Złożoność obliczeniowa

Wykłady dla III roku bioinformatyki

Paweł Urzyczyn
urzy@mimuw.edu.pl

27 października 2024, godzina 15: 57

1 Języki regularne

Definicja 1.1 *Słowo* nad alfabetem $\mathcal{A}$ to dowolny skończony ciąg elementów zbioru $\mathcal{A}$. Zwykle słowo w o długości n utożsamia się z funkcją $w : n \to \mathcal{A}$, gdzie $n = \{0, \ldots, n-1\}$. Piszemy wtedy $n = |w|$. Na przykład słowo *baba* to funkcja $w : 4 \to \{a, b\}$, spełniająca warunki

$$w(i) = \begin{cases} b, & \text{jeśli } i \text{ jest parzyste;} \\ a, & \text{jeśli } i \text{ jest nieparzyste} \end{cases}$$

Zbiór wszystkich słów n-literowych nad $\mathcal{A}$ (słów o długości n) pokrywa się więc ze zbiorem $\mathcal{A}^n$ wszystkich funkcji z n do $\mathcal{A}$. Zbiór wszystkich słów nad $\mathcal{A}$ oznaczamy przez $\mathcal{A}^*$. Szczególnym słowem jest jedyne słowo o długości 0. Jest to *słowo puste*, czyli funkcja pusta. Oznaczamy je przez ε. Zbiór słów niepustych oznaczamy przez $\mathcal{A}^+$.

Fakt 1.2 *Jeśli alfabet $\mathcal{A}$ jest przeliczalny, to zbiór wszystkich słów $\mathcal{A}^*$ też jest przeliczalny.*

Zazwyczaj przyjmuje się, że alfabet jest skończony.

Konkatenacją (złożeniem) słów $w : n \to \mathcal{A}$ i $v : m \to \mathcal{A}$ nazywamy słowo $w \cdot v$ powstałe przez dopisanie słowa v na końcu słowa w. Inaczej, $w \cdot v : n + m \to \mathcal{A}$, i dla $i < n + m$ mamy:

$$(w \cdot v)(i) = \begin{cases} w(i), & \text{jeśli } i < n; \\ v(i-n), & \text{w przeciwnym przypadku.} \end{cases}$$

Operacja konkatenacji jest łączna, na przykład:

$$ein \cdot (und \cdot zwanzig) = (ein \cdot und) \cdot zwanzig = einundzwanzig$$

Słowo puste jest elementem neutralnym konkatenacji, tj. $\varepsilon \cdot w = w \cdot \varepsilon = w$ dla dowolnego słowa w. A zatem algebra $\langle \mathcal{A}^*, \cdot, \varepsilon \rangle$ jest półgrupą z jednością. W istocie jest to półgrupa wolna generowana przez $\mathcal{A}$.

Fakt 1.3 *Jeśli $\subseteq$ oznacza zawieranie funkcji, to dla dowolnych słów $w, v \in \mathcal{A}^*$,*

$$w \subseteq v \quad \Leftrightarrow \quad \exists u \in \mathcal{A}^*(v = w \cdot u).$$

Dowód: Ćwiczenie. □

A zatem $w \subseteq v$ oznacza dokładnie tyle, że słowo w jest przedrostkiem (prefiksem) słowa v.

Definicja 1.4 *Język* nad alfabetem $\mathcal{A}$ to dowolny zbiór słów nad $\mathcal{A}$ (dowolny podzbiór $\mathcal{A}^*$).

Naturalne operacje na językach to działania mnogościowe: suma, iloczyn i różnica, a także *składanie* (konkatenacja) języków:

$$L_1 \cdot L_2 = \{w \cdot v \mid w \in L_1 \wedge v \in L_2\}.$$

Zamiast $L \cdot L$ można napisać L^2. Ogólniej, przyjmujemy że $L^0 = \{\varepsilon\}$ oraz $L^{n+1} = L \cdot L^n$. Język

$$L^* = \bigcup\{L^n \mid n \in \mathbb{N}\}$$

nazywamy *iteracją* języka L.

Definicja 1.5 Klasa *języków regularnych* (nad $\mathcal{A}$) to najmniejsza rodzina języków $\mathcal{R}$, spełniająca warunki:

- $\varnothing \in \mathcal{R}$;
- $\{\varepsilon\} \in \mathcal{R}$;
- $\{a\} \in \mathcal{R}$, dla wszystkich $a \in \mathcal{A}$;
- $L_1 \cdot L_2 \in \mathcal{R}$, dla wszystkich $L_1, L_2 \in \mathcal{R}$;
- $L_1 \cup L_2 \in \mathcal{R}$, dla wszystkich $L_1, L_2 \in \mathcal{R}$;
- $L^* \in \mathcal{R}$, dla wszystkich $L \in \mathcal{R}$.

Inaczej mówiąc, klasa $\mathcal{R}$ zawiera najprostsze możliwe języki i jest

zamknięta ze względu na operacje sumy, składania i iteracji. Języki regularne są zwykle reprezentowane za pomocą *wyrażeń regularnych*, które definiujemy tak:

- „$\varnothing$” jest wyrażeniem regularnym;
- „ε” jest wyrażeniem regularnym;
- jeśli $a \in \mathcal{A}$, to „a” jest wyrażeniem regularnym;
- jeśli α i β są wyrażeniami regularnymi, to $\alpha \cup \beta$ i $\alpha \cdot \beta$ są wyrażeniami regularnymi.
- jeśli α jest wyrażeniem regularnym, to α^* jest wyrażeniem regularnym.

Wyrażeniu regularnemu α odpowiada język regularny L_α, określony tak: $L_\varnothing = \varnothing$, $L_\varepsilon = \{\varepsilon\}$, $L_a = \{a\}$, $L_{\alpha\cup\beta} = L_\alpha \cup L_\beta$, $L_{\alpha\cdot\beta} = L_\alpha \cdot L_\beta$, $L_{\alpha^*} = L_\alpha^*$. Jeśli przyjmiemy kilka konwencji notacyjnych, to będziemy mogli w ogóle nie odróżniać języka od definiującego go wyrażenia:

- Jeśli $w \in \mathcal{A}^*$, to zamiast $\{w\}$ piszemy po prostu w;
- Jeśli $a \in \mathcal{A}$, to literę a utożsamiamy ze słowem jednoliterowym.

Na przykład aL oznacza język $\{aw \mid w \in L\}$. Oczywiście litera a, słowo jednoliterowe a i język a, to trzy różne rzeczy, ale zwykle z kontekstu wynika, którą z nich mamy na myśli.

Automaty skończone

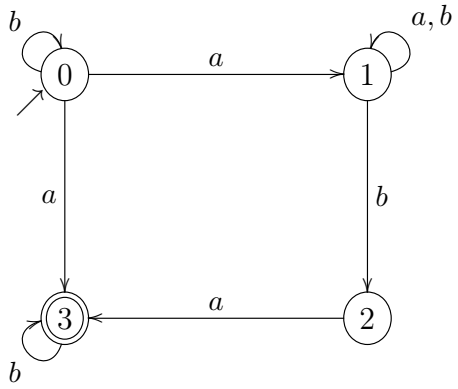
Niedeterministyczny *automat skończony* definiujemy jako krotkę: $\mathcal{M} = \langle Q, \mathcal{A}, \delta, q_0, F\rangle$, gdzie:

- $\mathcal{A}$ jest skończonym alfabetem;
- Q jest skończonym zbiorem *stanów*;
- $q_0 \in Q$ jest stanem *początkowym*;
- $F \subseteq Q$ jest zbiorem stanów *końcowych*;
- $\delta \subseteq (Q \times \mathcal{A}) \times Q$ jest *relacją przejścia*.

Automat jest *deterministyczny*, gdy δ jest funkcją:

$$\delta : (Q \times \mathcal{A}) \to Q.$$

Automat można przedstawiać jako graf, którego wierzchołki to stany. Stany q i p są połączone krawędzią o etykiecie a, gdy trójka $\langle q, a, p\rangle$ jest w relacji δ. Na obrazku 1 mamy przykład automatu o zbiorze stanów $\{0, 1, 2, 3\}$, nad alfabetem $\{a, b\}$, o relacji przejścia złożonej z trójek: $\langle 0, b, 0\rangle$, $\langle 0, a, 1\rangle$, $\langle 0, a, 3\rangle$, $\langle 1, a, 1\rangle$, $\langle 1, b, 1\rangle$, $\langle 1, b, 2\rangle$, $\langle 2, a, 3\rangle$, $\langle 3, b, 3\rangle$. Stanem początkowym naszego automatu jest 0, a stanem końcowym 3.



Rysunek 1: Taki sobie automat

Działanie automatu należy sobie wyobrażać tak: W stanie q automat czyta z wejścia literę a i przechodzi do nowego stanu p, spełniającego warunek $\langle q, a, p\rangle \in \delta$. Zapisujemy to tak:

$$\mathcal{M} \vdash q \xrightarrow{a} p,$$

albo tak:

$$q \xrightarrow{a}_{\mathcal{M}} p,$$

albo po prostu tak:

$$q \xrightarrow{a} p.$$

Relację $\xrightarrow{a}$ uogólniamy następująco: jeżeli $w = a_1 a_2 \dots a_n$, to $\mathcal{M} \vdash q \xrightarrow{w} p$ oznacza, że istnieje ciąg przejść:

$$q = s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \cdots \xrightarrow{a_{n-1}} s_{n-1} \xrightarrow{a_n} s_n = p.$$

Taki ciąg nazywamy *obliczeniem* automatu.[1]

Automat $\mathcal{M}$ *akceptuje* słowo w, wtedy i tylko wtedy, gdy $q_0 \stackrel{w}{\longrightarrow\!\!\!\!\!\rightarrow} q$, dla pewnego $q \in F$. Na przykład automat na rysunku 1 akceptuje słowa *baba* i *abba*, ale nie akceptuje słowa *aabb* ani słowa *baab*. Obliczenie akceptujące słowo *baba* wygląda tak:

$$0 \stackrel{b}{\longrightarrow} 0 \stackrel{a}{\longrightarrow} 1 \stackrel{b}{\longrightarrow} 2 \stackrel{a}{\longrightarrow} 3.$$

Uwaga: w ogólności może być wiele obliczeń realizujących relację $q_0 \stackrel{w}{\longrightarrow\!\!\!\!\!\rightarrow} q$, ale jeśli automat jest deterministyczny, to co najwyżej jedno. *Język akceptowany* przez $\mathcal{M}$ to język:

$$L(\mathcal{M}) = \{w \in \mathcal{A}^* \mid \mathcal{M} \text{ akceptuje } w\}.$$

Automaty skończone akceptują języki regularne

Załóżmy, że mamy automat $\mathcal{M} = \langle Q, \mathcal{A}, \delta, q_0, F\rangle$, i że $Q = \{q_0, \ldots, q_n\}$. Chcemy wyznaczyć język $L(\mathcal{M})$. W tym celu wygodnie jest rozważać n odmian automatu $\mathcal{M}$, różniących się stanami początkowymi. Niech $\mathcal{M}_i = \langle Q, \mathcal{A}, \delta, q_i, F\rangle$ i niech $L_i = L(\mathcal{M}_i)$, dla $i = 1, \ldots, n$. Oczywiście $L(\mathcal{M}) = L_0$.

Dalej niech $A_{ij} = \{a \in \mathcal{A} \mid q_i \stackrel{a}{\longrightarrow} q_j\}$. Nietrudno zauważyć, że nasze języki spełniają równania:

$$\begin{array}{lll} L_i = & A_{i0}L_0 \cup A_{i1}L_1 \cup \cdots \cup A_{in}L_n & \text{gdy } q_i \notin F; \\ L_i = & A_{i0}L_0 \cup A_{i1}L_1 \cup \cdots \cup A_{in}L_n \cup \varepsilon & \text{gdy } q_i \in F. \end{array}$$

Lemat 1.6 *Jeśli $A, B \subseteq \mathcal{A}^*$ oraz $\varepsilon \notin A$, to istnieje dokładnie jeden język $L \subseteq \mathcal{A}^*$ spełniający warunek:*

$$L = AL \cup B,$$

a mianowicie $L = A^*B$. Inaczej mówiąc, równanie $L = AL \cup B$ ma wtedy dokładnie jedno rozwiązanie $L = A^*B$.

Dowód: Sprawdzenie, że $L = A^*B$ jest rozwiązaniem równania jest łatwe:

$$AL \cup B = A(A^*B) \cup B = AA^*B \cup B = (AA^* \cup \varepsilon)B = A^*B = L.$$

Niech więc L będzie dowolnym językiem spełniającym $L = AL \cup B$. Przez indukcję pokażemy, że dla dowolnego k zachodzi $A^kB \subseteq L$. Stąd zaś wynika $A^*B \subseteq L$.

Dla $k = 0$ mamy $A^0B = B \subseteq AL \cup B = L$. Załóżmy więc, że $A^kB \subseteq L$. Wtedy

$$A^{k+1}B = A(A^kB) \subseteq AL \subseteq AL \cup B = L.$$

Aby pokazać, że $L \subseteq A^*B$, przypuśćmy, że $w \in L$. Przez indukcję ze względu na długość w pokażemy, że $w \in A^*B$. Załóżmy więc, że każde słowo v z języka L, którego długość jest mniejsza od $|w|$, należy także do A^*B. Skoro $w \in L = AL \cup B$, to albo $w \in AL$ albo $w \in B$. Jeśli $w \in B$, to $w \in A^*B$, i dobrze. Niech więc $w \in AL$, czyli $w = uv$ dla pewnych $u \in A$ i $v \in L$. Na dodatek $|v| < |w|$ (bo $u \in A$ oznacza, że $u \neq \varepsilon$). Z założenia indukcyjnego $v \in A^*B$, a stąd $w = uv \in AA^*B \subseteq A^*B$ i też jest dobrze. □

[1] Czasem obliczeniami nazywa się tylko te ciągi, które zaczynają się w q_0.

Uwaga: Jeśli $\varepsilon \in A$, to język $L = A^*B$ jest *najmniejszym* rozwiązaniem równania $L = AL \cup B$ (czyli najmniejszym punktem stałym operatora $L \mapsto AL \cup B$).

Wniosek 1.7 *Niech $\varepsilon \notin A_{ij}$, dla $i, j = 0, \ldots, n$. Wtedy istnieje dokładnie jedno rozwiązanie $L_0, \ldots, L_n$ układu rownań*

$$\begin{aligned} L_0 = &\quad A_{00}L_0 \cup A_{01}L_1 \cup \cdots \cup A_{0n}L_n \cup B_0 \\ L_1 = &\quad A_{10}L_0 \cup A_{11}L_1 \cup \cdots \cup A_{1n}L_n \cup B_1 \\ &\quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ L_n = &\quad A_{n0}L_0 \cup A_{n1}L_1 \cup \cdots \cup A_{nn}L_n \cup B_n, \end{aligned}$$

i wszystkie te języki są regularne, jeśli A_{ij} i B_i są regularne.

Dowód: Przypadek $n = 0$ wynika z poprzedniego lematu. Dla $n > 0$ postępujemy przez indukcję. W tym celu należy zauważyć, że nasz układ równań ma te same rozwiązania co układ złożony z równania

$$L_0 = A_{00}^*(A_{01}L_1 \cup \cdots \cup A_{0n}L_n \cup B_0)$$

i równań

$$\begin{aligned} L_1 = &\quad (A_{11} \cup A_{10}A_{00}^*A_{01})L_1 \cup \cdots \cup (A_{1n} \cup A_{10}A_{00}^*A_{0n})L_n \cup (B_1 \cup A_{10}A_{00}^*B_0) \\ &\quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ L_n = &\quad (A_{11} \cup A_{n0}A_{00}^*A_{01})L_1 \cup \cdots \cup (A_{1n} \cup A_{n0}A_{00}^*A_{0n})L_n \cup (B_1 \cup A_{n0}A_{00}^*B_0) \end{aligned}$$

Istotnie, przypuśćmy, że języki $L_0, \ldots, L_1$ stanowią rozwiązanie „starego" układu. Wtedy równość

$$L_0 = A_{00}^*(A_{01}L_1 \cup \cdots \cup A_{0n}L_n \cup B_0)$$

wynika z Lematu 1.6, bo $\varepsilon \notin A_{00}$. Wstawiając to do pozostałych równań otrzymujemy nowy układ.[2] Podobnie łatwo można sprawdzić, że każde rozwiązanie nowego układu jest też rozwiązaniem starego układu.

Pozostaje zauważyć, że współczynniki przy L_i w nowym układzie równań nadal nie zawierają słowa pustego i zastosować założenie indukcyjne, bo równań jest o jedno mniej. □

Twierdzenie 1.8 *Dla dowolnego automatu skończonego $\mathcal{M}$, język $L(\mathcal{M})$ jest regularny.*

Dowód: Jak już zauważyliśmy, język $L(\mathcal{M})$ jest pierwszą współrzędną rozwiązania pewnego układu równań o współczynnikach $A_{ij} = \{a \in \mathcal{A} \mid q_i \xrightarrow{a} q_j\}$. Oczywiście $\varepsilon \notin A_{ij}$, zatem teza wynika z Wniosku 1.7. □

Przykład 1.9 Dla automatu z obrazka mamy układ równań:

$$\begin{aligned} L_0 = &\quad bL_0 \cup aL_1 \cup aL_3 \\ L_1 = &\quad (a \cup b)L_1 \cup bL_2 \\ L_2 = &\quad aL_3 \\ L_3 = &\quad bL_3 \cup \varepsilon \end{aligned}$$

Po rozwiązaniu metodą eliminacji niewiadomych, dostaniemy $L_0 = b^*a((a \cup b)^*ba \cup \varepsilon)b^*$.

[2] W istocie przekształcamy układ równań stosując dobrze znaną metodę eliminacji niewiadomych.

Twierdzenie 1.10 *Dla dowolnego języka regularnego L istnieje automat skończony $\mathcal{M}$, który go akceptuje.*

Dowód: Przypomnijmy, że język jest regularny, wtedy i tylko wtedy, gdy:

- jest równy $\varnothing$ lub ε, lub jednoliterowy;
- jest konkatenacją dwóch języków regularnych;
- jest sumą dwóch języków regularnych;
- jest iteracją języka regularnego.

Jeśli L jest pusty lub $L = \varepsilon$, to wystarczy automat jednostanowy, gdzie F jest w pierwszym przypadku puste, w drugim nie. Dla języka złożonego z jednej litery a potrzeba dwóch stanów i przejścia $\langle q_0, a, q\rangle$, gdzie $F = \{q\}$.

Załóżmy, że $L = L_1 \cdot L_2$, gdzie $L_1 = L(\mathcal{M}_1)$ i $L_2 = L(\mathcal{M}_2)$. Niech $\mathcal{M}_1 = \langle Q_1, \mathcal{A}, \delta_1, q_0^1, F_1\rangle$ oraz $\mathcal{M}_2 = \langle Q_2, \mathcal{A}, \delta_2, q_0^2, F_2\rangle$. Wtedy $L = L(\mathcal{M})$, dla pewnego automatu

$$\mathcal{M} = \langle Q_1 \cup Q_2, \mathcal{A}, \delta, q_0^1, F\rangle,$$

gdzie relacja przejścia jest taka:

$$\delta = \delta_1 \cup \delta_2 \cup \{\langle f, a, q\rangle \mid f \in F_1 \wedge \langle q_0^2, a, q\rangle \in \delta_2\}.$$

Nasz automat jest więc sumą automatów $\mathcal{M}_1$ i $\mathcal{M}_2$, w której wszystkie krawędzie wychodzące ze stanu początkowego $\mathcal{M}_2$ zdublowano krawędziami wychodzącymi ze stanów końcowych $\mathcal{M}_1$. Automat $\mathcal{M}$ naśladuje więc najpierw automat $\mathcal{M}_1$ a po osiągnięciu stanu końcowego może przejść do symulacji automatu $\mathcal{M}_2$. Chcemy, żeby $\mathcal{M}$ akceptował słowa postaci uv, dla których istnieją przejścia $q_0^1 \overset{u}{\longrightarrow\!\!\!\!\!\rightarrow} f \in F_1$ w automacie $\mathcal{M}_1$, oraz przejścia $q_0^2 \overset{v}{\longrightarrow\!\!\!\!\!\rightarrow} f' \in F_2$ w automacie $\mathcal{M}_2$. Dlatego definiujemy $F = F_2$ gdy $q_0^2 \notin F_2$, ale musimy przyjąć $F = F_1 \cup F_2$ gdy $q_0^2 \in F_2$, tj. gdy $\mathcal{M}_2$ akceptuje słowo puste.

Niech teraz $L = L_1 \cup L_2$, i znowu $L_1 = L(\mathcal{M}_1)$ i $L_2 = L(\mathcal{M}_2)$, dla pewnych automatów $\mathcal{M}_1 = \langle Q_1, \mathcal{A}, \delta_1, q_0^1, F_1\rangle$ oraz $\mathcal{M}_2 = \langle Q_2, \mathcal{A}, \delta_2, q_0^2, F_2\rangle$. Wtedy $L = L(\mathcal{M})$, gdzie

$$\mathcal{M} = \langle Q_1 \cup Q_2 \cup \{q_0\}, \mathcal{A}, \delta, q_0, F\rangle.$$

Relacja przejścia jest taka:

$$\delta = \delta_1 \cup \delta_2 \cup \{\langle q_0, a, q\rangle \mid \langle q_0^1, a, q\rangle \in \delta_1\} \cup \{\langle q_0, a, q\rangle \mid \langle q_0^2, a, q\rangle \in \delta_2\},$$

a zbiorem stanów końcowych jest

$F_1 \cup F_2$, gdy $q_0^1 \notin F_1$ i $q_0^2 \notin F_2$;

$F_1 \cup F_2 \cup \{q_0\}$, w przeciwnym przypadku.

Ten automat na początku wybiera, czy chce symulować zachowanie $\mathcal{M}_1$, czy $\mathcal{M}_2$, a potem robi, co postanowił.

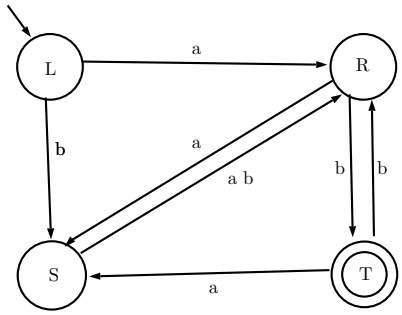
Pozostaje przypadek, gdy $L = L_1^*$. Niech $L_1 = L(\mathcal{M}_1)$, gdzie $\mathcal{M}_1 = \langle Q_1, \mathcal{A}, \delta_1, q_0^1, F_1 \rangle$. Język L jest akceptowany przez automat $\mathcal{M} = \langle Q_1, \mathcal{A}, \delta, q_0, F \rangle$, gdzie $F = F_1 \cup \{q_0\}$, z relacją przejścia

$$\delta = \delta_1 \cup \{\langle f, a, q \rangle \mid f \in F \wedge \langle q_0^1, a, q \rangle \in \delta_1\}.$$

Tę konstrukcję należy sobie wyobrażać jako „doklejenie" stanów końcowych automatu do jego stanu początkowego. □

2 Własności języków regularnych

Na rysunku 2 mamy automat niedeterministyczny, który akceptuje na przykład słowo *bbaab*. Obliczenie akceptujące jest takie: $\mathrm{L} \xrightarrow{b} \mathrm{S} \xrightarrow{b} \mathrm{R} \xrightarrow{a} \mathrm{S} \xrightarrow{a} \mathrm{R} \xrightarrow{b} \mathrm{T}$.



Rysunek 2: Jeszcze jeden automat

W tym obliczeniu mamy pętlę: automat dwukrotnie wchodzi w stan S. Jeśli ją usuniemy, to otrzymamy obliczenie $\mathrm{L} \xrightarrow{b} \mathrm{S} \xrightarrow{a} \mathrm{R} \xrightarrow{b} \mathrm{T}$, akceptujące słowo *bab*. Ale możemy też pętlę powtarzać: wtedy otrzymamy obliczenia akceptujące słowa *bbabaab*, *bbababaab*, itd. Poniższe twierdzenie jest uogólnieniem tej obserwacji.

Twierdzenie 2.1 (Lemat o pompowaniu)

Dla dowolnego języka regularnego L istnieje stała $n \in \mathbb{N}$ o następującej własności:

Jeżeli $w \in L$ oraz $|w| \geq n$, to słowo w można przedstawić w postaci $w = uvz$, gdzie

- $v \neq \varepsilon$;
- $|uv| \leq n$;
- $\forall i \in \mathbb{N}(uv^i z \in L)$ *(w szczególności $uz \in L$).*

Stałą n można efektywnie wyznaczyć, np. na podstawie automatu akceptującego L.

Dowód: Niech $\mathcal{M} = \langle Q, \mathcal{A}, \delta, q_0, F\rangle$ będzie automatem akceptującym L. Definiujemy n jako liczbę elementów zbioru Q. Weźmy takie $w = a_1 a_2 \cdots a_m \in L$, że $m = |w| \geq n$ i rozpatrzmy obliczenie akceptujące w. Jest to taki ciąg stanów:

$$q_0 = s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \cdots \xrightarrow{a_{m-1}} s_{m-1} \xrightarrow{a_m} s_m \in F.$$

Ponieważ stanów $s_0, s_1, \ldots, s_m$ jest $m + 1 > n$, więc dwa z nich muszą być takie same, tj. $s_i = s_j$ dla pewnych $i < j$. Co więcej, można wybrać liczby i, j tak, że $i, j \leq n$. Rozbicie słowa w na trzy części określamy tak:

- $u = a_1 \cdots a_i$;
- $v = a_{i+1} \cdots a_j$;
- $z = a_{j+1} \cdots a_m$.

Oczywiście $v \neq \varepsilon$, bo $i \neq j$, oraz $|uv| \leq n$, bo $j \leq n$. Trzeci warunek wynika stąd, że ciąg

$$q_0 = s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \cdots \xrightarrow{a_i} s_i \xrightarrow{a_{j+1}} s_{j+1} \xrightarrow{a_{j+2}} \cdots \xrightarrow{a_m} s_m,$$

a także każdy z ciągów

$$q_0 = s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \cdots \xrightarrow{a_i} s_i \xrightarrow{a_{i+1}} s_{i+1} \xrightarrow{a_{i+2}} \cdots \xrightarrow{a_j} s_j \xrightarrow{a_{i+1}} s_{i+1} \xrightarrow{a_{i+2}} \cdots \xrightarrow{a_j} s_j \xrightarrow{a_{i+1}} \cdots$$
$$\cdots \xrightarrow{a_{i+1}} s_{i+1} \xrightarrow{a_{i+1}} \cdots \xrightarrow{a_j} s_j \xrightarrow{a_{j+1}} s_{j+1} \cdots \xrightarrow{a_m} s_m,$$

jest poprawnym obliczeniem akceptującym automatu $\mathcal{M}$. □

Przykład 2.2 Symbol w^R oznacza słowo w napisane od końca, tj. dla dowolnego $i < |w|$ mamy $w^R(i) = w(|w| - i - 1)$. Język $L = \{ww^R \mid w \in \mathcal{A}^*\}$ nie jest regularny, bo dla dostatecznie dużego N, słowo $0^N 110^N$ nie daje się przedstawić w postaci $0^N 110^N = uvz$, spełniającej wymagania lematu o pompowaniu. Prefiks uv składałby się wtedy z samych zer (bo jest krótki) i przyjmując, że $|v| = k$, mielibyśmy np. $0^{N+2k}110^N \in L$.

$0^{N+ik}110^N \in L$, dla $i = -1, 0, 1, 2, \ldots$

Determinizacja

Twierdzenie 2.3 *Każdy język regularny jest akceptowany przez automat deterministyczny.*

Dowód: Niech $\mathcal{M} = \langle Q, \mathcal{A}, \delta, q_0, F\rangle$ będzie dowolnym automatem skończonym. Konstruujemy deterministyczny automat $\mathcal{M}' = \langle Q', \mathcal{A}, \delta', q_0', F'\rangle$, akceptujący ten sam język. Zbiorem stanów $\mathcal{M}'$ jest zbiór potęgowy $\mathsf{P}(Q)$. Stan początkowy to singleton $q_0' = \{q_0\}$. Zbiorem stanów końcowych jest rodzina tych podzbiorow Q, które mają niepuste przecięcia z F:

$$F' = \{Z \subseteq Q \mid Z \cap F \neq \varnothing\}.$$

Wreszcie funkcja przejścia jest określona tak:

$$\delta'(Z, a) = \{q \in Q \mid \exists p \in Z (p \xrightarrow{a} q)\}.$$

Powyższa konstrukcja jest oparta na takim pomyśle: warunek $\mathcal{M}' \vdash \{q_0\} \overset{w}{\longrightarrow\!\!\!\!\!\rightarrow} Z$ oznacza, że zbiór Z składa się ze wszystkich stanów osiągalnych z q_0 za pomocą relacji $\overset{w}{\longrightarrow\!\!\!\!\!\rightarrow}_{\mathcal{M}}$. Dokładniej:

$$(*) \qquad \mathcal{M}' \vdash q_0' \overset{w}{\longrightarrow\!\!\!\!\!\rightarrow} Z_w, \quad \text{gdzie} \quad Z_w = \{q \in Q \mid \mathcal{M} \vdash q_0 \overset{w}{\longrightarrow\!\!\!\!\!\rightarrow} q\}.$$

Warunku $(*)$ łatwo można dowieść przez indukcję ze względu na $|w|$, korzystając z równości $Z_{wa} = \delta'(Z_w, a)$. A zatem:

$$\begin{aligned} L(\mathcal{M}') &= \{w \in \mathcal{A}^* \mid \mathcal{M}' \vdash q_0' \overset{w}{\longrightarrow\!\!\!\!\!\rightarrow} Z, \text{ dla pewnego } Z \in F'\} \\ &= \{w \in \mathcal{A}^* \mid Z_w \in F'\} \\ &= \{w \in \mathcal{A}^* \mid Z_w \cap F \neq \varnothing\} \\ &= \{w \in \mathcal{A}^* \mid \mathcal{M} \vdash q_0 \overset{w}{\longrightarrow\!\!\!\!\!\rightarrow} q, \text{ dla pewnego } q \in F\} \\ &= L(\mathcal{M}). \end{aligned}$$

□

Przykład 2.4 Konstrukcja z dowodu twierdzenia 2.3 zwiększa wykładniczo liczbę stanów automatu. Nie da się tego poprawić. Rozpatrzmy język $L_k = \{0,1\}^*1\{0,1\}^{k-1}$, złożony z tych słów, w których symbol 1 znajduje się na k-tym miejscu od końca. Język L_k można rozpoznawać automatem niedeterministycznym o $k+1$ stanach, ale automat deterministyczny dla tego języka musi mieć co najmniej 2^k stanów.

Istotnie, przypuśćmy, że automat $\mathcal{M}$ akceptuje język L_k i niech w i v będą dwoma różnymi słowami długości k (powiedzmy, że $w(i) = 1$ ale $v(i) = 0$). Wtedy $w0^i \in L_k$, ale $v0^i \notin L_k$. Po przeczytaniu słowa v automat musi znaleźć się w innym stanie niż po przeczytaniu słowa w, inaczej słowo $v0^i$ też zostałoby zaakceptowane. Skoro każde słowo długości k prowadzi do innego stanu, liczba stanów automatu musi być co najmniej taka jak liczba takich słów. □

Wniosek 2.5 *Następujące warunki są równoważne:*

- *L jest językiem regularnym.*
- *L jest akceptowany przez pewien automat skończony.*
- *L jest akceptowany przez pewien deterministyczny automat skończony.*

Przez $-L$ oznaczamy język $\mathcal{A}^* - L$ (dopełnienie do $\mathcal{A}^*$).

Wniosek 2.6 *Jeśli język L jest regularny, to $-L$ też jest regularny*

Dowód: Jeśli deterministyczny automat $\mathcal{M} = \langle Q, \mathcal{A}, \delta, q_0, F\rangle$ akceptuje język L, to wystarczy zamienić F na $Q - F$ i otrzymamy automat akceptujący język $-L$. □

Klasa języków regularnych jest też zamknięta ze względu na różne inne działania, np.:

Fakt 2.7

1. *Jeśli języki L_1 i L_2 są regularne, to $L_1 \cap L_2$ jest regularny.*
2. *Jeśli L jest regularny, to $L^R = \{w^R \mid w \in L\}$ też jest regularny.*

Dowód: (1) Niech $\mathcal{M}_1 = \langle Q_1, \mathcal{A}, \delta_1, q_0^1, F_1\rangle$ oraz $\mathcal{M}_2 = \langle Q_2, \mathcal{A}, \delta_2, q_0^2, F_2\rangle$ będą automatami akceptującymi odpowiednio L_1 i L_2. Automat

$$\mathcal{M} = \langle Q_1 \times Q_2, \mathcal{A}, \delta, \langle q_0^1, q_0^2\rangle, F_1 \times F_2\rangle,$$

akceptujący iloczyn, ma relację przejścia określoną tak:

$$\delta = \{\langle\langle q_1, q_2\rangle, a, \langle q_3, q_4\rangle\rangle \mid \langle q_1, a, q_3\rangle \in \delta_1 \wedge \langle q_2, a, q_4\rangle \in \delta_2\}.$$

Automat $\mathcal{M}$ realizuje równolegle symulację obu automatów $\mathcal{M}_1$ i $\mathcal{M}_2$.

(2) Indukcja ze względu na długość wyrażenia regularnego definiującego L. □

Ilorazy

Zdefiniujemy teraz pojęcie lewo- i prawostronnego *ilorazu* języków.

$$\begin{aligned} L_1\backslash L_2 = &\quad \{x \in \mathcal{A}^* \mid \exists y \in L_2(y \cdot x \in L_1)\} \quad \text{(iloraz lewostronny)} \\ L_1/L_2 = &\quad \{x \in \mathcal{A}^* \mid \exists y \in L_2(x \cdot y \in L_1)\} \quad \text{(iloraz prawostronny)} \end{aligned}$$

W szczególności:

$$L\backslash w = \{x \in \mathcal{A}^* \mid wx \in L\};$$
$$L/w = \{x \in \mathcal{A}^* \mid xw \in L\}.$$

Lemat 2.8

- $L\backslash\varepsilon = L$;
- $(L\backslash w)\backslash v = L\backslash wv$ *oraz* $(L/w)/v = L/vw$;
- $L\backslash R = \bigcup\{L\backslash w \mid w \in R\}$;
- *Warunki* $\varepsilon \in L\backslash w$ *i* $w \in L$ *są równoważne.*

Dowód: Łatwy. □

Twierdzenie 2.9 *Język jest regularny wtedy i tylko wtedy, gdy ma tylko skończenie wiele różnych ilorazów lewostronnych.*

Dowód: ($\Longrightarrow$) Niech $L = L(\mathcal{M})$ dla pewnego deterministycznego $\mathcal{M} = \langle Q, \mathcal{A}, \delta, q_0, F\rangle$. Przyjmując $Q = \{q_0, \ldots, q_n\}$ możemy, jak już robiliśmy to wcześniej, rozważać automaty $\mathcal{M}_i = \langle Q, \mathcal{A}, \delta, q_i, F\rangle$ i języki $L_i = L(\mathcal{M}_i)$, dla $i = 1, \ldots, n$.

Jeśli teraz $q_0 \xrightarrow{w}\!\!\!\!\rightarrow q_i$, to $L_i = \{v \in \mathcal{A}^* \mid wv \in L\} = L\backslash w$, bo maszyna jest deterministyczna. Zatem dla dowolnego słowa w, iloraz $L\backslash w$ jest jednym z języków L_i. Ponieważ dla dowolnego S, zachodzi $L\backslash S = \bigcup\{L\backslash w \mid w \in S\}$, a takich sum jest tylko skończenie wiele, więc L ma tylko skończenie wiele ilorazów.

($\Longleftarrow$) Definiujemy deterministyczny automat $\mathcal{M} = \langle Q, \mathcal{A}, \delta, q_0, F\rangle$, gdzie:

- Q jest zbiorem wszystkich ilorazów języka L.
- $\delta(L', a) = L'\backslash a$, dla $L' \in Q$ i $a \in \mathcal{A}\}$.
- $q_0 = L$.
- $F = \{L' \in Q \mid \varepsilon \in L'\}$.

Łatwo widzieć, że dla dowolnego słowa w zachodzi $\mathcal{M} \vdash L \stackrel{w}{\longrightarrow\!\!\!\!\!\rightarrow} L\backslash w$. Zatem $w \in L(\mathcal{M})$ wtedy i tylko wtedy, gdy $\varepsilon \in L\backslash w$, czyli wtedy i tylko wtedy, gdy $w \in L$. □

Własności ilorazów lewostronnych i prawostronnych są analogiczne z powodu symetrii. Dlatego powyższe twierdzenie zachodzi też dla ilorazów prawostronnych. Wystarczy w tym celu zauważyć, że: $L/w = (L^R\backslash w^R)^R$ oraz $L\backslash w = (L^R/w^R)^R$.

Przykład 2.10 Język $L = \{0^n 1^n \mid n \in \mathbb{N}\}$ nie jest regularny, bo każdy iloraz $L\backslash 0^n$ jest inny.

3 Języki bezkontekstowe

Gramatyką bezkontekstową nad alfabetem $\mathcal{A}$ nazywamy twór postaci

$$G = \langle \mathcal{A}, \mathcal{N}, P, \xi_0 \rangle,$$

gdzie $\mathcal{N}$ jest skończonym alfabetem, $\xi_0 \in \mathcal{N}$ jest *symbolem początkowym*, a *produkcje* (czyli *reguły*) ze skończonego zbioru P mają kształt

$$\xi \Rightarrow v, \quad \text{gdzie } \xi \in \mathcal{N} \text{ oraz } v \in (\mathcal{A} \cup \mathcal{N})^*.$$

Gramatyki czasem zapisuje się w stylu notacji Backusa-Naura, np.

$$\xi_0 ::= a\xi_0 a \mid b\xi_0 b \mid a \mid b \mid \varepsilon$$

przedstawia gramatykę z jednym nieterminałem ξ_0 i produkcjach

$$\xi_0 \Rightarrow a\xi_0 a, \quad \xi_0 \Rightarrow b\xi_0 b, \quad \xi_0 \Rightarrow a, \quad \xi_0 \Rightarrow b, \quad \xi_0 \Rightarrow \varepsilon. \tag{*}$$

Pomocnicze symbole $\xi \in \mathcal{N}$ nazywamy *niekońcowymi* (lub *nieterminalnymi*), w odróżnieniu od *końcowych* czyli *terminalnych* symboli alfabetu $\mathcal{A}$. Piszemy $G \vdash w \to u$, albo $w \to_G u$ gdy w P jest taka reguła $\xi \Rightarrow v$, że $w = w_1 \xi w_2$, $u = w_1 v w_2$. Gdy wiadomo o jaką gramatykę chodzi, można pisać po prostu $w \to u$. Relacja $\twoheadrightarrow_G$ jest domknięciem przechodnio-zwrotnym relacji $\to_G$, tj. $w \twoheadrightarrow_G u$ (zapisywane też jako $G \vdash w \twoheadrightarrow u$) zachodzi, gdy istnieje ciąg:

$$w = w_0 \to_G w_1 \to_G \cdots \to_G w_{k-1} \to_G w_k = u,$$

nazywany *wyprowadzeniem* (*wywodem*, *redukcją*) w gramatyce G.

Przykład 3.1 Gramatyka o produkcjach

$$\xi_0 ::= \xi_a\xi_b \mid \xi_b\xi_a; \qquad \xi_a ::= \alpha\xi_a\alpha \mid a; \qquad \xi_b ::= \alpha\xi_b\alpha \mid b; \qquad \alpha ::= a \mid b,$$

generuje między innymi słowo *ababbbba*. Przykładowe wyprowadzenie może być takie:

$$\xi_0 \to \xi_a\xi_b \to \alpha\xi_a\alpha\xi_b \to \alpha\alpha\xi_a\alpha\alpha\xi_b \to \alpha\alpha a\alpha\alpha\xi_b \to \alpha\alpha a\alpha\alpha\alpha\xi_b\alpha \to \alpha\alpha a\alpha\alpha\alpha ba\alpha \to$$
$$\alpha\alpha a\alpha\alpha\alpha ba\alpha \to aba\alpha\alpha\alpha ba\alpha \to abab\alpha\alpha ba\alpha \to ababba\alpha ba\alpha \to ababbbba\alpha \to ababbbba.$$

Język *generowany* przez gramatykę G, to język

$$L(G) = \{w \in \mathcal{A}^* \mid G \vdash \xi_0 \twoheadrightarrow w\}.$$

Języki generowane przez gramatyki bezkontekstowe są nazywane językami *bezkontekstowymi*.

Nietrudno zauważyć, że gramatyka (*) generuje język wszystkich palindromów. A ta gramatyka opisuje mały kawałeczek języka polskiego:

⟨zdanie⟩ ::= ⟨grupa podmiotu⟩⟨grupa orzeczenia⟩;
⟨grupa podmiotu⟩ ::= ⟨przydawka⟩⟨grupa podmiotu⟩ | ⟨podmiot⟩;
⟨grupa orzeczenia⟩ ::= ⟨dopełnienie⟩⟨orzeczenie⟩;
⟨orzeczenie⟩ ::= ⟨czasownik⟩;
⟨podmiot⟩ ::= ⟨rzeczownik⟩;
⟨przydawka⟩ ::= ⟨przymiotnik⟩;
⟨dopełnienie⟩ ::= ⟨przysłówek⟩;
⟨rzeczownik⟩ ::= *pies* | *kot*;
⟨czasownik⟩ ::= *szczeka* | *śpi*;
⟨przymiotnik⟩ ::= *czarny* | *mały*;
⟨przysłówek⟩ ::= *głośno* | *smacznie*.

