4 skyrius

Uždavinių sudėtingumo klasės

4.1 Kalbų ir uždavinių ryšys

Abstrakčiu uždaviniu vadiname sąryšį $Q \subset I \times O$, kur I yra įėjimų (duomenų) aibė, o O yra išėjimų (galimų sprendinių) aibė.

Pavyzdys 4.1.1 (Uždavinys SHORTEST-PATH**).** Duota grafas $G = (V, E)$ ir dvi to grafo viršūnės $v_1, v_2 \in V$. Rasti trumpiausią kelią $K(v_1, v_2)$ (kelio ilgiu laikome jį sudarančių briaunų skaičių). Čia įėjimų aibė yra grafų aibės ir jų viršūnių porų aibės Dekarto sandauga, t.y., $I = \{(V, E, v_1, v_2)\}$, o išėjimų aibė yra aibė grafo viršūnių baigtinių sekų: $O = \{\{u_1, \ldots, u_n\}: u_i \in V\}$.

Trumpiausio kelio radimo uždavinys $\mathrm{SHORTEST\text{-}PATH} \subset I \times O$ yra *optimizavimo* uždavinio pavyzdys.

Pavyzdys 4.1.2 (Uždavinys PATH**).** Duota grafas $G = (V, E)$, dvi to grafo viršūnės $v_1, v_2 \in V$ ir natūralusis skaičius $k \in \mathbb{N}$. Rasti, ar grafe G tarp viršūnių v_1 ir v_2 egzistuoja kelias $K(v_1, v_2)$, ne ilgesnis už k, t.y., toks, kad $|K(v_1, v_2)| \leqslant k$.

Tai yra *egzistavimo* uždavinio pavyzdys. Kadangi atsakymą "taip" galima koduoti 1, o atsakymą "ne" — 0, tai egzistavimo uždavinių išėjimų aibė yra aibė $\{0, 1\}$. Pažymėję $I = \{(V, E, v_1, v_2, k)\}$, kur $k \in \mathbb{N}$, gauname, kad $\mathrm{PATH} \subset I \times \{0, 1\}$ yra egzistavimo uždavinys.

Iš šių pavyzdžių matyti, kad bet kurį optimizavimo uždavinį Q atitinka egzistavimo uždavinys E, o atskirą optimizavimo uždavinio atvejį $q = (i, o) \in Q$ atitinka atskirų egzistavimo uždavinio atvejų aibė $\{e(k) = (i, k, o')\colon k \in \mathbb{N}\} \subset E$. Čia abstrakčiu uždaviniu vadiname to paties tipo uždavinių klasę (tai ką mes 1 skyriuje žymėjome $\mathcal{U}$), atskirais uždavinio atvejais vadiname konkrečius tos klasės uždavinius ($U \in \mathcal{U}$).

Parodysime (ne visai griežtai), kad nagrinėjant uždavinių sudėtingumą, galima apsiriboti egzistavimo uždaviniais. Kaip ir anksčiau, uždavinio Q atskiro atvejo $q = (i, o) \in Q$ dydžiu $|q|$ vadiname parametrą n, apibūdinantį q dydį klasėje Q.

Tegu $f:\mathbb{N}\to\mathbb{N}$. Uždavinį Q vadiname *f-sunkiu*, jei egzistuoja algoritmas A, kurio sudėtingumas sprendžiant uždavinį Q

$$\mathrm{TIME}_A(q)=\Omega(f(|q|))\quad \forall q\in Q.$$

Lema 4.1.1. *Nagrinėjant uždavinių sudėtingumą, galima apsiriboti egzistavimo uždaviniais, t.y., jei $f:\mathbb{N}\to\mathbb{N}$ ir egzistavimo uždavinys E yra f-sunkus, tai ir jį atitinkantis optimizavimo uždavinys Q yra f-sunkus.*

Įrodymas. Tarkime, Q ir E yra vienas kitą atitinkantys optimizavimo ir egzistavimo uždaviniai, ir uždavinys E yra f-sunkus. Įrodysime, kad tada ir uždavinys Q yra f-sunkus.

Tarkime, kad uždavinys Q yra paprastesnis už uždavinį E, t.y., egzistuoja algoritmas A toks, kad $\mathrm{TIME}_A(q)=\mathrm{o}(f(|q|))\ \forall q\in Q$. Kaip jau aukščiau minėjome, kiekvieną $q\in Q$ atitinka aibė $\{e(k)\}\subset E$. Be to, $|q|=|e(k)|$, nes egzistavimo uždaviniui reikia tų pačių duomenų, kaip ir optimizavimo uždaviniui, papildomai dar nurodant natūralųjį skaičių k. Pridėję dar vieną žingsnį prie algoritmo A mes gautume algoritmą B, sprendžiantį bet kurį uždavinį $e(k_0)$: tam, kad išspręsti $e(k_0)$, reikia pirma išspręsti Q, o po to gautą sprendinį palyginti su skaičiumi k_0. Aišku, kad algoritmo B sudėtingumas tada būtų

$$\mathrm{TIME}_B(e(k_0))=\mathrm{o}(f(|Q|))=\mathrm{o}(f(|e(k_0)|)),$$

o tai prieštarautų prielaidai, kad uždavinys E yra f-sunkus. □

Remdamiesi šia lema, toliau nagrinėsime tik abstrakčius uždavinius $Q\subset I\times\{0,1\}$.

Konkrečiu uždaviniu vadiname sąryšį $\mathcal{U}\subset\{0,1\}^*\times\{0,1\}$. Taigi, konkretaus uždavinio duomenys yra nulių bei vienetų seka, o jo sprendinys yra skaičius 0 arba 1. Vietoje abstrakčių uždavinių galima nagrinėti konkrečius uždavinius, naudojant *kodavimą* $e:I\to\{0,1\}^*$. Tada bet kurį abstraktų uždavinį $Q\subset I\times\{0,1\}$ atitiks konkretus uždavinys $\mathcal{U}=e(Q)\subset\{0,1\}^*\times\{0,1\}$. Naudodami abstrakčių uždavinių kodavimą, galėsime lyginti įvairaus tipo uždavinių sudėtingumą (uždavinių apie grafus, logines formules bei schemas, veiksmų su matricomis ir pan.).

