \text{ (nelygybė tarp natūrinių skaičių)}\},\\
\sigma_3 &= \{(a, b)\colon b - a \text{ dalinasi iš 3 be liekanos}\},\\
\sigma_4 &= \sigma_2 \cap \sigma_3.
\end{aligned}$$

Nesunku patikrinti, kad σ_1 ir σ_3 — ekvivalentumo sąryšiai, o σ_1, σ_2 ir σ_4 — tvarkos sąryšiai. Iš šių trijų tvarkos sąryšių tik su sąryšiu σ_2 aibė A bus pilnai sutvarkyta.

Pavyzdys 1.2.4. Dvejetainių vektorių aibėje $\{0, 1\}^n$ apibrėžkime sąryšį $\preceq$. Tegu $\alpha = (\alpha_1, \ldots, \alpha_n)$ ir $\beta = (\beta_1, \ldots, \beta_n) \in \{0, 1\}^n$. Sakysime, kad α yra mažesnis arba lygus β ir žymėsime $\alpha \preceq \beta$, jei

$$\alpha_1 \leq \beta_1, \quad \alpha_2 \leq \beta_2, \quad \ldots, \quad \alpha_n \leq \beta_n.$$

Nesunku patikrinti, kad $\preceq$ yra tvarkos sąryšis, taigi aibė $\{0, 1\}^n$ su šiuo sąryšiu yra sutvarkyta. Tačiau ši aibė nėra pilnai sutvarkyta (kai $n \geq 2$), nes, pavyzdžiui, vektoriai $(0, 1)$ ir $(1, 0)$ yra “nepalyginami”.

Rasime objektų skaičių dviejose iš skyrelyje 1.1 apibrėžtų aibių. Priminsime, kad baigtinės aibės A galia $|A|$ vadiname jos elementų skaičių.

Teorema 1.2.1.

1. $|A_1 \times \cdots \times A_k| = |A_1| \times \cdots \times |A_k|$ ($A_1, \ldots A_k$ — *baigtinės netuščios aibės*);
2. $|\mathcal{P}(A)| = 2^{|A|}$ (*aibė* A — *baigtinė*).

Įrodymas. Pirma lygybė akivaizdi, nes k-vektoriaus $\alpha \in A_1 \times \cdots \times A_k$ i-oji koordinatė α_i gali būti bet kuris iš $|A_i|$ aibės A_i elementų ($i = 1, \ldots, k$). Todėl skirtingų k-vektorių bus $|A_1| \times \cdots \times |A_k|$.

Antrą lygybę įrodysime, remdamiesi pirmąja, iš kurios išplaukia, kad $|\{0,1\}^n| = 2^n$. Jei A tuščia, tai $\mathcal{P}(A) = \{\emptyset\}$ ir turime teisingą lygybę $1 = 2^0$. Tarkime, A — netuščia, t.y. $A = \{a_1, \ldots, a_n\}$ ir apibrėžkime funkciją $\chi : \mathcal{P}(A) \to \{0,1\}^n$, kur kiekvienam $B \subseteq A$ $\alpha = \chi(B)$ yra n-vektorius su koordinatėmis

$$\alpha_i = \begin{cases} 1, & a_i \in B; \\ 0, & a_i \notin B. \end{cases}$$

Vektorius $\chi(B)$ dar vadinamas poaibio B *charakteringuoju vektoriumi*. Jis nusako, kurie aibės A elementai priklauso poaibiui B, o kurie ne. Nesunku įsitikinti, kad kiekvieną dvejetainį vektorių $\alpha \in \{0,1\}^n$ atitinka lygiai vienas poaibis $B \in \mathcal{P}(A)$ toks, kad $\alpha = \chi(B)$. Taigi, χ yra bijekcija ir $|\mathcal{P}(A)| = |\{0,1\}^n| = 2^n$. □

Išvada 1.2.1.

$$C_n^0 + C_n^1 + \cdots + C_n^n = 2^n.$$

Įrodymas. Kadangi n-aibė turi lygiai C_n^k skirtingų k-poaibių, tai abi lygybės pusės išreiškia visų skirtingų n-aibės poaibių skaičių. □

Uždaviniai

1. Duota funkcija $f : A \to B$. Rasti aibės $A_1 \subset A$ vaizdą $f(A_1)$, aibės $B_1 \subset B$ pirmvaizdį $f^{-1}(B_1)$ ir funkcijos f reikšmių sritį $R_f = f(A)$. Nustatyti, ar ši funkcija yra injekcija, surjekcija, bijekcija.

   (a) $A = B = \mathbb{N} = \{0, 1, 2, \ldots\}$, $A_1 = B_1 = \{1, 2, 3, 4\}$, $f(x) = x + 1$;

   (b) $A = B = \mathbb{Z} = \{-2, -1, 0, 1, 2, \ldots\}$, $A_1 = B_1 = \{1, 2, 3, 4\}$, $f(x) = x^2$.

2. Sukonstruoti bijekcijas $f : A \to B$:

   (a) $f : \mathbb{Z} \to \mathbb{N}$;

   (b) $f : \mathbb{Z}_2 = \{\ldots, -4, -2, 0, 2, 4, \ldots\} \to \mathbb{N}$;

   (c) $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$;

   (d) $f : \mathbb{R} \to \mathbb{R}^+ = (0, \infty)$;

   (e) $f : [0, 1) \to (\frac{1}{4}, \frac{1}{2}]$.

3. Žmonių aibėje apibrėžti tokie sąryšiai:

   (a) $a\ \sigma_1\ b \Leftrightarrow$ "a ir b yra vienmečiai";

   (b) $a\ \sigma_2\ b \Leftrightarrow$ "a ir b turi tą patį senelį arba tą pačią senelę";

   (c) $a\ \sigma_3\ b \Leftrightarrow$ "a sveria daugiau už b".

Panagrinėkite, kurias iš keturių sąryšių savybių (refleksyvumas, simetriškumas, antisimetriškumas ir tranzityvumas) tenkina kiekvienas iš šių sąryšių. Kurie iš šių sąryšių bus ekvivalentumo sąryšiais?

4. Sveikųjų skaičių aibėje apibrėžti tokie sąryšiai: (1) "$a = b$", (2) "$a > b$", (3) "$a = |b|$", (4) "$a = b+1$", (5) "$a+b \leq 3$" ir (6) "a dalo b". Panagrinėkite, kurias iš trijų sąryšių savybių (refleksyvumas, simetriškumas, antisimetriškumas ir tranzityvumas) tenkina kiekvienas iš šių sąryšių. Kurie iš šių sąryšių bus tvarkos sąryšiais?

2 skyrius

Matematinės logikos pradmenys

Kaip žmonija gauna žinių apie ją dominančius objektus ? Visų pirma, duomenys renkami atliekant bandymus ir stebėjimus, o antra, mąstant ir samprotaujant. Samprotaudami susiduriame su dvejopo pobūdžio žiniomis: vienas jau turime, o kitas išvedame iš jų samprotavimais. Kaip sužinoti, ar teisingai išvedėme ? Dar senovėje žymiausi mąstytojai, ypač Aristotelis IV a. pr. m. e., stengėsi nustatyti tokias samprotavimo taisykles, dėsnius, schemas, kad būtų samprotaujama tik teisingai. Taip atsirado logikos mokslas. XIX amžiuje jisai buvo formalizuotas, buvo pradėti taikyti matematikos metodai, sudaryti logikos nagrinėjamų objektų matematiniai modeliai, ir taip atsirado matematinė logika.

2.1 Teiginiai

Apibrėžimas
Teiginys – *tai sakinys, kuris išreiškia tiesą arba netiesą.*

Pavyzdžiui:

1. Kolumbas atrado Ameriką.
2. Drambliai moka skraidyti.
3. Koks rytoj oras ?
4. Penki daugiau už tris (trumpiau: 5>3).
5. 23+(-48) >0
6. Kitose planetose gyvena protingos būtybės.

Visi šie sakiniai, išskyrus trečią, yra teiginiai. Jie yra arba teisingi, arba ne (bet ne abu iš karto !), nors mes galbūt ir negalim to nustatyti (pavyzdžiui, šeštas sakinys).
Jeigu teiginys išreiškia tiesą, jis vadinamas *teisingu*, jei netiesą – *klaidingu* arba *neteisingu*. Teiginius paprastai žymėsime mažosiomis raidėmis, pavyzdžiui, *p, q, r, s, …* Jei teiginys teisingas, sakome, kad jis įgyja reikšmę *t* (arba 1). Jei neteisingas – reikšmę *k* (arba 0). Taigi, teiginiai įgyja reikšmes iš aibės {*t, k*} (arba {1,0}). Reikšmes *t* ir *k* vadiname *loginėmis konstantomis*.

Matematinė logika nagrinėja, kaip nustatyti teiginio teisingumo reikšmę, kai teiginys yra *sudėtinis*, tai yra sudarytas iš kelių kitų teiginių.

2.2 Loginės operacijos

Iš paprastų teiginių galime sudaryti sudėtinius, naudodami *logines jungtis*, pavyzdžiui, “netiesa, kad…”, “ir”, “arba” ir t.t. Sudėtinio teiginio sudarymas pasinaudojus logine jungtimi vadinamas *logine operacija*.

Išskirsime tokias logines operacijas:

2.2.1 Neigimas

Jei p yra teiginys, tai teiginys "netiesa, kad p" ("ne p") vadinamas teiginio p *neiginiu* ir žymimas $\neg p$. Teiginio p neiginys $\neg p$ yra teisingas tada ir tik tada, kai teiginys p yra neteisingas. Galime sudaryti lentelę, vadinamą *teisingumo reikšmių lentele* (arba tiesiog *teisingumo lentele*, žr. dešinėje).

p	$\neg p$
t	k
k	t

2.2.2 Konjunkcija (loginė daugyba)

Jei p ir q yra teiginiai, tai teiginys "p ir q" vadinamas teiginių p ir q *konjunkcija* (arba *logine sandauga*) ir žymimas "$p\&q$" arba "$p \wedge q$". Teiginių p ir q konjunkcija yra teisingas teiginys tada ir tik tada, kai abu teiginiai p ir q yra teisingi. Teisingumo reikšmių lentelę žr. dešinėje.

p	q	$p\&q$
t	t	t
t	k	k
k	t	k
k	k	k

2.2.3 Disjunkcija (loginė sudėtis)

Jei p ir q yra teiginiai, tai teiginys "p arba q" vadinamas teiginių p ir q *disjunkcija* (arba *logine suma*) ir žymimas "$p \vee q$". Teiginių p ir q disjunkcija yra neteisingas teiginys tada ir tik tada, kai p ir q abu neteisingi. Kitaip sakant, disjunkcija teisinga tada ir tik tada, kai bent vienas iš p ir q yra teisingas. Teisingumo reikšmių lentelę žr. dešinėje.

p	q	$p \vee q$
t	t	t
t	k	t
k	t	t
k	k	k

2.2.4 Implikacija

Jei p ir q yra teiginiai, tai teiginys "jei p, tai q" (arba "iš p išplaukia q") vadinamas teiginių p ir q *implikacija* ir žymimas $p \rightarrow q$ arba $p \Rightarrow q$.

Pavyzdžiui, jei p yra "5>3", o q "5>0", tai $p \rightarrow q$ yra "jei 5>3, tai 5>0".

Implikacija $p \rightarrow q$ yra neteisingas teiginys tada ir tik tada, kai p yra teisingas, o q – neteisingas. Teisingumo reikšmių lentelę žr. dešinėje.

Pastebėkime, kad teiginių $p \rightarrow q$ ir $q \rightarrow p$ reikšmės gali skirtis, pavyzdžiui, "jei dabar saulėta, tai dabar diena" yra teisingas teiginys, o "jei dabar diena, tai dabar saulėta" yra nebūtinai teisingas (gali būti ir apsiniaukę).

p	q	$p \rightarrow q$
t	t	t
t	k	k
k	t	t
k	k	t

2.2.5 Ekvivalencija

Jei p ir q – teiginiai, tai teiginys "p tada ir tik tada, kai q" vadinamas teiginių p ir q *ekvivalencija* (arba *ekvivalentumu*) ir žymimas $p \leftrightarrow q$ arba $p \Leftrightarrow q$. Ekvivalencija $p \leftrightarrow q$ yra teisingas teiginys tada ir tik tada, kai teiginių p ir q teisingumo reikšmės sutampa, t.y., arba jie abu teisingi, arba jie abu klaidingi. Teisingumo reikšmių lentelę žr. dešinėje.

p	q	$p \leftrightarrow q$
t	t	t
t	k	k
k	t	k
k	k	t

2.2.6 Šeferio funkcija

Jei p ir q – teiginiai, tai teiginys "p ir q nesutaikomi" vadinamas teiginių p ir q *Šeferio funkcija* ir žymimas $p|q$ arba p/q (skaitome "p Šeferio funkcija q"). Šeferio funkcija $p|q$ yra neteisingas teiginys tada ir tik tada, kai abu teiginiai p ir q yra teisingi. Teisingumo reikšmių lentelę žr. dešinėje.

p	q	p/q
t	t	k
t	k	t
k	t	t
k	k	t

2.2.7 Griežta disjunkcija (suma moduliu 2)

p	q	$p \oplus q$
t	t	k
t	k	t
k	t	t
k	k	k

Jei p ir q yra teiginiai, tai teiginys "arba p, arba q" vadinamas teiginių p ir q *griežta disjunkcija* (arba *suma moduliu* 2) ir žymimas $p \oplus q$ arba $p+q$. Griežta disjunkcija yra teisingas teiginys tada ir tik tada, kai lygiai vienas iš teiginių p ir q yra teisingas. Teisingumo reikšmių lentelę žr. dešinėje.

p	q	$p \oplus q$
1	1	0
1	0	1
0	1	1
0	0	0

Vadinama "suma moduliu 2", nes jei pažymėtume t vienetu, o k nuliu, tai teisingumo reikšmių lentelė būtų tokia, kokia pavaizduota dešinėje. Tas pačias reikšmes gautume ir sudėję p ir q moduliu 2 (t.y. sudėję p ir q, ir paėmę dalybos iš dviejų liekaną).

2.2.8 Kitos loginės operacijos

p	q	
t	t	t
t	k	t
k	t	k
k	k	t

Galime apibrėžti ir kitas logines jungtis, kurios galbūt neturi pavadinimų ir atitikmenų šnekamojoje kalboje. Pavyzdžiui, tokią, kurios teisingumo reikšmių lentelė yra kaip dešinėje.

Iš viso jų galime apibrėžti tiek, kiek yra skirtingų teisingumo reikšmių lentelių, pavyzdžiui, dvinarėms loginėms jungtims jų yra 2^4=16.

2.3 Formulės

Kaip jau minėjau, teiginius žymime mažosiomis raidėmis *p,q,r,s,…,* ir jungiame juos loginėmis jungtimis. Matematinėje logikoje mums teiginio turinys yra visiškai nesvarbus. Mums svarbu tik tai, ar jis teisingas, ar ne. Todėl nuo šiol nebesigilinsime į teiginių turinį, o nagrinėsime juos kaip kažkokius kintamuosius (vadinsime juos *loginiais kintamaisiais),* galinčius įgyti reikšmes *t* arba *k,* ir iš jų sudarinėsime sudėtingesnius teiginius, naudodami logines jungtis.

Apibrėžimas

Loginėmis formulėmis *vadinami reiškiniai, gauti baigtinį skaičių kartų pavartojus loginių operacijų ženklus bei skliaustus raidžių jungimui. Formaliai tai galima apibrėžti taip:*

a) *loginės konstantos k ir t yra loginės formulės;*
b) *loginiai kintamieji (p,q,r,s,…) yra loginės formulės;*
c) *jei Q yra loginė formulė, tai (¬Q) taip pat yra loginė formulė;*
d) *jei P ir Q yra loginės formulės, tai (PΔQ) yra loginė formulė, kur Δ žymi bet kurią dvinarę loginę jungtį (pavyzdžiui,* ∨, &, →, ↔, / , ⊕ *,…).*

Pastaba

Kad supaprastintume formulių užrašymą, mes įvedame kai kurių loginių jungčių atlikimo tvarką (¬ ,&,∨, kitos), ir daug kur skliaustus praleidžiame.

Pavyzdys

Reiškiniai $p \rightarrow q$, $\neg q$, $\neg q \rightarrow r$, $(p \& q) \vee r$, $\neg(p \leftrightarrow q)$, $((p \rightarrow q) \& (\neg q \rightarrow r)) \rightarrow (q \vee \neg q)$ yra loginės formulės. Pagal loginės formulės apibrėžimą turėtume rašyti $((\neg q) \rightarrow r)$, bet rašome tiesiog $\neg q \rightarrow r$, nes susitarėme, kad bet kuriuo atveju neigimas bus skaičiuojamas prieš implikaciją. Taip pat galime rašyti $p \& q \vee r$ vietoj $(p \& q) \vee r$, nes vis tiek pirma atliekame konjunkciją, o tik paskui disjunkciją.

Žinodami, kokias reikšmes įgyja formules sudarantys teiginiai (t.y., loginiai kintamieji), galime apskaičiuoti formulės reikšmę. Apskaičiuodami naudojamės loginių operacijų teisingumo reikšmių lentelėmis. Taigi, į formulę galime žiūrėti kaip į funkciją, kurios argumentai įgyja reikšmes iš aibės {*k*, *t*} (arba {0, 1}), ir kuri įgyja reikšmes iš tos pačios aibės.

Pavyzdys

Sudarysime formulės $Q(p, q) := \neg p \rightarrow (q \& \neg (q \vee p))$ teisingumo reikšmių lentelę (čia ir toliau ‘:=’ yra pažymėjimo ženklas: dešinėje jo pusėje esantį reiškinį pažymime kairėje pusėje esančiu simboliu):

p	q	$\neg p$	$q \vee p$	$\neg(q \vee p)$	$q \& \neg(q \vee p)$	$Q(p,q)$
t	*t*	*k*	*t*	*k*	*k*	*t*
t	*k*	*k*	*t*	*k*	*k*	*t*
k	*t*	*t*	*t*	*k*	*k*	*k*
k	*k*	*t*	*k*	*t*	*k*	*k*

Jei į formulę įeina *n* loginių kintamųjų, tai teisingumo lentelė turės 2^n eilučių.

Jei formulė *A* priklauso nuo kintamųjų $p_1, p_2, \ldots, p_n$, tai ją žymėsime $A(p_1, p_2, \ldots, p_n)$.

Apibrėžimas

Tegu $A(p_1, p_2, \ldots, p_n)$ yra loginė formulė. Tada konkretus kintamųjų $p_1, p_2, \ldots, p_n$ reikšmių rinkinys vadinamas formulės A interpretacija.

Pavyzdys

$p = t, q = k$ yra formulės *Q* interpretacija. Formulė *Q* su šia interpretacija įgyja reikšmę *t*.

Matome, kad kiekvieną interpretaciją atitinka viena teisingumo reikšmių lentelės eilutė, ir atvirkščiai. Todėl, jeigu formulė priklauso nuo *n* kintamųjų, tai yra 2^n skirtingų jos interpretacijų.

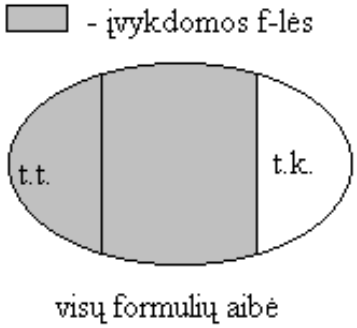
Apibrėžimas

Formulė vadinama tapačiai teisinga *(arba* tautologija*), jei ji teisinga su bet kuria interpretacija.*

Formulė vadinama tapačiai klaidinga *(arba* prieštara, *arba* neįvykdoma*), jeigu ji klaidinga su bet kuria interpretacija.*

Formulė vadinama įvykdoma, *jei atsiras interpretacija, su kuria jinai teisinga.*

Tapačiai teisingų (t.t.), tapačiai klaidingų (t.k.) ir įvykdomų formulių santykį galima pavaizduoti tokia diagrama:



Pavyzdys

Sudarę teisingumo reikšmių lenteles, įsitikinsite, kad formulės $p\to(q\to p)$ ir $\neg\neg p\leftrightarrow p$ yra tapačiai teisingos (t.t.), $p\&\neg p$ yra tapačiai klaidinga (t.k.), o formulė $p\to\neg p$ yra įvykdoma (jinai yra teisinga, kai $p=k$).

Akivaizdu, kad jei formulė A yra tapačiai teisinga, tai $\neg A$ yra tapačiai klaidinga, ir atvirkščiai. Pavyzdžiui, jei A priklauso nuo kintamųjų p ir q, žr. lentelę dešinėje.

p	q	A	$\neg A$
t	t	t	k
t	k	t	k
k	t	t	k
k	k	t	k

Pavyzdys

Nesunkiai įsitikinsite, kad $\neg(p\to(q\to p))$ yra tapačiai klaidinga.

Norint patikrinti, ar formulė $A(p_1, p_2, \ldots, p_n)$ yra tapačiai teisinga, pakanka sudaryti jos teisingumo reikšmių lentelę ir pažiūrėti, ar paskutiniame stulpelyje yra vien tik reikšmės t. Bet, sudarant teisingumo reikšmių lentelę, reikia apskaičiuoti formulės A reikšmę 2^n interpretacijų. Tai labai greitai auganti funkcija, todėl praktikoje naudoti teisingumo reikšmių lentelę formulės tapataus teisingumo nustatymui galima tik nedideliems n.

Atrodytų, kad patikrinti, ar formulė yra įvykdoma, yra daug lengviau negu patikrinti jos tapatų teisingumą, nes užtenka surasti vieną reikšmę t. Bet taip nėra. Iš tikro, tarkime, kad turime algoritmą, kuris bet kurią formulę A patikrina, ar jinai įvykdoma, ar ne. Bet žinome, kad formulė A yra tapačiai teisinga tada ir tik tada, kai $\neg A$ yra tapačiai klaidinga, t.y. kai $\neg A$ neįvykdoma. Taigi, pritaikome tą algoritmą formulei $\neg A$. Jei atsakymas, kad $\neg A$ įvykdoma, tai A nėra tapačiai teisinga, o jei ne, tai A yra tapačiai teisinga.

Apibrėžimas

Dvi loginės formulės A ir B vadinamos logiškai ekvivalenčiomis *(arba tiesiog* ekvivalenčiomis*), jei su bet kuria interpretacija jos įgyja tą pačią reikšmę. Žymime $A \sim B$ (arba $A \equiv B$).*

Pavyzdys

Nesunkiai patikrinsite, kad $p\&q \sim q\&p$, $p\&(q\&r) \sim (p\&q)\&r$, $p\vee(q\&\neg q) \sim \neg\neg p\vee(r\&\neg r)$, $p\to p \sim q\to q$. Žr., pavyzdžiui, lentelę:

p	q	$p\to p$	$q\to q$
t	t	t	t
t	k	t	t
k	t	t	t
k	k	t	t

p	q	t
t	t	t
t	k	t
k	t	t
k	k	t

Taigi, formulė A yra tapačiai teisinga, jei ji ekvivalenti loginei konstantai t, t.y. $A \sim t$. Iš tikro, loginė konstanta t yra loginė formulė, kuri su bet kuria interpretacija įgyja reikšmę t, pavyzdžiui, kaip kairėje.

Analogiškai, formulė A yra tapačiai klaidinga, kai jinai ekvivalenti loginei konstantai k. Be to, bet kurios dvi tapačiai teisingos formulės yra ekvivalenčios, ir bet kurios dvi tapačiai klaidingos formulės taip pat yra ekvivalenčios.

Atkreipsiu jūsų dėmesį į skirtumą tarp žymėjimų "$\leftrightarrow$" ir "$\sim$". Turėkite omenyje, kad "$\leftrightarrow$" – tai loginė operacija, kuri sujungia formules A ir B į sudėtinę formulę $A\leftrightarrow B$, o "$A \sim B$" tiesiog reiškia, kad formulės A ir B ekvivalenčios, t.y. jų reikšmės vienodos su bet kuria interpretacija. Bet tarp šių žymėjimų yra aiškus ryšys, kurį rodo šis teiginys.

Teiginys

Loginės formulės A ir B yra logiškai ekvivalenčios tada ir tik tada, kai loginė formulė $A\leftrightarrow B$ yra tapačiai teisinga.

Įrodymas. Prisiminkime, kad formulės $A\leftrightarrow B$ teisingumo reikšmių lentelė yra tokia, kaip parodyta dešinėje.

A	B	$A \leftrightarrow B$
t	t	t
t	k	k
k	t	k
k	k	t

Pagal formulės $A\leftrightarrow B$ teisingumo reikšmių lentelę, $A\leftrightarrow B$ yra teisinga tada ir tik tada, kai formulių A ir B reikšmės sutampa. Todėl $A\leftrightarrow B$ yra tapačiai teisinga tada ir tik tada, kai A ir B reikšmės sutampa su kiekviena kintamųjų interpretacija, t.y. tada ir tik tada, kai $A \sim B$. □

Šitą teiginį galime užrašyti taip: $A \sim B$ tada ir tik tada, kai $A\leftrightarrow B \sim t$.

Pavyzdys

$p\&q \sim q\&p$, todėl $(p\&q)\leftrightarrow(q\&p)$ yra tapačiai teisinga formulė, t.y. $(p\&q)\leftrightarrow(q\&p) \sim t$. Ir atvirkščiai, pavyzdžiui, žinome, kad $\neg\neg p\leftrightarrow p \sim t$, todėl $\neg\neg p \sim p$.

2.4 Logikos dėsniai

Apibrėžimas

Logikos dėsniu *vadiname tapačiai teisingą formulę*.

Logikos dėsniai naudojami samprotaujant. Kelis logikos dėsnius (t.y., tapačiai teisingas formules) jau sutikome, pavyzdžiui, $p\rightarrow(q\rightarrow p)$. Be to, kaip matėme ką tik įrodytame teiginyje, logikos dėsnį gauname iš kiekvienos ekvivalenčių formulių poros: jei $A \sim B$, tai formulė $A\leftrightarrow B$ yra logikos dėsnis. Pavyzdžiui, matėme, kad $p\&q \sim q\&p$. Taigi, $(p\&q)\leftrightarrow(q\&p)$ yra logikos dėsnis. Kadangi tai tokios artimos sąvokos, tai ekvivalenčių formulių porą irgi vadinsime *logikos dėsniu*, pavyzdžiui, $p\&q \sim q\&p$ vadinsime logikos dėsniu.

Kai kurie logikos dėsniai vartojami dažniau, kiti – rečiau. Dabar susipažinsime su svarbesniais dėsniais. Visų jų įrodymas labai paprastas. Tiesiog sudarome jų teisingumo reikšmių lenteles, ir patikriname, kad tai tikrai ekvivalenčios formulės.

Neigimo savybė:		
1. $\neg\neg p \sim p$		dvigubo neigimo dėsnis
Konjunkcijos savybės:	Disjunkcijos savybės:	
2. *a)* $p\&p \sim p$	*b)* $p\vee p \sim p$	idempotencijos dėsniai
3. *a)* $p\&\neg p \sim k$ (prieštaravimo dėsnis)	*b)* $p\vee\neg p \sim t$ (negalimo trečiojo dėsnis)	
4. *a)* $p\&q \sim q\&p$	*b)* $p\vee q \sim q\vee p$	komutatyvumo dėsniai
5. *a)* $(p\&q)\&r \sim p\&(q\&r)$	*b)* $(p\vee q)\vee r \sim p\vee(q\vee r)$	asociatyvumo dėsniai
Kiti dėsniai:		
6. *a)* $(p\vee q)\&r \sim (p\&r)\vee(q\&r)$	*b)* $(p\&q)\vee r \sim (p\vee r)\&(q\vee r)$	distributyvumo dėsniai
7. *a)* $\neg(p\&q) \sim \neg p\vee\neg q$	*b)* $\neg(p\vee q) \sim \neg p\&\neg q$	De Morgano dėsniai
8. *a)* $(p\&q)\vee p \sim p$	*b)* $(p\vee q)\&p \sim p$	absorbavimo dėsniai
9. $p\rightarrow q \sim \neg p\vee q$		implikacijos pašalinimo dėsnis
10. $p\leftrightarrow q \sim (p\rightarrow q)\&(q\rightarrow p)$		ekvivalencijos pašal. d.
11. $p/q \sim \neg(p\&q)$		Šeferio f-jos pašal. d.
12. $p\oplus q \sim \neg(p\leftrightarrow q)$		griežtos disj. pašal. d.

Pavyzdys

Sudarykime 8a) dėsnio teisingumo reikšmių lentelę ir įsitikinkime, kad iš tikrųjų tai yra logikos dėsnis, t.y. ekvivalenčių formulių pora (žr. dešinėje).

p	q	$p\&q$	$(p\&q)\vee p$
t	t	t	t
t	k	k	t
k	t	k	k
k	k	k	k

Iš komutatyvumo ir asociatyvumo dėsnių (ketvirtas ir penktas loginiai dėsniai) matome, kad formulėje, į kurią įeina vien kintamieji, skliaustai ir, pavyzdžiui, disjunkcijos ženklai, galima kintamuosius keisti vietomis, bet kaip juos suskliausti arba ir visai neskliausti – visos taip gautos formulės bus ekvivalenčios. Pavyzdžiui, $(p\vee q)\vee r \sim q\vee(p\vee r) \sim p\vee(q\vee r) \sim \ldots \sim p\vee q\vee r$ (jei skliaustų nėra, operacijas atliekame jų užrašymo tvarka). Tas pats galioja ir konjunkcijai, t.y. veiksmų atlikimo tvarka neturi reikšmės, vis tiek gausime tą patį rezultatą.

Atkreipkite dėmesį, kad distributyvumo dėsnis $(p\vee q)\&r \sim (p\&r)\vee(q\&r)$ labai primena gerai jums žinomą aritmetikos taisyklę $(p+q)r = pr + qr$, užtenka pakeisti disjunkciją "$\vee$" į sudėtį "+", ir konjunkciją "&" į daugybą "$\cdot$". Bet logikoje galima dar daugiau! Logikoje galioja dar vienas distributyvumo dėsnis: $(p\&q)\vee r \sim (p\vee r)\&(q\vee r)$, kurio atitikmuo $pq+r = (p+r)(q+r)$ negalioja aritmetikoje!

Distributyvumo dėsnius galima apibendrinti didesniam narių skaičiui, t.y.

$(p_1\vee p_2\vee\ldots\vee p_n)\&q \sim (p_1\&q)\vee(p_2\&q)\vee\ldots\vee(p_n\&q)$,

$(p_1\& p_2\&\ldots\&p_n)\vee q \sim (p_1\vee q)\&(p_2\vee q)\&\ldots\&(p_n\vee q)$.

De Morgano dėsnius taip pat galime apibendrinti:

$\neg(p_1\vee p_2\vee\ldots\vee p_n) \sim \neg p_1\&\neg p_2\&\ldots\&\neg p_n$,

$\neg(p_1\& p_2\&\ldots\& p_n) \sim \neg p_1\vee\neg p_2\vee\ldots\vee\neg p_n$.

1 pastaba

Loginiai dėsniai tebegalios, jei pakeisime kintamuosius bet kokiomis formulėmis.

Iš tikro, turime tokį teiginį:

1 teiginys

Tarkime, $A(p_1,\ldots,p_n)$ yra tapačiai teisinga formulė, o $B_1,\ldots,B_n$ yra bet kokios formulės. Tada $A(B_1,\ldots,B_n)$ yra tapačiai teisinga.

Įrodymas. Imkime bet kurias formulių $B_1,\ldots,B_n$ interpretacijas. Tarkime, kad su tomis interpretacijomis B_1 įgyja reikšmę α_1, B_2 įgyja reikšmę $\alpha_2,\ldots$, B_n – reikšmę α_n, kur $\alpha_i\in\{t, k\}$ $(1\le i\le n)$. Tada formulės $A(B_1,\ldots,B_n)$ reikšmė su tomis formulių $B_1,\ldots,B_n$ interpretacijomis bus lygi reikšmei $A(\alpha_1,\ldots,\alpha_n)$. Tačiau formulė $A(p_1,\ldots,p_n)$ yra tapačiai teisinga, todėl $A(\alpha_1,\ldots,\alpha_n)$ yra t. Parodėme, kad su bet kokia interpretacija formulė $A(B_1,\ldots,B_n)$ yra teisinga, o tai reiškia, kad $A(B_1,\ldots,B_n)$ yra tapačiai teisinga formulė. □

Išvada

Jei $A(p_1,\ldots,p_n) \sim B(p_1,\ldots,p_n)$, o $C_1,\ldots,C_n$ yra bet kokios formulės, tai $A(C_1,\ldots,C_n) \sim B(C_1,\ldots,C_n)$.

Įrodymas. Jei $A(p_1,\ldots,p_n) \sim B(p_1,\ldots,p_n)$, tai $A(p_1,\ldots,p_n) \leftrightarrow B(p_1,\ldots,p_n)$ yra tapačiai teisinga formulė, todėl, pagal pirmą teiginį, $A(C_1,\ldots,C_n) \leftrightarrow B(C_1,\ldots,C_n)$ yra tapačiai teisinga formulė ir $A(C_1,\ldots,C_n) \sim B(C_1,\ldots,C_n)$. □

Ši išvada leidžia naudotis išvardintais loginiais dėsniais ne tik tokia forma, kokia jie užrašyti, bet ir įstačius sudėtingesnes formules vietoj kintamųjų.

Pavyzdys

Absorbavimo dėsnis ne tik parodo, kad formulės $(p \vee q)\&p$ ir p yra ekvivalenčios, bet kad ekvivalenčios ir formulės $(P \vee Q)\&P$ bei P, kur P ir Q – bet kokios formulės. Todėl, pavyzdžiui, $((p \to q) \vee (r/s))\&(p \to q) \sim p \to q$ (čia įstatėme $p \to q$ vietoj P ir r/s vietoj Q).

Analogiškai, jei į De Morgano dėsnį $\neg(p\&q) \sim \neg p \vee \neg q$ vietoj kintamojo p įstatysime formulę $p/\neg r$, o vietoj q įstatysime $r \to q$, tai gausime logikos dėsnį

$$\neg((p/\neg r)\&(r \to q)) \sim \neg(p/\neg r) \vee \neg(r \to q).$$

2 pastaba

Ekvivalenčias formules gausime ir pakeitę dalį formulės jai ekvivalenčia formule.

Iš tikro, tarkime, A, B ir C yra bet kurios formulės, ir formulė B yra formulės A sudėtyje, t.y. B yra formulės A dalis.

Pavyzdys

Formulė $p \vee \neg q$ yra formulės $r\&(p \vee \neg q) \to q$ sudėtyje.

Beje, formulė B gali kelis kartus pasikartoti formulėje A. Formulę, kuri gaunama iš formulės A, pakeitus joje kurią nors formulę B formule C, žymėsime $A[B,C]$.

Pavyzdys

Jei A yra $r\&(p \vee \neg q) \to q$, B yra $p \vee \neg q$, C yra $\neg p \to \neg q$, tai $A[B,C]$ yra $r\&(\neg p \to \neg q) \to q$.

2 teiginys

Tarkime, B ir C yra ekvivalenčios formulės. Tada formulės $A[B,C]$ ir A irgi yra ekvivalenčios.

Įrodymas. Reikia įrodyti, kad formulių $A[B,C]$ ir A reikšmės su visomis interpretacijomis sutampa, t.y. kad jų teisingumo reikšmių lentelių paskutiniai stulpeliai sutampa. Sudarykime formulės A teisingumo reikšmių lentelę taip, kad iš pradžių joje būtų apskaičiuojamos formulės B reikšmės (įskaitant formules, įeinančias į B). Analogiškai sudarykime formulės $A[B,C]$ teisingumo lentelę taip, kad iš pradžių būtų apskaičiuojamos formulės C reikšmės. Tada iki stulpelių B ir C formulių A ir $A[B,C]$ teisingumo lentelių reikšmės gali skirtis, bet stulpeliuose B ir C jos pasidarys lygios, nes B ir C yra ekvivalenčios formulės. Tolesniuose stulpeliuose reikšmės taip pat liks lygios, nes formulės A ir $A[B,C]$ daugiau niekuo nesiskiria, todėl paskutiniai stulpeliai irgi sutaps. □

Pavyzdys

Tegu A yra formulė $[(p \to q) \vee (q \oplus p)] / (p \to q)$, B yra $p \to q$, ir C yra $\neg p \vee q$. Tada formulė $A[B,C]$ gali būti tokia: $[(\neg p \vee q) \vee (q \oplus p)] / (p \to q)$. Formulių A ir $A[B,C]$ teisingumo reikšmių lenteles galima skaičiuoti taip:

p	q	B	$q \oplus p$	$B \vee (q \oplus p)$	A
t	t	t	k	t	k
t	k	k	t	t	t
k	t	t	t	t	k
k	k	t	k	t	k

p	q	$\neg p$	C	$q \oplus p$	$C \vee (q \oplus p)$	$A[B,C]$
t	t	k	t	k	t	k
t	k	k	k	t	t	t
k	t	t	t	t	t	k
k	k	t	t	k	t	k

Taigi, skaičiuodami formulės A teisingumo reikšmių lentelę, iš pradžių apskaičiuojame formulę B, o paskui kitas formulės A dalis bet kuria tvarka. Skaičiuodami $A[B,C]$, iš pradžių apskaičiuojame formulę C, o paskui kitas formulės $A[B,C]$ dalis ta pačia tvarka, kaip ir formulei A. Matome, kad, kadangi formulės B ir C ekvivalenčios, tai stulpeliai B ir C, o ir

visi toliau einantys stulpeliai, sutampa. Todėl $[(p\to q)\ \vee\ (q\oplus p)]\ /\ (p\to q) \sim [(\neg p \vee q)\ \vee\ (q\oplus p)]\ /\ (p\to q)$.

Pavyzdys

Kadangi $p\to q \sim \neg p \vee q$, tai formulėje $[(p\to q)\ \vee (q\oplus p)]\ /\ (p\to q)$ galime pakeisti formulę $p\to q$ formule $\neg p \vee q$, taip gaudami formulę $[(\neg p \vee q)\ \vee (q\oplus p)]\ /\ (p\to q)$. Todėl $[(p\to q)\ \vee (q\oplus p)]\ /\ (p\to q) \sim [(\neg p \vee q)\ \vee (q\oplus p)]\ /\ (p\to q)$.

Naudojantis turimais logikos dėsniais ir ką tik gautais rezultatais, galima išvesti naujus logikos dėsnius. Antra pastaba tvirtina, kad pakeitę bet kurią formulę, įeinančią į formulę *A*, jai ekvivalenčia formule, gausime formulei *A* ekvivalenčią formulę. O pirmoji sako, kad galime naudoti logikos dėsnius, įstatę bet kokias formules vietoj kintamųjų.

Pavyzdys

$(p\to q)/r \sim$	implikacijos pašalinimo dėsnis ir 2 pastaba
$(\neg p \vee q)/r \sim$	Šeferio f-jos pašalinimo dėsnis ir 1 pastaba
$\neg[(\neg p \vee q)\&r] \sim$	De Morgano dėsnis ir 1 pastaba
$\neg(\neg p \vee q) \vee \neg r \sim$	De Morgano dėsnis ir 1 bei 2 pastabos
$(\neg(\neg p)\&\neg q) \vee \neg r \sim$	dvigubo neigimo dėsnis ir 2 pastaba
$(p\&\neg q) \vee \neg r$	

2.5 Esminiai ir fiktyvūs kintamieji

Apibrėžimas.

Sakysime, kad loginis kintamasis p_i loginėje formulėje $Q(p_1, p_2,\ldots,p_n)$ yra esminis, *jei galime rasti tokias reikšmes $\alpha_1, \alpha_2,\ldots,\alpha_{i-1}, \alpha_{i+1}, \ldots, \alpha_n \in\{t,k\}$, kad $Q(\alpha_1, \alpha_2,\ldots\alpha_{i-1}, t, \alpha_{i+1}, \ldots, \alpha_n) \neq Q(\alpha_1, \alpha_2,\ldots\alpha_{i-1}, k, \alpha_{i+1}, \ldots, \alpha_n)$. Priešingu atveju kintamasis vadinamas* fiktyviu *formulėje Q.*

Komentaras

Taigi, jei, esant kažkokioms fiksuotoms kitų kintamųjų reikšmėms, kintamojo p_i reikšmės pakeitimas iš *t* į *k* pakeičia formulės reikšmę, tai kintamasis p_i yra esminis. Jei kintamojo p_i reikšmės pakeitimas iš *t* į *k* nepakeičia formulės reikšmės, kad ir kokios būtų fiksuotos kitų kintamųjų reikšmės, tai kintamasis p_i vadinamas fiktyviu. Kaip matysime, tokiu atveju jis į formulę įeina fiktyviai, t.y. egzistuoja ekvivalenti formulė, į kurią kintamasis p_i neįeina.