► Język generowany przez gramatykę bezkontekstową można uważać za najmniejszy punkt stały pewnego operatora[3] (dokładniej: jedną ze współrzędnych najmniejszego punktu stałego pewnego wielowymiarowego operatora na językach). Zilustrujemy to na przykładzie. Niech G ma produkcje:

$$\xi ::= \eta\xi \mid a, \qquad \eta ::= \varepsilon \mid a\eta b$$

Niech $L = L(G)$ i niech R będzie językiem generowanym przez zmienioną gramatykę G, w której jako początkowy wybrano symbol η. Łatwo zauważyć, że języki L i R muszą spełniać równania:[4]

$$L = RL \cup a \quad \text{oraz} \quad R = \varepsilon \cup aRb.$$

Nie jest to jednak jedyne rozwiązanie tego układu. Inne rozwiązanie tworzą na przykład języki $L' = \mathcal{A}^*$ i $R' = R$. Dokładniej, rozwiązaniami naszego układu są wszystkie punkty stałe operatora

$$\Phi : P(\mathcal{A}^*) \times P(\mathcal{A}^*) \to P(\mathcal{A}^*) \times P(\mathcal{A}^*)$$

określonego wzorem $\Phi(\langle X, Y\rangle) = \langle YX \cup a, \varepsilon \cup aYb\rangle$.

Fakt 3.2 *Niech $L_0 = R_0 = \varnothing$ i niech $\langle L_{n+1}, R_{n+1}\rangle = \Phi(\langle L_n, R_n\rangle)$, tj. $\langle L_n, R_n\rangle = \Phi^n(\langle\varnothing, \varnothing\rangle)$.*

1. *Jeśli $L_\omega = \bigcup_{n\in\mathbb{N}} L_n$ i $R_\omega = \bigcup_{n\in\mathbb{N}} R_n$, to para $\langle L_\omega, R_\omega\rangle$ jest najmniejszym punktem stałym Φ.*
2. *Dla dowolnego $w \in L$ istnieje takie k, że $w \in L_k$;*
3. *Dla dowolnego $w \in R$ istnieje takie k, że $w \in R_k$.*

[3] Tekst oznaczony ► ··· ◄ można opuścić przy pierwszym czytaniu.
[4] W istocie mamy tutaj $R = \{a^n b^n \mid n \in \mathbb{N}\}$ oraz $L = R^* a$.

Dowód: Zbiór $P(\mathcal{A}^*) \times P(\mathcal{A}^*)$ z uporządkowaniem „po współrzędnych" zadanym przez inkluzję jest kratą zupełną, a przekształcenie Φ jest ciągłe. Zatem część pierwsza wynika z twierdzenia Tarskiego o punkcie stałym. Części drugą i trzecią można pokazać przez indukcję ze względu na długość wyprowadzenia. □

Z powyższego faktu wynika, że $\langle L_\omega, R_\omega \rangle = \langle L, R \rangle$, czyli, że para $\langle L, R \rangle$ jest najmniejszym punktem stałym operatora Φ. ◀

Języki regularne są bezkontekstowe

Fakt 3.3 *Każdy język regularny jest bezkontekstowy.*

Dowód: Niech $L = L(\mathcal{M})$ dla pewnego automatu skończonego $\mathcal{M} = \langle Q, \mathcal{A}, \delta, q_0, F \rangle$, gdzie $Q = \{q_0, \ldots, q_n\}$. Wtedy $L = L(G)$, gdzie gramatyka $G = \langle \mathcal{A}, Q, P, q_0 \rangle$ ma produkcje

$$P = \{q_i \Rightarrow aq_j \mid \mathcal{M} \vdash q_i \stackrel{a}{\longrightarrow\!\!\!\!\!\rightarrow} q_j\} \cup \{q_i \Rightarrow \varepsilon \mid q_i \in F\}.$$

Istotnie, każdemu wyprowadzeniu w G odpowiada pewne obliczenie automatu i na odwrót. □

Gramatyka G jest *prawostronna* wtedy i tylko wtedy, gdy wszystkie produkcje są postaci „$\xi \Rightarrow w\eta$" lub „$\xi \Rightarrow w$", gdzie $w \in \mathcal{A}^*$ oraz $\eta \in \Sigma$. Gramatyka użyta w dowodzie Faktu 3.3 jest prawostronna.

Fakt 3.4 *Jeżeli gramatyka G jest prawostronna, to język $L(G)$ jest regularny,*

Dowód: Równania wyznaczone przez produkcje

$$\xi_i ::= w_1\xi_{n_{i1}} \mid \ldots \mid w_k\xi_{n_{ik}} \mid w_{k+1} \mid \ldots \mid w_m$$

gramatyki prawostronnej mają postać

$$L_i = w_1 L_{i1} \cup \ldots \cup w_k L_{n_{ik}} \cup w_{k+1} \cup \ldots \cup w_m$$

a najmniejsze rozwiązanie układu takich równań tworzą pewne języki regularne. □

Standaryzacja i drzewa

Istotę bezkontekstowości wyraża następujący lemat:

Lemat 3.5 *Jeżeli $wv \twoheadrightarrow_G u$ w pewnej gramatyce bezkontekstowej G, to $u = u_1u_2$ dla pewnych słów u_1 i u_2, takich że $w \twoheadrightarrow_G u_1$ i $v \twoheadrightarrow_G u_2$.*

Dowód: Indukcja ze względu na długość wyprowadzenia. □

Lemat 3.5 należy rozumieć tak: kroki redukcji są wykonywane w różnych częściach słowa niezależnie od siebie. Ale oznacza to, że można je wykonywać w dowolnej kolejności, np. od lewej. Uściślimy teraz tę obserwację.

Mówimy, że wyprowadzenie

$$w = w_0 \to w_1 \to \cdots \to w_{k-1} \to w_k = u,$$

w gramatyce G jest *standardowe*, co zapisujemy jako $w \twoheadrightarrow^L u$, jeżeli dla każdego $i = 0, \ldots, k-1$:

$$w_i = u_i \xi_i v_i \to u_i x_i v_i = w_{i+1}, \text{ oraz } u_i \leq u_{i+1} \text{ dla wszystkich } j \geq i.$$

Inaczej, nigdy nie wykonuje się redukcji najpierw po prawej, potem po lewej.[5] Wyprowadzenie standardowe $\xi_0 \twoheadrightarrow^L w \in \mathcal{A}^*$ nazywamy też *lewostronnym.*

Lemat 3.6 *Jeśli $G \vdash w \twoheadrightarrow v$, to istnieje wyprowadzenie standardowe $G \vdash w \twoheadrightarrow^L v$ i to tej samej długości.*

► **Dowód**: Postępujemy przez indukcję ze względu na długość wyprowadzenia. Wyprowadzenie, które ma nie więcej niż jeden krok, jest oczywiście standardowe. Przypuśćmy więc, że każde wyprowadzenie złożone z n lub mniej kroków można zastąpić wyprowadzeniem standardowym o tej samej długości. Niech teraz wyprowadzenie $w \twoheadrightarrow v \to u$ ma $n+1$ kroków. Pierwsze n kroków można zastąpić wyprowadzeniem standardowym $w \twoheadrightarrow^L v$. Jeśli wyprowadzenie $w \twoheadrightarrow^L v \to u$ nie jest standardowe, to musi być tak:

- Ostatni krok wyprowadzenia $w \twoheadrightarrow^L v$ jest postaci $v' = v_1 \xi v_2 \eta v_3 \to v_1 \xi v_2 y v_3 = v$;
- Redukcja $v \to u$ jest postaci $v = v_1 \xi v_2 y v_3 \to v_1 x v_2 y v_3 = u$.

Zauważmy, że słowo v_3 jest sufiksem wszystkich słów występujących w wyprowadzeniu $w \twoheadrightarrow^L v$, bo redukcja $\eta \to y$, jako ostatnia, musiała zajść najbardziej na prawo. Mamy więc $w = w' v_3$ i standardowe wyprowadzenie $w' \twoheadrightarrow^L v_1 \xi v_2 \eta \to v_1 \xi v_2 y$.

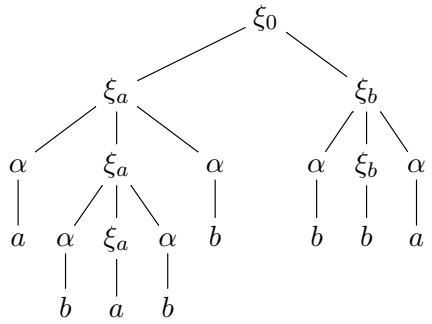
Rozpatrzmy teraz redukcję $w' \twoheadrightarrow^L v_1 \xi v_2 \eta \to v_1 x v_2 \eta$. Ta redukcja jest długości n, więc istnieje też standardowe wyprowadzenie $w' \twoheadrightarrow^L v_1 x v_2 \eta$. Ponieważ η jest ostatnim symbolem słowa $v_1 x v_2 \eta$, więc wyprowadzenie $w' \twoheadrightarrow^L v_1 x v_2 \eta \to v_1 x v_2 y$ też jest standardowe. To samo dotyczy wyprowadzenia $w = w' v_3 \twoheadrightarrow^L v_1 x v_2 \eta v_3 \to v_1 x v_2 y v_3 = u$. Oczywiście długość tego wyprowadzenia jest równa n. □ ◄

Przez *drzewo wywodu* słowa w w gramatyce G (por. rysunek 3) rozumiemy każde skończone drzewo etykietowane symbolami z $(\mathcal{A} \cup \mathcal{N})^*$ w taki sposób:

- Korzeń ma etykietę ξ_0.
- Etykiety liści są symbolami terminalnymi i czytane od lewej do prawej (w porządku leksykograficznym ścieżek) dają słowo w.
- Jeśli wierzchołek o etykiecie nieterminalnej ξ ma k córek o etykietach $a_1, \ldots, a_k$ (czytanych od lewej do prawej), to „$\xi \Rightarrow a_1 \ldots a_k$" jest produkcją gramatyki G.

Drzewo wywodu dla w istnieje wtedy i tylko wtedy, gdy $w \in L(G)$. Każde wyprowadzenie $\xi_0 \twoheadrightarrow w$ wyznacza pewne drzewo wywodu. Odwrotnie, jeśli istnieje drzewo wywodu słowa w, to istnieje też wyprowadzenie tego słowa w gramatyce. Jedno drzewo może odpowiadać różnym wyprowadzeniom, ale tylko jednemu wyprowadzeniu lewostronnemu. (Ale dla jednego słowa może istnieć więcej niż jedno drzewo wywodu.)

[5] Wyprowadzenie z przykładu 3.1 nie jest standardowe.



Rysunek 3: Drzewo wywodu dla wyprowadzenia z przykładu 3.1

Lemat 3.7 *Jeśli prawa strona każdej produkcji gramatyki G ma długość co najwyżej p, to drzewo wywodu dla słowa w o długości $|w| > p^m$ ma wysokość większą niż m.*

Dowód: Drzewo o stopniu rozgałęzienia nie przekraczającym p i wysokości nie przekraczającej m ma co najwyżej p^m liści. □

Twierdzenie 3.8 (Lemat o pompowaniu)

Dla dowolnego języka bezkontekstowego L istnieje stała $n \in \mathbb{N}$ o następującej własności:

Jeżeli $w \in L$ oraz $|w| \geq n$, to w można przedstawić w postaci $w = uvzxy$, gdzie

- $vx \neq \varepsilon$;
- $|vzx| \leq n$;
- $uv^i zx^i y \in L$ *dla dowolnego* $i \in \mathbb{N}$, *w szczególności* $uzy \in L$.

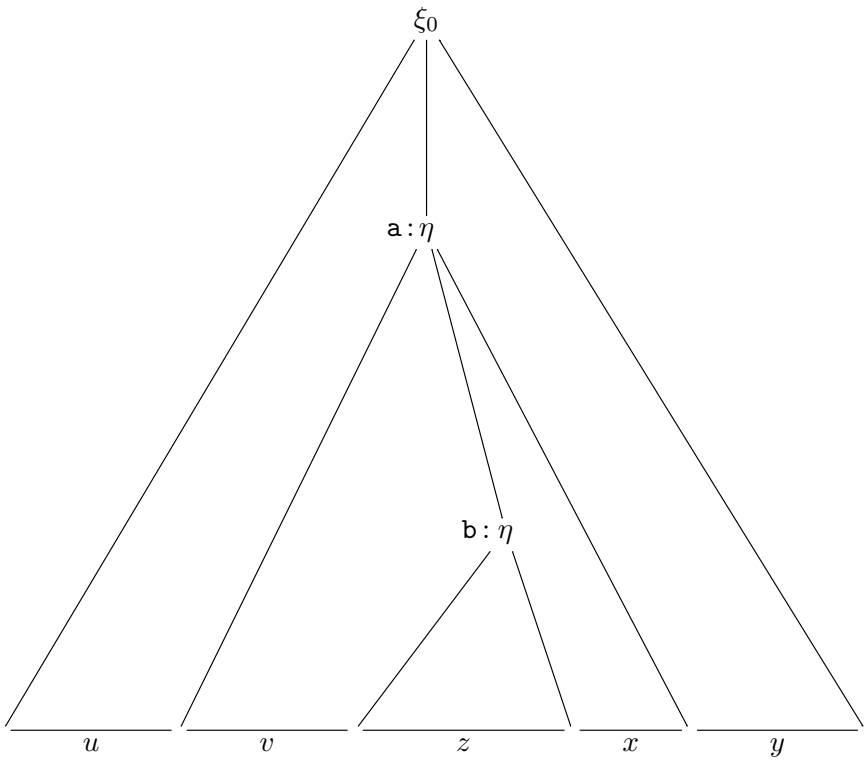
Stałą n można efektywnie wyznaczyć na podstawie gramatyki generującej L.

Dowód: Niech $L = L(G)$ i niech p będzie takie, że $|x| \leq p$ dla dowolnej produkcji $\xi \Rightarrow x$ gramatyki G. Na dodatek, niech $|\mathcal{N}| = m$. Wybieramy stałą $n = p^m + 1$. Przypuśćmy teraz, że $w \in L$ oraz $|w| \geq n$. Weźmy drzewo wywodu słowa w o najmniejszej możliwej liczbie wierzchołków. Wysokość tego drzewa (Lemat 3.7) musi być większa niż m, czyli niż liczba nieterminałów. Zatem na pewnej ścieżce w drzewie powtarza się nieterminalna etykieta, i to wśród ostatnich $m + 1$ wierzchołków tej ścieżki.

Niech η będzie takim powtarzającym się nieterminałem. Załóżmy, że η występuje jako etykieta pewnego wierzchołka `a` i jego potomka `b`. Załóżmy przy tym, że liczba wszystkich potomków wierzchołka `a` jest najmniejsza możliwa, tj. poniżej `a` nie ma już takich powtórzeń. Oznacza to, że wysokość poddrzewa zaczepionego w `a` jest nie większa od m.

Dla pewnych słów v, z, x mamy teraz $\eta \twoheadrightarrow v\eta x \twoheadrightarrow vzx$, bo również $\eta \twoheadrightarrow z$. Nasze drzewo odpowiada więc takiemu wyprowadzeniu:

$$\xi_0 \twoheadrightarrow u\eta y \twoheadrightarrow uv\eta xy \to uvzxy,$$



w którym dwa wyróżnione wystąpienia η dotyczą wierzchołków a i b. Można to wyprowadzenie modyfikować, usuwając lub powielając fragment drzewa odpowiadający redukcji $\eta \twoheadrightarrow v\eta x$. Otrzymujemy w ten sposób wyprowadzenia:

$$\xi_0 \twoheadrightarrow u\eta y \twoheadrightarrow uzy,$$

$$\xi_0 \twoheadrightarrow u\eta y \twoheadrightarrow uv\eta xy \twoheadrightarrow uv^2\eta x^2 y \twoheadrightarrow \cdots \twoheadrightarrow uv^i\eta x^i y \twoheadrightarrow uv^i z x^i y.$$

Nierówność $|vzx| \leq n$ zachodzi dlatego, że wyprowadzenie $\eta \twoheadrightarrow vzx$ odpowiada drzewu o wysokości co najwyżej m. Ponadto $vx \neq \varepsilon$, bo inaczej $uzy = w$ i drzewo wyprowadzenia $\xi_0 \twoheadrightarrow u\eta y \twoheadrightarrow uzy$ jest mniejsze. □

Przykład 3.9 W drzewie na rysunku 3 mamy dwa wystąpienia nieterminała ξ_a, jedno nad drugim. Słowo *ababbbba* można podzielić na pięć części $\varepsilon \cdot a \cdot bab \cdot b \cdot bba$ w ten sposób, że każde ze słów $\varepsilon \cdot a^i \cdot bab \cdot b^i \cdot bba$ należy do języka z przykładu 3.1.

Przykład 3.10 Język $\{a^k b^k c^k \mid k \in \mathbb{N}\}$ nie jest bezkontekstowy. Istotnie, przypuśćmy, że n jest stałą z lematu o pompowaniu, i że $uvzxy$ jest żądanym rozbiciem słowa $a^n b^n c^n$. Środkowa część vzx jest długości co najwyżej n i jest za krótka na to, aby występowały w niej zarówno litery a jak i c. Zatem w słowie uv^2zx^2y zabraknie albo liter a albo c.

Wniosek 3.11 *Klasa języków bezkontekstowych nie jest zamknięta ze względu na iloczyn.*

Dowód: Język z przykładu 3.10 jest iloczynem języków bezkontekstowych:

$$\{a^k b^k c^k \mid k \in \mathbb{N}\} = \{a^n b^n c^k \mid n, k \in \mathbb{N}\} \cap \{a^n b^k c^k \mid n, k \in \mathbb{N}\}.$$ □

4 Automaty ze stosem

Przez (niedeterministyczny) *automat ze stosem* rozumiemy krotkę:

$$\mathcal{M} = \langle Q, \mathcal{A}, \Sigma, \delta, q_0, \sigma_0, F \rangle,$$

w której:

- $\mathcal{A}$ jest skończonym alfabetem (wejściowym);
- Σ jest skończonym alfabetem stosu;
- Q jest skończonym zbiorem *stanów*;
- $q_0 \in Q$ jest stanem *początkowym*;
- $\sigma_0 \in \Sigma$ jest *początkowym symbolem stosu*;
- $F \subseteq Q$ jest zbiorem stanów *końcowych*;
- $\delta \subseteq Q \times \Sigma \times (\mathcal{A} \cup \{\varepsilon\}) \times Q \times \Sigma^*$ jest *relacją przejścia*.

Będziemy czasem używać skrótu A_ε na oznaczenie $\mathcal{A} \cup \{\varepsilon\}$ (podobnie Σ_ε). Parę postaci $\langle q, V \rangle$, gdzie q jest stanem i $V \in \Sigma^*$, nazywamy *konfiguracją* automatu. (Interpretacja: automat jest w stanie q, a zawartością stosu jest słowo V, przy czym wierzchołek stosu jest z lewej strony.)

Sens instrukcji $\langle q, \sigma, a, p, U \rangle \in \delta$ jest taki: ze stanu q można przejść do stanu p, czytając z wejścia literę a i zastępując literę σ na wierzchołku stosu słowem U.

Automat ze stosem można przedstawić w postaci grafu stanów. Krawędzie tego grafu muszą być oznaczone nie tylko literami alfabetu $\mathcal{A}$, ale także operacjami, które należy wykonać na stosie przy zmianie stanu. Jeśli więc $\langle q, \sigma, a, p, U \rangle \in \delta$, to stany p i q połączymy krawędzią oznaczoną tak: $q \xrightarrow[a]{\sigma \,:=\, U} p$

Działanie automatu opisujemy za pomocą relacji $\mathcal{M} \vdash \mathcal{C}_1 \overset{w}{\longrightarrow\!\!\!\!\!\rightarrow} \mathcal{C}_2$, pomiędzy konfiguracjami:

- Jeśli $\langle q, \sigma, a, p, U \rangle \in \delta$, to $\mathcal{M} \vdash \langle q, \sigma V \rangle \overset{a}{\longrightarrow\!\!\!\!\!\rightarrow} \langle p, UV \rangle$, dla dowolnego V. (Uwaga: a może być równe ε);
- Jeśli $\mathcal{M} \vdash \mathcal{C}_1 \overset{w_1}{\longrightarrow\!\!\!\!\!\rightarrow} \mathcal{C}_2$ i $\mathcal{M} \vdash \mathcal{C}_2 \overset{w_2}{\longrightarrow\!\!\!\!\!\rightarrow} \mathcal{C}_3$, to $\mathcal{M} \vdash \mathcal{C}_1 \overset{w_1 w_2}{\longrightarrow\!\!\!\!\!\rightarrow} \mathcal{C}_3$.

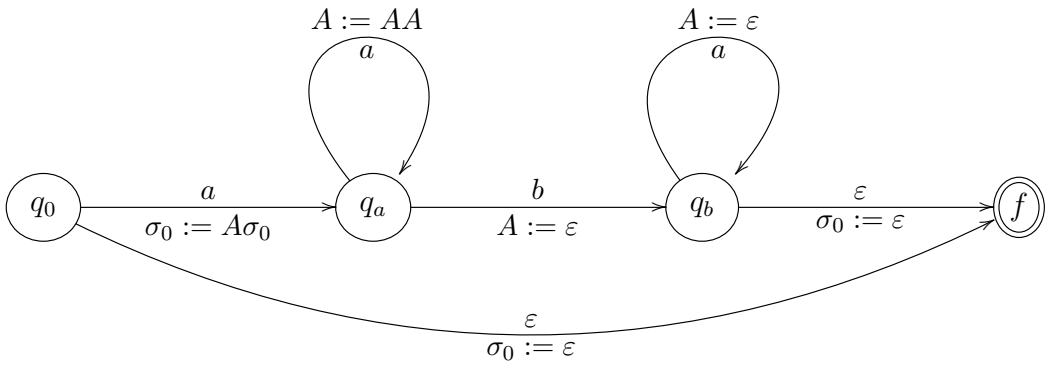
Pomijamy „$\mathcal{M} \vdash$” jeśli wiadomo o jaki automat chodzi. Język akceptowany przez automat ze stosem definiujemy tak:

$$L(\mathcal{M}) = \{w \in \mathcal{A} \mid \mathcal{M} \vdash \langle q_0, \sigma_0 \rangle \overset{w}{\longrightarrow\!\!\!\!\!\rightarrow} \langle q, U \rangle, \text{ dla pewnego } q \in F \text{ i pewnego } U\}.$$

Przez *obliczenie* rozumiemy oczywiście odpowiedni ciąg konfiguracji.

Przykład 4.1 Dla zaakceptowania języka $\{a^n b^n \mid n \in \mathbb{N}\}$ potrzeba automatu o czterech stanach q_0, q_a, q_b, f i alfabecie stosowym $\{\sigma_0, A\}$. Relacja przejścia składa się z piątek:

- $\langle q_0, \sigma_0, \varepsilon, f, \varepsilon\rangle$
- $\langle q_a, A, a, q_a, AA\rangle$;
- $\langle q_b, A, b, q_b, \varepsilon\rangle$;
- $\langle q_0, \sigma_0, a, q_a, A\sigma_0\rangle$;
- $\langle q_a, A, b, q_b, \varepsilon\rangle$;
- $\langle q_b, \sigma_0, \varepsilon, f, \varepsilon\rangle$.



W tym przykładzie każde akceptujące obliczenie kończy się z pustym stosem, tj. w konfiguracji $\langle f, \varepsilon\rangle$, a symbol początkowy stosu jest z niego usuwany dopiero na samym końcu. Zmiany stosu polegają zawsze albo na dołożeniu jednej litery (operacja *push*) albo na usunięciu jednej litery (operacja *pop*). Można zakładać, że zawsze tak jest (patrz lemat 4.8).

Automaty deterministyczne

Automat ze stosem $\mathcal{M} = \langle Q, \mathcal{A}, \Sigma, \delta, q_0, \sigma_0, F\rangle$ jest *deterministyczny*, jeśli spełnia takie warunki:

- Relacja przejścia jest funkcją częściową, $\delta : (Q \times \Sigma \times \mathcal{A}_\varepsilon) \to (Q \times \Sigma^*)$;
- Dla dowolnej pary $q \in Q$, $\sigma \in \Sigma$, jeżeli wartość $\delta(q, \sigma, \varepsilon)$ jest określona, to nie jest określona żadna z wartości $\delta(q, \sigma, a)$, gdzie $a \in \mathcal{A}$.

A zatem automat deterministyczny może (przy ustalonym wejściu) w każdej konfiguracji wykonać co najwyżej jeden ruch. Dla danej konfiguracji $\mathcal{C}$ takiego automatu, i danego słowa wejściowego, obliczenie rozpoczynające się od $\mathcal{C}$ może więc być tworzone tylko na jeden sposób.

Języki akceptowane przez deterministyczne automaty ze stosem nazywamy *deterministycznymi językami bezkontekstowymi*. Okazuje się, że klasa deterministycznych języków bezkontekstowych jest zamknięta ze względu na dopełnienie. (Jeśli L jest deterministycznym językiem bezkontekstowym, to $-L$ też.)

To jest nieco mniej oczywiste niż dla automatów skończonych (wniosek 2.6), bo deterministyczny automat ze stosem może się „zapętlić” (wchodząc w nieskończony ciąg epsilon-kroków) lub „zablokować” w konfiguracji nie pozwalającej na żaden ruch. Ale łatwo można każdy deterministyczny automat przerobić na równoważny automat, któremu te zachowania nie grożą

(lemat 4.2). Wtedy znowu wystarczy zamienić stany akceptujące na nieakceptujące i odwrotnie. Więcej szczegółów drobnym drukiem:

▶

Lemat 4.2 *Jeśli L jest deterministycznym językiem bezkontekstowym, to $L = L(\mathcal{M}')$ dla pewnego deterministycznego automatu ze stosem $\mathcal{M} = \langle Q, \mathcal{A}, \Sigma, \delta, q_0, \sigma_0, F\rangle$, który ma takie własności:*

1) *Jeśli $\delta(q, \sigma_0, a) = (p, W)$, to $W = W'\sigma_0$, gdzie $W' \in (\Sigma - \{\sigma_0\})^*$, jeśli zaś $\delta(q, \sigma, a) = (p, W)$, gdzie $\sigma \neq \sigma_0$, to $W \in (\Sigma - \{\sigma_0\})^*$. (Symbol σ_0 nie jest nigdy usuwany ze stosu ani nie jest tam dokładany.)*
2) *Dla dowolnej pary $q \in Q$, $\sigma \in \Sigma$, albo określona jest wartość $\delta(q, \sigma, \varepsilon)$, albo wszystkie wartości $\delta(q, \sigma, a)$, dla $a \in \mathcal{A}$. (Automat $\mathcal{M}$ w każdej konfiguracji może wykonać dokładnie jeden ruch.)*
3) *Dla dowolnego słowa w istnieje obliczenie postaci $\langle q_0, \sigma_0\rangle \xrightarrow{w}\!\!\!\!\rightarrow \langle q, X\rangle$, tj. automat zawsze czyta całe słowo wejściowe.*

Dowód: Załóżmy, że $L = L(\mathcal{M}')$, gdzie $\mathcal{M}' = \langle Q', \mathcal{A}, \Sigma', \delta', q_0', \sigma_0', F'\rangle$. Przerabiamy automat $\mathcal{M}'$ na automat $\mathcal{M}$. Dodajemy nowy stan początkowy $q_0 \neq q_0'$ i wybieramy nowy początkowy symbol stosu $\sigma_0 \neq \sigma_0'$. Definiujemy $\delta(q_0, \sigma_0, \varepsilon) = (q_0', \sigma_0'\sigma_0)$. W ten sposób uzyskamy warunek (1).

Teraz dodajemy nowy stan r i we wszystkich przypadkach gdy zarówno $\delta'(q, \sigma, \varepsilon)$ jest nieokreślone jak też nieokreślone jest $\delta'(q, \sigma, a)$ dla pewnego $a \in \mathcal{A}$, definiujemy

$$\delta(q, \sigma, a) = (r, \sigma).$$

Dla dowolnych a i σ definiujemy też $\delta(r, \sigma, a) = (r, \sigma)$. W ten sposób uzyskamy (2). Stan r nie będzie stanem akceptującym, więc język akceptowany się nie zmieni.

Dodajemy teraz nowy stan f i przyjmujemy $F = F' \cup \{f\}$. Definiujemy $\delta(f, \sigma, a) = (r, \sigma)$, dla wszystkich $a \in \mathcal{A}$.

Powiemy, że para $\langle q, \sigma\rangle$ jest *martwą konfiguracją*, gdy istnieje nieskończone obliczenie automatu $\mathcal{M}'$:

$$\langle q, \sigma\rangle = \langle p_0, V_0\rangle \xrightarrow{\varepsilon}\!\!\!\!\rightarrow \langle p_1, V_1\rangle \xrightarrow{\varepsilon}\!\!\!\!\rightarrow \langle p_2, V_2\rangle \cdots$$

Jeśli teraz któryś ze stanów p_i jest końcowy, to poprawiamy funkcję δ' tak: $\delta(q, \sigma, \varepsilon) = (f, \sigma)$. Jeśli żaden nie jest końcowy, to definiujemy $\delta(q, \sigma, \varepsilon) = (r, \sigma)$. Robimy tak dla każdej martwej konfiguracji. Ich liczba jest oczywiście skończona. W pozostałych przypadkach definiujemy δ tak samo jak δ'.

Musimy pokazać, że nasz nowy automat spełnia warunek (3). Jeśli automat $\mathcal{M}'$ ma obliczenie, które czyta całe słowo w, to oczywiście $\mathcal{M}$ też ma takie obliczenie. W przeciwnym razie mamy pewne właściwe podsłowo w' słowa w i obliczenie automatu $\mathcal{M}'$ postaci

$$\langle q_0, \sigma_0\rangle \xrightarrow{w'}\!\!\!\!\rightarrow \langle p_0, W_0\rangle \xrightarrow{\varepsilon}\!\!\!\!\rightarrow \langle p_1, W_1\rangle \xrightarrow{\varepsilon}\!\!\!\!\rightarrow \langle p_2, W_2\rangle \xrightarrow{\varepsilon}\!\!\!\!\rightarrow \cdots$$

Z ciągu słów W_i wybieramy najkrótsze możliwe słowo, powiedzmy W_k. Przypuśćmy, że $W_k = \tau \cdot W$ (słowo W_k jest niepuste, bo zachodzi warunek (1)). Wtedy para $\langle p_k, \tau\rangle$ jest martwą konfiguracją. Rzeczywiście: dla $m \geq k$, słowa W_m są postaci $W_m = W_m'W$, istnieje więc nieskończone obliczenie

$$\langle p_k, \tau\rangle \xrightarrow{\varepsilon}\!\!\!\!\rightarrow \langle p_{k+1}, W_{k+1}'\rangle \xrightarrow{\varepsilon}\!\!\!\!\rightarrow \langle p_{k+2}, W_{k+2}'\rangle \xrightarrow{\varepsilon}\!\!\!\!\rightarrow \cdots$$

W automacie $\mathcal{M}$ mamy teraz $\delta(p_k, \tau, \varepsilon) = (r, \tau)$ lub $\delta(p_k, \tau, \varepsilon) = (f, \tau)$. A zatem

$$\mathcal{M} \vdash \langle q_0, \sigma_0\rangle \xrightarrow{w'}\!\!\!\!\rightarrow \langle p_k, \tau \cdot W\rangle \xrightarrow{\varepsilon}\!\!\!\!\rightarrow \langle p, \tau \cdot W\rangle \xrightarrow{w''}\!\!\!\!\rightarrow \langle r, \tau \cdot W\rangle,$$

gdzie p to albo r albo f. Tak czy owak, obliczenie kończy się w stanie r, bo słowo w'' jest niepuste. □

◀

Fakt 4.3 *Klasa deterministycznych języków bezkontekstowych jest zamknięta ze względu na dopełnienie.*

▶ **Dowód**: Niech $L = L(\mathcal{M})$, gdzie $\mathcal{M} = \langle Q, \mathcal{A}, \Sigma, \delta, q_0, \sigma_0, F\rangle$ spełnia warunki Lematu 4.2. Zdefiniujemy automat $\mathcal{M}' = \langle Q', \mathcal{A}, \Sigma, \delta', q_0', \sigma_0, F'\rangle$, akceptujący dopełnienie języka L. Jego zbiorem stanów jest $Q' = Q \times \{0, 1, 2\}$, a jako stan początkowy wybieramy: $q_0' = \langle q_0, 0\rangle$ jeśli $q_0 \notin F$, a w przeciwnym przypadku $q_0' = \langle q_0, 1\rangle$. Definiujemy jeszcze $F' = \{\langle q, 2\rangle \mid q \in Q\}$.

Automat $\mathcal{M}'$ ma za zadanie naśladować obliczenia automatu $\mathcal{M}$ i do tego służy pierwsza współrzędna każdego stanu. Druga współrzędna zawiera informację o tym, czy automat $\mathcal{M}$ osiągnął (1) czy nie (0) jakiś stan akceptujący przeczytane dotychczas słowo. Wartość 2 oznacza, że stanu akceptującego nie było i już nie będzie, bo nie da się wykonać kroku epsilonowego.

Automat $\mathcal{M}'$ działa więc następująco. Jeśli stan q nie jest stanem akceptującym w $\mathcal{M}$, to każde przejście $q \xrightarrow[a]{\sigma:=V} p$ automatu $\mathcal{M}$, gdzie $a \in \mathcal{A}$, jest w automacie $\mathcal{M}'$ rozbite na dwa kroki postaci $\langle q, 0\rangle \xrightarrow[\varepsilon]{\sigma:=\sigma} \langle q, 2\rangle \xrightarrow[a]{\sigma:=V} \langle p, 1\rangle$ lub postaci $\langle q, 0\rangle \xrightarrow[\varepsilon]{\sigma:=\sigma} \langle q, 2\rangle \xrightarrow[a]{\sigma:=V} \langle p, 0\rangle$, zależnie od tego, czy $p \in F$ czy nie. Przejścia postaci $q \xrightarrow[\varepsilon]{\sigma:=V} p$ są zaś zastąpione odpowiednio przez $\langle q, 0\rangle \xrightarrow[\varepsilon]{\sigma:=V} \langle p, 1\rangle$ lub $\langle q, 0\rangle \xrightarrow[\varepsilon]{\sigma:=V} \langle p, 0\rangle$. A jeśli $q \in F$, to krok $q \xrightarrow[a]{\sigma:=V} p$ zastępujemy po prostu przez $\langle q, 1\rangle \xrightarrow[a]{\sigma:=V} \langle p, 1\rangle$ lub $\langle q, 1\rangle \xrightarrow[a]{\sigma:=V} \langle p, 0\rangle$ (jeśli $q \in F$, to stany $\langle q, 0\rangle$ i $\langle q, 2\rangle$ nie są używane). Jeśli więc po przeczytaniu słowa w maszyna $\mathcal{M}'$ wchodzi w stan postaci $\langle q, 2\rangle$, to jest to ostatni stan osiągalny dla tego słowa. To znaczy, że maszyna $\mathcal{M}$ nie mogła zaakceptować w. Na odwrót, jeśli $w \notin L$, to maszyna $\mathcal{M}$ czytając w wchodzi ostatecznie w stan $q \notin F$, w którym możliwe są tylko kroki literowe (por. lemat 4.2). Wtedy maszyna $\mathcal{M}'$ osiąga kolejno stany $\langle q, 0\rangle$ i $\langle q, 2\rangle$ i akceptuje.

A zatem słowo w jest akceptowane przez $\mathcal{M}'$ wtedy i tylko wtedy, gdy *nie jest* akceptowane przez $\mathcal{M}$, bo obliczenie automatu $\mathcal{M}$ dla w nie osiąga stanu akceptującego. Ścisła definicja funkcji przejścia automatu $\mathcal{M}'$ jest taka:

- Jeśli $\delta(q, \varepsilon, \sigma) = (p, V)$, to:
  - $\delta'(\langle q, 1\rangle, \varepsilon, \sigma) = (\langle p, 1\rangle, V)$;
  - $\delta'(\langle q, 0\rangle, \varepsilon, \sigma) = (\langle p, 1\rangle, V)$, dla $p \in F$;
  - $\delta'(\langle q, 0\rangle, \varepsilon, \sigma) = (\langle p, 0\rangle, V)$, dla $p \notin F$.
- Jeśli $\delta(q, a, \sigma) = (p, V)$, gdzie $a \in \mathcal{A}$, to:
  - $\delta'(\langle q, 1\rangle, a, \sigma) = (\langle p, 1\rangle, V)$, dla $p \in F$;
  - $\delta'(\langle q, 1\rangle, a, \sigma) = (\langle p, 0\rangle, V)$, dla $p \notin F$;
  - $\delta'(\langle q, 2\rangle, a, \sigma) = (\langle p, 1\rangle, V)$, dla $p \in F$;
  - $\delta'(\langle q, 2\rangle, a, \sigma) = (\langle p, 0\rangle, V)$, dla $p \notin F$;
  - $\delta'(\langle q, 0\rangle, \varepsilon, \sigma) = (\langle q, 2\rangle, \sigma)$.

□

◀

Fakt 4.4 *Klasa języków bezkontekstowych nie jest zamknięta ze względu na dopełnienie.*

Dowód: Rozpatrzmy języki:

$$L_1 = \{a^n b^n c^k \mid n, k \in \mathbb{N}\}, \quad L_2 = \{a^n b^k c^k \mid n, k \in \mathbb{N}\}.$$

Gdyby klasa języków bezkontekstowych była zamknięta ze względu na dopełnienie, to język

$$L = \{a^n b^n c^n \mid n \in \mathbb{N}\} = L_1 \cap L_2 = -(-L_1 \cup -L_2)$$

byłby bezkontekstowy. A nie jest (przykład 3.10). □

Wniosek 4.5 *Nie każdy język bezkontekstowy jest deterministyczny.*

Przykład 4.6 Konkretnym przykładem języka bezkontekstowego, który nie jest deterministyczny jest język $L_3 = \{a^n b^m c^k \mid n \neq m \vee m \neq k\}$. Gdyby ten język był deterministyczny, to język $L = \{a^n b^n c^n \mid n \in \mathbb{N}\}$ byłby bezkontekstowy, bo $L = -L_3 \cap a^* b^* c^*$, a łatwo pokazać, że iloczyn języka bezkontekstowego i języka regularnego musi być bezkontekstowy.

Akceptowanie języków bezkontekstowych

Twierdzenie 4.7 *Każdy język bezkontekstowy jest akceptowany przez pewien niedeterministyczny automat ze stosem.*

Dowód: Niech $L = L(G)$ dla pewnej gramatyki $G = \langle \mathcal{A}, \mathcal{N}, P, \xi_0 \rangle$. Konstruujemy automat $\mathcal{M} = \langle Q, \mathcal{A}, \Sigma, \delta, q_0, \sigma_0, F \rangle$ akceptujący L.

Zbiorem stanów jest $Q = \{q_0, q_1, q_2\}$, przy czym q_2 jest końcowy, tj. $F = \{q_2\}$. Jako alfabet stosu przyjmujemy $\Sigma = \mathcal{A} \cup \mathcal{N} \cup \{\sigma_0\}$. Relacja składa się z takich piątek:

- $\langle q_0, \sigma_0, \varepsilon, q_1, \xi_0 \sigma_0 \rangle$;
- $\langle q_1, \xi, \varepsilon, q_1, w \rangle$, dla dowolnej reguły „$\xi \Rightarrow w$” $\in P$;
- $\langle q_1, a, a, q_1, \varepsilon \rangle$, dla $a \in \mathcal{A}$;
- $\langle q_1, \sigma_0, \varepsilon, q_2, \varepsilon \rangle$.

Sens tej konstrukcji jest taki: automat najpierw umieszcza na stosie symbol początkowy ξ_0 gramatyki G. Potem cały czas mamy na stosie pewne słowo $x \in (\mathcal{A} \cup \mathcal{N})^*$, takie że $G \vdash \xi_0 \twoheadrightarrow ux$, dla pewnego $u \in \mathcal{A}^*$. Automat może w każdym kroku zrobić jedną z dwóch rzeczy:

1) Jeśli pierwszy symbol słowa x jest w $\mathcal{A}$, to ten sam symbol powinien pojawiać się na wejściu. Czytamy symbol z wejścia i usuwamy go ze stosu.

2) Jeśli pierwszy symbol ξ słowa x jest nieterminalny, to zastępujemy go przez prawą stronę którejś z odpowiednich produkcji (tu działa niedeterminizm).

Maszyna akceptuje, gdy na stosie już nic nie zostało oprócz symbolu końca stosu. Działanie jej możemy więc opisać taką równoważnością, która zachodzi dla dowolnych $w \in \mathcal{A}^*$ oraz dowolnych $x, y \in (\mathcal{A} \cup \mathcal{N})^*$:

$$\mathcal{M} \vdash \langle q_1, y\sigma_0 \rangle \xrightarrow{w} \langle q_1, x\sigma_0 \rangle \qquad \text{wtedy i tylko wtedy, gdy} \qquad G \vdash y \twoheadrightarrow^L wx$$

Ścisły dowód implikacji ($\Rightarrow$) można przeprowadzić przez indukcję ze względu na liczbę wystąpień stanu q_1 w obliczeniu, a dowód implikacji ($\Leftarrow$) przez indukcję ze względu na liczbę kroków wyprowadzenia lewostronnego. Jeśli w powyższej równoważności przyjmiemy $y = \xi_0$ i $x = \varepsilon$, to łatwo otrzymamy równość $L(\mathcal{M}) = L(G)$. □

Lemat 4.8 *Jeśli $L = L(\mathcal{M}')$ dla pewnego automatu ze stosem $\mathcal{M}' = \langle Q', \mathcal{A}, \Sigma', \delta', q_0', \sigma_0', F' \rangle$, to także $L = L(\mathcal{M})$, gdzie $\mathcal{M} = \langle Q, \mathcal{A}, \Sigma, \delta, q_0, \sigma_0, F \rangle$ ma takie własności:*

1) *Automat $\mathcal{M}$ ma tylko jeden stan końcowy: $F = \{f\}$, przyjmowany tylko na koniec obliczenia (nie ma w δ piątki postaci $\langle f, \ldots \rangle$).*

2) *Automat $\mathcal{M}$ akceptuje tylko z pustym stosem:*
$$L = L(\mathcal{M}) = \{w \in \mathcal{A}^* \mid \mathcal{M} \vdash \langle q_0, \sigma_0 \rangle \stackrel{w}{\longrightarrow\!\!\!\!\!\rightarrow} \langle f, \varepsilon \rangle\}.$$

3) *Relacja δ składa się wyłącznie z piątek postaci $\langle q, \sigma, a, p, \tau\sigma \rangle$ i $\langle q, \sigma, a, p, \varepsilon \rangle$, gdzie $\tau, \sigma \in \Sigma$ oraz $a \in \mathcal{A} \cup \{\varepsilon\}$. (Operacje wykonywane na stosie są tylko typu pop i push.)*

4) *Nie ma w δ piątki postaci $\langle q, \sigma, a, p, \sigma_0\sigma \rangle$, a jeśli $\langle q, \sigma_0, a, p, \varepsilon \rangle \in \delta$, to $p = f$. (Symbol σ_0 jest usuwany ze stosu tylko w ostatnim kroku przed zaakceptowaniem i nigdy nie jest tam dokładany.)*

► **Dowód**: Przerabiamy automat $\mathcal{M}'$ na automat $\mathcal{M}$. Najpierw dodajemy nowy stan początkowy q_0 i nowy początkowy symbol stosu σ_0. Dodajemy jedno nowe przejście $\langle q_0, \varepsilon, \sigma_0, q_0', \sigma_0'\sigma_0 \rangle$. Po wykonaniu tego przejścia, automat $\mathcal{M}$ robi to samo co automat $\mathcal{M}'$, z tą różnicą, że na dnie stosu leży dodatkowo symbol σ_0.