Sakysime, kad algoritmas A sprendžia konkretų uždavinį $\mathcal{U}=e(Q)\subset\{0,1\}^*\times\{0,1\}$ per laiką $\mathrm{O}(f(n))$ ir žymėsime $T_A^{\mathcal{U}}(n)=\mathrm{O}(f(n))$ jei $\forall i\in\{0,1\}^*$,

$$T_A(i)=\mathrm{O}(f(|i|)),$$

kur $|i|$ yra žodžio $i\in\{0,1\}^*$ ilgis. *Sudėtingumo klase* P vadinsime aibę konkrečių uždavinių, kuriems egzistuoja algoritmai, sprendžiantys tuos uždavinius per polinominį laiką, t.y.,

$$\mathrm{P}=\{\mathcal{U}\text{: }\exists A\exists k \text{ tokie, kad } T_A^{\mathcal{U}}(n)=\mathrm{O}(n^k)\}.$$

Kadangi naudodami įėjimų aibės kodavimą $e:I\to\{0,1\}^*$ abstrakčius uždavinius galime koduoti konkrečiais uždaviniais, tai galime kalbėti ir apie polinominio sudėtingumo abstrakčius

uždavinius. Tačiau reikia susitarti, kokį kodavimą galima naudoti. Pasirodo, koduojant aibę I skirtingais būdais, vienu atveju konkretus uždavinys $e(Q)$ gali priklausyti aibei P, o kitu atveju gali nepriklausyti. Taip, pavyzdžiui, gali atsitikti, natūraliuosius skaičius koduojant dvejetainiu pavidalu ir "vienetainiu" pavidalu (t.y., skaičių n koduojant $n+1$ vienetu), nes antrasis kodas bus eksponentiškai ilgesnis už pirmąjį.

Funkciją $f : \{0,1\}^* \to \{0,1\}^*$ vadiname *polinomiškai apskaičiuojama*, jei egzistuoja polinominio sudėtingumo algoritmas A, kuris randa $f(x)$. Du kodavimus $e_1, e_2 : I \to \{0,1\}^*$ vadiname *polinomiškai susijusiais*, jei egzistuoja polinomiškai apskaičiuojamos funkcijos $f_{12}, f_{21} : \{0,1\}^* \to \{0,1\}^*$ tokios, kad

$$f_{12}(e_1(i)) = e_2(i) \quad \text{ir} \quad f_{21}(e_2(i)) = e_1(i) \quad \forall i \in I.$$

Lema 4.1.2. *Tarkime, $Q \subset I \times \{0,1\}$ yra abstraktus uždavinys ir du kodavimai e_1, e_2 yra polinomiškai susiję. Tada $e_1(Q) \in \mathrm{P} \iff e_2(Q) \in \mathrm{P}$.*

Įrodymas. Kadangi teiginys yra simetriškas, tai pakanka jį įrodyti tik į vieną pusę. Tarkime, kad $e_1(Q) \in \mathrm{P}$, t.y., egzistuoja algoritmas A ir egzistuoja natūralusis skaičius k tokie, kad A sprendžia $e_1(Q)$ per laiką $\mathrm{O}(|e_1(i)|^k)$ $\forall i \in I$. Be to, egzistuoja algoritmas B ir egzistuoja natūralusis skaičius l tokie, kad B randa $e_1(i)$ iš $e_2(i)$ per $\mathrm{O}(|e_2(i)|^l)$ žingsnių.

Norėdami išspręsti uždavinį $e_2(Q)$, taikome algoritmų A ir B kompoziciją. Duotą žodį $e_2(i)$ iš pradžių pateikiame algoritmui B, kuris randa žodį $e_1(i)$ per polinominį laiką, ir to žodžio $e_1(i)$ ilgis yra ne didesnis už algoritmo B žingsnių skaičių: $|e_1(i)| \le T_B \le |e_2(i)|^l$. Po to žodį $e_1(i)$ pateikę algoritmui A, gauname sprendinį $\alpha \in \{0,1\}$. Bendras žingsnių skaičius bus

$$\mathrm{O}(|e_1(i)|^k) = \mathrm{O}(|e_2(i)|^{lk}).$$

Taigi, $e_2(Q) \in \mathrm{P}$. $\square$

Šiame skyriuje koduodami abstrakčius uždavinius laikysimės reikalavimo, kad kodavimas būtų polinomiškai susijęs su *standartiniu kodavimu*. Standartiniu kodavimu vadiname kodavimą e_0, kuris:

(1) natūraliuosius skaičius koduoja dvejetainiu pavidalu, t.y., $e_0(n) = \alpha_1\alpha_2\ldots\alpha_k$, kur $k = \lfloor \log_2 n \rfloor + 1$, jei $n > 0$, ir $k = 1$, jei $n = 0$;

(2) baigtinę aibę A koduoja žodžiu, sudarytu iš tos aibės elementų kodų:

$$e_0(A) = e_0(a_1)\#e_0(a_2)\#\ldots\#e_0(a_n);$$

(3) grafą koduoja žodžiu, sudarytu iš to grafo viršūnių aibės ir briaunų aibės kodų: $e_0(G) = e_0(V)\#e_0(E)$ ir t.t.

Šis reikalavimas leis laisvai operuoti su abstrakčių uždavinių sudėtingumu. Uždavinio duomenų $i \in I$ kodą $e(i) \in \{0,1\}^*$ toliau žymėsime kampuotais skliausteliais, t.y. vietoje $e(i)$ rašysime tiesog $\langle i \rangle$, laikydami, kad kodavimas yra polinomiškai susijęs su standartiniu.

Jei turime abėcėlę Σ, *kalba* vadiname bet kokią aibę žodžių toje abėcėlėje. Taigi, kalba vadiname bet kokią aibę $L \subset \Sigma^*$. Toliau visur naudosime abėcėlę $\{0,1\}$ ir kalba vadinsime bet kokią aibę $L \subset \{0,1\}^*$. Kadangi kiekvienas konkretus uždavinys yra atvaizdavimas $u : \{0,1\}^* \to \{0,1\}$, tai kiekvieną konkretų uždavinį U atitinka kalba $L_U = u^{-1}(1)$, tai yra aibė žodžių, kuriems uždavinio U sprendinys yra lygus 1. Pavyzdžiui, kalba

$$\mathrm{PRIME} = \{10, 11, 101, 111, 1011, \ldots\}$$

atitinka abstraktų egzistavimo uždavinį, ar duotas natūralusis skaičius yra pirminis.

Taigi šiame skyrelyje mes pademonstravome, kad kalbant apie uždavinių sudėtingumą, vietoje optimizavimo uždavinių galima apsiriboti egzistavimo uždaviniais, pastaruosius galima koduoti konkrečiais uždaviniais, kurių duomenys sudaryti tik iš nulių ir vienetų, o konkrečius uždavinius atitinka kalbos abėcėlėje $\{0,1\}$. Tai mums leidžia vietoje uždavinių sudėtingumo klasių toliau nagrinėti kalbų sudėtingumo klases

4.2 Sudėtingumo klasė P

Algoritmas A *išsprendžia* kalbą L, jei $\forall x \in \{0,1\}^*$

$$A(x) = \begin{cases} 1, & x \in L, \\ 0, & x \notin L. \end{cases}$$

Algoritmas A *priima* kalbą L, jei

$$A(x) = 1 \iff x \in L.$$

Taigi, jei žodis nepriklauso kalbai L, tai išsprendžiantis algoritmas išduoda atsakymą 0, tuo tarpu priimantis algoritmas išduoda atsakymą 0 arba niekada nesustoja.