Pavyzdys

Nustatysime, kurie loginiai kintamieji yra fiktyvūs, o kurie esminiai formulėje $Q(p,q,r) = (\neg p\&\neg q\&r) \vee (\neg p\&q\&r)$.

Sprendimas būtų toks. Reikia įsistatyti visus galimus kintamųjų reikšmių rinkinius ir pažiūrėti, ar formulės reikšmė priklauso nuo kintamojo reikšmės pakeitimo. Kad būtų paprasčiau, pirmiausia sudarysim formulės teisingumo reikšmių lentelę:

p	*q*	*r*	$\neg p$	$\neg q$	$\neg p\&\neg q\&r$	$\neg p\&q\&r$	$Q(p,q,r)$
t	*t*	*t*	*k*	*k*	*k*	*k*	*k*
t	*t*	*k*	*k*	*k*	*k*	*k*	*k*
t	*k*	*t*	*k*	*t*	*k*	*k*	*k*

t	*k*	*k*	*k*	*t*	*k*	*k*	*k*
k	*t*	*t*	*t*	*k*	*k*	*t*	*t*
k	*t*	*k*	*t*	*k*	*k*	*k*	*k*
k	*k*	*t*	*t*	*t*	*t*	*k*	*t*
k	*k*	*k*	*t*	*t*	*k*	*k*	*k*

Iš pirmos ir penktos eilučių matome, kad kintamasis *p* yra esminis. Iš tikro, pirmoje ir penktoje eilutėse kitų kintamųjų (*q* ir *r*) reikšmės nesikeičia (ir yra lygios atitinkamai *t* ir *t*), keičiasi tik kintamojo *p* reikšmė (*t* pirmoje eilutėje, *k* penktoje). Ir formulės reikšmė taip pat keičiasi ! Taigi, formulė *Q(p,q,r)* nuo kintamojo *p* priklauso iš esmės, jis yra esminis.

Kintamasis *q* yra fiktyvus. Iš tikro, imkim pirmą ir trečią lentelės eilutes. Jose skiriasi tik kintamojo *q* reikšmės (*t* pirmoje eilutėje, *k* trečioje), o kitų kintamųjų reikšmės lieka tos pačios (*t*). O formulės reikšmė nesikeičia – išlieka *k*. Bet to dar neužtenka, kad galėtume tvirtinti, kad *q* yra fiktyvus, nes mes patikrinome tik vieną porą interpretacijų. Galbūt kokioje kitoje poroje, kurioje keisis vien tik kintamojo *q* reikšmė, formulės reikšmė irgi keisis. Taigi, turime patikrinti visas galimas poras. Taip ir padarome: antroje ir ketvirtoje eilutėse keičiasi vien tik kintamojo *q* reikšmė, o formulės reikšmė nesikeičia; tas pats ir penktoje – septintoje, ir šeštoje – aštuntoje eilutėse. Patikrinom visas galimas interpretacijų poras, ir niekur formulės reikšmė nesikeičia, keičiantis vien tik kintamojo *q* reikšmei. Taigi, kintamasis *q* iš tikro yra fiktyvus.

Belieka patikrinti kintamąjį *r*. Pirma – antra eilutės: keičiasi vien tik kintamojo *r* reikšmė, o formulės reikšmė nesikeičia; trečia – ketvirta: nesikeičia; penkta – šešta: keičiasi ! Taigi, kintamasis *r* yra esminis. □

Jeigu kintamasis yra fiktyvus duotoje formulėje, tai mes galime rasti ekvivalenčią formulę, į kurią tas kintamasis neįeina.

Pavyzdys

Imkime tą pačią formulę $Q(p, q, r) \sim (\neg p \& \neg q \& r) \vee (\neg p \& q \& r)$. Pasinaudodami loginiais dėsniais, galime ją suprastinti: $Q(p, q, r) \sim [(\neg p \& r) \& \neg q] \vee [(\neg p \& r) \& q] \sim (\neg p \& r) \& (\neg q \vee q) \sim (\neg p \& r) \& t \sim \neg p \& r \sim P(p,r)$. Gavome formulę *P(p,r)*, ekvivalenčią formulei *Q(p, q, r)*, į kurią neįeina kintamasis *q*. □

Naudojantis logikos dėsniais gali būti sudėtinga pašalinti fiktyvius kintamuosius iš formulės. Parodysime, kaip galima tai padaryti naudojantis teisingumo reikšmių lentele.

Pavyzdys

Vėlgi imkime tą patį pavyzdį. Nustatėme, kad kintamasis q yra fiktyvus. Kaip jį pašalinti iš formulės ? Bandykime perdirbti teisingumo reikšmių lentelę taip, kad jinai nebepriklausytų nuo kintamojo q. Kadangi kintamasis q yra fiktyvus, tai iš teisingumo reikšmių lentelės galime išmesti stulpelį su q reikšmėmis. Palikime tik stulpelius su esminiais kintamaisiais p ir r bei stulpelį su formulės *Q(p,q,r)* reikšmėmis (žr. dešinėje).

p	*r*	*Q(p,q,r)*
t	*t*	*k*
t	*k*	*k*
t	*t*	*k*
t	*k*	*k*
k	*t*	*t*
k	*k*	*k*
k	*t*	*t*
k	*k*	*k*

Kadangi liko tik du kintamieji, lentelėje turėtų būti tik 4 eilutės. Iš tikro, lengva pastebėti, kad kai kurios lentelės eilutės kartojasi (taip yra dėl to, kad q yra fiktyvus – pakeitus vien tik jo reikšmę, formulės reikšmė išlieka ta pati, todėl ta eilučių pora skiriasi vien tik kintamojo q reikšme, o išmetus stulpelį su kintamojo q reikšmėmis ir iš viso nebesiskiria). Išmetame

besikartojančias eilutes ir gauname lentelę iš keturių eilučių (žr. dešinėje). Ši lentelė vaizduoja funkciją *P(p,r)*, priklausančią tik nuo kintamųjų p ir r, ir jos reikšmės duotoms kintamųjų p, q ir r reikšmėms sutampa su formulės *Q(p,q,r)* reikšmėmis, tai yra *Q(p,q,r) ~ P(p,r)*. Taigi, belieka tik užrašyti formulę *P(p,r)*, kurios teisingumo reikšmių lentelę radome, ir turėsime formulę, ekvivalenčią duotai formulei, kurioje nebėra fiktyvių kintamųjų. O kaip užrašyti formulę iš duotos teisingumo reikšmių lentelės, mes išmoksime netrukus, kai nagrinėsime normaliąsias formas. □

p	*r*	*P(p,r)*
t	*t*	*k*
t	*k*	*k*
k	*t*	*t*
k	*k*	*k*

Pastabos

1. Jei formulė yra tapačiai teisinga arba tapačiai klaidinga, tai visi į ją įeinantys kintamieji yra fiktyvūs.
2. Lengva patikrinti, kad jei formulės $Q(p_1,\dots p_k\ ,\ p_{k+1},\dots p_n)$ ir $P(p_1,\ p_2,\dots p_k)$ yra ekvivalenčios, tai kintamieji $p_{k+1},\dots p_n$ yra fiktyvūs formulėje *Q*.

2.6 Kontaktinės schemos

Vienas akivaizdžiausių matematinės logikos pritaikymų yra elektros grandinėse.

Kontaktinis elementas (pavyzdžiui, jungiklis arba relė), žymimas  , gali būti dvejose būsenose:

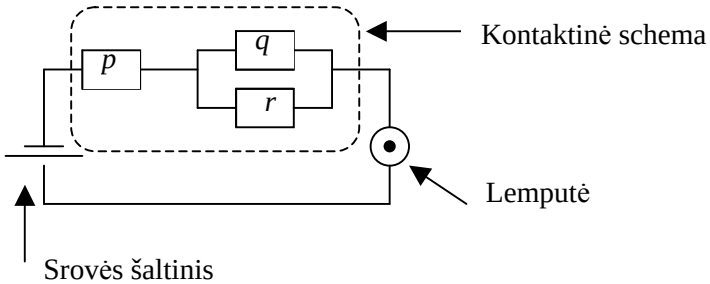
1) jis *praleidžia* elektros srovę, kontaktai sujungti, sakome, kad jis *įjungtas*. Žymėjimas:

2) kai kontaktai nesujungti, elementas *išjungtas*, jungiklis elektros srovės *nepraleidžia*. Žymėjimas:

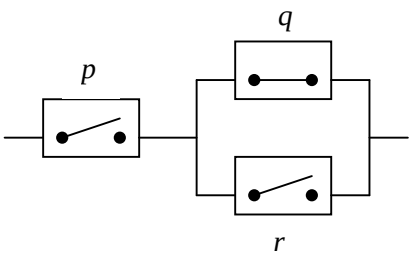

Kontaktinė schema – tai sujungtų kontaktinių elementų rinkinys su dviem išėjimais į išorę.

Pavyzdys

Čia jūs matote elektros grandinę, kurią sudaro šaltinis, lemputė ir kontaktinė schema:



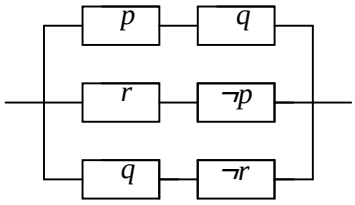
Kiekvieną kontaktinį elementą žymime raide. Jei, pavyzdžiui, elementai *p* ir *r* yra išjungti, o *q* – įjungtas, tai kontaktinė schema bus tokia:



Jinai srovės nepraleis, nes elementas p yra išjungtas.

Galima kelis elementus pažymėti ta pačia raide. Tai reiškia, kad jie išjungti ar įjungti vienu metu. Taip pat galima kontaktinius elementus žymėti ir raide su neiginiu (pavyzdžiui, $\neg p$). Tai reiškia, kad toks elementas bus išjungtas tada ir tik tada, kai elementas p bus įjungtas.

Pavyzdys



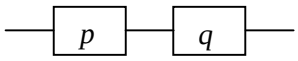
Ši schema praleis srovę, jei kontaktiniai elementai p ir q įjungti, arba jei r įjungtas, o p išjungtas, arba jei q įjungtas, o r išjungtas.

Elektros srovės praleidimo sąlygą galima užrašyti kaip formulę.

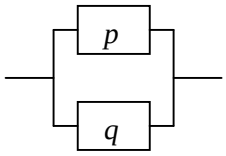
Pavyzdys

Paskutinio pavyzdžio schema praleis srovę tada ir tik tada, kai bus teisingas sudėtinis teiginys $(p \wedge q) \vee (r \wedge \neg p) \vee (q \wedge \neg r)$, kur p yra teiginys "jungiklis p įjungtas", o q ir r yra analogiški teiginiai.

Nuoseklų kontaktinių elementų jungimą atitinka konjunkcija, o lygiagretų – disjunkcija. Iš tikro: kontaktinė schema



praleis srovę tada ir tik tada, kai abu kontaktiniai elementai p ir q bus įjungti, tai yra, kai teiginys "p įjungtas ir q įjungtas" yra teisingas, tai yra, kai teiginys $p \wedge q$ yra teisingas. Analogiškai kontaktinė schema



praleis srovę tada ir tik tada, kai bent vienas kontaktinis elementas p arba q bus įjungtas, tai yra, kai teiginys "p įjungtas arba q įjungtas" yra teisingas, tai yra, kai teiginys $p \vee q$ yra teisingas.

Taigi, kiekvienai kontaktinei schemai galima parašyti ją atitinkančią formulę.

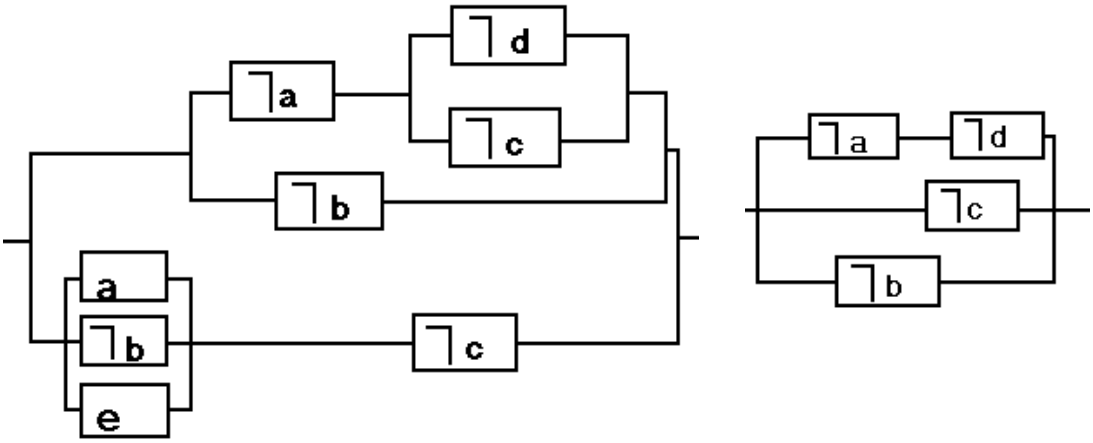
Teisingas ir atvirkščias tvirtinimas: jeigu turime formulę, kurioje yra tik loginės operacijos $\neg$, $\wedge$, $\vee$, ir $\neg$ yra tik prieš loginius kintamuosius, tai galima nubrėžti ją atitinkančią schemą.

Kai mokysimės apie normaliąsias formas, matysime, kad bet kuri formulė gali būti išreikšta tokiu pavidalu, taigi, bet kuriai formulei egzistuoja kontaktinė schema, kuri praleidžia srovę tada ir tik tada, kai ta formulė teisinga.

Kontaktinės schemos vadinamos *ekvivalenčiomis*, jei viena praleidžia srovę tada ir tik tada, kai kita taip pat praleidžia srovę. Žinome, kad schema praleidžia srovę tada ir tik tada, kai ją atitinkanti formulė yra teisinga. Jei kontaktinės schemos ekvivalenčios, tai kiekvienam kontaktinių elementų reikšmių rinkiniui jos abi kartu praleidžia srovę arba ne, todėl jas atitinkančios formulės yra kartu teisingos arba ne, tai yra formulės yra ekvivalenčios. Ir atvirkščiai, jei dvi formulės yra ekvivalenčios, tai jas atitinkančios kontaktinės schemos yra ekvivalenčios.

Pavyzdys

Tarkime, turime dvi schemas:



Jos yra ekvivalenčios, nes jas atitinkančios formulės yra ekvivalenčios. Iš tikro:
((¬a ∧ (¬d ∨ ¬c)) ∨ ¬b) ∨ ((a ∨ ¬b ∨ e) ∧ ¬c) ~
~ (¬a ∧ ¬d) ∨ (¬a ∧ ¬c) ∨ ¬b ∨(a ∧¬c) ∨ (¬b∧ ¬c) ∨ (e ∧ ¬c) ~
~ (¬a ∧ ¬d) ∨ (¬c ∧ (¬a ∨ a ∨ ¬b ∨ e)) ∨ ¬b ~
~ (¬a ∧ ¬d) ∨ (¬c ∧ (t ∨ ¬b ∨ e)) ∨ ¬b ~
~ (¬a ∧ ¬d) ∨ (¬c ∧ t) ∨ ¬b ~
~ (¬a ∧ ¬d) ∨ ¬c ∨ ¬b.

Tokiu būdu gavome metodą kontaktinių schemų ekvivalentiškumui patikrinti: imame jas atitinkančias logines formules ir nustatome, ar jos ekvivalenčios.

Į matematinę logiką suvedama keletas praktinių, įprastų inžinieriams, uždavinių. Pavyzdžiui, duotai schemai rasti jai ekvivalenčią ir paprastesnę. Kas ta paprastesnė schema, patikslinama konkrečiam uždaviniui, įvedus sudėtingumo sąvoką. Schemos sudėtingumu gali būti bendras elementų skaičius, skirtingomis raidėmis pažymėtų elementų skaičius, elementų, pažymėtų tam tikromis raidėmis skaičius ir t.t. Taip pat, žinant elementų kainas (skirtingų elementų jos gali būti nevienodos), apskaičiuojama schemos kaina ir stengiamasi rasti ekvivalenčią schemą, kurios kaina būtų mažiausia (schemos minimizacijos pagal kainą uždavinys).

2.7 Normaliosios formos

2.7.1 Normalioji disjunkcinė forma

Apibrėžimas

Elementaria konjunkcija (EK) *vadiname loginę formulę, į kurią įeina tik loginiai kintamieji ir jų neiginiai, sujungti konjunkcijomis.*

Pavyzdys

Formulės a ∧ ¬b ∧ c, $\neg p \wedge q \wedge r \wedge \neg s$, $\neg p_2$, p_3 yra elementarios konjunkcijos.

Kiekviena elementari konjunkcija yra arba tapačiai klaidinga, arba teisinga tik su viena interpretacija. Iš tikrųjų, jei į elementarią konjunkciją įeina kuris nors kintamasis ir jo neiginys, tai jinai yra tapačiai klaidinga.

Pavyzdys

Elementari konjunkcija $\neg p \wedge q \wedge r \wedge \neg q$ yra tapačiai klaidinga, nes

$$\neg p \wedge q \wedge r \wedge \neg q \sim \neg p \wedge r \wedge (q \wedge \neg q) \sim (\neg p \wedge r) \wedge k \sim k. \ \square$$

Priešingu atveju, jinai teisinga su vienintele interpretacija.

Pavyzdys

Elementari konjunkcija $\neg p \wedge q \wedge r$ yra teisinga, kai $\neg p$, q ir r yra teisingi, tai yra, kai $p=k$, $q=t$ ir $r=t$, o visais kitais atvejais klaidinga, nes bent vienas iš $\neg p$, q ar r bus klaidingas.

Apibrėžimas

Normaliąja disjunkcine forma (NDF) *vadiname formulę, kurios pavidalas yra* $\bigvee_{i=1}^{m} K_i$, *kur* K_i *– elementarios konjunkcijos.* Formulės *Q* normaliąja disjunkcine forma *vadiname jai ekvivalenčią NDF*.

Pavyzdys

Formulė $(a \wedge \neg b) \vee (b \wedge \neg c) \vee (\neg a \wedge b \wedge \neg c) \vee \neg b$ yra NDF.

Teorema

Kiekvienai loginei formulei egzistuoja NDF.

Pirmas įrodymas. Užrašome formulės teisingumo reikšmių lentelę ir iš jos gauname formulės NDF tokiu būdu: imame tik tas lentelės eilutes, kur formulės reikšmė yra *t*, ir iš kiekvienos tokios eilutės suformuojame vieną elementarią konjunkciją, konstantas t keisdami į atitinkamus loginius kintamuosius, o konstantas k – į jų neiginius.

Parodysime pavyzdžiu. Tarkime, formulės *Q(p,q,r)* teisingumo reikšmių lentelė yra parodyta dešinėje. Turime tris eilutes, kuriose formulės reikšmė yra *t*. Kiekvienai eilutei keičiame *t* į atitinkamą kintamąjį, *k* į jo neiginį. Iš pirmos eilutės gauname elementarią konjunkciją $p \wedge q \wedge r$, iš penktos $\neg p \wedge q \wedge r$, iš šeštos $\neg p \wedge q \wedge \neg r$, ir imame jų disjunkciją: $P(p,q,r) = (p \wedge q \wedge r) \vee (\neg p \wedge q \wedge r) \vee (\neg p \wedge q \wedge \neg r)$. Tai ir bus formulės *Q(p,q,r)* NDF. Iš tikro, pirma elementari konjunkcija teisinga tik su interpretacija *p=t*, *q=t*, *r=t*, antra su *p=k*, *q=t*, *r=t*, trečia su *p=k*, *q=t*, *r=k*, todėl jų disjunkcija teisinga tik su šiomis trimis interpretacijomis, o su kitomis – klaidinga. Taigi, formulės *P(p,q,r)* teisingumo reikšmių lentelė sutampa su formulės *Q(p,q,r)* teisingumo reikšmių lentele, todėl jos ekvivalenčios. Ir formulė *P(p,q,r)* yra elementarių konjunkcijų disjunkcija, todėl *P(p,q,r)* yra formulės *Q(p,q,r)* NDF.

p	*q*	*r*	*Q(p,q,r)*
t	*t*	*t*	*t*
t	*t*	*k*	*k*
t	*k*	*t*	*k*
t	*k*	*k*	*k*
k	*t*	*t*	*t*
k	*t*	*k*	*t*
k	*k*	*t*	*k*
k	*k*	*k*	*k*

O ką darome, kai formulės reikšmė niekada nebūna *t*, tai yra, kai formulė yra tapačiai klaidinga ? Tokiu atveju jos NDF yra bet kuri tapačiai klaidinga NDF, pavyzdžiui, $p \wedge \neg p$. $\square$

Pastaba

Įrodymas parodė, kaip, turint formulę, užduotą teisingumo lentele, užrašyti ją atitinkančią loginę formulę. Mes norėjome to išmokti, kai nagrinėjome esminius ir fiktyvius kintamuosius.

Antras įrodymas. Suvedame formulę į NDF, naudodami logikos dėsnius.

1 žingsnis. Pašaliname visas operacijas, išskyrus $\neg$, $\vee$, $\wedge$. Tam naudojamės šiais logikos dėsniais:

$p \oplus q \sim \neg(p \leftrightarrow q)$,
$p \leftrightarrow q \sim (p \to q) \wedge (q \to p)$,
$p \to q \sim \neg p \vee q$,
$p \mid q \sim \neg(p \wedge q)$.

2 žingsnis. Įkeliame neiginius į skliaustus (neiginiai turi likti tik prie loginių kintamųjų). Naudojamės De Morgano dėsniais

$\neg(p \vee q) \sim \neg p \& \neg q$,
$\neg(p \& q) \sim \neg p \vee \neg q$,

ir dėsniu

$\neg\neg p \sim p$.

3 žingsnis. Taikome distributyvumo dėsnį

$p \wedge (q \vee r) \sim (p \wedge q) \vee (p \wedge r)$

tol, kol gausime NDF.

Kiekvieno žingsnio eigoje ir po kiekvieno žingsnio galime prastinti, naudodami šiuos ir kitus dėsnius:

$p \vee p \sim p$,
$p \wedge p \sim p$,
$p \wedge \neg p \sim k$,
$p \vee \neg p \sim t$,
$(p \vee q) \& p \sim p$,
$(p \& q) \vee p \sim p$.

Po šių trijų žingsnių ir gausime duotos formulės NDF. □

Pavyzdys

Norime rasti formulės $\neg(p \to (q \wedge r))$ NDF.

1 žingsnis: pašaliname implikaciją:

$\neg(p \to (q \wedge r)) \sim \neg(\neg p \vee (q \wedge r))$.

2 žingsnis: įkeliame neiginius į skliaustus:

$\neg(\neg p \vee (q \wedge r)) \sim p \wedge \neg(q \wedge r) \sim p \wedge (\neg q \vee \neg r)$.

3 žingsnis: taikome distributyvumo dėsnį:

$p \wedge (\neg q \vee \neg r) \sim (p \wedge \neg q) \vee (p \wedge \neg r)$.

Suprastinti nėra ką. Gavome NDF. □

Pastaba

Formulei gali egzistuoti kelios NDF.

Pavyzdys

Formulė

$(p \wedge \neg q \wedge r) \vee (p \wedge \neg q \wedge \neg r) \vee (p \wedge \neg r)$

taip pat yra formulės iš praeito pavyzdžio NDF. Lengva tuo įsitikinti, patikrinus, kad tai iš tikro yra NDF ir kad šios formulės yra ekvivalenčios.

Išvada

Kiekvienai formulei egzistuoja jai ekvivalenti formulė, į kurią įeina tik neiginys, konjunkcija ir disjunkcija, ir neiginys yra tik prie kintamųjų.

Grįžtant prie kontaktinių schemų, dabar įsitikinome, kad tikrai bet kuriai formulei egzistuoja kontaktinė schema, kuri praleidžia srovę tada ir tik tada, kai ta formulė yra teisinga.

2.7.2 Normalioji konjunkcinė forma

Apibrėžimas

Elementaria disjunkcija (ED) *vadiname loginę formulę, į kurią įeina tik loginiai kintamieji ir jų neiginiai, sujungti disjunkcijomis.*

Pavyzdys

Formulės $p_1 \vee \neg p_2$, $\neg p \vee \neg q \vee r$, q, $\neg p_3$ yra elementarios disjunkcijos.

Kiekviena elementari disjunkcija yra arba tapačiai teisinga, arba klaidinga tik su viena interpretacija. Iš tikrųjų, jei į elementarią disjunkciją įeina kuris nors kintamasis ir jo neiginys, tai jinai yra tapačiai teisinga.

Pavyzdys

Formulė $\neg p \vee q \vee r \vee \neg q$ yra tapačiai teisinga, nes

$$\neg p \vee q \vee r \vee \neg q \sim \neg p \vee r \vee (q \vee \neg q) \sim (\neg p \vee r) \vee t \sim t. \ \square$$

Priešingu atveju, jinai klaidinga su vienintele interpretacija.

Pavyzdys

Formulė $\neg p \vee q \vee r$ yra klaidinga, kai $\neg p$, q ir r yra klaidingi, tai yra, kai $p = t$, $q = k$ ir $r = k$, o visais kitais atvejais bus teisinga, nes bent vienas iš $\neg p$, q ir r bus teisingas.

Apibrėžimas.

Normaliąja konjunkcine forma (NKF) *vadiname formulę, kurios pavidalas yra* $\overset{m}{\underset{i=1}{\&}} D_i$, *kur* D_i *– elementarios disjunkcijos.* Formulės Q normaliąja konjunkcine forma *vadiname jai ekvivalenčią NKF.*

Pavyzdys

Formulė $(p_1 \vee \neg p_3) \wedge (\neg p_1 \vee \neg p_2 \vee p_3) \wedge p_3$ yra NKF.

Teorema

Kiekvienai loginei formulei egzistuoja NKF.

Pirmas įrodymas. Sudarome formulės teisingumo lentelę ir iš jos gauname formulės NKF tokiu būdu: imama tik tas lentelės eilutes, kur formulės reikšmė yra k, ir iš kiekvienos tokios eilutės suformuojame vieną elementarią disjunkciją, konstantą k keisdami į atitinkamą loginį kintamąjį, o konstantą t – į jo neiginį.

p	*q*	*r*	*Q(p,q,r)*
t	*t*	*t*	*k*
t	*t*	*k*	*k*
t	*k*	*t*	*t*
t	*k*	*k*	*t*
k	*t*	*t*	*t*
k	*t*	*k*	*k*
k	*k*	*t*	*t*
k	*k*	*k*	*t*

Parodysime pavyzdžiu. Tarkime, formulės *Q(p,q,r)* teisingumo reikšmių lentelė yra parodyta dešinėje. Turime tris eilutes, kuriose formulės reikšmė yra *k*. Kiekvienai eilutei keičiame *k* į atitinkamą kintamąjį, *t* – į jo neiginį. Iš pirmos eilutės gauname elementarią disjunkciją $\neg p \vee \neg q \vee \neg r$, iš antros $\neg p \vee \neg q \vee r$, iš šeštos $p \vee \neg q \vee r$, ir imame jų konjunkciją: $P(p,q,r) = (\neg p \vee \neg q \vee \neg r) \wedge (\neg p \vee \neg q \vee r) \wedge (p \vee \neg q \vee r)$. Tai ir bus formulės *Q(p,q,r)* NKF. Iš tikro, pirma elementari disjunkcija klaidinga tik su interpretacija *p=t, q=t, r=t*, antra su *p=t, q=t, r=k*, trečia su *p=k, q=t, r=k*, todėl jų konjunkcija klaidinga tik su šiomis trimis interpretacijomis, o su kitomis – teisinga. Taigi, formulės *P(p,q,r)* teisingumo reikšmių lentelė sutampa su formulės *Q(p,q,r)* teisingumo reikšmių lentele, todėl jos ekvivalenčios. Ir formulė *P(p,q,r)* yra elementarių disjunkcijų konjunkcija, todėl *P(p,q,r)* yra formulės *Q(p,q,r)* NKF.

O ką darome, kai formulės reikšmė niekada nebūna *k*, tai yra, kai formulė yra tapačiai teisinga ? Tokiu atveju jos NKF yra bet kuri tapačiai teisinga NKF, pavyzdžiui, $p \vee \neg p$. □

Antras įrodymas. Analogiškas įrodymui apie NDF, tik trečiajame žingsnyje naudojame kitą distributyvumo dėsnį:

$$p \vee (q \,\&\, r) \sim (p \vee q) \,\&\, (p \vee r).$$ □

Pavyzdys

Norime rasti formulės $\neg(p \to q) \vee r$ NKF.

1 žingsnis: pašaliname implikaciją:

$$\neg(p \to q) \vee r \sim \neg(\neg p \vee q) \vee r.$$

2 žingsnis: įkeliame neiginius į skliaustus:

$$\neg(\neg p \vee q) \vee r \sim (p \,\&\, \neg q) \vee r.$$

3 žingsnis: taikome distributyvumo dėsnį:

$$(p \,\&\, \neg q) \vee r \sim (p \vee r) \,\&\, (\neg q \vee r).$$

Suprastinti nėra ką. Gavome NKF. □

Pastaba

Formulei gali egzistuoti kelios NKF.

2.7.3 Tobulos normaliosios formos

Apibrėžimas

Elementarioji konjunkcija vadinama tobula, *jei į ją po vieną kartą įeina kiekvienas kintamasis arba jo neiginys.*

Tobulas elementarias konjunkcijas sutrumpintai žymėsime TEK.

Pavyzdys

Formulės $(p \,\&\, \neg q) \vee (\neg p \,\&\, q \,\&\, \neg r) \vee (\neg p \,\&\, \neg q \,\&\, \neg p \,\&\, r)$ antroji elementari konjunkcija yra tobula, o pirmoji ir trečioji – ne, nes į pirmą neįeina nei kintamasis r, nei jo neiginys, o į trečią kintamojo p neiginys įeina du kartus.

Apibrėžimas

Formulės NDF vadinama tobula, *jei jos visos elementarios konjunkcijos yra tobulos.*

Tobulas NDF sutrumpintai žymėsime TNDF.

Pavyzdys

Formulės

$(p \,\&\, q) \vee (\neg p \,\&\, \neg q)$,

$(\neg p \,\&\, q \,\&\, r) \vee (\neg p \,\&\, q \,\&\, \neg r) \vee (p \,\&\, \neg q \,\&\, r)$

yra TNDF, o

$(p \,\&\, q) \vee (\neg p \,\&\, \neg q \,\&\, r)$,

$(p \,\&\, q \,\&\, \neg p) \vee (\neg p \,\&\, \neg q)$

nėra TNDF.

Teorema

Kiekvienai įvykdomai loginei formulei egzistuoja vienintelė TNDF.

Komentaras

Vienintelė, jei nekreipsim dėmesio į elementarių konjunkcijų sukeitimą vietomis.

Teoremos įrodymas. Jei formulė nėra tapačiai klaidinga, tai konstruodami jos NDF iš teisingumo lentelės, gauname jos TNDF, nes į kiekvieną elementarią konjunkciją įeina kiekvienas kintamasis arba jo neiginys po vieną kartą. Ir šita TNDF yra vienintelė, nes kiekvieną interpretaciją, su kuria formulė teisinga, atitinka viena tobula elementari konjunkcija. Iš tikro, jeigu būtų dvi skirtingos formulės TNDF, tai į jas įeitų skirtingos TEK, taigi, šios TNDF būtų teisingos su skirtingomis interpretacijomis, o tai prieštarauja tam, tad formulė yra ekvivalenti savo TNDF. □

Teorema

Dvi įvykdomos formulės yra ekvivalenčios tada ir tik tada, kai jų TNDF yra lygios.

Įrodymas.

"⇒" Ekvivalenčių formulių teisingumo lentelės sutampa, todėl iš jų sukonstruotos TNDF taip pat sutaps.

"⇐" Tarkime, kad dvi formulės turi tą pačią TNDF. Jos abi yra jai ekvivalenčios, todėl jos yra tarpusavyje ekvivalenčios. □

Jei turime formulės NDF, ją galime suvesti į TNDF, naudodami šį logikos dėsnį:

$$K \sim (K \& p) \vee (K \& \neg p),$$

kur K yra elementari konjunkcija. Ši taisyklė parodo, kaip mes fiktyviai galime prirašyti trūkstamą kintamąjį p.

Pavyzdys

Formulė $p \vee (\neg p \& \neg q) \vee \neg q$ yra NDF, bet ne TNDF. Pirmoje elementarioje konjunkcijoje trūksta q arba ¬q, trečioje – p arba ¬p. Prirašome juos fiktyviai:

$$p \vee (\neg p \& \neg q) \vee \neg q \sim$$
$$\sim (p \& q) \vee (p \& \neg q) \vee (\neg p \& \neg q) \vee (\neg q \& p) \vee (\neg q \& \neg p) \sim$$
$$\sim (p \& q) \vee (p \& \neg q) \vee (\neg p \& \neg q).$$

Gavome TNDF. □

Apibrėžimas

Elementarioji disjunkcija vadinama tobula, *jei į ją po vieną kartą įeina kiekvienas kintamasis arba jo neiginys.*

Tobulas elementarias disjunkcijas sutrumpintai žymėsime TED.

Apibrėžimas

Formulės NKF vadinama tobula, *jei jos visos elementarios disjunkcijos yra tobulos.*

Tobulas NKF sutrumpintai žymėsime TNKF.

Pavyzdys

Formulė $(p \vee q) \& (\neg p \vee q)$ yra TNKF.

Teorema

Kiekvienai loginei formulei, išskyrus tapačiai teisingą, egzistuoja vienintelė TNKF.

Įrodymas. Analogiškas teoremos apie TNDF įrodymui. □

Teorema

Tarkime, turime dvi formules, kurios nėra tapačiai teisingos. Jos yra ekvivalenčios tada ir tik tada, kai jų TNKF yra lygios.

Įrodymas. Analogiškas teoremos apie TNDF įrodymui. □

Jei turime formulės NKF, ją galime suvesti į TNKF, naudodami šį logikos dėsnį:

$$D \sim (D \vee p) \,\&\, (D \vee \neg p),$$

kur D yra elementari disjunkcija.

Apibrėžimas

Formulės ekvivalenčiu pertvarkymu *vadinsime jos suvedimą į jai ekvivalenčią formulę, naudojant logikos dėsnius.*

Pavyzdys

Atliekame formulės ekvivalenčius pertvarkymus:

$$(p \mid q) \to r \sim \neg(p \mid q) \vee r \sim \neg(\neg(p \,\&\, q)) \vee r \sim (p \,\&\, q) \vee r.$$

Teorema

Bet kurias dvi ekvivalenčias formules galima ekvivalenčiais pertvarkymais suvesti vieną į kitą.

Įrodymas. Žinome, kad abi ekvivalenčias formules galima ekvivalenčiais pertvarkymais suvesti į tą pačią TNDF (arba į TNKF). Todėl atvirkštiniais ekvivalenčiais pertvarkymais galime TNDF (arba TNKF) suvesti į jas. Todėl, kad suvestume vieną formulę į kitą, iš pradžių pirmąją suvedame į TNDF (arba TNKF), o paskui atvirkštiniais ekvivalenčiais pertvarkymais TNDF (arba TNKF) suvedame į antrąją formulę. □

2.8 Loginės išvados

Logika yra mokslas apie taisyklingus samprotavimo būdus, t.y. apie būdus, leidžiančius iš teisingų teiginių gauti teisingas išvadas. Tokius samprotavimo būdus mes, apie tai visai negalvodami, vartojame kiekvieną dieną. Šiame paragrafe mes sužinosime, kodėl jie taisyklingi.

Apibrėžimas

Teiginių seka vadinama samprotavimu. *Visi jo teiginiai, išskyrus paskutinį, vadinami* prielaidomis, *o paskutinis vadinamas* išvada.

Paprastai samprotavimas užrašomas taip: $A_1, A_2, \ldots, A_n \therefore B$, kur $A_1, A_2, \ldots, A_n$ yra prielaidos, o B – išvada. Simbolis $\therefore$ skaitomas "todėl". Kartais rašoma stulpeliu:

$$\begin{array}{l} A_1 \\ A_2 \\ \vdots \\ A_n \\ \therefore B \end{array} \qquad \text{arba} \qquad \begin{array}{l} A_1 \\ A_2 \\ \vdots \\ \underline{A_n} \\ \therefore B \end{array}$$

Apibrėžimas

Samprotavimas vadinamas pagrįstu, *jei išvada yra teisinga su kiekviena interpretacija, su kuria visos prielaidos yra teisingos.*

Pastaba

Jei nėra tokių interpretacijų, su kuriomis visos prielaidos būtų teisingos, samprotavimas laikomas pagrįstu.

Pavyzdys

Nustatysime, ar samprotavimas "Knyga yra ant stalo arba ant lentynos. Jos nėra ant lentynos. Todėl jinai yra ant stalo." pagrįstas, ar ne.

Pažymėkime raide p teiginį "Knyga yra ant stalo", o raide q – "Knyga yra ant lentynos". Tada duotas samprotavimas bus užrašytas taip: $p \vee q, \neg q \therefore p$. Pagal apibrėžimą, reikia imti visas interpretacijas, su kuriomis $p \vee q$ ir $\neg q$ yra teisingos, ir patikrinti, ar su jomis p yra teisinga (žr. dešinėje). Matome, kad tokia tėra tik viena interpretacija (p = t, q = k), ir su ja p yra teisinga. Todėl duotas samprotavimas yra pagrįstas. □

p	*q*	$p \vee q$	$\neg q$
t	*t*	*t*	*k*
t	*k*	*t*	*t*
k	*t*	*t*	*k*
k	*k*	*k*	*t*

Teiginys

Samprotavimas $A_1, A_2, \ldots, A_n \therefore B$ yra pagrįstas tada ir tik tada, kai formulė $(A_1 \& A_2 \& \ldots \& A_n) \rightarrow B$ yra tapačiai teisinga.

Įrodymas

"$\Rightarrow$" Tarkime, kad samprotavimas $A_1, A_2, \ldots, A_n \therefore B$ yra pagrįstas. Įrodysime, kad formulė $(A_1 \& A_2 \& \ldots \& A_n) \rightarrow B$ yra tapačiai teisinga.

Imkime bet kurią formulės $(A_1 \& A_2 \& \ldots \& A_n) \rightarrow B$ interpretaciją. Reikia įrodyti, kad šita formulė yra teisinga su šita interpretacija.

Jei su šita interpretacija bent viena iš formulių $A_1, A_2, \ldots, A_n$ yra klaidinga, tai $A_1 \& A_2 \& \ldots \& A_n$ yra taip pat klaidinga, todėl formulė $(A_1 \& A_2 \& \ldots \& A_n) \rightarrow B$ yra teisinga.

Jei su šia interpretacija visos formulės $A_1, A_2, \ldots, A_n$ yra teisingos, tai pagal pagrįsto samprotavimo apibrėžimą formulė B taip pat yra teisinga, todėl formulė $(A_1 \& A_2 \& \ldots \& A_n) \rightarrow B$ yra teisinga.

Todėl su bet kuria interpretacija formulė $(A_1 \& A_2 \& \ldots \& A_n) \rightarrow B$ yra teisinga.

"$\Leftarrow$" Tarkime, kad formulė $(A_1 \& A_2 \& \ldots \& A_n) \rightarrow B$ yra tapačiai teisinga. Įrodysime, kad samprotavimas $A_1, A_2, \ldots, A_n \therefore B$ yra pagrįstas.

Tarkime, kad prielaidos $A_1, A_2, \ldots, A_n$ yra teisingos. Reikia įrodyti, kad išvada B yra teisinga. Įrodysime prieštaros metodu. Tarkime, B nėra teisinga. Tada gauname, kad $(A_1 \& A_2 \& \ldots \& A_n) \rightarrow B$ yra klaidinga, nes $A_1 \& A_2 \& \ldots \& A_n$ yra teisinga, o B yra klaidinga. Taigi, gauname prieštarą tam, kad $(A_1 \& A_2 \& \ldots \& A_n) \rightarrow B$ yra tapačiai teisinga. □

Matome, kad kiekvieną pagrįstą samprotavimą atitinka logikos dėsnis, todėl pagrįsti samprotavimai irgi kartais vadinami logikos dėsniais.

Pavyzdys

Patikrinsime, kad pagrįstas samprotavimas iš paskutinio pavyzdžio tikrai atitinka tapačiai teisingą loginę formulę, t.y., kad formulė $Q(p,q) \sim ((p \vee q) \& \neg q) \rightarrow p$ yra tapačiai teisinga (žr. dešinėje).

p	*q*	$p \vee q$	$(p \vee q) \& \neg q$	*Q(p,q)*
t	*t*	*t*	*k*	*t*
t	*k*	*t*	*t*	*t*
k	*t*	*t*	*k*	*t*
k	*k*	*k*	*k*	*t*

Savybės

Tarkime, kad samprotavimas $A_1, A_2, \ldots, A_n \therefore B$ yra pagrįstas. Tada:

1) *Jei visos prielaidos $A_1, A_2, \ldots, A_n$ yra tapačiai teisingos, tai ir išvada B yra tapačiai teisinga.*
2) *Jei $A_1 \& A_2 \& \ldots \& A_n$ yra tapačiai klaidinga, tai B gali būti bet kokia formulė.*
3) *Jei B yra tapačiai teisinga, tai $A_1, A_2, \ldots, A_n$ gali būti bet kokios formulės.*

Įrodymas
Tiesiogiai išplaukia iš pagrįsto samprotavimo apibrėžimo. □

Iš antros savybės matome, kad jei A_1 & A_2 & …& A_n yra tapačiai klaidinga, tai bet kokia formulė B yra formulių A_1, A_2, …, A_n išvada.

