Teoria obliczeń i złożoności obliczeniowej

Michał Dobranowski

semestr zimowy 2024
v0.4

Poniższy skrypt zawiera materiał częściowo obejmujący wykłady z przedmiotu *Teoria obliczeń i złożoności obliczeniowej* prowadzone przez prof. dr. hab. inż. Piotra Faliszewskiego na trzecim roku Informatyki na AGH. W przeciwieństwie do wykładów, które teorię obliczeń i teorię złożoności obliczeniowej przedstawiały raczej w równym stopniu, niniejszy skrypt zdecydowanie skupia się na tej drugiej, rozszerzając materiał o przykłady, uwagi, dowody oraz dodatkowe twierdzenia.

Spis treści

1. Maszyna Turinga i enumerator

Definicja 1.1. Maszyna Turinga to krotka

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_Y, q_N, \square),$$

gdzie

- Q to skończony zbiór stanów,
- Σ to skończony alfabet wejściowy,
- Γ to skończony alfabet taśmy taki, że $\Sigma \subset \Gamma$ oraz $\square \in \Gamma \setminus \Sigma$,
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{\hookleftarrow, \hookrightarrow\}$ to funkcja przejścia,
- $q_0 \in Q$ to stan początkowy,
- $q_Y \in Q$ to stan akceptujący,
- $q_N \in Q$ to stan odrzucający.

Formalizację działania maszyny Turinga pozostawiamy jako ćwiczenie dla Czytelnika oraz odsyłamy do podręczników [3, 5].

Maszyna Turinga (będziemy ją zwali po prostu „maszyną") rozwiązuje pewien problem decyzyjny, czyli dla danego słowa x z języka Σ^* stwierdza, czy x należy do języka $L \subseteq \Sigma^*$. Dlatego też często będziemy używać pewnych określeń języka również dla odpowiadającego mu problemu decyzyjnego.

Maszyna M *akceptuje* język L, jeśli dla każdego $x \in \Sigma^*$, M akceptuje x wtedy i tylko wtedy, gdy $x \in L$. Przez $L(M)$ oznaczamy język akceptowany przez maszynę M.

Maszyna M *rozstrzyga* o języku L, jeżeli go akceptuje i kończy działania dla każdego $x \in \Sigma^*$. Jeśli taka maszyna istnieje, to mówimy, że język L jest *rozstrzygalny*. Również problem decyzyjny odpowiadający językowi L nazywamy rozstrzygalnym.

1.1. Klasy RE, coRE i R

Definicja 1.2 (klasa RE)**.** Klasę języków rekurencyjnie przeliczalnych (*recursively enumerable*) definiujemy jako

$$\mathsf{RE} = \{L \mid \exists M : M \text{ akceptuje język } L\}.$$

Klasa ta to klasa języków typu 0 w hierarchii Chomsky'ego. Dowód tego faktu niech pozostanie ćwiczeniem dla Czytelnika.

Definicja 1.3 (klasa R)**.** Klasę języków rozstrzygalnych (*recursive, decidable*) definiujemy jako

$$\mathsf{R} = \{L \mid \exists M : M \text{ rozstrzyga o języku } L\}.$$

Jeśli język jest rozstrzygalny, to mówimy, że istnieje *algorytm*, który stwierdza, czy dane słowo należy do tego języka.

Definicja 1.4 (klasa coRE)**.** Klasę dopełnień języków klasy RE definiujemy jako

$$\mathsf{coRE} = \{L \mid \overline{L} \in \mathsf{RE}\}.$$

Lemat 1.5

Jeśli $L_1, L_2 \in \mathsf{RE}$, to $L_1 \cap L_2 \in \mathsf{RE}$.

Dowód. Niech M_1 i M_2 będą maszynami Turinga akceptującymi odpowiednio języki L_1 i L_2. Skonstruujmy maszynę $M(x)$ akceptującą $L_1 \cap L_2$ w następujący sposób:

if $M_1(x) \wedge M_2(x)$ **then**
    akceptuj

Wówczas M akceptuje $L_1 \cap L_2$. □

Lemat 1.6

Jeśli $L_1, L_2 \in \mathsf{RE}$, to $L_1 \cup L_2 \in \mathsf{RE}$.

Dowód. Niech M_1 i M_2 będą maszynami Turinga akceptującymi odpowiednio języki L_1 i L_2. Skonstruujmy maszynę $M(x)$ akceptującą $L_1 \cup L_2$ w następujący sposób:

for $k = 1, 2, \ldots$ **do**
    if M_1 akceptuje x w k krokach **then**
        akceptuj
    if M_2 akceptuje x w k krokach **then**
        akceptuj

Wówczas M akceptuje $L_1 \cup L_2$. □

Twierdzenie 1.7

Zachodzi równość

$$\mathsf{R} = \mathsf{RE} \cap \mathsf{coRE}.$$

Dowód. Weźmy język $L \in \mathsf{R}$. Istnieje maszyna Turinga M, która rozstrzyga o L, a więc $L = L(M)$, więc $L \in \mathsf{RE}$. Ponadto, dopełnienie $\overline{L}$ jest rozstrzygalne, bo wystarczy zmienić stan akceptujący na odrzucający i na odwrót. Zatem $\overline{L} \in \mathsf{R}$, więc $L \in \mathsf{coRE}$. Tym samym udowodniliśmy, że $\mathsf{R} \subseteq \mathsf{RE} \cap \mathsf{coRE}$.

Teraz weźmy język L taki, że $L \in \mathsf{RE}$ oraz $L \in \mathsf{coRE}$. Wtedy istnieje maszyna Turinga M, która akceptuje L oraz maszyna Turinga N, która akceptuje $\overline{L}$. Rozumowaniem podobnym do dowodu lematu 1.6 skonstruujmy maszynę $M'(x)$, która akceptuje L i odrzuca $\overline{L}$, a więc rozstrzyga o L. Tym samym udowodniliśmy, że $\mathsf{R} \supseteq \mathsf{RE} \cap \mathsf{coRE}$. □

1.2. Wielotaśmowa maszyna Turinga

Niestety autorowi zabrakło motywacji do opisania wielotaśmowych maszyn Turinga oraz innych, pochodnych modeli obliczeń. Zdecydowanie jednak warto coś o nich wiedzieć, dlatego odsyłamy Czytelnika do literatury [1, 5].

1.3. Enumerator

Definicja 1.8. Enumerator to 2-taśmowa maszyna Turinga, która nie przyjmuje żadnego wejścia, a zamiast stanu akceptującego i odrzucającego ma stan wyliczający. Jeśli ten stan jest osiągnięty, to słowo znajdujące się na drugiej taśmie jest uznawane za „wydrukowane”, a taśma jest czyszczona.

Będziemy mówić, że język *akceptowany* przez enumerator E to język, który jest zbiorem słów wydrukowanych przez E.

Twierdzenie 1.9

Język L jest akceptowany przez enumerator wtedy i tylko wtedy, gdy $L \in \mathsf{RE}$.

Dowód wystarczalności. Załóżmy, że istnieje enumerator E, który wylicza słowa języka L. Możemy skonstruować maszynę $M(x)$:

for $y \in E$ **do**
  if $x = y$ **then**
    akceptuj

która akceptuje słowo x wtedy i tylko wtedy, gdy x jest wyliczane przez enumerator E. Zatem $L \in \mathsf{RE}$. □

Dowód konieczności. Niech $L \in \mathsf{RE}$. Istnieje więc maszyna M, która akceptuje L. Możemy stworzyć enumerator E:

for $k = 1, 2, \ldots$ **do**
  for $x \in \Sigma^{\leqslant k}$ **do**
    if M akceptuje x w k krokach **then**
      wylicz x

który wylicza słowa języka L (i nie zawiesi się, gdy $M(x)$ nie kończy działania dla pewnego x). □

Twierdzenie to wyjaśnia nazwę klasy RE — języki z tej klasy są rekurencyjnie przeliczalne, to znaczy, że można je wyliczyć za pomocą enumeratora.

Twierdzenie 1.10

Istnieje enumerator, który wylicza słowa języka L w kolejności ich niemalejących długości wtedy i tylko wtedy, gdy $L \in \mathsf{R}$.

Dowód wystarczalności. Załóżmy, że istnieje enumerator E, który wylicza słowa języka L w kolejności ich niemalejących długości. Możemy stworzyć maszynę $M(x)$:

for $y \in E$ **do**
  if $x = y$ **then**
    akceptuj
  if $|y| > |x|$ **then**
    odrzuć

która rozstrzyga o języku L (na pewno zakończy swoje działanie, ponieważ słów o długości nie większej niż $|x|$ jest skończenie wiele). Z tego wynika, że $L \in \mathsf{R}$. □

Dowód konieczności. Załóżmy, że $L \in \mathsf{R}$. Istnieje więc maszyna M, która rozstrzyga o L. Możemy skonstruować enumerator E:

for $k = 0, 1, 2, \ldots$ **do**
  for $x \in \Sigma^k$ **do**
    if $M(x)$ **then**
      wylicz x

który spełnia tezę. □

2. Uniwersalna maszyna Turinga, języki nierozstrzygalne

Definicja 2.1. Uniwersalna maszyna Turinga to maszyna Turinga U, która dla każdej maszyny Turinga M i słowa x symuluje działanie M na x.

Maszynę M z powyższej definicji możemy zakodować jako słowo w_M nad alfabetem $\{0,1\}$, kodując kolejne elementy krotki M. Takie kodowanie będziemy oznaczać jako $\langle M \rangle$. Ma to o tyle istotne znaczenie, że z takiego zapisu wynika, że zbiór kodów maszyn Turinga jest przeliczalny, w przeciwieństwie do zbioru języków, który jest nieprzeliczalny. Stąd wynika, że istnieją języki, których nie akceptuje żadna maszyna Turinga.

Powyższy fakt wykazuje się podobnie, jak to, że moc zbioru liczb rzeczywistych z przedziału $[0,1]$ jest większa niż moc zbioru liczb naturalnych, to znaczy metodą przekątniową Cantora.