Dodajemy teraz nowy stan f'' i takie przejścia:

$$\langle f'', \varepsilon, \sigma_0, f, \varepsilon \rangle;$$
$$\langle f', \varepsilon, \sigma', f'', \varepsilon \rangle;$$
$$\langle f'', \varepsilon, \sigma', f'', \varepsilon \rangle,$$

dla dowolnego stanu końcowego f' automatu $\mathcal{M}'$ i dowolnego $\sigma' \neq \sigma_0$.

Przejścia te pozwalają na opróżnienie stosu po osiągnięciu stanu f' i przejście do stanu końcowego maszyny $\mathcal{M}$. Jest to jedyny sposób, w jaki można usunąć ze stosu symbol σ_0.

Ponieważ nie dołożymy już żadnego przejścia wstawiającego σ_0 na stos, więc warunki (1), (2) i (4) będą spełnione. Dla spełnienia warunku (3) rozbijemy każdy krok maszyny $\mathcal{M}'$ na kilka kroków maszyny $\mathcal{M}$, typu *pop* lub *push*. Każdą piątkę $\langle q, a, \sigma, p, \tau_1 \ldots \tau_k \rangle$, która nie jest ani postaci *push* ani *pop*, zastąpimy mianowicie przez piątki:

- $\langle q, \sigma, a, r^{k-1}, \tau_{k-1}\sigma \rangle$, $\langle r^{k-1}, \tau_{k-1}, \varepsilon, r^{k-2}, \tau_{k-2}\tau_{k-1} \rangle, \ldots \langle r^2, \tau_2, \varepsilon, p, \tau_1\tau_2 \rangle$, gdy $k \geq 2$ oraz $\tau_k = \sigma$;
- $\langle q, \sigma, a, r, \sigma'\sigma \rangle$ i $\langle r, \sigma', \varepsilon, p, \varepsilon \rangle$, gdy $k = 1$ i $\tau_k = \sigma$ (symbol σ' jest dowolny, byle tylko $\sigma' \neq \sigma_0$);
- $\langle q, \sigma, a, r^k, \varepsilon \rangle$, $\langle r^k, \tau, \varepsilon, r^{k-1}, \tau_k\tau \rangle$ (dla dowolnego τ) oraz $\langle r^{k-1}, \tau_k, \varepsilon, r^{k-2}, \tau_{k-1}\tau_k \rangle, \ldots$ $\ldots \langle r^1, \tau_2, \varepsilon, p, \tau_1\tau_2 \rangle$, gdy $k > 0$ i $\tau_k \neq \sigma$.

Trzeba tylko pamiętać aby dla każdej eliminowanej piątki używać innych stanów r^i. □ ◄

Twierdzenie 4.9 *Dla dowolnego automatu ze stosem $\mathcal{M}$, język $L(\mathcal{M})$ jest bezkontekstowy.*

Dowód: Niech $\mathcal{M} = \langle Q, \mathcal{A}, \Sigma, \delta, q_0, \sigma_0, F \rangle$ i niech $Q = \{q_0, \ldots, q_n\}$. Zakładamy, że nasz automat spełnia warunki Lematu 4.8, w szczególności niech $F = \{q_n\}$. Definiujemy gramatykę $G = \langle \mathcal{A}, \mathcal{N}, P, \xi_0 \rangle$, gdzie $\mathcal{N} = \{\xi_{ij}^{\sigma} \mid i, j = 0, \ldots, n \wedge \sigma \in \Sigma\}$, oraz $\xi_0 = \xi_{0n}^{\sigma_0}$. Chodzi o to, aby z symbolu ξ_{ij}^{σ} wygenerować taki język:

$$L_{ij}^{\sigma} = \{w \in \mathcal{A}^* \mid \mathcal{M} \vdash \langle q_i, \sigma\rangle \xrightarrow{w}\!\!\!\!\rightarrow \langle q_j, \varepsilon\rangle\}.$$

Dokładniej, chcemy, żeby $L_{ij}^{\sigma} = L(G_{ij}^{\sigma})$, gdzie $G_{ij}^{\sigma} = \langle \mathcal{A}, \mathcal{N}, P, \xi_{ij}^{\sigma}\rangle$.

Jasne, że wtedy będziemy mieli $L(G) = L_{0n}^{\sigma_0} = L(\mathcal{M})$.

Produkcje dla nieterminału ξ_{ij}^{σ} zależą od możliwego przebiegu obliczenia $\langle q_i, \sigma\rangle \xrightarrow{w}\!\!\!\!\rightarrow \langle q_j, \varepsilon\rangle$. W zależności od pierwszego kroku, takie obliczenie może być trojakiej postaci (tutaj $a \in \mathcal{A}_{\varepsilon}$):

- $\langle q_i, \sigma\rangle \xrightarrow{a} \langle q_j, \varepsilon\rangle$ (pojedyncza operacja typu „pop");
- $\langle q_i, \sigma\rangle \xrightarrow{a} \langle q_k, \varepsilon\rangle \xrightarrow{b} \langle q_\ell, \tau\rangle \xrightarrow{w'}\!\!\!\!\rightarrow \langle q_j, \varepsilon\rangle$ (najpierw „pop", potem „push");
- $\langle q_i, \sigma\rangle \xrightarrow{a} \langle q_k, \tau\sigma\rangle \xrightarrow{w''}\!\!\!\!\rightarrow \langle q_\ell, \sigma\rangle \xrightarrow{w'''}\!\!\!\!\rightarrow \langle q_j, \varepsilon\rangle$ (najpierw „push"),

przy czym obliczenie $\langle q_k, \tau\sigma\rangle \xrightarrow{w''}\!\!\!\!\rightarrow \langle q_\ell, \sigma\rangle$ odbywa się bez usuwania σ ze stosu.

Dla $w \in L_{ij}^{\sigma}$, w drugim przypadku mamy $w' \in L_{\ell j}^{\tau}$, a w trzecim $w'' \in L_{k\ell}^{\tau}$ oraz $w''' \in L_{\ell j}^{\sigma}$. Dlatego gramatyka G ma takie produkcje:

- $\xi_{ij}^{\sigma} \Rightarrow a$, dla dowolnego przejścia $\langle q_i, \sigma, a, q_k, \varepsilon\rangle \in \delta$;
- $\xi_{ij}^{\sigma} \Rightarrow ab\xi_{\ell j}^{\tau}$, dla dowolnej pary przejść postaci $\langle q_i, \sigma, a, q_k, \varepsilon\rangle$ i $\langle q_k, \varepsilon, b, q_\ell, \tau\rangle$;
- $\xi_{ij}^{\sigma} \Rightarrow a\xi_{k\ell}^{\tau}\xi_{\ell j}^{\sigma}$, dla dowolnego przejścia $\langle q_i, \sigma, a, q_k, \tau\sigma\rangle \in \delta$.

Aby wykazać, że $L(G_{ij}^{\sigma}) = L_{ij}^{\sigma}$ postępujemy tak: inkluzji ($\subseteq$) dowodzimy przez indukcję ze względu na długość wyprowadzenia, a inkluzji ($\supseteq$) przez indukcję ze względu na długość obliczenia. □

Z powyższego wynika natychmiast nowy dowód faktu 3.3:

Wniosek 4.10 *Każdy język regularny jest bezkontekstowy.*

Dowód: Każdy automat skończony można uważać za automat ze stosem. □

Twierdzenie 4.9 uzasadnia też naszą definicję deterministycznego języka bezkontekstowego:

Wniosek 4.11 *Deterministyczne języki bezkontekstowe są bezkontekstowe.*

5 Hierarchia Chomsky'ego

Przez *gramatykę typu zero* rozumiemy krotkę $G = \langle \mathcal{A}, \mathcal{N}, P, \xi_0 \rangle$, w której znaczenie symboli $\mathcal{A}$, $\mathcal{N}$ i ξ_0 jest takie jak poprzednio, ale produkcje w zbiorze P są postaci $u \Rightarrow v$, gdzie słowa $u, v \in (\mathcal{A} \cup \mathcal{N})^*$ są zupełnie dowolne, byle tylko $u \neq \varepsilon$. Relacja redukcji $\to_G$ jest zdefiniowana podobnie jak dla gramatyk bezkontekstowych, mianowicie $x \to_G y$ (zapisywane też tak: $G \vdash x \to y$) zachodzi wtedy i tylko wtedy, gdy $x = x_1 u x_2$, $y = x_1 v x_2$, oraz $u \Rightarrow v$ jest pewną produkcją. Oczywiście notacja $\twoheadrightarrow_G$ oznacza istnienie (być może pustego) ciągu redukcji, a język generowany przez taką gramatykę definiujemy tak:

$$L(G) = \{w \in \mathcal{A}^* \mid \xi_0 \twoheadrightarrow_G w\}$$

Przykład 5.12 *Język* $\{a^n b^n c^n \mid n \in \mathbb{N}\}$ *jest generowany przez gramatykę, która ma produkcje:*

$$\xi_0 \Rightarrow \varepsilon, \quad \xi_0 \Rightarrow \eta;$$
$$\eta \Rightarrow a\beta\eta c, \quad \eta \Rightarrow abc;$$
$$\beta a \Rightarrow a\beta, \quad \beta b \Rightarrow bb.$$

Mówimy, że gramatyka jest *monotoniczna*, gdy jej produkcje są tylko postaci

- $\xi_0 \Rightarrow \varepsilon$;
- $u \Rightarrow v$, gdzie $|u| \leq |v|$ oraz ξ_0 nie występuje w słowie v.

Gramatyka z przykładu 5.12 jest monotoniczna. Jeśli język jest generowany przez gramatykę typu zero (odp. monotoniczną), to mówimy, że jest to język *typu zero* (odp. *monotoniczny*). Języki typu zero nazywamy też językami *rekurencyjnie przeliczalnymi*. Tradycyjna nazwa „zbiór rekurencyjnie przeliczalny" bierze się stąd, że elementy takiego zbioru można po kolei efektywnie generować, czyli „rekurencyjnie przeliczać". W tym celu należy systematycznie konstruować wszystkie możliwe wyprowadzenia gramatyki.

Fakt 5.13 *Każdy język bezkontekstowy jest monotoniczny.*

Dowód: Gramatyki bezkontekstowe nie muszą być monotoniczne, bo po prawej stronie produkcji może być słowo puste. Ale gramatykę bezkontekstową $G = \langle \mathcal{A}, \mathcal{N}, P, \xi_0 \rangle$, generującą język L można przerobić na gramatykę monotoniczną generującą ten sam język. W tym celu najpierw dodajemy nowy nieterminał ξ_0' i każdą regułę $\xi \Rightarrow v$, dla $\xi \neq \xi_0$, przerabiamy na $\xi \Rightarrow v'$, gdzie v' powstaje z v przez zamianę wszystkich ξ_0 na ξ_0'. Natomiast produkcje $\xi_0 \Rightarrow v$ zmieniamy na $\xi_0' \Rightarrow v$. Na koniec dodajemy produkcję $\xi_0 \Rightarrow \xi_0'$ i w ten sposób usunęliśmy symbol ξ_0 z prawych stron wszystkich produkcji.

Teraz niech $\mathcal{N}^\varepsilon = \{\xi \in \mathcal{N} \mid \xi \twoheadrightarrow_G \varepsilon\}$. Dla dowolnej produkcji $\eta \Rightarrow u$, jeśli w u występują symbole z $\mathcal{N}^\varepsilon$, to dodajemy do gramatyki wszystkie produkcje postaci $\eta \Rightarrow u'$, gdzie u' jest dowolnym niepustym słowem, które można otrzymać ze słowa u przez usunięcie jednego lub więcej wystąpień pewnych symboli ze zbioru $\mathcal{N}^\varepsilon$. (Na przyklad w przypadku produkcji $\eta \to a\eta' b\eta'' c$, gdzie $\eta', \eta'' \in \mathcal{N}^\varepsilon$ dodajemy produkcje $\eta \to ab\eta'' c$, $\eta \to a\eta' bc$, $\eta \to abc$.)

Po tej operacji usuwamy z gramatyki wszystkie produkcje postaci $\xi \Rightarrow \varepsilon$. Jeśli $\varepsilon \in L$ to dodajemy jeszcze produkcję $\xi_0 \Rightarrow \varepsilon$. Otrzymaliśmy bezkontekstową gramatykę monotoniczną.

Jeśli $w \in L$ i $w \neq \varepsilon$, to wyprowadzenie słowa w w gramatyce G można naśladować w gramatyce przerobionej, przewidując, z których wystąpień nieterminałów należących do $\mathcal{N}^\varepsilon$ ma zostać wyprowadzone słowo puste, i stosując zawczasu odpowiednie produkcje pomijające te wystąpienia. Na odwrót, wyprowadzenie terminalnego słowa w nowej gramatyce zawsze odpowiada pewnemu wyprowadzeniu w G, gdzie pewne nieterminały zostały zredukowane do słowa pustego. □

Języki kontekstowe: Gramatykę nazywamy *kontekstową* jeśli ma tylko produkcje postaci:

- $\xi_0 \Rightarrow \varepsilon$;
- $x\xi y \Rightarrow xvy$, gdzie $x, y, v \in (\mathcal{A} \cup \mathcal{N})^*$, $\xi \in \mathcal{N}$, $v \neq \varepsilon$ a symbol ξ_0 nie występuje w słowie v.

Języki generowane przez gramatyki kontekstowe nazywamy oczywiście językami *kontekstowymi*. Łatwo widzieć, że każda gramatyka kontekstowa jest monotoniczna. Na odwrót też:

Twierdzenie 5.14 *Język jest kontekstowy wtedy i tylko wtedy gdy jest monotoniczny.*

Dowód: Przypuśćmy, że G jest gramatyką monotoniczną. Przerabiamy ją na gramatykę kontekstową, akceptującą ten sam język.

Najpierw dodajemy nowe nieterminały ξ_a, dla wszystkich $a \in \mathcal{A}$. Dla dowolnego słowa x, przez x' oznaczmy słowo powstałe z x przez zamianę każdego $a \in \mathcal{A}$ na nieterminał ξ_a. Każdą produkcję $x \Rightarrow y$ zastępujemy przez produkcję $x' \Rightarrow y'$ i dodajemy wszystkie produkcje postaci $\xi_a \Rightarrow a$.

Po tym zabiegu mamy w gramatyce produkcje postaci $\xi_0 \Rightarrow \varepsilon$, $\xi \Rightarrow a$, oraz $x \Rightarrow y$, gdzie słowa x i y składają się z samych nieterminałów, a przy tym $|x| \leq |y|$. Pierwsze dwa rodzaje produkcji są dobre, a produkcje trzeciego rodzaju trzeba przerobić na kontekstowe. Weźmy więc taką produkcję, np.

$$\xi_1\xi_2 \ldots \xi_k \Rightarrow \eta_1\eta_2 \ldots \eta_n,$$

i przerabiajmy (pamiętając, że $k \leq n$). W tym celu dodamy do $\mathcal{N}$ nowe nieterminalne symbole $\tau_1, \ldots, \tau_n$, i zastąpimy naszą produkcję przez $k+n$ produkcji:

$$\begin{aligned}
\xi_1\xi_2 \ldots \xi_k &\Rightarrow \tau_1\xi_2 \ldots \xi_k \\
\tau_1\xi_2 \ldots \xi_k &\Rightarrow \tau_1\tau_2\xi_3 \ldots \xi_k \\
&\ldots \\
\tau_1 \ldots \tau_{k-2}\xi_{k-1}\xi_k &\Rightarrow \tau_1 \ldots \tau_{k-2}\tau_{k-1}\xi_k \\
\tau_1 \ldots \tau_{k-1}\xi_k &\Rightarrow \tau_1 \ldots \tau_n \\
\tau_1 \ldots \tau_n &\Rightarrow \eta_1\tau_2 \ldots \tau_n \\
\eta_1\tau_2 \ldots \tau_n &\Rightarrow \eta_1\eta_2\tau_3 \ldots \tau_n \\
&\ldots \\
\eta_1 \ldots \eta_{n-1}\tau_n &\Rightarrow \eta_1 \ldots \eta_n
\end{aligned}$$

Teraz już wszystkie produkcje są kontekstowe. Oczywiście nowa gramatyka generuje wszystkie słowa z języka $L(G)$. Ponieważ dla każdej z reguł eliminowanych w sposób opisany powyżej dobieramy nowe nieterminały τ_i, więc żadne dodatkowe słowo też nie może zostać wygenerowane. Zauważmy bowiem, że pozbycie się τ_i z generowanego słowa jest możliwe tylko wtedy gdy odpowiednia sekwencja dodanych reguł została wykonana w całości. □

Przykład 5.15 Pokażemy jak przerobić monotoniczną gramatykę z przykładu 5.12 na kontekstową. Najpierw we wszystkich produkcjach zamienimy litery terminalne na nieterminalne i dodamy reguły pozwalające na odtworzenie właściwych terminałów:

$$\xi_0 \Rightarrow \varepsilon, \quad \xi_0 \Rightarrow \eta;$$
$$\eta \Rightarrow \xi_a\beta\eta\xi_c, \quad \eta \Rightarrow \xi_a\xi_b\xi_c;$$
$$\beta\xi_a \Rightarrow \xi_a\beta, \quad \beta\xi_b \Rightarrow \xi_b\xi_b,$$
$$\xi_a \Rightarrow a, \quad \xi_b \Rightarrow b, \quad \xi_c \Rightarrow c.$$

W otrzymanej gramatyce jedna produkcja nie jest kontekstowa. Zastępujemy więc $\beta\xi_a \Rightarrow \xi_a\beta$ czterema produkcjami kontekstowymi, używającymi nowych nieterminałów τ i σ:

$$\beta\xi_a \Rightarrow \tau\xi_a, \quad \tau\xi_a \Rightarrow \tau\sigma, \quad \tau\sigma \Rightarrow \xi_a\sigma, \quad \xi_a\sigma \Rightarrow \xi_a\beta.$$

Teraz już wszystkie produkcje są kontekstowe. Otrzymana gramatyka generuje wszystkie słowa postaci $a^nb^nc^n$ i żadnych innych.

Hierarchia Chomsky'ego

Tak zwana *hierarchia Chomsky'ego* składa się z czterech poziomów:

- Języki typu zero, czyli *rekurencyjnie przeliczalne*.
- Języki typu jeden, czyli kontekstowe (monotoniczne).
- Języki typu dwa, czyli bezkontekstowe.
- Języki typu trzy, czyli regularne.

Inkluzje pomiędzy klasami języków tworzącymi kolejne szczeble hierarchii są właściwe, tj. dla każdego $n = 1, 2, 3$ istnieją języki typu $n-1$, które nie są typu n. Dla $n = 2, 3$ takie przykłady już znamy, przypadek $n = 1$ wyniknie z pewnych ogólniejszych faktów.

6 Maszyny Turinga

Niedeterministyczna, jednotaśmowa *maszyna Turinga* nad alfabetem $\mathcal{A}$ to krotka

$$\mathcal{M} = \langle \mathcal{A}, \Sigma, Q, \mathsf{B}, \delta, q_0, A, R \rangle$$

gdzie:

- Σ jest skończonym alfabetem, zawierającym $\mathcal{A}$ oraz symbol $\mathsf{B} \notin \mathcal{A}$ (blank);
- Q jest skończonym zbiorem *stanów*;
- $q_0 \in Q$ jest stanem *początkowym*;
- $A, R \subseteq Q$ są odpowiednio zbiorami stanów *akceptujących* i *odrzucających*;
- zbiory Σ i Q, oraz A i R są rozłączne, a suma $F = A \cup R$ to zbiór stanów *końcowych*;
- $\delta \subseteq (Q - F) \times \Sigma \times \Sigma \times Q \times \{-1, 0, +1\}$ jest *relacją przejścia.*

Zakładamy dla uproszczenia, że dla dowolnej pary (q, a), gdzie $q \notin F$ istnieje zawsze co najmniej jedna piątka $(q, a, b, p, i) \in \delta$. Maszyna jest *deterministyczna*, gdy δ jest funkcją:

$$\delta : (Q - F) \times \Sigma \to \Sigma \times Q \times \{-1, 0, +1\}.$$

Interpretacja tej definicji jest taka: maszyna zawsze znajduje się w dokładnie jednym ze swoich stanów i widzi dokładnie jedną klatkę taśmy (nieskończonej w obie strony). Na taśmie zapisane są znaki z alfabetu Σ. Relacja δ określa możliwe zachowanie maszyny: jeśli $(q, a, b, p, i) \in \delta$, to maszyna widząc a w stanie q może napisać b, przejść do stanu p i przesunąć głowicę o i klatek w prawo.

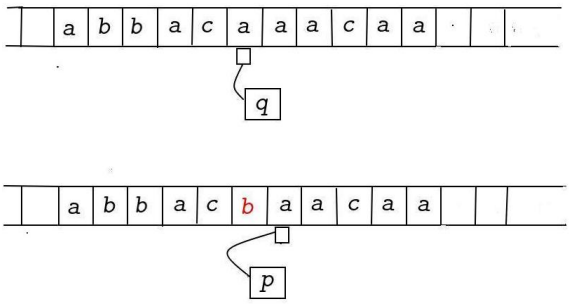
Przez *konfigurację* maszyny rozumie się zwykle słowo postaci wqv, gdzie $q \in Q$ oraz $w, v \in \Sigma^*$. Utożsamiamy konfiguracje wqv, $\mathsf{B}wqv$ i $wqv\mathsf{B}$. (Zawsze można więc zakładać, że słowo v nie kończy się blankiem, a słowo w nie zaczyna się od blanku.) Sens: na taśmie mamy słowo wv, z lewej i z prawej same blanki, a głowica maszyny patrzy na pierwszy znak na prawo od w. Konfigurację postaci $\mathcal{C}_w = q_0 w$, gdzie $w \in \mathcal{A}$, nazywamy *początkową*, a konfigurację postaci wqv, gdzie $q \in F$ nazywamy *końcową* (akceptującą lub odrzucającą, zależnie od stanu q).

► **Uwaga:** Ta definicja oznacza przyjęcie dwóch ważnych założeń interpretacyjnych. Po pierwsze, że dozwolone konfiguracje składają się z prawie samych blanków: na taśmie może być tylko skończenie wiele innych znaków. Po drugie, że nie odróżniamy od siebie sytuacji różniących się tylko przesunięciem na taśmie. Gdyby te dwa założenia odrzucić, to zawartość taśmy powinna być definiowana jako dowolna funkcja ze zbioru liczb całkowitych (numerów klatek na taśmie) w alfabet Σ, a konfiguracja maszyny jako para złożona z zawartości taśmy i stanu. W większości przypadków te dwa uproszczenia nie wpływają na treść uzyskanych wyników, ale nie zawsze tak jest. ◄

Relację $\to_{\mathcal{M}}$ na konfiguracjach definiuje się tak:

- Jeśli $(q, a, b, p, +1) \in \delta$, to $wqav \to_{\mathcal{M}} wbpv$;
- Jeśli $(q, a, b, p, 0) \in \delta$, to $wqav \to_{\mathcal{M}} wpbv$;
- Jeśli $(q, a, b, p, -1) \in \delta$, to $wcqav \to_{\mathcal{M}} wpcbv$;

Rysunek przedstawia konfigurację $\mathcal{C} = abbac\,q\,aaacaa$ i konfigurację $abbacb\,p\,aacaa$, którą można otrzymać z $\mathcal{C}$ jeśli do relacji przejścia należy piątka $\langle q, a, b, p, +1\rangle$.



Symbolem $\twoheadrightarrow_{\mathcal{M}}$ oznaczamy przechodnio-zwrotne domknięcie relacji $\rightarrow_{\mathcal{M}}$. Mówimy, że $\mathcal{M}$ *zatrzymuje się* dla konfiguracji $\mathcal{C}$ (dla słowa $w \in \mathcal{A}$), gdy $\mathcal{C} \twoheadrightarrow_{\mathcal{M}} \mathcal{C}'$ (odpowiednio, gdy $\mathcal{C}_w \twoheadrightarrow_{\mathcal{M}} \mathcal{C}'$), gdzie $\mathcal{C}'$ jest konfiguracją końcową. Jeżeli $\mathcal{C}_w \twoheadrightarrow_{\mathcal{M}} \mathcal{C}'$, gdzie $\mathcal{C}'$ jest konfiguracją akceptującą, to mówimy, że maszyna akceptuje słowo w.

Język *akceptowany* przez maszynę $\mathcal{M}$ definiujemy tak:

$$L(\mathcal{M}) = \{w \mid \mathcal{M} \text{ akceptuje } w\}.$$

W przypadku maszyny deterministycznej, mówimy, że maszyna *odrzuca* w, jeżeli $\mathcal{C}_w \twoheadrightarrow_{\mathcal{M}} \mathcal{C}'$, dla pewnej konfiguracji odrzucającej $\mathcal{C}'$. Maszyna deterministyczna jest *totalna*, jeżeli zatrzymuje się dla każdej konfiguracji początkowej.

Przez *obliczenie* maszyny, rozpoczynające się od konfiguracji $\mathcal{C}$, rozumiemy maksymalny ciąg konfiguracji $\mathcal{C} = \mathcal{C}_0 \rightarrow_{\mathcal{M}} \mathcal{C}_1 \rightarrow_{\mathcal{M}} \mathcal{C}_2 \rightarrow_M \cdots$. Obliczenie może być skończone (akceptujące lub odrzucające[6]) lub nieskończone. Dla danej konfiguracji $\mathcal{C}$, maszyna deterministyczna ma zawsze tylko jedno obliczenie rozpoczynające się od $\mathcal{C}$. Maszyna niedeterministyczna może mieć ich wiele.

Przykład 6.1 Relację przejścia maszyny Turinga można czasem przedstawić za pomocą tabelki.W wierszu q i kolumnie a znajdują się takie trójki (b, p, i), że $(q, a, b, p, i) \in \delta$. Nasz przykład to maszyna akceptująca język $\{ww \mid w \in \{a, b\}^*\}$. Stanem akceptującym jest OK. Dla czytelności tabeli nieistotne pola pozostały puste. Można tam wpisać cokolwiek.

W stanie q_0 maszyna zamazuje krzyżykiem obecnie oglądany symbol, i przechodzi do stanu q_a lub q_b, zależnie od tego, jaki to był symbol. Potem przesuwa głowicę w prawo, aż w pewnym momencie, w konfiguracji postaci $\#wq_aav$ lub $\#wq_bbv$ „postanowi" wracać. Wtedy przechodzi do konfiguracji postaci $\#wp\#v$ i przesuwa głowicę w lewo, aż nie natrafi na krzyżyk. Następnie cofa się o krok i przechodzi do stanu q_1. W tym stanie maszyna zachowuje się podobnie jak w stanie q_0, ale tym razem deterministycznie poszukujemy pierwszej litery na prawo od wcześniej postawionych krzyżyków. Jeśli ta litera jest taka jak trzeba, to zamazujemy ją i wracamy znowu na początek. Powtarzamy to tak długo, aż się nie okaże, że zamazaliśmy już wszystkie litery. Wtedy maszyna powracająca w lewo w stanie r natrafia na blank, przechodzi do stanu s i przesuwa głowicę w prawo. Jeśli natrafi znowu na blank, to akceptuje.

[6]Przy naszej definicji nie ma innej możliwości obliczenia skończonego.

	a	b	B	$\#$
q_0	$\#, q_a, +1$	$\#, q_b, +1$	$\mathsf{B}, \mathrm{OK}, 0$	
q_a	$a, q_a, +1$ $\#, p, -1$	$b, q_a, +1$	$\mathsf{B}, q_a, 0$	
q_b	$a, q_b, +1$	$b, q_b, +1$ $\#, p, -1$	$\mathsf{B}, q_a, 0$	
p	$a, p, -1$	$b, p, -1$		$\#, q_1, +1$
q_1	$\#, q^a, +1$	$\#, q^b, +1$		
q^a	$a, q^a, +1$	$b, q^a, +1$		$\#, r^a, +1$
q^b	$a, q^b, +1$	$b, q^b, +1$		$\#, r^b, +1$
r^a	$\#, r, -1$	$b, r^a, 0$	$\mathsf{B}, r^a, 0$	$\#, r^a, +1$
r^b	$a, r^b, 0$	$\#, r, -1$	$\mathsf{B}, r^b, 0$	$\#, r^b, +1$
r	$a, p, -1$	$b, p, -1$	$\mathsf{B}, s, +1$	$\#, r, -1$
s	$a, s, 0$	$b, s, 0$	$\mathsf{B}, \mathrm{OK}, 0$	$\#, s, +1$

Nasza maszyna nie ma stanów odrzucających. Jeśli natrafia na sytuację, która jej się nie podoba, to naburmuszona zostaje w miejscu, wykonując trywialne czynności. Obliczenie jest wtedy nieskończone.

Maszyny wielotaśmowe

Jednotaśmowe maszyny Turinga stanowią bardzo prosty formalny model obliczenia. Ze względu na tę prostotę, implementacja nawet prostego algorytmu za pomocą takiej maszyny bywa skomplikowana. Dlatego posługujemy się różnymi, bardziej realistycznymi, uogólnieniami, np. rozważamy maszyny, które mają kilka taśm roboczych. Można wtedy wyodrębnić taśmę wejściową, przeznaczoną tylko do odczytu. My przyjmiemy też model, w którym taśmy są nieskończone tylko w prawo.

Formalnie, niedeterministyczna, *k-taśmowa* maszyna Turinga jest definiowana podobnie do zwykłej, tj. jako krotka $\mathcal{M} = \langle \mathcal{A}, \Sigma, \mathsf{B}, Q, \delta, q_0, A, R \rangle$, ale ma relację przejścia

$$\delta : ((Q - F) \times (\mathcal{A} \cup \{\mathsf{B}\}) \times \Sigma^k) \times (\Sigma^k \times Q \times \{-1, 0, +1\}^{k+1}).$$

Maszyna wielotaśmowa jest *deterministyczna*, gdy δ jest funkcją.

$$\delta : ((Q - F) \times (\mathcal{A} \cup \{\mathsf{B}\}) \times \Sigma^k) \to (\Sigma^k \times Q \times \{-1, 0, +1\}^{k+1}).$$

Konfigurację takiej maszyny stanowi krotka postaci

$$\langle q, \langle w_0, v_0 \rangle, \langle w_1, v_1 \rangle, \ldots, \langle w_k, v_k \rangle \rangle,$$

przedstawiająca kolejno: stan, zawartość taśmy wejściowej i zawartość k taśm roboczych (blanki pomijamy). Interpretacja jest taka, że na i-tej taśmie zapisane jest słowo $w_i v_i$, a i-ta głowica widzi pierwszy symbol słowa v_i (jeśli $v_i = \varepsilon$, to pierwszy blank następujący po słowie w_i.)

Jeśli q jest stanem akceptującym (odp. odrzucającym, końcowym), to konfiguracja jak wyżej jest *akceptująca* (odp. *odrzucająca*, *końcowa*). Konfiguracja *początkowa* ma postać:

$$\mathcal{C}_w = \langle q_0, \langle \varepsilon, w\rangle, \langle \varepsilon, \varepsilon\rangle, \ldots, \langle \varepsilon, \varepsilon\rangle\rangle.$$

For the read-only input tape we assume that it always contains an input word preceded and followed by blanks (so that the beginning and the end of input can be recognized).[7] The meaning of $\langle w_0, v_0\rangle$ is such that $w_0 v_0 = w$ (where w is the input) and the input head is positioned at the first symbol of v_0 (or at the blank to the right of w_0 in case $v_0 = \varepsilon$).

We now define a single-step relation $\to_{\mathcal{M}}$ on configurations.[8] Let

$$\mathcal{C} = \langle q, \langle w_0, v_0\rangle, \langle w_1, v_1\rangle, \ldots, \langle w_k, v_k\rangle\rangle \text{ and } \mathcal{C}' = \langle q', \langle w'_0, v'_0\rangle, \langle w'_1, v'_1\rangle, \ldots, \langle w'_k, v'_k\rangle\rangle.$$

We write $\mathcal{C} \to_{\mathcal{M}} \mathcal{C}'$ (or simply $\mathcal{C} \to \mathcal{C}'$) iff there exists a tuple

$$\langle q, b_0, b_1, \ldots, b_k, c_1, \ldots, c_k, q', j_0, \ldots, j_k\rangle \in \delta$$

such that the following (where c_0 stands for b_0) holds for all $i = 0, \ldots, k$:

- Either $v_i = b_i u_i$ for some u_i, or $v_i = \varepsilon$ and $b_i = \mathsf{B}$; in the latter case we define $u_i = \varepsilon$ for uniformity.
- If $j_i = 0$ then $w'_i = w_i$ and $v'_i = c_i u_i$;
- If $j_i = +1$ then $w'_i = w_i c_i$ and $v'_i = u_i$;
- If $j_i = -1$ and $w_i = x_i d$ for some $d \in \Sigma$, then $w'_i = x_i$ and $v'_i = d c_i u_i$.
- If $j_i = -1$ and $w_i = \varepsilon$ then $w'_i = \varepsilon$ and $v'_i = c_i u_i$.[9]

The symbol $\twoheadrightarrow_{\mathcal{M}}$ stands for the reflexive-transitive closure of $\to_{\mathcal{M}}$ i.e., the least reflexive and transitive relation containing $\to_{\mathcal{M}}$. In other words, $\mathcal{C} \twoheadrightarrow_{\mathcal{M}} \mathcal{C}'$ holds iff there is a sequence of single steps (a computation) of the form:

$$\mathcal{C} = \mathcal{C}_0 \to_{\mathcal{M}} \mathcal{C}_1 \to_{\mathcal{M}} \mathcal{C}_2 \to_{\mathcal{M}} \cdots \to_{\mathcal{M}} \mathcal{C}_m = \mathcal{C}',$$

for some $m \in \mathbb{N}$, possibly zero. We say that the machine *accepts* a word w whenever $\mathcal{C}_w \twoheadrightarrow_{\mathcal{M}} \mathcal{C}_a$ for some accepting configuration $\mathcal{C}_a$. The *language accepted* by $\mathcal{M}$ is

$$L(\mathcal{M}) = \{w \mid \mathcal{M} \text{ accepts } w\}.$$

Maszyny wielotaśmowe potrafią w zasadzie to samo co zwykłe.

Fakt 6.2 *Jeśli $L = L(\mathcal{M})$, dla pewnej maszyny wielotaśmowej $\mathcal{M}$, to także $L = L(\mathcal{M}')$, dla pewnej maszyny jednotaśmowej $\mathcal{M}'$. Co więcej, jeśli $\mathcal{M}$ jest deterministyczna, to $\mathcal{M}'$ też jest deterministyczna.*

[7] Tak sie złożyło, że niektóre fragmenty tego tekstu są napisane po angielsku.

[8] Zmiana konfiguracji polega na zmianie stanu, poprawieniu pozycji wszystkich głowic i wpisaniu odpowiednich symboli na taśmach roboczych. Zawartość taśmy wejściowej nigdy nie ulega zmianie.

[9] Maszyna nie może wyjść poza lewy koniec taśmy.

Dowód: Zamiast używać wielu taśm, można się posłużyć jedną taśmą wielościeżkową. Na $k+1$ ścieżkach zapisujemy zawartość taśmy wejściowej i taśm roboczych, a na kolejnych $k+1$ ścieżkach zapisujemy położenia głowic. Formalnie oznacza to, że alfabet naszej maszyny $\mathcal{M}'$ jest taki: $\Sigma' = \mathcal{A} \times \{\mathsf{B}, \uparrow\} \times (\Sigma \times \{\mathsf{B}, \uparrow\})^k$. Jeden symbol takiego alfabetu zawiera (na współrzędnych nieparzystych) informacje o tym co znajduje się w klatkach $k+1$ taśm maszyny $\mathcal{M}$ położonych pionowo jedna nad drugą. Strzałka jako $2i$-ta współrzędna oznacza, że głowica i-tej taśmy ogląda odpowiednią klatkę.

Przypuśćmy na przykład, że $k = 2$ i że mamy taką konfigurację maszyny $\mathcal{M}$:

	a	b	c				
		$\uparrow$					
b	a	c	a				
$\uparrow$							
		b	b	a	b		
				$\uparrow$			

Odpowiadająca jej taśma maszyny $\mathcal{M}'$ będzie wyglądała tak:

	a	b	c				
		$\uparrow$					
b	a	c	a				
$\uparrow$							
		b	b	a	b		
				$\uparrow$			

Jeden ruch maszyny $\mathcal{M}$ jest naśladowany przez maszynę $\mathcal{M}'$ za pomocą takiej sekwencji ruchów: przesuwając głowicę z lewa na prawo, maszyna sprawdza jakie ciągi $k+1$ symboli widzi maszyna $\mathcal{M}$. Następnie $\mathcal{M}'$ ustala jaki należy wykonać ruch, a w drodze powrotnej z prawa na lewo poprawia pozycje strzałek i zawartość ścieżek. □

Obliczalność

Mówimy, że zbiór $Z \subseteq \mathcal{A}^*$ jest *rekurencyjny* (także: *obliczalny, rozstrzygalny*) jeśli $Z = L(\mathcal{M})$ dla pewnej deterministycznej i totalnej maszyny $\mathcal{M}$. Zbiór Z jest *częściowo obliczalny* (także: *częściowo rozstrzygalny*), jeśli $Z = L(\mathcal{M})$ dla jakiejkolwiek maszyny Turinga $\mathcal{M}$.

Uwaga: Jeśli r jest relacją k-argumentową w zbiorze $\mathcal{A}^*$, to można ją utożsamiać z językiem

$$L_r = \{w_1\$w_2\$\cdots\$w_k \mid (w_1, ..., w_k) \in r\}$$

nad alfabetem $\mathcal{A} \cup \{\$\}$. Mowimy więc o relacjach obliczalnych i częściowo obliczalnych.

Natomiast funkcja $f : \mathcal{A}^* \to \mathcal{B}^*$ jest *obliczalna* (albo *rekurencyjna*) wtedy, gdy istnieje deterministyczna totalna maszyna Turinga z wyróżnioną taśmą *wyjściową*,[10] która dla dowolnego

[10] Często żąda się, aby taśma wyjściowa była przeznaczona tylko do zapisu.

wejścia $w \in \mathcal{A}^*$ zatrzymuje się ze słowem $f(w) \in \mathcal{B}^*$ zapisanym na taśmie wyjściowej.

Fakt 6.3 *Jeśli Z jest częściowo obliczalny, to $Z = L(\mathcal{M})$ dla pewnej deterministycznej maszyny Turinga $\mathcal{M}$.*

Dowód: Niech $Z = L(\mathcal{M}')$, gdzie $\mathcal{M}' = \langle \Sigma, Q, \delta, q_0, A, R \rangle$ jest maszyną jednotaśmową. Bez straty ogólności możemy zakładać, że dla dowolnych q, a istnieje tyle samo (powiedzmy, N) piątek $\langle q, a, a', p, i \rangle \in \delta$. (Jeśli jest mniej, to którąś powtarzamy kilka razy.) Każda liczba $m = 1, \ldots, N$ oznacza więc pewien możliwy wybór.

Maszyna deterministyczna symuluje działanie maszyny $\mathcal{M}'$ za pomocą dwóch taśm roboczych.

Na taśmie pierwszej generowane są systematycznie (w pewnej ustalonej kolejności) wszystkie ciągi skończone $m_1, m_2, \ldots, m_k$ liczb mniejszych lub równych N. Każdy z tych ciągów odpowiada pewnemu fragmentowi początkowemu jakiegoś obliczenia maszyny $\mathcal{M}$: tego, które w i-tym kroku dokonuje wyboru m_i.

Po wypisaniu kolejnego nowego ciągu $m_1, m_2, \ldots, m_k$, na drugiej taśmie wykonuje się symulację k kroków odpowiedniego obliczenia maszyny $\mathcal{M}$.

Maszyna zatrzymuje się, jeśli symulowane obliczenie osiągnie stan akceptujący.

Inaczej mówiąc, maszyna $\mathcal{M}$ systematycznie przeszukuje wszerz drzewo wszystkich obliczeń maszyny niedeterministycznej. W ten sposób, prędzej czy później, wykryje obliczenie akceptujące, jeśli takie istnieje. □

Uwaga: Metoda użyta w dowodzie faktu 6.3 działa też dla maszyn wielotaśmowych (używamy jednej dodatkowej taśmy).