Pavyzdys
"Dabar lyja. Dabar nelyja. Todėl drambliai moka skraidyti." Tai pagrįstas samprotavimas, bet iš jo samprotaujant jokios naudos, nes jo prielaidos prieštarauja viena kitai. Išskirsime tokius samprotavimus.

Apibrėžimas
Samprotavimas vadinamas netinkamu, *jei jo prielaidos prieštaringos, t.y., jei jo prielaidų konjunkcija yra tapačiai klaidinga, ir* tinkamu *priešingu atveju*.

Taigi, ieškome pagrįstų tinkamų samprotavimų. Turėdami tokį samprotavimą, jo pagalba galime gauti teisingas išvadas iš teisingų prielaidų.

Apibrėžimas
Išvedimo taisyklė *yra pagrįstas samprotavimas.*

Turime, kad kiekvieną logikos dėsnį, kurio pavidalas yra (A_1 & A_2 & …& A_n) $\rightarrow$ B, atitinka išvedimo taisyklė A_1, A_2, …, $A_n \therefore$ B.

Pavyzdys
$p \rightarrow (p \vee q)$ yra logikos dėsnis, todėl galioja išvedimo taisyklė $p \therefore p \vee q$ (t.y. formulė $p \vee q$ yra išvada iš prielaidos p).
Štai dažniausiai naudojamų išvedimo taisyklių sąrašas:

1) $p \therefore p \vee q$ (prijungimas)
2) $p \& q \therefore p$ (atskyrimas)
3) $p, q \therefore p \& q$ (konjunktyvus sujungimas)
4) $p \vee q, \neg q \therefore p$ (disjunktyvus silogizmas)
5) $p \rightarrow q, p \therefore q$ (modus ponens, arba teisingos išvados taisyklė)
6) $p \rightarrow q, \neg q \therefore \neg p$ (modus tollens, arba neteisingos išvados taisyklė)
7) $p \rightarrow q \therefore \neg q \rightarrow \neg p$ (kontrapozicijos taisyklė)

Kiekvieną iš šių išvedimo taisyklių galima įrodyti, naudojant teisingumo reikšmių lenteles. Pavyzdžiui, pirmame šio paragrafo pavyzdyje įrodėme disjunktyvaus silogizmo taisyklę.
Štai pavyzdžiai, iliustruojantys duotas išvedimo taisykles.

1) Aš mėgstu ledus. Todėl aš mėgstu ledus arba cepelinus.
2) Aš mėgstu ledus ir cepelinus. Todėl aš mėgstu ledus.
3) Aš mėgstu ledus. Aš mėgstu cepelinus. Todėl aš mėgstu ledus ir cepelinus.
4) (pirmas šio paragrafo pavyzdys)
5) Jei šiandien sekmadienis, tai paskaitų nėra. Šiandien sekmadienis. Todėl paskaitų nėra.
6) Jei šiandien sekmadienis, tai paskaitų nėra. Paskaitos yra. Todėl šiandien ne sekmadienis.
7) Jei šiandien sekmadienis, tai paskaitų nėra. Todėl jei paskaitos yra, tai šiandien ne sekmadienis.

Aišku, išvedimo taisyklių yra žymiai daugiau. Jei mes gauname kokį naują pagrįstą samprotavimą, tai jį irgi galime naudoti kaip išvedimo taisyklę.

Yra keli būdai nustatyti, ar samprotavimas $A_1, A_2, \ldots, A_r \therefore B$ yra pagrįstas.

1 būdas

Naudojamės tuo, kad šis samprotavimas yra pagrįstas tada ir tik tada, kai loginė formulė $(A_1 \,\&\, A_2 \,\&\, \ldots \,\&\, A_r) \to B$ yra tapačiai teisinga. Taigi, naudodami teisingumo lentelę arba logikos dėsnius, patikriname, ar ši formulė yra tapačiai teisinga. Beje, šiuo būdu galime patikrinti ir ar samprotavimas yra tinkamas. Bet jei kintamųjų daug, tai teisingumo lentelė bus labai didelė, pavyzdžiui, jei turime 5 kintamuosius, tai lentelėje bus 32 eilutės, jei 6 – 64, ir t.t. Todėl naudojami ir kiti būdai.

2 būdas (loginė lygtis)

Bandome išspręsti loginę lygtį $(A_1 \,\&\, A_2 \,\&\, \ldots \,\&\, A_r) \to B = k$. Jei randame bent vieną sprendinį, reiškia, kairėje lygybės pusėje esanti formulė nėra tapačiai teisinga, todėl ją atitinkantis samprotavimas nėra pagrįstas. Iš kitos pusės, jei parodome, kad lygtis sprendinių neturi, tai reiškia, kad kairėje lygybės pusėje esanti formulė yra tapačiai teisinga, todėl ją atitinkantis samprotavimas yra pagrįstas. Ši lygtis lengvai susiveda į lygčių sistemą

$$\begin{cases} A_1 = t, \\ A_2 = t, \\ \quad \vdots \\ A_r = t, \\ B = k. \end{cases}$$

Šią lygčių sistemą bandome dar smulkinti, o paskui perrinkinėjame variantus arba bandome atspėti sprendinį.

Pavyzdys

Tarkime, turime samprotavimą $p \to q,\ q \to r \therefore r$. Jį atitinka loginė lygtis

$$((p \to q) \,\&\, (q \to r)) \to r = k.$$

Spręsdami šią lygtį, gauname lygčių sistemą

$$\begin{cases} p \to q = t \\ q \to r = t \\ r = k \end{cases}$$

Iš trečios lygties $r = k$ įstatome į antrą, gauname lygtį $q \to k = t$, kurią išsprendę gauname $q = k$. Įstatome tai į pirmą lygtį, gauname $p \to k = t$, todėl $p = k$. Gavome sprendinį $p = k$, $q = k$, $r = k$. Sprendinys iš tikro tenkina šią lygtį, tai yra, egzistuoja interpretacija, su kuria lygties kairė pusė yra klaidinga, taigi, samprotavimas nėra pagrįstas.

3 būdas (formaliosios dedukcijos metodas)

Išvedame išvadą nuosekliai, žingsnis po žingsnio, iš duotų prielaidų, naudodami išvedimo taisykles ir logikos dėsnius. Kiekvieno žingsnio rezultatas yra nauja prielaida tolesniems išvedimo žingsniams.

Pavyzdys

Įrodykime, kad samprotavimas

$$p \vee q,\ q \to r,\ (p \wedge s) \to v,\ \neg r,\ \neg q \to (u \wedge s) \therefore v$$

yra pagrįstas.

Patogumo dėlei prielaidas numeruokime ir rašykime stulpeliu. Gavę išvadą, šalia parašome, kokia išvedimo taisykle naudojantis ir iš kokių prielaidų ji gauta. Gautas išvadas naudojame kaip prielaidas tolesniuose išvedimo žingsniuose.

1.	$p \vee q$	
2.	$q \to r$	
3.	$(p \wedge s) \to v$	
4.	$\neg r$	
5.	$\neg q \to (u \wedge s)$	
6.	$\neg q$	(modus tollens iš 2, 4)
7.	p	(disjunktyvus silogizmas 1,6)
8.	$u \wedge s$	(modus ponens 5, 6)
9.	s	(atskyrimas 8)
10.	$p \wedge s$	(konjunktyvus sujungimas 7, 9)
11.	v	(modus ponens 3, 10)

Ats.: taip, samprotavimas yra pagrįstas, nes iš duotų prielaidų gavome išvadą naudodamiesi išvedimo taisyklėmis.

4 būdas (sąlyginio įrodymo metodas)

Naudojamas tada, kai išvada yra implikacija, tai yra $A \to B$ pavidalo formulė, arba disjunkcija, kuri lengvai perdirbama į implikaciją pagal implikacijos pašalinimo dėsnį. Taip pat gali būti naudojamas kaip dalis ilgesnio išvedimo pagal formaliąją dedukciją.

Taisyklė tokia: jei, padarę laikiną prielaidą p, galime įrodyti, kad galioja teiginys q, tai galime padaryti išvadą, kad yra teisinga formulė $p \to q$ (čia ir toliau "teiginys galioja" reiškia "teiginys yra teisingas").

Pavyzdys

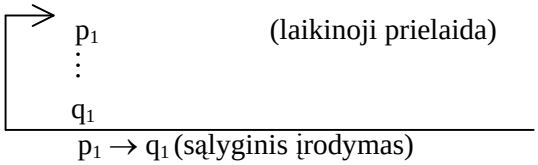
Įrodykime, kad samprotavimas $A \vee B \to C \therefore A \to C$ yra pagrįstas.

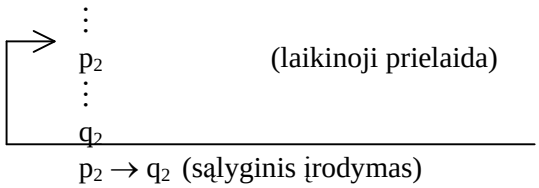
1.	$A \vee B \to C$	
2.	A	(laikina prielaida)
3.	$A \vee B$	(prijungimas 2)
4.	C	(modus ponens 1, 3)
5.	$A \to C$	(sąlyginis įrodymas 2 – 4)

Čia tokia linija mes pažymėjome laikinos prielaidos galiojimo sritį.

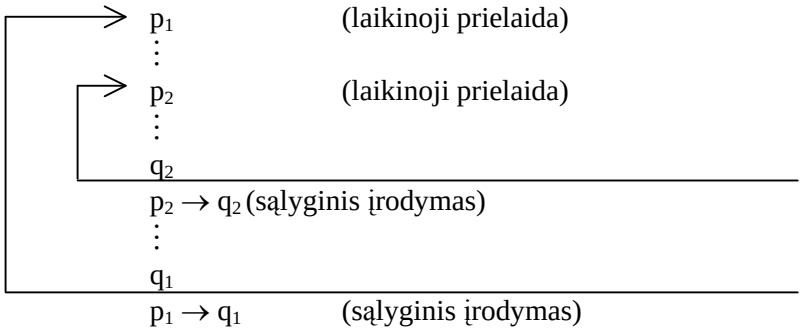
Pastabos

1. Neužmirškite "atsisveikinti" su laikina prielaida, t.y. uždaryti jos galiojimo srities.
2. Nei viena prielaida iš laikinosios prielaidos galiojimo srities negali būti naudojama už šios srities ribų.
3. Jei laikinųjų prielaidų padaroma daugiau nei viena, jų galiojimo sritys arba turi nesikirsti, arba turi būti viena kitoje, t.y. jų galiojimo sritis žyminčios linijos neturi kirstis. Pavyzdžiui, jei laikinų prielaidų yra dvi, jos viena kitos atžvilgiu turi būti išsidėsčiusios vienu iš šių dviejų būdų:





arba



5 būdas (prieštaros metodas, arba netiesioginis įrodymas)

Jam tinka viskas, kas pasakyta apie sąlyginio įrodymo metodą, įskaitant ir pastabas apie laikinos prielaidos galiojimo sritį, o taisyklė yra tokia: jei, padarę laikiną prielaidą p, galime išvesti prieštaravimą q & ¬q, tai galime padaryti išvadą, kad ¬p.

Pavyzdys

Įrodykime, kad samprotavimas $A \to (B \& C), \neg B \therefore \neg A$ yra pagrįstas.

1. $A \to (B \& C)$
2. $\neg B$
3. A (laikinoji prielaida)
4. B & C (modus ponens 1, 3)
5. B (atskyrimas 4)
6. ¬B & B (konjunktyvus sujungimas 2, 5)
7. ¬A (prieštaros metodas 3 – 6)

2.9 Predikatai

Panagrinėkime kelis pavyzdžius:

1. n yra pirminis skaičius.
2. Du iksai plius vienetas yra daugiau už nulį (trumpiau: 2x+1>0).
3. a+b=b-a.

Iš pirmo žvilgsnio atrodytų, kad šie sakiniai kažką teigia, kad jie yra teiginiai. Tačiau ką jie teigia – tiesą ar netiesą – nustatyti negalime, nes šie sakiniai priklauso nuo kintamųjų. Paėmę vieną kintamojo reikšmę, galime gauti teisingą teiginį, paėmę kitą reikšmę – neteisingą. Pavyzdžiui, pirmas sakinys yra teisingas, kai n = 2, ir neteisingas, kai n = 4.

Apibrėžimas

Predikatu *vadinamas sakinys su baigtiniu skaičiumi kintamųjų, kuris tampa teiginiu, vietoj kintamųjų irašius konkrečias jų reikšmes. Kintamojo* kitimo sritimi *vadinama aibė reikšmių, kurias galima įstatyti vietoj kintamojo.*

Pavyzdžiui, į trečią sakinį vietoj kintamųjų a ir b įstačius reikšmes atitinkamai 3 ir 5, gausime teiginį "3+5=5-3", kuris yra klaidingas teiginys. Trečio sakinio kintamųjų a ir b kitimo sritis gali būti, pavyzdžiui, sveikųjų skaičių aibė Z, racionaliųjų skaičių aibė Q, realiųjų skaičių aibė R ir kt.

Predikatus žymėsime didžiosiomis raidėmis, tarp skliaustų nurodydami į juos įeinančius kintamuosius. Pavyzdžiui, 1—3 sakinius galime pažymėti atitinkamai T(n), P(x) ir R(a,b).

Matome, kad predikatą galime laikyti funkcija, priklausančia nuo kelių kintamųjų, kuri, įstačius reikšmes vietoj tų kintamųjų, įgyja reikšmę t arba k, priklausomai nuo to, ar gauname teisingą teiginį, ar klaidingą. Pavyzdžiui, T(n) yra funkcija, kuri įgyja reikšmę t, kai n yra pirminis, ir k priešingu atveju.

Taigi, į predikatą vietoj kintamųjų įrašę konkrečias reikšmes, gauname teiginį. Yra ir kitas būdas iš predikato gauti teiginį: prirašant "egzistuoja" arba "kiekvienam". Pavyzdžiui, pirmą predikatą galime paversti teiginiu tokiu būdu:

Egzistuoja $n \in N$ toks, kad n yra pirminis skaičius,

kur $N = \{1, 2, 3, \ldots\}$ yra natūraliųjų skaičių aibė. Iš tikro, šis sakinys yra teiginys, nes galima nustatyti jo teisingumo reikšmę (tai teisingas teiginys). Taip pat ir sakinys

Kiekvienas $n \in N$ yra pirminis skaičius

yra teiginys (klaidingas). Sutrumpintai šiuos teiginius galima užrašyti atitinkamai

$\exists\, n \in N\ T(n)$, ir

$\forall\, n \in N\ T(n)$.

Ženklai $\exists$ ir $\forall$ vadinami atitinkamai *egzistavimo ir visuotinumo (bendrumo) kvantoriais*. Jie skaitomi atitinkamai "egzistuoja" ("yra") ir "kiekvienam" ("su visais"). Kintamojo kitimo sritis gali būti nenurodoma, jei ji yra numanoma arba nesvarbi, pavyzdžiui, $\forall n\ T(n)$.

Tarkime, kad Q(x) yra predikatas, o $x \in D$ (D – kintamojo x kitimo sritis). Tada:

1) Teiginys "$\forall x \in D\ Q(x)$" yra teisingas tada ir tik tada, kai teiginys Q(a) yra teisingas kiekvienam $a \in D$.
2) Teiginys "$\exists\ x \in D\ Q(x)$" yra teisingas tada ir tik tada, kai teiginys Q(a) yra teisingas bent vienam $a \in D$.

Kaip ir visus teiginius, teiginius su kvantoriais galima paneigti. Lengva įsitikinti, kad:

$\neg(\forall x \in D\ Q(x)) \sim \exists x \in D\ \neg Q(x)$, ir

$\neg(\exists x \in D\ Q(x)) \sim \forall x \in D\ \neg Q(x)$.

Predikatus, kaip ir teiginius, galima jungti loginėmis operacijomis. Pavyzdžiui, jei P(x) ir Q(x) yra predikatai, $x \in D$, tai galime sudaryti predikatus $\neg P(x)$, $P(x)\&Q(x)$, $P(x) \rightarrow Q(x)$ ir t.t.

Pavyzdys

Jei P(x) yra predikatas "x yra pirminis", Q(x) – "x yra lyginis", $x \in N$, tai predikatas $P(x)\&Q(x)$ yra "x yra pirminis ir lyginis". Pažymėkime jį R(x). Tada R(2) yra teisingas teiginys, o R(3) – klaidingas. □

Turėdami kokį nors predikatą P(x), kintamojo x kitimo sritį galime padalinti į dvi dalis: į tuos x, su kuriais P(x) yra teisingas teiginys, ir į tuos x, su kuriais tai yra klaidingas teiginys. Todėl kiekvienas predikatas P(x) apibrėžia aibę, vadinamą *teisingumo reikšmių*

aibe (arba tiesiog *teisingumo aibe*), žymimą $\{x \mid P(x)\}$ arba $\{x : P(x)\}$, kuri yra aibė tų x reikšmių, su kuriomis P(x) tampa teisingu teiginiu. Vadinasi,

- jei P(a) yra teisingas, tai $a \in \{x \mid P(x)\}$,
- jei P(a) yra klaidingas, tai $a \notin \{x \mid P(x)\}$.

Pavyzdžiui, jei P(x) yra antras predikatas iš paragrafo pradžios, tai jo teisingumo reikšmių aibė yra

$$\{x \mid P(x)\} = \{x \mid 2x + 1 > 0\} = (-1/2; \infty).$$

Šį intervalą gauname išsprendę nelygybę $2x + 1 > 0$, t.y. radę tas x reikšmes, su kuriomis predikatas $2x + 1 > 0$ tampa teisingu teiginiu.

Kadangi bet kuri lygtis, nelygybė ar jų sistema yra sakinys su kintamaisiais, tai jinai yra predikatas. Ją išspręsti – reiškia rasti to predikato teisingumo reikšmių aibę.

Beje, su žymėjimu $\{x \mid P(x)\}$ jau susidūrėme, mokydamiesi aibių teorijos. Tada sakėme, kad tai yra aibė elementų x, tenkinančių savybę P. Dabar matome, kad savybės matematiškai yra aprašomos predikatais.

Kaip patikrinti, ar samprotavimas pagrįstas, ar ne, jei į jį įeina teiginiai su kvantoriais ?

Pavyzdys

Turime tokį samprotavimą:

Bet kas, turintis laiko ir kantrybės, galėtų susiremontuoti savo mašiną. Deja, daugybė žmonių skundžiasi, kad negali patys susiremontuoti mašinos. Tai reiškia –daugybei žmonių tiesiog trūksta kantrybės.

Užrašykime jį matematiškai. Galime pažymėti tokius predikatus, kur kintamojo x kitimo aibė – visų žmonių aibė:

L(x) – “x turi laiko”,

K(x) – “x turi kantrybės”,

M(x) – “x gali susiremontuoti savo mašiną”.

Tada samprotavimas gali būti užrašytas taip:

$$\forall x \ (L(x) \wedge K(x)) \rightarrow M(x), \exists x \ \neg M(x), \therefore \exists x \ \neg K(x). \ \square$$

Samprotavimų, į kuriuos įeina kvantoriai, pagrįstumui nustatyti naudojame tuos pačius būdus kaip ir samprotavimams be kvantorių, tik dar įvedame papildomas taisykles kvantoriams panaikinti ir įvesti.

1 būdas (nepagrįstumui įrodyti)

Suvedame į loginę lygtį ir sprendžiame, kaip ir samprotavimams be kvantorių. Vienintelis keblumas – iš teiginių reikia pašalinti kvantorius. Tai darome tokiu būdu. Tegu $D = \{a_1, a_2, \ldots, a_n\}$ yra kintamojo x kitimo sritis. Tada

$$\forall x \in D \ P(x) \ \sim \ P(a_1) \wedge P(a_2) \wedge \ldots \wedge P(a_n), \text{ ir}$$

$$\exists x \in D \ P(x) \ \sim \ P(a_1) \vee P(a_2) \vee \ldots \vee P(a_n),$$

t.y. visuotinumo kvantorių galime išreikšti konjunkcija, o egzistavimo kvantorių – disjunkcija.

Jei aibė D yra didelė, tai ši konjunkcija ar disjunkcija gausis sudėtinga. Laimei, užtenka parodyti, kad samprotavimas nepagrįstas kuriame nors kitimo srities poaibyje, kad jis būtų nepagrįstas visoje kitimo aibėje. Kai kuriems samprotavimams užtenka imti poaibį iš vieno elemento, pavyzdžiui, kai į samprotavimą įeina teiginiai tik su vienos rūšies kvantoriumi, pavyzdžiui, tik su visuotinumo kvantoriumi. Kai įeina abu kvantoriai, dažniausiai reikia imti poaibį, sudarytą iš bent dviejų elementų, nes aibėje iš vieno elemento visuotinumo ir egzistavimo kvantoriai sutampa (iš tikro, jei $D=\{a\}$, tai $\forall x \in D \ P(x) \ \sim \ \exists x \in D \ P(x)$).

Tais atvejais, kai samprotavimo nepagrįstumas neįrodomas aibei iš dviejų elementų, jis galbūt gali būti įrodytas didesnei aibei. Galima įrodyti, kad užtenka samprotavimo pagrįstumą patikrinti visose aibėse, kurių elementų skaičius neviršija 2^n, kur n – į samprotavimą įeinančių predikatų skaičius. Jei jose neparodysime samprotavimo nepagrįstumo, tai samprotavimas pagrįstas.

Pavyzdys

Parodysime, kad pereito pavyzdžio samprotavimas nepagrįstas. Parodykime tai kintamojo x kitimo srities – visų žmonių aibės – poaibiui D iš dviejų elementų, D = {a, b}. Tuo pačiu samprotavimas bus nepagrįstas ir aibei iš visų žmonių.

Turime:

$\forall x\ (L(x) \wedge K(x)) \to M(x) \sim ((L(a) \wedge K(a)) \to M(a)) \wedge ((L(b) \wedge K(b)) \to M(b))$,

$\exists x\ \neg M(x) \sim \neg M(a) \vee \neg M(b)$,

$\exists x\ \neg K(x) \sim \neg K(a) \vee \neg K(b)$.

Pažymėkime:

$L(a) = p_1$, $L(b) = p_2$,

$K(a) = q_1$, $K(b) = q_2$,

$M(a) = r_1$, $M(b) = r_2$.

Turime parodyti, kad samprotavimas

$(p_1 \wedge q_1 \to r_1) \wedge (p_2 \wedge q_2 \to r_2), \neg r_1 \vee \neg r_2, \therefore \neg q_1 \vee \neg q_2$

yra nepagrįstas. Tam bandysime rasti loginės lygties

$(p_1 \wedge q_1 \to r_1) \wedge (p_2 \wedge q_2 \to r_2) \wedge (\neg r_1 \vee \neg r_2) \to (\neg q_1 \vee \neg q_2) = k$

bent vieną sprendinį. Gauname sistemą

$p_1 \wedge q_1 \to r_1 = t$,

$p_2 \wedge q_2 \to r_2 = t$,

$\neg r_1 \vee \neg r_2 = t$,

$\neg q_1 \vee \neg q_2 = k$.

Iš ketvirtos lygties iškart gauname, kad $\neg q_1 = \neg q_2 = k$, tai yra $q_1 = q_2 = t$. Kitų kintamųjų reikšmes bandome kažkaip parinkti, pavyzdžiui, iš trečios lygties matome, kad arba abu $\neg r_1$ ir $\neg r_2$ įgyja reikšmes t, arba vienas iš jų. Pradžiai tarkime, kad $\neg r_1 = \neg r_2 = t$. Jei sprendinių nerasime, nagrinėsime likusius atvejus. Taigi, $r_1 = r_2 = k$. Įstatome į pirmas dvi lygtis ir gauname

$p_1 \wedge t \to k = t$,

$p_2 \wedge t \to k = t$,

iš kur išvedame $p_1 = p_2 = k$. Taigi, gavome lygties sprendinį

$p_1 = k$, $p_2 = k$, $q_1 = t$, $q_2 = t$, $r_1 = k$, $r_2 = k$,

todėl samprotavimas nepagrįstas aibei iš dviejų elementų (žmonių), o tuo pačiu nepagrįstas ir visų žmonių aibei. □

2 būdas (pagrįstumui įrodyti)

Naudojame tuos pačius metodus (formalios dedukcijos, sąlyginio įrodymo, prieštaros), kaip ir samprotavimams be kvantorių, tik dar pridedame kvantorių paneigimo taisykles ir šias keturias išvedimo taisykles, leidžiančias pašalinti ir įvesti kvantorius.

1. $\forall x\ P(x) \therefore P(a)$, kur a žymi bet kurį laisvai pasirinktą objektą (universali instanciacija, sutrumpintai UI, – visuotinumo kvantoriaus pašalinimas).
2. $P(a) \therefore \forall x\ P(x)$, su sąlyga, kad a – simbolis, įvestas taikant UI (universali generalizacija, UG, – visuotinumo kvantoriaus įvedimas).

3. $\exists x \;\; P(x) \therefore P(a)$, su sąlyga, kad a – dar nepanaudotas simbolis (egzistencinė instancija, EI, – egzistavimo kvantoriaus pašalinimas).
4. $P(a) \therefore \exists x \;\; P(x)$, kur a – bet kurio objekto simbolis (egzistencinė generalizacija, EG, – egzistavimo kvantoriaus įvedimas).

Pavyzdys

Turime tokį samprotavimą:

Yra ne vienas muzikantas, kuris savęs nelaiko krepšinio sirgaliu. Lietuviai, kaip žinia, – visi yra didesni ar mažesni krepšinio gerbėjai. Taigi, dalis muzikantų, išeitų, nėra lietuviai.

Užrašykime jį matematiškai. Įveskime tokius predikatus:

M(x) “x yra muzikantas”,
K(x) “x yra krepšinio sirgalius (gerbėjas)”,
L(x) “x yra lietuvis”.

Tada šis samprotavimas užsirašys taip:

$\exists x \;\; M(x) \wedge \neg K(x), \forall x \;\; L(x) \rightarrow K(x) \therefore \exists x \;\; M(x) \wedge \neg L(x)$.

Įrodykime, kad jis pagrįstas.

1. $\exists x \;\; M(x) \wedge \neg K(x)$
2. $\forall x \;\; L(x) \rightarrow K(x)$
3. $M(a) \wedge \neg K(a)$ (EI iš 1 eilutės)
4. $L(a) \rightarrow K(a)$ (UI 2)
5. $\neg K(a)$ (atskyrimas 3)
6. $\neg L(a)$ (modus tollens 4,5)
7. $M(a)$ (atskyrimas 3)
8. $M(a) \wedge \neg L(a)$ (konjunktyvus sujungimas 6,7)
9. $\exists x \;\; M(x) \wedge \neg L(x)$ (EG 8) □

2.10 Teoremų įrodymo metodai

Šiame skyrelyje panagrinėsime, kokios išvedimo taisyklės dažniausiai yra naudojamos įrodinėjant matematines teoremas. Pagrindiniai teoremų įrodymo būdai yra *tiesioginis įrodymas, netiesioginis įrodymas, įrodymas suvedant į prieštarą* ir *įrodymas, naudojant matematinę indukciją*.

2.10.1 Tiesioginis įrodymas

Norėdami tiesiogiai įrodyti, kad iš prielaidos A išplaukia išvada B, naudodami informaciją, slypinčią prielaidoje A, mes žingsnis po žingsnio konstruojame samprotavimų grandinę $A \to B_1 \to B_2 \to \cdots \to B$, kurios gale ir yra įrodinėjama išvada B.

Pavyzdys 2.10.1. Įrodysime teoremą "*Jei n yra nelyginis sveikasis skaičius, tai skaičius n^2 taip pat yra nelyginis*".

Įrodymas. Tarkime, kad n yra nelyginis. Tada $n = 2k + 1$, kur k yra koks nors sveikasis skaičius. Pakėlę kvadratu, gauname

$$n^2 = 4k^2 + 4k + 1 = 2(2k^2 + 2k) + 1,$$

taigi, n^2 taip pat yra nelyginis. □

2.10.2 Netiesioginis įrodymas

Netiesioginio įrodymo esmę sudaro tai, kad pirmiausia mes darome prielaidą, kad įrodinėjamas teiginys yra neteisingas, ir iš šios bei kitų prielaidų išvedame teiginį, prieštaraujantį kuriai nors prielaidai. Tai įrodo, kad mūsų prielaida apie tą teiginį buvo neteisinga, t.y. įrodinėjamas teiginys teisingas. Šis būdas remiasi kontrapozicijos taisykle:

$$\frac{\neg B \to \neg A}{A \to B}.$$

Pavyzdys 2.10.2. Netiesioginio įrodymo būdu įrodysime teoremą "*Jei $3n + 2$ yra nelyginis, tai ir n yra nelyginis*".

Įrodymas. Tarkime, kad n yra lyginis. Tada $n = 2k$, kur k yra koks nors sveikasis skaičius. Gauname, kad $3n + 2 = 6k + 2 = 2(3k + 1)$ taip pat yra lyginis skaičius, o tai prieštarauna teoremos prielaidai. Taigi, n negali būti lyginis. □

2.10.3 Įrodymas, suvedant į prieštarą

Šis įrodymo būdas yra panašus į netiesioginį įrodymą. Kaip ir netiesioginiame įrodyme, mes darome prielaidą, kad įrodinėjamas teiginys yra neteisingas, ir iš šios bei kitų prielaidų gauname

prieštarą, t.y., išvedame kokį nors teiginį C ir jam priešingą teiginį $\neg C$. Taigi, šis įrodymo būdas remiasi išvedimo taisykle

$$\frac{\neg B \to C \,\&\, \neg C}{B}.$$

Pavyzdys 2.10.3. Įrodysime klasikinę matematinės analizės teoremą, kad $\sqrt{2}$ yra iracionalus skaičius.

Įrodymas. Tarkime, kad $\sqrt{2}$ yra racionalus skaičius. Tada atsiras tokie sveikieji skaičiai a ir b ($b \neq 0$), kad $\sqrt{2} = \frac{a}{b}$ ir a su b yra tarpusavyje pirminiai, t.y., $(a, b) = 1$ (priešingu atveju mes tol prastintume trupmeną $\frac{a}{b}$, kol rastume tokius a ir b). Pakėlę abi lygybės puses kvadratu, gauname:

$$\sqrt{2} = \frac{a}{b} \Rightarrow 2 = \frac{a^2}{b^2} \Rightarrow 2b^2 = a^2 \Rightarrow a^2 \text{ yra lyginis} \Rightarrow a \text{ yra lyginis}.$$

Taigi, atsiras sveikasis skaičius c, toks, kad $a = 2c$. Bet tada $2b^2 = 4c^2 \Rightarrow b^2 = 2c^2$, taigi ir b yra lyginis. Gauname, kad a ir b dalijasi iš 2, o tai prieštarauna anksčiau gautam teiginiui, kad $(a, b) = 1$. Gauta prieštara įrodo, kad $\sqrt{2}$ negali būti racionalus skaičius. □

2.10.4 Įrodymas, naudojant matematinę indukciją

Matematinė indukcija remiasi išvedimo taisykle

$$\frac{F(1) \,\&\, \forall n(F(n) \to F(n+1))}{\forall n F(n)}.$$

Pavyzdys 2.10.4. Įrodysime, kad *kiekvienam natūriniam skaičiui* n *pirmųjų* n *nelyginių teigiamų skaičių suma lygi* n^2, *t.y.*

$$1 + 3 + 5 + \cdots + (2n-1) = n^2 \quad (F(n)).$$

Įrodymas.

(1) $F(1)$: $1 = 1^2$ — teisinga;

(2) Tarkime, $F(n)$ teisingas, t.y. $1+3+5+\cdots+(2n-1) = n^2$. Tada pirmųjų $n+1$ nelyginių teigiamų skaičių suma bus $[1 + 3 + 5 + \cdots + (2n-1)] + (2n+1) = n^2 + (2n+1) = (n+1)^2$, t.y. $F(n+1)$ — teisingas. Įrodėme $F(n) \to F(n+1)$. Pagal apibendrinimo taisyklę gauname $\forall n(F(n) \to F(n+1))$.

(3) Iš (1) ir (2) pagal matematinę indukciją gauname, kad teiginys $F(n)$ teisingas kiekvienam natūriniam skaičiui n. □

Uždaviniai

1. Prieštaros būdu įrodykite teoremą *"Kiekvienam teigiamam sveikam skaičiui $n > 1$ iš to, kad jo daliklių suma lygi $n + 1$ (A), išplaukia, kad tas skaičius yra pirminis (B)"*.

2. Panaudoję matematinę indukciją įrodykite, kad *kiekvienam natūriniam skaičiui n teisinga lygybė*
$$1 + 2 + 2^2 + \cdots + 2^n = 2^{n+1} - 1.$$

2.11 Formalios teorijos

Formalia (aksiomine) teorija vadiname $\mathcal{T} = \langle A, F, Ax, R\rangle$, kur:

(1) A — teorijos $\mathcal{T}$ abėcėlė (skaiti arba baigtinė ženklų aibė). Baigtines teorijos $\mathcal{T}$ ženklų sekas vadiname teorijos $\mathcal{T}$ *reiškiniais*.

(2) F — teorijos $\mathcal{T}$ formulių aibė. F yra teorijos $\mathcal{T}$ reiškinių aibės poaibis. Paprastai egzistuoja efektyvi procedūra, leidžianti nustatyti, ar konkretus reiškinys yra formulė, ar ne.

(3) Ax — teorijos $\mathcal{T}$ aksiomų aibė, $Ax \subset F$ (t.y., kai kurios formulės laikomos aksiomomis).

(4) $R = \{R_1, \ldots, R_n\}$ — baigtinė teorijos $\mathcal{T}$ išvedimo taisyklių aibė. Kiekviena taisyklė R_i yra sąryšis formulių aibėje, siejantis $k \geq 2$ formulių $F_1, \ldots, F_k$. Formulę F_k vadiname tiesiogine išvada iš formulių $F_1, \ldots, F_{k-1}$ ir tokią taisyklę žymime
$$\frac{F_1, \ldots, F_{k-1}}{F_k}.$$

Išvedimo teorijoje $\mathcal{T}$ ir *teoremos* teorijoje $\mathcal{T}$ sąvokos jau buvo apibrėžtos ankstesniame skyrelyje. Dabar pateiksime labai paprastą formalios teorijos pavyzdį.

Pavyzdys 2.11.1. "Pagaliukų teorija" $\mathcal{P} = \langle A, F, Ax, R\rangle$, kur:

(1) $A = \{|\}$;

(2) $F = A^*$, t.y., bet kurį reiškinį laikome formule;

(3) $Ax = \{||\}$

(4) $R = \{R_1\}$, kur R_1 yra taisyklė
$$\frac{A}{AA},$$
o A yra bet kuri teorijos formulė. Čia reiškinys AA reiškia dvi formules A užrašytas be tarpo greta viena kitos (konkatenacijos operacija). Pagal šios teorijos formulių apibrėžimą tokiu atveju AA taip pat yra formulė.

Šioje teorijoje galime, pavyzdžiui, įrodyti "8 pagaliukų" teoremą $||||||||$:

(1) $||$ (aksioma);

(2) $||||$ (gauname iš (1) pagal išvedimo taisyklę R_1);

(3) $||||||||$ (gauname iš (2) pagal išvedimo taisyklę R_1).

Nesunku įsitikinti, kad ši teorija turi skaičią aibę teoremų pavidalo $||\dots||$, kur pagaliukų skaičius yra 2^n, $n = 1, 2, \dots$.

Dabar apibrėšime formalią teiginių teoriją $L = \langle A, F, Ax, R\rangle$:

(1) $A = \{\neg, \to, (,), p_1, p_2, \dots\}$, kur ženklus $\neg, \to$ vadiname *primityviomis operacijomis*, o ženklus $p_1, p_2, \dots$ vadiname *kintamaisiais*.

(2) (a) Bet kuris kintamasis yra formulė.
(b) Jei A ir B yra formulės, tai $(\neg A)$ ir $(A \to B)$ taip pat yra formulės.
(c) Reiškinys yra formulė tada ir tik tada, kai tai išplaukia iš (a) ir (b).

(3) Teorijos L aksiomomis bus formulės

(A1) $(A \to (B \to A))$;
(A2) $((A \to (B \to C)) \to ((A \to B) \to (A \to C)))$;
(A3) $((\neg B \to \neg A) \to ((\neg B \to A) \to B))$,

kur A, B, C yra bet kokios teorijos L formulės. Taigi, ši teorija turi 3 *aksiomų schemas*, į kurias įstatinėdami bet kurias formules gauname begalinę aksiomų aibę.

(4) Vienintelė teorijos L taisyklė yra taisyklė modus ponens:

$$\text{(MP)} \quad \frac{A \to B, A}{B}.$$

Pavyzdys 2.11.2. Formalioje teorijoje L įrodysime teoremą $(A \to A)$:

(1) $((A \to ((A \to A) \to A)) \to ((A \to (A \to A)) \to (A \to A)))$ (aksiomų schema (A2), į kurią įstatėme $A = A$, $B = (A \to A)$ ir $C = A$);

(2) $(A \to ((A \to A) \to A))$ (aksiomų schema (A1));

(3) $((A \to (A \to A)) \to (A \to A)))$ (iš (1) ir (2) pagal taisyklę MP);

(4) $(A \to (A \to A))$ (aksiomų schema (A1));

(5) $(A \to A)$ (iš (3) ir (4) pagal taisyklę MP);

Uždaviniai

1. Apibrėžkite tokią “pagaliukų teoriją”, kurios teoremomis bus formulės, sudarytos iš bet kokio nelyginio pagaliukų skaičiaus.

2. Teiginių teorijoje L įrodykite šias teoremas:

   (1) $((\neg A \to A) \to A)$;

   (2) $((\neg B \to \neg A) \to (A \to B))$.

3 skyrius

B$\bar{u}$lio funkcijos ir schemos

3.1 Paprasčiausios B$\bar{u}$lio funkcijos

B$\bar{u}$lio funkcija vadiname bet kurią funkciją $f : \{0,1\}^n \to \{0,1\}$, $n = 1, 2, \ldots$. Sakysime, kad funkcija f priklauso nuo n kintamųjų (yra n*-funkcija*) ir žymėsime $f(x_1, \ldots, x_n)$. Tokias funkcijas dar vadina *loginėmis* arba *logikos algebros funkcijomis*. B$\bar{u}$lio n-funkcijų aibę žymime P_2^n. Visų B$\bar{u}$lio funkcijų aibę žymime P_2. Taigi, $P_2 = \bigcup_{n=1}^{\infty} P_2^n$.

Kadangi B$\bar{u}$lio funkcijos yra baigtinės funkcijos, tai B$\bar{u}$lio funkciją f galima nusakyti jos reikšmių lentele:

x_1	x_2	$\ldots$	x_{n-1}	x_n	$f(x_1, \ldots, x_n)$
0	0	$\ldots$	0	0	$f(00\ldots00)$
0	0	$\ldots$	0	1	$f(00\ldots01)$
0	0	$\ldots$	1	0	$f(00\ldots10)$
$\vdots$	$\vdots$		$\vdots$	$\vdots$	$\vdots$
1	1	$\ldots$	1	1	$f(11\ldots11)$

Tokioje lentelėje kintamųjų reikšmių rinkiniai $\alpha = (\alpha_1, \ldots, \alpha_n) \in \{0,1\}^n$ paprastai išdėstomi nat$\bar{u}$ralia tvarka, atitinkančia skaičių $0, 1, \ldots, 2^n - 1$ dvejetainius kodus. Taigi ši lentelė turi 2^n eilučių (tai išplaukia ir iš teoremos 1.1). Kadangi paskutiniame stulpelyje kiekvienoje eilutėje galima pasirinkti bet kurią iš dviejų reikšmių 0 arba 1, tai skirtingų lentelių, o kartu ir skirtingų n-funkcijų bus $|P_2^n| = 2^{2^n}$.