Uwaga 2.2 (na temat przeliczalności domknięcia Kleene'ego zbiorów)

Alfabet Σ dowolnego języka L jest oczywiście skończony. Z tego wynika, że Σ^* jest przeliczalny (możemy wprowadzić porządek znany z $\mathbb{N}$, czyli języka nad cyframi), a więc każdy język $L \subseteq \Sigma^*$ jest przeliczalny.

Nawet jeśli weźmiemy zbiór A, który jest nieskończony, ale przeliczalny, to zbiór A^* również jest przeliczalny, co dociekliwy Czytelnik może udowodnić.

2.1. Problem stopu

Problem stopu (*halting problem*) to problem decyzyjny polegający na stwierdzeniu, czy dany program zatrzymuje się dla danego wejścia. Jest on fundamentalnie nierozstrzygalny, co udowodnili niezależnie od siebie Alonzo Church (za pomocą rachunku lambda) oraz jego student, Alan Turing (za pomocą omawianej tutaj teorii).

Twierdzenie 2.3

Problem stopu jest nierozstrzygalny.

Dowód. Niech H będzie językiem zdefiniowanym jako

$$H = \{\langle M, x \rangle \mid M \text{ kończy działanie dla } x\}.$$

Zakładamy nie wprost, że język ten jest rozstrzygalny, a więc istnieje maszyna Turinga M_H, która o nim rozstrzyga. Skonstruujmy maszynę D, przyjmującą na wejściu maszynę M:

if M_H akceptuje $\langle M, M \rangle$ **then**
    zapętl się
else
    akceptuj

Wówczas D akceptuje D wtedy i tylko wtedy, gdy M_H nie akceptuje $\langle D, D \rangle$, czyli gdy D nie kończy działania dla D, co prowadzi do sprzeczności. □

Z twierdzenia 2.3 można wyciągnąć bardzo wiele wniosków na temat nierozstrzygalności innych języków.

Twierdzenie 2.4

Każdy nieskończony język klasy RE posiada nieskończony podzbiór klasy R.

Dowód. Niech L będzie nieskończonym językiem klasy RE. Wówczas, na mocy twierdzenia 1.9, istnieje enumerator E, który wylicza słowa języka L w pewnej kolejności, weźmy $w_1, w_2, \ldots$. Zdefiniujmy zbiór indeksów I taki, że

$$I = \left\{ i \mid \forall_{j<i} \left(|w_j| < |w_i| \right) \right\},$$

a więc takich, że słowo w_i jest dłuższe, niż każde poprzednie. Wówczas zbiór I jest nieskończony, a słowa w_i dla $i \in I$ są wyliczane w kolejności niemalejących długości, więc, z twierdzenia 1.10, $\{w_i \mid i \in I\} \in \mathsf{R}$. □

2.2. Redukcje, trudność i zupełność, twierdzenie Rice'a

Definicja 2.5. Funkcja obliczalna to funkcja $f : \Sigma^* \to \Sigma^*$, dla której istnieje maszyna Turinga, która dla każdego $x \in \Sigma^*$ kończy działanie i zwraca $f(x)$.

Definicja 2.6 (redukcja *many-one*)**.** Język A *redukuje się* do języka B, co zapisujemy jako $A \leqslant_m B$, jeśli istnieje funkcja obliczalna f taka, że dla każdego $x \in \Sigma^*$ zachodzi

$$x \in A \iff f(x) \in B.$$

Można łatwo dowieść, że relacja $\leqslant_m$ jest relacją przechodnią.

Fakt 2.7. Jeśli $A \leqslant_m B$, prawdą jest:

$$B \in \mathsf{R} \implies A \in \mathsf{R},$$
$$B \in \mathsf{RE} \implies A \in \mathsf{RE},$$
$$\overline{A} \leqslant_m \overline{B}.$$

Będziemy mówić, że język L jest R-*trudny*, jeśli dla każdego $A \in \mathsf{R}$ zachodzi $A \leqslant_m L$. Jeśli dodatkowo $L \in \mathsf{R}$, to mówimy, że L jest R-*zupełny*. Analogiczne określenia stosujemy dla klas RE i coRE.

Rodzinę $\mathcal{L}$ języków będziemy nazywać *własnością* i mówić, że język L ma własność $\mathcal{L}$, jeśli $L \in \mathcal{L}$.

Twierdzenie 2.8 (Rice'a)

Niech $\mathcal{L}$ będzie nietrywialną[a] właściwością języków klasy RE. Wówczas język

$$B_{\mathcal{L}} = \{\langle M \rangle \mid L(M) \in \mathcal{L}\}$$

jest nierozstrzygalny.

[a] to znaczy różną od $\emptyset$ oraz RE

Dowód. Bez straty ogólności możemy założyć, że $\emptyset \notin \mathcal{L}$ (jeśli tak nie jest, to możemy operować na $\overline{\mathcal{L}}$). Weźmy pewien język $L \in \mathcal{L}$, który jest akceptowany przez maszynę M_L. Pokażemy, że problem stopu H redukuje się do $B_{\mathcal{L}}$. Tworzymy redukcję $f(\langle M, x \rangle) = \langle N \rangle$, gdzie maszyna $N(y)$ ma następujący kod:

if M kończy działanie dla x **then**
  if M_L akceptuje y **then**
    akceptuj

Jeśli $M(x)$ kończy działanie, to $L(N) = L \neq \emptyset$; w przeciwnym wypadku $L(N) = \emptyset$. Zatem

$$\langle M, x \rangle \in H \iff \langle N \rangle \in B_{\mathcal{L}} \iff L(N) \in \mathcal{L}.$$

□

Twierdzenie 2.9

Problemu stopu jest RE-zupełny.

Dowód. Oczywiście problem stopu $H \in \mathsf{RE}$, co wynika z definicji uniwersalnej maszyny Turinga. Weźmy dowolny język $L \in \mathsf{RE}$ i pokażmy, że $L \leqslant_m H$. Oznaczmy jako M_L maszynę, która go akceptuje i przyjmijmy, że nigdy nie odrzuca słowa (zamiast tego zawsze się zapętla). Skonstruujmy funkcję obliczalną

$$f(x) = \langle M_L, x \rangle .$$

Wówczas $x \in L \iff \langle M_L, x \rangle \in H$, a więc $L \leqslant_m H$. □

Twierdzenie 2.10

Każdy nietrywialny[a] język klasy R jest R-zupełny.

[a] to znaczy różny od $\emptyset$ oraz Σ^*

Dowód. Niech A, B będą dowolnymi nietrywialnymi rozstrzygalnymi językami oraz niech $x_Y \in B$, $x_N \notin B$. Możemy stworzyć funkcję obliczalną

$$f(z) = \begin{cases} x_Y, & \text{jeśli } z \in A, \\ x_N, & \text{jeśli } z \notin A, \end{cases}$$

która jest zwracana przez maszynę

if M_A akceptuje z **then**
    zwróć x_Y
else
    zwróć x_N

która zawsze kończy swoje działanie, a więc spełnia wszystkie założenia definicji 2.6. Zatem $A \leqslant_m B$, a więc B jest R-zupełny. □

3. Hierarchia arytmetyczna

Znamy już pewne klasy języków (czy też problemów decyzyjnych), mianowicie RE, coRE oraz ich część wspólną R. W tej sekcji, bazując na twierdzeniach 3.1, 3.2, wprowadzimy kolejne klasy języków, które są bardziej ogólne niż wyżej wymienione.

3.1. Charakterystyki klas RE i coRE

Twierdzenie 3.1 (Charakterystyka klasy RE)

Język A należy do klasy RE wtedy i tylko wtedy, gdy istnieje język $B \in \mathsf{R}$ taki, że

$$A = \{x \in \Sigma^* \mid (\exists y \in \Sigma^*)\, (\langle x, y \rangle \in B)\} .$$

Dowód. Jeśli $A \in \mathsf{RE}$, to istnieje maszyna Turinga M_A, która akceptuje A. Wtedy

$$\begin{aligned} A &= \{x \mid M_A \text{ akceptuje } x\} \\ &= \{x \mid (\exists k \in \mathbb{N})\,(M_A \text{ akceptuje } x \text{ w } k \text{ krokach})\} \\ &= \{x \mid (\exists y \in \Sigma^*)\,(M_A \text{ akceptuje } x \text{ w } |y| \text{ krokach})\} \\ &= \{x \mid (\exists y \in \Sigma^*)\,(\langle x, y\rangle \in B)\}. \end{aligned}$$

□

Twierdzenie 3.2 (Charakterystyka klasy coRE)
Język A należy do klasy coRE wtedy i tylko wtedy, gdy istnieje język $C \in \mathsf{R}$ taki, że

$$A = \{x \in \Sigma^* \mid (\forall y \in \Sigma^*)\,(\langle x, y\rangle \in B)\}\,.$$

Dowód. Na podstawie twierdzenia 3.1 wiemy, że istnieje język B taki, że

$$\overline{A} = \{x \in \Sigma^* \mid (\exists y \in \Sigma^*)\,(\langle x, y\rangle \in B)\}\,.$$

Wówczas

$$\begin{aligned} A &= \{x \in \Sigma^* \mid (\neg\exists y \in \Sigma^*)\,(\langle x, y\rangle \in B)\} \\ &= \{x \in \Sigma^* \mid (\forall y \in \Sigma^*)\,(\langle x, y\rangle \notin B)\} \\ &= \left\{x \in \Sigma^* \mid (\forall y \in \Sigma^*)\left(\langle x, y\rangle \in \overline{B}\right)\right\}, \end{aligned}$$

co, po wzięciu $C = \overline{B}$, kończy dowód. □

3.2. Definicje klas hierarchii arytmetycznej

Definicja 3.3 (klasa Σ_i)**.** Język A należy do klasy Σ_i, gdy istnieje język $B \in \mathsf{R}$ taki, że

$$A = \left\{ x \;\middle|\; \underbrace{(\exists y_1 \in \Sigma^*)\,(\forall y_2 \in \Sigma^*)\,(\exists y_3 \in \Sigma^*)\cdots}_{i \text{ naprzemiennych kwantyfikatorów}} (\langle x, y_1, y_2, \ldots, y_i\rangle \in B) \right\}.$$

Definicja 3.4 (klasa Π_i)**.** Język A należy do klasy Π_i, gdy istnieje język $C \in \mathsf{R}$ taki, że

$$A = \left\{ x \;\middle|\; \underbrace{(\forall y_1 \in \Sigma^*)\,(\exists y_2 \in \Sigma^*)\,(\forall y_3 \in \Sigma^*)\cdots}_{i \text{ naprzemiennych kwantyfikatorów}} (\langle x, y_1, y_2, \ldots, y_i\rangle \in C) \right\}.$$

Z definicji wynika, że $\Sigma_0 = \Pi_0 = \mathsf{R}$, Ponadto, prostym wnioskiem z twierdzeń 3.1 i 3.2 są równości $\Sigma_1 = \mathsf{RE}$ oraz $\Pi_1 = \mathsf{coRE}$. Uważny Czytelnik na pewno zauważy również, że nie tylko

$$\Sigma_0 \subset \Sigma_1 \subset \Sigma_2 \subset \cdots$$

oraz

$$\Pi_0 \subset \Pi_1 \subset \Pi_2 \subset \cdots,$$

ale też

$$\Sigma_i \subset \Pi_{i+1} \quad \text{oraz} \quad \Pi_i \subset \Sigma_{i+1}.$$

Przykład 3.5

Wykazać, że język

$$L = \left\{ \langle M \rangle \mid \overline{L(M)} \text{ jest skończony} \right\}$$

należy do klasy Σ_3.