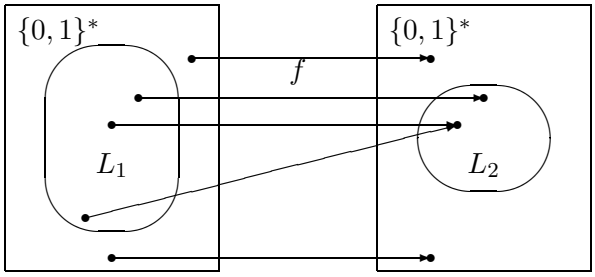
Sudėtingumo klase P vadiname polinomiškai išsprendžiamų kalbų klasę:

$$\mathrm{P} = \{L \subseteq \{0,1\}^* \colon \exists \text{ algoritmas } A \text{ toks, kad } A \text{ išsprendžia } L \text{ per polinominį laiką}\}.$$

Pavyzdys 4.2.1. Riboto ilgio kelio tarp dviejų grafo viršūnių egzistavimo uždavinį atitinka kalba

$$\mathrm{PATH} = \{\langle G, u, v, k\rangle \colon \begin{array}{l} G = (V, E) \text{ — grafas}, u, v \in V,\ k \geq 0 \text{ — sveikasis skaičius}, \\ \text{ir egzistuoja kelias } K(u,v) \text{ iš } u \text{ į } v \text{ ilgio} \leq k\}. \end{array}$$

Kadangi yra žinoma nemažai polinominių trumpiausio kelio radimo grafe algoritmų (pvz., Deikstros), tai pritaikę tokį algoritmą duomenims G, u, v ir jo gautą rezultatą (trumpiausio kelio ilgį) palyginę su skaičiumi k, gauname polinominį algoritmą, išsprendžiantį kalbą PATH, taigi $\mathrm{PATH} \in \mathrm{P}$.



Pav. 4.1: Polinominė redukcija.

Turing'o mašinos sustojimo problemą atitinka kalba

$$\mathrm{HALT} = \{\langle M, x\rangle\text{: Turing'o mašina } M \text{ duomenims } x \text{ sustoja, t.y., } M(x)\downarrow\}.$$

Yra žinoma, kad ši problema algoritmiškai neišsprendžiama, tačiau egzistuoja (nepolinominis) algoritmas, priimantis kalbą HALT. Šis algoritmas — tai universali Turing'o mašina U, modeliuojanti $M(x)$ darbą. Jei $M(x)\downarrow$, tai ir mašina $U(\langle M, x\rangle)$ sustos, t.y., priims žodį $\langle M, x\rangle$.

Pavyzdys 4.2.1 rodo, kad algoritmiškai išsprendžiamų ir priimamų kalbų klasės skiriasi. Tačiau jei mes apsiribosime tik polinominio sudėtingumo algoritmais, tai šios klasės sutaps.

Lema 4.2.1.

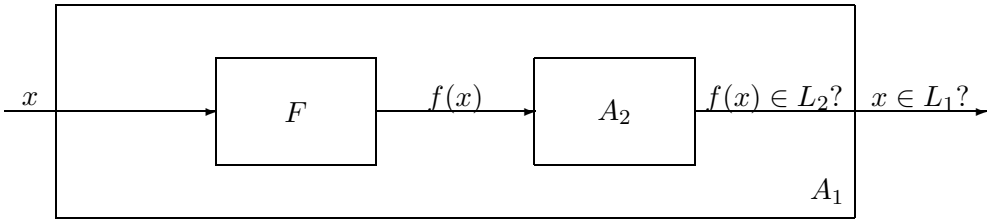
$$\mathrm{P} = \{L \subseteq \{0,1\}^*\text{: } \exists \textit{ algoritmas } A \textit{ toks, kad } A \textit{ priima } L \textit{ per polinominį laiką}\}.$$

Įrodymas. Kadangi kokią nors kalbą L išsprendžiantis polinominis algoritmas yra kartu ir kalbą L priimantis algoritmas, tai pakanka tik parodyti, kad jei egzistuoja kalbą L primantis polinominis algoritmas A, tai egzistuoja ir kalbą L išsprendžiantis polinominis algoritmas A'. Kadangi A polinominis, tai atsiras konstantos $c > 0$ ir $k \in \mathbb{Z}$ tokios, kad $A(x)$ sustoja po ne daugiau kaip $T = c|x|^k$ žingsnių kiekvienam $x \in L$. Algoritmas A' bus universalaus algoritmo modifikacija. Jis modeliuoja algoritmo $A(x)$ darbą ir skaičiuoja žingsnius. Jei po T žingsnių algoritmas $A(x)$ dar nesustojo, tai algoritmas $A'(x)$ sustoja ir išduoda atsakymą 0. Jei $A(x)$ sustoja kuriame nors žingsnyje $t \leq T$, tai $A'(x)$ irgi sustoja ir išduoda atsakymą sutampantį su $A(x)$ atsakymu. $\square$

Kalbą L_1 vadinsime *polinomiškai redukuojama* į kalbą L_2 ir žymėsime $L_1 \leq_{\mathrm{p}} L_2$ jei egzistuoja polinomialiai apskaičiuojama funkcija f tokia, kad

$$x \in L_1 \Longleftrightarrow f(x) \in L_2.$$

Polinominę redukciją iliustruoja 4.1 pav. Norint nustatyti, ar $x \in L_1$ pakanka nustatyti ar $f(x) \in L_2$.



Pav. 4.2: Lemos 4.2.2 įrodymo schema.

Pavyzdys 4.2.2. Apibrėžkime kalbas

$$\text{EULER-CYCLE} = \{\langle G\rangle\colon G \text{ — Euler'io grafas}\}$$

ir

$$\text{EVEN-SEQUENCE} = \{\langle(k_1, \ldots, k_n)\rangle\colon k_1, \ldots, k_n \text{ yra lyginiai sveikieji skaičiai}\}.$$

Kadangi yra žinoma teorema, kad grafas yra Euler'io tada ir tik tada, kai visų jo viršūnių laipsniai yra lyginiai, tai polinominis algoritmas F randantis duoto grafo visų viršūnių laipsnius polinomialiai redukuos kalbą EULER-CYCLE į kalbą EVEN-SEQUENCE: $\text{EULER-CYCLE} \leq_{\text{p}} \text{EVEN-SEQUENCE}$.

Polinominė redukcija leidžia lengvai įrodyti kalbų priklausomybę klasei P redukuojant vienas kalbas į kitas (t.y., nereikia konstruoti tiesioginio išsprendžiančio algoritmo).