Iš aukščiau pateiktos formulės matyti, kad yra tik keturios B$\bar{u}$lio funkcijos, priklausančios nuo vieno kintamojo. Tai *konstanta* 0, *konstanta* 1, *tapatingoji funkcija* $f(x) = x$ ir *neiginys* $f(x) = \overline{x}$ (arba $\neg x$; skaitome "ne x"). Šių funkcijų lentelės atrodo taip:

x	0	1	x	$\overline{x}$
0	0	1	0	1
1	0	1	1	0

Dabar išvardinsime kelias pagrindines 2-funkcijas (iš viso 2-funkcijų yra 16):

1. $f_1(x_1, x_2) = (x_1 \,\&\, x_2)$ — *konjunkcija* arba *loginė daugyba* (skaitome "x_1 ir x_2"); dažnai vietoje $x_1 \,\&\, x_2$ rašysime taip pat x_1x_2, praleisdami konjunkcijos ženklą taip kaip algebroje įprasta praleisti daugybos ženklą;
2. $f_2(x_1, x_2) = (x_1 \vee x_2)$ — *disjunkcija* arba *loginė sudėtis* (skaitome "x_1 arba x_2");
3. $f_3(x_1, x_2) = (x_1 \to x_2)$ — *implikacija* (skaitome "iš x_1 išplaukia x_2");
4. $f_4(x_1, x_2) = (x_1 \leftrightarrow x_2)$ — *ekvivalentumas* (skaitome "x_1 ekvivalentus x_2");
5. $f_4(x_1, x_2) = (x_1 \oplus x_2)$ — *suma moduliu 2* (skaitome "x_1 plius x_2");
6. $f_5(x_1, x_2) = (x_1 \,|\, x_2)$ — *Šeferio funkcija* (skaitome "x_1 Šeferio operacija su x_2").

Šių funkcijų lentelės atrodo taip:

x_1	x_2	$(x_1 \,\&\, x_2)$	$(x_1 \vee x_2)$	$(x_1 \to x_2)$	$(x_1 \leftrightarrow x_2)$	$(x_1 \oplus x_2)$	$(x_1 \,\vert\, x_2)$
0	0	0	0	1	1	0	1
0	1	0	1	1	0	1	1
1	0	0	1	0	0	1	1
1	1	1	1	1	1	0	0

Kitame skyrelyje įrodysime, kad kiekvieną bet kurio skaičiaus kintamųjų Būlio funkciją galima išreikšti keleto iš šių pagrindinių funkcijų superpozicija, t.y. užrašyti formule, į kurią įeina tik pasirinktos funkcijos. Tegu $B \subseteq P_2$. Pateiksime induktyvų *formulės virš* B apibrėžimą:

1. Kiekvieną $f(x_1, \ldots, x_n) \in B$ vadiname formule virš B.
2. Tegu $f(x_1, \ldots, x_n) \in B$, o $F_1, \ldots, F_n$ — formulės virš B arba kintamieji iš aibės $\{x_1, x_2, \ldots\}$. Tada $f(F_1, \ldots, F_n)$ — formulė virš B.

Kaip jau minėjome, bet kurią funkciją, išreikštą formule virš B, vadiname *funkcijų iš* B *superpozicija*.

Pavyzdys 3.1.1. Pažymėkime B_0 aibę $\{\overline{x}\,, x_1 \,\&\, x_2, x_1 \vee x_2\}$. Pagal formulės apibrėžimą reiškinys $(x_1 \vee (\overline{x}_1 \,\&\, x_2))$ yra formulė virš B_0, o reiškinys $x_1 \,\& \vee \overline{x}_2$ — ne formulė. Išreikšime implikaciją formule virš B_0: $(x_1 \to x_2) = (\overline{x_1} \vee x_2)$. Tai, kad abiejose lygybės pusėse stovinčios formulės išreiškia vieną ir tą pačią Būlio funkciją, galima įsitikinti iš reikšmių lentelės:

x_1	x_2	$(x_1 \to x_2)$	$\overline{x_1}$	$(\overline{x_1} \vee x_2)$
0	0	1	1	1
0	1	1	1	1
1	0	0	0	0
1	1	1	0	1

Paprastai išoriniai ir kai kurie vidiniai skliaustai formulėse praleidžiami, laikantis tokios operacijų atlikimo tvarkos: (1) neiginys $\neg$; (2) konjunkcija $\&$; (3) disjunkcija $\vee$; (4) likusios operacijos turi vienodą prioritetą; jų atlikimo tvarka nustatoma skliaustų pagalba. Kadangi konjukcija turi visas aritmetinės daugybos operacijos savybes, tai panašiai kaip aritmetikoje daugybos ženklas, konjunkcijos ženklas formulėse taip pat kartais yra praleidžiamas. Taigi pavyzdžio 3.1.1 pirmąją formulę rašysime pavidalu $x_1 \vee \overline{x}_1 x_2$.

Funkcijos $f(x_1, \ldots, x_n)$ kintamąjį x_i $(i = 1, \ldots, n)$ vadinsime *esminiu* kintamuoju, jei egzistuoja tokios likusių kintamųjų reikšmės $\alpha_1, \ldots, \alpha_{i-1}, \alpha_{i+1}, \ldots, \alpha_n$, kad

$$f(\alpha_1, \ldots, \alpha_{i-1}, 0, \alpha_{i+1}, \ldots, \alpha_n) \neq f(\alpha_1, \ldots, \alpha_{i-1}, 1, \alpha_{i+1}, \ldots, \alpha_n).$$

Kintamuosius, kurie nėra esminiai, vadinsime *fiktyviais*. Dvi funkcijas $f(x_1, \ldots, x_n)$ ir $g(y_1, \ldots, y_n)$ (kur dalis kintamųjų x_i gali sutapti su kai kuriais iš kintamųjų y_j) vadiname *lygiomis*, jei jos skiriasi tik fiktyviais kintamaisiais, t.y. jų reikšmių lentelės, į kurias įtrauksime visus skirtingus kintamuosius $x_1, \ldots, x_n$ ir $y_1, \ldots, y_m$, sutaps.

Pavyzdys 3.1.2. Funkcijos $g(x_1, x_2) = x_1 \& (x_2 \vee \overline{x}_2)$ kintamasis x_1 yra esminis, nes įstatę $\alpha_2 = 0$, gauname $g(0,0) \neq g(1,0)$. Tuo tarpu kintamasis x_2 yra fiktyvus, nes $g(0,0) = g(0,1)$ ir $g(1,0) = g(1,1)$. Tai matyti iš šios funkcijos reikšmių lentelės, kuri įrodo ir tai, kad $g(x_1, x_2)$ yra lygi tapatingai funkcijai $f(x_1) = x_1$:

x_1	x_2	$\overline{x}_2$	$x_2 \vee \overline{x}_2$	$x_1 \& (x_2 \vee \overline{x}_2)$
0	0	1	1	0
0	1	0	1	0
1	0	1	1	1
1	1	0	1	1

Jei dvi formulės F_1 ir F_2 virš tos pačios aibės išreiškia dvi lygias funkcijas f_1 ir f_2, tai tas formules vadinsime *ekvivalenčiomis* ir žymėsime $F_1 \sim F_2$. Pavyzdžiui, $x \& \overline{x} \sim 0$, $x \vee \overline{x} \sim 1$, $x \vee 0 \sim x$, $\overline{\overline{x}} \sim x$ (*dvigubo neigimo* dėsnis) ir t.t. Išvardinsime dar keletą ekvivalenčių formulių:

1. Pažymėkime $(x_1 \circ x_2)$ bet kurią iš operacijų $(x_1 \& x_2), (x_1 \vee x_2), (x_1 \oplus x_2)$. Visos jos yra *asociatyvios* ir *komutatyvios*:

$$\begin{aligned} x_1 \circ (x_2 \circ x_3) &\sim (x_1 \circ x_2) \circ x_3, \\ x_1 \circ x_2 &\sim x_2 \circ x_1. \end{aligned}$$

2. Konjunkcija ir disjunkcija tenkina du *distributyvumo* dėsnius:

$$\begin{aligned} x_1 \& (x_2 \vee x_3) &\sim (x_1 \& x_2) \vee (x_1 \& x_3), \\ x_1 \vee (x_2 \& x_3) &\sim (x_1 \vee x_2) \& (x_1 \vee x_3). \end{aligned}$$

3. *De Morgano* dėsniai (neiginio įkėlimas į skliaustus):

$$\begin{aligned} \overline{x_1 \,\&\, x_2} &\sim \overline{x}_1 \vee \overline{x}_2, \\ \overline{x_1 \vee x_2} &\sim \overline{x}_1 \,\&\, \overline{x}_2. \end{aligned}$$

Pažymėsime, kad visos šios formulės liks ekvivalenčios, įstačius į jas vietoje kintamųjų bet kokias formules, pvz. $\overline{F_1 \,\&\, F_2} \sim \overline{F}_1 \vee \overline{F}_2$. Pasinaudojus šia savybe, dviejų formulių ekvivalentumą galima įrodyti ne tik lentelės pagalba, bet ir suvedant vieną formulę į kitą, keletą kartų pritaikius formules, apie kurias jau žinoma, kad jos ekvivalenčios (pavyzdžiui, taikant aukščiau išvardintas ekvivalenčias formules). Tokiu atveju sakome, kad viena formulė gauta iš kitos *ekvivalenčiais pertvarkymais*.

Pavyzdys 3.1.3. Įrodysime, kad $\overline{x_1 \vee \overline{x}_1 x_2} \sim \overline{x}_1 \overline{x}_2$. Turime:

$$\begin{aligned} \overline{x_1 \vee \overline{x}_1 x_2} &\sim \overline{x}_1 \,\&\, \overline{\overline{x}_1 x_2} \quad \text{(de Morgano dėsnis)} \\ &\sim \overline{x}_1 \,\&\, (\overline{\overline{x}}_1 \vee \overline{x}_2) \quad \text{(de Morgano dėsnis)} \\ &\sim \overline{x}_1 \,\&\, (x_1 \vee \overline{x}_2) \quad \text{(dvigubo neigimo dėsnis)} \\ &\sim (\overline{x}_1 x_1) \vee (\overline{x}_1 \overline{x}_2) \quad \text{(distributyvumas)} \\ &\sim 0 \vee (\overline{x}_1 \overline{x}_2) \\ &\sim \overline{x}_1 \overline{x}_2. \end{aligned}$$

Uždaviniai

1. Sudarę funkcijų reikšmių lenteles, įrodykite, kad funkcijos f_1 ir f_2 yra lygios:
   (a) $f_1(x_1, x_2) = x_1 \leftrightarrow x_2, \quad f_2(x_1, x_2) = (x_1 \to x_2) \,\&\, (x_2 \to x_1)$;
   (b) $f_1(x_1, x_2) = x_1 \oplus x_2, \quad f_2(x_1, x_2) = (x_1 \vee x_2) \,\&\, \overline{(x_1 \,\&\, x_2)}$.
2. Pasinaudoję funkcijų reikšmių lentelėmis, išreikškite funkcijas $f_1(x_1, x_2) = x_1 \,|\, x_2$ ir $f_2(x_1, x_2) = x_1 \leftrightarrow x_2$ formulėmis virš $B_0 = \{\overline{x}\,, x_1 \,\&\, x_2, x_1 \vee x_2\}$.
3. Išreikškite visas šiame skyrelyje apibrėžtas Bulio funkcijas per neiginį ir disjunkciją, t.y. formulėmis virš $B_{\neg,\vee} = \{\overline{x}\,, x_1 \vee x_2\}$.
4. Išreikškite visas šiame skyrelyje apibrėžtas Bulio funkcijas per Šeferio operaciją, t.y. formulėmis virš $B_| = \{x_1 | x_2\}$.
5. Duota Bulio funkcija $f(x_1, x_2, x_3) = (\overline{x}_1 \to x_2) \,\&\, \overline{(\overline{x_1} \,\&\, \overline{x_2} \,\&\, x_3)}$. Kurie jos kintamieji yra esminiai ir kurie fiktyvūs?
6. Ekvivalenčiais pertvarkymais suprastinkite formules:
   (a) $x_1 \,\&\, (\overline{x}_2 \vee 1) \,\&\, (\overline{x}_1 \vee x_3)$;
   (b) $\overline{(\overline{x}_1 \vee \overline{x}_2)} \vee (x_1 \,\&\, (x_3 \vee 0))$.

3.2 Pilnos Būlio funkcijų sistemos

Pažvelgus į pagrindinių Būlio funkcijų reikšmių lentelės skyrelyje 3.1, nesunku pastebėti, kad šios funkcijos nesunkiai išreiškiamos vienos per kitas, tame tarpe, visas galime išreikšti per konjunkciją, disjunkciją ir neiginį, pvz.: $x_1 \,|\, x_2 = \overline{x_1 \,\&\, x_2}$, $x_1 \to x_2 = \overline{x}_1 \vee x_2$, $x_1 \oplus x_2 = (x_1 \vee x_2) \,\&\, \overline{(x_1 \,\&\, x_2)}$. Pasirodo, visas Būlio funkcijas galima išreikšti formulėmis, kuriose naudojamos tik trys šios operacijos ($^-$, $\vee$ ir $\&$).

Tegu $\sigma \in \{0,1\}$. Apibrėžkime kintamojo laipsnį x^σ, laikydami $x^1 = x$ ir $x^0 = \overline{x}$. Nesunku patikrinti, kad $\alpha^\sigma = 1 \Leftrightarrow \alpha = \sigma$ $(\alpha,\ \sigma \in \{0,1\})$.

Teorema 3.2.1 (Apie funkcijos išskaidymą). *Kiekvieną Būlio funkciją $f(x_1, \ldots, x_n)$ galima išreikšti pavidalu*

$$f(x_1, \ldots, x_n) = \bigvee_{(\sigma_1,\ldots,\sigma_m)\ \in\{0,1\}^m} x_1^{\sigma_1} \,\&\, \cdots \,\&\, x_m^{\sigma_m} \,\&\, f(\sigma_1, \ldots, \sigma_m, x_{m+1}, \ldots, x_n),$$

kur $1 \le m \le n$.

Įrodymas. Pakanka įrodyti, kad ši lygybė teisinga bet kuriam kintamųjų reikšmių rinkiniui $\alpha \in \{0,1\}^n$, t.y.

$$f(\alpha_1, \ldots, \alpha_n) = \bigvee_{(\sigma_1,\ldots,\sigma_m)\ \in\{0,1\}^m} \alpha_1^{\sigma_1} \,\&\, \cdots \,\&\, \alpha_m^{\sigma_m} \,\&\, f(\sigma_1, \ldots, \sigma_m, \alpha_{m+1}, \ldots, \alpha_n).$$

Kadangi $\alpha_1^{\sigma_1} \,\&\, \cdots \,\&\, \alpha_m^{\sigma_m} = 1 \Leftrightarrow \alpha_1 = \sigma_1, \ldots, \alpha_m = \sigma_m$, tai dešinė lygybės pusė lygi $f(\alpha_1, \ldots, \alpha_n)$, nes kitoms σ reikšmėms konjunkcijos virsta 0. $\square$

Išvada 3.2.1. *Kiekvieną Būlio funkciją $f(x_1, \ldots, x_n) \not\equiv 0$ galima išreikšti pavidalu*

$$f(x_1, \ldots, x_n) = \bigvee_{\sigma:\ f(\sigma)=1} x_1^{\sigma_1} \,\&\, \cdots \,\&\, x_n^{\sigma_n}.$$

Įrodymas. Pakanka teoremos formuluotėje paimti $m = n$ ir iš gautos disjunkcijos išmesti nulinius narius (t.y. tokius, kuriems $f(\alpha_1, \ldots, \alpha_n) = 0$), naudojantis ekvivalenčiomis formulėmis $x \vee 0 \sim x$, $x \,\&\, 1 \sim x$. $\square$

Kiekvieną konjunkciją $K = x_{i_1}^{\sigma_1} \,\&\, \cdots \,\&\, x_{i_k}^{\sigma_k}$, kur $x_{i_j} \in \{x_1, x_2, \ldots\}$ ir $i_j \ne i_l$ (kai $j \ne l$), vadinsime *elementaria konjunkcija*. Išraišką $D = K_1 \vee \cdots \vee K_m$, kur $K_i \ne K_j$ (kai $i \ne j$) — elementarios konjunkcijos, vadinsime *normaliąja disjunkcine forma* (arba NDF). Pagal išvadą 3.2.1 kiekvieną Būlio funkciją $f \ne 0$ galima išreikšti normaliąja disjunkcine forma.

Baigtinę ar begalinę funkcijų sistemą ($=$ aibę) $B = \{f_1, f_2, \ldots\}$ vadinsime *pilna*, jei kiekvieną $f \in P_2$ galima išreikšti formule virš B. Naudodamiesi išvada 3.2.1, gauname tokią teoremą.

Teorema 3.2.2. *Sistema* $B_0 = \{\overline{x}\,,\ x_1 \,\&\, x_2,\ x_1 \vee x_2\}$ *— pilna.*

Įrodymas. Pagal išvadą 3.2.1 kiekviena $f \not\equiv 0$ išreiškiama formule virš B_0. Belieka pastebėti, kad $0 \sim x_1 \,\&\, \overline{x}_1$. $\square$

Dabar pateikiame vieną teoremą, kuri padeda įrodyti, kad pasirinktoji Būlio funkcijų aibė yra pilna.

Teorema 3.2.3. *Tegu* B *— pilna sistema ir kiekvieną jos funkciją galima išreikšti formule virš* C. *Tada ir* C *— pilna.*

Įrodymas. Bet kurią Būlio funkciją galime išreikšti funkcijų iš B superpozicija, o po to kiekvieną funkciją iš B pakeisti funkcijų iš C superpozicija. $\square$

Išvada 3.2.2. *Sistemos* $B_{\&} = \{\overline{x}, x_1 \,\&\, x_2\}$ *ir* $B_{\vee} = \{\overline{x}, x_1 \vee x_2\}$ *— pilnos.*

Įrodymas. Kadangi B_0 yra pilna pagal teoremą 3.2.2, o $x_1 \vee x_2 \sim \overline{\overline{x}_1 \,\&\, \overline{x}_2}$, tai ir $B_{\&}$ — pilna. Naudodamiesi formule $x_1 \,\&\, x_2 \sim \overline{\overline{x}_1 \vee \overline{x}_2}$ analogiškai gauname, kad ir $B_{\vee}$ — pilna. $\square$

Išvada 3.2.3. *Šeferio funkcija* $x_1 \,|\, x_2$ *sudaro pilną sistemą.*

Įrodymas. $\overline{x} \sim x \,|\, x,\ x_1 \,\&\, x_2 \sim \overline{x_1 \,|\, x_2} \sim (x_1 \,|\, x_2) \,|\, (x_1 \,|\, x_2)$. Lieka pasinaudoti išvada 3.2.2. $\square$

Išvada 3.2.4. *Sistema* $B_{\mathrm{p}} = \{x_1 \oplus x_2, x_1 \,\&\, x_2, 1\}$ *yra pilna.*

Įrodymas. Kadangi $\overline{x} \sim x \oplus 1$, o aibė $\{\overline{x}, x_1 \,\&\, x_2\}$ pilna pagal išvadą 3.2.2, tai ir aibė B_{p} pilna. $\square$

Dabar panagrinėsime specialios struktūros formules virš aibės B_{p}.
Žegalkino polinomu vadiname reiškinį

$$\begin{aligned} A_{12\ldots n}x_1x_2\cdots x_n \;&\oplus\; A_{12\ldots n-1}x_1x_2\cdots x_{n-1} \oplus \cdots \oplus A_{23\ldots n}x_2x_3\cdots x_n \oplus \cdots \\ &\oplus\; A_{12}x_1x_2 \oplus \cdots \oplus A_{n-1,n}x_{n-1}x_n \oplus A_1x_1 \oplus \cdots \oplus A_nx_n \oplus A_0, \end{aligned}$$

kur koeficientai $A_{12\ldots n}, \ldots, A_0 \in \{0, 1\}$.

Teorema 3.2.4. *Kiekvieną Būlio funkciją vieninteliu būdu galima išreikšti Žegalkino polinomu.*

Įrodymas. Pagal išvadą 3.2.4 kiekvieną Būlio funkciją galima išreikšti formule virš B_{p}. Naudojantis distributyvumo dėsniu $x_1(x_2 \oplus x_3) \sim x_1x_2 \oplus x_1x_3$ ir ekvivalenčiomis formulėmis $x \,\&\, x \sim x$, $x \,\&\, 1 \sim x$, $x \oplus 0 \sim x$, $x \oplus x \sim 0$ bei $x \,\&\, 0 \sim 0$, šią formulę visada galima pertvarkyti į norimą pavidalą.

Belieka įrodyti vienatinumą. Žinome, kad $|P_2^n| = 2^{2^n}$. Kadangi visų galimų sandaugų $x_{i_1}x_{i_2}\cdots x_{i_k}$ $(0 \le k \le n)$ yra $|\mathcal{P}(\{x_1,\ldots,x_n\})| = 2^n$, o koeficientas $A_{i_1 i_2 \ldots i_k}$ lygus 0 arba 1, tai skirtingų Žegalkino polinomų yra taip pat 2^{2^n}. Taigi kiekvienai Būlio funkcijai teks lygiai po vieną polinomą. □

Iš teoremos įrodymo gauname du būdus duotos Būlio funkcijos Žegalkino polinomui rasti:

(1) *ekvivalenčiais pertvarkymais*;

(2) *neapibrėžtinių koeficientų metodu*: užrašyti duotos funkcijos $f(x_1,\ldots,x_n)$ Žegalkino polinomą $P(x_1,\ldots,x_n)$ su neapibrėžtais koeficientais $A_{i_1 i_2 \ldots i_k}$, kiekvienam vektoriui α sulyginti reikšmes $f(\alpha) = P(\alpha)$ ir išspręsti (skaičiuojant moduliu 2) gautą tiesinių lygčių sistemą, turinčią 2^n lygčių ir 2^n nežinomųjų; tokia sistema visada turės vienintelį sprendinį.

Pavyzdys 3.2.1. Dviem būdais rasime implikacijos $x_1 \to x_2$ Žegalkino polinomą.

(1) $x_1 \to x_2 \sim \overline{x_1} \vee x_2 \sim \overline{x_1\overline{x_2}} \sim x_1(x_2 \oplus 1) \oplus 1 \sim x_1x_2 \oplus x_1 \oplus 1$.

(2) Tegu $x_1 \to x_2 = Ax_1x_2 \oplus Bx_1 \oplus Cx_2 \oplus D$. Iš lygčių sistemos

$$\begin{cases} D = 1 \\ B \oplus D = 0 \\ C \oplus D = 1 \\ A \oplus B \oplus C \oplus D = 1 \end{cases}$$

gauname $D = B = A = 1$, $C = 0$, taigi $x_1 \to x_2 = x_1x_2 \oplus x_1 \oplus 1$.

Uždaviniai

1. Įrodykite, kad sistema $B = \{x_1 \to x_2,\ 0\}$ yra pilna.
2. Raskite disjunkcijos $x_1 \vee x_2$ Žegalkino polinomą: (a) ekvivalenčiais pertvarkymais; (b) neapibrėžtinių koeficientų metodu.

3.3 Uždaros Būlio funkcijų klasės

Dvi n-funkcijas $f(x_1,\ldots,x_n)$ ir $g(y_1,\ldots,y_n)$ vadiname *kongruenčiomis*, jei jos skiriasi tik kintamųjų žymėmis, t.y. galima rasti bijekciją $k : \{x_1,\ldots,x_n\} \to \{y_1,\ldots,y_n\}$ tokią, kad $f(k(x_1),\ldots,k(x_n)) = g(y_1,\ldots,y_n)$. Pavyzdžiui, $x_1 \,\&\, x_2$ ir $x_2 \,\&\, x_3$ yra kongruenčios funkcijos.

Tegu B — Būlio funkcijų sistema. Laikysime, kad kartu su su bet kuria funkcija f iš B aibei B priklauso ir visos funkcijos, kongruenčios f. Aibės B *uždariniu* $[B]$ vadiname aibę

visų galimų funkcijų iš B superpozicijų. Jei $[B] = B$, aibę B vadiname *uždara*. Pavyzdžiui, $\{x_1, x_1 \,\&\, x_2, x_1 \,\&\, x_2 \,\&\, x_3, \ldots\}$ — uždara funkcijų sistema, o $\{0, \overline{x}\}$ — neuždara, nes $[\{0, \overline{x}\}] = \{0, 1, x, \overline{x}\}$. Beje, pasinaudoję uždarumo terminu galėjome apibrėžti ir pilnumą: B — pilna, jei $[B] = P_2$.

Šiame skyrelyje apibrėšime penkias svarbias Bulio funkcijų klases ir įrodysime jų uždarumą.

Klasė T_0

Klasei T_0 priklauso funkcijos, *išsaugančios nulį*:

$$T_0 = \{ f(x_1, \ldots, x_n) \colon f(0, \ldots, 0) = 0\}.$$

Klasė T_1

Klasei T_1 priklauso funkcijos, *išsaugančios vienetą*:

$$T_1 = \{ f(x_1, \ldots, x_n) \colon f(1, \ldots, 1) = 1\}.$$

Sau dualių funkcijų klasė

Tegu $f(x_1, \ldots, x_n) \in P_2$. Funkciją $f^*(x_1, \ldots, x_n) = \overline{f(\overline{x_1}, \ldots, \overline{x_n})}$ vadiname *dualia funkcijai* f. Pavyzdžiui, $(x_1 \,\&\, x_2)^* = \overline{\overline{x_1} \,\&\, \overline{x_2}} = x_1 \vee x_2$, $(\overline{x})^* = \overline{\overline{x}} = \overline{x}$. *Sau dualių funkcijų* klase vadiname

$$S = \{ f(x_1, \ldots, x_n) \colon f^* = f\}.$$

Monotoninių funkcijų klasė

Skyrelyje 1.2 apibrėžėme tvarkos sąryšį aibėje $\{0, 1\}^n$:

$$\alpha \preceq \beta \iff \alpha_i \leq \beta_i \quad \forall i = 1, \ldots, n.$$

Funkciją $f(x_1, \ldots, x_n)$ vadiname *monotonine*, jei ji išsaugo tvarkos sąryšį $\preceq$, t.y.

$$\alpha \preceq \beta \Rightarrow f(\alpha) \leq f(\beta) \quad \forall \alpha, \beta \in \{0, 1\}^n.$$

Pavyzdžiui, $x_1 \,\&\, x_2$ — monotoninė funkcija, o $x_1 \to x_2$ — ne. Taigi

$$M = \{ f(x_1, \ldots, x_n) \colon f \text{— monotoninė}\}.$$

Tiesinių funkcijų klasė

B$\overline{\text{u}}$lio funkciją vadiname *tiesine*, jei jos Žegalkino polinomo laipsnis yra 1, t.y. $f(x_1,\dots,x_n) = A_1x_1 \oplus \dots \oplus A_nx_n \oplus A_0$, kur $A_0, A_1, \dots, A_n \in \{0,1\}$. Akivaizdu, kad kiekvienam n tėra dvi tiesinės n-funkcijos, iš esmės priklausančios nuo visų savo kintamųjų: $x_1 \oplus x_2 \oplus \dots \oplus x_n$ ir jos neiginys $x_1 \oplus x_2 \oplus \dots \oplus x_n \oplus 1$. Tiesinių funkcijų klasę žymėsime L:

$$L = \{\, f(x_1,\dots,x_n) \colon f \text{— tiesinė}\}.$$

Teorema 3.3.1. *B$\overline{\text{u}}$lio funkcijų klasės T_0, T_1, S, M, L yra uždaros.*

Įrodymas. Pagal uždarumo apibrėžimą klasė K bus uždara, jei iš to, kad $f(y_1,\dots,y_m) \in K$ ir $f_1(x_1,\dots,x_n),\dots f_m(x_1,\dots,x_n) \in K$ išplaukia, kad ir

$$g(x_1,\dots,x_n) = f(f_1(x_1,\dots,x_n),\dots f_m(x_1,\dots,x_n)) \in K$$

(pridedant fiktyvius kintamuosius visada galima pasiekti, kad funkcijos $f_1,\dots,f_m$ priklausytų nuo tų pačių kintamųjų $x_1,\dots,x_n$).

(1) $K = T_0$:

$$g(0,\dots,0) = f(f_1(0,\dots,0),\dots,f_m(0,\dots,0)) = f(0,\dots,0) = 0.$$

(2) $K = T_1$: įrodymas analogiškas atvejui (1).

(3) $K = S$: jei $f \in S$, tai $f(\alpha) = \neg f(\neg\alpha)$ ir $f(\neg\alpha) = \neg f(\alpha)$, todėl

$$\begin{aligned} g^*(\alpha) = \neg g(\neg\alpha) &= \neg f(f_1(\neg\alpha),\dots,f_m(\neg\alpha)) \\ &= \neg f(\neg f_1(\alpha),\dots,\neg f_m(\alpha)) = f(f_1(\alpha),\dots,f_m(\alpha)) = g(\alpha). \end{aligned}$$

(4) $K = M$: jei $\alpha \preceq \beta$, tai $f_i(\alpha) \le f_i(\beta)\ \forall i = 1,\dots,m$, taigi

$$\begin{aligned} \alpha \preceq \beta &\Rightarrow (f_1(\alpha),\dots,f_m(\alpha)) \preceq (f_1(\beta),\dots,f_m(\beta)) \\ &\Rightarrow g(\alpha) = f(f_1(\alpha),\dots,f_m(\alpha)) \le f(f_1(\beta),\dots,f_m(\beta)) = g(\beta). \end{aligned}$$

(5) $K = L$: pirmo laipsnio polinomų suma bus pirmo laipsnio polinomas arba konstanta. □

Uždaviniai

1. Nustatykite kurios iš šių sistemų yra uždaros: (a) $\{0, 1, x, \overline{x}\}$, (b) $\{x_1, x_1 \oplus x_2, x_1 \oplus x_2 \oplus x_3, \dots\}$ ir (c) $\{x_1, x_1 \vee x_2, x_1 \vee x_2 \vee x_3, \dots\}$.
2. Nustatykite kurios iš šių funkcijų priklauso klasėms T_0 ir T_1: (a) $\overline{x}$, (b) $x_1 \vee x_2$ ir (c) $x_1 \mid x_2$.
3. Nustatykite kurios iš šių funkcijų priklauso klasei S: (a) $\overline{x}$, (b) $x_1 \vee x_2$ ir (c) $x_1 \mid x_2$.
4. Nustatykite kurios iš šių funkcijų priklauso klasei M: (a) $\overline{x}$, (b) $x_1 \vee x_2$ ir (c) $x_1 \mid x_2$.
5. Nustatykite kurios iš šių funkcijų priklauso klasei L: (a) $\overline{x}$, (b) $x_1 \vee x_2$ ir (c) $x_1 \mid x_2$.

3.4 Pilnumo kriterijus

Lema 3.4.1. *Jei* $f(x_1,\dots,x_n)\notin S$*, tai iš* f *ir funkcijų* x *ir* $\overline{x}$ *galima gauti konstantą* 0 *arba* 1.

Įrodymas. Kadangi $f\notin S$, tai atsiras vektorius $\alpha\in\{0,1\}^n$ toks, kad

$$f(\overline{\alpha_1},\dots,\overline{\alpha_n})=f(\alpha_1,\dots,\alpha_n).$$

Į funkciją $f(x_1,\dots,x_n)$ įstatę n funkcijų $\phi_i(x)=x^{\alpha_i}$ $(i=1,\dots,n)$ gauname vieno kintamojo funkciją

$$\phi(x)=f(\phi_1(x),\dots,\phi_n(x)).$$

Ši funkcija tenkina

$$\begin{aligned}\phi(0) &= f(\phi_1(0),\dots,\phi_n(0))=f(0^{\alpha_1},\dots,0^{\alpha_n})=f(\overline{\alpha_1},\dots,\overline{\alpha_n})=f(\alpha_1,\dots,\alpha_n)\\ &= f(1^{\alpha_1},\dots,1^{\alpha_n})=f(\phi_1(1),\dots,\phi_n(1))=\phi(1).\end{aligned}$$

Taigi, $\phi(x)=$ const (0 arba 1). $\square$

Tegu $\alpha,\beta\in\{0,1\}^n$. Vektorius α ir β vadiname *kaimyniniais*, jei jie skiriasi tik viena savo koordinate, t.y.,

$$\begin{aligned}\alpha &= (\alpha_1,\dots,\alpha_{i-1},\alpha_i,\alpha_{i+1},\dots,\alpha_n),\\ \beta &= (\alpha_1,\dots,\alpha_{i-1},\overline{\alpha_i},\alpha_{i+1},\dots,\alpha_n).\end{aligned}$$

Vienetiniame n-mačiame kube $\{0,1\}^n$ kaimyninius vektorius atitinka tos kubo viršūnės, kurios yra sujungtos briauna.

Lema 3.4.2. *Jei* $f(x_1,\dots,x_n)\notin M$*, tai iš* f*, funkcijos* x *ir konstantų* 0 *ir* 1 *galima gauti neiginį* $\overline{x}$.

Įrodymas. Kadangi $f\notin M$, tai atsiras du vektoriai α ir β tokie, kad $\alpha\preceq\beta$, ir $f(\alpha)>f(\beta)$.

Pirmiausia parodysime, kad tada taip pat atsiras du kaimyniniai vektoriai $\widetilde{\alpha}$ ir $\widetilde{\beta}$ tokie, kad $\widetilde{\alpha}\preceq\widetilde{\beta}$ ir $f(\widetilde{\alpha})>f(\widetilde{\beta})$. Iš tiesų, jei α ir β nėra kaimyniniai, tai jie skiriasi ne viena, bet t koordinačių. Kadangi $\alpha\preceq\beta$, tai vektoriuje α tos koordinatės lygios 0, o vektoriuje β lygios 1. Todėl nesunku parinkti $t-1$ vektorių $\alpha^1,\dots,\alpha^{t-1}$ tokius, kad

$$\alpha=\alpha^0\preceq\alpha^1\preceq\dots\preceq\alpha^{t-1}\preceq\alpha^t=\beta$$

ir šioje grandinėje bet kurie du greta stovintys vektoriai α^{i-1} ir α^i būtų kaimyniniai. Kadangi $f(\alpha^0)>f(\alpha^t)$, tai būtinai atsiras du gretimi vektoriai α^{i-1} ir α^i kuriems taip pat $f(\alpha^{i-1})>f(\alpha^i)$, Pažymėkime šiuos vektorius $\widetilde{\alpha}$ ir $\widetilde{\beta}$. Galime laikyti, kad

$$\widetilde{\alpha} = (\alpha_1,\dots,\alpha_{i-1},0,\alpha_{i+1},\dots,\alpha_n),$$

$$\widetilde{\beta} \;=\; (\alpha_1, \ldots, \alpha_{i-1}, 1, \alpha_{i+1}, \ldots, \alpha_n).$$

Apibrėžę naują funkciją

$$\phi(x) = f(\alpha_1, \ldots, \alpha_{i-1}, x, \alpha_{i+1}, \ldots, \alpha_n),$$

gauname

$$\begin{array}{rcl} \phi(0) & = & f(\alpha_1, \ldots, \alpha_{i-1}, 0, \alpha_{i+1}, \ldots, \alpha_n) = f(\widetilde{\alpha}) > f(\widetilde{\beta}) \\ & = & f(\alpha_1, \ldots, \alpha_{i-1}, 1, \alpha_{i+1}, \ldots, \alpha_n) = \phi(1). \end{array}$$

Tai reiškia, kad $\phi(0) = 1$, o $\phi(1) = 0$, taigi $\phi(x) = \overline{x}$. $\square$

Lema 3.4.3. *Jei $f(x_1, \ldots, x_n) \notin L$, tai iš f, funkcijų x, $\overline{x}$ ir konstantų 0 ir 1 galima gauti konjunkciją $x_1 \,\&\, x_2$.*

Įrodymas. Kadangi $f(x_1, \ldots, x_n) \notin L$, tai į šios funkcijos Žegalkino polinomą b$\overline{\text{u}}$tinai įeina dviejų ar daugiau kintamųjų sandauga $x_{i_1} x_{i_2} \cdots x_{i_s}$. Galime laikyti, kad tarp šių kintamųjų yra x_1 ir x_2. Tada funkcijos $f(x_1, \ldots, x_n)$ Žegalkino polinomą galima išreikšti pavidalu

$$\begin{array}{rcl} \displaystyle\sum_{(i_1, \ldots, i_s)} A_{i_1 \ldots i_s} x_{i_1} x_{i_2} \cdots x_{i_s} & = & x_1 x_2 f_1(x_3, \ldots, x_n) \oplus x_1 f_2(x_3, \ldots, x_n) \\ & & \oplus\, x_2 f_3(x_3, \ldots, x_n) \oplus f_4(x_3, \ldots, x_n), \end{array}$$

kur dėl polinomo vienatinumo $f_1(x_3, \ldots, x_n) \not\equiv 0$ (priešingu atveju sandauga $x_1 x_2$ išnyktų). Tegu $\alpha_3, \ldots, \alpha_n$ toks kintamųjų reikšmių rinkinys, kad $f_1(\alpha_3, \ldots, \alpha_n) = 1$. Įstatę į f šias reikšmes, gauname dviejų kintamųjų funkciją

$$g(x_1, x_2) = f(x_1, x_2, \alpha_3, \ldots, \alpha_n) = x_1 x_2 \oplus \alpha x_1 \oplus \beta x_2 \oplus \gamma,$$

kur $\alpha, \beta, \gamma \in \{0, 1\}$. Kadangi turime funkcijas x, $\overline{x}$ ir konstantas, tai galime vietoje kintamųjų x_1 ir x_2 įstatyti atitinkamai reiškinius $x_1 \oplus \beta$ ir $x_2 \oplus \alpha$ ir dar pridėti konstantą $\alpha\beta \oplus \gamma$. Gauname naują funkciją

$$\begin{array}{rcl} \phi(x_1, x_2) & = & g(x_1 \oplus \beta, x_2 \oplus \alpha) \oplus \alpha\beta \oplus \gamma \\ & = & (x_1 \oplus \beta)(x_2 \oplus \alpha) \oplus \alpha(x_1 \oplus \beta) \oplus \beta(x_2 \oplus \alpha) \oplus \gamma \oplus \alpha\beta \oplus \gamma = x_1 x_2, \end{array}$$

taigi lema įrodyta. $\square$

Teorema 3.4.1 (Pilnumo kriterijus). *Funkcijų sistema B pilna tada ir tik tada kai ji nėra poaibis nė vienos iš klasių T_0, T_1, S, M, L.*

Įrodymas. Būtinumas. Tarkime, kad B yra kurios nors iš duotų klasių poaibis: $B \subseteq C$, kur $C = T_0$, T_1, S, M arba L. Kadangi jos visos yra uždaros, tai

$$[B] \subseteq [C] = C.$$

Taigi, B negali būti pilna, nes nė viena iš šių klasių nesutampa su P_2.

Pakankamumas. Kadangi $B \not\subseteq T_0, T_1, S, M, L$, tai aibėje B atsiras funkcijos $f_0 \notin T_0$, $f_1 \notin T_1$, $f_s \notin S$, $f_m \notin M$, $f_l \notin L$ (kai kurios iš jų gali sutapti tarpusavyje). Įrodymas susideda iš trijų žingsnių.

(1) Naudodami f_0, f_1 ir f_s gausime konstantas 0 ir 1. Turime $f(0, \ldots, 0) = 1$. Galimi du atvejai.

(a) $f_0(1, \ldots, 1) = 1$. Šiuo atveju funkcija $\phi(x) = f(x, \ldots, x) \equiv 1$, taigi konstantą 1 jau gavome. Įstatę visus vienetus į funkciją f_1 gauname kitą konstantą: $f_1(1, \ldots, 1) = 0$.

(b) $f_0(1, \ldots, 1) = 0$. Tada funkcija $\phi(x) = f(x, \ldots, x) = \overline{x}$, nes $\phi(0) = 1$ ir $\phi(1) = 0$. Turėdami neiginį pagal lemą apie sau nedualią funkciją iš f_s galime gauti kurią nors konstantą, o iš jos neiginio pagalba ir kitą konstantą.

(2) Naudodami konstantas 0 ir 1 pagal lemą apie nemonotoninę funkciją iš f_m galime gauti neiginį $\overline{x}$.

(3) Turėdami konstantas ir neiginį pagal lemą apie netiesinę funkciją iš funkcijos f_l galime gauti konjunkciją $x_1 \,\&\, x_2$.

Kadangi sistema $\{\overline{x}, x_1 \,\&\, x_2\}$ pilna, tai pagal teoremą 3.2.3 ir sistema B pilna.

Uždaviniai

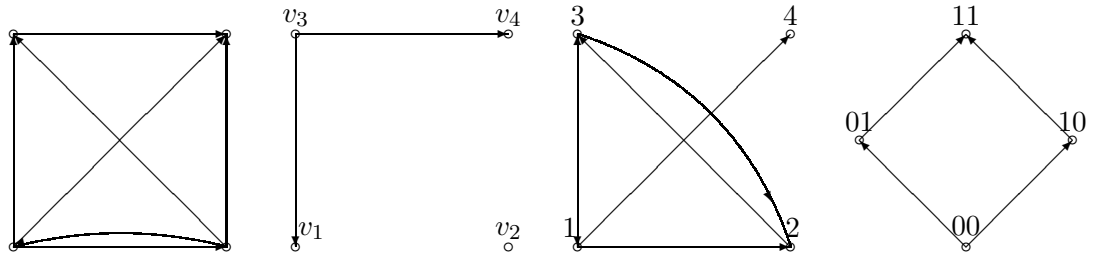
1. Naudodami pilnumo kriterijų ištirkite ar duotos funkcijų sistemos yra pilnos:

   (a) $B = \{x_1 \to x_2, x_1 \oplus x_2\}$,

   (b) $B = \{\overline{x}, x_1x_2x_3 \vee \overline{x_1x_2}x_3 \vee \overline{x_1}x_2\}$.