Rozwiązanie. Mamy

$$\begin{aligned} L &= \left\{ \langle M \rangle \mid \overline{L(M)} \text{ jest skończony} \right\} \\ &= \{ \langle M \rangle \mid (\exists t \in \mathbb{N})\, (\forall x \in \Sigma^*)\, (|x| > t \implies M(x) \text{ akceptuje}) \} \\ &= \{ \langle M \rangle \mid (\exists t \in \mathbb{N})\, (\forall x \in \Sigma^*)\, (|x| \leqslant t \vee M(x) \text{ akceptuje}) \} \\ &= \{ \langle M \rangle \mid (\exists t \in \mathbb{N})\, (\forall x \in \Sigma^*)\, (|x| \leqslant t \vee (\exists k \in \mathbb{N})\, (M(x) \text{ akceptuje w } k \text{ krokach})) \} \\ &= \{ \langle M \rangle \mid (\exists t \in \mathbb{N})\, (\forall x \in \Sigma^*)\, (\exists k \in \mathbb{N})\, (|x| \leqslant t \vee M(x) \text{ akceptuje w } k \text{ krokach}) \}, \end{aligned}$$

więc $L \in \Sigma_3$. □

Uwaga

Powyższy przykład można uogólnić do pewnego sposobu szukania górnego ograniczenia na dokładną klasą w hierarchii arytmetycznej, zwanego algorytmem Tarskiego-Kuratowskiego. Wystarczy sprowadzić opis danego języka do przedrostkowej postaci normalnej, a następnie policzyć kwantyfikatory i stwierdzić, który z nich występuje jako pierwszy.

W trakcie tego procesu warto pamiętać o kilku rzeczach:

1. Możemy używać zmiennych naturalnych zamiast słów ze zbioru Σ^* (tak zrobiliśmy na przykład w powyższym dowodzie), ponieważ zamiast $\exists t \in \mathbb{N}$ możemy napisać $\exists t \in \Sigma^*$ i używać $|t|$ jako liczby naturalnej.
2. Możemy łączyć kwantyfikatory leżące obok siebie, na przykład $\forall x \in \Sigma^*\ \forall y \in \Sigma^*$ możemy zapisać jako $\forall \langle x, y \rangle \in \Sigma^*$. Oczywiście taki zapis jest mniej czytelny — chodzi tylko o to, żeby nie liczyć kwantyfikatorów podwójnie przy określaniu klasy.
3. Czasami możemy zmieniać kolejność kwantyfikatorów w taki sposób, aby uzyskać węższą klasę. Należy jednak uważać, ponieważ nie zawsze jest to możliwe.

Przykład 3.6

Określ klasę w hierarchii arytmetycznej języka

$$L = \{ \langle M \rangle \mid L(M) \text{ jest rozstrzygalny} \}.$$

Rozwiązanie.

$$\begin{aligned} L &= \{ \langle M \rangle \mid L(M) \in \mathsf{R} \} \\ &= \{ \langle M \rangle \mid \exists \text{ maszyna } M' \text{ rozstrzygająca o } L(M) \} \\ &= \left\{ \langle M \rangle \;\middle|\; (\exists \langle M' \rangle)\, (\forall x \in \Sigma^*)\, (\forall l \in \mathbb{N})\, (\exists k \in \mathbb{N}) \begin{array}{l} \text{maszyny } M', M \text{ akceptują } x \text{ w } k \text{ krokach} \\ \text{lub nie akceptują } x \text{ w } l \text{ krokach} \end{array} \right\} \end{aligned}$$

Język L należy więc do klasy Σ_3. □

4. Klasy złożoności obliczeniowej

Niech M będzie maszyną Turinga. *Złożoność czasowa* M to funkcja $f\colon \mathbb{N} \to \mathbb{N}$, gdzie $f(n)$ to maksymalna liczba kroków, jakie M wykonuje na wejściu długości n. *Złożoność*

pamięciowa M to funkcja $g\colon \mathbb{N} \to \mathbb{N}$, gdzie $g(n)$ to maksymalna liczba komórek taśmy, jakie M używa na wejściu długości n. Definiujemy klasy języków:

$$\mathsf{TIME}(t(n)) = \{L \mid \exists M \text{ rozstrzygająca o } L \text{ w czasie } \mathcal{O}(t(n))\}$$

oraz

$$\mathsf{SPACE}(s(n)) = \{L \mid \exists M \text{ rozstrzygająca o } L \text{ w pamięci } \mathcal{O}(s(n))\}.$$

Korzystając z tych definicji, możemy również zdefiniować klasę języków decyzyjnych, które są rozstrzygalne w czasie wielomianowym

$$\mathsf{P} = \bigcup_{k=1}^{\infty} \mathsf{TIME}\left(n^k\right),$$

klasę języków decyzyjnych, które są rozstrzygalne w czasie wykładniczym

$$\mathsf{EXP} = \bigcup_{k=1}^{\infty} \mathsf{TIME}\left(2^{n^k}\right)$$

oraz klasy języków, które są rozstrzygalne w pamięci wielomianowej i wykładniczej, odpowiednio

$$\mathsf{PSPACE} = \bigcup_{k=1}^{\infty} \mathsf{SPACE}\left(n^k\right),$$

$$\mathsf{EXPSPACE} = \bigcup_{k=1}^{\infty} \mathsf{SPACE}\left(2^{n^k}\right).$$

Oczywiste jest, że $\mathsf{P} \subseteq \mathsf{EXP}$ i $\mathsf{PSPACE} \subseteq \mathsf{EXPSPACE}$. Łatwo zauważyć również, że $\mathsf{P} \subseteq \mathsf{PSPACE}$ i $\mathsf{EXP} \subseteq \mathsf{EXPSPACE}$ (konsumowanie pamięci zajmuje czas). Możemy natomiast udowodnić nieco mniej oczywisty fakt.

Fakt 4.1. Zachodzi inkluzja $\mathsf{PSPACE} \subseteq \mathsf{EXP}$.

Dowód. Maszyna Turinga ma wykładniczo wiele konfiguracji taśmy w stosunku do długości tej taśmy. Nie istnieje więc problem, który można rozwiązać w pamięci wielomianowej, ale nie można go rozwiązać w czasie wykładniczym. □

4.1. Niedeterministyczna maszyna Turinga

Niedeterministyczna maszyna Turinga to uogólnienie maszyny Turinga, które pozwala na wiele możliwych stanów, w których może znaleźć się maszyna w danym momencie.

Definicja 4.2. Niedeterministyczna maszyna Turinga to krotka

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_Y, q_N, \square),$$

gdzie

- Q to skończony zbiór stanów,
- Σ to skończony alfabet wejściowy,
- Γ to skończony alfabet taśmy taki, że $\Sigma \subset \Gamma$ oraz $\square \in \Gamma \setminus \Sigma$,
- $\delta \subseteq (Q \times \Gamma) \times (Q \times \Gamma \times \{\hookleftarrow, \hookrightarrow\})$ to **relacja** przejścia,
- $q_0 \in Q$ to stan początkowy,
- $q_Y \in Q$ to stan akceptujący,
- $q_N \in Q$ to stan odrzucający.

Ścieżką obliczeń nazwiemy skończony ciąg stanów i symboli taśmy, który zaczyna się w stanie q_0 i jest osiągalny za pomocą relacji przejścia δ.

Niedeterministyczna maszyna Turinga akceptuje słowo x o długości n, jeśli istnieje ścieżka obliczeń, która kończy się w stanie q_Y. Język jest rozstrzygany przez niedeterministyczną maszynę Turinga M w czasie $f(n)$, jeśli dla każdego słowa x o długości n każda ścieżka obliczeń kończy się w czasie $f(n)$ oraz M akceptuje x wtedy i tylko wtedy, gdy $x \in L$.

Analogicznie do już zdefiniowanych klas złożoności obliczeniowej, definiujemy klasy języków decyzyjnych, które są rozstrzygalne przez niedeterministyczną maszynę Turinga; oznaczamy je jako NP, NPSPACE, NEXP i tym podobne.

4.2. Redukcje wielomianowe, jeszcze raz o trudności i zupełności

Definicja 4.3 (redukcja wielomianowa *many-one*)**.** Język A *redukuje się* do języka B w czasie wielomianowym, co zapisujemy jako $A \leqslant_m^p B$, jeśli istnieje funkcja f obliczalna w czasie wielomianowym taka, że dla każdego $x \in \Sigma^*$ zachodzi

$$x \in A \iff f(x) \in B.$$

Nasze definicje trudności i zupełności języków w klasach RE, coRE czy też szerszych Σ_i i Π_i nie mają zbyt dużego sensu w kontekście klas P, NP, PSPACE, EXP, NEXP itp. (z powodu twierdzenia 2.10). W tych klasach zamiast zwykłej redukcji funkcją obliczalną używamy redukcji wielomianowej, co odpowiednio zmienia definicje trudności i zupełności.