Lema 4.2.2. *Jei* $L_1 \leq_{\text{p}} L_2$ *ir* $L_2 \in \text{P}$*, tai ir* $L_1 \in \text{P}$.

Įrodymas. Tarkime, kad A_2 yra polinominis algoritmas išsprendžiantis kalbą L_2 ir F yra polinominis algoritmas apskaičiuojantis funkciją f tokią, kad $x \in L_1 \iff f(x) \in L_2$. Konstruosime polinominį algoritmą A_1 išsprendžiantį L_1.

Algoritmas A_1 bus paprasčiausia algoritmų F ir A_2 superpozicija (žr.4.2 pav.). Norint nustatyti ar $x \in L_1$ reikia apskaičiuoti $f(x)$ ir gautą rezultatą pateikti kaip duomenis algoritmui A_2. Jo atsakymas rodys ar $x \in L_1$. Kadangi dviejų polinomų suma yra polinomas, tai algoritmas bus polinominis. □

4.3 Sudėtingumo klasė NP

Klasę NP galima apibrėžti kaip klasę kalbų L, kurioms egzistuoja nedeterminuota Turing'o mašina M priimanti L per polinominį laiką, t.y., $\forall x \in L$ atsiras skaičiavimo medžio šaka, kurioje M sustoja po ne daugiau kaip po $\text{O}(|x|^k)$ žingsnių ir išduoda atsakymą 1. Pateiksime kitą klasės NP

apibrėžimą, kuris leidžia žymiai palengvinti įrodymus, kad nagrinėjamos kalbos priklauso klasei NP. Yra įrodyta, kad šie apibrėžimai yra ekvivalentūs.

Sakysime kad algoritmas A (turintis 2 įėjimus) *patvirtina* žodį $x \in \{0,1\}^*$ jei egzistuoja kitas žodis $y \in \{0,1\}^*$ toks, kad $A(x,y) = 1$. Žodį y vadinsime žodžio x *liudijimu*. Kalba L vadinsime *patvirtinta* algoritmo A, jei

$$L = \{x \in \{0,1\}^* \text{: egzistuoja } y \in \{0,1\}^* \text{ toks, kad } A(x,y) = 1\}.$$

Sudėtingumo klase NP vadinsime aibę kalbų, kurios gali būti patvirtintos per *polinominį laiką*, naudojant *polinominio ilgio liudijimus*. Taigi, $L \in \mathrm{NP}$ tada ir tik tada kai egzistuoja polinominis algoritmas A, turintis du įėjimus, ir egzistuoja sveikas skaičius l toks, kad

$$L = \{x \in \{0,1\}^* \text{: egzistuoja liudijimas } y \in \{0,1\}^* \text{ ilgio } |y| = \mathrm{O}(|x|^l) \text{ toks, kad } A(x,y) = 1\}.$$

Pavyzdys 4.3.1. Panagrinėkime kalbą

$$\text{HAM-CYCLE} = \{\langle G = (E,V)\rangle \text{: } G \text{ — Hamilton'o grafas}\}.$$

Niekam dar nepavyko surasti polinominio algoritmo išsprendžiančio šią kalbą. Bandant perrinkti visus galimus Hamilton'o ciklus, gausime $\mathrm{O}(n!)$ sudėtingumo algoritmą. Tačiau, tarkime, pas jus ateina draugas ir sako: "Žinai, aš tavo grafe G radau Hamilton'o ciklą" bei pateikia jums viršūnių seką H. Aišku, kad tokiu atveju nesunku rasti polinominį algoritmą, kuris patikrina ar ta seka H tikrai bus Hamilton'o ciklas. Tereikia patikrinti ar H yra viršūnių aibės V kėlinys ir ar tikrai aibėje E egzistuoja briaunos jungiančios gretimas (o taip pat paskutinę ir pirmąją) sekos H viršūnes. Tam pakanka $\mathrm{O}(n^2)$ operacijų ($n = |V|$). Taigi, sekos H kodas $y = \langle H\rangle$ bus liudijimas, kad jūsų grafas G yra Hamilton'o grafas, ir kalba HAM-CYCLE priklausys klasei NP.

Akivaizdu, kad jei kalba L priklauso klasei P, tai ir jos papildinys $\overline{L}$ (arba $\text{co-}L$), kuris apibrėžiamas kaip

$$\overline{L} = \{0,1\}^* \setminus L = \{x \in \{0,1\}^* \text{: } x \notin L\}$$

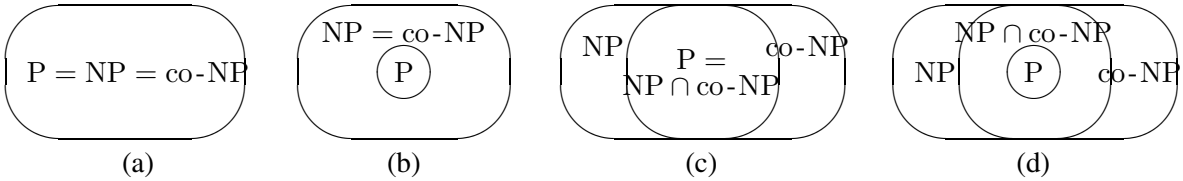
taip pat priklauso klasei P (tereikia algoritmo išsprendžiančio kalbą L atsakymus atimti iš 1). Ar tai galioja ir klasei NP, nėra žinoma. Pavyzdžiui, neaišku, ar kalba NONHAMILTON, kur

$$\text{NONHAMILTON} = \{0,1\}^* \setminus \text{HAM-CYCLE} = \{\langle G\rangle \text{: } G \text{ nėra Hamilton'o grafas}\}$$

priklauso NP, nes šiuo atveju yra sudėtinga rasti polinominio ilgio liudijimą, kad duotame grafe neegzistuoja Hamilton'o ciklo.

Sudėtingumo klase co-NP vadinsime aibę kalbų

$$\text{co-NP} = \{L \text{: } \overline{L} \in \mathrm{NP}\}.$$



(a) (b) (c) (d)

Pav. 4.3: Sudėtingumo klasės P, NP ir co-NP.

Lema 4.3.1. $\mathrm{P} \subseteq \mathrm{NP}$.

Įrodymas. Bet kuriam žodžiui $x \in \{0,1\}^*$ liudijimu y paėmę tuščią žodį, gauname, kad tas pats algoritmas kuris polinomiškai išsprendžia kalbą L, kartu ir patvirtina tą kalbą per polinominį laiką. Taigi, $L \in \mathrm{P} \Rightarrow L \in \mathrm{NP}$. $\square$

Lema 4.3.2. $\mathrm{P} \subseteq \mathrm{co\text{-}NP}$.