3.5 Būlio schemos

Tegu V — netuščia aibė. *Orientuotu grafu* (*orgrafu*) vadinsime porą $G = (V, E)$, kur $E \subseteq V^2 \setminus \text{diag}(V^2)$, o $\text{diag}(V^2) = \{(v, v)\colon v \in V\}$ yra aibės V^2 įstrižainė. Aibės V elementus vadinsime *viršūnėmis*, o aibės E elementus — *lankais*. Taigi, kiekvienas lankas $e \in E$ yra 2-vektorius sudarytas iš skirtingų viršūnių: $e = (u, v)$, kur $u, v \in V$ ir $u \neq v$. Tokias viršūnes vadinsime *sujungtomis lanku* e ir sakysime, kad lankas e išeina iš viršūnės u ir įeina į viršūnę v. Nedidelės eilės orgrafus (grafo eilė $= |V|$) patogu vaizduoti grafiškai plokščia diagrama, kurioje kiekviena viršūnė $v \in V$ vaizduojama tašku (arba mažu apskritimu) su greta prirašyta žyme v, o kiekvienas lankas $e = (u, v)$ vaizduojamas linija, jungiančia taškus pažymėtus u ir v su rodykle linijos gale, nukreipta į viršūnę v (ši linija negali daugiau eiti per jokią kitą viršūnę). Pav. 3.1



3.1: Orgrafų pavyzdžiai.

pavaizduoti keturi orgrafai. Kadangi ne kiekvieną grafą galima pavaizduoti plokščia diagrama taip, kad lankai nesusikirstų, tai reikia atkreipti dėmesį, kad paprasti lankų susikirtimo taškai nelaikomi grafo viršūnėmis.

Pavyzdys 3.5.1. Bet kuri aibė V su joje apibrėžtu dviviečiu antirefleksyviu sąryšiu σ yra orientuotas grafas (V, σ). Pavyzdžiui, vienetinio dvimačio kubo viršūnių aibė $\{0,1\}^2$ su tvarkos sąryšiu $\prec$ yra orgrafas vaizduojamas pav. refgrafai(d).

Grafo viršūnės v *įėjimo laipsniu* indg(v) vadiname į šią viršūnę įeinančių lankų skaičių, o *išėjimo laipsniu* outdg(v) — iš šios viršūnės išeinančių lankų skaičių. *Keliu jungiančiu dvi grafo viršūnes* u *ir* v vadiname lankų seką $K(u,v) = \{(v_0,v_1),(v_1,v_2),\ldots,(v_{k-1},v_k)\}$, kurioje $v_0 = u$, $v_k = v$ ir du gretimi sekos lankai turi bendrą viršūnę. Kelią $K(u,v)$ sudarančių lankų skaičių k vadiname kelio $K(u,v)$ ilgiu. Vietoje lankų išvardijimo kelią $K(u,v)$ dažnai apibrėžia išvardijant tik viršūnes, per kurias eina šis kelias: $K(u,v) = \{u, v_1, \ldots, v_{k-1}, v\}$. Netuščią kelią $K(u,u)$ vadiname *ciklu*. Pav. refgrafai(c) pavaizduotas grafas turi ciklą $C = \{1231\}$.

Būlio schema virš bazės $B_2 = \{\&, \vee, \neg\}$ vadiname orientuotą grafą be ciklų, kurio viršūnių įėjimo laipsnis yra ne didesnis už 2, ir viršūnės turi žymes, atitinkančias tokias taisykles:

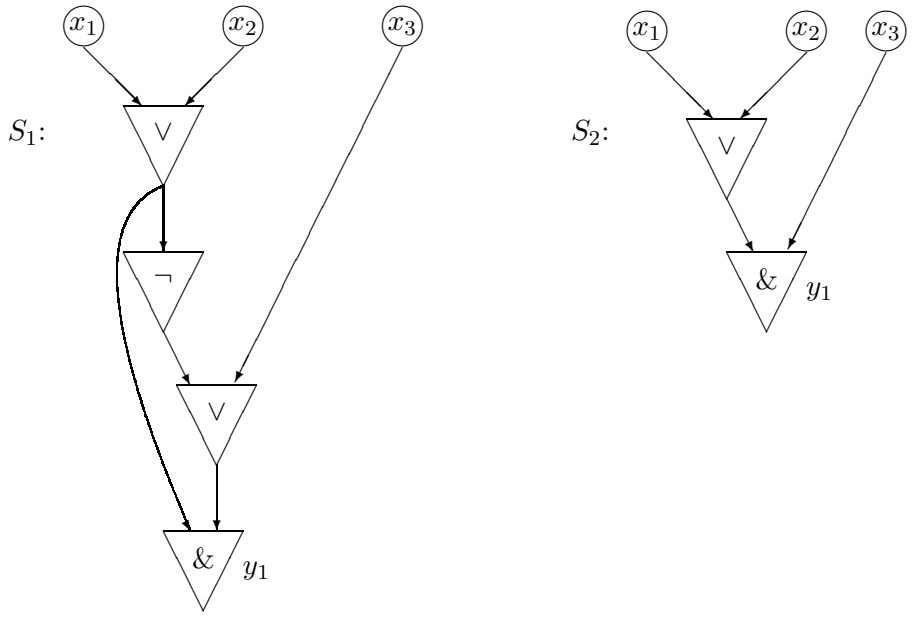
(1) jei indg$(v) = 0$, tai viršūnė v pažymėta žyme x_i, kur $x_i \in \{x_1, \ldots, x_n\}$;

(2) jei indg$(v) = 1$, tai viršūnė v pažymėta neiginio ženklu $\neg$;

(3) jei indg$(v) = 2$, tai viršūnė v pažymėta konjunkcijos ženklu $\&$ arba disjunkcijos ženklu $\vee$;

(4) kai kurios viršūnės papildomai pažymėtos žymėmis y_j, kur $y_j \in \{y_1, \ldots, y_m\}$;

Pirmo tipo grafo viršūnes vadinsime schemos *įėjimais* ir jas vaizduosime skrituliukais $\bigcirc$, antro ir trečio tipo viršūnes vadinsime *funkciniais elementais* ir vaizduosime trikampiais $\bigtriangledown$. Ketvirto tipo viršūnes vadinsime schemos *išėjimais*. Kiekvienas išėjimas yra kartu ir įėjimas arba funkcinis elementas. Pastebėkime, kad iš viršūnių išeinančių lankų skaičius yra neribojamas. Būlio schemas



3.2: Dvi Bulio schemos, realizuojančios tą pačią Bulio funkciją.

taip pat vadina schemomis iš funkcinių elementų bei loginėmis schemomis. Pav. 3.2 pavaizduotos dvi Bulio schemos S_1 ir S_2.

Induktyviai apibrėšime Bulio schemos S realizuojamą Bulio funkcijų šeimą $F = \{f_1, \ldots, f_m\}$:

(1) Įėjimams x_i priskiriame funkcijas $f(x_1, \ldots, x_n) = x_i$.

(2) Jei į funkcinį elementą E, kur $E \in \{\neg, \&, \vee\}$ ateina lankai iš viršūnių G ir H (arba tik G neiginio atveju), kurioms jau priskirtos funkcijos g ir h, tai elementui E priskiriame funkciją $e = E(g, h)$.

(3) Schema S realizuoja funkcijų šeimą $F = \{f_1, \ldots, f_m\}$, kur funkcijos $f_1, \ldots, f_m$ yra priskirtos jos išėjimams $y_1, \ldots, y_m$.

Kai $m = 1$, tada tiesiog sakysime, kad schema S realizuoja Bulio funkciją f. Pavyzdžiui, pav. 3.2 abi schemos S_1 ir S_2 realizuoja tą pačią funkciją $f(x_1, x_2, x_3) = (x_1 \vee x_2) \,\&\, x_3$, nes $(x_1 \vee x_2) \,\&\, (\neg(x_1 \vee x_2) \vee x_3) \sim (x_1 \vee x_2) \,\&\, x_3$. Dalinis Bulio schemų atvejis yra formulės virš bazės B_0: kiekvieną formulę atitinka schema su tiek funkcinių elementų, kiek bazės B_0 elementų $\neg, \vee, \&$ įeina į formulę. Tokioje schemoje iš kiekvienos viršūnės išeina ne daugiau kaip vienas lankas. Kadangi schemose kiekvienos viršūnės išėjimo laipsnis neribojamas, tai schemos yra galingesnis Bulio funkcijų realizavimo modelis už formules. Pavyzdžiui, aukščiau pateikta

formulė atitinkanti schemą S_1 turi 5 funkcinius elementus, o pati schema S_1 tik 4 funkcinius elementus.

Schemos S *sudėtingumu* $L(S)$ vadiname funkcinių elementų skaičių schemoje S. Pav. 3.2 vaizduojamų schemų sudėtingumai yra $L(S_1) = 4$ ir $L(S_2) = 2$. Matome, kad vieną Būlio funkciją galima realizuoti skirtingo sudėtingumo schemomis. Būlio funkcijų šeimos $F = \{f_1, \ldots, f_m\}$ *sudėtingumu* vadiname

$$L(F) = \min_{S \text{ realizuoja } F} L(S).$$

Vienos funkcijos f sudėtingumą žymėsime tiesiog $L(f)$. Pavyzdžiui, nesunku įrodyti, kad pav. 3.2 dviem schemomis realizuojamos Būlio funkcijos $f(x_1, x_2, x_3) = (x_1 \vee x_2) \,\&\, x_3$ sudėtingumas yra lygus 2. Iš tikrųjų, iš vienos pusės $L(f) \leq 2$, nes $L(S_2) = 2$, tuo tarpu iš kitos pusės šios funkcijos negalima realizuoti schema iš 1 elemento, nes ji iš esmės priklauso nuo visų savo 3 kintamųjų, o vienas funkcinis elementas būtų sujungtas tik su 2 ar 1 įėjimu.

Baigdami šį skyrelį pateiksime trivialų viršutinį Būlio funkcijų sudėtingumo įvertį. Kitame skyrelyje mes šį įvertį dar pagerinsime.

Lema 3.5.1. *Kiekvienai Būlio funkcijai* $f(x_1, \ldots, x_n)$ $L(f) \leq n2^{n+1}$.

Įrodymas. Jei $f(x_1, \ldots, x_n) \equiv 0$, tai naudojant formulę $0 \sim x \,\&\, \neg x$ funkciją f galime realizuoti schema iš dviejų elementų. Kadangi $2 \leq n2^{n+1}$ $\forall n = 1, 2, \ldots$, tai šiuo atveju teorema teisinga.

Jei $f(x_1, \ldots, x_n) \not\equiv 0$, tai pagal išvadą 3.2.1 ją galima išreikšti tobula normaliąja disjunkcine forma

$$f(x_1, \ldots, x_n) = \bigvee_{\sigma:\ f(\sigma)=1} x_1^{\sigma_1} \,\&\, \cdots \,\&\, x_n^{\sigma_n}. \tag{3.1}$$

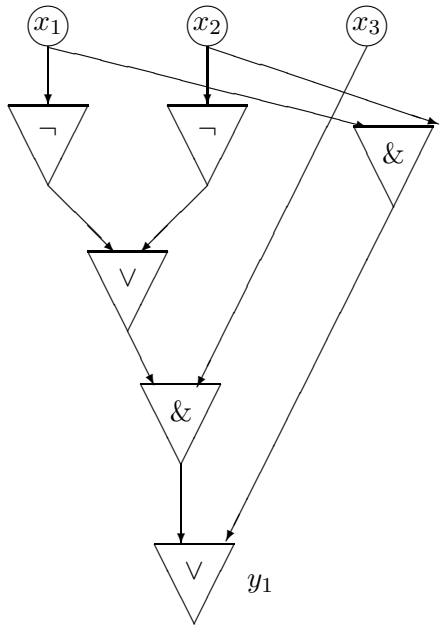
Kiekvienai elementariai konjunkcijai $x_1^{\sigma_1} \,\&\, \cdots \,\&\, x_n^{\sigma_n}$ realizuoti pakanka $\leq 2n - 1$ funkcinio elemento ($\leq n$ neiginių ir $n - 1$ konjunkcijos). Pačių elementarių konjunkcijų šioje formoje bus $r \leq 2^n$. Tam, kad pagaliau gauti funkciją f dar reikės panaudoti $r - 1$ disjunkciją sujungti elementarioms konjunkcijoms į vieną formulę. Tokiu būdu pagal formulę (3.1) gausime schemą S tokią, kad

$$L(S) \leq (2n - 1)r + r - 1 = 2nr - 1 \leq 2n \cdot 2^n = n2^{n+1}.$$

Taigi $L(f) \leq n2^{n+1}$. $\square$

Uždaviniai

1. Supaprastinkite Būlio schemą, pavaizduotą Pav. 3.3.
2. Realizuokite kuo paprastesne Būlio schema (virš bazės B_2) funkciją, išreikštą formule $(x_1 \oplus x_2) \,\&\, \neg(x_1 \,\&\, \neg x_2)$.
3. Realizuokite Būlio schema Būlio funkciją $f(x_1, x_2, x_3) = 1 \iff$ lygiai vienas $x_i = 1$.



3.3: Schema sudėtingumo 6.

3.6 Būlio funkcijų sudėtingumas

Kadangi Būlio schemos yra susiję su kompiuteriuose naudojamomis mikroschemomis, tai svarbu mokėti konstruoti kuo paprastesnes schemas ir mokėti įvertinti Būlio funkcijų sudėtingumą. Pažymėkime

$$L(n) = \max_{f(x_1,\dots,x_n)} L(f), \quad n = 1, 2, \dots .$$

Funkciją $L : \mathbb{N} \to \mathbb{N}$ vadina Shannon'o funkcija. Ji išreiškia sudėtingiausių Būlio funkcijų sudėtingumo priklausomybę nuo argumentų skaičiaus n.

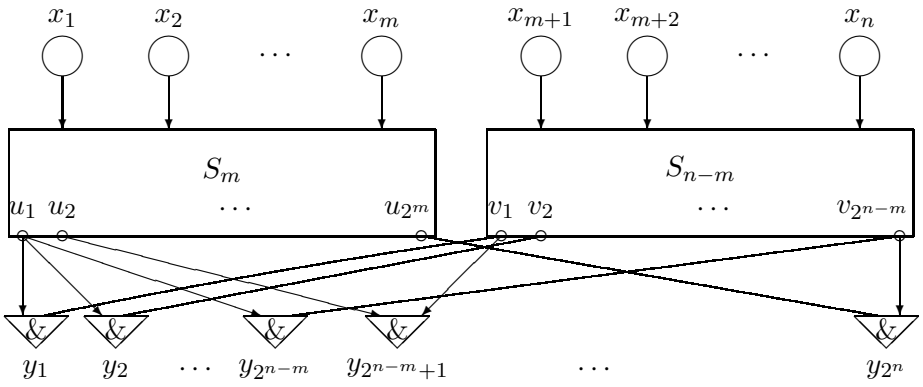
Priminsime pora apibrėžimų iš funkcijų teorijos, kurie mums bus reikalingi kalbant apie sudėtingumą. Tegu $f, g : \mathbb{N} \to \mathbb{R}$. Žymėsime:

(1) $f \lesssim g$ ("f asimptotiškai mažesnė arba lygi g"), jei

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} \le 1;$$

(2) $f \sim g$ ("f asimptotiškai lygi g"), jei $f \lesssim g$ ir $g \lesssim f$.

Dabar realizuosime visas elementarias konjunkcijas ilgio n schema, turinčia n įėjimų ir 2^n išėjimų. Pažymėkime $K_n = \{x_1^{\sigma_1} \,\&\, \cdots \,\&\, x_n^{\sigma_n},\ \sigma \in \{0,1\}^n\}$.



3.4: Būlio schema, realizuojanti visas elementarias konjunkcijas ilgio n.

Lema 3.6.1. $L(K_n) \sim 2^n$.

Įrodymas. Kadangi schema realizuojanti K_n turi 2^n išėjimų, tai $L(K_n) \gtrsim 2^n$, todėl tereikia įrodyti $L(K_n) \lesssim 2^n$. Tegu $m = [n/2]$, kur $[x]$ reiškia skaičiaus x sveikąją dalį. Kiekvieną konjunkciją ilgio n galima išskaidyti į dvi konjunkcijas ilgio m ir $n-m$:

$$x_1^{\sigma_1} \,\&\, \cdots \,\&\, x_n^{\sigma_n} = (x_1^{\sigma_1} \,\&\, \cdots \,\&\, x_m^{\sigma_m}) \,\&\, (x_{m+1}^{\sigma_{m+1}} \,\&\, \cdots \,\&\, x_n^{\sigma_n}).$$

Vienai konjunkcijai $x_1^{\sigma_1} \,\&\, \cdots \,\&\, x_m^{\sigma_m}$ realizuoti reikia $\leq 2m-1$ funkcinių elementų ($\leq m$ neiginių ir $m-1$ konjunkcijos). Kadangi tokių konjunkcijų viso yra 2^m, tai schemos S_m, realizuojančios visas elementarias konjunkcijas, sudarytas iš pirmųjų m kintamųjų, sudėtingumas bus $\leq (2m-1)2^m$. Analogiškai, schemos S_{n-m}, realizuojančios visas elementarias konjunkcijas, sudarytas iš paskutinių $n-m$ kintamųjų, sudėtingumas bus $\leq (2(n-m)-1)2^{n-m}$. Sujungę konjunkcijos elementais kiekvieną schemos S_m išėjimą su kiekvienu schemos S_{n-m} išėjimu, gauname schemą S, realizuojančią sistemą K_n (žr. pav. 3.4). Jos sudėtingumas

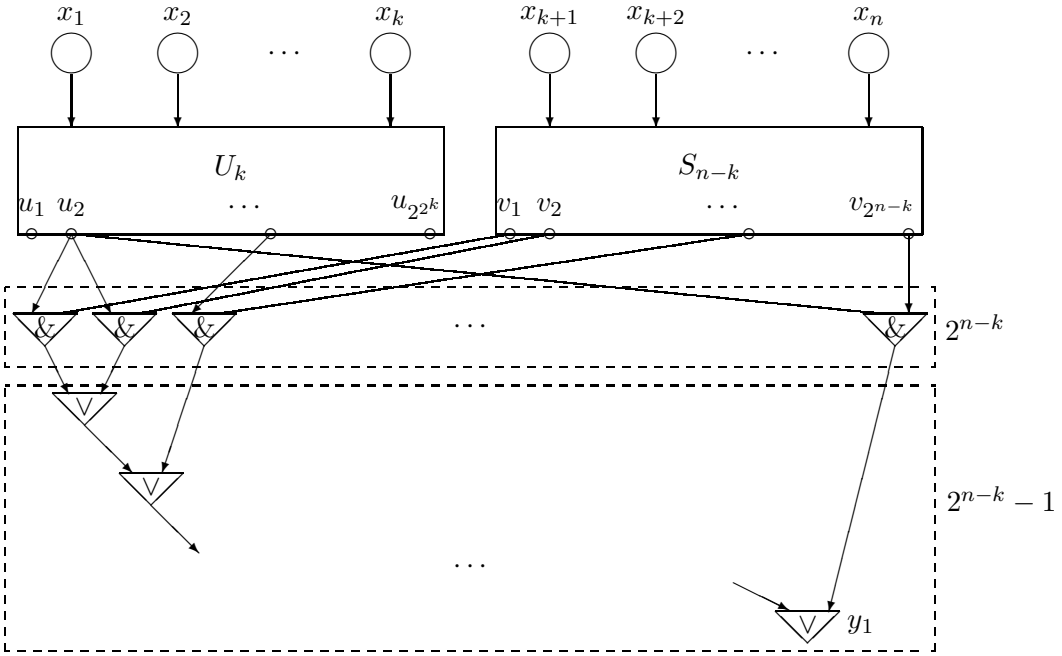
$$\begin{aligned} L(S) &= L(S_m) + L(S_{n-m}) + 2^n \\ &\leq (2m-1)2^m + (2(n-m)-1)2^{n-m} + 2^n \\ &\leq (n-1)2^{n/2} + (n+1)2^{n/2+1} + 2^n \sim 2^n, \end{aligned}$$

kur pasinaudojome nelygybėmis $m = [n/2] \leq n/2$ ir $n - m \leq n/2 + 1$.

Dabar pagerinsime skyrelyje 3.5 gautą viršutinį Būlio funkcijų sudėtingumo įvertį.

Teorema 3.6.1.

$$L(n) \lesssim 12 \cdot \frac{2^n}{n}.$$



3.5: B ̄ulio schema, realizuojanti B ̄ulio funkciją $f(x_1, \ldots, x_n)$.

Įrodymas. Duota B ̄ulio funkcija $f(x_1, \ldots, x_n)$. Konstruosime ją realizuojančią schemą S. Pagal teoremą 3.2.1

$$f(x_1, \ldots, x_n) = \bigvee_{(\sigma_{k+1}, \ldots, \sigma_n) \in \{0,1\}^{n-k}} x_{k+1}^{\sigma_{k+1}} \,\&\, \cdots \,\&\, x_n^{\sigma_n} \,\&\, f(x_1, \ldots, x_k, \sigma_{k+1}, \ldots, \sigma_n),$$

kur $1 \leq k \leq n$.

Schemą S sudarys 3 blokai (žr. pav. 3.5):

(1) universali schema U_k, trivialiai (kaip lemoje 3.5.1) realizuojanti visas 2^{2^k} B ̄ulio funkcijų, priklausančių nuo pirmųjų k kintamųjų,

(2) schema S_{n-k}, realizuojanti pagal lemą 2.5 visas paskutiniųjų $n-k$ kintamųjų elementarias konjunkcijas ilgio $n-k$, ir

(3) sujungimų blokas, sudarytas iš 2^{n-k} konjunkcijų ir $2^{n-k}-1$ disjunkcijos.

Gautos schemos S sudėtingumas bus

$$L(S) \leq 3 \cdot 2^{n-k} + k \cdot 2^{k+1} \cdot 2^{2^k}.$$

Parinkę $k = [\log_2 n] - 1$, gauname nelygybes $k \le \log_2 n - 1$ ir $n - k \le n - \log_2 n + 2$ (nes $-k \ge \log_2 n - 2$). Naudodami taip pat nelygybę $2^k \le n/2$, gauname

$$L(S) \le 3 \cdot 2^{n-\log_2 n+2} + \log_2 n \cdot n \cdot 2^{n/2} = 12 \cdot \frac{2^n}{n} + \mathrm{o}\left(\frac{2^n}{n}\right) \sim 12 \cdot \frac{2^n}{n}.$$

Naudojant sudėtingesnę konstrukciją šis viršutinis įvertis buvo 12 kartų pagerintas (t.y., vietoje konstantos 12 iš tikrųjų tinka konstanta 1). Be to, buvo gautas asimptotiškai tokio pat dydžio įvertis iš apačios. Taigi, teisingas toks rezultatas, kurį pateikiame be įrodymo:

Teorema 3.6.2.

$$L(n) \sim \frac{2^n}{n}.$$

Taip pat buvo įrodyta, kad *beveik visos* Būlio funkcijos $f(x_1, \ldots, x_n)$ yra labai sudėtingos, t.y. joms reikia $\ge 2^n/n$ funkcinių elementų. "Beveik visos" reiškia, kad paprastesnių Būlio funkcijų skaičius yra be galo mažas dydis palyginus su visų n kintamųjų Būlio funkcijų skaičiumi 2^{2^n}. Tačiau tie įrodymai yra nekonstruktyvūs, jie parodo tik, kad egzistuoja sudėtingos Būlio funkcijos, bet nepateikia pačių funkcijų. Taigi, paradoksalu, bet nežinoma jokia konkreti, t.y. efektyviai išreikšta, didelio sudėtingumo Būlio funkcijų seka $\{f_n(x_1, \ldots, x_n)\}$. Kol kas yra gauti tik tiesiniai (t.y., pavidalo $\mathrm{const} \cdot n$) apatiniai Būlio funkcijų sudėtingumo įverčiai. Pavyzdžiui, tiesinėms funkcijoms $l_n(x_1, \ldots, x_n) = x_1 \oplus \cdots \oplus x_n$ yra įrodyta, kad $L(l_n) = 4n - 4$. Efektyvių Būlio funkcijų sudėtingumo apatinių įverčių problema yra viena iš svarbiausių neišspręstų diskrečiosios matematikos ir informatikos problemų.

Uždaviniai

1. Įrodykite $L(l_n) \le 4n - 4$, sukonstruodami minimalią schemą, realizuojančią tiesinę funkciją $l_n = x_1 \oplus \cdots \oplus x_n$, $n = 2, 3, \ldots$. *Nurodymas*: atskirai panagrinėkite atvejus $n = 2$ ir $n = 3$.

2. Realizuokite Būlio schema, turinčia $2n$ įėjimų ir $n + 1$ išėjimą, dvejetainių skaičių sudėtį, t.y., jei schemos įėjimus pažymėsime $x_1, \ldots, x_n$ ir $z_1, \ldots, z_n$, o išėjimus $y_1, \ldots, y_{n+1}$, tai turi būti teisinga lygybė tarp dvejetainių skaičių:

   $$y_{n+1} y_n \ldots y_1 = x_n \ldots x_1 + z_n \ldots z_1,$$

   kur vyriausi dvejetainių skaičių bitai stovi skaičių pradžioje. Tokią Būlio schemą vadina dvejetainiu sumatoriumi. Koks jūsų pasiūlytos schemos sudėtingumas?

4 skyrius

Kodavimas

Kodavimas nuo seno vaidina svarbų vaidmenį matematikoje.

Pavyzdžiai. 1. *Dešimtainė skaičiavimo sistema* — tai natūraliųjų skaičių kodavimo būdas. Romėniški skaičiai — kitas natūraliųjų skaičių kodavimo būdas.

2. *Dekartinės koordinatės* — tai būdas geometrines figūras užkoduoti skaičiais.

Su kompiuterių atsiradimu kodavimo reikšmė labai išaugo. Dabar tai pagrindinis uždavinys daugelyje programavimo sričių, pavyzdžiui,

- duomenų (skaičių, teksto, grafinių objektų) vaizdavimas kompiuterio atmintyje,
- informacijos apsauga,
- klaidų ištaisymas siunčiant duomenis nepatikimais ryšių kanalais,
- duomenų spaudimas duomenų bazėse.

Net patį programos rašymą kartais (ir, beje, visiškai teisingai) vadina kodavimu, o programos tekstą — kodu.

Šioje kurso dalyje mes susipažinsime su keliais svarbiausiais kodavimo teorijos uždaviniais: duomenų spaudimu, klaidas taisančiais kodais, kriptografija. Tai labai plačios diskrečiosios matematikos sritys, ir šiame kurse mes jas tik vos paliesime. Prieš tai dar supažindinsiu su bazine kodavimo teorijos dalimi: abėcėliniu kodavimu ir optimaliu kodavimu.

Pavyzdžiai.

- *Duomenų spaudimas:*
  - *Morzės kodas.* Jį sudarė amerikietis S. Morzė kaip priedą prie kito savo išradimo — telegrafo. Dviejų simbolių — taško ir brūkšnio — kombinacijomis koduojamos raidės ir skaičiai. Jų atskyrimui dar naudojama pauzė. Kad tekstą būtų galima užkoduoti kuo trumpesne taškų ir brūkšnių seka, Morzė savo kodą sudarė taip, kad dažniau vartojamas raides atitiktų trumpesni kodai, o rečiau vartojamas — ilgesni. Pavyzdžiui, raidę e atitinka kodas tik iš vieno simbolio — taško, o raidę y — iš keturių: `-.--`.

  - Kompiuteriuose dažnai naudojamas failų spaudimas, kad jie užimtų mažiau vietos. Windows aplinkoje dažniausiai sutinkamas suspaustų failų formatas — zip.
  - Šiuolaikiniuse modemuose, perduodant skaitmeninius duomenis, jie taip pat spaudžiami.
  - Siunčiant kitų planetų nuotraukas iš kosminių zondų irgi naudojamas duomenų spaudimas.

- *Klaidas taisantys kodai:* prieš siunčiant simbolius nepatikimu kanalu, kur jie gali būti iškraipyti, jie užkoduojami, kad gavėjas turėtų galimybę pagal gautus galbūt iškraipytus simbolius atkurti tai, kas buvo siųsta.

  - Paprasčiausias kodavimo būdas — pakartoti kiekvieną siunčiamą simbolį n kartų. Gavus tą n simbolių seką dekoduojama tuo simboliu, kuris daugiau kartų pasitaiko. Akivaizdu, jog tokiu būdu didindami ryšio patikimumą, didiname ir perdavimo išlaidas bei mažiname perdavimo greitį.
  - Klaidas taisantys kodai naudojami visose skaitmeninėse technologijose: kompiuteriuose, modemuose, kompaktinėse plokštelėse, kosminiuose zonduose ir taip toliau. Pavyzdžiui, dėl kompaktinėje plokštelėje naudojamų klaidas taisančių kodų muzikos kokybė nenukentėtų net ir pradūrus joje 2mm skersmens skylę!

- *Kriptografija:*

  - Kriptografija naudojama nuo senų laikų slaptiems pranešimams šifruoti. Jau Romos imperijos laikais buvo naudojami slapti kodai.
  - Sakoma, kad sąjungininkų pergalę Antrajame pasauliniame kare didele dalimi nulėmė tai, kad jie sugebėjo iššifruoti vokiečių slaptus kodus.
  - Iš simetrinių šifravimo sistemų dabar labiausiai paplitusi DES, kurią neužilgo pakeis AES, nes didėjant kompiuterių greičiams ir galimybėms DES darosi per silpna.
  - Perspektyviausiomis laikomos vadinamos viešo rakto kriptosistemos, pavyzdžiui, RSA, kuriose naudojama pora raktų — viešas ir privatus. Norintys man perduoti pranešimą, užšifruoja jį mano viešu, visiems prieinamu, raktu, o jį dešifruoti galiu tik aš, turėdamas savo privatų raktą.
  - Norintys susirašinėti slapta gali šifruoti savo elektroninį paštą. Tam yra specialios programos, pavyzdžiui, PGP.

4.1 Abėcėlinis kodavimas

Šiame paragrafe šnekėsime apie tokius kodus, kur pranešimas skaidomas po vieną simbolį, ir kiekvienas simbolis užkoduojamas atskirai, nepriklausomai nuo kitų. Tokie kodai vadinami *abėcėliniais kodais*.

Abėcėle vadinsime pasirinktą baigtinę netuščią aibę $\mathcal{A}$, jos elementus vadinsime *simboliais*. Baigtinės aibės $\mathcal{B}$ elementų skaičių žymėsime $|\mathcal{B}|$. Abėcėlę $\mathcal{A} = \{0, 1\}$ vadiname *dvinare* abėcėle.

Apibrėžimas. *Baigtinę abėcėlės $\mathcal{A}$ simbolių seką $a_1a_2\cdots a_s$ vadinsime* s ilgio žodžiu. *Jei* $\mathbf{x}$ *yra žodis, tai* $|\mathbf{x}|$ *žymi jo ilgį. Jei* $\mathbf{x}$, $\mathbf{y}$ *yra du tos pačios abėcėlės žodžiai, tai* $\mathbf{xy}$ *žymi žodį, kuris gaunamas tiesiog sujungiant* $\mathbf{x}$ *ir* $\mathbf{y}$. *Abėcėlės $\mathcal{A}$ ilgio m žodžių aibę žymėsime $\mathcal{A}^m$, visų žodžių aibę — $\mathcal{A}^*$.*

Lengva pastebėti, kad

$$\mathcal{A}^* = \bigcup_{m\geq 0} \mathcal{A}^m.$$

Apibrėžimas. *Tegu $\mathcal{A}$, $\mathcal{B}$ yra dvi abėcėlės.* Kodu *vadinsime injektyvų atvaizdį $c : \mathcal{A} \to \mathcal{B}^*$. Jei c yra kodas, tai jį pratęsime iki atvaizdžio $c^* : \mathcal{A}^* \to \mathcal{B}^*$ tokiu būdu:*

$$c^*(a_1a_2\cdots a_s) = c(a_1)c(a_2)\cdots c(a_s).$$

Abėcėlės $\mathcal{B}$ žodį $c^*(a_1a_2\cdots a_s)$ vadinsime žodžio $a_1a_2\cdots a_s$ kodu. *Abėcėlės $\mathcal{A}$ simbolių kodus $c(a)$, $a \in \mathcal{A}$, vadinsime* elementariais kodais.

Pavyzdys. Tegu $\mathcal{A} = \{A, B, C\}$, $\mathcal{B} = \{0, 1\}$. Tarkime, kodas c yra toks: $c(A) = 0$, $c(B) = 1$ ir $c(C) = 01$. Tada $c^*(BCAA) = 10100$. □

Dabar pateiksime įvairių kodų, kurie buvo arba yra naudojami, pavyzdžių.

Pavyzdžiai.

1. *Graikų ugnies kodas.* Klasikinėje Graikijoje naudotas karinėms žinioms perduoti. Abėcėlę $\mathcal{A}$ sudaro 24 graikiškos raidės, $\mathcal{B} = \{1, 2, 3, 4, 5\}$, čia 1 žymi vieną degantį fakelą ir t.t. Raidė α užkoduojama 11, β — 12, ir t.t. Žodis 'mn' buvo perduodamas uždegant po m ir n fakelų dviejose kalno viršūnės vietose.

2. *Cezario kodas.* Šį kodą pirmame amžiuje prieš Kristų naudojo Romos karvedys Julijus Cezaris slaptiems pranešimams šifruoti. Abi abėcėlės $\mathcal{A}$ ir $\mathcal{B}$ sudarytos iš tų pačių lotyniškų raidžių. Raidė a keičiama raide $c(a)$, kuri abėcėlėje stovi trimis pozicijomis toliau. Pavyzdžiui, raidė A keičiama raide D, raidė B — raide E, ir t.t.

3. *ASCII kodas.* Šis kodas (American Standard Code for Information Interchange) koduoja 128 simbolius (didžiąsias ir mažąsias raides, skaičius, skyrybos ženklus ir specialius simbolius) dvinarės abėcėles ilgio 7 žodžiais. ASCII kodo išplėstame variante ilgio 8 žodžiais koduojami 256 simboliai. Pavyzdžiui, raidės A ASCII kodas yra 1000001, raidės B — 0100001.

4. *Dešimtainių skaitmenų kodavimas.* Skaičius, užrašytus dešimtainėje sistemoje, dažnai tenka koduoti dvinarės abėcėlės žodžiais. Pateiksime keletą tokių kodų. Paprasčiausia tiesiog užrašyti skaitmenį dvejetainėje sistemoje, žr. 4.1 lentelės T stulpelį.

   Grėjaus (R.M. Gray) kodas pasižymi tuo, kad paeiliui einančių dešimtainių skaitmenų kodai skiriasi tik vienu simboliu (žr. G stulpelį).

   Kodas '2 iš 5' priskiria skaitmenims penkių simbolių dvinarius žodžius; lygiai du simboliai kiekviename kodo žodyje yra vienetukai (žr. '2 iš 5' stulpelį).

   Paskutiniams stulpelyje surašyti dešimtainių skaitmenų ASCII kodai.

skaitmuo	T	G	2 iš 5	ASCII
0	0000	0000	11000	0000110
1	0001	0001	00011	1000110
2	0010	0011	00101	0100110
3	0011	0010	00110	1100110
4	0100	0110	01001	0010110
5	0101	0111	01010	1010110
6	0110	0101	01100	0110110
7	0111	0100	10001	1110110
8	1000	1100	10010	0001110
9	1001	1101	10100	1001110

Lentelė 4.1: Dešimtainių skaitmenų kodavimas

Abėcėlės $\mathcal{A}$ žodžių aibėje $\mathcal{A}^*$ paprasčiausius tvarkos sąryšius apibrėšime pasinaudodami tuo, kad ilgesnius žodžius galima skaidyti į trumpesnius.

Apibrėžimas. *Žodį* $\mathbf{x} \in \mathcal{A}^*$ *vadinsime žodžio* $\mathbf{y} \in \mathcal{A}^*$ priešdėliu *(arba* prefiksu*), jei egzistuoja toks žodis* $\mathbf{z} \in \mathcal{A}^*$*, kad* $\mathbf{y} = \mathbf{xz}$*.*

Jei $\mathbf{x}$ *yra žodžio* $\mathbf{y}$ *priešdėlis, tai žymėsime* $\mathbf{x} \prec \mathbf{y}$*. Jei dviejų žodžių* $\mathbf{x}$ *ir* $\mathbf{y}$ m *ilgio priešdėlis yra tas pats, tai rašysime* $\mathbf{x} \sim_m \mathbf{y}$*.*

Pavyzdys. Žodis “aš” yra žodžio “ašara” priešdėlis. □

Tuščią žodį $\emptyset$, neturintį nei vieno simbolio, natūralu laikyti kiekvieno žodžio priešdėliu.

Teorema. *Sąryšis ’*$\prec$*’ yra tvarka aibėje* $\mathcal{A}^*$*, o ’*$\sim_m$*’ yra ekvivalentumo sąryšis.*

Užduotis. Įrodykite šią teoremą. Iš tikro, įrodymas yra labai paprastas, užtenka patikrinti tvarkos apibrėžimą.

Jei $c : \mathcal{A} \to \mathcal{B}^*$ yra kodas, tai c yra injektyvus atvaizdis. Tačiau iš to neišplaukia, kad jo tęsinys $c^* : \mathcal{A}^* \to \mathcal{B}^*$ taip pat injektyvus.

Apibrėžimas. *Kodą* $c : \mathcal{A} \to \mathcal{B}^*$ *vadiname* iššifruojamu*, jei jo tęsinys* $c^* : \mathcal{A}^* \to \mathcal{B}^*$ *yra injektyvus atvaizdis.*

Pavyzdys. Tegu $\mathcal{A} = \{A, B, C\}$, $\mathcal{B} = \{0, 1\}$. Tada kodas

$$c(A) = 00,\ c(B) = 10,\ c(C) = 11 \tag{4.1}$$

yra iššifruojamas, taip pat iššifruojamas ir kodas

$$c(A) = 0,\ c(B) = 01,\ c(C) = 0011, \tag{4.2}$$

gi kodas

$$c(A) = 0,\ c(B) = 01,\ c(C) = 010$$

nėra toks. □

Pastebėkime svarbų iššifruojamų kodų (4.1) ir (4.2) skirtumą. Jeigu mums perduodamas (4.1) kodu užkoduotas abėcėlės $\mathcal{A}$ žodis ir mes gaunamą seką skaitome simbolis po simbolio, tai siunčiamą užkoduotą simbolį galėsime atpažinti tuoj pat, kai tik perskaitysime paskutinį jo elementaraus kodo ženklą. Tačiau taip nebus, jei naudojamas (4.2) kodas. Pavyzdžiui, tik perskaitę visą siunčiamą '001' seką, galime nuspręsti, kad pirmąjį simbolį reikia iššifruoti 'A'. Kodą, kurio kiekvieną elementarų kodą galima atpažinti (dekoduoti) vos jį perskaičius, vadinsime *p-kodu*. Formalus jo apibrėžimas yra kiek kitoks, bet reiškia tą patį:

Apibrėžimas. *Kodą $c : \mathcal{A} \to \mathcal{B}^*$ vadinsime* p-kodu, *jei joks elementarus kodas $c(a)$, $a \in \mathcal{A}$, nėra kito elementaraus kodo $c(a')$, $a' \in \mathcal{A}$, $a' \neq a$, priešdėlis.*

Pavyzdys. (4.2) kodas yra iššifruojamas, tačiau nėra p-kodas. □

Teorema. *Bet kuris p-kodas yra iššifruojamas.*

Įrodymas. Tarkime priešingai: kodas $c : \mathcal{A} \to \mathcal{B}^*$ yra p-kodas, bet ne iššifruojamas. Taigi, kodo c tęsinys $c^* : \mathcal{A}^* \to \mathcal{B}^*$ nėra injekcija, t.y., egzistuoja du skirtingi $\mathcal{A}^*$ žodžiai $a_{i_1}a_{i_2}\cdots a_{i_k}$ ir $a_{j_1}a_{j_2}\cdots a_{j_l}$, kuriuos c^* atvaizduoja į tą patį žodį b, t.y.

$$c^*(a_{i_1}a_{i_2}\cdots a_{i_k}) = b_{i_1}b_{i_2}\cdots b_{i_k} = b = b_{j_1}b_{j_2}\cdots b_{j_l} = c^*(a_{j_1}a_{j_2}\cdots a_{j_l}),$$

čia b_i yra simbolio $a_i \in \mathcal{A}$ elementarus kodas, t.y. $b_i = c(a_i) \in \mathcal{B}^*$ $\forall i$. Tegu t yra mažiausias toks indeksas, kad $b_{i_t} \neq b_{j_t}$. Tada $b_{i_t}b_{i_{t+1}}\cdots b_{i_k} = b_{j_t}b_{j_{t+1}}\cdots b_{j_l}$, todėl egzistuoja toks žodis $b' \in \mathcal{B}^*$, kad $b_{i_t} = b_{j_t}b'$ arba $b_{i_t}b' = b_{j_t}$, o tai prieštarauja tam, kad kodas c yra p-kodas. □

Teorema. (Krafto-Makmilano) *Tegu abėcėlės $\mathcal{A} = \{a_1, a_2, \ldots, a_m\}$, $\mathcal{B} = \{b_1, b_2, \ldots, b_n\}$, ir tegu duoti m natūraliųjų skaičių $d_1, d_2, \ldots, d_m$. Toks iššifruojamas kodas $c : \mathcal{A} \to \mathcal{B}^*$, kad $|c(a_i)| = d_i$ kiekvienam i, egzistuoja tada ir tik tada, kai*

$$\sum_{i=1}^{m} \frac{1}{n^{d_i}} \leq 1. \tag{4.3}$$

Be to, bent vienas iš tokių kodų yra p-kodas.