4.3. Problem SAT

Problem SAT (*satisfiability*) to problem spełnialności danej formuły logicznej, a więc stwierdzenia, czy istnieje takie wartościowania jej zmiennych, które sprawia, że formuła jest prawdziwa.

Często używana jest wersja problemu SAT, w której formuła jest w koniunkcyjnej postaci normalnej (CNF, *conjunctional normal form*), a więc jest koniunkcją klauzul, gdzie klauzula to alternatywa literałów, a literał to zmienna lub jej negacja. Przykładowo, formuła

$$(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2 \vee x_3 \vee x_4)$$

jest w postaci CNF.

Fakt 4.4 (algorytm Tseitina)**.** Każdą formułę logiczną można sprowadzić do postaci CNF w czasie wielomianowym, a rozmiar formuły wyjściowej będzie liniowy w stosunku do rozmiaru formuły wejściowej.

Twierdzenie 4.5 (Cooka-Levina)
Problem SAT jest NP-zupełny.

Dowód. Łatwo zaobserwować, że problem SAT jest w klasie NP. Niedeterministyczna maszyna Turinga może niedeterministycznie wybrać wartościowanie zmiennych i sprawdzać, czy formuła jest spełniona. Pokażemy, że problem SAT jest NP-trudny, a więc jest NP-zupełny.

Niech H będzie problemem takim, że

$$H = \left\{ \langle N, x, 0^t \rangle \;\middle|\; \begin{array}{l}\text{niedeterministyczna maszyna Turinga } N \\ \text{akceptuje słowo } x \text{ w } t \text{ krokach}\end{array} \right\}.$$

Łatwo zauważyć, że H jest problemem NP-trudnym (można do niego sprowadzić każdy inny problem z NP) oraz sam należy do NP (maszyna M_H może niedeterministycznie wybierać relację δ i symulować działanie maszyny N). Wystarczy więc pokazać, że H redukuje się do SAT.

Zdefiniujemy następujące zmienne logiczne:

- $Q_{i,k}$: w i-tym kroku maszyna M_H jest w stanie q_k,
- $P_{i,j}$: w i-tym kroku głowica maszyny M_H jest na j-tym polu,
- $S_{i,j,l}$: w i-tym kroku na j-tym polu taśmy znajduje się symbol l.

Będziemy chcieli tak skonstruować formułę logiczną ϕ, że ϕ jest spełnialna wtedy i tylko wtedy, gdy istnieje ścieżka obliczeń maszyny M_H akceptująca słowo x czasie wielomianowym. W tym celu do formuły ϕ dodajemy klauzule, które:

1. wymuszają, że w każdym kroku maszyna M_L znajduje się w dokładnie jednym stanie, to jest
$$\bigwedge_i \left(Q_{i,1} \vee \cdots \vee Q_{i,|Q|}\right),$$
$$\bigwedge_i \bigwedge_{k \neq k'} \left(\neg Q_{i,k} \vee \neg Q_{i,k'}\right);$$
2. wymuszają, że w każdym kroku głowica maszyny M_L znajduje się na dokładnie jednym polu;
3. wymuszają, że w każdym kroku na każdym polu taśmy znajduje się dokładnie jeden symbol;
4. wymuszają zgodność słowa wejściowego, to jest
$$\bigwedge_{j \leqslant |x|} \left(S_{1,j,x_j}\right) \quad \bigwedge_{|x|<j} \left(S_{1,j,\square}\right);$$
5. wymuszają poprawność przejść między stanami;
6. wymuszają, że istnieje ścieżka obliczeń, która kończy się w stanie akceptującym, to jest
$$\left(Q_{1,Y} \vee Q_{2,Y} \vee \cdots\right).$$

Klauzul tych jest wielomianowo wiele, więc pokazaliśmy, że $H \leqslant_m^p$ SAT, a więc problem SAT jest NP-zupełny. □

5. Klasyczne problemy NP-zupełne

W tej sekcji pokażemy najważniejsze problemy NP-zupełne oraz dowody ich NP-zupełności. W ramach dodatku A przedstawiamy graf, który może ułatwić Czytelnikowi nagiwację przez wszystkie twierdzenia tej sekcji.

Dowodząc NP-zupełności problemu, wystarczy pokazać, że problem jest w klasie NP oraz że jest NP-trudny. Tę pierwszą cześć z reguły będziemy zostawiać jako ćwiczenie dla Czytelnika, gdyż jest to zwykle dosyć proste — niedeterministyczna maszyna Turinga może po prostu wybierać odpowiedź (wartościowanie formuły logicznej, podzbiór wierzchołków w grafie, itp.) i sprawdzać, czy jest dobrym *świadkiem*, czyli czy spełnia warunki problemu.

5.1. Problem 3-SAT i jego bardziej szczegółowe wersje

Problem spełnialności formuł w postaci CNF, gdzie każda klauzula ma co najwyżej k literałów będziemy nazywać problemem k-SAT. Problem 3-SAT jest NP-zupełny, co udowodnimy jako twierdzenie 5.2, jednak już 2-SAT jest w klasie P, a rozwiązujący go algorytm jest opisany chociażby na cp-algorithms.

Możemy również w inny sposób wprowadzić ograniczenia na formułę, które nie sprawią, że problem przestanie być NP-zupełny, a może stać się bardziej przydatny. Takim ograniczeniem będzie chociażby maksymalna liczby wystąpień każdej zmiennej w formule. Problem k-SAT, w którym każda zmienna występuje co najwyżej ℓ razy nazywamy (k, ℓ)-SAT. W ramach twierdzenia 5.3 pokażemy, że problem $(3, 3)$-SAT również jest NP-zupełny.

Uwaga 5.1

Często można spotkać również alternatywną definicję problemu k-SAT; mianowicie, że jest to problem spełnialności formuł w postaci CNF, gdzie każda klauzula ma *dokładnie* k literałów.

Zazwyczaj nie ma to żadnego znaczenia w kontekście przeprowadzanych redukcji, ale jeśli zdefiniujemy $(3, 3)$-SAT jako problem spełnialności formuł w postaci CNF, gdzie każda klauzula ma *dokładnie* trzy literały, a każda zmienna występuje co najwyżej trzy razy, to ten problem jest już w P, a nawet więcej — formuła w takiej postaci zawsze jest tautologią, co dociekliwy Czytelnik może udowodnić za pomocą twierdzenia Halla o kojarzeniu małżeństw.

Warto zauważyć, że problem SAT, w którym każda zmienna występuje co najwyżej dwa razy, jest w P, podobnie jak problem SAT, w którym każda klauzula ma co najwyżej jeden niezanegowany literał. Oba te fakty dociekliwy Czytelnik powinien udowodnić.

Twierdzenie 5.2

Problem 3-SAT jest NP-zupełny.

Dowód. Zauważmy, że klauzula

$$a_1 \vee a_2 \vee \cdots \vee a_n$$

jest równoważna formule

$$((a_1 \vee a_2) \Leftrightarrow b) \wedge (b \vee a_3 \vee \cdots \vee a_n),$$

a z kolei

$$((a_1 \vee a_2) \Leftrightarrow b)$$

jest równoważne

$$(\neg a_1 \vee b) \wedge (\neg a_2 \vee b) \wedge (a_1 \vee a_2 \vee \neg b).$$

W ten sposób możemy zredukować rozmiar każdej klauzuli z n do $\max(3, n-1)$. Powtarzając ten proces wielokrotnie, dokonujemy redukcji problemu SAT do 3-SAT. □

Twierdzenie 5.3

Problem $(3, 3)$-SAT jest NP-zupełny.

Dowód. Jeśli jakaś zmienna x występuje $m > 3$ razy w formule ϕ, to jej i-te wystąpienie możemy zastąpić nową zmienną x_i, a do formuły ϕ dodać klauzule

$$(x_1 \vee \overline{x_2}) \wedge (x_2 \vee \overline{x_3}) \wedge ... \wedge (x_{m-1} \vee \overline{x_m}) \wedge (x_m \vee \overline{x_1}).$$

Jeśli x_1 jest fałszywe, to fałszywe musi być również x_2, a tym samym $x_3, \dots, x_m$, co oznacza, że wszystkie zmienne x_i muszą być albo fałszywe, albo prawdziwe. W ten sposób zredukowaliśmy m-krotne występowanie zmiennej x do 3-krotnego występowania zmiennych x_i, a więc 3-SAT $\leqslant_m^p$ $(3,3)$-SAT. □

5.2. Problemy Independent Set, Clique i Vertex Cover

Problem Independent Set to problem stwierdzenia, czy w danym grafie nieskierowanym G istnieje zbiór k wierzchołków niezależnych, czyli takich, że żadne dwa z nich nie są połączone krawędzią. Taki zbiór nazywamy *zbiorem niezależnym*.

Twierdzenie 5.4

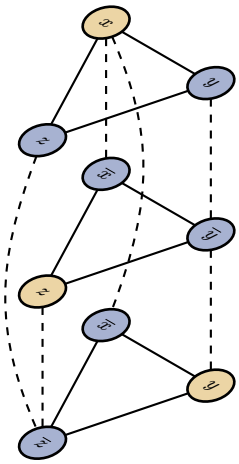
Problem Independent Set jest NP-zupełny.

Dowód. Przeprowadzimy redukcję problemu 3-SAT do problemu Independent Set. Niech ϕ będzie formułą w postaci CNF, gdzie każda klauzula ma co najwyżej trzy literały. Zbudujemy graf G w następujący sposób:

- Dla każdego literału w każdej klauzuli tworzymy wierzchołek; łączymy krawędziami wierzchołki odpowiadające literałom w tej samej klauzuli.
- Łączymy krawędziami wierzchołki odpowiadające przeciwstawnym literałom w różnych klauzulach (to znaczy takie, że jeden jest zanegowany, a drugi nie).

Graf G ma wtedy zbiór k wierzchołków niezależnych wtedy i tylko wtedy, gdy formuła ϕ zawierająca k klauzul jest spełnialna. Zbiór k wierzchołków niezależnych w grafie G odpowiada zbiorowi k wartościowań zmiennych, które spełniają formułę ϕ.