Fakt 6.4 *Następujące warunki są równoważne:*

1. *Z jest zbiorem obliczalnym;*
2. *$-Z$ jest zbiorem obliczalnym;*
3. *Z i $-Z$ są częściowo obliczalne.*

Dowód: Jeśli dwie różne deterministyczne maszyny akceptują języki Z i $-Z$, to należy uruchomić je jednocześnie i zobaczyć, która zaakceptuje słowo wejściowe. Tę pracę może wykonać jedna maszyna deterministyczna. Ona jest totalna, bo każde słowo jest albo w Z albo w $-Z$. Reszta jest łatwa. □

Twierdzenie 6.5 *Język L jest częściowo obliczalny wtedy i tylko wtedy, gdy jest rekurencyjnie przeliczalny, tj. $L = L(G)$ dla pewnej gramatyki G (typu zero).*

Dowód: ($\Leftarrow$) Niech $L = L(G)$. Konstruujemy maszynę niedeterministyczną $\mathcal{M}$ z jedną taśmą pomocniczą. Na początku umieszcza się na tej taśmie symbol początkowy gramatyki, a następnie niedeterministycznie wykonuje redukcje. Tj. w każdej fazie pracy, maszyna wybiera redukcję $x \Rightarrow y$ i zastępuje na taśmie pewne wystąpienie słowa x przez y. (To może wymagać

przesunięcia pozostałej zawartości taśmy w lewo lub w prawo.) Potem następuje sprawdzenie, czy zawartość obu taśm, wejściowej i roboczej, jest taka sama. Jeśli tak, to maszyna akceptuje.

($\Rightarrow$) Niech $L = L(\mathcal{M})$ i załóżmy, że $\mathcal{M}$ jest deterministyczną maszyną jednotaśmową. Dla uproszczenia załóżmy, że $\mathcal{M}$ ma tylko jeden stan końcowy q_f (akceptujący). Definiujemy gramatykę G, której alfabet nieterminalny składa się ze stanów i symboli maszyny $\mathcal{M}$, oraz dodatkowo z nawiasów kwadratowych (na oznaczenie początku i końca konfiguracji).

Zbiór produkcji gramatyki G dzieli się na dwie części. Pierwszą grupę stanowią produkcje pozwalające wyprowadzić dowolne słowo postaci $w\,[q_0 w]$.[11] Druga grupa produkcji pozwala na symulację obliczenia maszyny w prawej części słowa. Są to takie produkcje:

- $qa \Rightarrow bp$, gdy $\delta(q,a) = (b,p,+1)$;
- $qa \Rightarrow pb$, gdy $\delta(q,a) = (b,p,0)$;
- $q] \Rightarrow bp]$, gdy $\delta(q,\mathsf{B}) = (b,p,+1)$;
- $q] \Rightarrow pb]$, gdy $\delta(q,\mathsf{B}) = (b,p,0)$;
- $cqa \Rightarrow pcb$ i $[qa \Rightarrow [p\mathsf{B}b$, gdy $\delta(q,a) = (b,p,-1)$;
- $cq] \Rightarrow pcb]$, gdy $\delta(q,\mathsf{B}) = (b,p,-1)$;
- $aq_f \Rightarrow q_f$ i $q_f a \Rightarrow q_f$, gdy $a \neq [,]$;
- $[q_f] \Rightarrow \varepsilon$.

Zauważmy, że wówczas zachodzi równoważność:

$$[q_0 w] \twoheadrightarrow_G [q_f] \quad \text{wtedy i tylko wtedy gdy} \quad w \in L(\mathcal{M}),$$

bo każdy ciąg postaci $[q_0 w] \to_G x_1 \to_G x_2 \to_G \cdots \to_G [q_f]$ musi reprezentować obliczenie maszyny dla słowa w, zakończone „wycieraniem" wszystkich symboli oprócz q_f.

Jeśli odpowiednio dobierzemy produkcje z pierwszej grupy, to każdy ciąg redukcji, w którym występują słowa zawierające stany maszyny $\mathcal{M}$, musi zaczynać się od $\xi_0 \twoheadrightarrow_G w\,[q_0 w]$, dla pewnego w. A zatem $L(G) = L(\mathcal{M})$, bo wyprowadzenie w gramatyce G, kończące się słowem terminalnym może wyglądać tylko tak:

$$\xi_0 \twoheadrightarrow w\,[q_0 w] \twoheadrightarrow w\,[q_f] \to w. \qquad \square$$

Uwaga: Jeśli gramatyka G w pierwszej części dowodu jest monotoniczna, to maszyna $\mathcal{M}$ używa tylko tyle swojej taśmy, ile zajmuje słowo wejściowe. Taka maszyna nazywa się *automatem liniowo ograniczonym* (LBA).

Na odwrót, jeśli jednotaśmowa maszyna $\mathcal{M}$ jest liniowo ograniczona, to można pokazać, że język $L(\mathcal{M})$ jest monotoniczny. Wymaga to nieco oszczędniejszej symulacji niż wyżej.

[11] Ćwiczenie: jak to zrobić?

7 Rozstrzygalność problemów decyzyjnych

Oczywiście sens pojęcia obliczalności jest taki: zbiór słów Z jest obliczalny wtedy, gdy istnieje algorytm, który dla danego słowa w zawsze poprawnie odpowiada "tak" lub "nie" na pytanie, czy $w \in Z$. Takie pytania nazywamy często "problemami decyzyjnymi". Formalnie, *problem decyzyjny* to po prostu zbiór słów nad ustalonym alfabetem. W praktyce "problemami decyzyjnymi" nazywamy na przykład takie pytania:

– *"Czy dany graf skończony ma ścieżkę Hamiltona?"*

– *"Czy dana formuła rachunku predykatów jest tautologią?"*[12]

gdzie daną wejściową, czyli *instancją* problemu, nie jest słowo, ale jakiś skończony obiekt (formuła, graf, itp.). Jest to możliwe wtedy, gdy określimy sposób kodowania danych w postaci słów (np. skończony graf można przedstawić za pomocą słowa, wypisując listę jego wierzchołków i krawędzi).

Po pierwsze, nasze pojęcie "(nie)rozstrzygalnego problemu decyzyjnego" ma zastosowanie tylko do tych obiektów matematycznych, które dają się przedstawić w *skończony* sposób. Nie jest więc problemem decyzyjnym zadanie:

– *"Czy dany graf nieskończony jest spójny?"*

Po drugie, poprawne sformułowanie problemu wymaga poprawnego kodowania. Albo więc musimy tak kodować grafy, aby każde słowo nad ustalonym alfabetem reprezentowało pewien graf, albo problem ścieżki Hamiltona trzeba sformułować tak:

– *"Czy dane słowo jest kodem grafu, który ma ścieżkę Hamiltona?"*

Wtedy słowa nie będące kodami nie powinny być akceptowane.

Nierozstrzygalność

Bardzo podstawowa idea, która kiedyś nie była wcale tak oczywista, jak się to może dzisiaj nam wydawać, wynika z prostej obserwacji. Ponieważ każdy algorytm też można zapisać za pomocą znaków pewnego alfabetu, to i on może stanowić daną wejściową dla innego algorytmu. Nie ma wiec istotnej różnicy miedzy danymi i programem. Maszyna, która wykonuje podany na wejściu algorytm, to nic innego jak interpreter pewnego języka programowania.

Nasze algorytmy to w istocie maszyny Turinga. Każdą maszynę $\mathcal{M}$ nad alfabetem $\mathcal{A}$ możemy przedstawić za pomocą pewnego słowa $kod_{\mathcal{M}} \in \mathcal{A}^*$. Można to zrobić rutynowymi metodami, kodując wszystkie informacje o maszynie, w tym relację (funkcję) przejścia, za pomocą odpowiedniego ciągu znaków. Dwie różne maszyny będą przy tym zawsze miały różne kody. Dla danego słowa v, niech $\mathcal{M}_v$ oznacza maszynę o kodzie v. Jeśli takiej maszyny nie ma, to przyjmujemy za $\mathcal{M}_v$ jakąkolwiek ustaloną maszynę.

Rozpatrzmy teraz relację (tzw. *problem stopu*)

$$S = \{(v, w) \mid w \in L(\mathcal{M}_v)\},$$

[12] Nazwa „problem decyzyjny" (Entscheidungsproblem) została po raz pierwszy użyta właśnie w odniesieniu do tego zadania.

czyli w istocie język $S = \{\, v\$w \mid w \in L(\mathcal{M}_v)\}$. *Uniwersalna maszyna Turinga*, to maszyna akceptująca język S. Maszyna ta, dla danych słów v, w, interpretuje słowo v jako kod pewnej maszyny $\mathcal{M}$ i symuluje jej zachowanie dla wejścia w. Widzimy więc, że:

Wniosek 7.1 *Język uniwersalny S jest rekurencyjnie przeliczalny.*

Inaczej mówiąc, decyzyjny problem stopu (*„Czy dana maszyna akceptuje dane słowo?”*) jest częściowo rozstrzygalny. Za chwile się jednak okaże, że nie jest rozstrzygalny. W tym celu zbadamy jego szczególny przypadek. Rozpatrzmy mianowicie zbiór:

$$K = \{v \mid v \in L(\mathcal{M}_v)\}$$

Lemat 7.2 *Zbiór $-K$ nie jest rekurencyjnie przeliczalny, a zatem K nie jest rekurencyjny.*

Dowód: Przypuśćmy, że istnieje taka maszyna $\mathcal{M}$, że $L(\mathcal{M}) = -K$. Niech v będzie takie, że $\mathcal{M} = \mathcal{M}_v$. Jeśli $v \in K$, to $v \in L(\mathcal{M}_v) = -K$. Zatem $v \notin K$. Ale wtedy $v \notin L(\mathcal{M}_v) = -K$, czyli $v \in K$. Tak czy owak, jest źle. □

Twierdzenie 7.3 *Problem stopu dla maszyn Turinga jest nierozstrzygalny.*

Dowód: W przeciwnym razie przynależność do zbioru K byłaby też rozstrzygalna. Aby stwierdzić, czy $v \in K$, wystarczyłoby sprawdzić, czy $(v, v) \in S$. □

Uwaga: Jeżeli interesuje nas tylko język $L(\mathcal{M})$, to stany odrzucające maszyny $\mathcal{M}$ są nieistotne i wygodnie jest nie odróżniać od siebie obliczeń odrzucających i nieskończonych. Dlatego czasem przyjmuje się $R = \varnothing$ i pisze $\mathcal{M} = \langle \Sigma, Q, \delta, q_0, F\rangle$. Wtedy stwierdzenia „maszyna akceptuje dane słowo” i „maszyna zatrzymuje się dla danego słowa” są równoważne.

Problem stopu jest często formułowany tak: *„Czy dana maszyna $\mathcal{M}$ zatrzymuje się dla danego słowa w?”* i w tej wersji też jest nierozstrzygalny.

Redukowalność: Metoda użyta w dowodzie Twierdzenia 7.3 jest typowa. Mówimy, że problem (zbiór słów) A *redukuje się* do problemu B i piszemy $A \leq B$, gdy istnieje funkcja obliczalna f o takiej własności:

$$w \in A \quad \Longleftrightarrow \quad f(w) \in B.$$

W praktyce oznacza to tyle, że istnieje algorytm przekształcający każdą możliwą instancję w problemu A w instancję $f(w)$ problemu B, w ten sposób, że pytanie „Czy $w \in A$?” można sprowadzić do pytania „Czy $f(w) \in B$?”.

Fakt 7.4 *Niech $A \leq B$. Jeśli B jest rozstrzygalny (rekurencyjnie przeliczalny), to A też jest rozstrzygalny (rekurencyjnie przeliczalny). Ponadto, jeśli $-B$ jest rekurencyjnie przeliczalny, to $-A$ też jest rekurencyjnie przeliczalny.*

Dowód: Łatwy. □

Metoda redukcji jest standardowym sposobem dowodzenia nierozstrzygalności. Oto przykład jej zastosowania. Przez *problem przepisywania słów* rozumiemy takie zadanie:

Dana gramatyka typu zero i dwa słowa w i v. Czy istnieje wyprowadzenie $G \vdash w \twoheadrightarrow v$?

Twierdzenie 7.5 *Problem przepisywania słów jest nierozstrzygalny.*

Dowód: Wiemy już, że gramatyki typu zero generują wszystkie języki rekurencyjnie przeliczalne. Co więcej, dla danej maszyny Turinga $\mathcal{M}$, potrafimy *efektywnie* skonstruować[13] taką gramatykę $G_{\mathcal{M}}$, że $L(G_{\mathcal{M}}) = L(\mathcal{M})$. Zatem, dla dowolnego słowa w:

$$w \in L(\mathcal{M}) \quad \text{wtedy i tylko wtedy, gdy} \quad G_{\mathcal{M}} \vdash \xi_0 \twoheadrightarrow w.$$

Mamy więc redukcję problemu stopu do problemu słów. Funkcja, która realizuje tę redukcję, przypisuje każdej parze $(\mathcal{M}, w)$ (instancji problemu stopu) trójkę $(G_{\mathcal{M}}, \xi_0, w)$, czyli instancję problemu słów.

Jeszcze ściślej, jest to funkcja przekształcająca każde słowo x (nad pewnym ustalonym alfabetem) reprezentujące parę postaci $(\mathcal{M}, w)$, w słowo $f(x)$, które (w jakiś ustalony sposób) reprezentuje odpowiednią trójkę. □

Problemy dotyczące maszyn Turinga

Wiele problemów dotyczących maszyn Turinga jest nierozstrzygalnych, bo są to zadania bardzo bliskie problemowi stopu. Przykładem jest pytanie: „*Czy dana maszyna akceptuje słowo puste?*" W większości przypadków nierozstrzygalność takich zadań dowodzi się w rutynowy sposób. Jest nawet twierdzenie Rice'a, które mówi, że każdy problem dotyczący *języków* akceptowanych przez maszyny Turinga (a nie samych maszyn) musi być albo trywialny albo nierozstrzygalny. Jako przykład rozpatrzmy następujący *problem bezkontekstowości*:

Dana maszyna Turinga M, czy język $L(M)$ jest bezkontekstowy?

Fakt 7.6 *Problem bezkontekstowości jest nierozstrzygalny.*

Dowód: Mamy udowodnić, że nie istnieje algorytm rozstrzygający ten problem. Inaczej, że nie istnieje deterministyczna maszyna, która:

- zatrzymuje się dla dowolnego słowa wejściowego;
- akceptuje wtedy i tylko wtedy, gdy słowo wejściowe jest kodem maszyny Turinga akceptującej język bezkontekstowy.

Przypuśćmy, że taka maszyna istnieje i nazwijmy ją T. Pokażemy, że wtedy rozstrzygalny jest problem stopu: *Dana maszyna Turinga M i słowo w, czy $w \in L(M)$?* Ale problem stopu jest nierozstrzygalny, więc maszyna T nie może istnieć.

[13] Tj. podać algorytm realizujący tę konstrukcję.

Jak rozstrzygać problem stopu z pomocą maszyny T? Mamy dany kod maszyny Turinga M i słowo w. Możemy zakładać bez straty ogólności, że maszyna M albo akceptuje w albo się zapętla. (Nie ma takiej opcji jak zatrzymanie w stanie odrzucającym.)

Mając słowo w i kod M potrafimy obliczyć kod nowej maszyny, którą oznaczymy przez N_{Mw}. Ta maszyna N_{Mw} dla danego słowa wejściowego x działa tak:

1. Najpierw symuluje zachowanie maszyny M na wejściu w.

2. Potem sprawdza, czy słowo x ma postać $a^n b^n c^n$ dla pewnego $n \in \mathbb{N}$.

Zachowanie maszyny N_{Mw} w pierwszej fazie w ogóle nie zależy od wejścia x. Jeśli $w \notin L(M)$, to ta faza się nie kończy i maszyna N_{Mw} się zapętla. Dopiero w drugiej fazie istotne jest x. A zatem jeśli $w \notin L(M)$, to $L(N_{Mw}) = \varnothing$, a jeśli $w \in L(M)$, to $L(N_{Mw}) = \{a^n b^n c^n \mid n \in \mathbb{N}\}$. Język pusty jest bezkontekstowy, a język $\{a^n b^n c^n \mid n \in \mathbb{N}\}$ nie jest. Czyli:

$$w \in L(M) \quad \text{wtedy i tylko wtedy, gdy} \quad \text{język } L(N_{Mw}) \text{ nie jest bezkontekstowy.} \qquad (*)$$

Ostatecznie, dla danych M, w należy zapytać, czy język $L(N_{Mw})$ jest bezkontekstowy (tj. uruchomić maszynę T na wejściu M, w). Jeśli tak, to $w \notin L(M)$, w przeciwnym razie $w \in L(M)$.

Uwaga: Istotne jest to, że konstrukcja kodu $N_{M,w}$ z kodu M i słowa w jest efektywna, czyli obliczalna i może być zrealizowana przez jakąś maszynę Turinga. Równoważność (*) stwierdza więc to, że problem stopu redukuje się do problemu

Dana maszyna Turinga M, czy język $L(M)$ nie jest bezkontekstowy?

a więc do dopełnienia (!) problemu postawionego na początku. □

Problem odpowiedniości Posta

Problem odpowiedniości Posta (PCP) jest klasycznym przykładem problemu nierozstrzygalnego. Formułujemy go tak:

Dany jest skończony zbiór par słów $\{(x_i, y_i) \mid i = 1, \ldots, n\}$. Czy istnieje taki niepusty ciąg $i_1, i_2, \ldots, i_m$, że zachodzi równość $x_{i_1} x_{i_2} \ldots x_{i_m} = y_{i_1} y_{i_2} \ldots y_{i_m}$?

Poszukiwany ciąg $i_1, i_2, \ldots, i_m$ nazywamy *rozwiązaniem* danej instancji PCP. Także słowo

$$x_{i_1} x_{i_2} \ldots x_{i_m}$$

bywa wtedy nazywane rozwiązaniem.

Problem odpowiedniości $\{(x_i, y_i) \mid i = 1, \ldots, n\}$ przedstawia się często w taki sposób:

x_1	x_2	$\ldots$	x_n
y_1	y_2	$\ldots$	y_n

Na przykład taki system:

a^2	b^2	ab^2
a^2b	ba	b

ma rozwiązanie 1213 (bo $a^2 \cdot b^2 \cdot a^2 \cdot ab^2 = a^2b^2a^3b^2 = a^2b \cdot ba \cdot a^2b \cdot b$), a systemy

a^2b	a
a^2	ba^2

ba^2	ba^2
b	a^2b

nie mają rozwiązań.

Twierdzenie 7.7 *Problem odpowiedniości Posta jest nierozstrzygalny.*

Dowód: Redukcja problemu przepisywania słów do PCP. Niech $G = \langle \mathcal{A}, \mathcal{N}, P, \xi_0 \rangle$ bedzie gramatyką typu zero i niech $\Sigma = \mathcal{A} \cup \mathcal{N}$. Ustalmy slowa $w, v \in \Sigma^*$. Skonstruujemy (efektywnie) problem odpowiedniości P, który będzie miał rozwiązanie wtedy i tylko wtedy, gdy $G \vdash w \twoheadrightarrow v$.

Alfabet będzie taki:

$$\Delta = \Sigma \cup \{\overline{a} \mid a \in \Sigma\} \cup \{*, \overline{*}\} \cup \{\texttt{[}, \texttt{]}\}.$$

Jeśli $u = a_1 \dots a_r$, to symbol $\overline{u}$ oznacza oczywiście słowo $\overline{a}_1 \dots \overline{a}_r$.

Reguły przedstawimy tabelką, w której a oznacza dowolny symbol alfabetu Σ, a $\alpha \Rightarrow \beta$ jest dowolną produkcją gramatyki (w zadaniu mamy wszystkie pary $(\beta, \overline{\alpha})$ i $(\overline{\beta}, \alpha)$).

$\texttt{[}w*$	a	$\overline{a}$	$*$	$\overline{*}$	$\texttt{]}$	$\texttt{]}$	β	$\overline{\beta}$
$\texttt{[}$	$\overline{a}$	a	$\overline{*}$	$*$	$\overline{*}v\texttt{]}$	$*\overline{v}\texttt{]}$	$\overline{\alpha}$	α

Na przykład jeśli $w = \texttt{baca}$, $v = \texttt{owad}$, a regułami naszej gramatyki są

$$\texttt{ba} \Rightarrow \texttt{ow} \qquad \texttt{ca} \Rightarrow \texttt{ad},$$

to otrzymujemy zadanie:

$\texttt{[baca}*$	$\texttt{a}$	$\overline{\texttt{a}}$	$\cdots$	$*$	$\overline{*}$	$\texttt{]}$	$\texttt{]}$	$\texttt{ow}$	$\overline{\texttt{ow}}$	$\texttt{ad}$	$\overline{\texttt{ad}}$
$\texttt{[}$	$\overline{\texttt{a}}$	$\texttt{a}$	$\cdots$	$\overline{*}$	$*$	$*\overline{\texttt{owad}}\texttt{]}$	$\overline{*}\texttt{owad]}$	$\overline{\texttt{ba}}$	$\texttt{ba}$	$\overline{\texttt{ca}}$	$\texttt{ca}$

Załóżmy teraz, że $w = w_0 \to_G w_1 \to_G w_2 \to_G \cdots \to_G w_k = v$, i dla ustalenia uwagi przyjmijmy, że k jest parzyste. Wtedy nasz problem odpowiedniości można rozwiązać tak:

$$\texttt{[}w_0 * \cdot \overline{w}_1\overline{*} \cdot w_2 * \cdots * \overline{w}_{k-1}\overline{*} \cdot w_k \cdot \texttt{]} = \texttt{[} \cdot w_0 * \cdot \overline{w}_1\overline{*} \cdot w_2 * \cdots * \overline{w}_{k-1} \cdot \overline{*}w_k\texttt{]}$$

Rzeczywiście, jeśli mamy np. $w_0 = x\alpha y \to_G x\beta y = w_1$, przez zastosowanie produkcji $\alpha \Rightarrow \beta$, to słowa w_0* i $\overline{w}_1\overline{*}$ można przedstawić jako konkatenacje odpowiadających sobie słów z dolnej i górnej części tabelki. Podobnie dla słów $\overline{w}_1\overline{*}$ i w_2* i tak dalej.

Na dodatek, jeśli mamy jakiekolwiek rozwiązanie problemu P, to takie rozwiązanie musi wyznaczać pewne wyprowadzenie $w \twoheadrightarrow_G v$. Rozwiązanie musi się zaczynać od pary ($\texttt{[}w*$, $\texttt{[}$), jest to bowiem jedyna para, która zaczyna się tak samo (po to są kreski). Jedynym sposobem zgodnego przedłużenia słów z tej pary jest dopisanie do $\texttt{[}$ słowa $w*$. To jednak oznacza, że do słowa $\texttt{[}w*$ musimy dopisać słowo postaci $\overline{u}_1\overline{*}$ gdzie $w \twoheadrightarrow_G u_1$. Zatem mamy teraz słowa

$$\texttt{[}w * \overline{u}_1\overline{*} \qquad\qquad \texttt{[}w*$$

i musimy do drugiego z nich dopisać $\overline{u_1}\overline{*}$. Ale wtedy do pierwszego dopiszemy u_2*, gdzie $u_1 \twoheadrightarrow_G u_2$ i tak dalej. To sie może skończyć tylko użyciem jednej z par zawierających `]`, co oznacza, że $w \twoheadrightarrow v$.

Uwaga: Warunek $u_1 \twoheadrightarrow_G u_2$ powyżej nie musi oznaczać $u_1 \to u_2$. Na przykład przekształcenie bacy w owada może wyglądać nie tylko tak: [`baca`$*\overline{$`owca`$*}$`owad`], ale także tak:

$$[\texttt{baca}*\overline{\texttt{baca}*}\texttt{owad} * \overline{\texttt{owad}}].$$ □

Przykładem zastosowania PCP jest taki fakt:

Twierdzenie 7.8 *Problem pustości dla języków kontekstowych jest nierozstrzygalny.*

Dowód: Dla danej instancji P problemu odpowiedniości Posta, można łatwo skonstruować maszynę Turinga, rozpoznającą dokładnie te słowa, które są rozwiązaniami tego systemu. Maszyna ta nie używa więcej taśmy, niż zajmowało słowo wejściowe. Modyfikując nieco konstrukcję z dowodu twierdzenia 6.5, możemy z tej maszyny uzyskać gramatykę monotoniczną generującą zbiór rozwiązań systemu P. W ten sposób zredukujemy problem odpowiedniości Posta do problemu pustości dla języków kontekstowych. □

Układanie kafelków

Istnieje wiele wariantów problemu układania kafelków (ang.: *tiling problem*). Zwykle zakłada się, że dany jest skończony zbiór T *kafelków* i relacje $H, V \subseteq T \times T$ zgodności poziomej i pionowej. *Pokryciem* zbioru $A \subseteq \mathbb{Z} \times \mathbb{Z}$ nazwiemy taką funkcję $P : A \to T$, że:

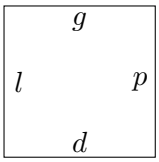
$$\langle P(x,y), P(x+1,y)\rangle \in H, \text{ zawsze gdy } \langle x,y\rangle \in A \text{ i } \langle x+1,y\rangle \in A;$$
$$\langle P(x,y), P(x,y+1)\rangle \in V, \text{ zawsze gdy } \langle x,y\rangle \in A \text{ i } \langle x,y+1\rangle \in A.$$

Pokrycie może spełniać jakiś *warunek początkowy*, np. $P(0,0) = t_0$.

Jeśli $T \subseteq C^4$, gdzie C jest zbiorem *kolorów*, to mamy do czynienia z *dominem*. Relacje H, V definiujemy wtedy następująco:

$$\begin{aligned} \langle t, t'\rangle \in H &\quad\Leftrightarrow\quad p(t) = l(t');\\ \langle t, t'\rangle \in V &\quad\Leftrightarrow\quad g(t) = d(t'), \end{aligned}$$

gdzie $g(t)$, $p(t)$, $d(t)$, $l(t)$ oznaczają cztery współrzędne klocka t interpretowane jako kolory czterech boków kwadratu (w kolejności góra, prawo, dół, lewo[14]).



Dwie najbardziej znane wersje problemu układania kafelków są takie:

[14] Kafelków nie wolno obracać!

A) Dany jest zbiór T i relacje H, V; czy istnieje pokrycie całej płaszczyzny $\mathbb{Z} \times \mathbb{Z}$?

B) Dany jest zbiór T i relacje H, V, oraz kafelek początkowy $t_0 \in T$; czy istnieje pokrycie P pierwszej ćwiartki $\mathbb{N} \times \mathbb{N}$ spełniające warunek początkowy $P(0,0) = t_0$?

Oba powyższe problemy są nierozstrzygalne. Dowód nierozstrzygalności problemu w wersji (A) jest jednak znacznie trudniejszy niż dla wersji (B). Dlatego my zajmiemy się tylko tym drugim.

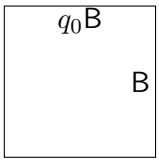
Kodowanie maszyny Turinga: Aby udowodnić nierozstrzygalność problemu (B) zredukujemy problem stopu dla deterministycznych maszyn Turinga do nieistnienia pokrycia. To jest, dla danej deterministycznej maszyny M skonstruujemy efektywnie domino T wraz z klockiem początkowym $t_0 \in T$ i udowodnimy

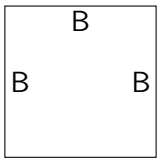
Lemat 7.9 *Następujące warunki są równoważne:*

1) *Maszyna M ma nieskończone obliczenie dla pustego słowa wejściowego.*
2) *Istnieje takie pokrycie P zbioru $\mathbb{N} \times \mathbb{N}$ klockami domina T, że $P(0,0) = t_0$.*

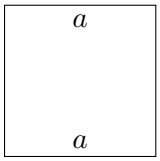
Zbiór kolorów domina T składa się z następujących elementów:
– symbole alfabetu maszyny M;
– kolory postaci qa, gdzie q jest stanem i a symbolem alfabetu M;
– kolory postaci $\overleftarrow{q}$ i $\overrightarrow{q}$, gdzie q jest stanem M;
– jeden kolor „neutralny” (tego koloru są na obrazkach krawędzie bez oznaczeń).

Teraz opiszemy klocki domina T. Zaczniemy od dwóch szczególnych klocków:

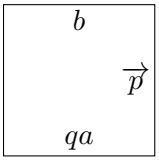
$t_0 =$ 

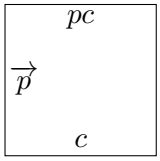
$t_1 =$ 

Powyżej q_0 to stan początkowy. Dla każdego symbolu a z alfabetu maszyny mamy klocek:

$t_a =$ 

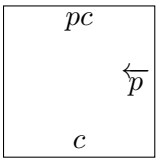
Dalsze klocki reprezentują funkcję przejścia maszyny. Jeśli $\delta(q, a) = (b, p, +1)$, to dla dowolnego c z alfabetu maszyny mamy klocki:[15]

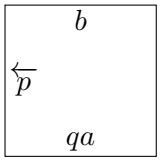
$t^L_{qa} =$ 

$t^R_{qa} =$ 

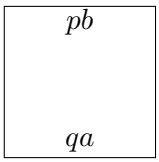
[15] Jeśli $t^R_{qa} = t^R_{q'a'}$, to mamy wspólny „prawy” klocek dla dwóch różnych instrukcji. Ale to nic nie szkodzi.

Analogicznie, dla $\delta(q,a) = (b,p,-1)$:[16]

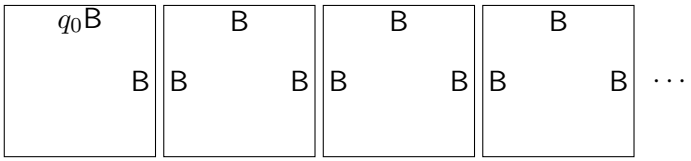
$t^L_{qa} =$ 

$t^R_{qa} =$ 

W przypadku $\delta(q,a) = (b,p,0)$ mamy tylko jeden klocek:

$t_{qa} =$ 

Dowód lematu 7.9 naszkicujemy w trudniejszym kierunku (2) ⇒ (1). Załóżmy że pokrycie P spełnia warunek początkowy $P(0,0) = t_0$. Do klocka t_0 pasuje z prawej strony tylko klocek t_1, tj. musi być $P(1,0) = t_1$. Ale do klocka t_1 pasuje z prawej też tylko t_1, więc $P(n,0) = t_1$ dla wszystkich n. A zatem pierwsza „warstwa" naszego pokrycia wygląda tak:



Z górnej krawędzi tej warstwy odczytamy ciąg $q_0\mathsf{BBBB}\ldots$, który stanowi zapis konfiguracji początkowej maszyny. Przypuśćmy teraz (indukcja), że ciąg kolorów na górnej krawędzi warstwy m jest postaci $a_1,\ldots,a_n,qa,b_1,\ldots b_k,\mathsf{B},\mathsf{B},\ldots$ i przedstawia konfigurację otrzymaną po m krokach obliczenia maszyny M. W szczególności, istnieje dokładnie jedno takie n, że klocek $P(n,m)$ ma na górnej krawędzi kolor postaci qa. Wtedy klocek $P(n,m+1)$ musi mieć qa na dolnej krawędzi, czyli musi być postaci t_{qa}, t^L_{qa} lub t^R_{qa}. W pierwszym przypadku, dla wszystkich pozycji $n' \neq n$ musimy mieć $P(n',m+1) = t_x$ dla odpowiedniego symbolu x. W pozostałych dwóch przypadkach z lewej lub z prawej strony klocka $P(n,m+1)$ stoi odpowiednio klocek t^R_{qa} lub t^L_{qa}, bo żaden inny nie pasuje. A inne klocki też muszą być postaci t_x. W ten sposób „przenosimy" informację z dolnej krawędzi warstwy $m+1$ do górnej krawędzi, zmieniając przy tym zapisaną konfigurację.

Pokazaliśmy przez indukcję, że poprawne pokrycie zbioru $\{0,\ldots,m\}\times\mathbb{N}$ oznacza, że maszyna wykonuje co najmniej $m+1$ kroków dla pustego słowa wejściowego. Skoro więc istnieje pokrycie całej pierwszej ćwiartki, to obliczenie musi być nieskończone. Istotnie, funkcja przejścia M nie jest określona dla stanu końcowego q_a, a więc nie ma klocka, który na dolnej krawędzi miałby kolor postaci $q_a a$.

Wniosek 7.10 *Problem układania kafelków (wersja B) jest nierozstrzygalny.*

[16] Bez straty ogólności zakładamy, że maszyna nigdy nie wykonuje takiego kroku, gdy głowica znajduje się nad pierwszą klatką taśmy.

8 Złożoność obliczeniowa

Od tej pory zakładamy, że rozważane maszyny Turinga mają taśmę wejściową tylko do odczytu i dowolną liczbę taśm roboczych. Zakładamy też, że maszyna zawsze czyta całe słowo wejściowe. Oznacza to w szczególności, że każde obliczenie składa się z co najmniej tylu kroków ile wynosi długość słowa wejściowego.

Konwencja: Symbol n rezerwujemy na oznaczenie długości słowa wejściowego.

Mówimy, że (deterministyczna lub nie) maszyna Turinga ma *złożoność czasową* $T(n)$, jeśli dla dowolnego n każde obliczenie tej maszyny dla każdego słowa wejściowego w o długości n składa się z co najwyżej $T(n)$ kroków. Zgodnie z przyjętym wcześniej założeniem musi zachodzić warunek $T(n) \geq n$.

Maszyna ma *złożoność pamięciową* $S(n)$ jeśli dla dowolnego n w każdym obliczeniu dla dowolnego słowa wejściowego w o długości n, maszyna używa co najwyżej $S(n)$ klatek na każdej taśmie roboczej. Przyjmujemy $S(n) \geq 1$, bo klatki wskazywane przez głowice maszyny uważamy za używane.

Zatem, mówiąc o złożoności czasowej $T(n)$ i pamięciowej $S(n)$, mamy w istocie na myśli $\max\{T(n), n\}$ i $\max\{S(n), 1\}$.

Uwaga: Każda maszyna o złożoności czasowej $T(n)$ ma też złożoność pamięciową $T(n)$, bo w czasie $T(n)$ można zapisać co najwyżej $T(n)$ klatek taśmy.

Fakt 8.1 *(1) Jeśli maszyna ma określoną złożoność czasową lub pamięciową, to język akceptowany przez tę maszynę jest obliczalny.*

(2) Każdy język obliczalny jest akceptowany przez pewną deterministyczną maszynę o określonej złożoności pamięciowej $S(n)$ i czasowej $T(n)$. Przy tym funkcje $S(n)$ i $T(n)$ są obliczalne.

Dowód: (1) Dla maszyn deterministycznych teza jest dość oczywista, bo wtedy dla dowolnego słowa wejściowego maszyna musi się zatrzymać albo zapętlić przez powtórzenie konfiguracji. Można więc skonstruować inną maszynę, która zawsze się zatrzymuje.

Załóżmy więc, że niedeterministyczna maszyna $\mathcal{M}$ ma złożoność pamięciową $S(n)$. Niech C_w oznacza konfigurację początkową dla słowa wejściowego w o długości n.

Rozpatrzmy następujący deterministyczny algorytm generujący listę wszystkich konfiguracji $\mathcal{C}$ maszyny $\mathcal{M}$, które są osiągalne z C_w. Na początku zapisujemy na liście konfigurację $\mathcal{C}_w$. Następnie powtarzamy taką operację:

(*) *Dla każdej konfiguracji $\mathcal{C}$ z listy:*
znajdź wszystkie takie konfiguracje $\mathcal{C}'$, że $\mathcal{C} \to_{\mathcal{M}} \mathcal{C}'$.
Dopisz do listy wszystkie takie $\mathcal{C}'$, których tam jeszcze nie ma.

Ponieważ liczba wszystkich konfiguracji jest skończona, więc za którymś razem operacja (*) nie dopisze już do listy żadnej nowej konfiguracji. To znaczy, że mamy już wszystkie konfiguracje osiągalne z $\mathcal{C}_w$ i pozostaje sprawdzić, czy jest wśród nich jakaś konfiguracja akceptująca.

(2) Mamy deterministyczną i totalną maszynę $\mathcal{M}$ akceptującą dany język. Nietrudno ją przerobić na maszynę obliczającą maksymalny czas pracy $T(n)$ maszyny $\mathcal{M}$ dla wejścia długości n

i maszynę zliczającą maksymalną pamięć $S(n)$. Należy po prostu uruchomiać maszynę $\mathcal{M}$ kolejno dla wszystkich słów o danej długości n. □

Klasy złożoności

Języki obliczalne klasyfikujemy ze względu na ich złożoność czasową i pamięciową. W tym celu określamy *klasy złożoności*:

$\textsc{Dtime}(T(n)) = \{L \mid L$ jest rozpoznawany przez deterministyczną maszynę Turinga o złożoności czasowej $T(n)\}$;

$\textsc{Ntime}(T(n)) = \{L \mid L$ jest rozpoznawany przez niedeterministyczną maszynę Turinga o złożoności czasowej $T(n)\}$;

$\textsc{Dspace}(S(n)) = \{L \mid L$ jest rozpoznawany przez deterministyczną maszynę Turinga o złożoności pamięciowej $S(n)\}$;

$\textsc{Nspace}(S(n)) = \{L \mid L$ jest rozpoznawany przez niedeterministyczną maszynę Turinga o złożoności pamięciowej $S(n)\}$;

Zazwyczaj interesuje nas złożoność mierzona z dokładnością do czynnika stałego. Używamy wtedy notacji $\mathcal{O}$. Dla funkcji $F, G : \mathbb{N} \to \mathbb{N}$ stwierdzenie *„funkcja F jest $\mathcal{O}(G)$"* oznacza, że istnieje taka stała $c > 0$, że $F(n) \leq c \cdot G(n)$, dla każdego n. Na przykład funkcja $3n^2 + 4n$ jest $\mathcal{O}(n^2)$, ale nie jest $\mathcal{O}(1000n \log n)$.

Używając notacji $\mathcal{O}$ można też napisać np. $\textsc{Dspace}(\mathcal{O}(n^2))$ na oznaczenie sumy wszystkich klas postaci $\textsc{Dspace}(c \cdot n^2)$.

Przykład 8.2 Zbiór wszystkich palindromów należy do klasy $\textsc{Dspace}(3(\lceil \log n \rceil))$ i do klasy $\textsc{Dtime}(3n)$.[17] Można bowiem skonstruować dwie maszyny akceptujące ten język. Pierwsza kopiuje wejście na taśmę roboczą i porównuje słowo wejściowe z jego kopią czytaną wstecz. Ta maszyna ma złożoność czasową $3n$ i złożoność pamięciową $\mathcal{O}(n)$.

Druga maszyna porównuje kolejno pierwszy symbol słowa wejściowego z ostatnim, drugi z przedostatnim, i tak dalej. Pozycje symboli, które należy porównać, maszyna pamięta przy pomocy dwóch liczników, reprezentowanych na taśmie roboczej jako słowa binarne długości $\lceil \log n \rceil$. (Na zapamiętanie pozycji na taśmie długości n wystarczy $\lceil \log n \rceil$ bitów.) Ta maszyna ma złożoność pamięciową $3(\lceil \log n \rceil)$. Ale jej złożoność czasowa jest $\mathcal{O}(n^2)$, bo taśma wejściowa jest czytana n razy. □

When analyzing the complexity of a problem we are usually concerned with "large" inputs and we disregard anomalies occurring only for "small" inputs (i.e., for only finitely many of them).

Fakt 8.3 *If $S_1(n) = S_2(n)$ almost everywhere, then* $\textsc{Dspace}(S_1(n)) = \textsc{Dspace}(S_2(n))$*, and similarly for* Nspace, Dtime *and* Ntime.

[17] Czyli do $\textsc{Dspace}(\mathcal{O}(\log n))$ i do $\textsc{Dtime}(\mathcal{O}(n))$.

Dowód: Przypuśćmy, że $S_1(n) = S_2(n)$, dla wszystkich $n > n_0$ i niech $L = L(\mathcal{M}_1)$, gdzie $\mathcal{M}_1$ jest maszyną o złożoności pamięciowej $S_1(n)$. Chcemy skonstruować maszynę $\mathcal{M}_2$, która symuluje $\mathcal{M}_1$ w pamięci $S_2(n)$. Oczywista idea jest taka: dla słów długości większej niż n_0 maszyna $\mathcal{M}_2$ powinna robić to samo co maszyna $\mathcal{M}_1$, a jeśli słowo wejściowe jest za krótkie, to pamięć robocza jest w ogóle niepotrzebna, bo do rozpoznawania skończonego zbioru słów wystarczy automat skończony.

A zatem maszyna $\mathcal{M}_2$ na początku działa jak automat skończony: czytając słowo wejściowe od lewej zapamiętuje w stanach wewnętrznych pierwsze n_0 jego liter. Jeśli słowo jest „krótkie", to maszyna akceptuje lub odrzuca na podstawie stanu, a w przeciwnym razie rozpoczyna symulację maszyny $\mathcal{M}_1$.

Dla złożoności czasowej trzeba tę konstrukcję nieco poprawić, bo na razie maszyna $\mathcal{M}_2$ w przypadku „długiego" wejścia wykonuje za dużo o n_0 kroków. Poprawka jest taka: w fazie początkowej maszyna $\mathcal{M}_2$ nie tylko zapamiętuje litery z wejścia ale także symuluje obliczenie maszyny $\mathcal{M}_1$, z tym, że domniemane położenie głowicy wejściowej $\mathcal{M}_1$ jest również zapamiętane przez stan wewnętrzny. Jeśli pierwsze n_0 kroków nie doprowadziło do rozstrzygnięcia, maszyna $\mathcal{M}_2$ kontynuuje symulację bez straty czasu. Głowica wejściowa $\mathcal{M}_2$ pozostaje wtedy w pozycji n_0 aż do momentu, w którym głowica $\mathcal{M}_1$ powinna do tego miejsca dotrzeć.[18] □

Złożoność obliczeniową rozważamy zwykle z dokładnością do stałej:

Fakt 8.4 (Kompresja taśmy) *Dla dowolnej funkcji S i dowolnej stałej $c > 0$:*

$$\textsc{Dspace}(S(n)) = \textsc{Dspace}(c \cdot S(n));$$
$$\textsc{Nspace}(S(n)) = \textsc{Nspace}(c \cdot S(n)).$$

Dowód: Niech $c > 1$ (dla $c < 1$ zamiast c i $S(n)$ rozważamy $\frac{1}{c}$ i $c \cdot S(n)$) i niech $L = L(\mathcal{M})$ dla pewnej deterministycznej maszyny $\mathcal{M}$ o złożoności pamięciowej $c \cdot S(n)$. Niech $k = \lceil 2c \rceil$. Konstruujemy maszynę $\mathcal{M}'$, symulującą $\mathcal{M}$, ale używającą alfabetu Σ^k. To znaczy, że jeden „nowy" symbol odpowiada k „starym". Na przykład zamiast takiej taśmy maszyny $\mathcal{M}$

|	*b*	|	*c*	|	*a*	|	*b*	|	*c*	|	*b*	|	*c*	|	*a*	|	*b*	|	*b*	|	*c*	|	*b*	|	|
					↑																						

maszyna $\mathcal{M}'$ będzie (przy $k = 4$) używać taśmy 4 razy krótszej.

|	*bcab*	|	*cbca*	|	*bbcb*	|	
	↑						

Oczywiście maszyna $\mathcal{M}'$ zapamiętuje pozycję głowicy wewnątrz bieżącego k-literowego segmentu w swoim stanie wewnętrznym. Ma więc nie tylko większy alfabet, ale także więcej stanów. Ale taśma używana przez $\mathcal{M}$' ma długość $\lceil \frac{c \cdot S(n)}{k} \rceil \leq S(n)$.