Be įrodymo.

Kaip sudaryti tokį p-kodą iš teoremos ? Pailiustruosime sudarymą pavyzdžiu.

Pavyzdys. Tegu $\mathcal{A} = \{A, B, C, D, E, F, G\}$, $\mathcal{B} = \{0, 1, 2\}$, ir norime sudaryti p-kodą $c : \mathcal{A} \to \mathcal{B}^*$ tokį, kad $|c(A)| = 1$, $|c(B)| = 1$, $|c(C)| = 2$, $|c(D)| = 2$, $|c(E)| = 3$, $|c(F)| = 3$ ir $|c(G)| = 3$. Nesunku patikrinti, kad šie skaičiai tenkina (4.3) sąlygą:

$$\frac{1}{3} + \frac{1}{3} + \frac{1}{3^2} + \frac{1}{3^2} + \frac{1}{3^3} + \frac{1}{3^3} + \frac{1}{3^3} = 1,$$

todėl toks p-kodas egzistuoja. Jį konstruojame tokiu būdu:

- Iš pradžių parenkame ilgio 1 elementarius kodus, šiuo atveju $c(A)$ ir $c(B)$. Paprastumo dėlei, priskirkime jiems pirmus abėcėlės $\mathcal{B}$ elementus: $c(A) = 0$, $c(B) = 1$.

- Parinksime ilgio 2 elementarius kodus, šiuo atveju $c(C)$ ir $c(D)$. Jų priešdėliai negali būti jau panaudoti elementarūs kodai, t.y.0 ir 1, kitaip negausime p-kodo. Taigi, pirmas simbolis turi būti iš aibės $\mathcal{B} \setminus \{0, 1\}$, šiuo atveju tai gali būti tik2. Antru simboliu galime imti bet kurį abėcėlės $\mathcal{B}$ elementą. Vėlgi imkime pirmus abėcėlės $\mathcal{B}$ elementus, nors tai ir nėra būtina: $c(C) = 20$ ir $c(D) = 21$.

- Dabar ilgio 3 elementarūs kodai. Priešdėlių 0, 1, 20 ir 21 nebegalime naudoti, lieka tik priešdėlis 22. Trečiu simboliu vėlgi galime parinkti bet kurį abėcėlės $\mathcal{B}$ elementą. Gauname $c(E) = 220$, $c(F) = 221$, $c(G) = 222$.

- Gavome reikiamą kodą. Atkreipkite dėmesį, kad paskutiniame etape panaudojome visus abėcėlės $\mathcal{B}$ elementus. Bendru atveju, jei (4.3) sąlygoje nelygybė yra griežta, gali likti nepanaudotų simbolių.

4.2 Optimalus kodavimas

Praktikoje norima, kad užkoduotas pranešimas būtų kuo trumpesnis.

Tarkime, turime abėcėlę $\mathcal{A} = \{a_1, a_2, \ldots, a_m\}$ ir konkretų iššifruojamą kodą $c : \mathcal{A} \to \mathcal{B}^*$. Pažymėkime $b_i = c(a_i) \in \mathcal{B}^*$, $1 \le i \le m$. Tada bet kuris kodas, gautas sukeitus vietomis kodo c elementarius kodus $b_1, b_2, \ldots, b_m$, irgi bus iššifruojamas.

Pavyzdys. Tarkime, turime abėcėles $\mathcal{A} = \{A, B, C\}$, $\mathcal{B} = \{0, 1\}$, ir iššifruojamą kodą c, apibrėžtą tokiais elementariais kodais: $c(A) = 0$, $c(B) = 10$, $c(C) = 11$. Tada sukeitę elementarius kodus vietomis, irgi gausime iššifruojamą kodą c', pavyzdžiui, $c'(A) = 10$, $c'(B) = 11$, $c'(C) = 0$. □

Jei elementarių kodų ilgiai vienodi, tai sukeitus juos vietomis, užkoduoto pranešimo ilgis nepasikeis. Bet jei jie skirtingi, tai užkoduoto pranešimo ilgis priklauso nuo to, kiek kokių simbolių yra pranešime S, kurį reikia užkoduoti, ir kokie elementarūs kodai kokiems abėcėles $\mathcal{A}$ simboliams yra priskirti. Jei turime konkretų pranešimą ir konkretų kodą, tai nesunku taip sukeisti vietomis elementarius kodus, kad užkoduoto pranešimo ilgis būtų trumpiausias.

Tarkime, simbolis a_1 pranešime S pasirodo k_1 kartų, simbolis a_2 — k_2 kartus, ir t.t. Kiekvienam i elementaraus kodo $c(a_i)$ ilgį pažymėkime l_i. Tada, jei $k_i \le k_j$ ir $l_i \ge l_j$, tai $k_i l_i + k_j l_j \le k_i l_j + k_j l_i$. Iš tikrųjų, tegu $k_j = k_i + a$, $l_i = l_j + b$, $a, b \ge 0$. Tada

$$\begin{aligned}(k_i l_j + k_j l_i) - (k_i l_i + k_j l_j) &= (k_i l_j + (k_i + a)(l_j + b)) - (k_i(l_j + b) + (k_i + a) l_j) \\ &= (k_i l_j + k_i l_j + k_i b + a l_j + ab) - (k_i l_j + k_i b + k_i l_j + a l_j) \\ &= ab \ge 0.\end{aligned}$$

Todėl, kad gautume trumpiausią užkoduotą pranešimą, elementarius kodus abėcėlės $\mathcal{A}$ simboliams priskiriame tokiu būdu: išrikiuojame abėcėlės $\mathcal{A}$ simbolius taip, kad dažniau pranešime S pasirodantys simboliai stovėtų prieš rečiau pasirodančius simbolius, o elementarius kodus išrikiuojame jų ilgių didėjimo tvarka, ir priskiriame juos simboliams šia tvarka. Trumpiau tariant, dažniau pasitaikančius simbolius koduojame trumpesniais žodžiais, o rečiau — ilgesniais.

Pavyzdys. Imkime kodą iš pereito pavyzdžio ir pranešimą $S = ACBBCBBC$. Dažniausia raidė yra B, po to eina C ir A. Elementarūs kodai, išrikiuoti ilgių didėjimo tvarka, bus $0, 10, 11$. Taigi, elementarius kodus simboliams priskiriame taip: $c(B) = 0$, $c(C) = 10$, $c(A) = 11$. Užkodavę šiuo kodu žodį S, gausime žodį $c(S) = 111000100010$, kurio ilgis 12. Su šiais elementariais kodais mes negalėtume užkoduoti žodžio S trumpesniu kodu. □

Pastaba. Šis paprastas metodas leidžia minimizuoti kodo ilgį tik turint fiksuotą pranešimą S ir fiksuotą kodą c.

O ką daryti, kai reikia parinkti trumpiausią kodą, dar nežinant pranešimo, kurį reiks užkoduoti, ir dėl to negalime suskaičiuoti, kiek kokių simbolių jame yra? Dažnai žinome bent kiekvieno simbolio pasirodymo būsimuose pranešimuose dažnius, arba, kitaip sakant, tikimybes.

Apibrėžimas. Informacijos šaltiniu *vadinsime abėcėlę* $\mathcal{A} = \{a_1, a_2, \ldots, a_m\}$ *su skaičių* $p_1, p_2, \ldots, p_m$ *rinkiniu, tenkinančiu savybes:*

$$\sum_{i=1}^{m} p_i = 1, \quad 0 \leq p_i \leq 1 \; \forall i.$$

Šaltinis mums pateikia abėcėlės $\mathcal{A}$ simbolių sekas, o skaičiai $p_1, p_2, \ldots, p_m$ reiškia abėcėlės simbolių $a_1, a_2, \ldots, a_m$ pasirodymo tikimybes. Laikome, kad šaltinis sugeneruoja eilinį simbolį su šiomis tikimybėmis visiškai nepriklausomai nuo prieš tai buvusių ir paskui eisiančių simbolių (toks šaltinis vadinamas *be atminties*). Toks šaltinio modelis gerai vaizduoja atsitiktinių simbolių sekos generavimą, bet, pavyzdžiui, nelabai tinka žmonių kalbai modeliuoti, ir visiškai netinka vaizdams modeliuoti.

Pavyzdys. Šaltinis, kurio abėcėlė $\mathcal{A} = \{A, B, C\}$ su tikimybėmis $p_1 = 0.5$, $p_2 = 0.3$, $p_3 = 0.2$. Tai reiškia, kad šis šaltinis gali sugeneruoti tris simbolius: A, B, C su tokiomis tikimybėmis: tikimybė, kad pasirodys simbolis A, yra 0.5, kad simbolis B — 0.3, kad simbolis C — 0.2. □

Tarkime, turime informacijos šaltinį su abėcėle $\mathcal{A} = \{a_1, a_2, \ldots, a_m\}$ ir tikimybėmis $p_1, p_2, \ldots, p_m$, bei iššifruojamą kodą $c : \mathcal{A} \to \mathcal{B}^*$. Apibrėžkime vidutinį šio kodo žodžių ilgį $l(c)$ (duotam šaltiniui):

$$l(c) = \sum_{i=1}^{m} p_i \left| c(a_i) \right|.$$

Šį dydį galime interpretuoti kaip vidutinį abėcėlės $\mathcal{B}$ simbolių skaičių, kurio reikia vienam abėcėlės $\mathcal{A}$ simboliui užkoduoti.

Pavyzdys. Imkime šaltinį iš pereito pavyzdžio: abėcėlę $\mathcal{A} = \{A, B, C\}$ su tikimybėmis $p_1 = 0.5$, $p_2 = 0.3$, $p_3 = 0.2$. Imkime abėcėlę $\mathcal{B} = \{0, 1\}$ ir iššifruojamą kodą c, apibrėžtą taip: $c(A) = 11$, $c(B) = 10$, $c(C) = 0$. Tada $l(c) = 0.5 \cdot 2 + 0.3 \cdot 2 + 0.2 \cdot 1 = 1.8$. Tai reiškia, kad jei imtume kažkokį šio šaltinio sugeneruotą 10 simbolių ilgio pranešimą, tai jį užkodavus šiuo kodu, vidutinis užkoduoto pranešimo ilgis būtų 18 simbolių. Dabar imkime

iššifruojamą kodą c', apibrėžtą taip: $c'(A) = 0$, $c'(B) = 10$, $c'(C) = 110$. Tada $l(c') = 0.5 \cdot 1 + 0.3 \cdot 2 + 0.2 \cdot 3 = 1.7$. Taigi, duotam šaltiniui antrasis kodas geresnis, nors jo elementarūs kodai ilgesni už pirmojo. □

Kadangi bet kokį iššifruojamą kodą galime pakeisti p-kodu, kuris turi tiek pat elementarių kodų ir jų ilgiai tie patys (žr. Krafto-Makmilano teoremą), tai toliau apsiribosime pastaraisiais.

Apibrėžimas. *p-kodą $\bar{c} : \mathcal{A} \to \mathcal{B}^*$ vadinsime* optimaliu *(duotam šaltiniui), jei $l(\bar{c}) = \min_c l(c)$; čia minimumas imamas pagal visus p-kodus $c : \mathcal{A} \to \mathcal{B}^*$.*

Koks bebūtų informacijos šaltinis, optimalus kodas būtinai egzistuoja. Iš tikrųjų, tegu $p_* = \min p_i$, $s_i = |c(a_i)|$ $\forall i$, o $N \geq 1$ — bet koks skaičius. Nesunku įsitikinti, kad bet kokiam kodui c, kuriam $\max s_i \geq N/p_*$, galioja nelygybė $l(c) \geq N$. Iš tikro,

$$l(c) = \sum_{i=1}^{m} p_i s_i \geq p_* \sum_{i=1}^{m} s_i \geq p_* \max s_i \geq p_* \cdot N/p_* = N.$$

Tegu dabar $\hat{c}$ yra koks nors p-kodas, $\hat{l} = l(\hat{c})$. Tada optimalaus kodo reikia ieškoti kodų, tenkinančių sąlygą $\max s_i \leq \hat{l}/p_*$, aibėje. Kadangi ši aibė baigtinė, tai optimalus kodas tikrai egzistuoja.

Kaip sudaryti optimalų kodą? Iš pradžių parodysime, kaip gauti Šenono–Fano kodą (C.E. Shannon, R.M. Fano), kuris ne visada yra optimalus. Tegu $n = |\mathcal{B}|$. Kodo $c : \mathcal{A} \to \mathcal{B}^*$ žodžiams parinkime tokius ilgius $s_1, s_2, \ldots, s_m$, kad būtų

$$\frac{1}{n^{s_i}} \leq p_i < \frac{1}{n^{s_i - 1}}, \quad i = 1, \ldots, m.$$

p-kodas, kurio žodžių ilgiai tenkina šias nelygybes, vadinamas *Šenono–Fano kodu*. Jis visada egzistuoja, nes tokiu būdu parinkti ilgiai tenkina Krafto–Makmilano nelygybę:

$$\sum_{i=1}^{m} \frac{1}{n^{s_i}} \leq \sum_{i=1}^{m} p_i = 1.$$

Turėdami žodžių ilgius, patį p-kodą sudarome pagal pereito paragrafo pabaigoje aprašytą algoritmą.

Pavyzdys. Šaltinis tegu būna toks: $\mathcal{A} = \{A, B, C, D\}$ su tikimybėmis $p_1 = 0.4$, $p_2 = 0.3$, $p_3 = 0.2$, $p_4 = 0.1$. Abėcėlė $\mathcal{B} = \{0, 1\}$. Tada Šenono–Fano kodo žodžių ilgiai bus tokie: s_1 bus 2, nes $2^{-2} \leq 0.4 < 2^{-1}$, s_2 irgi bus 2, s_3 bus 3, nes $2^{-3} \leq 0.2 < 2^{-2}$, ir s_4 bus 4. Ir kodas galėtų būti toks: $c(A) = 00$, $c(B) = 01$, $c(C) = 101$, $c(D) = 1110$. Jo $l(c) = 2.4$. Šis kodas nėra optimalus. Netrukus išmoksime sudaryti optimalų kodą ir įsitikinsime, kad optimalus būtų, pavyzdžiui, toks kodas c': $c'(A) = 0$, $c'(B) = 10$, $c'(C) = 110$, $c'(D) = 111$, kurio $l(c') = 1.9$. □

Dabar panagrinėsime algoritmą, kuris duotam šaltiniui visada sudaro optimalų kodą. Tai *Hafmano* (D.A. Huffman) *algoritmas*. Jo pagalba gauti kodai irgi vadinami *Hafmano kodais* (r-nariais Hafmano kodais, kai norėsime pabrėžti, kiek simbolių naudojama koduojant).

Iš pradžių panagrinėsime kodavimo dvinarės abėcėlės žodžiais atvejį.

Turime informacijos šaltinį S su abėcėle $\mathcal{A} = \{a_1, a_2, \ldots, a_m\}$ ir tikimybėmis $p_1, p_2, \ldots, p_m$. Algoritmas Hafmano kodui c sudaryti būtų toks. Pernumeruokime abėcėlės $\mathcal{A}$ elementus taip, kad $p_1 \geq p_2 \geq \cdots \geq p_m$.

- Jei $m = 1$, tai $c(a_1) = 0$.
- Jei $m = 2$, tai $c(a_1) = 0$, $c(a_2) = 1$.
- Jei $m > 2$, tai mažiname abėcėlę $\mathcal{A}$ po vieną simbolį tokiu būdu: pažymėkime S' šaltinį su abėcėle $\mathcal{A}' = \{a_1, a_2, \ldots, a_{m-2}, b\}$ ir tikimybėmis $p_1, p_2, \ldots, p_{m-2}, p_{m-1} + p_m$, t.y. sujungiame du simbolius su mažiausiomis tikimybėmis į vieną, o jų tikimybes sudedame. Taip darome, kol gauname abėcėlę iš dviejų simbolių. Tada optimalus kodas yra akivaizdus: vieną simbolį reikia koduoti 0, kitą — 1. Grįžtame atgal, iš abėcėlės $\mathcal{A}'$ optimalaus kodo c' sudarydami abėcėlės $\mathcal{A}$ optimalų kodą c tokiu būdu:

$$\begin{aligned} c(a_i) &= c'(a_i) \ \forall i \leq m-2, \\ c(a_{m-1}) &= c'(b)0, \\ c(a_m) &= c'(b)1, \end{aligned}$$

  t.y. iš simbolio b kodo gauname kodus simboliams a_{m-1} ir a_m, tiesiog prirašydami atitinkamai 0 ir 1, o kitų simbolių a_i kodai lieka tokie patys.

Pavyzdys. Sudarykime Hafmano kodą šaltiniui iš pereito pavyzdžio. Šaltinio abėcėlė $\mathcal{A} = \{A, B, C, D\}$, ir tikimybės $p_1 = 0.4$, $p_2 = 0.3$, $p_3 = 0.2$, $p_4 = 0.1$. Abėcėlė $\mathcal{B} = \{0, 1\}$. Hafmano kodo sudarymą vaizduosime lentele. Kad būtų mažiau rašymo, pačių abėcėlės $\mathcal{A}$ simbolių nerašysime, o tik jų tikimybes. Stulpelyje $\mathcal{A}_k$ surašytos k-tajame žingsnyje gautos abėcėlės $\mathcal{A}_k$ simbolius atitinkančios tikimybės, žvaigždutės iš dešinės žymi, kurie simboliai kitu žingsniu bus jungiami į vieną, pabraukimas reiškia, kad tikimybė (simbolis) ankstesniu žingsniu yra gauta iš dviejų. Stulpeliai c_k bus naudojami grįžtant užrašyti abėcėlės $\mathcal{A}_k$ simbolių kodams. Taigi, vis sudėdami po dvi mažiausias tikimybes ir išrikiuodami gautas tikimybes mažėjimo tvarka, gauname tokią lentelę:

$\mathcal{A}_1$	c_1	$\mathcal{A}_2$	c_2	$\mathcal{A}_3$	c_3
0.4		0.4		$\underline{0.6}$	
0.3		0.3^*		0.4	
0.2^*		$\underline{0.3}^*$			
0.1^*					

Dabar grįždami sukonstruosime Hafmano kodus kiekvienai iš abėcėlių $\mathcal{A}_3$, $\mathcal{A}_2$ ir $\mathcal{A}_1$. Abėcėlėje $\mathcal{A}_3$ yra tik du simboliai, todėl vieną iš jų užkoduojame 0, kitą — 1. Abėcėlės $\mathcal{A}_2$ simbolių kodus gauname taip: iš to abėcėlės $\mathcal{A}_3$ simbolio, kuris buvo gautas sujungus du abėcėlės $\mathcal{A}_2$ simbolius, kodo gauname tų dviejų sujungtų simbolių kodus, prirašydami atitinkamai 0 ir 1.

Analogiškai gauname abėcėlės $\mathcal{A}_1 = \mathcal{A}$ simbolių kodus, t.y. gauname ieškomą Hafmano kodą. Lentelė pasidarys tokia:

$\mathcal{A}_1$	c_1	$\mathcal{A}_2$	c_2	$\mathcal{A}_3$	c_3
0.4	0	0.4	0	$\underline{0.6}$	1
0.3	10	0.3^*	10	0.4	0
0.2^*	110	$\underline{0.3}^*$	11		
0.1^*	111				

Matome, kad gavome tą kodą, kuris pateiktas pereitame pavyzdyje. □

Algoritmą nesunku pakeisti taip, kad jis tiktų Hafmano kodui n-narės abėcėlės žodžiais sudaryti. Mažinant abėcėlę, vienu simboliu reikia keisti ne porą, bet n simbolių su mažiausiomis tikimybėmis. Tiesa, gali atsitikti, kad po paskutinio sumažinimo gausime abėcėlę iš mažiau nei n simbolių. Siekdami, kad jų liktų lygiai n (nes tik tokiu atveju kodas bus optimalus), nustatykime, kiek simbolių reikia sujungti pirmu žingsniu.

Tegu $|\mathcal{A}| = m$, $|\mathcal{B}| = n$, ir abėcėlės mažinimo procese atlikome s žingsnių, pirmajame sujungdami $u \le n$ simbolių, o visuose kituose po n. Tada $m = (u-1)+(s-1)(n-1)+n = s(n-1)+u$, arba

$$u \equiv m \bmod (n-1), \quad 2 \le u \le n.$$

Ši sąlyga vienareikšmiškai apibrėžia pirmu žingsniu sujungiamų simbolių skaičių u. Primenu, kad $a \equiv b \bmod c$ reiškia, kad $a - b$ dalinasi iš c be liekanos, arba, kitaip pasakius, kad a padaliję iš c gausime tą pačią liekaną, kaip ir b padaliję iš c.

Pavyzdys. Sudarysime trinarį Hafmano kodą šaltiniui su tikimybėmis $0.4, 0.2, 0.2, 0.1, 0.05, 0.05$. Visų pirma apskaičiuokime, kiek pirmu žingniu reikės sujungti simbolių. Šiuo atveju $m = 6$, $n = 3$, todėl u apskaičiuosime iš sąlygų $u \equiv 6 \bmod 2$ ir $2 \le u \le 3$. Iš pirmos sąlygos gauname, kad u yra lyginis (nes u padaliję iš 2 turime gauti tą pačią liekaną, kaip ir 6 padaliję iš 2), todėl iš antros nusprendžiame, kad $u = 2$. Taigi, pirmu žingsniu sujungsime 2 simbolius, o kitais žingsniais — po 3. Lentelė bus tokia:

$\mathcal{A}_1$	c_1	$\mathcal{A}_2$	c_2	$\mathcal{A}_3$	c_3
0.4	1	0.4	1	$\underline{0.4}$	0
0.2	2	0.2	2	0.4	1
0.2	00	0.2^*	00	0.2	2
0.1	01	0.1^*	01		
0.05^*	020	$\underline{0.1}^*$	02		
0.05^*	021				

Teorema. *Hafmano kodai yra optimal̄us.*

Be įrodymo.

4.3 Duomenų spaudimas

Ankstesniuose paragrafuose tyrinėjome abėcėlinį kodavimą, kai kiekvienas simbolis yra koduojamas atskirai ir naudojamasi jų pasirodymo tikimybėmis. Matėme, kad taip darant, neįmanoma užkoduoti trumpesniu kodu, negu tai daro optimalus Hafmano kodas. Šiame paragrafe panagrinėsime, kokius neabėcėlinius kodus galima naudoti duomenų spaudimui, kad suspaustume duomenis labiau, nei leidžia optimalus abėcėlinis kodavimas.

Tarkime, turime kažkokį pranešimą, kuriuo nors visuotinai priimtu būdu užkoduotą (pavyzdžiui, tekstas paprastai koduojamas ASCII kodu) ir saugomą kompiuterio atmintyje. Dažnai tas užkodavimas nėra optimalus. Pavyzdžiui, ASCII kodas kiekvienam iš 256 simbolių priskiria 8 bitų ilgio žodį (bitas — tai simbolis iš abėcėlės $\mathcal{B} = \{0, 1\}$). Bet paprastai tekstuose jų naudojama žymiai mažiau, negu 256 (priklausomai nuo kalbos — apie 60–80, įskaitant didžiąsias ir mažąsias raides, skaitmenis, skyrybos ženklus). Be to, simbolių pasirodymo tekste tikimybės yra skirtingos, ir kiekvienai kalbai yra žinoma, kaip dažnai pasikartoja duotas simbolis šia kalba parašytame tekste. Taigi, galima pasirinkti mažesnę abėcėlę su jos simbolių pasirodymo dažniais, ir sudaryti optimalų abėcėlinį šia kalba parašytų tekstų kodavimą. Žinant simbolių pasirodymų dažnius, nesunku apskaičiuoti, kad dažniausiai sutinkamoms kalboms tokio kodo vidutinis žodžių ilgis $l(c)$ būtų šiek tiek mažesnis už 6, t.y. palyginus su ASCII kodu užkoduoto teksto ilgis sutrumpėtų 25% ar šiek tiek daugiau.

Apibrėžimas. *Kodavimas, leidžiantis gauti trumpesnį užkoduotą pranešimą nei pradinis pranešimas, vadinamas* duomenų spaudimu *arba* duomenų glaudinimu. *Spaudimo kokybė išreiškiama* spaudimo koeficientu, *kuris paprastai matuojamas procentais ir parodo, kiek procentų suspaustas pranešimas yra trumpesnis už pradinį.*

Praktikoje naudojamos suspaudimo programos (zip, arj ir kitos) tekstinius failus suspaudžia 70% ir daugiau. Tai reiškia, kad jos naudoja ne abėcėlinį kodavimą (abėcėlinis, kaip matėme, gali pasiekti tik šiek tiek daugiau nei 25%). Kokį kodavimą jos naudoja?

Idėja būtų tokia. Užkoduojamas ne kiekvienas simbolis atskirai, o simbolių seka. Pavyzdžiui, pranešimas skaidomas į žodžius (primenu, kad žodis — tai bet kokia simbolių seka, taigi, nebūtinai mums įprasta prasme, kai laikome, kad žodžius vieną nuo kito skiria tarpai ir skyrybos ženklai), kiekvienas žodis laikomas naujos abėcėlės simboliu ir užkoduojamas, naudojant kurį nors abėcėlinį kodą tai naujai abėcėlei. Gautas žodžių ir jų kodų rinkinys vadinamas *žodynu*. Tada užkoduotas pranešimas bus sudarytas iš žodyno ir iš pradinio pranešimo žodžių kodų sekos. Dekoduojant tiesiog pakeisime kodus atitinkamais žodžiais iš žodyno.

Pavyzdys. Tarkime, reikia koduoti lietuviškus tekstus. Dalinkime juos į žodžius pagal natūralias kalbos taisykles: žodžius vieną nuo kito skiria tarpai ir skyrybos ženklai. Galima daryti prielaidą, kad kiekviename konkrečiame tekste yra ne daugiau, kaip 2^{16} skirtingų žodžių (paprastai jų būna mažiau). Tokiu būdu, kiekvienam žodžiuigalima priskirti numerį — sveikąjį skaičių iš 2 baitų (baitas yra 8 bitų seka). Kadangi vidutiniškai lietuviški žodžiai yra sudaryti daugiau, nei iš dviejų raidžių (o kiekviena raidė užrašoma vienu baitu), tai vietoj kiekvieno žodžio užrašę jo numerį iš 2 baitų, tekstą neblogai suspaudžiame (apie 65% normaliems lietuviškiems tekstams, nes vidutinis lietuviškų žodžių ilgis yra apie 6 raides). Aišku, dar turime priskaičiuoti ir žodyno dydį, nes jį reikia perduoti kartu su užkoduotu tekstu, bet jei pradinis

tekstas buvo didelis (šimtai tūkstančių ar milijonai raidžių), tai prijungus žodyną, užkoduoto teksto ilgis padidėja palyginus nedaug. □

Praktikoje, kad nereikėtų kartu perduoti ir žodyno, naudojamas metodas, vadinamas *adaptyviuoju spaudimu*. Jo esmė tokia. Analizuojant pradinį tekstą, vienu metu dinamiškai sudarinėjamas žodynas ir koduojamas tekstas. Žodyno saugot nereikia, nes dekoduojant jis vėl dinamiškai sudaromas iš užkoduoto teksto.

Panagrinėkime šios idėjos paprasčiausią realizaciją, vadinamą *Lempelo–Zivo algoritmu*. Turime tekstą, kurį norime suspausti. Pradžioje žodyne yra tik tuščias žodis. Tekstas skaidomas į žodžius taip. Tarkime, iki i-tojo simbolio tekstas jau suskaidytas. Einamasis žodis prasidės $i+1$-uoju simboliu, ir baigsis j-uoju simboliu tokiu būdu, kad tai būtų ilgiausias žodyne esantis žodis plius vienas simbolis, t.y. žodis nuo $i+1$-ojo simbolio iki $j-1$-ojo dar yra žodyne, o žodžio nuo $i+1$-ojo simbolio iki j-ojo jau nėra.

Pavyzdys. Pavyzdžiui, jei užkoduojame tekstą, į kurį įeina žodžiai “diskrečioji matematika”, ir iki raidės ‘j’ jau suskaidėme tekstą į žodžius, tai einamasis žodis prasidės raide ‘j’. Jei žodyne, pavyzdžiui, jau buvo žodis “ji”, bet nebuvo žodžio “ji_” (čia ‘_’ žymi tarpo simbolį), tai einamasis žodis bus “ji_”. Jei jau buvo žodis “ji mate”, bet nebuvo žodžio “ji matem”, tai einamasis žodis bus “ji matem”. □

Prie užkoduoto teksto prijungiame žodyne rasto žodžio numerį ir tą papildomą simbolį, o einamąjį žodį dedame į žodyną. Ir kartojame šią procedūrą, kol užkoduojame visą tekstą. Taigi, užkoduotas tekstas bus seka porų (p, q), kur p yra žodžio numeris žodyne, o q — simbolis. O sudarytą žodyną galime išmesti, kad jis neužimtų vietos, nes mes galėsime jį atstatyti dekodavimo metu.

Dekoduojame taip. Pradžioje žodyne yra tik tuščias žodis. Imame einamąją porą (p, q) iš užkoduoto teksto. Imame p-tąjį žodyno žodį, prie jo prijungiame simbolį q ir gauname einamąjį žodį. Jį prijungiame prie dekoduojamo teksto ir įdedame į žodyną. Kartojame šią procedūrą, kol dekoduojame visą tekstą.

Pavyzdys. Šis algoritmas tinka ilgiems tekstams, nes juose raidžių kombinacijos pradeda kartotis. Kadangi čia galime duoti tik trumpą pavyzdį, imkime tokį tekstą, kuriame kai kurios raidžių kombinacijos kartojasi, pavyzdžiui, $S = oi\_oi\_ojojoi\_oi\_ojoi\_ojoi$, čia ženklu _ žymime tarpo simbolį. Užkoduokime šį pranešimą. Pradžioje žodyne D yra tik tuščias žodis, ir jo numeris yra 0.

- Išskirkime pirmą žodį duotame tekste. Jis prasidės pirma raide o. Žiūrime, ar žodis o yra žodyne. Nėra. Taigi, ilgiausias žodyne esantis tinkamas žodis yra tuščias, todėl prie užkoduoto teksto C (kuris irgi pradžioje yra tuščias) prijungiame porą $(0, o)$, o į žodyną pirmu numeriu įdedame žodį o.

- Išskirkime antrą žodį. Jis prasidės antra raide: i, todėl analogiškai prie C prijungiame $(0, i)$, o į D antru numeriu įdedame i. Lygiai taip pat paskui prie C prijungiame $(0, \_)$, o į D trečiu numeriu įdedame _.

- Ketvirtas žodis vėl prasidės raide o. Ir ši raidė jau yra žodyne. Todėl einamasis žodis bus oi, kurio žodyne dar nėra: prie C prijungiame $(1, j)$, kur 1 yra žodžio o numeris žodyne, o j yra papildomas simbolis, o į D ketvirtu numeriu įdedame oi.

- Penktas žodis prasidės raide _, kuri yra žodyne, o $\_o$ jau nėra žodyne, todėl tai ir bus einamasis žodis: prie C prijungiame $(3, o)$, kur 3 yra žodžio _ numeris žodyne, o o yra papildomas simbolis, o į D penktu numeriu įdedame $\_o$.

- Taip tęsdami, gauname tokius žodyną D ir užkoduotą pranešimą C:

D	
1	o
2	i
3	_
4	oi
5	$\_o$
6	j
7	oj
8	$oi\_$
9	oi_o
10	jo
11	$i\_$
12	ojo

C	
0	o
0	i
0	_
1	i
3	o
0	j
1	j
4	_
8	o
6	o
2	_
7	o
2	

  Paskutinėje C poroje nėra simbolio, nes baigėsi tekstas.

Matome, kad šiame pavyzdyje užkoduotas pranešimas C nėra trumpesnis už pradinį pranešimą S. Taip yra dėl to, kad pavyzdys labai trumpas. Jei jis būtų ilgesnis, žodžiai žodyne gautųsi ilgesni, ir pakeitę juos jų numeriais labai sutrumpintume pranešimą.

Dabar žodyną D galima išmesti, kad neužimtų vietos, ir laikyti tik užkoduotą pranešimą C. Jį dekoduosime taip. Žodynas D vėl tuščias.

- Imame pirmą porą $(0, o)$ iš C. Einamąjį žodį gauname taip: 0-inį žodyno žodį (t.y. tuščią žodį) jungiame su raide o, gauname žodį o, kurį ir prijungiame prie konstruojamo pranešimo S bei įdedame į žodyną pirmu numeriu. Lygiai tą patį padarome su $(0, i)$ ir $(0, \_)$. Po šių operacijų $S = oi\_$ ir D turi 3 žodžius: o, i ir _.

- Tada imame porą $(1, i)$, taigi, pirmą žodymo žodį (t.y. o) jungiame su raide i, gauname žodį oi, kurį ir prijungiame prie konstruojamo pranešimo S bei įdedame į žodyną ketvirtu numeriu.

- Taip tęsdami, gauname tą patį žodyną D, kaip ir užkoduodami, ir tuo pačiu dekoduojame užkoduotą pranešimą D.

Pastabos. 1. Praktikoje žodyno augimą reikia apriboti. Paprastai riba yra 2^{16} žodžių.

2. Praktikoje naudojami įvairūs šio algoritmo pagerinimai, pavyzdžiui, galima pradžioje imti ne tuščią žodyną, o jau su įdėtais ilgio 1 žodžiais.

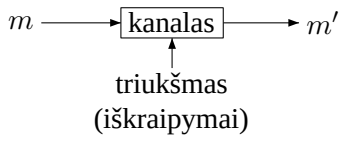
Pabaigai dar kelios pastabos apie duomenų spaudimą. Iki šiol šnekėjome tik apie *spaudimą be informacijos praradimo*, t.y. pradinis pranešimas gali būti pilnai atgamintas iš užkoduoto pranešimo. Bet norint suspausti skaitmeninius vaizdus, garsą ar video signalą, gali būti prasminga kalbėti apie *spaudimą, prarandantį informaciją*, nes ši informacija paprastai skirta žmogui, ir galbūt ne visos pradinio pranešimo detalės yra būtinos ar net apskritai pajuntamos.

Vaizdams ir garsui suspausti be informacijos praradimo taikomos spėjamosios (prediktyvinės) schemos. Užkodavimo idėja tokia: remiantis iki šiol gautais duomenimis, numatyti kitos duomenų porcijos, pavyzdžiui, vaizdo taško ar garso imties, reikšmę, ir užkoduoti skirtumą tarp to spėjimo ir tikros reikšmės. Skirtumai paprastai būna maži, todėl čia gerai veikia Hafmano kodavimas, mažas reikšmes užkoduojantis trumpais žodžiais. Dekoduojant daroma taip: norint gauti tikrą reikšmę, lygiai taip pat numatoma kita reikšmė, ir pridedamas dekoduotas skirtumas. Pavyzdžiui, pats paprasčiausias numatymo būdas — tarti, kad kita reikšmė bus lygiai tokia pati, kaip ir prieš tai buvusi. Gana gerai tinka spausti paprastai kompiuterinei grafikai ar juodai–baltam faksui. O neprarandantis informacijos JPEG vaizdų spaudimas numatymui naudoja aplinkinių taškų tiesines funkcijas, pavyzdžiui, viena iš naudojamų tiesinių funkcijų yra tokia: numatoma, kad spalvos reikšmė $I_{i,j}$ vaizdo taške (i, j) bus lygi $I_{i-1,j} + I_{i,j-1} - I_{i-1,j-1}$. Panašios schemos gerai veikia ir garsui, ir video signalui.

Bet vaizdui ir garsui suspausti dažnai užtenka kodavimo, prarandančio informaciją. Iš tikro, iki šiol mes darėme prielaidą, kad pradinis pranešimas yra skaitmeninio pavidalo ir turi būti perduotas be informacijos praradimo. Vaizdui ir garsui ši prielaida ne visada tinka, nes pradinis pranešimas gali būti analoginis, arba kad ir skaitmeninis, bet su didesne rezoliucija, negu mums reikia, pavyzdžiui, 16 bitų garsas kalbai įrašyti. Tokiais atvejais pirmas spaudimo žingsnis būtų sumažinti reikšmių, kurias gali įgyti kiekvienas pradinio pranešimo taškas, skaičių — tai vadinama *kvantavimu*. Pavyzdžiui, sumažinti 24 bitų spalvas iki 8 bitų, t.y. pasirinkti 256 spalvas iš 2^{24}. Praktikoje taikomi metodai suskaido pradinį pranešimą į dalis, pavyzdžiui, garsas gali būti skaidomas į dažnių juostas, vaizdas — į ryškumą ir spalvą. Tada kiekviena dalis kvantuojama atskirai, atsižvelgiant į įvairius faktorius, pavyzdžiui, į žmogaus galimybes pajausti tą dalį. Tai leidžia geriau suspausti, pasinaudojant, pavyzdžiui, tuo, kad žmonių kalbai įrašyti dažniai, aukštesni nei 7 KHz, yra nebūtini, arba kad žmogaus akis nepastebi mažų spalvos pasikeitimų, o tik ryškumo, ir pan. JPEG ir MPEG naudoja tokius spaudimo būdus vaizdui ir garsui.

4.4 Klaidas taisantys kodai

Panagrinėkime tokią schemą. Mes turime kažkokį pranešimą m ir norime jį kam nors perduoti. Perduodant pranešimą, galimas jo iškraipymas. Tai galima pavaizduoti grafiškai šitaip:

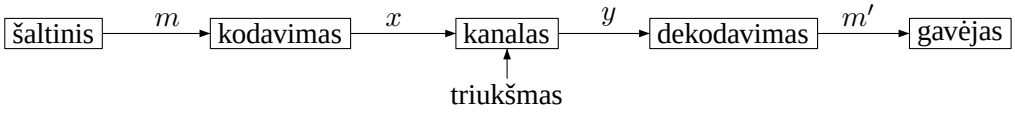


Pranešimas m perduodamas ryšio kanalu, kuriame jį gali iškraipyti triukšmas. Iš kanalo išeina pranešimas m', kuris gali skirtis nuo m ($m' \neq m$).

Pavyzdys. Kanalų pavyzdžiai:

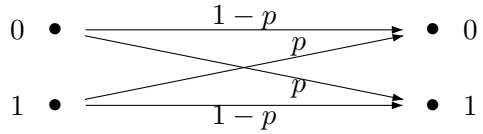
- Telefono linija. Informacija gali būti iškraipyta dėl triukšmo.
- Kosminis zondas siunčia Marso nuotraukas į Žemę.
- Ląstelės dalijimasis, kurio metu motininės ląstelės DNR perduoda informaciją dukterinės ląstelės DNR (perduodama informacija gali būti iškraipyta dėl radiacijos, ir tada įvyksta mutacija).
- Kietas diskas: informacija į jį užrašoma, o po kurio laiko nuskaitoma. Per tą laiką jinai galėjo būti iškraipyta (dėl šiluminio triukšmo, radiacijos ir pan.)