W ten sposób pokazaliśmy, że 3-SAT $\leqslant_m^p$ Independent Set, a więc problem Independent Set jest NP-trudny. Jest on również w NP, więc jest NP-zupełny. □



Rysunek 1: Graf odpowiadający formule $\phi = (x \vee y \vee z) \wedge (\neg x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z)$. Istnieje zbiór $k = 3$ wierzchołków niezależnych (zaznaczony), a więc formuła ϕ jest spełnialna.

Problem CLIQUE to problem stwierdzenia, czy w danym grafie nieskierowanym G istnieje klika (podgraf pełny) rzędu k, a VERTEX COVER to problem stwierdzenia, czy w danym grafie nieskierowanym G istnieje zbiór k wierzchołków, które pokrywają wszystkie krawędzie grafu (to znaczy, że każda krawędź ma przynajmniej jeden koniec w tym zbiorze). Zbiór wierzchołków pokrywających wszystkie krawędzie grafu nazywamy *pokryciem wierzchołkowym*.

Lemat 5.5

Niech $G = (V, E)$ będzie grafem nieskierowanym, a $S \subseteq V$ pewnym podzbiorem jego wierzchołków. Poniższe warunki są równoważne:

(a) S jest zbiorem niezależnym w G,

(b) $V \setminus S$ jest pokryciem wierzchołkowym G,

(c) S jest zbiorem wierzchołków kliki w grafie $\overline{G}$.

Dowód. Udowodnimy trzy zależności, które razem dowodzą równoważności.

(a) $\implies$ (b): Załóżmy, że C nie jest pokryciem wierzchołkowym. Istnieje więc niepokryta krawędź uv, to znaczy taka, że $u, v \notin C$. Z tego wynika, że $u, v \in S$, a więc S nie jest zbiorem wierzchołków niezależnych, co prowadzi do sprzeczności z założeniem.

(b) $\implies$ (a): Załóżmy, że S nie jest zbiorem niezależnym. Istnieje więc krawędź uv, która łączy wierzchołki $u, v \in S$. Z tego wynika, że $u, v \notin C$, a więc C nie pokrywa krawędzi uv, co prowadzi do sprzeczności z założeniem.

(a) $\iff$ (c): Z definicji dopełnienia $uv \in E \iff uv \notin \overline{E}$. □

Twierdzenie 5.6

Problem CLIQUE jest NP-zupełny.

Dowód. Wniosek z lematu 5.5 oraz twierdzenia 5.4. Wystarczy zauważyć, że dopełnienie grafu możemy zbudować w czasie wielomianowym. □

Twierdzenie 5.7

Problem VERTEX COVER jest NP-zupełny.

Dowód. Wniosek z lematu 5.5 oraz twierdzenia 5.4. Zbiór niezależny o liczności k w grafie G istnieje wtedy i tylko wtedy, gdy w tym grafie istnieje pokrycie wierzchołkowe o liczności $|V| - k$. □

5.3. Problem 3-COLOR

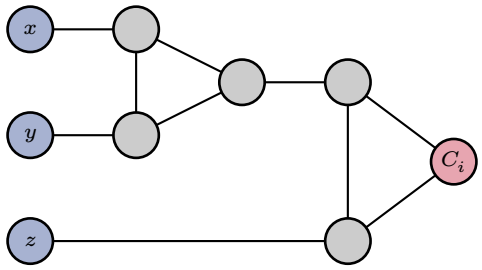
Problem 3-COLOR to problem stwierdzenia, czy dany graf nieskierowany G można pokolorować trzema kolorami w taki sposób, żeby żadne dwa sąsiednie wierzchołki nie miały tego samego koloru.

Twierdzenie 5.8

Problem 3-COLOR jest NP-zupełny.

Dowód. Przeprowadzimy redukcję problemu 3-SAT do problemu 3-Color. Niech ϕ będzie formułą w postaci CNF, gdzie każda klauzula ma co najwyżej trzy literały. Zbudujemy graf G w taki sposób, że jego wierzchołkami będą literały (w postaci zarówno zmiennych, jak i ich negacji) oraz klauzule. Ponadto, do grafu G dodamy specjalne wierzchołki T (prawda), F (fałsz) i S (inny) oraz krawędzie między nimi. Dodamy również wszystkie krawędzie między wierzchołkami reprezentującymi literały a wierzchołkiem S (literały powinny mieć kolor taki jak P lub F) oraz między wierzchołkami reprezentującymi klauzule a wierzchołkami F i S (klauzule powinny mieć taki kolor jak P).

Dla każdej klauzuli $C_i = (x \vee y \vee z)$ tworzymy dodatkowo następującą strukturę:



Oczywiście jeśli klauzula ma mniej niż 3 literały, to odpowiednio tę strukturę upraszczamy. Nietrudno zauważyć, że tak stworzony graf G jest 3-kolorowalny wtedy i tylko wtedy, gdy formuła ϕ jest spełnialna, ponieważ nasza struktura symuluje alternatywę logiczną.

W ten sposób pokazaliśmy, że 3-SAT $\leqslant_m^p$ 3-Color, a więc problem 3-Color jest NP-trudny. Jest on również w NP, więc jest NP-zupełny. □

5.4. Problem (Directed) Hamiltonian Path

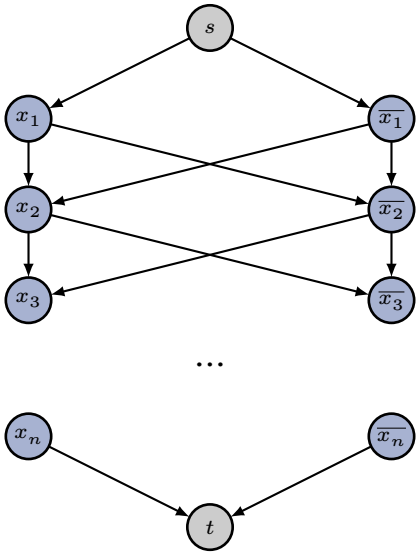
Problem Directed Hamiltonian Path to problem stwierdzenia, czy w danym grafie skierowanym G istnieje ścieżka Hamiltona $s \rightsquigarrow t$, czyli taka, która przechodzi przez każdy wierzchołek dokładnie raz. Problem Hamiltonian Path jest analogiczny, jedynie graf G jest nieskierowany.

Twierdzenie 5.9

Problem Directed Hamiltonian Path jest NP-zupełny.

Dowód. Przeprowadzimy redukcję problemu 3-SAT do problemu Directed Hamiltonian Path. Niech $\phi = C_1 \wedge \cdots \wedge C_k$ będzie formułą w postaci CNF, gdzie każda klauzula C_i ma co najwyżej trzy literały. Zbudujemy graf skierowany G w następujący sposób:

1. W grafie tworzymy dwa wyróżnione wierzchołki: s oraz t. Dla każdej zmiennej x_j tworzymy również dwa wierzchołki: x_j i $\overline{x_j}$. Dodajemy krawędzie $s\,x_1$, $s\,\overline{x_1}$, $x_n\,t$, $\overline{x_n}\,t$ oraz dla każdego $1 < j < n$ krawędzie $x_j\,x_{j+1}$, $x_j\,\overline{x_{j+1}}$, $\overline{x_j}\,x_{j+1}$ i $\overline{x_j}\,\overline{x_{j+1}}$, otrzymując w ten sposób poniższy graf.

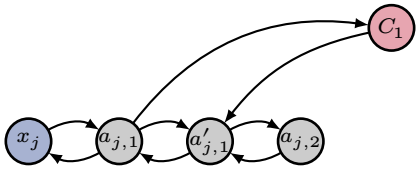


W takim grafie na pewno nie ma ścieżki Hamiltona, ponieważ musimy odwiedzić wszystkie wierzchołki, a na razie możemy odwiedzić tylko jeden z każdej pary $x_j, \overline{x_j}$.

2. Dla każdej pary x_j, $\overline{x_j}$ tworzymy dodatkowe wierzchołki $a_{j,1}, a'_{j,1}, \ldots, a_{j,k}, a'_{j,k}$ i łączymy je w następujący sposób:

Teraz możemy już znaleźć ścieżkę Hamiltona; każda taka ścieżka będzie odpowiadała pewnemu wartościowaniu formuły ϕ (przejście ścieżką $x_j \rightsquigarrow \overline{x_j}$ oznacza wartościowanie $x_j =$ prawda, a przejście ścieżką $\overline{x_j} \rightsquigarrow x_j$ oznacza wartościowanie $x_j =$ fałsz).

3. Dla każdej klauzuli C_i tworzymy odpowiadający jej wierzchołek C_i. Jeśli klauzula C_i zawiera x_j, to tworzymy krawędzie $a_{j,i}\, C_i$ oraz $C_i\, a'_{j,i}$, a jeśli zawiera $\overline{x_j}$, to tworzymy krawędzie $C_i\, a_{j,i}$ oraz $a'_{j,i}\, C_i$. Na przykład dla klauzuli C_1 zawierającej x_j otrzymujemy graf



Nietrudno zauważyć, że w grafie G istnieje ścieżka Hamiltona $s \rightsquigarrow t$ (przechodząca również przez wierzchołki C_i) wtedy i tylko wtedy, gdy formuła ϕ jest spełnialna. W ten sposób pokazaliśmy, że 3-SAT $\leqslant_m^p$ Directed Hamiltonian Path, a więc problem Directed Hamiltonian Path jest NP-trudny. Jest on również w NP, więc jest NP-zupełny. □

Twierdzenie 5.10

Problem Hamiltonian Path jest NP-zupełny.

Dowód. Pokażemy redukcję problemu Directed Hamiltonian Path do Hamiltonian Path. Niech G będzie grafem skierowanym, a G' grafem nieskierowanym, który otrzymujemy z G w taki sposób, że każdy wierzchołek $v \in G$ przepisujemy do G' jako trzy (połączone ścieżką) wierzchołki: v_{in}, v_{mid} i v_{out}, a każdą krawędź skierowaną $uv \in G$ przepisujemy do grafu G' jako krawędź $u_{\text{out}}v_{\text{in}}$.