Przypadek niedeterministyczny jest podobny. □

Fakt 8.5 (Przyspieszanie liniowe) *Jeśli $T(n) \geq n(1+\varepsilon)$ dla pewnego $\varepsilon > 0$, to dla dowolnej stałej $c > 1$ zachodzą równości:*

[18] Tu przypomnijmy nasze założenie, że maszyna zawsze czyta całe słowo.

$$\textsc{Dtime}(T(n)) = \textsc{Dtime}(c \cdot T(n));$$
$$\textsc{Ntime}(T(n)) = \textsc{Ntime}(c \cdot T(n)).$$

Dowód: Let $c > 1$ and let $L = L(\mathcal{M})$, for some deterministic machine $\mathcal{M}$ of time complexity $c \cdot T(n)$. We construct a machine $\mathcal{M}'$ that simulates $\mathcal{M}$ and works c times faster. As in the proof of Fakt 8.4, tape segments of length k (for a certain k) are represented as single symbols of $\mathcal{M}$. This time however, the contents of the input tape is also represented this way on an additional work tape (this requires at most $n + \lceil \frac{n}{k} \rceil$ machine steps at the beginning of the computation). This additional work tape is then used instead of the input.

The machine $\mathcal{M}'$ proceeds in phases of 8 steps representing at least k steps of $\mathcal{M}$. At the beginning of each phase, every machine head is scanning a certain k-tuple of symbols:

|	*bcab* |	*cbca* |	*bbcb* |
		↑	

The machine now moves on every tape one step to left, and two steps to the right, and again one step to left, returning to the initial position. But now it has seen both the adjacent tape cells (on every tape) and adjusted its state to "remember" the contents of these cells. On the basis of this information, the machine can determine the future behaviour of $\mathcal{M}$, until one of the heads leaves the $3k$ cells under control, or the computation halts. In the latter case, $\mathcal{M}'$ accepts or rejects depending on whether $\mathcal{M}$ accepts or rejects. In the former case, the next four steps of $\mathcal{M}'$ are used to adjust the contents of the three cells to represent the configuration of $\mathcal{M}$ at the moment when one of the tape heads leaves the $3k$ cells of $\mathcal{M}'$:

|	*bcab* |	*cbcb* |	*aaba* |	|
				↑

This happens after at least k steps of $\mathcal{M}$, and our simulation used only 8 steps. Thus the machine $\mathcal{M}'$ is of time complexity

$$T_1(n) = n + \lceil \tfrac{n}{k} \rceil + 8 \cdot \lceil \tfrac{c}{k} \cdot T(n) \rceil \leq n(1 + \tfrac{1}{k}) + \tfrac{8c}{k} \cdot T(n) + 9.$$

Ponieważ $T(n) \geq n(1 + \varepsilon)$, więc $n \leq \frac{1}{1+\varepsilon} T(n)$, a zatem dla prawie wszystkich n zachodzi

$$T_1(n) \leq \frac{1 + \frac{1}{k}}{1 + \varepsilon} \cdot T(n) + \frac{8c}{k} \cdot T(n) + 9 = T(n) \cdot \left(\frac{1}{1 + \varepsilon} + \frac{1}{k} \left(\frac{1}{1 + \varepsilon} + 8c \right) + \frac{9}{T(n)} \right) \leq T(n).$$

Jest tak dlatego, że $\frac{1}{1+\varepsilon} < 1$, a pozostałe dwa składniki sumy w nawiasie mogą być dowolnie małe przy dostatecznie dużych k i n. Konkluzja wynika z faktu 8.3. □

Twierdzenie 8.6 (o hierarchii)

(1) Dla dowolnej obliczalnej funkcji $T(n)$ istnieje taka funkcja obliczalna $T_1(n)$, że:

- *(a)* $\textsc{Dtime}(T(n)) \subsetneq \textsc{Dtime}(T_1(n))$ *oraz*
- *(b)* $\textsc{Ntime}(T(n)) \subsetneq \textsc{Ntime}(T_1(n))$.

(2) Dla dowolnej obliczalnej funkcji $S(n)$ istnieje taka funkcja obliczalna $S_1(n)$, że:

- *(a)* $\textsc{Dspace}(S(n)) \subsetneq \textsc{Dspace}(S_1(n))$ *oraz*
- *(b)* $\textsc{Nspace}(S(n)) \subsetneq \textsc{Nspace}(S_1(n))$.

Dowód: (1a) Niech $\mathcal{M}_w$ oznacza deterministyczną maszynę o kodzie w. Rozpatrzmy język

$$L = \{w \mid \mathcal{M}_w \text{ nie akceptuje w } T(|w|) \text{ krokach słowa } w\}.$$

Pokażemy, że $L \notin \textsc{Dtime}(T(n))$. Istotnie, przypuśćmy, że $L = L(\mathcal{M}_x)$ dla pewnej maszyny $\mathcal{M}_x$ o złożoności czasowej $T(n)$, i zapytajmy, czy kod x maszyny $\mathcal{M}_x$ należy do $L(\mathcal{M}_x)$. Jeśli tak, to zgodnie z definicją L, maszyna $\mathcal{M}_x$ nie akceptuje x w czasie $T(|x|)$. Ale wtedy $\mathcal{M}_x$ w ogóle nie akceptuje słowa x, bo jest to maszyna o złożoności $T(n)$. A więc $x \notin L(\mathcal{M}_x)$, sprzeczność. Z drugiej strony, założenie $x \notin L$ w podobny sposób implikuje $x \in L$.

Ale język L jest obliczalny i na mocy faktu 8.1 musi należeć do jakiejś klasy $\textsc{Dtime}(T'(n))$. Zatem $\textsc{Dtime}(T(n)) \subsetneq \textsc{Dtime}(T_1(n))$, gdzie $T_1(n) = \max\{T(n), T'(n)\}$.

W pozostałych przypadkach stosujemy podobne rozumowanie przekątniowe. □

Powyższy rezultat można znacznie uściślić, przy pewnych dodatkowych założeniach. Powiemy, że niemalejąca funkcja $F(n)$ jest *konstruowalna* (ang. *proper* lub *constructible*), gdy istnieje deterministyczna maszyna Turinga $\mathcal{M}$, która

- ma złożoność pamięciową $\mathcal{O}(F(n))$ i złożoność czasową $\mathcal{O}(n + F(n))$;
- dla dowolnego n i dowolnego słowa długości n wypisuje na ustalonej taśmie roboczej dokładnie $F(n)$ symboli.

That is, a proper function is one that can be computed within time and space proportional to its own value. (An exception is made for functions $F(n) < n$, in which case the time bound is linear.) Większość funkcji, które są rozważane w teorii złożoności jest konstruowalna (np. n^2, n^3, 2^n, $\lceil\sqrt{n}\rceil$, $\lfloor\log n\rfloor$, itp.[19]).

W poniższym twierdzeniu występuje notacja o małe: powiemy „G jest $o(F)$" wtedy, kiedy funkcja G jest asymptotycznie mała w porównaniu z F, tj. $\lim_{n\to\infty} \frac{G(n)}{F(n)} = 0$.

Twierdzenie 8.7 *Niech $F(n), G(n)$ będą konstruowalne i takie, że $F(n) \geq G(n)$.*

1. *Jeśli $G(n)$ jest $o(F(n))$, to $\textsc{Dspace}(G(n)) \subsetneq \textsc{Dspace}(F(n))$.*
2. *Jeśli $G(n)\log G(n)$ jest $o(F(n))$, to $\textsc{Dtime}(G(n)) \subsetneq \textsc{Dtime}(F(n))$.*

Dowód: ▶ Pokażemy część pierwszą, stosując metodę przekątniową. Bez straty ogólności możemy założyć, że jeśli w jest kodem maszyny $\mathcal{M}$, to każde słowo postaci $\$\$\ldots\$w$ też jest kodem tej maszyny. W szczególności każda maszyna ma nieskończenie wiele kodów.

Określimy deterministyczną maszynę $\mathcal{M}$, działającą w pamięci $\mathcal{O}(F(n))$. Maszyna $\mathcal{M}$ na wejściu w długości n symuluje działanie maszyny $\mathcal{M}_w$ na wejściu w, używając w tym celu ustalonego obszaru pamięci rozmiaru $F(n)$. Ponieważ maszyna $\mathcal{M}_w$ może mieć dowolnie wiele taśm, przy konstrukcji maszyny $\mathcal{M}$ trzeba zastosować symulację jak w dowodzie faktu 6.2. Jeśli symulacja jest udana, oraz $\mathcal{M}_w$ odrzuca w, to $\mathcal{M}$ akceptuje. W pozostałych przypadkach (udana symulacja akceptująca lub symulacja nieudana z powodu wyczerpania pamięci) maszyna $\mathcal{M}$ odrzuca w.

Uwaga: pamięć $F(n)$ maszyny $\mathcal{M}$ nie zawsze pozwala na poprawną symulację maszyn działających w pamięci $F(n)$, bo mogą one mieć np. obszerny alfabet i dużo taśm. Tymczasem alfabet $\mathcal{M}$ jest skończony i konieczne jest kodowanie pojedynczych symboli za pomocą słów. Inny problem pojawia się dla $F(n) < \log n$: może wtedy zabraknąć miejsca na zapisanie położenia głowicy wejściowej maszyny $\mathcal{M}_w$.

[19] Symbol log oznacza oczywiście logarytm dwójkowy.

Przypuśćmy, że język $L = L(\mathcal{M})$ należy do klasy $\textsc{Dspace}(G(n))$, tj. że mamy $L = L(\mathcal{M}')$ dla pewnej maszyny $\mathcal{M}'$ działającej w pamięci G. Załóżmy, że $\mathcal{M}' = \mathcal{M}_v$. Jeśli $w = \$\$\ldots\$v$, to wtedy także $\mathcal{M}' = \mathcal{M}_w$. Ponieważ liczba taśm i rozmiar alfabetu maszyny $\mathcal{M}'$ są ustalone, więc symulacja maszyny $\mathcal{M}' = \mathcal{M}_w$ na wejściu w jest możliwa w pamięci $c \cdot G(n)$ dla pewnej stałej c. Ponadto, jeśli słowo w jest dostatecznie długie (ma na początku dostatecznie dużo dolarów), to mamy $c \cdot G(n) < F(n)$, gdzie $|w| = n$, a to oznacza, że:

$$\mathcal{M} \text{ akceptuje } w \qquad \text{wtedy i tylko wtedy, gdy} \qquad \mathcal{M}' \text{ odrzuca } w.$$

A zatem $L \neq L(\mathcal{M}')$. Uwaga: jeśli $F(n) < \log n$, to symulacja jest też poprawna, bo można zakładać, że $w = \$\$\ldots\$v$, gdzie v jest bardzo krótkie w porównaniu z w. Dlatego informację o położeniu głowicy wejściowej można przechowywać w pamięci $\log |v| \leq F(n)$.

Dowód części drugiej jest podobny, ale trochę trudniejszy. Czynnik $\log n$ to „narzut" czasowy za symulację wielu taśm maszyny $\mathcal{M}_w$ na dwóch taśmach maszyny $\mathcal{M}$. ◀ □

Istnieją też twierdzenia o ostrych hierarchiach niedeterministycznych:

Twierdzenie 8.8 *Niech $F(n), G(n)$ będą konstruowalne i takie, że $F(n) \geq G(n) \geq \log n$.*

1. *Jeśli $G(n)$ jest $o(F(n))$, to $\textsc{Nspace}(G(n)) \subsetneq \textsc{Nspace}(F(n))$.*
2. *Jeśli $G(n+1)$ jest $o(F(n))$, to $\textsc{Ntime}(G(n)) \subsetneq \textsc{Ntime}(F(n))$.*

► Założenie o konstruowalności w twierdzeniach 8.7 i 8.8 jest istotne. Funkcje, które nie są konstruowalne, mogą zachowywać się w sposób nieprzewidywalny. Na przykład mamy następujące twierdzenie Trachtenbrota o luce:

Twierdzenie 8.9 *Istnieje taka obliczalna funkcja T, że $\textsc{Dtime}(T(n)) = \textsc{Dtime}(2^{T(n)})$.*

Dowód: Niech $\mathcal{M}_v$ oznacza deterministyczną maszynę Turinga o kodzie v. Przez $t(v, w)$ oznaczmy liczbę kroków obliczenia maszyny M_v dla wejścia w (możliwe, że $t(v, w) = \infty$). Symbolem P oznaczymy zaś dwuargumentowy predykat w $\mathbb{N}$, zdefiniowany tak:

$$P(n, k) \equiv \forall i \leq n \forall w, v(|v| \leq n \wedge |w| = n \to t(i, w) \leq k \vee (v, w) > 2^k).$$

A zatem $P(n, k)$ zachodzi wtedy i tylko wtedy, gdy dla dowolnego $i = 1, \ldots, n$ i dowolnego słowa wejściowego w długości n, liczba kroków obliczenia jakie wykonuje na wejściu w każda maszyna o kodzie nie dłuższym niż n jest albo „mała" (co najwyżej k) albo „duża" (wieksza od 2^k), ale nigdy nie „średnia". Zauważmy, że do każdego n można dobrać k o własności $P(n, k)$. Istotnie, wystarczy przyjąć

$$k = \max\{t(v, w) \mid |w| = n \wedge |v| \leq n \wedge t(v, w) \neq \infty\},$$

czyli maksimum z długości skończonych obliczeń maszyn o kodach nie dłuższych niż n na słowach długości n. (Jest takich obliczeń skończenie wiele.)

Niech teraz $T(n)$ będzie najmniejszą liczbą k o własności $P(n, k)$ (piszemy $T(n) = \mu k.P(n, k)$). Wtedy funkcja T jest całkowitą funkcją obliczalną. Przypuśćmy teraz, że $L \in \textsc{Dtime}(2^{T(n)})$, tj. $L = L(M)$ dla pewnej maszyny M o złożoności czasowej $2^{T(n)}$. Maszyna M ma jakiś kod; powiedzmy, że $M = M_v$.

Dla $n > |v|$ (a więc dla prawie wszystkich n), z warunku $P(n, T(n))$ wynika, że M_v może akceptować słowa długości n w czasie poniżej $T(n)$ lub powyżej $2^{T(n)}$. Ale to drugie jest niemożliwe, bośmy założyli, że M_v działa w czasie $2^{T(n)}$. Zatem M_v ma w istocie złożoność $T(n)$. □ ◀

Relacje pomiędzy klasami

In this section we consider the known inclusions between space and time complexity classes. It is convenient to state first the following lemma. A configuration of a Turing machine is *of size* m iff it uses at most m cells on each work tape.

Lemat 8.10 *Assume that* $S(n) \geq \log n$ *for all* n*. Let* $\mathcal{M}$ *be a Turing machine, and let* $\mathbf{C}_w$ *be the set of all configurations of* $\mathcal{M}$ *of size* $S(|w|)$ *with* w *as the input word. There exists a constant* c *such that, for all* w*, the set* $\mathbf{C}_w$ *has at most* $2^{c \cdot S(|w|)}$ *elements.*

Dowód: Assume that $\mathcal{M}$ has k internal states, r work tapes and an m-element alphabet. Let $|w| = n$. To count the configurations of size $S(n)$ we multiply the following:

- the number of states k;
- the number of possible positions of work heads ($S(n)$ for each);
- the number $n + 2$ of possible positions of the input head;
- the number of all possible contents of the work tapes ($m^{S(n)}$ for each).

We obtain $k \cdot S(n)^r \cdot (n+2) \cdot (m^{S(n)})^r$. Clearly,

$$k \cdot S(n)^r \cdot (n+2) \cdot (m^{S(n)})^r \leq 2^{\log k + r \log S(n) + 1 + \log n + rS(n) \log m} \leq 2^{c \cdot S(n)},$$

holds for a sufficiently large constant c and almost all n. □

Zauważmy, że do zapisania liczby $2^{c \cdot S(n)}$ wystarczy pamięć $\mathcal{O}(S(n))$. Jeśli funkcja S jest konstruowalna, to można oznaczyć taki obszar pamięci i użyć go jako binarnego licznika czasu. Stąd wynika taki fakt:

Fakt 8.11 *Jeśli* $L \in \textsc{Dspace}(S(n))$ (*odp.* $L \in \textsc{Nspace}(S(n))$) *gdzie* $S \geq \log$ *jest funkcją konstruowalną, to* $L = L(\mathcal{M})$ *dla pewnej deterministycznej* (*odp. niedeterministycznej*) *maszyny* $\mathcal{M}$*, której każde obliczenie jest skończone i która ma złożoność pamięciową* $S(n)$*.*

Następujące twierdzenie wyraża podstawowe związki między klasami złożoności.

Twierdzenie 8.12

(1) $\textsc{Dtime}(T(n)) \subseteq \textsc{Ntime}(T(n))$*;*

(2) $\textsc{Ntime}(T(n)) \subseteq \textsc{Dspace}(T(n))$*;*

(3) $\textsc{Dspace}(S(n)) \subseteq \textsc{Nspace}(S(n))$*.*

Jeśli funkcja $S(n)$ *jest konstruowalna oraz* $S(n) \geq \log n$*, to także*

(4) $\textsc{Dspace}(S(n)) \subseteq \bigcup_{c>0} \textsc{Dtime}(2^{c \cdot S(n)})$*;*

(5) $\textsc{Nspace}(S(n)) \subseteq \bigcup_{c>0} \textsc{Dtime}(2^{c \cdot S(n)})$*.*

Dowód: Zawierania (1) i (3) są oczywiste. Inkluzja (2) wynika stąd, że dana maszyna niedeterministyczna $\mathcal{M}$ może zostać zastąpiona maszyną deterministyczną przeglądającą wszystkie możliwe obliczenia maszyny $\mathcal{M}$. Podobnie jak w dowodzie faktu 6.3, maszyna deterministyczna symuluje działanie maszyny $\mathcal{M}$, przeszukując wszerz jej drzewo obliczeń. W tym celu,

na dodatkowej taśmie roboczej systematycznie generuje kolejne ciągi liczb $m_1, m_2, \ldots, m_k$, odpowiadające początkowym fragmentom obliczeń maszyny $\mathcal{M}$.

Do generowania kolejnych ciągów $m_1, m_2, \ldots, m_k$ potrzebna jest pamięć rozmiaru $T(n)$. Wykonanie obliczenia odpowiadajacego danemu ciągowi też wymaga pamięci rozmiaru $T(n)$, bo $T(n)$ jest także złożonością pamięciową $\mathcal{M}$ (w czasie $T(n)$ można użyć co najwyżej $T(n)$ klatek taśmy). Zatem całkowita potrzebna pamięć jest $\mathcal{O}(T(n))$.

Część (4) wynika z lematu 8.10 i faktu 8.11. Maszyna deterministyczna $\mathcal{M}$ o złożoności pamięciowej $S(n)$ osiąga co najwyżej $2^{c\cdot S(n)}$ konfiguracji dla wejścia rozmiaru n. Na mocy faktu 8.11 można założyć, że maszyna $\mathcal{M}$ zawsze się zatrzymuje. Zatem obliczenia maszyny mogą mieć długość co najwyżej $2^{c\cdot S(n)}$. Po takiej liczbie kroków maszyna musi się zatrzymać, bo inaczej powtórzy się jakaś konfiguracja.

Zajmiemy się teraz częścią (5). Nie możemy użyć metody zastosowanej w części (2), bo otrzymamy czas podwójnie wykładniczy od $S(n)$.

Załóżmy, że mamy niedeterministyczną maszynę $\mathcal{M}$ o złożoności pamięciowej $S(n)$. Niech $\mathbf{C}_w$ oznacza zbiór konfiguracji jak w lemacie 8.10. Ma on co najwyżej $f(n) = 2^{c\cdot S(n)}$ elementów.

Rozpatrzmy teraz algorytm podobny do użytego w dowodzie faktu 8.1. Aby utworzyć listę wszystkich konfiguracji osiągalnych z $\mathcal{C}_w$ iterujemy procedurę:

> *Dla każdej dotychczas nie zapisanej konfiguracji* $\mathcal{C} \in \mathbf{C}_w$
> *szukaj takiego* $\mathcal{C}'$ *na liście, że* $\mathcal{C}' \to_{\mathcal{M}} \mathcal{C}$.
> *Jeśli jest, to dodaj* $\mathcal{C}$ *do listy.*

Postępujemy tak, aż dodamy do listy konfigurację końcową, albo wyczerpiemy wszystkie możliwości (stwierdzimy, że nic nowego nie da się już dopisać).

Na liście nie może być więcej niż $f(n)$ konfiguracji, więc powyższa pętla wykonywana jest co najwyżej tyle razy. Za każdym razem rozważamy $f(n)$ konfiguracji i dla każdej z nich wykonujemy $\mathcal{O}(f(n))$ kroków. Cały algorytm wymaga więc liczby kroków rzędu $\mathcal{O}(f(n)^3)$, a to można oszacować przez $2^{c_1\cdot S(n)}$ dla pewnej stałej c_1. □

O związkach pomiędzy klasami złożoności wciąż niewiele wiadomo. Poniższe twierdzenie należy do nielicznych wyników „pozytywnych" w tej dziedzinie.

Twierdzenie 8.13 (Savitch, 1970) *Jeśli funkcja* $S(n)$ *jest konstruowalna, oraz* $S(n) \geq \log n$, *to* $\textsc{Nspace}(S(n)) \subseteq \textsc{Dspace}(S(n)^2)$.

Dowód: Niech $\mathcal{M}$ będzie niedeterministyczną maszyną o złożoności pamięciowej $S(n)$. Na podstawie lematu 8.10 możemy przyjąć bez straty ogólności, że dla pewnego c ma ona złożoność czasową $2^{c\cdot S(n)}$. Niech $|w| = n$. Dla $\mathcal{C}_1, \mathcal{C}_2 \in \mathbf{C}_w$ (gdzie $\mathbf{C}_w$ jest jak w lemacie 8.10), oraz $i \leq c \cdot S(n)$, piszemy $\mathcal{C}_1 \to^i \mathcal{C}_2$, gdy maszyna $\mathcal{M}$ może przejść od konfiguracji $\mathcal{C}_1$ do $\mathcal{C}_2$ w co najwyżej 2^i krokach. Jeśli $i = 0$, to warunek $\mathcal{C}_1 \to^i \mathcal{C}_2$ można sprawdzić w pamięci $S(n)$. Dla $i > 1$, wystarczy zbadać, czy dla jakiegoś $\mathcal{C} \in \mathbf{C}_w$ zachodzi $\mathcal{C}_1 \to^{i-1} \mathcal{C} \to^{i-1} \mathcal{C}_2$. Jeśli więc przez $\varphi(i)$ oznaczymy pamięć potrzebną na deterministyczne sprawdzanie czy $\mathcal{C}_1 \to^i \mathcal{C}_2$, to dostaniemy oszacowanie

$$\varphi(i) \leq S(n) + \varphi(i-1),$$

bo oczywiście pamięć użyta do sprawdzenia warunku $\mathcal{C}_1 \to^{i-1} \mathcal{C}$ może być ponownie użyta do sprawdzenia warunku $\mathcal{C} \to^{i-1} \mathcal{C}_2$. (Składnik $S(n)$ jest potrzebny do zapisania konfiguracji $\mathcal{C}$.)

Ostatecznie dostajemy $\varphi(i) \leq i \cdot S(n)$, w szczególności $\varphi(c \cdot S(n)) \leq c \cdot S(n)^2$.

A zatem w pamięci $\mathcal{O}(S(n)^2)$ można deterministycznie sprawdzić, czy z konfiguracji początkowej da się otrzymać konfigurację końcową. A nic więcej nam nie potrzeba. □

► A oto przykład wyniku „negatywnego": jedno z nielicznych znanych twierdzeń dotyczących roli niedeterminizmu w złożoności czasowej. Dowód z ulgą pomijamy.

Twierdzenie 8.14 (Paul, Pippenger, Szemeredi, Trotter, 1983)

$$\textsc{Ntime}(\mathcal{O}(n)) \neq \textsc{Dtime}(\mathcal{O}(n)).$$ ◄

9 Złożoność hierarchii Chomsky'ego

A finite automaton, accepting a regular language, can be seen as a very restricted kind of a Turing machine. This machine is "one-way", that is, it always moves its input head to the right, and it never uses any work space. Thus, regular languages belong to Dspace(1). It may appear surprising that the converse is also true.

Twierdzenie 9.1 *The class of regular languages coincides with* Dspace(1) *and* Nspace(1).

Dowód: Let $\mathcal{M}$ be a nondeterministic machine of space complexity 1. The difference between $\mathcal{M}$ and an NFA is that $\mathcal{M}$ can move its head freely in each direction. Without loss of generality we can assume however, that:

- The machine can enter an acccepting state only after its input head has been moved to the rightmost position.
- The machine always moves either left or right, except when entering an accepting state.

An accepting computation for $w = abcdab$ can thus be represented as in Rysunek 4.

Consider the history of the computation from the point of view of a single tape cell. Write down this history, recording states and moves. For instance, for the first "b" this history can be recorded as

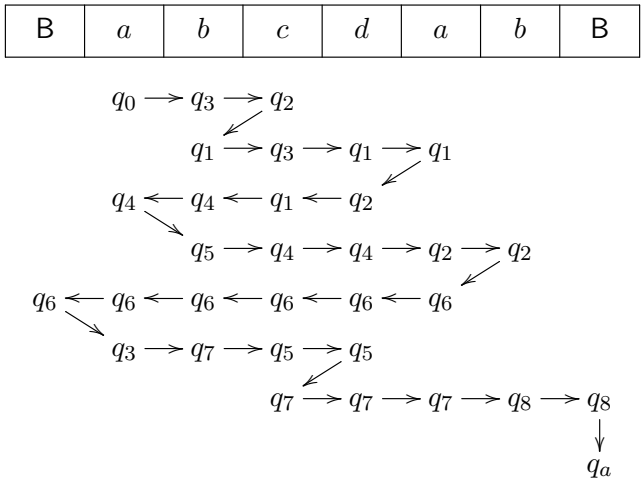
$$q_3^{\rightarrow}\ q_1^{<}\ q_4^{\leftarrow}\ q_5^{\rightarrow}\ q_6^{\leftarrow}\ q_7^{\rightarrow}$$

where $q_3^{\rightarrow}$ means that the machine entered this cell from the left in state q_3 and then went to the right. The symbol $q_1^{<}$ means that the machine entered this cell from the right and then returned to the right. Similarly, the history of "d" is as follows:

$$q_1^{\rightarrow}\ q_2^{\leftarrow}\ q_4^{\rightarrow}\ q_6^{\leftarrow}\ q_5^{>}\ q_7^{\rightarrow}$$

If there is a computation accepting a word w then there is one that never enters the same state twice while scanning the same tape cell. Indeed, we can simply skip the part of the computation between these two identical configurations. This means that the number of our

Rysunek 4:



"histories" (called also *crossing sequences*) can be assumed constant. (Tj. ta liczba zależy tylko od liczby stanów automatu, a nie od długości słowa.)

Suppose now that a certain history is associated to a tape cell containing a symbol a. The history associated to the next tape cell must be such that the two histories together create a consistent "two cell history". For instance, suppose that the transition relation of $\mathcal{M}$ contains the following tuples:

$$\langle q_3, a, q_2, +1\rangle, \langle q_2, b, q, +1\rangle, \langle q_3, b, q_2, -1\rangle, \langle q_2, a, q,' -1\rangle, \langle q_2, a, q_4, +1\rangle,$$
$$\langle q_4, b, q_1, -1\rangle, \langle q_1, a, q'', -1\rangle, \langle q_5, a, q_5, +1\rangle, \langle q_5, b, q''', +1\rangle, \langle q_1, b, q'''', +1\rangle.$$

Then the history

$$q_3^{\rightarrow}\ q_2^{\leftarrow}\ q_2^{\rightarrow}\ q_1^{\leftarrow}\ q_1^{>}\ q_5^{\rightarrow}$$

can be placed to the left of

$$q_2^{\rightarrow}\ q_3^{\leftarrow}\ q_4^{>}\ q_5^{\rightarrow}\ q_1^{<}$$

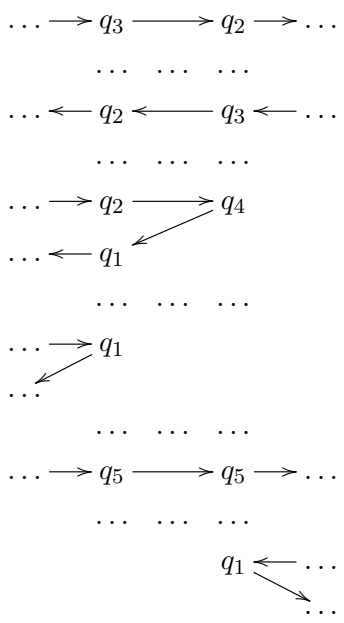
to represent a local behaviour of the machine, as shown in Rysunek 5.

The history $q_3^{\rightarrow}\ q_2^{\leftarrow}\ q_2^{\rightarrow}\ q_1^{\leftarrow}\ q_1^{>}\ q_5^{\rightarrow}$ can perhaps be followed by $q_1^{>}\ q_1^{\rightarrow}\ q_4^{\leftarrow}\ q_1^{\rightarrow}$ as well (provided the machine is capable of making the appropriate state changes), but not by

$$q_2^{\leftarrow}\ q_3^{\leftarrow}\ q_4^{>}\ q_5^{\rightarrow}\ q_1^{<} \quad \text{nor} \quad q_3^{>}\ q_3^{>}\ q_5^{<}.$$

We construct a nondeterministic finite automaton $\mathcal{M}_f$ to recognize $L(\mathcal{M})$. The automaton attempts to reconstruct the computation of $\mathcal{M}$ by assigning a crossing sequence (a history) to every symbol written on the input tape. W każdym kolejnym kroku stan wewnętrzny automatu przedstawia ciąg przecięć (crossing sequence) związany z ostatnio przeczytaną literą. For every further symbol read from the input, the automaton guesses a new crossing sequence, and verifies the correctness of the guess against the previous crossing sequence and the transition relation of $\mathcal{M}$. Relacja przejścia automatu jest tak zdefiniowana, że przejście od stanu do stanu jest możliwe wtedy i tylko wtedy, gdy odpowiednie ciągi przecięć mogą ze sobą poprawnie sąsiadować. Ostatecznie, poprawne obliczenie maszyny $\mathcal{M}$ istnieje wtedy i tylko wtedy, gdy

Rysunek 5:



istnieje „dobrze składalny" ciąg ciągów przecięć. A to zachodzi wtedy i tylko wtedy, gdy $\mathcal{M}_f$ akceptuje poprzez osiągnięcie stanu końcowego (tj. ciągu przecięć, w którym pojawia się stan końcowy automatu M. □

Języki bezkontekstowe

The above result gives an exact characterization of complexity of regular languages. For context-free languages, we do not have an exact characterization. We can only state an upper bound: all context-free languages can be recognized in polynomial time.

Twierdzenie 9.2 *All context-free languages belong to the class* $\mathbf{P} = \bigcup_k \textsc{Dtime}(n^k)$.

Dowód: Let $L = L(G)$ be a context-free language. Without loss of generality we can assume that $\varepsilon \notin L$ (exercise), and that G is in Chomsky normal form. That is, all productions of G are of type:

$$\xi \Rightarrow \xi_1\xi_2 \quad \text{or} \quad \xi \Rightarrow a,$$

where ξ, ξ_1, ξ_2 are nonterminals and a is a terminal. In order to check if $w \in L$ one can apply the following method:

- If $|w| = 1$ then check if G contains a production $\xi_0 \Rightarrow w$, where ξ_0 is the initial symbol.
- Otherwise, for each production of the form $\xi_0 \Rightarrow \xi_1\xi_2$ check if w can be split into two parts $w_1w_2 = w$, such that $\xi_1 \twoheadrightarrow_G w_1$ and $\xi_2 \twoheadrightarrow_G w_2$.

A naive implementation of the above idea would result in a recursive algorithm of exponential

complexity. Better results can be obtained if we collect and store reusable information.[20]

Assume that the input word is of the form $w = a_1 a_2 \dots a_n$. For $i \leq j$, let $w_{ij} = a_i \dots a_j$ and $V_{ij} = \{\xi \mid \xi \twoheadrightarrow_G w_{ij}\}$. Our algorithm will construct and store all the sets V_{ij}, beginning with V_{ii} and ending with V_{1n}. Then we simply check if $\xi_0 \in V_{1n}$. The sets V_{ij} can be written down on a work tape of size $\mathcal{O}(n^2)$.

The sets V_{ii} consist of all the nonterminals ξ such that the production $\xi \Rightarrow a_i$ is in G. The construction of V_{ii} requires linear time.

Now suppose that, for some $d = 1, \dots, n-1$, we have already constructed all sets V_{ij}, for $j - i < d$, and we want to construct V_{ij} for $j - i = d$. For each such pair i, j and for each $\ell = i, \dots, j-1$, we consider all productions of the form $\xi \Rightarrow \xi_1 \xi_2$. For every such production we check if $\xi_1 \in V_{i\ell}$ and $\xi_2 \in V_{\ell+1,j}$. If so, we add ξ to the set V_{ij}.

For each pair i, j (there is less than n such pairs, for any fixed d) we have $d < n$ possible values of ℓ. For each ℓ, we must check all sets $V_{i\ell}$ and $V_{\ell+1,j}$ (on the $\mathcal{O}(n^2)$ work tape). The total time needed for each $d = 2, \dots, n$ is thus at most n^4 and the whole computation takes at most n^5 steps. □

Języki kontekstowe

For context-sensitive (or monotonic) languages, we have the equality

$$\text{CSL} = \textsc{Nspace}(n),$$

because the "linear bounded automata" (LBA) are essentially Turing machines of space complexity n. The following are two famous problems concerning context-sensitive languages, known since early 60's:

1) Is every context-sensitive language accepted by a deterministic LBA?
2) Is the class of context-sensitive languages closed under complementation?

Problem (1) still remains open, and can be re-stated as:

$$\textsc{Nspace}(n) = \textsc{Dspace}(n)?$$

Problem (2) was positively solved in in 1985 independently by Immerman and Szelepcsényi. This solution follows from Twierdzenie 9.4 below. We first consider an important special case.

Lemat 9.3 *If* $L \in \textsc{Nspace}(\log n)$ *then* $-L \in \textsc{Nspace}(\log n)$.

Dowód: Assume that $L = L(\mathcal{M})$, where $\mathcal{M}$ is a nondeterministic machine of space complexity $\log n$, and fix an input word w. Let $\mathbf{C}_w$ be as in Lemat 8.10. Then the cardinality of $\mathbf{C}_w$ is at most $c \cdot n^k$, for some constant c.

Claim 0: *For any* $\mathcal{C}_1, \mathcal{C}_2 \in \mathbf{C}_w$*, a nondeterministic machine can verify in logarithmic space that* $\mathcal{C}_1 \twoheadrightarrow_{\mathcal{M}} \mathcal{C}_2$ *in at most a given number of steps* $d \leq c \cdot n^k$.

[20] Ta metoda nazywa się „algorytm CYK", od nazwisk: Cocke, Younger i Kasami.

The machine can simply *guess* in order all the configurations occurring in the computation from $\mathcal{C}_1$ to $\mathcal{C}_2$, and verify step by step the correctness of all the guesses. This requires as much space as is needed to store two configurations and a counter to control the length of the computation. The procedure terminates when the counter exceeds the maximum count $c \cdot n^k$ or the given bound d.

Claim 1: *Let $R = \{\mathcal{C} \mid \mathcal{C}_w \twoheadrightarrow \mathcal{C}\}$ and let N be the cardinality of R. A nondeterministic machine can compute the number N in logarithmic space.*

Define $R_d = \{\mathcal{C} \mid \mathcal{C}_w \twoheadrightarrow \mathcal{C}$ in at most d steps$\}$ and $N_d = |R_d|$. We will show how to compute all the numbers N_d in logarithmic space. Then $N = N_d$, where $d \leq c \cdot n^k$ is the least number such that $N_d = N_{d+1}$.

For the beginning, $N_0 = 1$. Let us suppose that N_d was already computed. Our goal is to compute N_{d+1}.

First observe that we can list all $\mathcal{C} \in R_d$, using the following trick. We enumerate all configurations $\mathcal{C} \in \mathbf{C}_w$, and for each of these, we guess if $\mathcal{C} \in R_d$ or not. Every positive guess is then verified in logarithmic space (see Claim 0 above). If anything goes wrong, we reject. We keep a count of the positive guesses, which must be equal at the end to the number N_d. Otherwise we reject.

To compute N_{d+1} we proceed as follows. Enumerate all $\mathcal{C} \in \mathbf{C}_w$ and for each such $\mathcal{C}$ we list all $\mathcal{C}' \in R_d$ as described above. But each time a $\mathcal{C}' \in R_d$ is found, we check in addition if $\mathcal{C} = \mathcal{C}'$ or $\mathcal{C}' \to_{\mathcal{M}} \mathcal{C}$. If so, we conclude that $\mathcal{C} \in R_{d+1}$. We count all such $\mathcal{C}$. If the nondeterministic procedure was successful, i.e., we made all and only correct guesses, the final value of the counter must be N_{d+1}.

Claim 2: A nondeterministic machine can check in logarithmic space that no accepting configuration belongs to R, i.e., that $w \notin L(\mathcal{M})$.

Now we know the number N, and we can list all $\mathcal{C} \in R$, applying the same trick again. Enumerate all $\mathcal{C} \in \mathbf{C}_w$ and guess which ones belong to R. Verify all positive guesses, and reject in case of any failure. Also reject if less than N configurations in R were found, or if any of these is an accepting configuration. Accept otherwise, i.e. when no accepting configuration is reachable from $\mathcal{C}_w$.

Ten dowód można podsumować tak:

- Znając N_d potrafimy generować całe R_d;
- Znając R_d potrafimy obliczyć N_{d+1};
- Znając N potrafimy generować całe R. □

If $\mathcal{K}$ is a complexity class, then by *co-*$\mathcal{K}$ we denote the class $\{-L \mid L \in \mathcal{K}\}$, of complements of languages in $\mathcal{K}$.

Twierdzenie 9.4 (Immerman, Szelepcsényi, 1985) *Jeśli funkcja $S(n) \geq \log n$ jest konstruowalna, to* NSPACE$(S(n))$ = *co-*NSPACE$(S(n))$*, tj. klasa* NSPACE$(S(n)$ *jest zamknięta ze względu na dopełnienie.*

Dowód: We know from Lemat 9.3 that NSPACE$(\log n)$ = *co-*NSPACE$(\log n)$. To generalize this equation to arbitrary $S(n)$ we apply a proof technique known as "padding", which is often

used to "raise" various equations between "small" complexity classes to equations between "large" classes.

Of course it suffices to show *co*-NSPACE($S(n)$) $\subseteq$ NSPACE($S(n)$), so let us assume that L is in NSPACE($S(n)$), and we will prove that $-L \in$ NSPACE($S(n)$). Consider the language $L_1 = \{w\$^{2^{S(|w|)}-|w|} \mid w \in L\}$, where \$ is not in the alphabet of L. This language consists of all words $w\$\ldots\$$ of length $n = 2^{S(|w|)}$ such that $w \in L$. To accept a word $w\$\ldots\$ \in L_1$ we can simulate a machine which recognizes L in space $S(n)$. But now the length of the input is $2^{S(|w|)}$ so our simulation works in space $\log n$. Also $S(|w|) = \log n$ space suffices to store a counter necessary to check that the number of \$'s is exactly $2^{S(|w|)} - |w|$.

It follows that $L_1 \in$ NSPACE($\log n$). By Lemat 9.3 we have $-L_1 \in$ NSPACE($\log n$). Let $-L_1 = L(\mathcal{M})$, where $\mathcal{M}$ is of space complexity $\log n$. We construct a machine $\mathcal{M}_1$ to recognize $-L$ in space $S(n)$. Maszyna $\mathcal{M}_1$ działa dla wejścia w tak, jak maszyna $\mathcal{M}$ dla wejścia $w\$\ldots\$$. Jeśli pojawia się potrzeba wyjścia poza słowo wejściowe w, maszyna „udaje", że czyta dolary, pamiętając jedynie pozycję, w której powinna się znajdować głowica maszyny $\mathcal{M}$. □

A więc mamy następujące relacje pomiędzy klasami złożoności i hierarchią Chomsky'ego:

- Klasa języków regularnych pokrywa się z klasą DSPACE(1) = NSPACE(1).
- Klasa języków bezkontekstowych zawiera się w klasie P
- Klasa języków kontekstowych pokrywa się z klasą NSPACE(n).