Norime, kad iš kanalo gautas pranešimas m' būtų lygus pradiniam pranešimui m su kuo didesne tikimybe. Kaip tai padaryti ? Vienas kelias būtų gerinti kanalo charakteristikas, bet jis reikalauja daug lėšų. Klaidas taisantys kodai yra kitas sprendimas: priimame kanalą tokį, koks jis yra, bet perduodami juo informaciją, naudojame tam tikrus metodus, padedančius aptikti ir ištaisyti kanalo padarytas klaidas. Tai yra, pranešimas m prieš siunčiant į kanalą yra užkoduojamas, o gavus užkoduotą pranešimą iš kanalo, jis yra dekoduojamas. Schema būtų tokia:



Laikysime, kad m yra dvinaris (binarinis) vektorius $(m_1, m_2, \ldots, m_k)$, t.y. dvinarės abėcėlės $\mathcal{A} = \{0, 1\}$ vektorius. Paprastai mes jį rašysime kaip ilgio k žodį $m_1 m_2 \cdots m_k$. Prieš siųsdami į kanalą, jį užkoduojame dvinariu ilgio n žodžiu x, pridėdami papildomos informacijos, kuri leis aptikti ir ištaisyti klaidas. Taigi, žodis x paprastai būna ilgesnis negu m, t.y. $n \geq k$. Išejęs iš kanalo galbut iškraipytas užkoduotas žodis y yra dekoduojamas, ir randamas žodis m'. Naudojant gerus kodavimo būdus, tikimybė, kad $m' \neq m$, labai sumažėja, bet užtat išauga simbolių kiekis, kurį reikia persiųsti kanalu.

Kanalą sumodeliuoti matematiškai galima įvairiais būdais. Mes naudosime vieną paprasčiausių modelių, vadinamą *dvinariu simetriniu kanalu* su klaidos tikimybe p. Laikysime, kad į kanalą siunčiami simboliai iš abėcėlės $\mathcal{A} = \{0, 1\}$, iš kanalo išeina irgi tos pačios abėcėlės simboliai, o kiekvieno simbolio iškraipymo tikimybė yra p, $0 \leq p < 0.5$. Grafiškai šį kanalo modelį galima pavaizduoti taip:



Matome, kad į kanalą gali įeiti simboliai 0 ir 1, išeina irgi 0 ir 1, simbolis 0 išlieka 0 su tikimybe $1-p$ ir keičiasi į 1 su tikimybe p, analogiškai ir simbolis 1.

Apibrėžimas. (Klaidas taisančiu) kodavimu *vadinsime injektyvų atvaizdį* $c : \mathcal{A}^k \to \mathcal{A}^n$. *Aibė* $C = c(\mathcal{A}^k) = \{c(m) : m \in \mathcal{A}^k\} \subseteq \mathcal{A}^n$ *vadinama* (klaidas taisančiu) kodu. *Kodo* C *vektoriai vadinami* kodo žodžiais. *Santykis* $R = k/n$ *vadinamas* kodo koeficientu.

Kodo koeficientas parodo, kuri į kanalą pasiųstų simbolių dalis yra naudinga informacija, o kuri tik pridėta klaidų aptikimui ir ištaisymui. Kuo jis didesnis, tuo geriau, nes tuo didesnė informacijos dalis yra persiunčiama.

Pavyzdys. (Pakartojimo kodas) Tegu $k = 1$, $n = 3$. Taigi, pranešimą dalijame į ilgio 1 žodžius, ir kiekvieną tokį žodį užkoduojame ilgio 3 žodžiu taip: $c(0) = 000$, $c(1) = 111$. Tada kodas

$$C = \{000, 111\} \subset \{0,1\}^3 = \{000, 001, 010, 011, 100, 101, 110, 111\},$$

ir kodo koeficientas $R = 1/3$. □

Taigi, visą informaciją, kurią norime išsiusti, skaidome į k ilgio dvinarius vektorius (blokus) m, kiekvieną bloką užkoduojame n ilgio kodo žodžiu x (t.y. dvinariu vektoriumi iš kodo C) ir siunčiame į kanalą. Iš kanalo gauname iškraipytą vektorių y, kuris gali nebepriklausyti kodui C, t.y. y yra bet kuris vektorius iš aibės $\mathcal{A}^n$. Dekoduojame, vektoriui y priskirdami vektorių $m' \in \mathcal{A}^k$. Dekodavimas paprastai vykdomas dviem etapais: pirmiausiai vektoriui y priskiriame kodo žodį $x' \in C$, o tada pasinaudojame kodavimo c atvirkštine funkcija $c^{-1} : C \to \mathcal{A}^k$, kad žodžiui $x' \in C$ priskirtume $m' \in \mathcal{A}^k$. Pirmame etape naudojama funkcija $f : \mathcal{A}^n \to C$ vadinama *dekodavimo taisykle*.

Pavyzdys. Imkime tą patį pakartojimo kodą. Dekodavimas galėtų būti toks: kokių simbolių gautame žodyje daugiau, tokiu simboliu ir dekoduojame, pavyzdžiui, jei $y = 101$, tai jį dekoduojame 1, nes vektoriuje y yra du vienetai ir tik vienas nulis. Kitaip sakant, jei apibrėžtume atstumą tarp dviejų vektorių kaip skaičių koordinačių, kuriose jie skiriasi, tai dekoduotume imdami artimiausią (matuojant tuo atstumu) kodo žodį, ir tada imdami pranešimą, atitinkantį tą kodo žodį. Pvz, atstumas tarp 101 ir 000 yra du, tarp 101 ir 111 yra vienas, tai pasirenkame 111, nes jis arčiau 101, nei 000, ir dekoduojame 1, nes jis atitinka 111. □

Pavyzdyje įvestą atstumą ir dekodavimo procedūrą galime aprašyti formaliai.

Apibrėžimas. Hemingo *(R.W. Hamming)* atstumu *(arba tiesiog* atstumu*) tarp dviejų vektorių* $x = (x_1, x_2, \ldots, x_n) \in \mathcal{A}^n$ *ir* $y = (y_1, y_2, \ldots, y_n) \in \mathcal{A}^n$*, žymimu* $d(x, y)$*, vadinsime koordinačių, kuriose jie skiriasi, skaičių:*

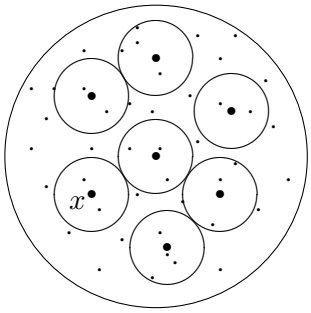
$$d(x, y) = |\{i : 1 \leq i \leq n,\ x_i \neq y_i\}|.$$

Apibrėžimas. Minimalaus atstumo *dekodavimo taisykle vadinsime tokią dekodavimo taisyklę* $f : \mathcal{A}^n \to C$*, kuri kiekvienam* $y \in \mathcal{A}^n$ *priskiria arčiausiai (matuojant Hemingo atstumu) esantį kodo* C *žodį* x'*, t.y. priskiria tokį* $x' \in C$*, kad*

$$d(y, x') = \min_{z \in C} d(y, z).$$

Šios taisyklės naudojimas grindžiamas taip. Kadangi klaidos tikimybė kanale yra mažesnė nei $1/2$, tai klaida yra mažiau tikėtina, nei klaidos nebuvimas. Lygiai taip pat dvi klaidos yra mažiau tikėtinos, nei viena, ir t.t. Taigi, didžiausia tikimybė, kad į kanalą buvo išsiųstas kodo žodis, mažiausiai tesiskiriantis nuo iš kanalo gauto žodžio, nes tokiu atveju buvo padaryta mažiausiai klaidų. Todėl ir dekoduojame kodo žodžiu, mažiausiai tesiskiriančiu nuo iš kanalo gauto žodžio.

Dekodavimą naudojant minimalaus atstumo dekodavimo taisyklę grafiškai galima pavaizduoti taip:



Čia didysis skritulys su taškais yra aibė visų vektorių iš $\mathcal{A}^n$, pajuodinti taškai priklauso kodui $C \subseteq \mathcal{A}^n$. Iš kanalo gautas vektorius y gali būti bet kuris taškas. Paži̇urime, kuris kodo žodis (pajuodintas taškas) yra arčiausiai, juo ir dekoduojame. Pavyzdžiui, pasirinkime kurį nors kodo žodį x. Tarkime, artimiausias kitas kodo C žodis yra atstumu h nuo x. Tai reiškia, kad jei y yra nutolęs nuo x mažesniu atstumu, nei $h/2$, tai jis tikrai bus dekoduotas žodžiu x, nes šis kodo C žodis yra arčiausiai. Taigi, apie x galime apibrėžti spindulio $h/2$ skritulį, ir visi jam priklausantys taškai bus dekoduojami to skritulio centru — kodo žodžiu x. Pažymėkime d patį mažiausią atstumą tarp bet kurių skirtingų kodo C žodžių, t.y.

$$d = \min_{x,z \in C,\, x \neq z} d(x, z).$$

Skaičius d vadinamas kodo C *minimaliu atstumu*. Tai jei apie kiekvieną kodo žodį apibrėšime spindulio $d/2$ skritulį, skritulyje esantys taškai bus dekoduojami skritulio centru.

Tarkime, į kanalą pasiuntėme žodį x, ir kanale jame buvo padaryta t klaidų. Iš kanalo išėjo vektorius y, esantis atstumu t nuo x. Jei $t < d/2$, tai y priklausys žodžio x skrituliui, todėl dekuoduosime žodžiu x, t.y. ištaisysime kanalo padarytas klaidas. Jei $t > d/2$, tai y gali atsidurti jau kito kodo žodžio skritulyje, ir tokiu atveju bus dekoduotas neteisingai.

Apibrėžimas. *Jei, naudojant minimalaus atstumo dekodavimo taisyklę, dekoduojama visada teisingai, kai siųstame žodyje yra ne daugiau kaip t klaidų, tai kodą C vadiname* t klaidų taisančiu kodu.

t *klaidų taisantį kodą vadiname* tiksliai t klaidų taisančiu kodu, *jei jis nėra* $t + 1$ *klaidą taisantis kodas.*

Taigi, kaip matėme, kodas C yra t klaidų taisantis kodas, jei $t < d/2$. Dėl to jis yra tiksliai t klaidų taisantis kodas, jei t yra didžiausias sveikas skaičius, mažesnis už $d/2$. Nesunku įsitikinti, kad skaičiaus t išraiška yra tokia: $t = [(d-1)/2]$, čia $[a]$ yra skaičiaus a sveikoji dalis, t.y. didžiausias sveikas skaičius $\leq a$. Taigi, įrodėme tokią teoremą:

Teorema. *Kodas C yra tiksliai $[(d-1)/2]$ klaidų taisantis kodas.*

Todėl stengiamasi sudaryti tokius kodus, kurių minimalus atstumas d būtų kuo didesnis, kad kodas ištaisytų kuo daugiau klaidų.

Kartais svarbu ne tik tai, kiek klaidų kodas ištaiso, bet ir kiek jų aptinka. Kadangi į kanalą siunčiame tik kodo žodžius, tai, jei iš kanalo išeina ne kodo žodis, reiškia, buvo padaryta klaidų. Todėl t klaidų aptinkantis kodas gali būti apibrėžiamas taip:

Apibrėžimas. *Kodą C vadiname t* klaidų aptinkančiu kodu, *jei bet kuriame kodo žodyje įvykus ne daugiau kaip t klaidų, gautas vektorius nepriklauso kodui C.*

t klaidų aptinkantį kodą vadiname tiksliai t klaidų aptinkančiu kodu, *jei jis nėra $t+1$ klaidą aptinkantis kodas.*

Jei įvyko d iškraipymų, tai gali būti, kad gavome kitą kodo žodį, todėl kodas C yra t klaidų aptinkantis kodas, jei $t < d$. Gauname tokį rezultatą:

Teorema. *Kodas C yra tiksliai $d-1$ klaidą aptinkantis kodas.*

Panagrinėkime kelis paprastų kodų pavyzdžius.

Pavyzdžiai.

1. *Pakartojimo n kartų kodas.* Tai prieš tai buvusio pavyzdžio apibendrinimas. Čia $k = 1$, o n — kažkoks fiksuotas natūralusis skaičius. Ilgio 1 žodžius užkoduojame ilgio n žodžiais: $c(0) = 00\cdots0$, $c(1) = 11\cdots1$. Kodas $C = \{00\cdots0, 11\cdots1\} \subseteq \{0,1\}^n$. Kodo koeficientas $R = 1/n$ — labai žemas, užtat kodas, kaip matysime, ištaiso ir aptinka daug klaidų. Labai paprasta rasti šio kodo minimalų atstumą — iškart matome, kad $d = n$. Taigi, tai tiksliai $[(n-1)/2]$ klaidų taisantis ir tiksliai $n-1$ klaidą aptinkantis kodas. Dekodavimas vyksta taip: gavę iš kanalo vektorių y, suskaičiuojame, ko daugiau jame yra — nulių ar vienetų, tuo ir dekoduojame. Jei n lyginis, gali gautis taip, kad vienetų ir nulių bus po lygiai. Tokiam atvejui turime įsivesti kažkokią atskirą taisyklę, pavyzdžiui, atsitiktinai pasirinkti 0 ar 1, arba visada dekoduoti 0, ir pan.

2. *Kontrolinio simbolio kodas.* Kartais svarbu ne ištaisyti klaidas, o labai nesunkiai jas aptikti. Šis kodas tam ir skirtas. Pranešimą $m = (m_1, m_2, \ldots, m_k)$ užkoduojame, prijungdami vieną papildomą simbolį x_{k+1}, vadinamą *kontroliniu simboliu*, t.y. užkoduotas pranešimas $x = (m_1, m_2, \ldots, m_k, x_{k+1})$. Taigi, čia k yra kažkoks fiksuotas natūralusis skaičius, o $n = k+1$. Kontrolinis simbolis apskaičiuojamas taip, kad užkoduotame vektoriuje gautųsi lyginis vienetų skaičius. Pavyzdžiui, jei $m = 1001$, tai $x = 10010$, o jei $m = 1011$, tai $x = 10111$. Kodas C yra sudarytas iš visų ilgio n dvinarių vektorių, turinčių lyginį vienetų skaičių. Kodo koeficientas $R = k/(k+1)$ yra labai aukštas, artimas vienetui, užtat kodas, kaip matysime, klaidų visai netaiso, ir aptinka tik vieną klaidą. Minimalus atstumas irgi nesunkiai randamas — tai $d = 2$. Iš tikro, atstumas tarp kodo žodžių $000\cdots0$ ir $110\cdots0$ yra 2, ir jokie du kodo žodžiai nėra nutolę atstumu 1 vienas nuo kito (jei taip būtų, viename iš jų vienetų skaičius būtų nelyginis). Todėl šis kodas taiso $[(d-1)/2] = 0$ klaidų, ir aptinka $d-1 = 1$ klaidą. Tai yra, jei padaryta viena klaida, tai kodas būtinai ją aptiks, o jei daugiau, tai gali ir nebeaptikti. Bet gali ir aptikti. Pavyzdžiui, šis kodas aptiks, kad padaryta klaidų, jei jų skaičius nelyginis.

Dekoduodami tiesiog patikriname, ar vienetų skaičius gautame vektoriuje lyginis. Jei taip, tai dekoduotas vektorius bus gautas, atmetus paskutinę koordinatę. Jei ne, reiškia, buvo klaidų. Teks paprašyti persiųsti iš naujo.

3. *Knygų numeracijos sistema ISBN.* Tai praktikoje taikomo kontrolinio simbolio kodo pavyzdys. Kiekviena šiais laikais išleidžiama knyga turi ISBN (International Standard Book Number) numerį, sudarytą iš dešimties dešimtainių skaitmenų. Pavyzdžiui, V.Stakėno knygutės "Informacijos kodavimas" ISBN numeris yra 9986-19-183-1. Pirmi devyni skaitmenys $a_1, a_2, \ldots, a_9$ rodo šalį, kurioje išleista knyga, leidyklą ir knygos numerį, o dešimtas a_{10} yra kontrolinis, apskaičiuojamas pagal tokią formulę:

$$a_{10} \equiv \sum_{i=1}^{9} i a_i \bmod 11.$$

(t.y. apskaičiuojame sumą, daliname ją iš 11 ir imame liekaną). Dalinant iš 11, liekana gali gautis 10. Tokiu atveju kontrolinis simbolis a_{10} žymimas ženklu X. Taigi, $k = 9$, $n = 10$, $R = 9/10$. Šis kodas klaidų netaiso, bet aptinka pavienes klaidas ir dviejų greta stovinčių simbolių sukeitimą vietomis (tai dažniausiai pasitaikančios klaidos rašant knygos ISBN numerį). Dekodavimas yra lygiai toks pat, kaip ir kodavimas: apskaičiuojame tą pačią liekaną ir palyginame su a_{10}. Jei nelygu, reiškia, įvyko klaida, reikia prašyti atsiųsti ISBN numerį iš naujo.

4. *Lietuvos piliečių asmens kodas.* Kiekvienas Lietuvos pilietis turi savo asmens kodą, kuriame irgi naudojamas kontrolinis simbolis. Asmens kodo struktūra:

$$LY_1Y_2M_1M_2D_1D_2X_1X_2X_3K,$$

kur

L — lytis. Lietuviai turi 6 lytis:

1 — vyras, gimęs XIX amžiuje,
2 — moteris, gimusi XIX amžiuje,
3 — vyras, gimęs XX amžiuje,
4 — moteris, gimusi XX amžiuje,
5 — vyras, gimęs XXI amžiuje,
6 — moteris, gimusi XXI amžiuje.

$Y_1Y_2M_1M_2D_1D_2$ — gimimo data:

Y_1Y_2 — metai šimtmetyje,
M_1M_2 — mėnuo,
D_1D_2 — diena.

$X_1X_2X_3$ — eilės numeris tarp tą dieną gimusiųjų, priskiriamas Gyventojų registro.

K — kontrolinis skaičius, apskaičiuojamas dauginant kiekvieną asmens kodo skaitmenį iš svorio koeficiento ir sumuojant:

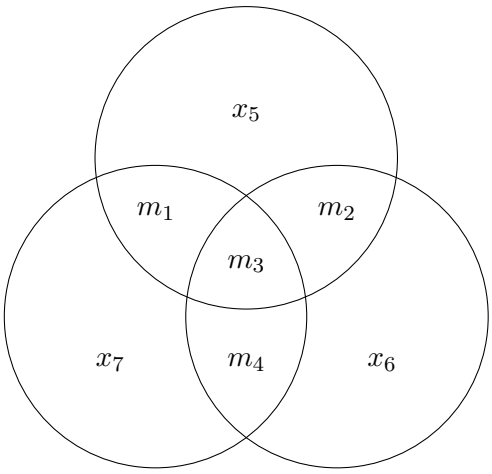
$$S = L \cdot 1 + Y_1 \cdot 2 + Y_2 \cdot 3 + M_1 \cdot 4 + M_2 \cdot 5 + D_1 \cdot 6 + D_2 \cdot 7 + X_1 \cdot 8 + X_2 \cdot 9 + X_3 \cdot 1.$$

Suma S dalinama iš 11, ir jei liekana nelygi 10, ji yra asmens kodo kontrolinis skaičius.
Jei liekana lygi 10, tuomet skaičiuojama nauja suma su tokiais svorio koeficientais:

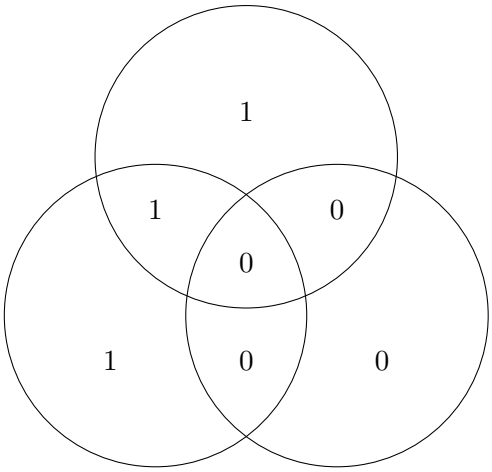
$$S = L \cdot 3 + Y_1 \cdot 4 + Y_2 \cdot 5 + M_1 \cdot 6 + M_2 \cdot 7 + D_1 \cdot 8 + D_2 \cdot 9 + X_1 \cdot 1 + X_2 \cdot 2 + X_3 \cdot 3.$$

Dar kartą daliname iš 11, ir jei liekana nelygi 10, ji yra asmens kodo kontrolinis skaičius. Jei vėl liekana yra 10, kontrolinis skaičius imamas lygiu 0.

5. *Hemingo kodas.* Yra ištisa šeima dvinarių Hemingo kodų. Mes nagrinėsime tik vieną iš jų, kurio parametrai yra tokie: $k = 4$, $n = 7$, $R = 4/7$. Pranešimą $m = (m_1, m_2, m_3, m_4)$ užkoduojame žodžiu $x = (m_1, m_2, m_3, m_4, x_5, x_6, x_7)$. Užkodavimą galima pavaizduoti grafiškai taip. Surašome šiuos septynis simbolius į susikertančius skritulius:



Simboliai x_5, x_6, x_7 parenkami taip, kad kiekviename skritulyje esančių vienetų skaičius būtų lyginis. Pavyzdžiui, jei $m = (1, 0, 0, 0)$, tai $x = (1, 0, 0, 0, 1, 0, 1)$, nes:



Dekoduojame tokiu būdu. Parodysime, kad šis kodas visada ištaiso pavienę klaidą. Tarkime, klaida įvyko pozicijoje, kuri priklauso tik vienam skrituliui, pavyzdžiui, pozicijoje x_5. Dekoduodami patikriname, kuriuose skrituliuose vienetų skaičius yra nelyginis. Matome, kad viršutiniame skritulyje jis nelyginis, kituose — lyginis. Padarome išvadą, kad klaida įvyko pozicijoje, kuri priklauso viršutiniam skrituliui ir nepriklauso kitiems skrituliams. Tokia pozicija tėra tik viena, ir tai būtent x_5, galime ištaisyti joje padarytą klaidą. Lygiai taip pat vienareikšmiškai nustatome klaidos poziciją, jei klaida padaryta simbolyje, priklausiančiame lygiai dviems iš trijų skritulių, pavyzdžiui, m_2 (randame, kad vienetų skaičius nelyginis viršutiniame ir dešiniajame skrituliuose, ir nusprendžiame, kad klaida padaryta pozicijoje, kuri priklauso šiems dviems skrituliams, ir nepriklauso trečiajam), ir jei klaida padaryta simbolyje, priklausančiame visiems trims skrituliams. Taigi, kad ir kur būtų padaryta klaida, mes galime rasti jos poziciją ir ją ištaisyti.

Bet dviejų klaidų kodas jau nebeištaiso. Pavyzdžiui, tarkime, klaidos įvyko pozicijose m_4 ir x_7. Tada vienetų skaičius nelyginis pasidarys tik dešiniajame skritulyje, dėl to ištaisysime x_6: dekoduodami ne tik neištaisysime klaidų, bet dar daugiau jų įvelsime. Žodžiu, šis kodas yra tiksliai vieną klaidą taisantis kodas. Ir tiksliai 2 klaidas aptinkantis, nes padarius dvi klaidas, būtinai kuriame nors skritulyje vienetų skaičius pasidarys nelyginis, iš ko ir nuspręsime, kad įvyko klaida, o įvykus trims klaidoms, vienetų skaičius visuose skrituliuose gali išlikti lyginis, ir klaidų galime nepastebėti (taip bus, pavyzdžiui, jei klaidos įvyks m_1, x_5 ir x_7 pozicijose).

Tai mums leidžia rasti dar vieną kodo parametrą: minimalus šio kodo atstumas $d = 3$.

Baigiamosios pastabos. Daviau tik mažiukus pavyzdukus ir pačią teorijos pradžią, o iš tikro klaidas taisančių kodų teorija yra didžiulė ir šiuo metu sparčiai besivystanti. Tai viena iš kelių matematikos sričių, kur naujausios matematinės teorijos turi tiesioginį praktinį pritaikymą. Ir atvirkščiai, tai pavyzdys, kaip iš praktinio uždavinio išsivystė graži matematinė teorija.

4.5 Kriptografija

Įsivaizduokite tokią situaciją: du asmenys — A ir B (vadinkime juos Algiu ir Birute) — nori pasikeisti slapta informacija, o trečias asmuo Z (Zigmas) nori ją sužinoti. Kad Zigmas jos nesužinotų, perduodama informacija yra šifruojama. *Šifravimas* — tai duomenų kodavimas, norint paslėpti jų turinį. Užšifruotas pranešimas vadinamas *šifru*. *Kriptografija* yra mokslas, kuriantis ir nagrinėjantis įvairias šifravimo sistemas, dar vadinamas *kriptosistemomis* arba *šifrais*.

Įvairios kriptosistemos naudojamos jau nuo antikos laikų. Jau minėjau Cezario šifrą, kai lotynų abėcėlės raidė užšifruojama raide, esančia trimis pozicijomis abėcėlėje toliau, t.y. raidė A užšifruojama raide D, raidė B - raide E, ir t.t. Tokios paprastos kriptosistemos pasižymi tuo, kad norint išsaugoti pranešimo slaptumą, reikia slėpti patį šifravimo algoritmą. Kai Zigmas algoritmą sužinos, reikės galvoti kitą algoritmą. Geriau būtų, jei šifravimo algoritmas turėtų parametrą, vadinamą *raktu*, kurį ir reikėtų slėpti. Jei Zigmas jį sužinos, pakeisime jį kitu raktu, ir Zigmas vėl negalės dešifruoti mūsų pranešimų, nors ir naudosime tą patį algoritmą.

Paprasčiausias šifravimo su raktu pavyzdys — tai pagerintas Cezario kodas: perstumkime raides ne per tris pozicijas, o per kintamą pozicijų skaičių, priklausantį nuo rakto. Tarkime, raktas yra žodis `aibė`, jo raidės yra atitinkamai 1-oji, 13-oji, 3-ioji ir 9-oji lietuviškos abėcėlės raidės. Norėdami užšifruoti pranešimą, pirmą jo raidę perstumiame per vieną poziciją, antrą per trylika, trečią per tris, ketvirtą per devynias, penktą vėl per vieną ir t.t. Nors Zigmas ir žinotų, kad Algis ir Birutė naudoja tokį šifravimo algoritmą, norėdamas dešifruoti, turėtų kažkokiu būdu sužinoti ir raktą.

Antrojo pasaulinio karo metu tokios rūšies šifravimo algoritmai, tik, aišku, žymiai sudėtingesni, ir buvo naudojami (garsiausias jų pavyzdys yra vokiečių ENIGMA). Buvo kuriamos sudėtingos mašinos, norint dešifruoti priešininko pranešimus. Tos mašinos ir tapo šiuolaikinių kompiuterių pradininkėmis.

Tokios kriptosistemos išsivystė į tokias dabar plačiai naudojamas kriptosistemas, kaip DES, IDEA ir kt. Jos vadinamos *slapto rakto* kriptosistemomis. Jos pasižymi tuo, kad ir šifravimui, ir dešifravimui naudojamas tas pats raktas (kuris dėl to turi būti slaptas, nes jei Zigmas jį sužinotų, irgi galėtų dešifruoti Algio Birutei siunčiamus pranešimus). Geriausiai žinoma slaptojo rakto kriptosistema yra DES. Šifruojant šia kriptosistema, pranešimo simboliai yra daug kartų keičiami vietomis, sudėdinėjami tarpusavyje ir su rakto simboliais, ir pan. Tobulėjant kompiuteriams, ši sistema darosi jau nebe tokia saugi, ir ją ateityje pakeis naujas slapto rakto kriptosistemų standartas, vadinamas AES.

Didžiausias slaptojo rakto kriptosistemų trūkumas yra tai, kad raktas turi būti slaptas. Jei slaptai bendrauti tarpusavyje nori tūkstančiai žmonių, tai kiekviena jų pora turės turėti savo slaptą raktą. Kaip tvarkyti tiek raktų, ir, dar svarbiau, kaip jais apsikeisti, kad raktas išliktų slaptas? 1976 metais Diffie ir Hellman pasiūlė visiškai naujo tipo kriptosistemą, kuri išsprendžia šias problemas. Tokio tipo kriptosistemos vadinamos *viešo rakto* kriptosistemomis. Jos pasižymi tuo, kad turi vieną raktą šifravimui, o kitą dešifravimui. Raktas šifravimui gali būti viešas, prieinamas visiems, ir jis vadinamas *viešuoju* raktu. Raktas dešifravimui turi būti slaptas, kad tik jį žinantis galėtų dešifruoti pranešimą. Jis vadinamas *privačiuoju* raktu. Taigi, Birutė, norinti gauti šifruotus pranešimus, susigeneruoja sau raktų porą — viešąjį ir privatųjį raktus. Viešąjį paskelbia visiems, ir Algis tuo Birutės viešu raktu šifruoja pranešimus jai. Bi-

rutė, gavusi pranešimą, jį dešifruoja savo slaptu raktu. Kadangi to rakto daugiau niekas nežino, niekas kitas negali to pranešimo dešifruoti, tik Birutė.

Aišku, viešo rakto kriptosistemos turi būti tokios, kad viešojo rakto žinojimas neleistų rasti privačiojo rakto ir dešifruoti pranešimo. Paprastai viešo rakto kriptosistema sudaroma, naudojant funkciją f, kurios reikšmės yra lengvai apskaičiuojamos, bet jos atvirkštinės funkcijos f^{-1} reikšmės yra labai sunkiai apskaičiuojamos, nebent žinotum kai ką daugiau apie tą funkciją, kažkokį jos parametrą. Tada pranešimas m užšifruojamas, paprasčiausiai apskaičiuojant $f(m)$ reikšmę, o turėdamas $f(m)$, pradinį pranešimą $m = f^{-1}(f(m))$ gali apskaičiuoti tik tas, kuris žino kažkokios papildomos slaptos informacijos apie atvirkštinę funkciją f^{-1}, nes bendru atveju jos reikšmės yra labai sunkiai apskaičiuojamos. Ta papildoma slapta informacija apie f^{-1} ir yra privatus raktas.

Geriausiai žinoma viešo rakto kriptosistema yra RSA. Trumpai pasiaiškinkime, kokią funkciją jinai naudoja.

Visų pirma apibrėžkime Oilerio funkciją $\phi(n)$ — tai skaičius tokių m, $1 \leq m < n$, kurie yra tarpusavyje pirminiai su n, t.y.

$$\phi(n) = |\{m : 1 \leq m < n,\ \gcd(m, n) = 1\}|,$$

kur $\gcd(m, n)$ yra skaičių m ir n bendras didžiausias daliklis. Apie Oilerio funkciją mums užteks žinoti tokį faktą: jei p ir q yra du skirtingi pirminiai skaičiai, tai $\phi(pq) = (p-1)(q-1)$.

Raktų parinkimas. Birutė pasirenka du skirtingus pirminius skaičius p, q ir juos sudaugina: $n = pq$. Dar jinai pasirenka natūraliųjų skaičių e, kad jis būtų tarpusavyje pirminis su $\phi(n) = (p-1)(q-1)$. Viešas raktas ir bus pora $K_v = (e, n)$. Privatus raktas bus toks skaičius d, kad $ed \equiv 1 \bmod \phi(n)$. Primenu, kad $a \equiv b \bmod c$ reiškia, kad padalijęs a iš c gausime tą pačią liekaną, kaip ir padalijęs b iš c. Lengva pastebėti, kad pastaroji sąlyga yra ekvivalenti sąlygai $a - b \equiv 0 \bmod c$, t.y. skirtumą $a - b$ padalijus iš c, liekana bus 0, kitaip sakant, $a - b$ dalinasi iš c be liekanos, o tai reiškia, kad egzistuoja sveikasis skaičius t toks, kad $a - b = tc$, arba kitaip $a = b + tc$.

Tokį skaičių d rasti galima naudojantis Euklido algoritmu dviejų skaičių bendram didžiausiam dalikliui rasti, kurio čia nenagrinėsime. Skaičių p ir q daugiau nebereikės, ir juos galima ištrinti (ir netgi patartina, kad jie nepakliūtų į rankas Zigmui).

Šifravimas. Pranešimai, kuriuos bus galima siųsti Birutei, yra skaičiai nuo 2 iki $n - 1$. Algis, turėdamas Birutės viešą raktą (e, n), užšifruoja pranešimą m tokiu būdu: $c \equiv m^e \bmod n$, $2 \leq c < n$ (apskaičiuoja m^e, dalija rezultatą iš n ir ima liekaną), ir siunčia užšifruotą pranešimą c Birutei.

Dešifravimas. Dešifravimo algoritmas visiškai toks pat kaip ir šifravimo — dešifruotas pranešimas r bus gaunamas tokiu būdu: $r \equiv c^d \bmod n$.

Įsitikinkime, kad dešifruotas pranešimas r sutampa su pradiniu pranešimu m. Parodykime, kad $r \equiv m \bmod n$. Iš pradžių parodykime, kad $r \equiv m \bmod p$. Jei $m \equiv 0 \bmod p$, tai $c \equiv 0 \bmod p$, todėl $r \equiv m \bmod p$. Tegu $\gcd(m, p) = 1$. Žinome, kad $c^d \equiv (m^e)^d \equiv m^{ed} \bmod p$. Be to, $ed \equiv 1 \bmod \phi(n)$. Ši sąlyga reiškia, kad egzistuoja sveikasis skaičius t toks, kad $ed = 1 + t\phi(n) = 1 + t(p-1)(q-1)$. Įstatę šią lygybę, gauname $m^{ed} \equiv m^{1+t(p-1)(q-1)} \equiv m\big(m^{p-1}\big)^{t(q-1)} \bmod p$. Dabar galime pasinaudoti mažąja Ferma teorema, kuri tvirtina, kad jei p — pirminis skaičius, tai su bet kokiu sveikuoju skaičiumi a, kuriam $\gcd(a, p) = 1$,

$a^{p-1} \equiv 1 \bmod p$. Taigi, gauname $r \equiv m\big(m^{p-1}\big)^{t(q-1)} \equiv m \cdot 1^{t(q-1)} \equiv m \bmod p$. Lygiai taip pat gauname, kad $r \equiv m \bmod q$. Todėl $r \equiv m \bmod n$. Be to, žinome, kad $m < n$ ir $r < n$, todėl $r = m$, ką ir reikėjo įrodyti.

Tarkime, jog Zigmas sužinojo, kad užkoduotas pranešimas yra c. Jis taip pat žino Birutės viešą raktą (e, n). Norėdamas rasti pradinį pranešimą m, jis turi išspręsti tokią lygtį: $m^e \equiv c \bmod n$. Pasirodo, tai padaryti yra labai sunku (aišku, pakankamai dideliam n). Jis gali bandyti ir kitaip: iš lygties $ed \equiv 1 \bmod \phi(n)$ rasti privatų raktą d. Bet $\phi(n) = (p-1)(q-1)$ jis taip pat neturi. Kad galėtų apskaičiuoti $\phi(n)$, jis turi iš pradžių rasti p ir q, t.y. jis turi išskaidyti n pirminiais dauginamaisiais. Dideliems n tai irgi labai sunkus uždavinys.

Kad RSA kriptosistema būtų saugi, Birutės pasirinkti pirminiai skaičiai p ir q turi būti labai dideli, apie 100–200 dešimtainių skaitmenų. Dabar taikomi algoritmai leidžia greitai patikrinti, ar tokios eilės skaičius yra pirminis, ar ne (tai užtrunka tik kelias minutes). Taigi, Birutei nekils problemų parenkant p ir q. O Zigmui kils, jei jis bandys skaidyti n pirminiais dauginamaisiais. Gi n bus sudarytas iš apie 200–400 skaitmenų, o dabartiniai algoritmai tokios eilės skaičiui išskaidyti sugaištų šimtus metų.

Pavyzdys. *Raktų parinkimas.* Visų pirma turime parinkti raktus. Be abejo, mes negalime pasirinkti tokių didelių skaičių p ir q, kokie turėtų būti iš tikrųjų, bet tam, kad pamatytume, kaip vaikia RSA, užteks ir nedidelių skaičių. Imkime, pavyzdžiui, $p = 7$, $q = 11$. Tada $n = 77$. Dabar reikia pasirinkti e, tarpusavyje pirminį su $\phi(n) = (p-1)(q-1) = 6 \cdot 10 = 60$. Imkime, pavyzdžiui, $e = 13$. Taigi, viešas raktas bus pora $(13, 77)$. Privatus raktas bus toks d, kad $ed \equiv 1 \bmod \phi(n)$, t.y. $13d \equiv 1 \bmod 60$. Galite nesunkiai patikrinti, kad $d = 37$ (iš tikro, $ed = 13 \cdot 37 = 481 = 1 + 8 \cdot 60$).

Šifravimas. Tarkime, kažkas nori mums perduoti pranešimą $m = 2$. Jisai užšifruoja šį pranešimą, pasinaudodamas mūsų viešu raktu: $2^{13} = 8192 \equiv 30 \bmod 77$ (dalija 2^{13} iš 77 ir ima liekaną). Užšifruotas pranešimas $c = 30$.

Dešifravimas. Kad dešifruotume pranešimą, mums reikia apskaičiuoti $30^{37} \bmod 77$. Pasinaudosime modulio savybe: jei $a' \equiv b' \bmod c$ ir $a'' \equiv b'' \bmod c$, tai $a'a'' \equiv b'b'' \bmod c$. Taigi, jei $a' \equiv b' \bmod c$, tai $a'^2 \equiv b'^2 \bmod c$. Gauname tokią skaičiavimų moduliu 77 seką:

$$\begin{aligned}
30^1 &\equiv 30 \\
30^2 &\equiv 900 \equiv 53 \\
30^4 &\equiv 53^2 \equiv 2809 \equiv 37 \\
30^8 &\equiv 37^2 \equiv 1369 \equiv 60 \\
30^{16} &\equiv 60^2 \equiv 3600 \equiv 58 \\
30^{32} &\equiv 58^2 \equiv 3364 \equiv 53.
\end{aligned}$$

Taigi,

$$30^{37} = 30^{32} \cdot 30^4 \cdot 30^1 \equiv 53 \cdot 37 \cdot 30 = 1961 \cdot 30 \equiv 36 \cdot 30 = 1080 \equiv 2 \bmod 77.$$

Gavome pranešimą, kuris ir buvo užšifruotas.

Zigmas. Ką gali padaryti Zigmas? Jei jam pasiseka gauti užšifruotą pranešimą $c = 30$, jisai turi rasti tokį m, kad $m^{13} \equiv 30 \bmod 77$. Šiuo metu nėra žinoma metodų, kurie būtų žymiai

greitesni už paprasčiausią galimų reikšmių perrinkimą. Taigi, Zigmas bandytų visus m nuo 2 iki $n-1$ įstatyti ir tikrinti, ar jie tenkina lygybę. Kai n labai didelis, jis užgaištų tam labai ilgai.

Kitas kelias, kurį gali bandyti Zigmas: išskaidyti $n = 77$. Kaip matėme, dideliems n tai irgi užima labai daug laiko. □

Elektroninis parašas. Kartais svarbu ne pranešimo slaptumas, o jo tapatumas. Tokiu atveju naudojamas *elektroninis parašas* (dar vadinamas *skaitmeniniu parašu*).

Tarkime, Algis siunčia pranešimą Birutei. Prie siunčiamo pranešimo pridedamas kažkoks duomenų kiekis, kuris vadinamas elektroniniu parašu. Jisai užtikrina, kad pranešimą tikrai pasiuntė Algis, o ne koks nors kitas asmuo, ir kad pranešimas pakeliui nebuvo Zigmo pakeistas.

Turint viešo rakto kriptosistemą, galima sudaryti elektroninio parašo schemą. Imkime, pavyzdžiui, RSA.

Algis turi savo viešą raktą $K_v = (e, n)$ ir privatų $K_p = d$. Jisai nori pasiųsti pranešimą m Birutei. Prieš siųsdamas jis jį užšifruoja savo privačiu raktu (pasirašo), t.y. apskaičiuoja $c \equiv m^d \bmod n$, ir siunčia Birutei abu — ir pranešimą m, ir elektroninį parašą c. Birutė, norėdama įsitikinti, kad pranešimą m atsiuntė tikrai Algis, dešifruoja c pasinaudodama Algio viešu raktu, t.y. apskaičiuoja $r \equiv c^e \bmod n$, ir patikrina, ar $r = m$. Jei taip, tai tikrai siuntė Algis, nes tik jis žino savo privatų raktą d, todėl tik jis galėjo apskaičiuoti $c \equiv m^d \bmod n$.

Be abejo, galima derinti pasirašymą su šifravimu: Algis gali pasirašyti pranešimą savo privačiu raktu ir užšifruoti jį Birutės viešu raktu. Taip bus ir išsaugotas pranešimo slaptumas, ir užtikrintas jo tapatumas.

5 skyrius

Algoritmai ir jų sudėtingumas

5.1 Algoritmai

Nors *algoritmo* sąvoką paprastai sieja su programomis bei kompiuteriais, šis žodis jau yra žinomas daugiau kaip tūkstantį metų. Jis kilęs iš žymaus Rytų mokslininko Abu Ja'far Mohammed ibn Mûsâ al-Khowârizmî (783–850) vardo. XII amžiuje Europoje pasirodė šio mokslininko traktato apie aritmetiką vertimas iš arabų į lotynų kalbą, kuris vadinosi "Dixit algorizmi" ("Al Chorezmis pasakė"). Kadangi šiame traktate buvo aprašomi veiksmai su skaičiais pozicinėje skaičiavimo sistemoje (pavydžiui, sudėtis stulpeliu), tai šie veiksmai, o vėliau ir kitos įvairios procedūros buvo pradėta vadinti algoritmais. Vieną iš tokių procedūrų, kuri tebenaudojama iki šiol dviejų sveikų skaičių bendram didžiausiam dalikliui rasti, dar apie 300 m. p.m.e. apraše Euklidas.