Oczywiście nie każda nieskierowana ścieżka w grafie G' odpowiada dokładnie jednej skierowanej ścieżce w grafie G, ale jeśli ograniczymy się tylko do ścieżek Hamiltona, to ten fakt istotnie zachodzi, ponieważ taka ścieżka musi przechodzić dokładnie raz przez wszystkie wierzchołki v_{mid} (a więc raz wchodzi i raz wychodzi z wierzchołka). W ten sposób pokazaliśmy, że Directed Hamiltonian Path $\leqslant_m^p$ Hamiltonian Path, a więc problem Hamiltonian Path jest NP-trudny. Jest on również w NP, więc jest NP-zupełny. □

Bardzo łatwo zauważyć, że problemy Directed Hamiltonian Cycle i Hamiltonian Cycle również są NP-zupełne — w dowodach wystarczy dodać krawędź ts.

5.5. Problemy Set Cover i X3C

Problem Set Cover to problem stwierdzenia, czy dany zbiór $B = \{b_1, \dots, b_n\}$ można reprezentować jako sumę mnogościową co najwyżej t zbiorów z danej rodziny podzbiorów B, oznaczanej jako $\mathcal{S} = \{S_1, \dots, S_m\} \subseteq \mathcal{P}(B)$. Taka reprezentacja nazywa się *pokryciem* zbioru.

Podobnie jak dla problemu SAT, będziemy chcieli wprowadzić pewne ograniczenie danych tego problemu, które, jak udowodnimy, będzie równoważne pełnemu problemowi. W tym przypadku będzie to problem X3C (*exact cover by 3-sets*), w którym dodatkowo $n = 3t$ oraz $|S_i| = 3$ dla każdego i (z czego wynika, że zbiory z $\mathcal{S}$ muszą być rozłączne).

Twierdzenie 5.11

Problem Set Cover jest NP-zupełny.

Dowód. Pokażemy redukcję problemu 3-SAT do problemu Set Cover. Niech $\phi = C_1 \wedge \dots \wedge C_k$ będzie formułą w postaci CNF, gdzie każda klauzula C_i ma co najwyżej trzy literały. Niech zbiór B zawiera zmienne oraz klauzule:

$$B = \{x_1, \dots, x_n, C_1, \dots, C_k\},$$

a każda zmienna x_j ma dwa odpowiadające zbiory:

$$S_{x_j=0} = \{x_j\} \cup \left\{ C_i \;\middle|\; \text{klauzula } C_i \text{ zawiera (zaprzeczony) literał } \overline{x_j} \right\},$$
$$S_{x_j=1} = \{x_j\} \cup \left\{ C_i \;\middle|\; \text{klauzula } C_i \text{ zawiera (niezaprzeczony) literał } x_j \right\},$$

należące do rodziny $\mathcal{S}$.

Zbiór B można pokryć n zbiorami z $\mathcal{S}$ wtedy i tylko wtedy, gdy formuła ϕ jest spełnialna (jeśli formuła nie jest spełnialna, to nie uda się pokryć wszystkich klauzul C_i, a w przeciwnym wypadku będziemy mogli to zrobić, wybierając jeden z każdej pary zbiorów $S_{x_j=b}$). W ten sposób pokazaliśmy, że 3-SAT $\leqslant_m^p$ Set Cover, a więc problem Set Cover jest NP-trudny. Jest on również w NP, więc jest NP-zupełny. □

Twierdzenie 5.12
Problem X3C jest NP-zupełny.

Dowód. Dowód będzie rozwinięciem dowodu twierdzenia 5.11.

Zauważmy, że jeśli zamiast 3-SAT rozważymy $(3,3)$-SAT, to wtedy każdy zbiór z $\mathcal{S}$ ma co najwyżej 4 elementy (zmienna oraz co najwyżej 3 klauzule). Jeśli jednak istnieje zbiór $S_{x_j=b}$ mający dokładnie 4 elementy, to znaczy, że zmienna x_j jest użyta w takiej samej formie, to znaczy zawsze zanegowana, lub zawsze niezanegowana. Możemy ją więc spokojnie pominąć (wartościując ją na $b \in \{0,1\}$) i rozważać tylko te zbiory z rodziny $\mathcal{S}$, które mają co najwyżej 3 elementy. Aby nie wprowadzać zbyt wielu oznaczeń, przyjmiemy, że takie zmienne nie występują w formule ϕ.

Mamy już więc redukcję $(3,3)$-SAT do problemu pokrycia zbiorami o maksymalnie 3 elementach. Aby otrzymać redukcję do problemu X3C, należy wykonać dwa dosyć techniczne zabiegi:

1. Dla każdego $j \leqslant n$, do rodziny $\mathcal{S}$ dodajemy wszystkie zawierające x_j podzbiory zbioru $S_{x_j=b}$ (zamiast samego zbioru $S_{x_j=b}$). Na przykład

$$\{x_j, C_1, C_2\} \longrightarrow \{x_j, C_1, C_2\}, \{x_j, C_1\}, \{x_j, C_2\}, \{x_j\}.$$

   W ten sposób zagwarantujemy, że jeśli istnieje pokrycie, to istnieje również pokrycie o takiem samej liczności, które jest dokładne (*exact*), a więc składa się ze zbiorów rozłącznych.

2. Musimy pokryć zbiór zawierający $n+k$ elementów dokładnie n zbiorami dokładnie 3-elementowymi. Aby to umożliwić, dodajemy dodatkowe $3n-(n+k)$ elementów $d_1, \dots, d_{2n-k}$ do zbioru B. Ponadto, każdy zbiór $S \in \mathcal{S}$, który ma mniej niż trzy elementy, zastępujemy zbiorami uzupełnionymi do trzech elementów przez dowolne elementy $d_1, \dots, d_{2n-k}$ (wszystkie ich kombinacje). Na przykład dla d_1, d_2, d_3 mamy

$$\begin{aligned}
\{x_j\} &\longrightarrow \{x_j, d_1, d_2\}, \{x_j, d_1, d_3\}, \{x_j, d_2, d_3\}, \\
\{x_j, C_1\} &\longrightarrow \{x_j, C_1, d_1\}, \{x_j, C_1, d_2\}, \{x_j, C_1, d_3\}, \\
\{x_j, C_1, C_2\} &\longrightarrow \{x_j, C_1, C_2\}.
\end{aligned}$$

   W ten sposób gwarantujemy, że każdy zbiór z $\mathcal{S}$ ma dokładnie 3 elementy, a jednocześnie zbiór B można pokryć n zbiorami wtedy i tylko wtedy, jeśli przed tym krokiem istniało pokrycie.

Teraz mamy już pewną instancję problemu X3C, a więc pokazaliśmy, że $(3,3)$-SAT się do tego problemu redukuje, z czego wynika, że problem X3C również jest NP-zupełny. □

5.6. Problemy Subset Sum i Partition

Problem Subset Sum to problem stwierdzenia, czy dany zbiór liczb naturalnych $A = \{a_1, \dots, a_n\}$ zawiera podzbiór sumujący się do danej liczby $t \in \mathbb{N}$.

Czytelnik może znać algorytm rozwiązania tego problemu, który działa w czasie $\mathcal{O}(nt)$. Może wydawać się on wielomianowy, ale nie jest wielomianowy względem rozmiaru danych wejściowych, a jedynie względem *wartości* t. Takie algorytmy będziemy nazywać *pseudowielomianowymi*. Problemy NP-zupełne, które przestają być NP-zupełne, jeśli zastosujemy unarne kodowanie liczb nazywamy *słabo NP-zupełnymi*.

Twierdzenie 5.13

Problem Subset Sum jest NP-zupełny.

Dowód. Pokażemy redukcję problemu X3C do problemu Subset Sum. Niech $(B, \mathcal{S}, k)$ będzie instancją problemu X3C, gdzie $\mathcal{S} = \{S_1, \dots, S_m\}$ i $|B| = 3k$. Zdefiniujmy zbiór liczb

$$A = \left\{ s_i \;\middle|\; \sum_{j=1}^{3k} (m+1)^j \left[b_j \in S_i\right] \right\},$$

gdzie $\left[b_j \in S_i\right]$ jest nawiasem Iversona. Niech

$$t = \sum_{j=1}^{3k} (m+1)^j.$$

Zbiór A zawiera podzbiór sumujący się do t wtedy i tylko wtedy, gdy istnieje dokładnie k zbiorów z rodziny $\mathcal{S}$, które pokrywają zbiór B. Pokazaliśmy, że X3C $\leqslant_m^p$ Subset Sum, a więc problem Subset Sum jest NP-trudny. Jest on również w NP, więc jest NP-zupełny. □

Problem Partition to problem stwierdzenia, czy dany zbiór liczb naturalnych $A = \{a_1, \dots, a_n\}$ można podzielić na dwa podzbiory o tej samej sumie. Problem ten jest oczywiście szczególnym przypadkiem problemu Subset Sum, w którym $t = \frac{1}{2}\sum_{i=1}^{n} a_i$. Redukcję z problemu Subset Sum do problemu Partition zostawiamy jako ćwiczenie dla Czytelnika.

6. Struktura klas złożoności obliczeniowej

W porównaniu do klas języków szerszych niż R struktura klas języków złożoności obliczeniowej jest dużo mniej oczywista. Znanym nierozstrzygniętym pytaniem jest w szczególności to, czy P = NP (chociaż wydaje się, że odpowiedź na to pytanie jest negatywna) oraz czy NP = coNP. W tej sekcji udowodnimy kilka ważnych zależności, które jednak są znane.

Uwaga 6.1 (o zamkniętości klas ze względu na dopełnienie)

Wszystkie deterministyczne klasy złożoności obliczeniowej (jak P czy EXP) są zamknięte ze względu na dopełnienie, to znaczy, że jeśli język L należy do danej klasy, to należy do niej również $\overline{L}$ (ponieważ w deterministycznej maszynie Turinga możemy po prostu zamienić stany akceptujące z odrzucającymi).

Sytuacja wygląda zupełnie inaczej w przypadku klas niedeterministycznych. Zamkniętość ze względu na dopełnienie chociażby klasy NP jest problemem otwartym.