Įrodymas. Akivaizdu, kad klasė P yra uždara papildinio atžvilgiu, t.y., $L \in \mathrm{P} \iff \overline{L} \in \mathrm{P}$ (užtenka algoritmo atsakymus 1 ir 0 sukeisti vietomis). Todėl pagal ankstenę lemą $L \in \mathrm{P} \Rightarrow L \in \mathrm{co\text{-}NP}$. $\square$

Keturi galimi klasių P, NP ir $\mathrm{co\text{-}NP}$ tarpusavio sąryšių variantai yra vaizduojami 4.3 pav. Kuris variantas yra teisingas, kol kas nėra žinoma. Dauguma tyrinėtojų linkę manyti, kad labiausiai tikėtina yra situacija (d):

$$\mathrm{P} \subset \mathrm{NP} \cap \text{co-NP} \subset \mathrm{NP}, \text{co-NP}.$$

4.4 Sudėtingumo klasė NPC

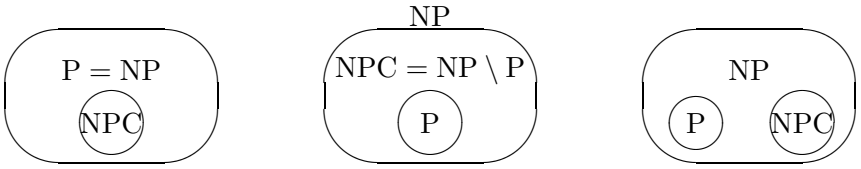
Kalbą L vadiname NP*-sunkia*, jei $L' \leq_{\mathrm{p}} L$ kiekvienai $L' \in \mathrm{NP}$. Kalbą L vadiname NP*-pilna*, jei:

(1) $L \in \mathrm{NP}$ ir

(2) L yra NP-sunki.

NP-pilnų kalbų klasę žymėsime NPC. Dabar parodysime, kad NP-sunkios kalbos betarpiškai siejasi su problemos "$\mathrm{P} = \mathrm{NP}$?" sprendimu.

Teorema 4.4.1. *(1) Jei bent viena* NP*-pilna kalba yra polinomiškai išsprendžiama (priklauso* P*), tai* $\mathrm{P} = \mathrm{NP}$.



(a) (b) (c)

Pav. 4.4: Sudėtingumo klasės P, NP ir NPC.

(2) Jei bent viena NP*-pilna kalba nėra polinomiškai išsprendžiama (nepriklauso* P*), tai ir visos kitos* NP*-pilnos kalbos nėra polinomiškai išsprendžiamos (*$\mathrm{P} \cap \mathrm{NPC} = \emptyset$*).*

Įrodymas. Tarkime, kad $L \in \mathrm{NPC}$ ir $L \in \mathrm{P}$. Jei $L' \in \mathrm{NP}$, tai pagal NP-pilnumo apibrėžimą $L' \leq_{\mathrm{p}} L$, taigi pagal 4.2.2 lemą ir $L' \in \mathrm{P}$. Taigi, $\mathrm{NP} \subseteq \mathrm{P}$, o tai reiškia, kad $\mathrm{P} = \mathrm{NP}$.

Norėdami įrodyti antrą teoremos dalį tarkime, kad $L \in \mathrm{NPC}$ ir $L \notin \mathrm{P}$. Jei atsirastų kalba L' tokia, kad $L' \in \mathrm{NPC}$ ir $L' \in \mathrm{P}$, tai kadangi $L \leq_{\mathrm{p}} L'$, pagal 4.2.2 lemą gautume $L \in \mathrm{P}$, o tai prieštarautų pradinei prielaidai. □

Pav. 4.4 vaizduoja tris galimus klasių P, NP ir NPC tarpusavio sąryšių variantus. Dauguma tyrinėtojų linkę manyti, kad teisingas yra variantas (c):

$$\mathrm{P} \subset \mathrm{NP}, \quad \mathrm{NPC} \subset \mathrm{NP} \quad \text{ir} \quad \mathrm{P} \cap \mathrm{NPC} = \emptyset.$$

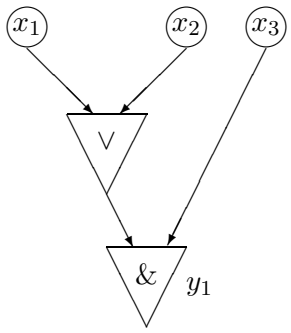
Kadangi NP-pilni uždaviniai yra tokie svarbūs algoritmų sudėtingumo teorijoje, tai kiekvieno naujo uždavinio įtraukimas į šią klasę vertinamas kaip naujas mokslinis rezultatas. Šiuo metu yra žinoma keli šimtai, o kartu su įvairiomis tų pačių uždavinių versijomis gal ir keli tūkstančiai NP-pilnų uždavinių. Norint įrodyti, kad kalba L yra NP-sunki, reikia įrodyti, kad į kalbą L galima polinomialiai redukuoti bet kurią kitą klasės NP kalbą. Tiesiogiai tai įrodyti būna gana sunku. Tačiau jei mes tai tiesiogiai įrodytume bent7 vienai kalbai, tai visoms kitoms kalboms galima būtų taikyti tokią lemą:

Lema 4.4.1. *Jei* $L' \leq_{\mathrm{p}} L$ *ir* $L' \in \mathrm{NPC}$*, tai* L *yra* NP*-sunki. Be to, jei* $L \in \mathrm{NP}$*, tai tada* $L \in \mathrm{NPC}$.

Įrodymas. Kadangi L' yra NP-sunki, tai kiekvienai $L'' \in \mathrm{NP}$ turime $L'' \leq_{\mathrm{p}} L'$. Polinominė redukcija tenkina tranzityvumo dėsnį (dviejų polinomų suma vėl bus polinomas), todėl gauname $L'' \leq_{\mathrm{p}} L$, taigi L yra NP-sunki. Jei dar žinoma $L \in \mathrm{NP}$, tai pagal apibrėžimą L bus NP-pilna. □

Ši lema mums duoda paprastą būdą kaip įrodyti, kad kuri nors kalba L yra NP-sunki:

(1) Reikia įrodyti, kad $L \in \mathrm{NP}$.



Pav. 4.5: Išpildoma Būlio schema.

(2) Reikia pasirinkti jau žinomą NP-pilną kalbą L'.

(3) Reikia pateikti algoritmą F realizuojantį funkciją $f : \{0,1\}^* \to \{0,1\}^*$, kuri kalbos L' žodžius atvaizduoja į kalbos L žodžius.

(4) Reikia įrodyti, kad bet kokiam $x \in \{0,1\}^*$ bus teisinga $x \in L'$ tada ir tik tada kai $f(x) \in L$.

(5) Reikia įrodyti, kad algoritmas F yra polinominio sudėtingumo.