10 Najważniejsze klasy złożoności

Następujące skróty oznaczają często spotykane klasy złożoności.

$$\begin{aligned}
\text{L} = \textsc{Logspace} &= \textsc{Dspace}(\log n);\\
\text{NL} = \textsc{Nlogspace} &= \textsc{Nspace}(\log n);\\
\text{P} = \textsc{Ptime} &= \textstyle\bigcup_{k\in\mathbb{N}} \textsc{Dtime}(n^k);\\
\text{NP} = \textsc{NPtime} &= \textstyle\bigcup_{k\in\mathbb{N}} \textsc{Ntime}(n^k);\\
\textsc{Pspace} &= \textstyle\bigcup_{k\in\mathbb{N}} \textsc{Dspace}(n^k) = \bigcup_{k\in\mathbb{N}} \textsc{Nspace}(n^k);\\
\textsc{Exptime} &= \textstyle\bigcup_{k\in\mathbb{N}} \textsc{Dtime}(2^{n^k});\\
\textsc{Nexptime} &= \textstyle\bigcup_{k\in\mathbb{N}} \textsc{Ntime}(2^{n^k});\\
\textsc{Expspace} &= \textstyle\bigcup_{k\in\mathbb{N}} \textsc{Dspace}(2^{n^k}) = \bigcup_{k\in\mathbb{N}} \textsc{Nspace}(2^{n^k}).
\end{aligned}$$

Zauważmy, że w definicji klas PSPACE i EXPSPACE nie ma znaczenia (nie)determinizm, a to z powodu twierdzenia Savitcha. Z Twierdzenia 8.12 wynikają następujące inkluzje:

$$\begin{aligned}
\textsc{Logspace} \subseteq \textsc{Nlogspace} \subseteq \text{P} \subseteq \text{NP}&\\
\subseteq \textsc{Pspace} \subseteq \textsc{Exptime} \subseteq \textsc{Nexptime} \subseteq \textsc{Expspace}.&
\end{aligned}$$

O żadnej z tych inkluzji nie wiadomo, czy jest właściwa. Jednak niektóre z nich muszą być właściwe. Wynika to z twierdzeń o hierarchii 8.7 i 8.8: mamy więc

$$\textsc{Nlogspace} \neq \textsc{Pspace} \neq \textsc{Expspace}$$

i podobnie dla czasu:

$$\mathrm{P} \neq \textsc{Exptime} \text{ i } \mathrm{NP} \neq \textsc{Nexptime}.$$

Dzięki technice zwanej „paddingiem" (por. dowód tw. 9.4), wiadomo, że w tych sprawach każde rozstrzygnięcie pozytywne przenosi się na większe klasy złożoności (a zatem odpowiedzi negatywne przenoszą się w dół). Na przykład mamy taki fakt.

Fakt 10.1 *Jeśli* $\mathrm{P} = \mathrm{NP}$*, to* $\textsc{Exptime} = \textsc{Nexptime}$*.*

Dowód: Przypuśćmy, że $\mathrm{P} = \mathrm{NP}$ i niech $L \in \textsc{Nexptime}$. Mamy niedeterministyczną maszynę Turinga, rozpoznającą język L w czasie wykładniczym, tj. w czasie $2^{p(n)}$, gdzie $p(n)$ jest pewnym wielomianem. Rozpatrzmy język $L_1 = \{w\$^{2^{p(|w|)}-|w|} \mid w \in L\}$. Jeśli teraz $n = |w\$^{2^{p(|w|)}-|w|}|$, to $2^{p(|w|)} = n$, zatem język L_1 jest rozpoznawalny w niedeterministycznym czasie liniowym. Mamy więc $L_1 \in \mathrm{NP}$, a zatem także $L_1 \in \mathrm{P}$. Istnieje więc maszyna Turinga $\mathcal{M}$, rozpoznająca język L_1 w czasie n^k. Można ją łatwo przerobić na maszynę $\mathcal{M}_1$, rozpoznającą L w czasie $(2^{p(n)})^k = 2^{k \cdot p(n)}$. Maszyna $\mathcal{M}_1$ wykonuje te same czynności co maszyna $\mathcal{M}$. Jedyna różnica polega na tym, że inne jest słowo wejściowe. Zamiast czytać dolary ze słowa wejściowego, maszyna $\mathcal{M}_1$ pamięta więc tylko pozycję, w jakiej powinna się znajdować głowica maszyny $\mathcal{M}$. □

Redukcje i zupełność

Mówimy, że język $L_1 \subseteq \mathcal{A}_1^*$ jest *sprowadzalny w pamięci logarytmicznej* (lub *logarytmicznie redukowalny*) do innego języka $L_2 \subseteq \mathcal{A}_2^*$, gdy istnieje funkcja $f : \mathcal{A}_1 \to \mathcal{A}_2$, obliczalna przez deterministyczną maszynę w pamięci logarytmicznej[21] i spełniająca warunek

$$w \in L_1 \quad \Leftrightarrow \quad f(w) \in L_2,$$

dla dowolnego słowa $w \in \mathcal{A}_1^*$. Piszemy wtedy $L_1 \leq_{\log} L_2$. (Czasem żąda się tylko aby funkcja f była obliczalna w czasie wielomianowym. Mówi się wtedy o sprowadzalności wielomianowej i używa symbolu $\leq_p$. Oczywiście $L \leq_{\log} L'$ implikuje $L \leq_p L'$.)

Mamy następujący oczywisty fakt:

Fakt 10.2 *Niech* $\mathcal{K}$ *oznacza jedną z klas złożoności*

$$\textsc{Logspace}, \textsc{Nlogspace}, \mathrm{P}, \mathrm{NP}, \textsc{Pspace}, \textsc{Exptime}, \textsc{Nexptime}, \textsc{Expspace}.$$

Jeśli $L \leq_{\log} L' \in \mathcal{K}$*, to* $L \in \mathcal{K}$*.*

[21] Uwaga: słowo wynikowe $f(w)$ może być dłuższe niż $\log n$. Dlatego zakładamy, że maszyna obliczająca funkcję f wyprowadza pojedynczo kolejne symbole słowa $f(w)$, nie przechowując go nigdy w całości na taśmie roboczej. W ten sposób pamięć robocza maszyny może być logarytmiczna.

Fakt 10.2 pozostaje w mocy dla relacji $\leq_p$, wyjąwszy (być może) przypadek Logspace.

Mówimy, że język L jest *trudny* dla klasy złożoności $\mathcal{K}$ (ze względu na logarytmiczne redukcje), jeśli każdy język $L' \in \mathcal{K}$ redukuje się wielomianowo do L. Jeśli na dodatek język L sam należy do $\mathcal{K}$, to mówimy, że jest *zupełny* w tej klasie ($\mathcal{K}$-zupełny).

Fakt 10.3 *Niech $\mathcal{K}$ oznacza jedną z klas złożoności*

Nlogspace, P, NP, Pspace, Exptime, Nexptime, Expspace.

Jeśli L jest $\mathcal{K}$-trudny, oraz $L \leq_{\log} L'$, to L' jest $\mathcal{K}$-trudny.

Istnieje też pojęcie trudności i zupełności ze względu na redukcje wielomianowe. Dla klas zawierających NP nie ma wiekszego znaczenia, którym z nich się posługujemy; najważniejsze wyniki są takie same, niezależnie od tego, czy mowa o sprowadzalności wielomianowej czy logarytmicznej. Ale dla klas Nlogspace i P tak nie jest. Na przykład każdy język z klasy P jest oczywiście P-zupełny ze względu na sprowadzalność wielomianową.

Następująca łatwa obserwacja wskazuje na znaczenie problemów zupełnych w teorii złożoności:

Fakt 10.4 *Niech $\mathcal{K}_1$ i $\mathcal{K}_2$ oznaczają dwie spośród klas*

Logspace, Nlogspace, P, NP, Pspace, Exptime, Nexptime, Expspace

i niech $\mathcal{K}_1 \subseteq \mathcal{K}_2$. Jeśli jakiś język $\mathcal{K}_2$-zupełny należy do $\mathcal{K}_1$, to $\mathcal{K}_1 = \mathcal{K}_2$.

Pamięć logarytmiczna

Gdy mowa o złożoności pamięciowej, często pojawia się założenie $S(n) \geq \log n$. Jest to bardzo podstawowe wymaganie: pamięć $\log n$ jest konieczna do tego aby maszyna mogła np. znać położenie własnej głowicy wejściowej. Dlatego klasa Logspace jest najmniejszą „naturalną" klasą złożoności. Nieformalnie, algorytmy logarytmiczne można realizować za pomocą skończonej liczby „palców" wskazujących pozycje w słowie wejściowym. Szereg ważnych zadań decyzyjnych można rozwiązywać za pomocą takich maszyn. W szczególności sprawdzanie składniowej poprawności formuły rachunku zdań, i obliczenie jej wartości jest możliwe w Logspace.

Nie wiemy, czy klasa Nlogspace jest istotnym rozszerzeniem Logspace. Znane są jednak problemy w Nlogspace, których nie umiemy rozwiazywać deterministycznie w pamięci logarytmicznej. Jednym z nich jest *problem osiągalności* dla grafów skierowanych:

Dany graf G i wierzchołki a i b. Czy w G istnieje droga od a do b?

Aby przedstawić ten problem w postaci odpowiedniej dla maszyn Turinga, tj. w postaci języka formalnego, musimy ustalić sposób kodowania grafów jako słów.

Reprezentacja grafu skierowanego o m wierzchołkach i k krawędziach to lista złożona z par numerów wierzchołków (rozdzielonych np. przez krzyżyki). Na początku listy możemy jeszcze umieścić liczbę wierzchołków. Oczywiście wszystko jest w postaci binarnej, więc długość kodu grafu jest $\mathcal{O}(k \cdot \log m)$. Tego samego rzędu jest długość n słowa wejściowego dla maszyny, która ma rozwiązywać problem osiągalności. A pamiętajmy, że złożoność mierzymy ze względu na długość reprezentacji grafu, a nie liczbę jego wierzchołków.

Twierdzenie 10.5 *Problem osiągalności jest* NLOGSPACE*-zupełny.*

Dowód: Aby się przekonać, że istnieje droga od a do b, wystarczy „zgadnąć" odpowiedni ciąg krawędzi. Maszyna niedeterministyczna może to łatwo zrobić, a jeśli będzie oszczędnie używać pamięci, to jej złożoność będzie logarytmiczna. W szczególności nie wolno przepisywać numerów wierzchołków na taśmę roboczą, bo to może wymagać pamięci liniowej. Ale my tylko musimy umieć sprawdzać, że koniec jednej krawędzi jest taki sam jak początek innej, a to potrafimy zrobić przesuwając dwa palce po taśmie.

Teraz pokażemy, że osiągalność jest NLOGSPACE-trudna. Niech więc $L = L(\mathcal{M}) \in$ NLOGSPACE. Bez straty ogólności możemy założyć, że przed zakończeniem obliczenia maszyna zeruje taśmy robocze i „parkuje" wszystkie głowice na początku taśm. W ten sposób, dla każdego w konfiguracja końcowa $\mathcal{K}_w$ jest wyznaczona jednoznacznie.

Dla danego słowa w długości n, zbiór $\mathbf{C}_w$ (por. lemat 8.10) ma $\mathcal{O}(n^k)$ elementów, gdzie k jest pewna stałą. Konfiguracje z $\mathbf{C}_w$ tworzą graf G_w, w którym krawędzie $(\mathcal{C}, \mathcal{C}')$ odpowiadają przejściom $\mathcal{C} \to \mathcal{C}'$ maszyny $\mathcal{M}$. Kod grafu G_w, wraz z kodami konfiguracji początkowej $\mathcal{C}_w$ i końcowej $\mathcal{K}_w$, można deterministycznie generować na podstawie relacji przejścia maszyny $\mathcal{M}$. Ponieważ mamy równoważność

$$w \in L(\mathcal{M}) \qquad \text{wtedy i tylko wtedy, gdy} \qquad \mathcal{K}_w \text{ jest osiągalne z } \mathcal{C}_w \text{ w grafie } G_w,$$

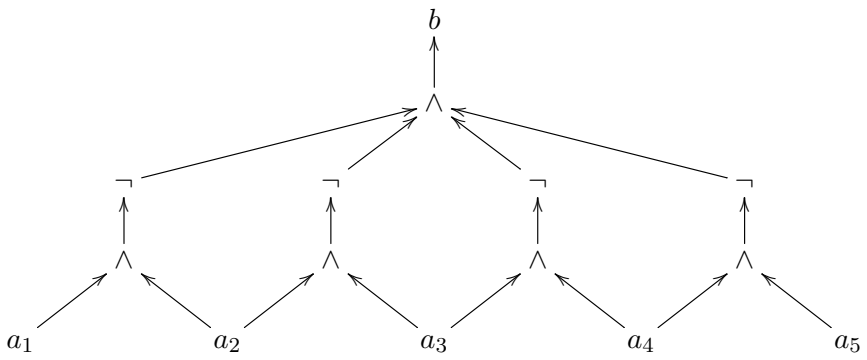
więc określiliśmy właśnie redukcję języka L do problemu osiągalności. Redukcja ta jest logarytmiczna, bo systematycznie generowane krawędzie grafu (pary konfiguracji maszyny $\mathcal{M}$) zajmują tylko logarytmiczny obszar pamięci. □

Moralny sens powyższego dowodu jest taki: w istocie każdy problem z klasy NLOGSPACE jest szczególnym przypadkiem problemu osiągalności.

Czas wielomianowy

Najbardziej znanym problemem zupełnym w klasie P jest *problem wartości sieci boolowskiej* czy też *obwodu logicznego* (circuit value problem). Zamiast definicji sieci narysujmy przykład.

Rysunek 6:



W grafie na obrazku 6 wierzchołki a_i są *wejściowe*, a wierzchołek b jest *końcowy*. Podając na wejście wartości logiczne zmiennych a_i możemy kolejno przyporządkować wartości wszystkim

wierzchołkom, w zależności od tego jaką operację logiczną im przypisano i jakie wartości mają ich poprzedniki w sieci. (Oczywiście sieć musi być grafem zorientowanym acyklicznym, o jednym wierzchołku końcowym). Na przykład jeśli wartościami wierzchołków a_1, a_2, a_3, a_4, a_5 są odpowiednio $0, 1, 1, 0, 0$ to wartością wierzchołka b jest 0. Wartość wierzchołka końcowego nazywamy *wartością sieci*. Problem wartości sieci to oczywiście takie zadanie:

Dana jest sieć i wartości wierzchołków początkowych. Czy wartością sieci jest 1?

Fakt 10.6 *Problem wartości sieci jest w klasie P.*

Dowód: Sieć wraz z wartościami początkowymi może być reprezentowana na wejściu maszyny Turinga jako słowo o długości $n \leq \mathcal{O}(m^2 \log m)$, gdzie m jest liczbą jej wierzchołków. Ewaluacja sieci może bez trudu być zrealizowana w czasie wielomianowym od n: należy przepisać kod sieci na taśmę roboczą i obliczać wartości kolejnych wierzchołków (jest ich nie więcej niż n) aż nie uzyskamy wartości końcowej. □

Okazuje się, że problem wartości sieci nie tylko jest w klasie P, ale że jest też P-zupełny. Aby to wykazać rozpatrzymy pewne jego naturalne uogólnienie.

Sieć boolowska o m wejściach oblicza funkcję typu $\{0,1\}^m \to \{0,1\}$. Można równie dobrze mówić o sieciach obliczających funkcje typu $D^m \to D$ gdzie D jest dowolną ustaloną algebrą skończoną. Wierzchołki takich sieci są etykietowane działaniami w D, a obliczanie ich wartości jest także problemem z klasy P. Nazwijmy go *uogólnionym problemem wartości sieci*:

Dana algebra D, sieć nad D, wartości początkowe i wartość końcowa d.
Czy wartością sieci jest d?

Lemat 10.7 *Uogólniony problem wartości sieci jest* P*-zupełny.*

Dowód: Niech $L \in \textsc{Dtime}(n^k)$. Dla uproszczenia załóżmy, że maszyna $\mathcal{M}$ rozpoznająca L jest „klasyczną" maszyną jednotaśmową,[22] która dla każdego wejścia długości n wykonuje *dokładnie* n^k kroków.

Skonstruujemy sieć nad algebrą o dziedzinie $D = \Sigma \cup (\Sigma \times Q) \cup \{\text{☺}, \text{☹}\}$ i czterech operacjach: trójargumentowej $t : D^3 \to D$ i dwuargumentowych $l, r, f : D^2 \to D$. Wierzchołkami sieci są pary $\langle i, j \rangle$, gdzie $0 \leq i, j \leq n^k$, oraz pewna liczba wierzchołków dodatkowych. Wierzchołki wejściowe są postaci $\langle 0, j \rangle$. Idea jest taka, że wartość wierzchołka $\langle i, j \rangle$ powinna być literą zapisaną w j-tej klatce taśmy po wykonaniu i kroków obliczenia. Chyba, że w czasie i głowica akurat znajduje się w pozycji j; wtedy wartością wierzchołka $\langle i, j \rangle$ ma być para złożona ze stanu maszyny i zawartości klatki j w chwili i. Dla danego słowa wejściowego $w = a_1 \dots a_n$ maszyny $\mathcal{M}$ wierzchołkom początkowym nadajemy wartości $\mathsf{B}, \langle q_0, a_1 \rangle, a_2, \dots, a_n, \mathsf{B}, \dots, \mathsf{B}$, odpowiadające zawartości taśmy w chwili 0.

Konstrukcja sieci nie jest trudna, jeśli uświadomimy sobie, że zawartość j-tej klatki w chwili i jest jednoznacznie wyznaczona przez zawartość klatek $j-1$, j, $j+1$ w chwili $i-1$. Inaczej mówiąc, wartość pozycji $\langle i, j \rangle$ można obliczyć z wartości trzech pozycji $\langle i-1, j-1 \rangle$, $\langle i-1, j \rangle$, $\langle i-1, j+1 \rangle$. Potrzebna do tego operacja trójargumentowa t jest zdeterminowana przez relację

[22] Łatwo pokazać, że dla każdej deterministycznej maszyny $\mathcal{M}$ o złożoności czasowej $T(n)$ istnieje klasyczna maszyna deterministyczna o złożoności czasowej $\mathcal{O}(T(n)^2)$ rozpoznająca ten sam język.

przejścia maszyny $\mathcal{M}$. Ta operacja jest etykietą wierzchołków $\langle i, j\rangle$ dla $i > 0$ oraz $j \neq 0, n^k$. Poprzednikami takich wierzchołków są oczywiście $\langle i-1, j-1\rangle$, $\langle i-1, j\rangle$ oraz $\langle i-1, j+1\rangle$. W naszej algebrze D musimy jeszcze mieć dwie operacje binarne l i r do „obsługi" skrajnych klatek taśmy, które będą etykietami dla $\langle i, 0\rangle$ i $\langle i, n^k\rangle$. Poprzednikami wierzchołka $\langle i, 0\rangle$ są $\langle i-1, 0\rangle$ oraz $\langle i-1, 1\rangle$, a poprzedniki $\langle i, n^k\rangle$ to $\langle i-1, n^k-1\rangle$ oraz $\langle i-1, n^k\rangle$.

Konstrukcja naszej sieci wymaga jeszcze fazy końcowej. Jest to drzewo binarne, którego liśćmi są wierzchołki postaci $\langle n^k, j\rangle$. Węzły tego drzewa etykietowane są operacją dwuargumentową f, której zadaniem jest przekazanie w górę informacji o stanie końcowym.

$$f(x,y) = \begin{cases} \text{☺}, & \text{jeśli } x \text{ lub } y \text{ jest postaci ☺ lub } \langle q_f, a\rangle; \\ \text{☹}, & \text{w przeciwnym przypadku.} \end{cases}$$

Możemy teraz łatwo się przekonać, że zachodzi równoważność: maszyna $\mathcal{M}$ akceptuje w wtedy i tylko wtedy, gdy wartością naszej sieci jest ☺. Co więcej, dla danego w o długości n, całą konstrukcję naszej sieci można przeprowadzić w pamięci logarytmicznej. □

Twierdzenie 10.8 *Problem wartości sieci jest* P*-zupełny.*

Dowód: Zredukujemy uogólniony problem do jego podstawowej wersji. Przypuśćmy, że mamy sieć nad r-elementową algebrą D. Każdy z tych elementów może być reprezentowany za pomocą $m = \lceil \log r \rceil$ wartości boolowskich. Każda l-argumentowa operacja w D może też być przedstawiona z pomocą m funkcji zerojedynkowych, z których każda ma $l \cdot m$ argumentów. A każda funkcja zerojedynkowa może być wyrażona za pomocą działań logicznych $\vee$, $\wedge$ i $\neg$. A zatem sieć nad D można przekształcić w zwykłą sieć boolowską kosztem m-krotnego zwiększenia liczby wierzchołków i r-krotnego zwiększenia liczby krawędzi. Także i ta konstrukcja jest redukcją w pamięci logarytmicznej. □

Przykład 10.9 Rozpatrzmy funkcję $F : \{\text{a,b,c,d}\}^2 \to \{\text{a,b,c,d}\}$, która jest określona tabelką:

F	a	b	c	d
a	a	b	a	c
b	d	a	b	a
c	c	b	a	b
d	a	a	d	a

Przyjmując kodowanie a = 00, b = 01, c = 10, d = 11, możemy tę funkcję zastąpić dwoma czteroargumentowymi operacjami binarnymi i dla każdej z nich zdefiniować obwód logiczny.

Pierwszy bit

F1	00	01	10	11
00	0	0	0	1
01	1	0	0	0
10	1	0	0	0
11	0	0	1	0

Drugi bit

F2	00	01	10	11
00	0	1	0	0
01	1	0	1	0
10	0	1	0	1
11	0	0	1	0

Pierwszy bit wyraża się formułą o 4 zmiennych p, q, r, s:

$$\text{F1}(p,q,r,s) = (\neg p \wedge \neg q \wedge r \wedge s) \vee (\neg p \wedge q \wedge \neg r \wedge \neg s) \vee (p \wedge \neg q \wedge \neg r \wedge \neg s) \vee (p \wedge q \wedge r \wedge \neg s).$$

Zmienne p, q reprezentują element a, zmienne r, s reprezentują element b.

11 Klasa NP

Pytanie o równość P = NP, jest głośnym problemem otwartym. Przypomnijmy, że z faktów 10.2, 10.3 i 10.4 wynika szczególne znaczenie problemów NP-zupełnych dla tego zagadnienia:

Fakt 11.1

0) Jeśli $L \leq_p L' \in \mathrm{P}$*, to* $L \in \mathrm{P}$*.*

1) Jeśli problem (język) L *jest* NP*-zupełny, a przy tym* $L \in \mathrm{P}$*, to* $\mathrm{P} = \mathrm{NP}$*.*

2) Jeśli L *jest* NP*-zupełny, oraz* $L \leq_p L' \in \mathrm{NP}$*, to* L' *jest* NP*-zupełny.*

Problem spełnialności formuł rachunku zdań (SAT) to następujący problem decyzyjny:

Dana formuła zdaniowa, czy istnieje wartościowanie spełniające tę formułę?

Nietrudno zauważyć, że SAT $\in$ NP. Problem SAT był pierwszym znanym problemem NP-zupełnym.

Twierdzenie 11.2 (Cook, Levin) *Problem* SAT *jest* NP*-zupełny.*

Dowód: Dla dowolnej niedeterministycznej maszyny Turinga $\mathcal{M}$, działającej w czasie wielomianowym $f(n)$ i dowolnego słowa w definiuje się formułę $\Phi_{\mathcal{M},w}$, o takiej własności:

$$w \in L(\mathcal{M}) \qquad \text{wtedy i tylko wtedy, gdy} \qquad \Phi_{\mathcal{M},w} \text{ jest spełnialna.}$$

Konstrukcja formuły $\Phi_{\mathcal{M},w}$ jest efektywna. Jeśli ustalimy maszynę $\mathcal{M}$, to konstrukcja ta jest wykonywalna w pamięci logarytmicznej ze względu na długość słowa w. Zatem dowolny język $L \in$ NP sprowadzamy w ten sposób do problemu SAT.

Pozostaje opisać budowę formuły $\Phi_{\mathcal{M},w}$. Załóżmy, że nasza maszyna działa w czasie $f(n)$, ma k stanów, t taśm i używa alfabetu Σ. Stan początkowy *start*, stan końcowy *stop*. Niech $|w| = n$. Formuła $\Phi_{\mathcal{M},w}$ jest zbudowana ze zmiennych zdaniowych

- $p_{i,j,\ell,a}$, dla $i \leq f(n)$, $j \leq f(n)$, $\ell \leq t$, $a \in \Sigma$;
- $q_{i,s}$, dla $i \leq f(n)$, $s \leq k$;
- $r_{i,j,\ell}$, dla $i \leq f(n)$, $j \leq f(n)$, $\ell \leq t$;
- $r_{i,j,0}$, dla $i \leq f(n)$, $j \leq n+1$.

Idea jest taka, aby każde wartościowanie spełniające naszą formułę odpowiadało pewnemu obliczeniu maszyny $\mathcal{M}$. Interesują nas więc te wartościowania ρ, które dla pewnego obliczenia maszyny spełniają warunki:

1. $\rho(q_{i,s}) = 1$, gdy po i krokach obliczenia maszyna jest w stanie s.
2. $\rho(p_{i,j,\ell,a}) = 1$, gdy po i krokach obliczenia, na j-tym miejscu ℓ-tej taśmy stoi symbol a.
3. $\rho(r_{i,j,\ell}) = 1$, gdy po i krokach obliczenia głowica ℓ-tej taśmy roboczej jest w pozycji j.
4. $\rho(r_{i,j,0}) = 1$, gdy po i krokach obliczenia głowica taśmy wejściowej jest w pozycji j.

Formuła $\Phi_{\mathcal{M},w}$ jest długą koniunkcją formuł tak wybranych, że spełnienie wszystkich na raz jest możliwe tylko przez wartościowania ρ o własnościach (1)–(4). Na przykład, mamy takie człony koniunkcji:

- $p_{i,j,\ell,a_1} \vee p_{i,j,\ell,a_2} \vee \cdots \vee p_{i,j,\ell,a_d}$, gdzie $\Sigma = \{a_1, \ldots, a_d\}$ (sens: w danej klatce taśmy musi być jakiś symbol);
- $\neg(p_{i,j,\ell,a_x} \wedge p_{i,j,\ell,a_y})$, dla $x \neq y$ (sens: ale nie dwa na raz);
- $p_{0,j,\ell,\mathsf{B}}$ (sens: na taśmach roboczych na początku są same blanki);
- $q_{0,start} \wedge q_{f(n),stop}$ (sens: zaczynamy i kończymy we właściwych stanach).

i tak dalej. Najważniejszą część formuły stanowią te człony, które reprezentują związki pomiędzy kolejnymi konfiguracjami maszyny. Przypuśćmy na przykład, że maszyna ma dwie taśmy robocze, i taką relację przejścia, że:

- w stanie s, widząc na taśmie wejściowej a, a na taśmach roboczych b i c, maszyna może:
- albo przejść do stanu s', zapisując na taśmach roboczych d i e i przesuwając wszystkie głowice w prawo,
- albo przejść do stanu s'', zapisując na taśmach roboczych e i a, i przesuwając głowicę wejściową w prawo, a obie robocze w lewo.

Wtedy nasza formuła ma dla dowolnych $i, j_1, j_2 \leq f(n)$ oraz $j_0 \leq n$ taki człon koniunkcji:

$(q_{i,s} \wedge r_{i,j_0,0} \wedge r_{i,j_1,1} \wedge r_{i,j_2,2} \wedge p_{i,j_0,0,a} \wedge p_{i,j_1,1,b} \wedge p_{i,j_2,2,c}) \rightarrow (q_{i+1,s'} \wedge r_{i+1,j_0+1,0} \wedge r_{i+1,j_1+1,1} \wedge r_{i+1,j_2+1,2} \wedge p_{i+1,j,1,d} \wedge p_{i+1,j,2,e}) \vee (q_{i+1,s''} \wedge r_{i+1,j_0+1,0} \wedge r_{i+1,j_1-1,1} \wedge r_{i+1,j_2-1,2} \wedge p_{i+1,j,1,e} \wedge p_{i+1,j,2,a})$.

Zwróćmy uwagę na to, że liczba wszystkich członów naszej koniunkcji jest wielomianowa i można je wszystkie po kolei generować używając do tego tylko logarytmicznej pamięci. □

Z twierdzenia 11.2 wynika natychmiast:

Wniosek 11.3 *Jeśli* $\mathrm{SAT} \in \mathrm{P}$*, to* $\mathrm{P} = \mathrm{NP}$*.*

Z twierdzenia 11.2 wynika też następująca charakteryzacja klasy NP.

Wniosek 11.4 *Dla każdego języka* $L \in \mathrm{NP}$ *istnieje taki język* $L' \in \mathrm{P}$*, i taki wielomian* p*, że dla dowolnego słowa* x *zachodzi równoważność:*

$$x \in L \quad \Leftrightarrow \quad \exists v\, (|v| \leq p(|x|) \wedge x\$v \in L')$$

Dowód: Ponieważ $L \leq_{\log} \mathrm{SAT}$, więc dowolne słowo x należy do L wtedy i tylko wtedy, gdy odpowiednia formuła Φ_x jest spełnialna. Inaczej: gdy istnieje spełniające ją wartościowanie v, co zapiszemy $v \models \Phi_x$. Takie wartościowanie przedstawiamy za pomocą ciągu zerojedynkowego długości nie większej niż długość formuły. A ta długość jest wielomianowa. To oznacza, że dla pewnego wielomianu p mamy równoważność

$$x \in L \quad \Leftrightarrow \quad \exists v\, (|v| \leq p(|x|) \wedge v \models \Phi_x),$$

i możemy przyjąć $L' = \{x\$v \mid v \models \Phi_x\}$. □

A zatem przynależność słowa x do języka L z klasy NP można zweryfikować sprawdzając deterministycznie w czasie wielomianowym poprawność „certyfikatu" zadanego słowem v. Sens powyższego wniosku jest więc następujący: problemy z klasy NP to dokładnie[23] te problemy, których rozwiązania mają „łatwe certyfikaty".

Problem spełnialności pozostaje NP-zupełny nawet w dość szczególnym przypadku. Wystarczy rozważać formuły w *koniunkcyjnej postaci normalnej*, tj. formuły postaci

$$(\alpha_{11} \vee \cdots \vee \alpha_{1k_1}) \wedge (\alpha_{21} \vee \cdots \vee \alpha_{2k_2}) \wedge \cdots \wedge (\alpha_{m1} \vee \cdots \vee \alpha_{mk_m}),$$

gdzie każde α_{ij} jest albo zmienną zdaniową albo jest postaci $\neg p$, gdzie p jest zmienną zdaniową. (Mówimy, że każde α_{ij} jest *literałem*.) Wariant problemu SAT, oznaczany skrótem CNF-SAT, polega na ustaleniu, czy dana formuła w koniunkcyjnej postaci normalnej jest spełnialna.

Uwaga: Wiadomo, że każda formuła jest równoważna pewnej formule w postaci CNF. Ale rozmiar tej ostatniej jest w najgorszym przypadku wykładniczy w stosunku do rozmiaru formuły początkowej. Zatem następujący fakt nie jest oczywisty.

Fakt 11.5 CNF-SAT *is* NP*-complete.*

Dowód: Chcemy zredukować problem SAT do CNF-SAT. First transform the given formula[24] so that all negations are applied to propositional variables only. This procedure takes only polynomial time and can at most duplicate the size of the formula. Then we eliminate subformulas of the form

$$(\phi_1 \wedge \cdots \wedge \phi_n) \vee (\psi_1 \wedge \cdots \wedge \psi_m),$$

where all the ϕ's and ψ's are disjunctions of literals, by observing that the above is satisfiable if and only if the following formula (where p must be a new variable) is satisfiable:

$$(p \vee \phi_1) \wedge \cdots \wedge (p \vee \phi_n) \wedge (\neg p \vee \psi_1) \wedge \cdots \wedge (\neg p \vee \psi_m).$$

The number of times this procedure is applied is less than the length of the formula, and the increase in size is quadratic in the worst case.

The resulting formula is not equivalent to the initial one (because new variables are introduced) but the former is satisfiable if and only if so is the latter. We conclude that we have just defined a reduction SAT $\leq_{\log}$ CNF-SAT. □

A further restriction is to consider only CNF formulas with each of the components consisting of at most three disjuncts:

$$(\alpha_{11} \vee \alpha_{12} \vee \alpha_{13}) \wedge (\alpha_{21} \vee \alpha_{22} \vee \alpha_{23}) \wedge \cdots \wedge (\alpha_{m1} \vee \alpha_{m2} \vee \alpha_{m3})$$

przy czym każde α_{ij} jest literałem. Ten wariant problemu SAT oznaczamy przez 3-CNF-SAT.

Twierdzenie 11.6 *Problem* 3-CNF-SAT *jest* NP*-zupełny.*

Dowód: Consider a single disjunction $(\alpha_1 \vee \cdots \vee \alpha_n)$, where $n \geq 4$, and let $q_1, \ldots, q_{n-3}$ be new variables (not occurring in α_i). Observe that $(\alpha_1 \vee \cdots \vee \alpha_n)$ is satisfiable iff the following is satisfiable:

[23] Twierdzenie odwrotne do wniosku 11.4 jest oczywiste.

[24] Możemy zakładać, że w formule nie ma już implikacji.

$$(\alpha_1 \vee \alpha_2 \vee q_1) \wedge (\alpha_3 \vee \neg q_1 \vee q_2) \wedge \cdots \wedge (\alpha_{n-2} \vee \neg q_{n-4} \vee q_{n-3}) \wedge (\alpha_{n-1} \vee \alpha_n \vee \neg q_{n-3}).$$

Using this trick we can "break" long disjunctions into shorter ones in linear time. This constitutes a reduction CNF-SAT $\leq_{\log}$ 3-CNF-SAT. □

Uwaga: Alternatywę $\alpha_1 \vee \alpha_2$ można zawsze zastąpić przez $\alpha_1 \vee \alpha_2 \vee \alpha_2$, więc za instancję problemu 3-CNF-SAT można uważać każdą koniunkcję alternatyw *co najwyżej* trójczłonowych.

Kolorowanie grafu

Obecnie znanych jest wiele problemów NP-zupełnych. Na przykład problem *kolorowania grafu*, polegający na stwierdzeniu czy wierzchołki danego grafu G można tak pokolorować daną liczbą kolorów, aby sąsiadujące wierzchołki miały inne kolory.

Given a graph G and a number k, is it possible to assign a color to each node of G in such a way that adjacent nodes are always of different colors?

Twierdzenie 11.7 *Problem kolorowania grafu jest* NP*-zupełny.*

Dowód: Redukcja problemu 3-CNF-SAT do kolorowania. Przypuśćmy, że mamy formułę

$$\varphi = (\alpha_{11} \vee \alpha_{12} \vee \alpha_{13}) \wedge (\alpha_{21} \vee \alpha_{22} \vee \alpha_{23}) \wedge \cdots \wedge (\alpha_{m1} \vee \alpha_{m2} \vee \alpha_{m3}),$$

w której każde α_{ij} jest jedną ze zmiennych zdaniowych $p_1, \ldots, p_n$ lub jej negacją. Bez straty ogólności można zakładać, że $n > 3$. Skonstruujemy taki graf G, który można pokolorować $n+1$ kolorami wtedy i tylko wtedy, gdy formuła φ jest spełnialna.

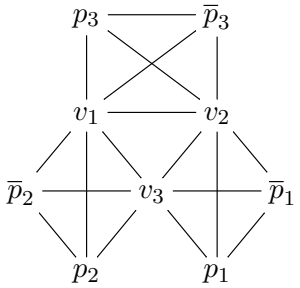
Graf G będzie miał $3n + m$ wierzchołków, które oznaczymy przez:

$$p_1, \ldots, p_n, \overline{p}_1, \ldots, \overline{p}_n, v_1 \ldots, v_n, f_1 \ldots, f_m.$$

Krawędzie grafu będą takie:

- (v_i, v_j), dla $i \neq j$;
- (v_i, p_j) i $(v_i, \overline{p}_j)$, dla $i \neq j$;
- $(p_i, \overline{p}_i)$, dla wszystkich i;
- (p_i, f_k), gdy $p_i \neq \alpha_{k\ell}$ dla $\ell = 1, 2, 3$;
- $(\overline{p}_i, f_k)$, gdy $\neg p_i \neq \alpha_{k\ell}$ dla $\ell = 1, 2, 3$.

Dla dwóch zmiennych i trzech kolorów w naszym grafie będzie więc podgraf takiego kształtu:



A jeśli nasza formuła zaczyna sie tak:

$$\varphi = (p_1 \vee p_2) \wedge (\neg p_1 \vee p_2) \wedge \neg p_2 \wedge \ldots,$$

to dodatkowo wierzchołek f_1 reprezentujący pierwszy człon koniunkcji jest połączony z wierzchołkami $\overline{p}_1$, $\overline{p}_2$, p_3, $\overline{p}_3$ a wierzchołek f_2 z wierzchołkami $p_1, \overline{p}_2, p_3, \overline{p}_3$. Wierzchołek f_3 jest połączony ze wszystkim oprócz $\overline{p}_2$.

Jeśli chcemy pokolorować tak skonstruowany graf, to wierzchołki v_i muszą mieć inne kolory. Nazwiemy je kolorami *zwykłymi*. Dla dowolnego i, oba wierzchołki p_i i $\overline{p}_i$ są połączone z $n-1$ spośród wierzchołków v_i, a zatem jeden z nich musi mieć n-ty kolor zwykły, a drugi musi mieć jeszcze inny kolor, który nazwiemy *specjalnym*.

Jeśli mamy wartościowanie ρ spełniające naszą formułę, to kolory wierzchołków p_i i $\overline{p}_i$ możemy ustalić tak: przypiszemy wierzchołkowi p_i kolor zwykły jeżeli $\rho(p_i) = 1$, w przeciwnym razie nadamy mu kolor specjalny. Pozostaje przypisać kolory wierzchołkom f_k. Wybierzmy dowolne $k = 1, \ldots, m$. Dla pewnego $\ell = 1, 2, 3$ musi być $\rho(\alpha_{k\ell}) = 1$. Przypuśćmy, że $\alpha_{k\ell} = p_i$. Wtedy p_i ma zwykły kolor. Ponieważ wierzchołek f_k nie jest połączony z wierzchołkiem p_i, więc można mu bezpiecznie nadać ten sam kolor.

Na odwrót, jeżeli istnieje sposób na pokolorowanie naszego grafu, to kolory przypisane wierzchołkom p_i definiują pewne wartościowanie ρ. To wartościowanie spełnia formułę φ. Aby się o tym przekonać, zauważmy najpierw, że wierzchołki f_k muszą mieć kolory zwykłe. A to dlatego, że $n > 3$ i musi istnieć taka zmienna p_i która nie występuje w k-tym członie koniunkcji (także zanegowana). Wtedy f_k jest połączone zarówno z p_i jak i z $\overline{p}_i$, a któryś z tych dwóch wierzchołków ma kolor specjalny.

Przypuśćmy teraz, że k-ty człon koniunkcji ma na przykład postać $p_3 \vee p_5 \vee \neg p_1$. Gdyby wartość tego członu byla zerem, to $\rho(p_3) = \rho(p_5) = 0$ oraz $\rho(p_1) = 1$. To znaczy, że kolor wierzchołków $p_3, p_5, \overline{p}_1$ jest specjalny, a kolory wierzchołków $\overline{p}_3, \overline{p}_5, p_1$ są zwykłe. Wierzchołek f_k jest połączony z tymi ostatnimi, więc nie może przyjąć żadnego z ich kolorów. Ale nie może też mieć żadnego innego zwykłego koloru, bo dla $i \neq 1, 3, 5$ mamy dwie krawędzie $(f_k, p_i), (f_k, \overline{p}_i)$.

Wracając do przykładu z rysunku: tego grafu nie da się pokolorować czterema kolorami. Jeśli wierzchołki v_1, v_2, v_3 są odpowiednio niebieskie, zielone i czerwone, to f_3 i $\overline{p}_2$ muszą być zielone. Dalej f_1 nie może być zielone (bo przylega do $\overline{p}_2$) ani czerwone (bo przylega do p_3 i $\overline{p}_3$), więc musi być niebieskie. Dla f_2 zabrakło koloru. □

Exact Cover

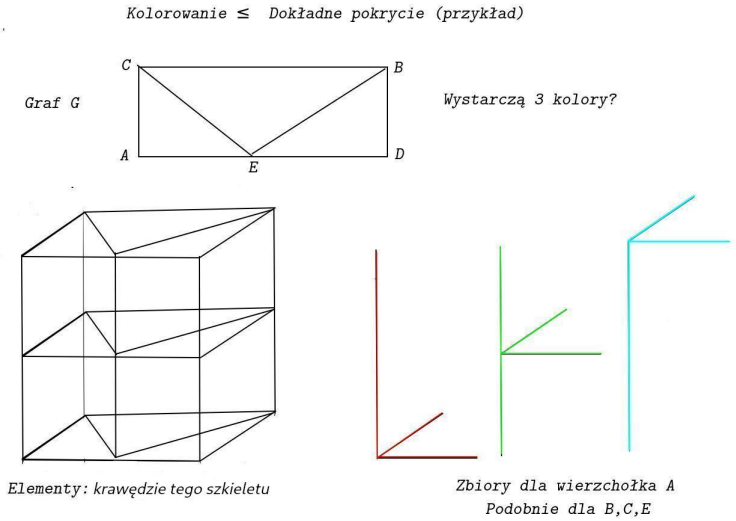
The *Exact Cover* problem is formulated as follows:

(EC) *Given a finite set U and subsets $S_1, S_2, \ldots, S_\ell$ of U, is there a subfamily $\{S_{i_1}, \ldots, S_{i_p}\}$ of $\{S_1, S_2, \ldots, S_\ell\}$ such that the sets S_{i_j} are pairwise disjoint and $S_{i_1} \cup \cdots \cup S_{i_p} = U$?*

Przykład 11.8 Rodzina zbiorów

$$\mathcal{R} = \{\{1,2,3\}, \{2,3\}, \{3\}, \{1\}, \{2,5\}, \{2,4,6\}, \{4,6\}, \{3,4,5,6\}, \{3,4,5\}, \{3,7,8\}\}$$

ma dokładne pokrycie $\{\{1\},\{3,7,8\},\{2,5\},\{4,6\}\}$. Jeśli z $\mathcal{R}$ usuniemy np. zbiór $\{2,5\}$, to dokładnego pokrycia już nie będzie.