Twierdzenie 6.2

Jeśli P = NP, to coNP = NP.

Dowód. Skoro P = NP i klasa P jest zamknięta ze względu na dopełnienia, to klasa NP również jest zamknięta ze względu na dopełnienie. □

Uwaga 6.3 (o zamkniętości klas ze względu na morfizmy)

Funkcję $f : \Sigma_1^* \to \Sigma_2^*$ nazywamy *morfizmem*, jeśli dla dowolnego $a = a_1 \cdots a_n \in \Sigma_1^*$, zachodzi $f(a) = f(a_1) \cdots f(a_n)$ (to znaczy, że f jest jednoznacznie określona przez funkcję pojedynczych symboli).

Łatwo pokazać, że klasa NP jest zamknięta ze względu na morfizmy, to znaczy, że jeśli $L \in \mathsf{NP}$ i f jest morfizmem, to $f(L) \in \mathsf{NP}$. Dowód jest następujący: jeśli M jest niedeterministyczną maszyną Turinga rozstrzygającą o L, to możemy zbudować maszynę M', która na wejściu $f(a)$ zgaduje a oraz symuluje działanie M na a.

Taki sam tok rozumowania nie zadziała jednak wewnątrz klasy P. Możemy udowodnić, że jeśli klasa P jest zamknięta ze względu na morfizmy, to $\mathsf{P} = \mathsf{NP}$. W tym celu rozważmy problem SAT. Sprawdzenie wartościowania formuły logicznej jest oczywiście w P, w szczególności

$$L = \{(\phi, w) \mid w \text{ jest świadkiem spełnialności formuły } \phi\}$$

należy do P. Weźmy pewien alfabet Σ kodujący[a] pary (ϕ, w) oraz taki morfizm f, że $f(\phi) = \phi$ oraz $f(w) = 0$. Wtedy $f(L)$ redukuje się do SAT, więc należy do P, tylko jeśli $\mathsf{P} = \mathsf{NP}$.

Na koniec warto zauważyć, że powyższy fakt nie implikuje, że zamkniętość ze względu na morfizmy każdego podzbioru P oznacza równość $\mathsf{P} = \mathsf{NP}$. Można chociażby udowodnić, że klasa języków regularnych ma tę własność.

[a] dla formalności będziemy chcieli przeznaczyć rozłączne zbiory symboli na ϕ oraz w

6.1. Klasy P, NP i problemy NP-pośrednie

Twierdzenie 6.4 (Ladnera)

Jeśli $\mathsf{P} \neq \mathsf{NP}$, to istnieje język $L \in \mathsf{NP} \setminus \mathsf{P}$ taki, że L nie jest NP-zupełny.

Język, który spełnia warunki twierdzenia Ladnera, nazywamy *NP-pośrednim* (ang. *NP-intermediate*).

Twierdzenie 6.5 (Uogólnione twierdzenie Ladnera)

Jeśli $\mathsf{P} \neq \mathsf{NP}$ oraz $B \in \mathsf{NP} \setminus \mathsf{P}$, to istnieje język $A \in \mathsf{NP} \setminus \mathsf{P}$ taki, że $A \leqslant_m^p B$, ale $B \not\leqslant_m^p A$.

Z powyższego twierdzenia wynika, że jeśli $\mathsf{P} \neq \mathsf{NP}$, to istnieje nieskończenie wiele klas języków między P a NP, a więc i nieskończenie wiele problemów NP-pośrednich.

Definicja 6.6. Język L jest *rzadki* (ang. *sparse*) jeśli istnieje wielomian s taki, że dla każdej naturalnej liczby n

$$\left|L \cap \Sigma^{\leqslant n}\right| \leqslant s(n).$$

Twierdzenie 6.7 (Mahaneya)

Jeśli istnieje rzadki język NP-trudny, to $\mathsf{P} = \mathsf{NP}$.

Dowód z [4]. Niech L będzie NP-trudnym językiem rzadkim. Pokażemy wielomianowy algorytm rozwiązujący SAT.

Niech ϕ będzie dowolną formułą logiczną o n zmiennych $x_1, \ldots, x_n$. Jeśli zaczęlibyśmy wartościować kolejne zmienne, to stworzymy pełne drzewo binarne o n poziomach, gdzie na

poziomie 0 jest sama formuła ϕ, na poziomie 1 są formuły $\phi[x_1=0]$ i $\phi[x_1=1]$, na poziomie 2 formuły $\phi[x_1=0,x_2=0]$ i $\phi[x_1=0,x_2=1]$ oraz $\phi[x_1=1,x_2=0]$ i $\phi[x_1=1,x_2=1]$ i tak dalej. Takie drzewo ma kluczową własność: dla każdego poziomu ℓ

$$\phi \text{ jest spełnialna} \iff \text{istnieje spełnialna formuła na poziomie } \ell. \tag{1}$$

Oczywiście nie chcemy budować takiego drzewa (byłoby to wykładnicze), ale spróbujemy znacznie zmiejszyć liczbę formuł na każdym poziomie, zachowując jednak własność 1.

Niech $f : \Sigma^* \to \Sigma^*$ będzie wielomianową redukcją z SAT do L. Oznaczy przez s funkcję ograniczającą liczbę słów z L o odpowiednio małej długości (jak w definicji 6.6), a przez r funkcję ograniczającą długość słowa powstałego z f (to znaczy, że $|f(\tau)| \leqslant r(|\tau|)$ dla każdej formuły τ). Będziemy stopniowo budować kolejne poziomy drzewa w taki sposób, że każdym poziomie ℓ najpierw tworzymy wszystkie dzieci formuł z poziomu $\ell-1$, a następnie usuwamy wszystkie takie formuły $\phi_{\ell,j}$, że

$$i < j \text{ oraz } f(\phi_{\ell,1} \vee \phi_{\ell,i}) = f(\phi_{\ell,1} \vee \phi_{\ell,j}),$$

gdzie $\phi_{\ell,i}$ to częściowo wartościowane formuły występujące na poziomie ℓ.

Mamy dwa istotnie różne przypadki ze względu na liczbę pozostałych formuł na danym poziomie:

(a) Liczba formuł jest większa niż $s(r(2n+5))$. (Wielomian $2n+5$ wziął się z faktu, że do funkcji f przekazywaliśmy dwie formuły oraz dodatkowe pięć znaków: $() \vee ()$.) Wtedy możemy usunąć formułę $\phi_{\ell,1}$, ponieważ na pewno nie jest ona spełnialna (ponieważ przynajmniej jedno słowo $f(\phi_{\ell,1} \vee \phi_{\ell,i})$ nie należy do L, więc $\phi_{\ell,1} \vee \phi_{\ell,i}$ nie jest spełnialna, więc $\phi_{\ell,1}$ też nie jest spełnialna). Powtarzamy ten krok aż do momentu, gdy liczba formuł jest mniejsza lub równa $s(r(2n+5))$, nie tracąc jednak własności 1.

(b) Liczba formuł jest mniejsza lub równa $s(r(2n+5))$. Wtedy kontynuujemy budowę drzewa na kolejnym poziomie.

Na ostatnim poziomie sprawdzamy, czy istnieje tautologia. Jeśli tak, to formuła ϕ jest spełnialna, w przeciwnym wypadku nie jest. Znaleźliśmy wielomianowy algorytm rozwiązujący SAT, więc $\mathsf{P} = \mathsf{NP}$. □

Uwaga 6.8 (o istnieniu języków rzadkich poza klasą NP)

Załóżmy, że $\mathsf{P} \neq \mathsf{NP}$. Twierdzenie Mahaneya mówi jedynie o tym, że nie ma rzadkich języków NP-trudnych. Nie znaczy to, że wszystkie rzadkie języki są „łatwiejsze" (w sensie należenia do węższej klasy) od języków NP-trudnych. W szczególności więc nie wyklucza ono istnienia rzadkich języków, które w ogóle nie należą do NP.

Pokażemy, że istnieje rzadki (a nawet unarny) język, który nie należy do NP, a jest rozstrzygalny (istnieją również nierozstrzygalne języki unarne). Niech L będzie dowolnym rozstrzygalnym językiem nienależącym do klasy EXPSPACE (z twierdzenia o hierachii pamięciowej taki język na pewno istnieje), a L' jego unarnym kodowaniem. Język L' nie należy do PSPACE, a więc tym bardziej do NP.

6.2. Klasy EXP i NEXP

Twierdzenie 6.9

Jeśli $\mathsf{P} = \mathsf{NP}$, to $\mathsf{EXP} = \mathsf{NEXP}$.

Dowód. Pokażemy typową metodę zwaną *paddingiem*. Załóżmy, że $\mathsf{P} = \mathsf{NP}$ oraz niech $L \in \mathsf{NEXP}$. Istnieje więc niedeterministyczna maszyna Turinga M, która rozstrzyga o L w 2^{n^k} krokach. Niech

$$L' = \{x\diamondsuit^{2^{|x|^k}} \mid x \in L\},$$

gdzie $\diamondsuit$ jest specjalnym symbolem, który nie należy do alfabetu języka L. Konstruujemy niedeterministyczną maszynę Turinga M', która rozstrzyga o L' w taki sposób, że dla każdego $y \in L'$ sprawdza, czy y jest we właściwej formie, a następnie symuluje M przez $2^{|x|^k}$ kroków. Robi to w czasie $\mathcal{O}(2^{|x|^k})$, a więc wielomianowym w stosunku do $|y|$. Z tego wynika, że $L' \in \mathsf{NP}$.

Zgodnie z naszym założeniem $L' \in \mathsf{P} = \mathsf{NP}$, a więc istnieje deterministyczna maszyna Turinga M'_D, która rozstrzyga o L' w czasie wielomianowym. Możemy więc skonstruować deterministyczną maszynę Turinga $M_D(x)$, która symuluje $M'_D(x\diamondsuit^{2^{|x|^k}})$ w czasie wykładniczym względem $|x|$. Z tego wynika, że $L \in \mathsf{EXP}$. □

Definicja 6.10. Język L jest *unarny*, jeśli jego alfabet składa się z jednego symbolu, to znaczy $L \subseteq \{0\}^*$.