Pirmasis uždavinys, kuris mums leis praverti klasės NPC duris ir įkišti į tarpdurį kojį (t.y., šį uždavinį) bus Būlio schemos išpildomumo uždavinys CIRCUIT-SAT.

4.5 Uždavinys CIRCUIT-SAT

Šiame skyrelyje nagrinėsime Būlio schemas su neribotu įėjimų skaičiumi ("unbounded fan-in circuits", angl.), t.y., sudarytas iš funkcinių elementų $\&, \vee, \neg$, kur elementai $\&$ realizuoja bet kokio ilgio konjunkcijas $z_1 \,\&\, \cdots \,\&\, z_k$, o elementai $\vee$ realizuoja bet kokio ilgio disjunkcijas $z_1 \vee \cdots \vee z_l$. Būlio schemą S su n įėjimų $x_1, \ldots, x_n$ ir 1 išėjimu vadiname *išpildoma*, jei egzistuoja kintamųjų $x_1, \ldots, x_n$ reikšmių rinkinys $(\alpha_1, \ldots, \alpha_n) \in \{0,1\}^n$ toks, kad įstačius į schemos S įėjimus reikšmes $x_1 = \alpha_1, \ldots, x_n = \alpha_n$, šios schemos išėjime gauname reikšmę 1. Pav. 4.5 vaizduojama schema S yra išpildoma, nes parinkę įėjimų reikšmes $x_1 = x_2 = x_3 = 1$, schemos išėjime gauname 1.

Kadangi Būlio schemos yra orientuotieji žymėtieji grafai, jas galima koduoti nuliukais ir vienetukais bet kuriuo būdu, naudojamu koduoti grafams. Taigi, kiekvienai schemai S galime priskirti jos kodą $\langle S \rangle$. Jei schema S turi n įėjimų ir m funkcinių elementų, tai akivaizdu, kad jų skaičius yra ne didesnis už schemos kodo ilgį ($n, m \leq \langle S \rangle$), nes kiekvienai grafo viršūnei koduoti prireiks bent vieno bito. Pažymėkime

$$\text{CIRCUIT-SAT} = \{\langle S \rangle \colon S \text{ — išpildoma Būlio schema}\}.$$

Akivaizdu, kad jei Būlio schema turi n įėjimų, tai perrinkus visas galimas jų reikšmes $(\alpha_1, \ldots, \alpha_n) \in \{0,1\}^n$ galima nustatyti, ar duotoji schema yra išpildoma (brutalios jėgos algoritmas). Kai schemos dydis polinomiškai priklauso nuo įėjimų skaičiaus, gauname, kad brutalios jėgos algoritmas yra eksponentinio sudėtingumo schemos kodo ilgio atžvilgiu. Deja, nieko geriau uždaviniui CIRCUIT-SAT nėra žinoma.

Teorema 4.5.1. CIRCUIT-SAT $\in$ NPC.

Įrodymas. Pagal NP pilnų kalbų apibrėžimą reikia įrodyti, kad:

(1) CIRCUIT-SAT $\in$ NP ir

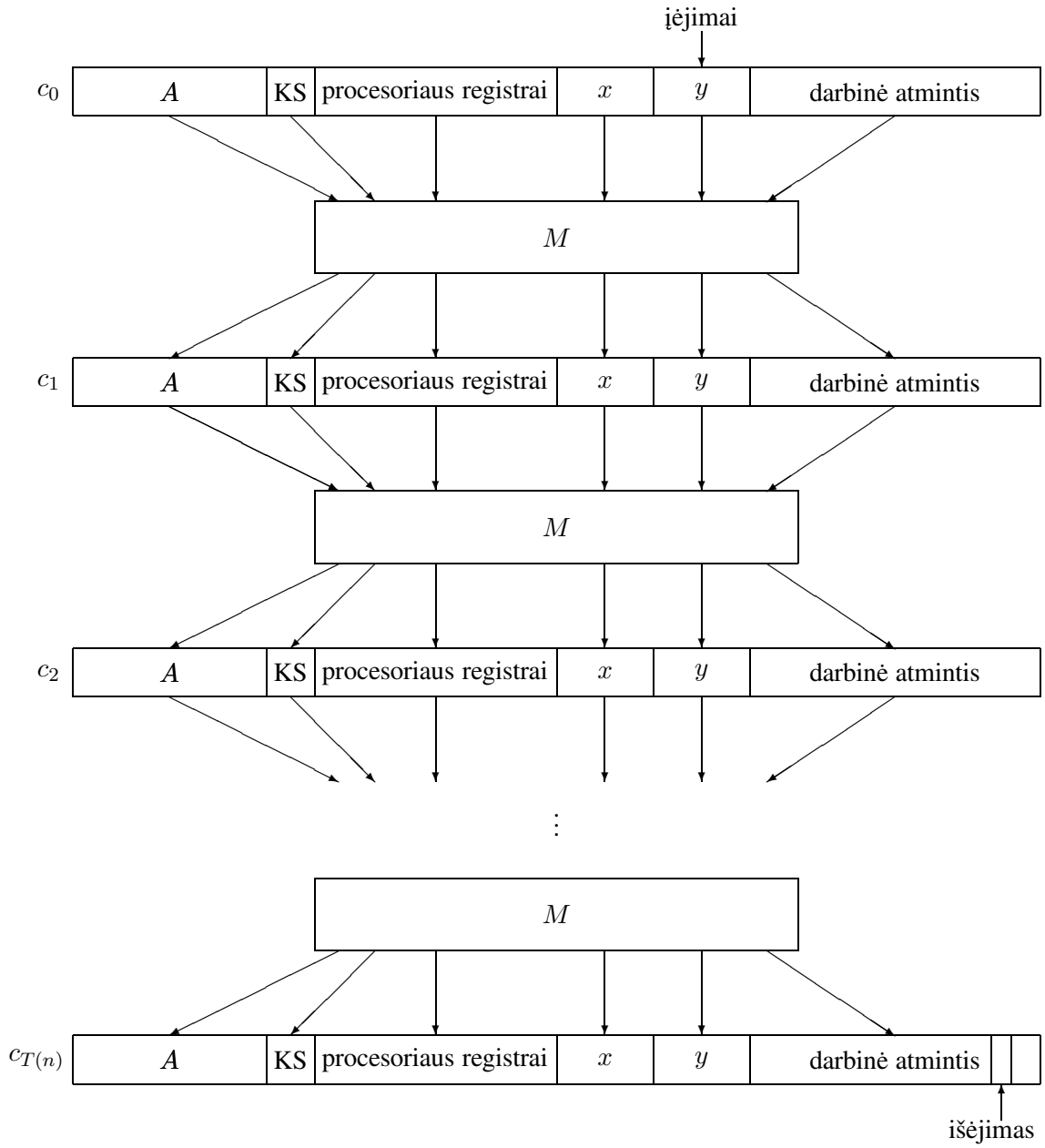
(2) uždavinys CIRCUIT-SAT yra NP-sunkus.