Twierdzenie 11.9 *The exact cover problem is* NP*-complete.*

Dowód: The exact cover problem is in NP, because a nondeterministic machine can simply guess the sets S_{i_j} and verify the condition. To show NP-hardness, we reduce the graph coloring problem to the exact cover problem.

Let $G = (V, E)$ be a graph and let k be a given number of colors. Define

$$U = V \cup \{\langle e, i\rangle \mid e \in E \wedge i \in \{1, \ldots, k\}\},$$

and consider the family of all subsets of U of the form:

- $S_{vi} = \{v\} \cup \{\langle e, i\rangle \mid e \in E, \text{ and } v \in e\}$, and
- $T_{ei} = \{\langle e, i\rangle\}$,

for arbitrary $v \in V$, $e \in E$ and $i \in \{1, \ldots, k\}$. (Since edges in an undirected graph are simply pairs of nodes, the notation $v \in e$ means that v is adjacent to e.)

We claim that G is k-colorable if and only if there is a disjoint family of the S_{vi}'s and T_{ei}'s which covers the whole set U.

Suppose that we can paint G with k colors. Then the family

$$\mathcal{F} = \{S_{vi} \mid v \text{ is assigned the } i\text{-th color}\}$$

consists of pairwise disjoint sets. Indeed, each v belongs to only one such set, and for $w \neq v$, if $e \in S_{vi} \cap S_{wj}$ then $i = j$ and $e = \{w, v\}$. Thus the adjacent nodes w and v would have the same color.

To obtain a family of disjoint sets summing up to U one adds to $\mathcal{F}$ all the sets T_{ei}, where neither end of e has color i.

On the other hand, if we can cover U with a family $\mathcal{F}$ of disjoint sets of the form S_{vi} and T_{ei}, then each node v belongs to only one $S_{vi} \in \mathcal{F}$. Choose i as the color of v. Two adjacent nodes w, v cannot share a color, because the sets S_{vi} and S_{wi} are not disjoint and cannot both belong to $\mathcal{F}$. □

Pokrycie wierzchołkowe

A subset $C \subseteq V$ of the set of nodes of a graph $G = (V, E)$ is called a *vertex cover* iff every edge in G is adjacent to a node in C. The *Vertex Cover* problem is to determine if a given graph has a vertex cover of a given size k.

Dany graf $G = (V, E)$ i liczba k. Czy istnieje taki k-elementowy zbiór $P \subseteq V$, że każda krawędź ma przynajmniej jeden koniec w zbiorze P?

Twierdzenie 11.10 *The vertex cover problem is* NP*-complete.*

Dowód: We reduce 3-CNF-SAT to the vertex cover problem (which is easily seen to be in NP). For a given formula

$$\varphi = (\alpha_{11} \vee \alpha_{12} \vee \alpha_{13}) \wedge (\alpha_{21} \vee \alpha_{22} \vee \alpha_{23}) \wedge \cdots \wedge (\alpha_{m1} \vee \alpha_{m2} \vee \alpha_{m3}),$$

we construct a graph $G = (V, E)$, where

- $V = \{\langle i, j\rangle \mid i = 1, \ldots, m \text{ and } j = 1, 2, 3\}$;
- $E = \{\{\langle i, j\rangle, \langle i, k\rangle\} \mid j \neq k\} \cup \{\{\langle i, j\rangle, \langle \ell, k\rangle\} \mid \ell \neq k \text{ and } \alpha_{ij} = \neg\alpha_{\ell k}\}$.

We show that φ is satisfiable iff G has a vertex cover of cardinality $2m$. Suppose first that a valuation ρ satisfies φ. Then for each i, at least one among $\alpha_{i1}, \alpha_{i2}, \alpha_{i3}$ is true under ρ. Choose j_i so that $\rho(\alpha_{ij_i}) = 1$, and define $C = \{\langle i, j\rangle \mid j \neq j_i\}$. Then C is a vertex cover.

Indeed, there are two elements of C in each "triangle" of the form $\langle i, 1\rangle, \langle i, 2\rangle, \langle i, 3\rangle$. Thus every edge of type $\{\langle i, j\rangle, \langle i, k\rangle\}$ is adjacent to one of them. Consider an edge of type $\{\langle i, j\rangle, \langle \ell, k\rangle\}$, where $\ell \neq k$. If both $\langle i, j\rangle, \langle \ell, k\rangle \notin C$ then $\rho(\alpha_{ij}) = \rho(\alpha_{\ell k}) = 1$, which is impossible as α_{ij} and $\alpha_{\ell k}$ are negations of each other.

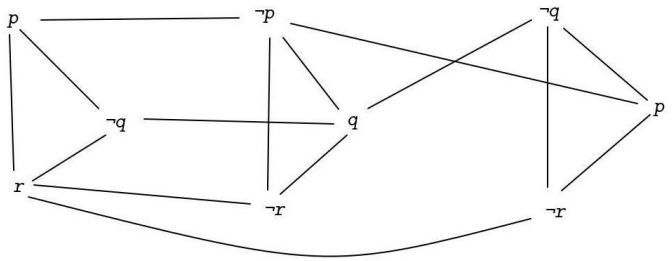
For the other direction, assume that G has a vertex cover C of $2m$ elements. To account for edges of the form $\{\langle i, j\rangle, \langle i, k\rangle\}$, the set C must contain two nodes $\langle i, j\rangle$, $\langle i, \ell\rangle$ for each i. Let j_i be such that $\langle i, j_i\rangle \notin C$. Define a valuation ρ so that

$$\rho(p) = \begin{cases} 1, & \text{jeśli } p = \alpha_{ij_i} \text{ for some } i; \\ 0, & \text{w przeciwnym przypadku.} \end{cases}$$

Observe that it is impossible that $\alpha_{ij_i} = p$, and then $\alpha_{kj_k} = \neg p$ for any i, k, because in this case we have an edge connecting two nodes $\langle j, j_i\rangle, \langle k, j_k\rangle \notin C$. It follows that if $\alpha_{kj_k} = \neg p$ then $\rho(p) = 0$. We conclude that $\rho(\alpha_{ij_i}) = 1$ for each i and our formula is satisfied. □

Pokrycie wierzchołkowe: przykład

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



Czy istnieje pokrycie wierzchołkowe rozmiaru 6?

Równy podział

Our next example is the *Equal Partition* problem, given as follows:

(EP) *Given a finite multiset* $A = \{a_1, \ldots, a_k\} \subseteq \mathbb{N} - \{0\}$*, is there a subset* $\{a_{i_1}, \ldots, a_{i_\ell}\} \subseteq A$ *such that* $a_{i_1} + \cdots + a_{i_\ell} = \frac{1}{2}(a_1 + \cdots + a_k)$*?*

Twierdzenie 11.11 *The equal partition problem is* NP*-complete.*

Dowód: Again, the problem can easily be seen to be in NP: It is enough to guess the subset and calculate the two sums. The NP-hardness can be shown by a logarithmic reduction from the exact cover problem.

Niech U będzie skończonym zbiorem i niech $S_1, S_2, \ldots, S_\ell$ będą podzbiorami U. Bez straty ogólności możemy zakładać, że $U = \{1, 2, \ldots, m\}$, dla pewnego m. Niech r będzie ustaloną liczbą. Podzbiór $\{j_1, \ldots, j_p\}$ zbioru U możemy kodować jako sumę

$$2^{j_1 r} + \ldots + 2^{j_p r}.$$

Jedynki w zapisie binarnym tej sumy wskazują na elementy zbioru. Niech a_i oznacza kod zbioru S_i i niech N będzie kodem całego U. Zapis binarny liczby N ma jedynki na wszystkich pozycjach postaci jr dla $j \leq m$.

Zauważmy teraz, że (dla dostatecznie dużego r) rodzina $\{S_{i1}, \ldots, S_{ik}\}$ stanowi dokładne pokrycie U wtedy i tylko wtedy, gdy $a_{i1} + \cdots + a_{ik} = N$. (Jedynki muszą być dostatecznie daleko od siebie, aby suma kolumny bitów wysokości ℓ, czyli liczba nie większa od ℓ, zmieściła się pomiędzy nimi. Wystarczy więc wybrać $r > \log \ell$.)

Niech $M = a_1 + \cdots + a_\ell$. Można zakładać, że $M \geq N$, bo inaczej rozwiązania na pewno nie ma i redukcja może być trywialna. Pokażemy, że równy podział zbioru liczb

$$A = \{a_1, \ldots, a_\ell, M + N, 2M - N\}.$$

istnieje wtedy i tylko wtedy, gdy rodzina $\{S_1, S_2, \ldots, S_\ell\}$ ma dokładne pokrycie. Główna obserwacja jest taka, że dwie ostatnie liczby $M + N$ i $2M - N$ są za duże na to, aby znalazły

się w tej samej składowej podziału. Zatem równy podział jest możliwy wtedy i tylko wtedy, gdy istnieją $a_{i1}, \ldots, a_{ip}$ o własności

$$a_{i1} + \cdots + a_{ip} + 2M - N = 2M,$$

czyli $a_{i1} + \cdots + a_{ip} = N$, co odpowiada rozwiązaniu problemu dokładnego pokrycia. □

Maksymalny przekrój

Problem *maksymalnego przekroju* MAX-CUT jest określony tak:

Dany jest graf, którego krawędzie mają wagi będące liczbami naturalnymi.
Czy zbiór wierzchołków grafu można podzielić na dwie rozłączne części w taki sposób, że suma wag krawędzi pomiędzy tymi częściami będzie co najmniej taka, jak dana liczba α?

Twierdzenie 11.12 *Problem maksymalnego przekroju jest* NP*-zupełny.*

Dowód: Redukujemy problem równego podziału (*Equal Partition*) do problemu maksymalnego przekroju. Dane są liczby $a_1, \ldots, a_k$ o sumie S. Tworzymy graf pełny o wierzchołkach $\{1, \ldots, k\}$, w którym krawędź łącząca wierzchołek i z wierzchołkiem j ma wagę $a_i a_j$. Każdy podział multizbioru liczb $a_1, \ldots, a_k$ na dwie części $a_{i_1} + \cdots + a_{i_m} = S_1$ oraz $a_{j_1} + \cdots + a_{j_r} = S_2$ wyznacza przekrój o wadze $S_1 \cdot S_2$. Skoro $S_1 + S_2 = S$ wystarczy zapytać o przekrój wagi $\frac{1}{4}S^2$. Taki przekrój jest możliwy tylko dla $S_1 = S_2 = \frac{1}{2}S$. □

Trójpłciowe małżeństwa

The problem of *Three-Dimensional Matching* (3DM) generalizes the well-known "marriage problem". We are given a ternary relation $\Delta \subseteq B \times G \times H$, where each of B, G and H has the same number of elements, and we ask the question:

Is there a $\Delta_0 \subseteq \Delta$ such that each element of B (respectively G, H) is a first (resp. second, third) coordinate of exactly one triple in Δ_0?

Twierdzenie 11.13 *The problem* 3DM *is* NP*-complete.*

Dowód: Clearly, the problem is in NP. We prove that it is NP-hard, by a polynomial reduction from 3-CNF-SAT. Consider a formula

$$\varphi = (\alpha_{11} \vee \alpha_{12} \vee \alpha_{13}) \wedge (\alpha_{21} \vee \alpha_{22} \vee \alpha_{23}) \wedge \cdots \wedge (\alpha_{n1} \vee \alpha_{n2} \vee \alpha_{n3}).$$

We construct an instance of 3DM which is solvable if and only if the formula can be satisfied. We first define the H component of the product $B \times G \times H$. Let p be a propositional variable and assume that p occurs in φ exactly m' times *positively* (without negation) and exactly m'' times *negatively* (with negation). Let $m_p = \max\{m', m''\}$.

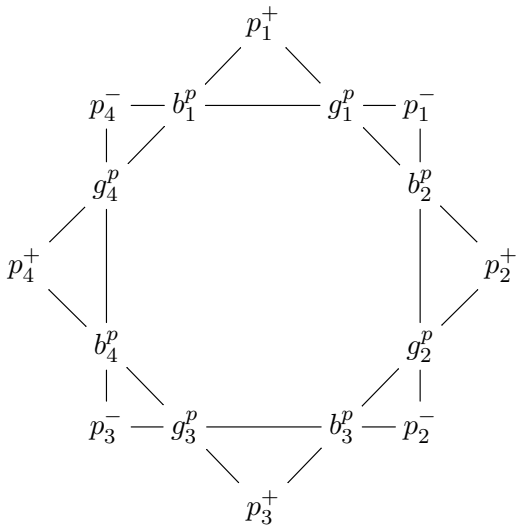
We define $H_p = \{p_i^+ \mid i = 1, \ldots, m_p\} \cup \{p_i^- \mid i = 1, \ldots, m_p\}$, and H is the union of H_p, for all p occurring in φ. The elements of H of the form p_i^+, for $i \leq m'$ correspond to the positive ocurrences of p in φ as certain α_{jk}. The elements of the form p_i^-, for $i \leq m''$ represent negative occurrences of p, of the form $\alpha_{jk} = \neg p$. We will identify p_i^+ and p_i^- with the corresponding α_{jk}.

Each of the sets B and G is a union of a family of sets to be defined. We begin with $B_p = \{b_1^p, \ldots, b_m^p\}$ and $G_p = \{g_1^p, \ldots, g_m^p\}$, where $m = m_p$ is chosen for p as above. Define $\Delta_p \subseteq B_p \times G_p \times H_p$ this way:

$$\Delta_p = \{\langle b_i^p, g_i^p, p_i^+\rangle \mid i = 1, \ldots m_p\} \cup \{\langle b_i^p, g_{i-1}^p, p_i^-\rangle \mid i = 2, \ldots m_p\} \cup \{\langle b_1^p, g_m^p, p_1^-\rangle\}.$$

The relation Δ_p, for $m = 4$, is illustrated by Figure 7.

Rysunek 7:



For each $i = 1, \ldots, n$ we also define a relation $\Delta^i = \{\langle b_i, g_i, \alpha_{ij}\rangle \mid j = 1, 2, 3\}$, where α_{ij} (which must be a propositional variable p or a negation $\neg p$) is understood as the element of H, and b_i, g_i are new.

Let B' be the union of all B_p and of the set $\{b_i \mid i = 1, \ldots n\}$ and similarly let G' be the union of all G_p and $\{g_i \mid i = 1, \ldots n\}$. Since B' and G' have fewer elements than H, we must define $B = B' \cup \{\beta^1, \ldots, \beta^r\}$ and $G = G' \cup \{\gamma^1, \ldots, \gamma^r\}$, for an appropriate r. Finally we define Δ as follows:

$$\Delta = \bigcup_p \Delta_p \cup \bigcup_{i=1}^n \Delta^i \cup \{\langle \beta^i, \gamma^i, h\rangle \mid i = 1, \ldots, r \text{ and } h \in H\}.$$

We want to prove that the formula φ is satisfiable iff the matching problem has a solution. Suppose first that $[\![\varphi]\!]_\rho = 1$ for some ρ. Then a matching $\Delta_0 \subseteq \Delta$ can be obtained as follows.

- From each Δ^i we choose a triple $\langle b_i, g_i, \alpha_{ij}\rangle$ satisfying $\rho(\alpha_{ij}) = 1$.
- From each Δ_p we choose all the triples containing:
  - p_i^-, if $\rho(p) = 1$;
  - p_1^+, if $\rho(p) = 0$.

  That is, now we only pick literals assigned the value 0 by the valuation ρ. It follows that we will never re-use any element of H chosen in the previous step.

- For each $i = 1, \ldots, r$ we choose any triple $\langle \beta^i, \gamma^i, h \rangle$ where h is any element of H that has not yet been used.

The reader can easily check that the above construction is correct, because every element is used exactly once.

Now we prove the converse. Suppose that $\Delta_0 \subseteq \Delta$ defines a matching. First observe that for every p there are only two possibilities. Either all triples containing p_i^- belong to Δ_p or all triples containing p_i^+ belong to Δ_p. Define $\rho(p) = 1$ in the first case, and $\rho(p) = 0$ in the second. Of course, for all i there must be something in the intersection $\Delta_0 \cap \Delta^i$ to include the elements b_i, g_i in the matching. But all literals α_{ij} with $\rho(\alpha_{ij}) = 0$ are already engaged, so if $\langle b_i, g_i, \alpha_{ij} \rangle \in \Delta_0$ then it must be the case that $\rho(\alpha_{ij}) = 1$. □

Plecak i inne

Jeszcze jeden znany problem NP-zupełny to *problem plecaka* (Knapsack). W uproszczeniu:

Dane są liczby $a_1, \ldots, a_m$, ograniczenie dolne L i ograniczenie górne U (rozmiar plecaka). Czy istnieje taki podciąg $a_{i_1}, \ldots, a_{i_\ell}$, że $L \leq a_{i_1} + \cdots + a_{i_\ell} \leq U$?

Twierdzenie 11.14 *The knapsack problem is* NP*-complete.*

Dowód: This one is easy. Given a partition problem $A = \{a_1, \ldots, a_k\}$ take

$$L = U = \frac{1}{2}(a_1 + \cdots + a_k)$$

and we have reduced EP to the knapsack problem. □

Oto kilka innych przykładów zadań NP-zupełnych:

- Problem kliki: *Dany graf $G = (V, E)$ i liczba k. Czy G ma k-elementowy podgraf pełny?*
- Problem cyklu Hamiltona: *Czy dany graf ma cykl Hamiltona?*
- Problem komiwojażera: *Dany graf pełny z nieujemnymi wagami krawędzi i stała M. Czy istnieje taki cykl Hamiltona w tym grafie, którego łączna waga nie przekracza M?*
- Problem pokrycia kwadratu: *Czy można pokryć kwadrat $n \times n$ za pomocą n rodzajów klocków domina (por. wniosek 7.10)?*
- Problem pakowania (*Bin Packing*): *Czy można podzielić dany zbiór $\{a_1, \ldots, a_k\}$ na ℓ części, każda o sumie nie przekraczającej s?*

Algorytmy pseudowielomianowe

Oto pewien algorytm rozwiązujący (uproszczony) problem plecaka. Z danych liczb $a_1, \ldots, a_m$ chcemy wybrać taki podciąg $a_{i_1}, \ldots, a_{i_\ell}$, aby suma $S = a_{i_1} + \cdots + a_{i_\ell}$, była możliwie największa,

ale nie przekraczała ustalonej stałej G. (Zdroworozsądkowo możemy zakładać, że wszystkie liczby a_i są mniejsze lub równe G, ale w sumie dają więcej niż G.) W tym celu nieco uogólnimy nasze zadanie. Niech $S(g,i)$ oznacza największą możliwą sumę postaci $a_{i_1} + \cdots + a_{i_\ell}$, gdzie $i_1, \ldots, i_\ell \leq i$, która jednak nie przekracza stałej g. Inaczej, $S(g,i)$ to najlepsze możliwe rozwiązanie dla ograniczenia górnego g, wybrane z liczb $a_1, \ldots, a_i$. Docelowo nas interesuje wartość $S(G,m)$. Ponieważ $S(0,i) = 0 = S(g,0)$ oraz:

$$S(g,i+1) = \begin{cases} S(g,i), & \text{jeśli } g < a_{i+1}; \\ \max\{S(g,i), S(g-a_{i+1},i) + a_{i+1}\}, & \text{w przeciwnym przypadku,} \end{cases}$$

więc wszystkie wartości $S(g,i)$ można obliczyć wypełniając kolejno, wiersz po wierszu, prostokątną tablicę rozmiaru $m \times G$. Niech $a = \max\{a_i \mid i \leq m\}$. Wtedy $G \leq m \cdot a$, więc czas działania takiej procedury jest wielomianem dwóch zmiennych m i a, bo równanie rekurencyjne (*) stosujemy $m \cdot G$ razy. Oczywiście nasz algorytm jest wielomianowy ze względu na wielkość liczb występujących w zadaniu, a nie ze względu na faktyczny rozmiar danych wejściowych n, bo a może być wykładnicze w stosunku do n: mamy $m + \log a \leq n \leq c \cdot m \log a$. W takiej sytuacji mówimy o algorytmie *pseudowielomianowym*.

12 Aproksymacje

Wiele problemów NP-zupełnych, np. problem plecaka albo pakowania, to decyzyjne wersje problemów minimalizacji lub maksymalizacji.

W problemie plecaka tak naprawdę chodzi o to, żeby znaleźć największą możliwą sumę nie przekraczającą danej stałej. A w przypadku pakowania chcielibyśmy wiedzieć ile pojemników wielkości s trzeba zamówić, żeby spakować przedmioty wielkości $a_1, \ldots, a_k$.

Znane algorytmy, które zawsze znajdują optymalne rozwiązania, są w przypadku takich problemów niestety wykładnicze. Ale czas potrzebny na znalezienie jakiegoś rozwiązania można znacznie poprawić, jeśli pogodzimy się z tym, że uzyskane rozwiązanie nie będzie optymalne.

Aproksymacja dla pakowania

Rozpatrzmy na przykład problem Bin Packing. Mamy przedmioty o wagach $a_1, \ldots, a_k$, które chcemy upakować do możliwie niewielkiej liczby wiader o danym udźwigu s. Dla wygody przyjmijmy, że $s = 1$; wtedy można zakładać, że $a_1, \ldots, a_k \leq 1$. Postępujemy najprościej: dla $i = 1, \ldots, k$, wkładamy przedmiot numer i do pierwszego wiadra, w którym się zmieści. Taki algorytm jest oczywiście wielomianowy, jego złożoność jest rzędu n^3. Rozwiązanie, które otrzymamy, raczej nie będzie optymalne, ale może nie będzie aż takie złe. Zauważmy bowiem, że przy takim postępowaniu, co najwyżej jedno wiadro jest wypełnione mniej niż w połowie. Użyjemy więc co najwyżej $\lceil 2A \rceil$ wiader, gdzie $A = a_1 + \cdots + a_k$. Ponieważ optymalne rozwiązanie nie może być mniejsze niż $\lceil A \rceil$, więc używamy w najgorszym razie dwa razy więcej wiader niż potrzeba.

Rozwiązania aproksymacyjne

Algorytmy aproksymacyjne stosują się do zadań, w których poszukujemy najmniejszej lub największej wartości liczbowej *opt* spełniającej dane warunki. Wersje decyzyjne takich zadań zwykle powstają poprzez wyznaczenie granicznej wartości, przy której rozwiązanie jest akceptowalne. W przypadku rozwiązania nieoptymalnego o wartości *rozw*, spełniającego warunek

$$\frac{|rozw - opt|}{\max\{rozw, opt\}} \leq \varepsilon,$$

mówimy o ε-aproksymacji. Przy tej definicji mamy zawsze $0 < \varepsilon < 1$, a jeśli przyjmiemy oznaczenie $k = \dfrac{1}{1-\varepsilon}$ (czyli $\varepsilon = 1 - \dfrac{1}{k}$), to zachodzi nierówność:[25]

- $rozw \leq k \cdot opt$, w przypadku zadania minimalizacji;
- $rozw \geq \dfrac{opt}{k}$, w przypadku zadania maksymalizacji.

A więc rozwiązanie przybliżone jest w najgorszym razie k razy większe lub k razy mniejsze od optymalnego. Dlatego na przykład rozwiązanie $\frac{1}{3}$-aproksymacyjne jest też nazywane rozwiązaniem $\frac{3}{2}$-aproksymacyjnym.[26] W naszym przykładzie uzyskaliśmy algorytm $\frac{1}{2}$-aproksymacyjny dla problemu pakowania. Czasami można uzyskać dużo lepsze współczynniki aproksymacji. Istnieją na przykład pseudowielomianowe algorytmy dla problemu plecaka. Możemy w zapisie binarnym liczb a_i skreślić k najmniej znaczących bitów i wykonać taki algorytm na tak „skróconych" (czyli 2^k razy mniejszych) danych. Wtedy czas pracy algorytmu będzie naprawdę wielomianowy, ale rozwiązanie nie będzie optymalne. W istocie można zrobić to w ten sposób, że otrzymany błąd będzie dowolnie mały.

Fakt 12.1 *Dla dowolnego $\varepsilon > 0$ istnieje wielomianowa ε-aproksymacja dla problemu plecaka.*

Mówi się, że *próg aproksymacji* dla plecaka jest zerowy. Jednak nie zawsze jest tak dobrze.

Fakt 12.2 *Jeśli dla pewnego $\varepsilon > 0$ istnieje wielomianowa ε-aproksymacja dla problemu komiwojażera, to* $\mathrm{P} = \mathrm{NP}$.

Dowód: Pokażemy, że ε-aproksymacja komiwojażera implikuje, że problem cyklu Hamiltona jest w klasie P. Krawędziom danego grafu G przypisujemy wagę 1. Jeśli G ma m wierzchołków, to pomiędzy wierzchołkami, które w G nie są połączone, dodajemy nowe krawędzie o wadze $m/(1-\varepsilon)$. W ten sposób otrzymujemy graf pełny, do którego stosujemy ε-aproksymację problemu komiwojażera. Jeśli w grafie G był cykl Hamiltona, to rozwiązanie optymalne ma wagę m, w przeciwnym razie musiała być użyta przynajmniej jedna „nowa" krawędź i rozwiązanie optymalne ma wagę większą od $m/(1-\varepsilon)$, czyli większą od m.

Algorytm aproksymacyjny w pierwszym przypadku musi dać wynik spełniajacy warunek $rozw \leq m/(1-\varepsilon)$. W drugim przypadku, wynik aproksymacji jest co najwyżej tak dobry jak rozwiązanie optymalne, a więc większy od $m/(1-\varepsilon)$. Aby ustalić czy w G jest cykl

[25] Zauważmy, że wtedy $k > 1$.

[26] Bywają też inne definicje. W przypadku minimalizacji, mówi się np. o β-aproksymacji, gdy $\beta \geq 1$ i iloraz $rozw/opt$ nie przekracza β.

Hamiltona, wystarczy więc sprawdzić czy rozwiązanie otrzymane przez aproksymację osiąga wartość $m/(1-\varepsilon)$. □

A zatem dla problemu komiwojażera próg aproksymacji jest 1, chyba że P=NP. (Podobnie jest w przypadku problemu kliki.) Zauważmy jednak, że komiwojażerowie często podróżują w przestrzeni euklidesowej.

Fakt 12.3 *Dla grafów spełniających nierówność trójkąta, problem komiwojażera ma wielomianową* 1/3*-aproksymację.*

Dowód: Dla danego grafu pełnego G z podanymi wagami krawędzi, postępujemy tak:

1. Znajdujemy minimalne drzewo T rozpinające graf G.[27]
2. Przez T_1 oznaczmy zbiór wierzchołków w T o nieparzystych stopniach (jest ich parzysta liczba). Dla wierzchołków z T_1 szukamy w grafie G skojarzenia pełnego S o minimalnej wadze.[28]
3. Drzewo T wraz z krawędziami skojarzenia S tworzy (multi)graf Eulera $T \oplus S$. Wybieramy w nim cykl Eulera C.
4. Cykl C to ciąg wierzchołków grafu G, być może z powtórzeniami. W tym cyklu „omijamy" powtarzające się wierzchołki „na skróty" tj. używając bezpośrednich połączeń grafu pełnego G. Otrzymujemy cykl D bez powtórzeń.

Załóżmy, że optymalny cykl komiwojażera w grafie G ma wagę *opt*. Jeśli pominiemy w tym cyklu wierzchołki o parzystych stopniach (nie należące do T_1) a wierzchołki z T_1 połączymy „na skróty", to otrzymamy cykl w G przechodzący przez wszystkie wierzchołki z T_1. Ponieważ mamy nierówność trójkąta, więc przejścia „na skróty" nie psują wagi i waga otrzymanego cyklu jest co najwyżej *opt*. Wybierając z niego co drugą krawędź, dostaniemy skojarzenie dla wierzchołków T_1. Można to zrobić na dwa sposoby; ten lepszy daje skojarzenie o wadze co najwyżej $\frac{1}{2}\,opt$.

A zatem waga skojarzenia S jest co najwyżej $\frac{1}{2}\,opt$. Waga drzewa T jest oczywiście nie większa od *opt*, bo po usunięciu dowolnej krawędzi otrzymuje się drzewo rozpinające. A zatem cykl Eulera C ma wagę co najwyżej $\frac{3}{2}\,opt$. Waga cyklu D, czyli nasze *rozw*, też nie jest gorsza (znowu nierówność trójkąta). Ostatecznie dostajemy

$$\frac{|rozw - opt|}{\max\{rozw, opt\}} = \frac{rozw - opt}{rozw} = 1 - \frac{opt}{rozw} \leq 1 - \frac{2}{3} = \frac{1}{3}$$ □

[27] Wybieramy zawsze najkrótszą krawędź z tych, które można dołączyć do drzewa.
[28] Wiadomo, że można to zrobić w czasie wielomianowym, ale algorytm jest trudny.

13 Klasa *co*-NP

Nie wiadomo, czy klasa NP jest zamknięta ze względu na dopełnienie. Przez *co*-NP oznaczamy klasę dopełnień języków z klasy NP:

$$co\text{-NP} = \{L \mid -L \in \text{NP}\}.$$

Fakt 13.1 *Jeśli L jest* NP*-zupełny, oraz $L \in$ co-NP, to* NP $=$ *co*-NP.
Dowód: Dla dowolnego $L' \in$ NP mamy $L' \leq_{\log} L$, więc także $-L' \leq_{\log} -L$. Stąd łatwo widzieć, że $-L' \in$ *co*-NP. □

Nie wiadomo także, czy klasa P jest właściwą podklasą iloczynu NP $\cap$ *co*-NP. Przykładem problemu należącego do tego iloczynu, dla którego nieznany jest algorytm wielomianowy, jest problem *gry z warunkiem parzystości (parity game)*.

Gra z warunkiem parzystości

Dany jest skończony graf skierowany, którego wierzchołki są podzielone na wierzchołki *Gracza* i wierzchołki *Oponenta*. Z każdego wierzchołka wychodzi co najmniej jedna krawędź. Ponadto z każdym wierzchołkiem związana jest liczba naturalna zwana jego *priorytetem*. Jeden z wierzchołków jest wskazany jako początkowy. Gra polega na przesuwaniu pionka z wierzchołka do wierzchołka zgodnie z kierunkiem krawędzi, a o wyborze krawędzi decyduje ten przeciwnik, na którego polu znajduje się pionek. Gra toczy się w nieskończoność, a więc niektóre wierzchołki muszą pojawić się nieskończenie wiele razy. Spośród nich wybiera się wierzchołek o najwyższym priorytecie. Jeśli ten priorytet jest liczbą parzystą, to wygrywa Gracz, w przeciwnym przypadku wygrywa Oponent.

Strategią Gracza (Oponenta) nazywamy funkcję, która każdemu wierzchołkowi Gracza (Oponenta) przypisuje jedną z krawędzi wychodzących z tego wierzchołka. Można udowodnić, że w grze z warunkiem parzystości jeden z przeciwników zawsze ma strategię wygrywającą, tj. taką strategię, która gwarantuje wygraną niezależnie od posunięć drugiego z graczy. Zadanie polega na tym, aby dla danej gry ustalić, czy Gracz ma w niej strategię wygrywającą.

Fakt 13.2 *Problem gry z warunkiem parzystości należy do iloczynu* NP $\cap$ *co*-NP.

Dowód: Jeśli istnieje wygrywająca strategia Gracza, to można ją odgadnąć i sprawdzić czy faktycznie jest wygrywająca. Sprawdzenie polega na usunięciu z gry krawędzi wychodzących z wierzchołków Gracza ale nie należących do odgadniętej strategii i ustaleniu, że w otrzymanym grafie nie ma pętli, osiągalnej z wierzchołka początkowego, na której najwyższy priorytet jest nieparzysty. A więc problem jest w klasie NP.

A jeśli nie istnieje wygrywająca strategia Gracza, to istnieje wygrywająca strategia Oponenta i też można ją odgadnąć. Zatem dopełnienie naszego zadania też jest w NP. □

Liczby pierwsze

Klasycznym przykładem języka należącego do iloczynu NP $\cap$ *co*-NP, był przez lata zbiór PRIMES, złożony z binarnych reprezentacji liczb pierwszych.

$$\text{PRIMES} = \{w \in \{0,1\}^* \mid w \text{ is a binary representation of a prime number}\}.$$

This language is obviously in *co*-NP (if a number is composite, one can guess the factorization and verify it fast). It is less obvious that it is also in NP. The latter is based on the following facts, which we mention without proof:

Twierdzenie 13.3 (Fermat, 1640)

1. *If a number r is not divisible by a prime number p then $r^{p-1} \mod p = 1$.*
2. *A number p is prime if and only if there exists a number r such that $1 < r < p$ and:*
   (a) $r^{p-1} \mod p = 1$;
   (b) $r^i \mod p \neq 1$, for all i such that $i < p-1$.
3. *Jeśli p jest pierwsze, to $\forall r{<}p\, \exists j{<}p\, \forall \ell{<}p\, (r^\ell \mod p = 1 \quad \Leftrightarrow \quad j \mid \ell)$*

On the basis of the above characterization one can design a nondeterministic recursive algorithm to check if a given p is prime.

Twierdzenie 13.4 (V. Pratt, 1975) PRIMES $\in$ NP.

Dowód: Warunek 2b twierdzenia 13.3 wystarczy sprawdzić dla liczb i postaci $\frac{p-1}{q}$, gdzie q jest dzielnikiem pierwszym liczby $p-1$. Ponieważ liczba $p-1$ ma co najwyżej $\log(p-1)$ dzielników pierwszych, warunek trzeba będzie sprawdzać tylko $\log(p-1)$ razy. Oczywiście najpierw trzeba odgadnąć rozkład liczby $p-1$ na czynniki. Potęgowanie liczb wymaga czasu proporcjonalnego do n^3, gdzie n jest rozmiarem danych wejściowych (w naszym przypadku liczba n jest $\mathcal{O}(\log(p-1))$, a więc czas działania algorytmu jest sumą czasu $c \cdot (\log(p-1))^4$ (zużytego na zgadywanie rozkładu i sprawdzanie warunku 2b) i czasu potrzebnego na rekurencyjne sprawdzenie pierwszości wszystkich odgadniętych dzielników liczby $p-1$. Pokażemy przez indukcję, ze ten czas nie przekracza $c \cdot (\log p)^5$. Niech $q_1, \ldots, q_k$ będą wszystkimi dzielnikami pierwszymi liczby $p-1$ (z możliwymi powtórzeniami). Z założenia indukcyjnego wynika, że łączny czas jest ograniczony przez

$$c \cdot (\log(p-1))^4 + \Sigma_{j=1}^k c \cdot (\log q_j)^5 = c \cdot ((\Sigma_{j=1}^k \log q_j)^4 + \Sigma_{j=1}^k (\log q_j)^5),$$

ponieważ $\log(p-1) = \Sigma_{j=1}^k \log q_j$. Powyższe wyrażenie można oszacować z góry przez

$$c \cdot (\Sigma_{j=1}^k \log q_j)^5 \leq c \cdot (\log p)^5.$$

Faktyczny czas może być krótszy, bo pierwszość każdego q_j sprawdzamy tylko raz. □

Później okazało się, że prawdziwe jest dużo silniejsze twierdzenie.

Twierdzenie 13.5 (Agrawal, Kayal, Saxena, 2002) PRIMES $\in$ P.

The proof of this theorem, contrary to what might have been expected, does not lead to a solution to the *factorization problem:*

Given a natural number, find all its prime divisors.

Protokół RSA (1977)

It is still believed that there is no feasible algorithm to solve the factorization problem. (At least no such algorithm is known despite many attempts to design it.) This belief is the basis for the *RSA public-key cryptosystem*, named for its authors: Rivest, Shamir and Adleman.

The idea of public-key cryptography is as follows. A person willing to receive encrypted messages chooses two large prime numbers p and q and computes:

- The product $p \cdot q$;
- The number $N = (p-1)(q-1)$;
- A number e relatively prime to N;
- A number $d < N$ such that $d \cdot e \mod N = 1$.

Then the pair of numbers $\langle p \cdot q, e \rangle$ is published as the receiver's public key. The other numbers are kept in secret. The correctness of the above definition follows from this lemma.

Lemat 13.6 *If e is relatively prime to N then there is exactly one number $d < N$ such that $d \cdot e \mod N = 1$.*

Dowód: First observe that each of the numbers $e \cdot 1, e \cdot 2, \ldots e \cdot (N-1)$ has a different remainder with respect to N. Indeed, otherwise we have $e(i-j) = k \cdot N$ for some i and j, and since N and e have no common prime divisor, it must be the case that N divides $i-j$. But $i - j < N$, a contradiction.

Since there are N possible remainders, exactly one of the numbers $e \cdot i$ has the remainder 1. □

Using the public code, anybody can send a message $x < p \cdot q$. The message is encoded as

$$y = x^e \mod pq,$$

and can be made public. The receiver uses his private key d to decode the message as

$$z = y^d \mod pq.$$

By the following lemma, the message x can be properly decoded.

Lemat 13.7 *If $y = x^e \mod pq$ and $z = y^d \mod pq$ then $z = x$.*

Dowód: Mamy $z = y^d \mod pq = x^{de} \mod pq = x^{kN+1} \mod pq$ dla pewnego k, ponieważ $d \cdot e \mod N = 1$. Aby udowodnić, że $z = x$, wystarczy wykazać, że z i x mają taką samą resztę z dzielenia przez pq. Ale reszta z dzielenia dowolnej liczby przez pq jest jednoznacznie wyznaczona przez jej reszty modulo p i modulo q. Zatem wystarczy pokazać, że $z \equiv x \mod p$ i $z \equiv x \mod q$.

Jeśli x nie dzieli się przez p, to z małego twierdzenia Fermata (twierdzenie 13.3(1)) mamy $x^{p-1} \mod p = 1$. Stąd $x^N = x^{(p-1)(q-1)} \equiv 1 \mod p$ i dalej $x^{kN+1} \equiv x \mod p$. Jeżeli zaś x jest podzielne przez p, to reszta z dzielenia x przez p jest zero, tak samo jak reszta z dzielenia liczby x^{kN+1} przez p. Podobne wyniki dostajemy przy dzieleniu przez q. □

Suppose now that a third party can factorize the publicly announced number $p \cdot q$ to find p and q. Then the numbers N and d can be computed from p, q, e, and the message can be decrypted. Thus, the above construction relies on the difficulty of factorization.

14 Algorytmy probabilistyczne

Rozważamy algorytmy (np. maszyny Turinga), w których decyzje co do następnego kroku podejmowane są (w pewnych stanach) losowo i badamy prawdopodobieństwo tego, że dane słowo zostanie zaakceptowane. Przypuśćmy, że mamy język L i taką „losową" maszynę $\mathcal{M}$, o takich własnościach:

- Maszyna $\mathcal{M}$ odrzuca każde słowo $w \notin L$.
- Jeśli $w \in L$, to $\mathcal{M}$ akceptuje w z prawdopodobieństwem co najmniej $\frac{1}{2}$.

Jeśli na dodatek maszyna $\mathcal{M}$ ma wielomianową złożoność czasową, to powiemy, że język L należy do klasy RP. Mówimy wtedy o algorytmie wielomianowym typu *Monte Carlo*. Zauważmy, że powtarzając wielokrotnie taki algorytm można ustalić czy $w \in L$, z dowolnie małym prawdopodobieństwem błędu (mniejszym niż prawdopodobieństwo wadliwego działania komputera). Zatem, z praktycznego punktu widzenia, algorytm typu Monte Carlo jest równie dobry jak wielomianowy algorytm deterministyczny.

Znane są probabilistyczne algorytmy rozstrzygające czy dana liczba jest pierwsza. Pierwszy taki algorytm wykorzystywał tzw. *test Millera-Rabina*. Jedna z wersji tego testu jest taka:

Fakt 14.1 (Rabin, 1976) *Liczba p jest pierwsza wtedy i tylko wtedy, gdy dla każdego $r < p$:*

1. $r^{p-1} \mod p = 1$;
2. *jeśli* $p - 1 = i \cdot 2^k$ *dla pewnych* k, i, *to* $\mathrm{NWD}(r^i - 1, p) \in \{1, p\}$.

Jeśli zaś liczba p jest złożona, to co najmniej połowa spośród liczb $r < p$ nie spełnia któregoś z powyższych warunków.[29]

Test Millera-Rabina (warunki 1,2) spełniają wszystkie liczby pierwsze, a liczby złożone spełniają go z prawdopodobieństwem co najwyżej $\frac{1}{2}$, które po odpowiedniej liczbie powtórzeń staje się zaniedbywalnie małe. Mamy więc:

Fakt 14.2 PRIMES $\in$ *co*-RP.

Dowód: Dla danego p algorytm probabilistyczny losuje liczbę $r < p$ i sprawdza warunki (1,2) lematu 14.1. □

W świetle twierdzenia 13.5 teoretyczne znaczenie powyższego faktu jest nieistotne, ważne jest jednak praktyczne znaczenie algorytmów probabilistycznych.

Klasę RP $\cap$ *co*-RP oznaczamy przez ZPP, i mówimy że języki $L \in$ ZPP mają wielomianowe algorytmy typu *Las Vegas*. Nazwa ZPP bierze się stąd, że wtedy prawdopodobieństwo błędu jest zerowe.

Fakt 14.3 *Jeśli $L \in$ ZPP, to istnieje losowa maszyna Turinga, której oczekiwany czas pracy jest wielomianowy i która akceptuje wszystkie słowa z L i odrzuca wszystkie słowa z $-L$.*

[29] Uwaga: Istnieją liczby złożone p spełniające warunek (1) dla wszystkich $r < p$, np. liczba $561 = 3 \cdot 11 \cdot 13$.