"Algoritmas" yra pirminė matematikos bei informatikos sąvoka, panašiai kaip sąvokos "skaičius", "aibė" ir kitos. Neformaliai (intuityviai) algoritmą galima apibrėžti kaip objektą, turintį šias savybes:

(1) *Programiškumas*. Tai reiškia, kad algoritmas suprantamas kaip baigtinis taisyklių arba komandų rinkinys (dar vadinamas programa), nusakantis kaip vienus objektus (duomenis), atlikus baigtinį skaičių nesudėtingų operacijų perdirbti į kitus objektus (rezultatus).

(2) *Diskretumas*. Tai reiškia, kad algoritmas apibrėžia nuoseklų duomenų apdorojimo (skaičiavimo) procesą, suskirstytą į atskirus etapus (žingsnius).

(3) *Determinuotumas*. Paprastai reikalaujama, kad po kiekvieno žingsnio būtų vienareikšmiškai apibrėžtas kitas žingsnis arba būtų nurodyta, jog skaičiavimo procesas pasibaigė. Teorinėms reikmėms kartais naudojami ir nedeterminuoti algoritmai, leidžiantys keletą galimų kito žingsnio pasirinkimo variantų.

(4) *Žingsnių elementarumas (lokalumas)*. Taisyklė, nusakanti kaip pakeisti duomenis per 1 žingsnį turi būti paprasta ir lokali (nežymiai pakeičianti duomenis). Pavyzdžiui, algorit-

mas negali turėti tokios komandos kaip “Dabar įrodykite Didžiąją Ferma Teoremą laipsnio rodikliui $n = 73$”.

(5) *Masiškumas*. Tai reiškia, kad algoritmas turėtų būti pritaikomas įvairiems duomenims iš tam tikros leistinų pradinių duomenų aibės (paprastai begalinės), o algoritmo darbo rezultatai taip pat priklauso apibrėžtai leistinų rezultatų aibei. Pavyzdžiui, taisyklė “norint sudėti du skaičius 2 ir 3 reikia užrašyti, kad atsakymas lygus 5” nėra laikoma algoritmu, nes jos negalima pritaikyti norint sudėti du bet kokius sveikuosius skaičius. Paprastai algoritmuose duomenys ir rezultatai būna *konstruktyvūs objektai*. Konstruktyviu objektu vadiname baigtinį objektą, turintį diskrečią struktūrą ir vidinę koordinačių sistemą. Pavyzdžiui, sveikasis skaičius užrašytas pozicinėje skaičiavimo sistemoje yra konstruktyvus objektas. Tuo tarpu baigtinė aibė kaipo tokia dar nėra konstruktyvus objektas. Ji tampa konstruktyviu objektu, tik įvedus kokią nors tvarką tarp jos elementų.

Prie aukščiau išvardintų savybių dažnai dar yra pridedama *rezultatyvumo* savybė, reikalaujanti, kad algoritmas po baigtinio žingsnių skaičiaus sustotų ir išduotų kokį nors rezultatą. Tačiau algoritmų teorijoje paprastai nagrinėjami ir tie algoritmai, kurie kai kuriems pradiniams duomenims gali dirbti be galo arba jiems sustojus rezultatas būna neapibrėžtas. Tokie algoritmai realizuoja ne visur apibrėžtas funkcijas. Dabar pateiksime algoritmų pavyzdžių.

Pavyzdys 5.1.1. Turime sutvarkytą objektų (pvz., skaičių) sąrašą $A = \{a_1 < a_2 < \cdots < a_n\}$ ilgio n. Jame reikia rasti duotą objektą x arba nustatyti, kad tokio objekto sąraše nėra, t.y. apskaičiuoti funkciją

$$\text{label}(A, x) = \begin{cases} i, & \exists a_i = x, \\ 0, & a_i \neq x \ \forall i = 1, \ldots, n. \end{cases}$$

Naudosime *binariosios paieškos* algoritmą, kurio idėja yra sąrašą nuosekliai dalinti pusiau ir lyginti objektą x su viduriniu sąrašo objektu tam, kad nustatyti, kuriame iš dviejų gautų perpus trumpesnių sąrašų reikia tęsti paiešką.

function label = binary_search(A, x)
$i := 1$; $j := n$;
while $i \neq j$ **do**
begin
  $m := \lceil (i + j)/2 \rceil$;
  if $x = a_m$ **then begin** label $:= m$; **return; end**
    else if $x > a_m$ **then** $i := m + 1$ **else** $j := m$;
end;
if $x = a_i$ **then** label $= i$ **else** label $= 0$;
return

Nesunku įsitikinti, kad šis algoritmas atlikęs ne daugiau kaip $c \cdot \log n$ žingsnių (kur c yra nedidelė konstanta) sustos ir išduos atsakymą (rezultatą) label. Taigi, jo sudėtingumas (žr. skyrelį 4.5) bus O$(\log n)$.

Pavyzdys 5.1.2. Turime grafą $G = (V, E)$, turintį n viršūnių ir m briaunų. Reikia nustatyti, ar šis grafas yra Hamiltono grafas, t.y. ar egzistuoja ciklas C, prasidedantis bet kurioje grafo viršūnėje, pereinantis lygiai po 1 kartą per kiekvieną kitą grafo viršūnę ir grįžtantis į pradinę viršūnę. Taigi, konstruosime algoritmą, apskaičiuojantį funkciją

$$\text{HAMILTON}(V, E) = \begin{cases} \text{YES}, & \text{jei grafe } G = (V, E) \text{ egzistuoja Hamiltono ciklas} \\ \text{NO}, & \text{priešingu atveju.} \end{cases}$$

Naudosime *pilno perrinkimo* algoritmą, kurio idėja yra procedūros generate_all_permutations pagalba generuoti skaičių $123 \ldots n$ visus galimus kėlinius $i_1 i_2 \ldots i_n$ ir naudojant procedūrą verify tikrinti ar viršūnių seka $v_{i_1}, v_{i_2}, \ldots, v_{i_n}$ grafe G sudaro Hamiltono ciklą tol, kol rasime pirmąjį tokį ciklą arba perrinksime visus kėlinius ir įsitikinsime, kad tokio ciklo duotajame grafe nėra.

function HAMILTON = pilnas_perrinkimas(V, E)
$n :=$ size(V);
DERINIAI := generate_all_permutations(n);
$nfact :=$ size(DERINIAI);
HAMILTON := NO;
for $i := 1$ **to** $nfact$ **do**
  if verify(DERINIAI(i)) **then** HAMILTON := YES;
return

Koks šio algoritmo sudėtingumas? Visų galimų kėlinių generavimui reikės O$(n \cdot n!)$ žingsnių (pabandykite patys sukonstruoti algoritmą, generuojantį visus skirtingus kėlinius; tai gana įdomus uždavinys). Kiekvieno kėlinio patikrinimui procedūra verify atliks O(n^2) žingsnių (reikės patikrinti ar aibėje E yra visos ciklo $C = v_{i_1}, v_{i_2}, \ldots, v_{i_n}$ briaunos $(v_{i_1}, v_{i_2}), \ldots, (v_{i_{n-1}}, v_{i_n})$, (v_{i_n}, v_{i_1}). Gauname bendrą sudėtingumą O$(n^2 \cdot n!)$. Pavyzdžiui, nedideliam grafui turinčiam 10 viršūnių šis algoritmas turės atlikti apie 3,6 mlrd. žingsnių (laikant, kad konstanta c, įeinanti į žymėjimą "O" yra lygi 10).

Trumpa Hamiltono ciklo egzistavimo uždavinio analizė rodo, kad šis uždavinys teoriškai yra paprastas (nesunku sukonstruoti algoritmą), tačiau praktiškai jis yra labai sunkus. Aišku, kad grafams turintiems 15 ir daugiau viršūnių aukščiau pateiktas algoritmas jau netiks. Šis uždavinys demonstruoja, kad algoritmų sudėtingumo analizė yra svarbi informatikos ir diskrečiosios matematikos sritis.

5.2 Determinuotos Turing'o mašinos

Turing'o mašina (arba *1-juoste determinuota Turing'o mašina*) vadiname rinkinį

$$M = \langle \Sigma, Q, \delta, q_0, F \rangle,$$

kur:

- $\Sigma = \{a_1, \ldots, a_n\}$ yra baigtinė aibė, vadinama *abėcėle*;
- $Q = \{q_0, q_1, \ldots, q_k\}$ yra baigtinė *būsenų aibė*;
- $\delta : Q \times \Sigma \to Q \times \Sigma \times \{\mathrm{K}, \mathrm{N}, \mathrm{D}\}$ yra dalinė (ne visur apibrėžta) *pėrėjimų funkcija*;
- $q_0 \in Q$ – pradinė būsena; ir
- $F \subseteq Q$ yra galutinių būsenų aibė.

Determinuota Turing'o mašina interpretuojama kaip mašina, turinti begalinę juostą, suskirstytą į ląsteles, ir palei juostą slenkančią skaitymo bei rašymo galvutę, kuri fiksuotu laiko momentu mato vieną juostos ląstelę ir gali perskaityti koks abėcėlės Σ ženklas ten įrašytas, vietoje jo įrašyti kitą ženklą bei pasislinkti per vieną ląstelę į kairę arba į dešinę, arba likti stovėti toje pat vietoje. Aibė $\{\mathrm{K}, \mathrm{N}, \mathrm{D}\}$ vadinama galimų galvutės postūmių aibe, kur K atitinka postūmį į kairę, D atitinka postūmį į dešinę, ir N reiškia, kad galvutė niekur nejuda.

Kadangi funkcija δ yra baigtinė funkcija, tai jos reikšmes galima pateikti lentele, kurios pirmuose dviejuose stulpeliuose bus išvardintos galimos jos dviejų argumentų reikšmės $q_p \in Q$ ir $a_s \in \Sigma$, o likusiuose trijuose stulpeliuose nurodomi trys funkcijos reikšmės komponentai $q_r \in Q$, $a_t \in \Sigma$ ir $J \in \{\mathrm{K}, \mathrm{N}, \mathrm{D}\}$. Tokios lentelės eilutės dažnai vadinamos komandomis ir užrašomos pavidalu

$$q_{i_1} a_{j_1} \to q_{i_2} a_{j_2} J.$$

Vienas iš Turing'o mašinos abėcėlės ženklų vadinamas tuščiu ("blank")ir žymimas B $(\mathrm{B} \in \Sigma)$. Jis yra skirtas tuščioms ląstelėms žymėti ir negali būti naudojamas pradinių duomenų bei tarpinių rezultatų, su kuriais dirba Turing'o mašina kodavimui. Pradiniu laiko momentu mašina visada yra pradinėje būsenojeq_0 ir mato pirmą iš kairės netuščią ląstelę, kurioje prasideda pirmasis algoritmui vykdyti reikalingas duomuo. Duomenys tarp savęs atskiriami vienu ar daugiau tuščių ženklų ar kitais sutartiniais abėcėlės ženklais. Dešiniau nuo paskutinių duomenų visa juosta vėl užpildyta tuščiais ženklais. Taigi, kiekvienu laiko momentu tik baigtinė juostos sritis gali turėti netuščias ląsteles (kadangi po baigtinio žingsnių skaičiaus galvutė bus pasiekusi tik baigtinį ląstelių skaičių į kairę ir į dešinę nuo pradinės ląstelės). Ši netuščia sritis (pradedant pirmąja iš kairės netuščia ląstele ir baigiant paskutine iš dešinės netuščia ląstele) dar yra vadinama reikšmine juostos dalimi.

Turing'o mašina perdirba žodžius abėcėlėje $A/\{\mathrm{B}\}$ į kitus žodžius toje pat abėcėlėje. Kiekvienu laiko momentu Turing'o mašiną galima pilnai apibrėžti nurodant jos būseną, galvutės vietą

ir reikšminės juostos dalies turinį. Tokį momentinį Turing'o mašinos aprašą dar vadina *konfigūracija*. Taigi, Turing'o mašinos konfigūracija yra žodis pavidalo

$$K = a_{j_1} \dots a_{j_{l-1}} q_{i_1} a_{j_l} \dots a_{j_m},$$

kuris reiškia, kad duotu momentu juostos reikšminėje dalyje stovi žodis $a_{j_1} \dots a_{j_{l-1}} a_{j_l} \dots a_{j_m}$, mašina yra būsenoje q_{i_1}, o jos skaitymo ir rašymo galvutė mato ženklą a_{j_l}. Tokiu atveju mašina ieško komandos, kurios kairėje pusėje stovi pora $q_{i_1} a_{j_l}$. Jei tokios komandos nėra, mašina sustoja. Tarkime, mašina rado tokią komandą pavidalo

$$q_{i_1} a_{j_l} \to q_{i_2} a_{j_t} J.$$

Šiuo atveju mašina toje ląstelėje, kurią mato skaitymo bei rašymo galvutė, ištrina ženklą a_{j_l}, vietoje jo įrašo ženklą a_{j_l}, atlieka judesį J (į kairę, į dešinę arba niekur) ir pereina į naują būseną q_{i_2}. Tarkime, nagrinėjamoje komandoje judesys $J = \mathrm{D}$. Įvykdžius šią komandą mašinos konfigūracija pasikeičia iš konfigūracijos K į naują konfigūraciją

$$K' = a_{j_1} \dots a_{j_{l-1}} a_{j_t} q_{i_2} a_{j_{l+1}} \dots a_{j_m}.$$

Jei nauja mašinos būsena q_{i_2} yra iš aibės F, mašina sustoja. Jei ne, ieško kitos komandos.

Taigi, Turing'o mašinos darbą nuo pradinio momento iki sustojimo (tarkime, po s žingsnių) galima pilnai aprašyti nurodant konfigūracijų seką $K_0 \to K_1 \to \dots \to K_s$, kur K_0 yra pradinė konfigūracija, o K_s yra galutinė konfigūracija po sustojimo. Mašinai sustojus jos *darbo rezultatu* vadinsime žodį, užrašytą netuščiose ląstelėse nuo sustojimo vietos iki pirmos iš kairės (skaičiuojant nuo sustojimo vietos) tuščios ląstelės. Jei mašina sustoja prie tuščios ląstelės arba visai nesustoja, jos darbo rezultatą laikome neapibrėžtu.

Tarkime, duota Turing'o mašina M ir žodis $u \in (A/\{\mathrm{B}\})^*$. Mašinos M darbo rezultatą, jai pradėjus savo darbą iš pradinės konfigūracijos $K_0 = q_0 u$, žymėsime $M(u)$. Jei $M(u)$ neapibrėžtas, žymėsime $M(u) \uparrow$.

Toliau naudosime abėcėlę $\{0, 1, \mathrm{B}\}$ ir apibrėšime kaip Turing'o mašina skaičiuoja dalines funkcijas $f : \mathbb{N}^n \to \mathbb{N}$. Iš natūraliųjų skaičių sudarytą vektorių $\mathbf{u} = (u_1, \dots, u_n)$ juostoje vaizduosime žodžiu $\tilde{u}$, sudarytu iš skaičių $u_1, \dots, u_n$ dvejetainių kodų, atskirtų tuščiais ženklais B. Sakysime, kad mašina M ***skaičiuoja funkciją*** $f : \mathbb{N}^n \to \mathbb{N}$, jei $\forall \mathbf{u} \in \mathbb{N}^n$:

(1) $f(\mathbf{u})$ reikšmė apibrėžta tada ir tik tada kai $M(\tilde{u})$ apibrėžtas, ir

(2) jei $f(\mathbf{u})$ apibrėžta, tai $M(\tilde{u})$ yra $f(\mathbf{u})$ dvejetainis kodas.

Dalinę funkciją $f : \mathbb{N}^n \to \mathbb{N}$ vadiname *apskaičiuojama pagal Turing'ą*, jei egzistuoja Turing'o mašina M, skaičiuojanti f. Apskaičiuojamų pagal Turing'ą funkcijų klasę žymėsime $\mathcal{T}$.

Pavyzdys 5.2.1. Parodysime, kad funkcija

$$f(x) = \begin{cases} 1, & x \text{ dalinasi iš } 2, \\ 0, & \text{priešingu atveju} \end{cases}$$

yra apskaičiuojama pagal Turing'ą. Nesunku įsitikinti, kad šią funkciją skaičiuos tokia mašina:

$$\begin{aligned} q_0 0 &\to q_0 0\mathrm{D} \\ q_0 1 &\to q_0 1\mathrm{D} \\ q_0 \mathrm{B} &\to q_1 \mathrm{BK} \\ q_1 0 &\to q_2 1\mathrm{N} \\ q_1 1 &\to q_2 0\mathrm{N} \end{aligned}$$

kurios galutinių būsenų aibė yra $F = \{q_2\}$.

5.3 Nedeterminuotos Turing'o mašinos

Nedeterminuota Turing'o mašina vadiname rinkinį

$$M = \langle \Sigma, Q, \delta, q_0, F \rangle,$$

kur:

- $\Sigma = \{a_1, \ldots, a_n\}$ yra baigtinė aibė, vadinama *abėcėle*;
- $Q = \{q_0, q_1, \ldots, q_k\}$ yra baigtinė *būsenų aibė*;
- $\delta \subseteq Q \times \Sigma \times Q \times \Sigma \times \{\mathrm{K}, \mathrm{N}, \mathrm{D}\}$ yra *pėrėjimų aibė*;
- $q_0 \in Q$ – pradinė būsena; ir
- $F \subseteq Q$ yra galutinių būsenų aibė.

Nesunku pastebėti, kad nedeterminuota Turing'o mašina skiriasi nuo determinuotos tik tuo, kad vietoje perėjimų funkcijos δ čia turime perėjimų aibę δ, o tai reiškia, kad nedeterminuota Turing'o mašina gali turėti kelias komandas su vienoda kairiąja ir skirtingomis dešiniosiomis pusėmis. Tokiu atveju laikome, kad skaičiavimo procesas išsišakoja ir mašina vienu metu pereina į tiek skirtingų naujų konfigūracijų, kiek buvo tokių skirtingų komandų. Jei kurioje nors šakoje mašina vėl randa kelias skirtingas komandas (su vienoda kairiąja puse), tai ši šaka išsišakoja į kelias naujas šakas ir t.t. Taigi, vietoje nuoseklios konfigūracijų sekos turėtos determinuotos Turing'o mašinos atveju, čia mes gauname *skaičiavimo medį*, sudarytą iš konfigūracijų K_i.

Nedeterminuotos Turing'o mašinos paprastai naudojamos ne bet kokioms funkcijoms apskaičiuoti, o tik taip vadinamų *egzistavimo problemų* sprendimui, t.y. kai reikia tik atsakyti, ar sprendinys egzistuoja ar ne. Pavyzdžiui, uždavinys HAMILTON kai reikia nustatyti, ar duotame grafe egzistuoja Hamiltono ciklas yra egzistavimo problema. Tokiems uždaviniams pakanka naudoti abėcėlę $\Sigma = \{0, 1, \mathrm{B}\}$, kadangi pradinius duomenis galima koduoti dvejetainiais skaičiais. Pasirinkus tokią abėcėlę, bet kuri nedeterminuota Turing'o mašina M skaičiuos funkciją $f : \{0,1\}^* \to \{0,1\}$.

Jei $x \in \{0,1\}^*$, tai laikysime, kad $f(x) = 1$, jei atsiras tokia konfigūracijų seka (t.y. skaičiavimo medžio šaka) $K_0 \to K_1 \to \cdots \to K_s$, kur K_0 yra pradinė konfigūracija $K_0 = q_0x$ o K_s yra galutinė konfigūracija, kurioje mašina M sustoja ir išduoda atsakymą 1. Jei duotai x reikšmei tokios šakos skaičiavimo medyje kuri baigtųsi 1 nėra, tai laikome, kad $f(x) = 0$. Šiuo atveju visose skaičiavimo medžio šakose mašina išduoda 0 arba dirba be galo.

Pavyzdys 5.3.1. Pateiksime pavyzdį nedeterminuotos Turing'o mašinos, kuri tikrina ar išpildoma yra dviejų literalų (kintamųjų be neiginio arba su neiginiu) konjunkcija $F = F_1 \,\&\, F_2$, kur $F_1, F_2 \in \{x_1, \neg x_1, x_2, \neg x_2\}$. Pasirinkę abėcėlę $\Sigma = \{\mathrm{B}, 0, 1, 2, \neg, \&\}$, kintamuosius x_1 ir x_2 koduosime atitinkamai skaičiais 1 ir 2. Pavyzdžiui, jei pradinė formulė yra $F = x_1 \,\&\, \neg x_2$, tai mašinos įėjimas bus $x = 1\&\neg 2$. Atsakymas $M(x)$ bus 1, jei duota formulė yra išpildoma ir 0 priešingu atveju. Viena iš galimų mašinų (programų) gali atrodyti taip:

$$
\begin{aligned}
q_0 1 &\to q_1 1\mathrm{D}\\
q_0 1 &\to q_2 0\mathrm{D}\\
q_0 2 &\to q_3 1\mathrm{D}\\
q_0 2 &\to q_4 0\mathrm{D}\\
q_0 \neg &\to q_0 \neg\mathrm{D}\\
q_1 \& &\to q_1 \&\mathrm{D}\\
q_1 \neg &\to q_1 \neg\mathrm{D}\\
q_1 1 &\to q_1 1\mathrm{D}\\
q_1 2 &\to q_1 1\mathrm{D}\\
q_1 2 &\to q_1 0\mathrm{D}\\
q_1 \mathrm{B} &\to q_5 1\mathrm{K}\\
q_2 \& &\to q_2 \&\mathrm{D}\\
q_2 \neg &\to q_2 \neg\mathrm{D}\\
q_2 1 &\to q_2 0\mathrm{D}\\
q_2 2 &\to q_2 1\mathrm{D}\\
q_2 2 &\to q_2 0\mathrm{D}\\
q_2 \mathrm{B} &\to q_5 1\mathrm{K}\\
q_3 \& &\to q_3 \&\mathrm{D}\\
q_3 \neg &\to q_3 \neg\mathrm{D}\\
q_3 1 &\to q_3 1\mathrm{D}\\
q_3 1 &\to q_3 0\mathrm{D}\\
q_3 2 &\to q_3 1\mathrm{D}\\
q_3 \mathrm{B} &\to q_5 1\mathrm{K}\\
q_4 \& &\to q_4 \&\mathrm{D}\\
q_4 \neg &\to q_4 \neg\mathrm{D}\\
q_4 1 &\to q_4 1\mathrm{D}\\
q_4 1 &\to q_4 0\mathrm{D}\\
q_4 2 &\to q_4 0\mathrm{D}
\end{aligned}
$$

$$
\begin{aligned}
q_4\mathrm{B} &\to q_5 1\mathrm{K}\\
q_5 1 &\to q_5 1\mathrm{K}\\
q_5 \neg &\to q_7 0\mathrm{D}\\
q_5 \& &\to q_5 \&\mathrm{K}\\
q_5 0 &\to q_6 0\mathrm{K}\\
q_5\mathrm{B} &\to q_7 1\mathrm{D}\\
q_6 \neg &\to q_5 \neg\mathrm{K}\\
q_6 \& &\to q_7 0\mathrm{D}\\
q_7\alpha &\to q_8 \mathrm{BK}
\end{aligned}
$$

kur q_8 yra galutinė būsena, o paskutinė komanda reiškia iš viso 5 komandas gaunamas įstatant $\alpha = 0, 1, \neg, \&, \mathrm{B}$).

Būsenoje q_0 mašina įstato vietoje pirmo sutikto kintamojo dvi galimas reikšmes 0 ir 1 (lygiagrečiai) ir įsimena koks tai buvo kintamasis ir kokia reikšmė įstatyta, kad sutikus vėl formulėje tą patį kintamąjį įstatytų tą pačią reikšmę (būsena q_1 atitinka vietoje x_1 įstatyta 1, būsena q_2 atitinka vietoje x_1 įstatyta 0, būsena q_3 atitinka vietoje x_2 įstatyta 1 ir būsena q_4 atitinka vietoje x_2 įstatyta 0). Būsenose q_1–q_4 mašina perbėga toliau formulės F kodą x iš kairės į dešinę įstatydama reikšmes vietoje kito rasto kintamojo. Būsenoje q_5 mašina grįžta atgal iš kairės į dešinę ir nustatinėja ar toms reikšmėms duota formulė yra teisinga, t.y. virsta 1. Nustačius, kad formulė klaidinga (atitinkamai teisinga) mašina įrašo į juostą 0 (atitinkamai 1), po to pereina į būseną q_6, įrašo po atsakymo tarpą B, paeina į kairę ir pereina į galutinę būseną q_8, t.y. sustoja.

5.4 Apskaičiuojamos funkcijos

Norint formaliai nagrinėti algoritmus kaip matematinius objektus, buvo pasiūlyti įvairūs skaičiavimo modeliai: Turing'o mašinos (beje, apibrėžtos gerokai anksčiau už realių kompiuterių atsiradimą), RAM (tiesioginės kreipties į atmintį) mašinos, dalinės rekursyviosios funkcijos, Markovo algoritmai. Dalinių funkcijų $f : \mathbb{N}^n \to \mathbb{N}$, apskaičiuojamų naudojant išvardintus modelius, klases pažymėkime atitinkamai $\mathcal{T}$ (jau buvo skyrelyje 4.1), RAM, DR ir M. Buvo įrodyta, kad nepaisant šių modelių skirtingumo, kiekviename iš jų gauname tą pačią apskaičiuojamų funkcijų klasę. Šį rezultatą pateikiame be įrodymo:

Teorema 5.4.1. *$\mathcal{T} = RAM = DR = M$.*

Remdamasis šiuo rezultatu Church'as pasiūlė tezę, skelbiančią, kad bet kuris iš šių skaičiavimo modelių sutampa su intuityvia algoritmo sąvoka, t.y. jei mes galime sugalvoti kokį nors neformalų algoritmą, tai visada šį algoritmą galima formalizuoti (tiksliai užrašyti) kiekviename iš modelių. Pažymėję raide $\mathcal{A}$ klasę algoritminių intuityviąja prasme funkcijų $f : \mathbb{N}^n \to \mathbb{N}$, gauname, kad Church'o tezė skelbia:

$$\mathcal{A} = \mathcal{T} = \mathrm{RAM} = \mathrm{DR} = \mathrm{M}.$$

Kadangi apskaičiuojamų funkcijų klasė nepriklauso nuo modelio pasirinkimo, tai funkcijas, apskaičiuojamas pagal Turing'ą, toliau vadinsime tiesiog *apskaičiuojamomis funkcijomis*. Jei dalinės funkcijos $f : \mathbb{N}^n \to \mathbb{N}$, reikšmė taške $\mathbf{x} \in \mathbb{N}^n$ apibrėžta, žymėsime $f(\mathbf{x}) \downarrow$, jei neapibrėžta, žymėsime $f(\mathbf{x}) \uparrow$. Įrodysime keletą rezultatų apie apskaičiuojamas funkcijas.

Teorema 5.4.2. Teorema 4.2. *Egzistuoja neapskaičiuojama funkcija* $f : \mathbb{N} \to \mathbb{N}$.

Įrodymas. I būdas (nekonstruktyvus). Yra žinoma, kad natūrinių skaičių aibės poaibių aibės galia yra kontinuumas. Kadangi kiekvieną poaibį $A \subseteq \mathbb{N}$ atitinka jo charakteringoji funkcija $\chi_A : \mathbb{N} \to \mathbb{N}$ tokia, kad

$$\chi_A(x) = \begin{cases} 1, & x \in A; \\ 0, & x \notin A, \end{cases}$$

tai funkcijų $f : \mathbb{N} \to \mathbb{N}$ taip pat yra ne mažiau, negu kontinuumas. Kita vertus, kiekvieną apskaičiuojamą funkciją $f : \mathbb{N} \to \mathbb{N}$ atitinka kokia nors Turing'o mašina, t.y. baigtinė komandų seka. Nesunku parodyti, kad visas Turing'o mašinas galima sunumeruoti natūriniais skaičiais (žr. įrodymą II būdu). Taigi, apskaičiuojamų funkcijų aibė yra skaiti. Kadangi skaičiosios aibės galia yra mažesnė už kontinuumą, tai egzistuoja neapskaičiuojamos funkcijos.

II būdas (konstruktyvus). Įvedus apribojimą, kad Turing'o mašinos komandose naudojamų būsenų numeriai neviršytų bendro komandų skaičiaus (kiekvieną programą galima perrašyti taip, kad ji tenkintų šį apribojimą), mes gausime, kad pasirinkus konkrečią abėcėlę $A = \{0, 1, \mathrm{B}\}$ kiekvienam natūriniam skaičiui $n \in \mathbb{N}$ egzistuoja tik baigtinis skaičius skirtingų Turing'o mašinų, turinčių lygiai n komandų. Taigi, visas galimas Turing'o mašinas galima išdėstyti paeiliui, imant iš pradžių visas galimas mašinas, turinčias 1 komandą ir jas išdėstant abėcėlės tvarka kaip žodyne, po to imant visas galimas mašinas iš 2 komandų ir jas išdėstant apibrėžta tvarka ir t.t. Dar daugiau, nesunku parodyti, kad šią Turing'o mašinų numeraciją natūraliaisiais skaičiais galima atlikti efektyviai, t.y. sukonstruoti algoritminę funkciją iš Turing'o mašinų aibės į natūraliųjų skaičių aibę tokią, kad pagal pateiktą mašinos komandų seką galima rasti duotos mašinos numerį (skaitinį jos kodą) ir atvirkščiai, pagal mašinos numerį galima dekoduoti ir surasti, kokios komandos ją sudaro. Taigi, Turing'o mašinos sudaro skaičią aibę

$$\mathcal{M} = \{M_1, M_2, \ldots\}.$$

Kadangi vieną ir tą pačią apskaičiuojamą funkciją galima apskaičiuoti su be galo daug skirtingų Turing'o mašinų (pridedant, pavyzdžiui, nereikšmingas komandas), tai aukščiau pateikta Turing'o mašinų numeracija duoda taip pat ir efektyvią apskaičiuojamų vieno kintamojo funkcijų $\phi : \mathbb{N} \to \mathbb{N}$ numeraciją

$$\mathcal{T} = \{\phi_1, \phi_2, \ldots\},$$

kurioje kiekviena apskaičiuojama funkcija gauna be galo daug skirtingų numerių (numeracija su pasikartojimais).

Dabar mes galime apibrėžti dalinę funkciją

$$f(x) = \begin{cases} 0, & \phi_x(x) \uparrow; \\ \uparrow, & \phi_x(x) \downarrow. \end{cases}$$

Jei ši funkcija būtų apskaičiuojama, tai ji patektų į apskaičiuojamų funkcijų numeraciją ir gautų, pavydžiui, numerį $x_0 \in \mathbb{N}$. Tada funkcija ϕ_{x_0} tenkintų prieštaringą lygybę

$$\phi_{x_0}(x_0) = \begin{cases} 0, & \phi_{x_0}(x_0) \uparrow; \\ \uparrow, & \phi_{x_0}(x_0) \downarrow. \end{cases}$$

Taigi, taip apibrėžta funkcija negali būti apskaičiuojama.

Kas yra parašęs bent vieną programą dažnai susiduria su reiškiniu, kai programa kažką daro, bet nesustoja ir neišduoda rezultatų. Dažniausiai paaiškėja, kad dėl programoje įveltos klaidos ji "užsiciklina" ir dirba be galo ilgai. Todėl būtų labai patogu, jei egzistuotų algoritmas, kuriam pateikus jūsų programos kodą ir pradinius duomenis, jis atsakytų, ar programa tiems duomenims sustos ar ne. Deja, dabar mes įrodysime, kad sustojimo problema yra algoritmiškai neišsprendžiama. □

Teorema 5.4.3. *Sustojimo problema $\phi_x(y) \downarrow$? yra algoritmiškai neišsprendžiama.*

Įrodymas. Tarkime, kad sustojimo problema algoritmiškai išsprendžiama. Tada būtų išsprendžiama ir paprastesnė problema $\phi_x(x) \downarrow$?, nes jai galima būtų pritaikyti tą patį algoritmą. Taigi, jei sustojimo problema būtų algoritmiškai išsprendžiama, tada funkcija

$$\kappa(x) = \begin{cases} 1, & \phi_x(x) \downarrow; \\ 0, & \phi_x(x) \uparrow \end{cases}$$

būtų apskaičiuojama. Naudodami šią funkciją mes galime išreikšti teoremos 4.2 įrodyme naudotą funkciją f:

$$f(x) = \begin{cases} 0, & \kappa(x) = 0; \\ \uparrow, & \kappa(x) = 1. \end{cases}$$

Pakeitę funkciją κ skaičiuojantį algoritmą taip, kad vietoje sustojimo kai $\kappa(x) = 1$ jis dirbtų be galo ilgai, mes gausime algoritmą, skaičiuojantį funkciją f. Kadangi buvome įrodę, kad funkcija f nėra apskaičiuojama, gauname prieštaravimą. □

5.5 Algoritmų sudėtingumas

Ankstesniuose skyreliuose parodėme, kad vienos problemos yra algoritmiškai išsprendžiamos, kitos — ne. Pavyzdžiui, Hamiltono ciklo grafe egzistavimo problema yra algoritmiškai išsprendžiama, o sustojimo problema bei 10-ji Hilberto problema (klausianti, ar duotoji Diofanto lygčių sistema turi sprendinį sveikųjų skaičių aibėje) yra algoritmiškai neišsprendžiamos. Tačiau net ir

tai, kad egzistuoja uždavinio sprendimo algoritmas, kartais nepadeda išspręsti duoto uždavinio, nes tas algoritmas turi atlikti tiek žingsnių, kiek kartais negalėtų per visą mūsų gyvenimą atlikti greičiausias kompiuteris. Taigi, tokiu atveju teoriškai algoritmas egzistuoja, bet praktiškai jis neduoda jokios naudos. Todėl svarbu mokėti matuoti uždavinių sudėtingumą ir nustatyti ar duotam uždaviniui egzistuoja tokio sudėtingumo algoritmas, kurį mes galėtume praktiškai realizuoti ir gauti sprendinį per mums priimtiną laiko tarpą.

Uždavinių klase vadinsime vienodo tipo bet skirtingo dydžio ir priklausančių nuo skirtingų duomenų uždavinių aibę. Pavyzdžiui, klasę HAMILTON sudaro tokie uždaviniai, kuriuose reikia nustatyti ar duotame grafe egzistuoja Hamiltono ciklas. Aišku, kad kokį bepaimtume algoritmą, jo žingsnių skaičius priklausys tiek nuo grafo dydžio, tiek nuo jo struktūros, t.y. nuo pradinių duomenų ilgio ir nuo pačių duomenų. Kadangi yra sunku aprėpti visus galimus duomenis, tai nuo antro parametro paprastai apsiribojama, nagrinėjant kiek konkretus algoritmas darys žingsnių blogiausiu atveju, t.y. jei iš vienodo dydžio duomenų pakliūs tokie, kurie yra nepalankiausi šiam algoritmui.

Tarkime, parametras n charakterizuoja uždavinio dydį palyginus jį su kitais tos pačios klasės uždaviniais. Uždavinio dydis dažniausiai priklauso nuo jo pradinių duomenų dydžio. Pradiniai duomenys algoritmuose gali būti sveikieji skaičiai, aibės (masyvai), matricos, grafai ir įvairūs kiti objektai. Pavyzdžiui, kuo didesnis yra natūralusis skaičius, tuo ilgesnis yra jo dvejetainis kodas. Kuo yra didesnė matrica, tuo daugiau ji turi eilučių ir stulpelių. Kuo didesnis grafas, tuo daugiau jis turi viršūnių ir lankų. Taigi, jei A yra natūralusis skaičius, tai jo dydis $n = |A| = \lceil \log_2 A \rceil$; jei $A = \{a_1, \ldots, a_n\}$ yra aibė, tai jos dydis yra elementų skaičius $n = |A|$; jei A yra kvadratinė $n \times n$ matrica, tai jos dydis yra matricos eilė n. Konkretaus uždavinio U iš klasės $\mathcal{U}$ dydį žymėsime $|U|$.

Tarkime, $\mathcal{U}$ yra uždavinių klasė ir A yra konkretus algoritmas šios klasės uždaviniams spręsti. Konkretaus uždavinio $U \in \mathcal{U}$ sudėtingumu sprendžiant jį algoritmu A vadiname algoritmo A žingsnių skaičių iš pradinės konfigūracijos iki sustojimo ir žymėsime $L_A(U)$. Algoritmo žingsnių skaičių sprendžiant uždavinį U dar vadina laiku arba laiko sudėtingumu ir žymi $T_A(U)$, o panaudotos atminties kiekį vadina erdve arba erdvės sudėtingumu ir žymi $S_A(U)$. Uždavinių klasės $\mathcal{U}$ sprendžiant šios klasės uždavinius algoritmu A sudėtingumu vadiname sudėtingiausio iš vienodo dydžio uždavinių sudėtingumą ir žymime

$$L_A^{\mathcal{U}}(n) = \max_{U \in \mathcal{U}:\ |U|=n} L_A(U).$$

Naudojant du skirtingus algoritmus A ir B gausime skirtingus sudėtingumus $L_A^{\mathcal{U}}(n)$ ir $L_B^{\mathcal{U}}(n)$. Todėl uždavinių klasės $\mathcal{U}$ sudėtingumu vadinsime dydį nepriklausantį nuo konkretaus algoritmo, o būtent pasirinksime paties geriausio algoritmo sudėtingumą:

$$L^{\mathcal{U}}(n) = \min_A L_A^{\mathcal{U}}(n).$$

Kai uždavinių klasė $\mathcal{U}$ yra numanoma, kartais jos sudėtingumą žymi tiesiog $L(n)$. Taigi, uždavinių klasės sudėtingumas yra ta pati Shannon'o funkcija, kurią mes jau naudojome, kai algoritmai buvo schemos, jų sudėtingumas buvo schemų dydis, o uždavinių klasė buvo Būlio funkcijų

realizacija schemomis. Pavyzdžiui, yra žinoma, kad

$$L^{\text{binary\_search}}(n) = \mathrm{O}(\log_2 n), \quad L^{\text{HAMILTON}}(n) = \mathrm{O}(n^2 \cdot n!).$$

Kaip matome, pirmasis uždavinys yra labai paprastas, o antras labai sudėtingas. Jam nėra žinoma jokio *polinominio algoritmo*, t.y. tokio algoritmo A, kuriam egzistuotų konstantos $c > 0$ ir $k > 0$ tokios, kad būtų patenkinta nelygybė

$$L^{\text{HAMILTON}}(n) \leq \mathrm{O}(cn^k).$$

Tokių uždavinių, kuriems yra žinomi polinominiai algoritmai, visumą informatikoje žymi raide P. Taigi,

$$\mathrm{P} = \{\mathcal{U}\colon L^{\mathcal{U}}(n) = \mathrm{Pol}(n) = \mathrm{O}(cn^k)\}.$$

Jau matėme, kad nedeterminuoti algoritmai turi daugiau galimybių už determinuotus. Pavyzdžiui, visiškai paprasta sukonstruoti nedeterminuotus polinominius algoritmus uždaviniams HAMILTON arba SAT, kur SAT uždaviniai reiškia, kad yra duota Būlio formulė, priklausanti nuo n kintamųjų ir reikia nustatyti, ar ši formulė yra išpildoma. Spręsdami šį uždavinį mes naudodavome eksponentinio sudėtingumo algoritmą, sudarydami teisingumo lentelę, turinčią 2^n eilučių.

Tokių uždavinių, kuriems yra žinomi nedeterminuoti polinominiai algoritmai, visumą informatikoje žymi raide NP:

$$\mathrm{NP} = \{\mathcal{U}\colon \exists \text{ nedet. algoritmas } A\colon L_A^{\mathcal{U}}(n) = \mathrm{Pol}(n) = \mathrm{O}(cn^k)\}.$$

Kadangi determinuotus algoritmus galima kartu laikyti ir nedeterminuotais, tai akivaizdu, kad $\mathrm{P} \subseteq \mathrm{NP}$. Dauguma mokslininkų palaiko hipotezę, kad $\mathrm{P} \neq \mathrm{NP}$, tačiau jos įrodyti niekam nepavyksta. Tai viena centrinių informatikos ir diskrečiosios matematikos problemų. Neformaliai uždaviniai iš klasės P dar yra vadinami lengvais (paprastais), o uždaviniai, kuriems kol kas nežinomi polinominiai algoritmai (t.y. nežinoma ar jie priklauso P) vadinami sunkiais uždaviniais.