(1) Liudijimu bus schemos S įėjimų reikšmių rinkinys $y = (\alpha_1, \ldots, \alpha_n) \in \{0,1\}^n$, kuriam schemos S išėjimo reikšmė bus lygi 1. Liudijimo ilgis yra ne didesnis už schemos kodo ilgį $|\langle S\rangle|$. Kadangi kiekvienas funkcinis schemos elementas turi ne daugiau kaip $|\langle S\rangle|$ įėjimų, o pačių elementų taip pat yra ne daugiau kaip $|\langle S\rangle|$, tai per polinominį žingsnių skaičių galime apskaičiuoti kiekvieno funkcinio elemento bei schemos išėjimo reikšmę. Taigi, egzistuoja polinominio sudėtingumo algoritmas $A(x, y)$, toks, kad $\langle S\rangle \in$ CIRCUIT-SAT $\leftrightarrow \exists y$: $A(\langle S\rangle, y) = 1$ ir liudijimo y ilgis yra polinominis.

(2) Tegu L — bet kuri kalba iš klasės NP. Įrodysime, kad $L \leq_{\mathrm{p}}$ CIRCUIT-SAT. Konstruosime polinominio sudėtingumo algoritmą F, kuris bet kuriam žodžiui $x \in \{0,1\}^*$ priskirs schemą $S = F(x)$ tokią, kad $x \in L$ tada ir tik tada, kai S yra išpildoma, t.y., $\langle S\rangle \in$ CIRCUIT-SAT.

Kadangi $L \in$ NP, tai egzistuoja polinominis algoritmas $A(x, y)$, patikrinantis kalbą L per polinominį laiką. Galime laikyti, kad algoritmas A yra programa, parašyta žemo lygio kalba, naudojančia atminties adresus. Tada kompiuterio atmintyje programa A yra saugoma kaip komandų sąrašas, kur kiekvieną komandą sudaro operacijos kodas, operandų adresai kompiuterio atmintyje ir adresas, kur reikia įrašyti operacijos rezultatą. Specialioje atminties vietoje laikomas *komandų skaitiklis* KS, saugantis adresą komandos, kurią reikės vykdyti. Vykdant programą, šis adresas nuolat keičiasi, nurodydamas arba sekančią komandą, stovinčią po jau įvykdytos, arba kurią nors kitą komandą, jei buvo vykdoma sąlyginė ar ciklo komanda.

Laikysime, kad be vykdomos programos ir komandų skaitiklio kompiuterio atmintį dar sudaro procesoriaus registrai, pradiniai programos duomenys ir darbinė atmintis (žr. 4.6 pav.). Fiksuotą atminties būseną (t.y., atminties ląstelių (bitų) reikšmes tam tikru laiko momentu) vadinsime *konfigūracija*. Vienos komandos vykdymą atitinka ankstesnės konfigūracijos atvaizdavimas į naują konfigūraciją, kurį atlieka kompiuterio procesorius. Kadangi bet kuri konfigūracija yra nuliukų ir vienetukų rinkinys, tai šį atvaizdavimą galima realizuoti Būlio schema M. Kadangi pagal algoritmų savybes kiekviena komanda turi būti elementari, tai visada galima išsirinkti tokį komandų rinkinį, kad vienai komandai įvykdyti pakaks polinominio dydžio Būlio schemos (priklausomai nuo konfigūracijos ilgio).



Pav. 4.6: Kompiuterio konfigūracijų seka, atitinkanti algoritmo $A(x, y)$ darbą. Konfigūracija c_i vaizduoja kompiuterio būseną po i-ojo algoritmo A žingsnio. Pradinėje konfigūracijoje c_0 visų bitų, išskyrus liudijimo lauką y, reikšmės yra fiksuotos. Kiekvieną konfigūraciją į sekančią perdirba Būlio schema M. Jungtinės schemos išėjimas yra vienas iš darbinės atminties bitų.

Pažymime $T(n)$ algoritmo A žingsnių skaičių blogiausiu atveju duomenims ilgio n. Tegu $k \geq 1$ yra tokia konstanta, kad $T(n) = \mathrm{O}(n^k)$ ir liudijimo y ilgis taip pat yra $\mathrm{O}(n^k)$. Taip visada galima pasirinkti, nes algoritmo A sudėtingumas yra polinominis jo įėjimo ilgio atžvilgiu, o jo įėjimo ilgis yra $n + |y|$, kur $|y|$ savo ruožtu yra polinomas n atžvilgiu.

Dabar algoritmo A darbą galime vaizduoti $T(n)$ konfigūracijų seka $c_0, c_1, \ldots, c_{T(n)}$, kur pradinę konfigūraciją c_0 sudaro programa A, komandų skaitiklis KS, procesoriaus registrai, pradiniai duomenys x ir y bei darbinė atmintis. Konfigūraciją c_i padavus į schemos M įėjimus, jos išėjimuose gauname konfigūraciją c_{i+1} ($i = 0, 1, \ldots, T(n) - 1$).

Aprašysime algoritmą F, kuris pagal duotą x konstruoja schemą S tokią, kad S būtų išpildoma tada ir tik tada, kai egzistuoja liudijimas y toks, kad $A(x, y) = 1$. Algoritmas F pirmiausia apskaičiuoja $n = |x|$ ir konstruoja Būlio schemą S', sudarytą iš $T(n)$ schemos M kopijų. Šios schemos įėjimas bus pradinė skaičiavimo $A(x, y)$ konfigūracija, o išėjimas bus konfigūracija $c_{T(n)}$. Dabar schemą S nesunku gauti iš schemos S'. Pirmiausia schemos S' įėjimus, atitinkančius programą A, komandų skaitiklį, procesoriaus registrus, pradinę darbinės atminties konfigūraciją ir įėjimą x, fiksuojame, t.y., prijungiame tiesiai prie žinomų jų reikšmių. Taigi, laisvi lieka tik schemos įėjimai, atitinkantys liudijimą y. Antra, visus schemos S' išėjimus ignoruojame, išskyrus vieną, kuriame gaunmas algoritmo $A(x, y)$ darbo rezultatas. Šis išėjimas ir bus konstruojamos schemos S išėjimas.

Lieka įrodyti, kad:

(a) S yra išpildoma tada ir tik tada, kai egzistuoja poliniminio ilgio liudijimas y toks, kad $A(x, y) = 1$ ir

(b) algoritmas F yra polinominio sudėtingumo $n = |x|$ atžvilgiu.