Zauważmy, że każdy język unarny jest rzadki. W szczególności więc z twierdzenia Mahaneya (6.7) wynika, że jeśli istnieje unarny język NP-trudny, to $\mathsf{P} = \mathsf{NP}$. Możemy też dowieść inny fakt dotyczący języków unarnych, będący silniejszą wersją twierdzenia 6.9.

Twierdzenie 6.11

Jeśli każdy unarny język z NP jest w P, to EXP = NEXP.

Dowód. Dowód jest analogiczny do dowodu twierdzenia 6.9, dlatego nie będziemy go tutaj powtarzać. Założymy jedynie dodatkowo (i bez straty ogólności), że $L \in \mathsf{NEXP}$ jest językiem nad alfabetem $\{0, 1\}$ oraz zmienimy definicję języka L'. Zamiast

$$L' = \{x\diamondsuit^{2^{|x|^k}} \mid x \in L\},$$

możemy zdefiniować go jako unarny język

$$L' = \{0^m \mid x \in L\},$$

gdzie zapisem binarnym liczby m będzie $x10^{2^{|x|^k}}$. □

6.3. Maszyny z wyrocznią

Maszyna M z wyrocznią A to maszyna Turinga, która ma możliwość rozstrzygania o języku A w ramach jednego kroku.

Definicja 6.12 (autoredukcja)**.** Język A jest autoredukowalny, jeśli istnieje maszyna Turinga z wyrocznią A, która rozstrzyga o A, ale dla wejścia x nigdy nie pyta wyroczni o x.

Definicja 6.13 (samoredukcja)**.** Język A jest samoredukowalny, jeśli jest autoredukowalny oraz dla wejścia x pyta wyrocznię jedynie o słowa krótsze od x.

O językach samoredukowalnych mówimy, że mają naturalny algorytm rekurencyjny.

Twierdzenie 6.14

Każdy język NP-zupełny jest autoredukowalny.

Dowód. Niech L będzie językiem NP-zupełnym, funkcja f wielomianową redukcją z L do SAT, a funkcja g wielomianową redukcją SAT do L. Pokażemy konstrukcję maszyny Turinga M z wyrocznią L, która dla wejścia x nie pyta o x.

Dla wejścia x maszyna M oblicza $\phi = f(x)$. Jeśli formuła ϕ nie ma żadnych zmiennych, to maszyna zwraca, czy ϕ jest tautologią. W przeciwnym wypadku maszyna oblicza $x_0 = g(\phi[z_1 = 0])$ i $x_1 = g(\phi[z_1 = 1])$. Jeśli

(a) $x \neq x_0$ i $x \neq x_1$, to maszyna pyta wyrocznię o $g(\phi[z_1 = 0])$, $g(\phi[z_1 = 1])$ i zwraca, czy przynajmniej jedna z odpowiedzi jest twierdząca;

(b) w przeciwnym wypadku ($x = x_b$) wiemy, że $x \in L \iff x_b \in L$, więc próbujemy wartościować kolejne zmienne ($\phi[z_1 = b, z_2 = 0]$, $\phi[z_1 = b, z_2 = 1]$), powtarzając ten krok aż do momentu, gdy ϕ nie ma już zmiennych lub zajdzie (a).

□

Twierdzenie 6.15

Jeśli $\mathsf{P} \neq \mathsf{NP}$, to nie wszystkie języki NP-zupełne są samoredukowalne.

Klasę języków rozstrzygalnych w czasie wielomianowym przez (nie)deterministyczną maszynę Turinga z wyrocznią A oznaczamy jako P^A (odpowiednio NP^A). Łatwo zauważyć, że $\mathsf{P}^A = \mathsf{P}^{\overline{A}}$ oraz $\mathsf{NP}^A = \mathsf{NP}^{\overline{A}}$.

Ponadto, dla dowolnego zbioru języków $\mathcal{C}$ definiujemy

$$\mathsf{P}^{\mathcal{C}} = \bigcup_{A \in \mathcal{C}} \mathsf{P}^A$$

oraz analogicznie dla $\mathsf{NP}^{\mathcal{C}}$.

Fakt 6.16. Dla każdego $\mathcal{C}$-zupełnego języka A zachodzą równości

$$\mathsf{P}^A = \mathsf{P}^{\mathcal{C}} \quad \text{i} \quad \mathsf{NP}^A = \mathsf{NP}^{\mathcal{C}}.$$

Dowód. W ramach (nie)deterministycznej maszyny Turinga z wyrocznią $B \in \mathcal{C}$ możemy dokonać wielomianowej redukcji z dowolnego języka B do A. □

Fakt 6.17. Jeśli $\mathsf{P}^{\mathsf{NP}} = \mathsf{NP}$, to $\mathsf{coNP} = \mathsf{NP}$.

Dowód. Klasa P^{NP} jest deterministyczna, a więc zamknięta ze względu na dopełnienie (zobacz uwagę 6.1). Z tego wynika, że klasa NP też musi być zamknięta ze względu na dopełnienie, więc $\mathsf{NP} = \mathsf{coNP}$. □

Problemem, który należy do klasy P^{NP} jest na przykład Exact Clique. Jest to problem stwierdzenia, czy największa klika w danym grafie G ma rozmiar dokładnie k. Możemy dla każdego $k' \in \{1, \ldots, |V(G)|\}$ zapytać wyrocznię o istnienie kliki rzędu $\geqslant k$ (jest to instancja zwykłego problemu Clique) i zwrócić wynik, jeśli ostatnie k', na które odpowiedź jest twierdząca, jest równe danemu k.

Fakt 6.18. Zachodzi równość

$$\mathsf{P}^{\mathsf{NP}\cap\mathsf{coNP}} = \mathsf{NP}\cap\mathsf{coNP}.$$

Dowód. Oczywiście $\mathsf{NP}\cap\mathsf{coNP} \subseteq \mathsf{P}^{\mathsf{NP}\cap\mathsf{coNP}}$. Pokażemy, że $\mathsf{P}^{\mathsf{NP}\cap\mathsf{coNP}} \subseteq \mathsf{NP}$ oraz $\mathsf{P}^{\mathsf{NP}\cap\mathsf{coNP}} \subseteq \mathsf{coNP}$.

Niech $B \in \mathsf{P}^{\mathsf{NP}\cap\mathsf{coNP}}$ oraz $A \in \mathsf{NP}\cap\mathsf{coNP}$ będzie językiem wyroczni, a którą pyta deterministyczna maszyna Turinga M_D rozstrzygająca o B. Konstruujemy niedeterministyczną maszynę Turinga M_N, która działa tak samo jak M_D, tylko gdy M_D pyta wyrocznię o $x \in A$, to M_N niedeterministycznie zgaduje odpowiedź. W ten sposób $B \in \mathsf{NP}$.

Skoro B należy do deterministycznej klasy $\mathsf{P}^{\mathsf{NP}\cap\mathsf{coNP}}$, to również $\overline{B}$ należy do tej klasy, a więc możemy powtórzyć nasze rozumowanie dla $\overline{B}$, dowodząc, że $B \in \mathsf{coNP}$. □

6.4. Hierarchia wielomianowa

Analogicznie do hierarchii arytmetycznej, definiujemy hierarchię wielomianową. Głównym pytaniem tej sekcji będzie uogólnienie pytania $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ przez pytanie o skończoność hierarchii wielomianowej.

Definicja 6.19. Rekurencyjnie definiujemy klasy Σ_i^p, Π_i^p i Δ_i^p:

$$\Sigma_0^p = \Pi_0^p = \Delta_0^p = \mathsf{P},$$

$$\begin{aligned}\Sigma_i^p &= \mathsf{NP}^{\Sigma_{i-1}^p},\\ \Pi_i^p &= \mathsf{co}\Sigma_i^p = \mathsf{coNP}^{\Sigma_{i-1}^p},\\ \Delta_i^p &= \mathsf{P}^{\Sigma_{i-1}^p}.\end{aligned}$$

Ponadto,

$$\mathsf{PH} = \bigcup_{i\in\mathbb{N}} \Sigma_i^p.$$

Łatwo pokazać, że $\Sigma_1^p = \mathsf{NP}$, $\Pi_1^p = \mathsf{coNP}$, a $\Delta_1^p = \mathsf{P}$.

Twierdzenie 6.20
Jeśli $\Sigma_k^p = \Sigma_{k+1}^p$ dla pewnego $k \geqslant 0$, to $\mathsf{PH} = \Sigma_k^p$.

Dowód. Skoro $\Sigma_k^p = \Sigma_{k+1}^p$, to również

$$\mathsf{NP}^{\Sigma_k^p} = \mathsf{NP}^{\Sigma_{k+1}^p},$$

więc z definicji

$$\Sigma_{k+1}^p = \Sigma_{k+2}^p.$$

Kontynuując ten tok rozumowania dostajemy $\Sigma_k^p = \Sigma_{k+1}^p = \Sigma_{k+2}^p = \ldots = \mathsf{PH}$. □

Twierdzenie 6.21
Jeśli $\Sigma_k^p = \Pi_k^p$ dla pewnego $k \geqslant 1$, to $\mathsf{PH} = \Sigma_k^p$.

7. Literatura

[1] Adleman, L.M., Loui, M.C. (1981). Space-bounded simulation of multitape Turing machines. *Mathematical Systems Theory, 14*, 215-222.

[2] Faliszewski, P. (2024). Teoria obliczeń i złożoności obliczeniowej. *Wykłady prowadzone na Akademii Górniczo-Hutniczej w Krakowie.*

[3] Garey, M.R., Johnson, D.S. (1979). Computers and Intractability: A Guide to the Theory of NP-Completeness. *W.H. Freeman & Co.*

[4] Grochow, J.A. (2016). NP-hard sets are not sparse unless P=NP: An exposition of a simple proof of Mahaney's Theorem, with applications. *Electronic Colloquium on Computational Complexity, TR16.*

[5] Sipser, M. (2013). Introduction to the Theory of Computation (3rd edition). *Cengage Learning.*

A. Graf redukcji między problemami

Kolorem czerwonym zaznaczono redukcje, które są oczywiste — jeden problem jest szczególnym przypadkiem drugiego.