(a) Tarkime, kad egzistuoja polinominio ilgio liudijimas y toks, kad $A(x, y) = 1$. Pateikę šias y reikšmes schemos S įėjimuose, gausime, kad jos išėjimas yra lygus 1 (nes sutampa su $A(x, y)$ išėjimu), taigi schema S išpildoma. Ir atvirkščiai, jei S išpildoma, tai atsiras įėjimas y toks, kad $S(y) = 1$, o tai reikš, kad ir $A(x, y) = 1$. Kadangi schemos S įėjimams išskirta lygiai $\mathrm{O}(n^k)$ bitų, tai liudijimo ilgis bus polinominis $|x|$ atžvilgiu.

(b) Įrodysime, kad schema S yra polinominio dydžio ir algoritmas F yra polinominio sudėtingumo $n = |x|$ atžvilgiu. Pirmiausia įrodysime, kad kiekviena konfigūracija c_i yra polinominio dydžio $n = |x|$ atžvilgiu. Programos dydis nepriklauso nuo x ir yra konstanta, kaip ir komandų skaitikliui bei procesoriaus registrams skirta atmintis. Įėjimo x dydis yra n, o liudijimo y dydis yra $\mathrm{O}(n^k)$. Kadangi algoritmas A daro ne daugiau, kaip $\mathrm{O}(n^k)$ žingsnių, tai jam reikalingos darbinės atminties dydis taip pat yra polinominis n atžvilgiu. Taigi, kiekviena konfigūracija yra polinominio dydžio n atžvilgiu, o viena schema M taip pat yra polinominio dydžio konfigūracijos dydžio atžvilgiu. Kadangi tokių schemų M skaičius $T(n)$ taip pat yra polinominis n atžvilgiu, tai ir schema S bus polinominio dydžio n atžvilgiu. Akivaizdu, kad ir ją konstruojantis algoritmas F bus polinominio sudėtingumo. $\square$

4.6 Kiti NP-pilni uždaviniai

Nagrinėsime Būlio formules virš bazės $B = \{\&, \vee, \neg, \to, \leftrightarrow\}$. Formulė $F(x_1, \ldots, x_n)$ *išpildoma*, jei egzistuoja kintamųjų reikšmių rinkinys $(\alpha_1, \ldots, \alpha_n) \in \{0,1\}^n$, paverčiantis formulę teisinga: $F(\alpha_1, \ldots, \alpha_n) = 1$. Pavyzdžiui, Būlio formulė $F(x_1, x_2, x_3) = \neg(x_1 \to x_2)\&\neg x_3$ yra išpildoma, nes $F(1,0,0) = 1$. Pažymėkime

$$\mathrm{SAT} = \{\langle F\rangle\text{: } F \text{ — išpildoma Būlio formulė}\}.$$

Teorema 4.6.1. $\mathrm{SAT} \in \mathrm{NPC}$.

Dabar nagrinėsime specialaus pavidalo Būlio formules virš bazės $B_0 = \{\&, \vee, \neg\}$. *Litera* vadinsime kintamąjį arba jo neiginį, t.y., $p, \neg p$ yra literos. *Elementariąja disjunkcija* vadiname skirtingų literų disjunkciją. *Normaliąja konjunkcine forma* (CNF, angl.) vadiname bet kokią skirtingų elementarių disjunkcijų konjunkciją:

$$F = D_1 \,\&\, D_2 \& \cdots \& \, D_m, \quad \text{kur } D_i = x_{i_1}^{\sigma_1} \vee \cdots \vee x_{i_k}^{\sigma_k} \ (x_{i_j} \neq x_{i_l}) \text{ ir } D_p \neq D_r.$$

Pagaliau normaliąją konjunkcinę formą vadinsime *3-normaliąja konjunkcine forma* (3-CNF, angl.), jei kiekviena jos elementarioji disjunkcija yra sudaryta iš lygiai 3 skirtingų literų. Pažymėkime

$$\text{3-CNF-SAT} = \{\langle F\rangle\text{: } F \text{ yra išpildoma 3-normalioji konjunkcinė forma}\}.$$

Pavyzdžiui, jei $F(p,q,r) = (p \vee q \vee r)\,\&\,(\neg p \vee \neg q \vee \neg r)\,\&\,(p \vee \neg q \vee \neg r)\,\&\,(\neg p \vee q \vee \neg r)$, tai $\langle F\rangle \in \text{3-CNF-SAT}$, nes $F(1,1,0) = 1$.

Teorema 4.6.2. $\text{3-CNF-SAT} \in \mathrm{NPC}$.

Paprastą (t.y., be kartotinių briaunų ir kilpų) grafą vadiname *pilnu*, jei bet kurios dvi jo viršūnės yra sujungtos briauna. Pilni grafai, turintys n viršūnių yra žymimi K_n. Kadangi kiekviena briauna turi du galus, tai pilnas grafas K_n turi $n(n-1)/2$ briaunų.

Grafo *klika* vadiname bet kokį pilną jo pografį. Klikos dydžiu vadiname jos viršūnių skaičių. Pavyzdžiui, bet kuris grafas, turintis n viršūnių ir m briaunų turi lygiai n klikų dydžio 1 ir m klikų dydžio 2. Optimizavimo uždavinyje MAX-CLIQUE reikia duotame grafe surasti maksimalaus dydžio kliką. Jį atitinkantis egzistavimo uždavinys CLIQUE klausia, ar duotame grafe egzistuoja bent viena klika dydžio $k \in \mathbb{N}$:

$$\mathrm{CLIQUE} = \{\langle G, k\rangle\text{: grafe } G \text{ egzistuoja klika dydžio } k\}.$$

Teorema 4.6.3. $\mathrm{CLIQUE} \in \mathrm{NPC}$.

Grafo $G = (V, E)$ *viršūniniu denginiu* vadiname jo viršūnių poaibį $V' \subseteq V$ tokį, kad kiekvienai briaunai $(u, v) \in E$ turime $u \in V'$ arba $v \in V'$ (arba abu atvejai teisingi). Taigi, viršūnės

iš aibės V' "uždengia" bent vieną kiekvienos briaunos galą. Viršūninio denginio dydžiu vadiname jo viršūnių skaičių. Optimizavimo uždavinyje $\mathrm{MIN\text{-}VERTEX\text{-}COVER}$ reikia rasti mažiausią duotojo grafo viršūninį denginį. Jį atitinkantis egzistavimo uždavinys $\mathrm{VERTEX\text{-}COVER}$ klausia, ar egzistuoja duotojo grafo viršūninis denginys dydžio $k \in \mathbb{N}$:

$$\mathrm{VERTEX\text{-}COVER} = \{\langle G, k\rangle\text{: grafas } G \text{ turi viršūninį denginį dydžio } k\}.$$

Teorema 4.6.4. $\mathrm{VERTEX\text{-}COVER} \in \mathrm{NPC}$.