Dowód: Skoro $L \in$ ZPP, to mamy dwie maszyny losowe M_1 i M_2, działające w pamięci wielomianowej. Maszyna M_1 akceptuje wszystkie słowa z L, a słowa z $-L$ odrzuca z prawdopodobieństwem co najmniej 50%, maszyna M_2 zachowuje się odwrotnie. Słowo zaakceptowane przez M_1 i odrzucone przez M_2 na pewno więc należy do L, a słowo odrzucone przez M_1 i zaakceptowane przez M_2 należy do $-L$. Należy więc uruchomić obie maszyny. Jeśli rezultaty ich obliczeń są jednoznaczne, to mamy odpowiedź; jeśli nie, powtarzamy procedurę do skutku. Prawdopodobieństwo wątpliwości maleje wykładniczo, więc średni czas oczekiwania na konkluzję jest wielomianowy. □

Rozszerzeniem klasy RP $\cup$ *co*-RP jest klasa BPP. Język L należy do tej klasy, jeśli mamy maszynę losową $\mathcal{M}$, działającą w czasie wielomianowym z błędem co najwyżej 1/3:

- Jeśli $w \in L$, to $\mathcal{M}$ akceptuje w z prawdopodobieństwem co najmniej $\frac{2}{3}$;
- Jeśli $w \in -L$, to $\mathcal{M}$ odrzuca w z prawdopodobieństwem co najmniej $\frac{2}{3}$.

W tym przypadku prawdopodobieństwo błędu 1/3 jest też umowne (ważne, aby było mniejsze od 50%). Można to prawdopodobieństwo dowolnie zmniejszyć, wykonując obliczenie losowe wielokrotnie i podejmując końcową decyzję „większością głosów".

Fakt 14.4 *Jeśli $L \in$ BPP, to dla dowolnego ε istnieje algorytm wielomianowy rozpoznający L z prawdopodobieństwem błędu mniejszym od ε.*

Dowód: Wykonujemy test $2k$ razy dla dostatecznie dużego k i otrzymujemy ciąg odpowiedzi $S \in \{0,1\}^{2k}$, gdzie 1 oznacza akceptację a 0 odrzucenie. Przypuśćmy, że w ciągu S jest więcej odpowiedzi błędnych (b) niż poprawnych ($p < b$), tj. decyzja większością głosów będzie niepoprawna. Prawdopodobieństwo uzyskania każdego takiego ciągu S jest

$$P_S \leq (\tfrac{1}{3})^b (\tfrac{2}{3})^p \leq (\tfrac{1}{3})^k (\tfrac{2}{3})^k \leq (\tfrac{2}{9})^k,$$

zatem łączne prawdopodobieństwo uzyskania złego wyniku głosowania jest co najwyżej 2^{2k} razy większe (bo tyle jest wszystkich ciągów). Ale

$$2^{2k} \cdot (\tfrac{2}{9})^k = (\tfrac{8}{9})^k,$$

a to dla dostatecznie dużego k może być dowolnie małe. □

Jeszcze jedno alternatywne podejście do obliczeń losowych polega na rozważaniu zwykłych niedeterministycznych maszyn Turinga i badaniu proporcji liczby obliczeń akceptujących do obliczeń odrzucających (klasa PP).

Dowody interakcyjne

Jak już zauważyliśmy, klasę NP można scharakteryzować jako klasę tych języków, dla których istnieją „krótkie" (wielomianowe) certyfikaty. Mowa o tym we wniosku 11.4.

Podobnie w przypadku problemów z klasy RP można mówić o krótkich certyfikatach losowych. Taki, losowy lub nie, „certyfikat", to inaczej abstrakcyjnie rozumiany „dowód" przynależności

słowa do języka. Niezależnie od tego jak trudne jest naprawdę jego uzyskanie, pozwala on na łatwą weryfikację poprawności. Ten aspekt teorii złożoności obliczeniowej, dotyczący nie tyle algorytmicznych metod poszukiwania rozwiązań, co sposobów weryfikacji rozwiązań dostarczonych „z zewnątrz", jest przedmiotem intensywnych badań, na przykład ze względu na protokoły kryptograficzne.

Protokół interakcyjny jest uogólnieniem zarówno niedeterminizmu jak i losowości. Nieformalnie wyobrażamy go sobie tak, że Weryfikator (o ograniczonych możliwościach obliczeniowych) zadaje losowo wybrane pytania Proponentowi, i na podstawie jego odpowiedzi decyduje o tym, czy słowo wejściowe zostanie zaakceptowane, czy nie. Uściślimy to z pomocą takiej definicji:

Maszyna Turinga z wyrocznią A to maszyna wyposażona w dodatkową *taśmę zapytań*, na której może zostać zapisane dowolne słowo, oraz w *stan pytający* $q_?$ i stany *odpowiadające* q_{tak} i q_{nie}. W stanie pytającym działanie maszyny zależy od tego, czy słowo x, aktualnie znajdujące się na taśmie zapytań, należy do języka A. Jeśli tak, maszyna wchodzi w stan q_{tak}, w przeciwnym razie przyjmuje stan q_{nie}. Można oczywiście rozważać zachowanie tej samej maszyny $\mathcal{M}$ z różnymi wyroczniami A i używać oznaczenia $\mathcal{M}(A)$.

Przez *protokół interakcyjny* będziemy rozumieli ustaloną losową maszynę $\mathcal{M}(X)$ ze zmienną wyrocznią X, działającą w czasie wielomianowym. Powiemy, że język L jest w klasie IP, gdy istnieje dla niego protokół interakcyjny $\mathcal{M}(X)$, oraz wyrocznia A_L, o takich własnościach:

- Jeśli $w \in L$, to $\mathcal{M}(A_L)$ akceptuje w z prawdopodobieństwem co najmniej $\frac{2}{3}$;
- Jeśli $w \notin L$, to każda maszyna $\mathcal{M}(A)$ odrzuca w z prawdopodobieństwem co najmniej $\frac{2}{3}$.

Oczywiście zarówno NP $\subseteq$ IP (w NP nie ma losowości, a certyfikat odgrywa rolę wyroczni) jak też i BPP $\subseteq$ IP (w BPP nie ma wyroczni). Klasyczny przykład języka z klasy IP, to *problem nieizomorfizmu grafów*. Dane są dwa grafy (w postaci listy par wierzchołków), należy ustalić, że nie są one izomorficzne. Zauważmy, że jest to problem z klasy *co*-NP.

Fakt 14.5 *Problem nieizomorfizmu grafów jest w klasie* IP.

Dowód: Opiszemy protokół nieformalnie. Dane są grafy G i H. Weryfikator (czyli maszyna $\mathcal{M}$) wybiera losowo jeden z nich, dokonuje na nim losowej permutacji i pokazuje Proponentowi wynik tej permutacji. Proponent ma powiedzieć, czy pokazany mu graf jest izomorficzny z grafem G. Jeśli jest to permutacja grafu G, to prawidłową odpowiedzią jest *tak*; jeśli była to permutacja grafu H, to za prawidłową uważamy odpowiedź *nie*. Jeśli Proponent się „pomylił", to Weryfikator odrzuca parę $\langle G, H\rangle$, w przeciwnym razie powtarza procedurę pewną liczbę razy i akceptuje, jesli do „pomyłki" ani razu nie doszło.

Załóżmy, że grafy G i H nie są izomorficzne. Proponent potrafiący rozróżniać nieizomorficzne grafy odpowiada zawsze prawidłowo. Reprezentuje on „dobrą" wyrocznię A_L; maszyna $\mathcal{M}(A_L)$ z tą wyrocznią akceptuje słowa z języka L (pary nieizomorficznych grafów) z prawdopodobieństwem 1. Jeśli jednak słowo wejściowe w reprezentuje parę grafów izomorficznych, to prawdopodobieństwo tego, że Proponent odpowie prawidłowo jest zawsze $1/2$. Zatem już dwukrotna poprawna odpowiedź obniża prawdopodobieństwo błędu do $1/4$. □

Okazuje się, że dowody interakcyjne są możliwe dla szerokiej klasy problemów.

Twierdzenie 14.6 (Shamir) IP = PSPACE.

Dowód: Trudny dowód inkluzji $\supseteq$ opuścimy. Uzasadnienie inkluzji $\subseteq$ wymaga przerobienia protokołu $\mathcal{M}(X)$ na zwykłą maszynę Turinga działającą w pamięci wielomianowej. Maszyna ta oblicza prawdopodobieństwo akceptacji przy różnych możliwych przebiegach protokołu, żeby ustalić, czy istnieje wyrocznia, przy której szanse zaakceptowania słowa wejściowego osiągają próg $2/3$. Jeśli tak, to można słowo zaakceptować (bo będzie zaakceptowane przy dobrej wyroczni – a z założenia wiadomo, że dobra wyrocznia istnieje.) W przeciwnym razie nie pomoże żadna wyrocznia i słowo powinno zostać odrzucone. □

Klasa IP obejmuje problemy mające protokoły interakcyjne, w których zarówno liczba losowań jak i zapytań może być wielomianowa. Oba te parametry można oczywiście ograniczać w rozmaity sposób. Klasa $\mathrm{PCP}(f(n), g(n))$ to klasa tych zadań, dla których Weryfikator może wykonać $f(n)$ losowań, i zadać $g(n)$ pytań Proponentowi. Mamy bardzo ciekawe twierdzenie:

Twierdzenie 14.7 (Arora i inni) $\mathrm{NP} \subseteq \mathrm{PCP}(\log n, 1)$.

15 Klasa PSPACE

Formuły zdaniowe drugiego rzędu definiujemy przez indukcję tak:

- Zmienne zdaniowe $p, q, r, \ldots$ są formułami;
- Logical constants *true* and *false* are formulas;
- Jeśli α i β są formułami, to $\neg\alpha$, $\alpha \vee \beta$ i $\alpha \wedge \beta$ są formułami;
- Jeśli α jest formułą i p jest zmienną zdaniową, to $\forall p\, \alpha$ i $\exists p\, \alpha$ są formułami.

Wartość $[\![\varphi]\!]_\rho$ formuły φ przy danym wartościowaniu ρ definiuje się podobnie jak w klasycznym rachunku zdań, przyjmując dodatkowo, że

$$[\![\forall p\, \alpha]\!]_\rho = \max\{[\![\alpha[p := \mathit{true}]]\!]_\rho, [\![\alpha[p := \mathit{false}]]\!]_\rho\}$$
$$[\![\exists p\, \alpha]\!]_\rho = \min\{[\![\alpha[p := \mathit{true}]]\!]_\rho, [\![\alpha[p := \mathit{false}]]\!]_\rho\}$$

Tautologią (zdaniową drugiego rzędu) nazywamy formułę prawdziwą przy każdym wartościowaniu. Na przykład $\exists p(\neg q \vee p)$ jest tautologią.

The QBF problem is as follows:

Czy dana zdaniowa formuła drugiego rzędu jest tautologią?

In fact one can only consider *closed* formulas, where each variable is bound by a quantifier. In this case the above *validity* problem is equivalent to the satisfiability problem for QBF. A closed formula is either a tautology, or it is not satisfiable.

Twierdzenie 15.1 *Problem* QBF *jest* PSPACE*-zupełny.*

Dowód: Let $\mathcal{M}$ be a deterministic machine of polynomial space complexity $f(n)$, and let w be an input word. As in the proof of Theorem 11.2, we want to construct a (closed) formula which is valid iff $\mathcal{M}$ accepts w. This time, however, we cannot designate different variables for each computation step, because the number of steps can be exponential in $n = |w|$. We can however use part of the construction from the proof of Theorem 11.2.

Consider a vector of variables $\vec{q}$ consisting of:

- $p_{j,\ell,a}$, for $j \leq f(n)$, $\ell \leq t$, $a \in \Sigma$;
- q_s, for $s \leq k$;
- $r_{j,\ell}$, for $j \leq f(n)$, $\ell \leq t$;
- $r_{j,0}$, for $j \leq n$.

Such a vector *represents* a configuration $\mathcal{C}$ under a valuation ρ, when the following holds:

1. $\rho(q_s) = 1$ iff the internal state in the configuration $\mathcal{C}$ is s.
2. $\rho(p_{j,\ell,a}) = 1$ iff j-th symbol on the ℓ-th tape in the configuration $\mathcal{C}$ is a.
3. $\rho(r_{j,\ell}) = 1$ iff, in the configuration $\mathcal{C}$, the j-th cell is scanned on the ℓ-th work tape.
4. $\rho(r_{j,0}) = 1$ iff, in the configuration $\mathcal{C}$, the input head scans the j-th cell of the input.

One can now write a formula $C(\vec{q})$ with free propositional variables $\vec{q}$, such that

$$[\![C]\!]_\rho = 1 \text{ iff } \vec{q} \text{ represents some machine configuration under } \rho.$$

The technique applied to obtain this formula is the same as we used in the proof of Theorem 11.2. In a similar fashion we can also write formulas $A(\vec{q})$ and $I_w(\vec{q})$, such that

$$[\![A]\!]_\rho = 1 \text{ iff } \vec{q} \text{ represents an accepting configuration under } \rho.$$

$$[\![I_w]\!]_\rho = 1 \text{ iff } \vec{q} \text{ represents the initial configuration } \mathcal{C}_w \text{ under } \rho.$$

We can also write a formula $T(\vec{q}_1, \vec{q}_2)$, expressing a single computation step:

$$[\![T(\vec{q}_1, \vec{q}_2)]\!]_\rho = 1 \text{ iff } \vec{q} \text{ and } \vec{q}_2 \text{ represent respectively } \mathcal{C} \text{ and } \mathcal{C}' \text{ such that } \mathcal{C} \to_{\mathcal{M}} \mathcal{C}'.$$

Let the relation $\mathcal{C}_1 \to^i \mathcal{C}_2$ be defined (as in the proof of Savitch's theorem) by

$$\mathcal{C}_1 \to^i \mathcal{C}_2 \quad \text{iff} \quad \mathcal{C}_1 \twoheadrightarrow_{\mathcal{M}} \mathcal{C}_2 \text{ in at most } 2^i \text{ steps.}$$

If we can construct in polynomial time a formula $T_i(\vec{q}, \vec{q}_2)$ expressing this relation, i.e.,

$$[\![T_i(\vec{q}_1, \vec{q}_2)]\!]_\rho = 1 \text{ iff } \vec{q} \text{ and } \vec{q}_2 \text{ represent respectively } \mathcal{C} \text{ and } \mathcal{C}' \text{ such that } \mathcal{C} \to^i \mathcal{C}',$$

then we can express acceptance of the word w by the formula

$$\exists \vec{q}_1 \vec{q}_2 \, (I_w(q_1) \wedge T_{cf(n)}(\vec{q}_1, \vec{q}_2) \wedge A(\vec{q}_2)).$$

Of course, T_0 is just T. Assume that T_i has already been defined. If we take

$$\exists \vec{q}_3 \, (T_i(\vec{q}_1, \vec{q}_3) \wedge T_i(\vec{q}_3, \vec{q}_2)),$$

as a definition of T_{i+1}, this duplicates the size of the formula, and $T_{cp(n)}$ would be exponentially long. Instead, we apply a trick. In the formula below, the notation $\vec{q} \leftrightarrow \vec{p}$ for vectors of propositional variables stands for the conjunction of equivalences of the corresponding coordinates. Of course, we can express equivalence with help of $\wedge$, $\vee$ and $\neg$. Implication can be eliminated too, and is left here only to increase readability.

$$T_{i+1}(\vec{q}_1, \vec{q}_2) = \exists \vec{q}_3 \forall \vec{p}_1 \forall \vec{p}_2 (((\vec{p}_1 \leftrightarrow \vec{q}_1) \wedge (\vec{p}_2 \leftrightarrow \vec{q}_3)) \vee ((\vec{p}_1 \leftrightarrow \vec{q}_3) \wedge (\vec{p}_2 \leftrightarrow \vec{q}_2)) \rightarrow T_i(\vec{p}_1, \vec{p}_2))$$

Clearly, the size of $T_{cf(n)}$ is polynomial in n and can be constructed in log space. □

Another PSPACE-complete problem is CSL-*membership:*

Given a context-sensitive language L and a word w, does w belong to L?

Recall that a language is context-sensitive iff it is in NSPACE(n). The language L in the above problem is a part of the input and may be represented e.g. in the form of (a code of) a *linear bounded automaton*, which is a variant of Turing machine with work tape explicitly restricted to linear size. The size of the input is thus the length of w plus the length of the description of the machine.

Twierdzenie 15.2 *The* CSL-*membership problem is* PSPACE-*complete.*

Dowód: CSL-membership can be verified in nondeterministic space $n \log n$. A Turing machine can recognize CSL-membership by simulating the computation of a given machine $\mathcal{M}$ on a given word w. This simulation requires space $\mathcal{O}(|w| \cdot \log m)$, where m is the number of tape symbols of $\mathcal{M}$. This is proportional to $n \log n$, where n is the size the input.

It remains to show PSPACE-hardness. Let $L \in$ NSPACE($p(n)$), where $p(n) \geq n$ is a polynomial. To reduce L to CSL-membership, define $L_1 = \{w\$^{p(|w|)-|w|} \mid w \in L\}$, and observe that $L_1 \in$ NSPACE(n). Let $\mathcal{M}$ be a fixed machine recognizing L_1 in NSPACE(n). For any word w:

$$w \in L \quad \text{iff} \quad w\$^{p(|w|)-|w|} \in L_1 = L(\mathcal{M}).$$

Since $\mathcal{M}$ is fixed, the size of $\mathcal{M}$ is constant, and thus the reduction is in logarithmic space. □

Hierarchia wielomianowa i alternacja

Przypomnijmy, że *maszyna Turinga z wyrocznią* A to maszyna, która w dowolnym momencie może uzyskać „z zewnątrz" informację, czy dane słowo x należy do języka A. Przyjmujemy następujące oznaczenia:

$$\mathrm{P}^A = \{L \mid L \text{ jest akceptowany w czasie wielomianowym przez DTM z wyrocznią } A\};$$
$$\mathrm{NP}^A = \{L \mid L \text{ jest akceptowany w czasie wielomianowym przez NTM z wyrocznią } A\},$$

gdzie skróty DTM i NTM oznaczają oczywiście deterministyczną (odp. niedeterministyczną) maszynę Turinga. Dalej, dla dowolnej klasy języków $\mathcal{K}$, niech

$$\mathrm{P}^{\mathcal{K}} = \bigcup \{\mathrm{P}^A \mid A \in \mathrm{P}\};$$
$$\mathrm{NP}^{\mathcal{K}} = \bigcup \{\mathrm{P}^A \mid A \in \mathrm{NP}\}.$$

Przykład 15.3 (1) Niech EF oznacza następujący problem:

Dana formuła zdaniowa i liczba $k \in \mathbb{N}$. Czy istnieje równoważna formuła, zawierająca nie więcej niż k wystąpień literałów?

Problem EF należy do klasy $\mathrm{NP}^{\mathrm{NP}}$. Istotnie: niedeterministyczna maszyna może dla danej formuły φ odgadnąć poszukiwaną formułę ψ i zapytać wyroczni czy $\neg(\varphi \leftrightarrow \psi)$ jest spełnialne.

(2) Problem CNF-SAT należy do klasy P^{EF}. Maszyna deterministyczna najpierw sprawdza, czy dana formuła jest tautologią (por. ćwiczenie 94) a jeśli nie, pyta wyroczni, czy istnieje równoważna formuła bez wystąpień literałów. Taka formuła musiałaby być stałą logiczną $\bot$ (a więc byłaby niespełnialna).

(3) Z powyższego wynika, że $\mathrm{NP} \subseteq \mathrm{P}^{\mathrm{EF}}$.

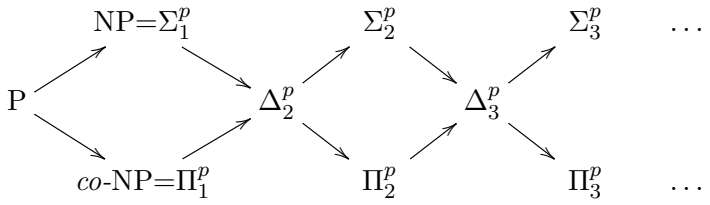
(4) Jeśli A jest językiem Pspace-zupełnym, to $\textsc{Pspace} \subseteq \mathrm{P}^A \subseteq \mathrm{NP}^A \subseteq \textsc{Pspace}$, a więc mamy $\mathrm{P}^A = \mathrm{NP}^A$. Istnieje też taki problem B, że $\mathrm{P}^B \neq \mathrm{NP}^B$.

Hierarchia wielomianowa

Dla dowolnej liczby $n \in \mathbb{N}$ definiujemy przez indukcję klasy języków Σ^p_n, Π^p_n i Δ^p_n. Przez analogię do hierarchiii arytmetycznej, mówimy, że klasy te tworzą *hierarchię wielomianową*, a ich sumę oznaczamy przez PH.

- $\Sigma^p_0 = \Pi^p_0 = \Delta^p_0 = \mathrm{P}$;
- $\Sigma^p_{n+1} = \mathrm{NP}^{\Sigma^p_n}$;
- $\Pi^p_{n+1} = \textit{co}\text{-}\Sigma^p_{n+1}$;
- $\Delta^p_{n+1} = \mathrm{P}^{\Sigma^p_n}$.

Łatwo widzieć, że $\Delta^p_1 = \mathrm{P}$, $\Sigma^p_1 = \mathrm{NP}$, $\Pi^p_1 = \textit{co}\text{-NP}$, $\Delta^p_2 = \mathrm{P}^{\mathrm{NP}}$, $\Sigma^p_2 = \mathrm{NP}^{\mathrm{NP}}$, itd. Oczywiste inkluzje pomiędzy klasami hierarchii wielomianowej przedstawiają strzałki na rysunku poniżej:



Nie wiadomo, czy którakolwiek z tych inkluzji jest ostra. Podobnie jak ta poniżej:

Fakt 15.4 $\mathrm{PH} \subseteq \textsc{Pspace}$.

Dowód: Przez indukcję ze względu na n dowodzimy, że $\Sigma^p_n, \Pi^p_n, \Delta^p_n \subseteq \textsc{Pspace}$. W kroku indukcyjnym zastępujemy wyrocznię obliczeniem o wielomianowej złożoności pamięciowej. □

Mówimy, że $m+1$-argumentowa R na słowach jest *wielomianowo zrównoważona*, gdy istnieje takie k, że dla dowolnej krotki $\langle w, x_1, x_2, \ldots, x_m\rangle \in R$ i dowolnego $i = 1, \ldots, m$ zachodzi

nierówność $|x_i| \leq |w|^k$. Podobieństwo hierarchii wielomianowej do hierarchii arytmetycznej wyraża się poprzez następujące uogólnienie wniosku 11.4.

Fakt 15.5 *Język L należy do klasy Σ_n^p wtedy i tylko wtedy, gdy istnieje wielomianowo zrównoważona relacja R, obliczalna w czasie wielomianowym i taka, że dla dowolnego słowa w:*

$$w \in L \qquad \text{wtedy i tylko wtedy, gdy} \qquad \exists x_1 \forall x_2 \exists x_3 \ldots Q_n x_n . \langle w, x_1, x_2, \ldots, x_n \rangle \in R,$$

gdzie kwantyfikatory są naprzemiennie postaci $\exists$ i $\forall$, a rodzaj Q_n zależy od parzystości n.

Dowód na razie opuszczamy. Zauważmy, że z powyższego faktu wynika dualna własność dla klas Π_n^p (z kwantyfikatorem ogólnym na początku). Jeszcze inną definicję hierarchii wielomianowej otrzymamy z pomocą pojecia *alternacji*.

Maszyny alternujące

Maszyna niedeterministyczna akceptuje w sposób *egzystencjalny*. Wystarczy, że istnieje obliczenie akceptujące. Pojęciem dualnym jest maszyna akceptująca na sposób *uniwersalny*, kiedy wszystkie obliczenia muszą być akceptujące. Maszyna *alternująca* może używać obu sposobów. It is defined as an ordinary Turing machine, but all its non-final states are classified as either *universal* or *existential*. As a consequence, the configurations of such a machine are also universal, existential, accepting or rejecting.

We say that an alternating machine $\mathcal{M}$ *accepts* a configuration $\mathcal{C}$, iff one of the following conditions hold:

- $\mathcal{C}$ is an accepting configuration;
- $\mathcal{C}$ is an existential configuration, and there is a $\mathcal{C}'$ such that $\mathcal{C} \to_{\mathcal{M}} \mathcal{C}'$ and $\mathcal{M}$ accepts $\mathcal{C}'$;
- $\mathcal{C}$ is a universal configuration, there is at least one $\mathcal{C}'$ such that $\mathcal{C} \to_{\mathcal{M}} \mathcal{C}'$ and $\mathcal{M}$ accepts every such $\mathcal{C}'$.

A word w is *accepted* by $\mathcal{M}$ (belongs to $L(\mathcal{M})$) iff $\mathcal{C}_w$ is accepted by $\mathcal{M}$.

Obliczeniem takiej maszyny jest drzewo konfiguracji. Jego korzeniem jest konfiguracja początkowa. Wierzchołki odpowiadające konfiguracjom egzystencjalnym mają tylko po jednym dziecku. Natomiast wierzchołki w których stan jest uniwersalny mają jako nastepniki *wszystkie* możliwe konfiguracje osiągalne w jednym kroku. A zatem w stanie egzystencjalnym maszyna jak zwykle „zgaduje" następny krok, a w stanie uniwersalnym musi „jednocześnie" sprawdzić wszystkie możliwości. (Sekwencyjna implementacja takiego zachowania wymaga rekursji.) Inna interpretacja alternacji jest taka: mamy do czynienia z grą, w której Gracz i jego Oponent mogą podejmować decyzje o przejściu do wybranego stanu. Gracz ma takie prawo w stanach egzystencjalnych, a Oponent w uniwersalnych. Każda gałąź obliczenia (które jest drzewem) odpowiada pojedynczej *partii* gry. Obliczenie (akceptujące) maszyny to strategia (wygrywająca) gracza.

An alternating machine is of *time complexity* $T(n)$ iff every sequence of configurations of the form $\mathcal{C}_w \to_{\mathcal{M}} \mathcal{C}_1 \to_{\mathcal{M}} \cdots \to_{\mathcal{M}} C_m$ is of length at most $T(|w|)$. It is of *space complexity* $S(n)$ iff $\mathcal{C}_w \twoheadrightarrow_{\mathcal{M}} \mathcal{C}'$ implies that $\mathcal{C}'$ uses at most $S(|w|)$ tape cells on each work tape. We define

$$\textsc{Atime}(T(n)) = \{L \mid L = L(\mathcal{M}), \text{ for some alternating Turing machine } \mathcal{M} \text{ of time complexity } T(n)\};$$

$$\textsc{Aspace}(S(n)) = \{L \mid L = L(\mathcal{M}), \text{ for some alternating Turing machine } \mathcal{M} \text{ of space complexity } S(n)\}.$$

Używamy też oznaczeń Alogspace, Aptime, itp.

Twierdzenie 15.6

1. Alogspace = Ptime;
2. Aptime = Pspace;
3. Apspace = Exptime.

Dowód: (1) Obliczanie wartości sieci boolowskiej to problem z klasy Alogspace (ćwiczenie 108), który jednocześnie jest P-zupełny (twierdzenie 10.8). Dlatego Ptime $\subseteq$ Alogspace. Dowód inkluzji przeciwnej opiera się (podobnie jak dowód twierdzenia 8.12(2)) na oszacowaniu z lematu 8.10. Przy logarytmicznej pamięci, mamy co najwyżej wielomianową liczbę konfiguracji w zbiorze $\mathbf{C}_w$, powiedzmy n^k. Możemy wszystkie te konfiguracje wypisać i kolejno oznaczać jako akceptujące lub odrzucające. Trzeba w tym celu przeglądać zbiór $\mathbf{C}_w$ co najwyżej n^k razy, za każdym razem wykonując wielomianową liczbę kroków.

(2) Inkluzja z prawej do lewej wynika stąd, że Pspace-zupełny problem QBF można rozwiązywać w czasie wielomianowym na maszynie alternującej (ćwiczenie 109). Inkluzja odwrotna wymaga symulacji maszyny alternującej $\mathcal{M}$ na maszynie deterministycznej, która przeszukuje w głąb drzewo obliczeń maszyny $\mathcal{M}$, zapamiętując na „stosie" historię obliczenia.

Część (3) pozostawiamy jako ćwiczenie 110. □

Podobne związki zachodzą dla innych klas: alternacja zawsze powoduje przejście w hierarchii „o jeden szczebel".

W każdej partii gry (gałęzi obliczenia) prawo wykonania kolejnego ruchu może przechodzić od Gracza do Oponenta i z powrotem. Liczba takich zmian (*alternacji*) jest w zasadzie dowolna, można jednak przyjąć ograniczenie na ich liczbę. Powiemy że obliczenie rozpoczynające się od stanu egzystencjalnego ma n *alternacji*, gdy w każdej jego gałęzi (w każdej rozgrywce) jest co najwyżej $n-1$ kroków zmieniajacych typ stanu z egzystencjalnego na uniwersalny lub odwrotnie. (Np. maszyna niedeterministyczna to maszyna o obliczeniach z 1 alternacją.) Mamy następującą charakteryzację hierarchii wielomianowej (dowód skwapliwie opuszczamy):

Fakt 15.7 *Język L należy do klasy Σ_n^p wtedy i tylko wtedy, gdy jest akceptowany w czasie wielomianowym przez pewną maszynę Turinga z n alternacjami i z egzystencjalnym stanem początkowym.*

Nie wiadomo, czy każdy język z klasy Aptime jest akceptowany przez maszynę o ograniczonej liczbie alternacji. Pytanie to jest równoważne problemowi PH $\stackrel{?}{=}$ Pspace.

16 Sieci boolowskie

Oprócz „klasycznych" podejść do teorii złożoności (badanie czasu i pamięci potrzebnych dla maszyny Turinga) rozważane są także inne modele, zwłaszcza probabilistyczne i równoległe. Do tych ostatnich należy pojęcie *sieci boolowskiej* czyli *obwodu logicznego*. O sieciach mowa była już przy okazji P-zupełności. Sieci logiczne mogą jednak same być używane do definiowania języków nad alfabetem $\{0,1\}$. Powiemy, że sieć o n wejściach *akceptuje* słowo $w = a_1 a_2 \dots a_n \in \{0,1\}^*$, gdy wynikiem działania sieci dla wartości wejściowych $a_1, a_2, \dots, a_n$ jest 1. W przykładzie na rysunku 6 mamy $n = 5$, a sieć akceptuje te słowa długości 5, w których nie występują dwie jedynki obok siebie.

Oczywiście sieć o n wejściach może rozpoznawać tylko słowa długości n. Aby mówić o rozpoznawaniu języka potrzebujemy nieskończonej rodziny sieci $C_0, C_1, C_2, \dots$ w której sieć C_n ma n wejść. Sieci logicznych można też użyć jako miary złożoności. Interesuje nas rozmiar sieci (liczba bramek) i jej wysokość (maksymalna długość drogi od wejścia do wyjścia), oczywiście jako funkcje zmiennej n. My rozważamy tylko takie rodziny sieci, których rozmiary są wielomianowe, tj. istnieje taki wielomian $p(n)$, że rozmiar każdej sieci C_n nie przekracza $p(n)$.

Jeśli istnieje taka wielomianowa rodzina sieci, że dla dowolnego n sieć C_n rozpoznaje dokładnie słowa ze zbioru $L_n = \{w \mid w \in L \wedge |w| = n\}$, to mówimy, że język L należy do klasy P/poly. Klasa P/poly tym różni się od pozostałych klas złożoności, że należą do niej nawet języki nierozstrzygalne. Odpowiada to „niejednostajnej" obliczalności, która może być pożyteczna, jeśli umiemy konstruować pewne konkretne sieci C_n dla dostatecznie dużych n. Ma więc sens np. pytanie czy NP $\subseteq$ P/poly. Niestety, odpowiedź jest jak zwykle nieznana. Co więcej, odpowiedź pozytywna jest mało prawdopodobna (dowód obecnie pomijamy):

Fakt 16.1 *Zawieranie* NP $\subseteq$ P/poly *implikuje kolaps hierarchii wielomianowej.*

Aspekt niejednostajności sieci można usunąć, jeśli założymy, że sieci są efektywnie konstruowalne – zwykle żąda się aby konstrukcja sieci C_n była możliwa w pamięci $\log n$. Mówimy wtedy o *jednostajnej* (*uniform*) rodzinie sieci. Odtąd mowa tylko o rodzinach jednostajnych.

Observe that, for a uniform family of circuits, the size is always polynomial. We have the following definitions:

$$\mathrm{AC}_k = \{L \mid L \text{ is recognized by circuits of depth at most } \log^k n\}$$
$$\mathrm{NC}_k = \{L \mid L \text{ is recognized by circuits with binary gates of depth at most } \log^k n\}$$

Klasa AC_k składa się z tych języków, które są rozpoznawane przez sieci o rozmiarze wielomianowym i wysokości ograniczonej przez $\log^k n$, dla pewnego k. Języki z klasy AC_0 są rozpoznawane przez wielomianowe sieci o stałej wysokości.

Bardziej ograniczone są klasy NC_k. Tutaj żądamy, aby każda bramka logiczna miała co najwyżej dwa argumenty. Sumę wszystkich AC_k (NC_k) oznaczamy przez AC (NC).

Przykład 16.2 *Problem osiągalności w grafie (por. twierdzenie 10.5) należy do klasy* AC_1.

Dowód: Graf to w istocie macierz binarna $m \times m$, gdzie m jest liczbą wierzchołków. Nietrudno zaprojektować sieć o $2m^2$ wejściach i m^2 wyjściach (lub też m^2 oddzielnych sieci) do

mnożenia dwóch takich macierzy. Używając mnożenia macierzy można dalej skonstruować sieć obliczającą domknięcie przechodnie danej relacji (reprezentowanej przez macierz). Istotnie, macierz relacji należy podnieść do potęgi m, co wymaga $\mathcal{O}(\log m)$ mnożeń; taka też będzie wysokość sieci, bo każde mnożenie wymaga tylko 2 poziomów. A zatem potrafimy skonstruować sieć sprawdzającą czy istnieje droga między dwoma wierzchołkami grafu. □

Twierdzenie 16.3

1. $\mathrm{NC}_k \subseteq \mathrm{AC}_k \subseteq \mathrm{NC}_{k+1}$, *dla każdego* k. *A zatem* $\mathrm{NC} = \mathrm{AC}$.
2. $\mathrm{NC}_1 \subseteq$ Logspace.
3. Nlogspace $\subseteq \mathrm{AC}_1$.
4. $\mathrm{NC} \subseteq \mathrm{P}$.

Dowód: The first part is very easy: A gate with a polynomial number of inputs can be "unfolded" to a tree of binary gates of logarithmic height. The depth of a circuit is then multiplied by $\mathcal{O}(\log n)$.

Podobnie łatwa jest część ostatnia: konstrukcja i ewaluacja sieci o wielomianowym rozmiarze jest możliwa w wielomianowym czasie. Jeśli na dodatek sieć jest logarytmicznej wysokości i ma tylko binarne bramki, to można jej wartość obliczać z pomocą rekurencyjnej procedury („oblicz wartości obu argumentów i stąd uzyskaj wartość danej bramki") której implementacja wymaga binarnego stosu o logarytmicznej głębokości. Stąd otrzymujemy część (2).

Część (3) jest uogólnieniem przykładu 16.2. Należy zastosować tę samą metodę do grafu przejść między konfiguracjami maszyny Turinga (por. dowód twierdzenia 10.5). □

Hierarchia, o której mowa w twierdzeniu 16.3(1) jest nietrywialna. Z następującego sławnego (i trudnego) twierdzenia wynika bowiem, że $\mathrm{AC}_0 \neq \mathrm{NC}_1$.

Twierdzenie 16.4 (Furst, Saxe, Sipser, 1984) *Następujące języki*

- PARITY $= \{w \in \{0,1\}^* \mid$ w słowie w jest parzysta liczba jedynek$\}$;
- MAJORITY $= \{w \in \{0,1\}^* \mid$ w słowie w jest więcej jedynek niż zer$\}$,

nie należą do klasy AC_0.

Ćwiczenia

1. Skonstruować automaty skończone akceptujące języki:
   (a) $(ab)^*(b^* \cup a)$;
   (b) $\{w \in \{0,1\} \mid$ słowo w nie zawiera podsłowa $1001\}$.

R 2. Niech język $L \subseteq \{0,1\}$ składa się ze wszystkich słów, w ktorych podsłowo 1011 występuje tylko na samym końcu albo wcale. Skonstruować deterministyczny automat skończony akceptujący ten język i wyrażenie regularne, które go definiuje.

3. Znaleźć język akceptowany przez automat skończony $\mathcal{M} = \langle Q, \mathcal{A}, \delta, q_0, F\rangle$, gdzie $Q = \{0,1,2\}$, $F = \{2\}$, $q_0 = 0$, a relacja δ składa się z trójek:
$$\langle 0,a,1\rangle, \langle 1,a,1\rangle, \langle 1,b,1\rangle, \langle 1,b,2\rangle, \langle 2,b,2\rangle, \langle 2,a,0\rangle.$$

4. Skonstruować możliwie najprostsze niedeterministyczne automaty skończone rozpoznające:
   (a) język $\{vaw \in \{a,b\}^* \mid \#(b,w)$ jest podzielne przez $3\}$;
   (b) język składający się ze wszystkich słów nad alfabetem $\{a,b\}$, w których występuje podsłowo *baba* lub *abba*,

   oraz możliwie najprostsze automaty deterministyczne rozpoznające te języki.

5.* Automat z ε-przejściami definiuje się tak samo jak niedeterministyczny automat skończony, ale relacja przejścia jest podzbiorem zbioru $(Q \times \mathcal{A} \cup \{\varepsilon\}) \times Q$. Automat akceptuje słowo w, jeżeli $w = a_1 \dots a_n$, dla pewnych $a_1, \dots, a_n \in \mathcal{A} \cup \{\varepsilon\}$, oraz istnieje ciąg przejść:
$$q_0 = s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \cdots \xrightarrow{a_{n-1}} s_{n-1} \xrightarrow{a_n} s_n,$$
gdzie s_n jest stanem końcowym. Udowodnić, że każdy taki automat akceptuje język regularny.

6. Rozpatrzmy następujące języki:
   - $L_1 = \{(abb)^k \mid k \in \mathbb{N}\} - \{w \mid \#(a,w)$ jest parzyste$\}$;
   - $L_2 = L(G)$, gdzie gramatyka G ma symbol początkowy ξ_0 oraz następujące produkcje:
     $\xi_0 \Rightarrow a\xi_0$, $\xi_0 \Rightarrow b\xi_2$;
     $\xi_2 \Rightarrow b\xi_0$, $\xi_2 \Rightarrow a\xi_3$, $\xi_2 \Rightarrow b\xi_2$;
     $\xi_3 \Rightarrow a\xi_0$, $\xi_2 \Rightarrow b\xi_0$, $\xi_3 \Rightarrow \varepsilon$.

   Dla każdego z tych języków, proszę:
   (a) nie odwołując się do automatu, uzasadnić, że jest to język regularny;
   (b) skonstruować deterministyczny automat skończony akceptujący ten język.

7. Udowodnić, że następujące języki nie są regularne:
   (a) $\{w \in \{a,b\}^* \mid$ liczba wystąpień a i b w słowie w jest taka sama$\}$;
   (b) $\{ww \mid w \in \{a,b\}^*\}$;
   (c) $\{ww^R \mid w \in \{a,b\}^*\}$.

8. Opisać automaty akceptujące języki:
   (a) $L_4 = \{w \in \{a,b\}^* \mid$ w słowie w nie występują segmenty postaci *abba* ani *baba*$\}$;
   (b) $L_5 = \{w \in \{a,b\}^* \mid$ liczba wystąpień segmentu *abba* w słowie w jest taka sama jak liczba wystąpień segmentu *baba*.$\}$

9. Udowodnić, że języki $\{ww \mid w \in \{a,b\}^*\}$ oraz $\{a^n b^n c^n \mid n \in \mathbb{N}\}$ nie są bezkontekstowe.

R 10. Rozważamy słowa nad alfabetem $\{a,b\}$. Słowo jest *zrównoważone* wtedy i tylko wtedy, gdy ma tyle samo liter a co liter b. Które z języków L_1, L_2, L_3 są regularne? Które są bezkontekstowe?
   (a) Język L_1 składa się ze słów postaci $w = w_1 \dots w_k$, gdzie $k > 0$, a każde słowo w_i jest zrównoważone i ma długość 4.
   (b) Język L_2 składa się ze słów postaci $w = w_1 w_2 w_3 w_4$, gdzie każde z czterech słów w_1, w_2, w_3, w_4 jest zrównoważone i niepuste.

(c) Język L_3 składa się ze słów postaci $w = w_1w_2w_3w_4$, gdzie każde z czterech słów w_1, w_2, w_3, w_4 jest zrównoważone i każde z nich ma tę samą długość.

R 11. Niech L_1, L_2, L_3 będą jak w zadaniu 10. Dla jakich funkcji S_1, S_2, S_3 zachodzi $L_1 \in \textsc{Dspace}(S_1(n))$, $L_2 \in \textsc{Dspace}(S_2(n))$, $L_3 \in \textsc{Dspace}(S_3(n))$?

12. Które z następujących języków są regularne? Które są bezkontekstowe?

R (a) $L_a = \{w \in \{a,b\}^* \mid \#(a,w) = 2 \cdot \#(b,w) + 1\}$;[30]
R (b) $L_b = \{a^n b^m a^m b^n \mid n, m \in \mathbb{N}\}$;
(c) $L_c = \{a^n b a^k b a^m \mid n \cdot m = k\}$;
R (d) $L_d = \{w \in \{a,b\}^* \mid \#(a,w) = \#(b,w)^2 + 1\}$;
(e) $L_e = \{w \in \{a,b\}^* \mid \#(a,w) = \#(b,w)\}$;
R (f) $L_f = \{vwv^R \mid w, v \in \{a,b\}^*\}$;
(g) $L_g = \{vwv^Rw^R \mid w, v \in \{a,b,c\}^*\}$;
R (h) $L_h = \{a^m b^n c^k d^l \mid m + k \equiv n + l \textbf{ mod } 3\}$;
R (i) $L_i = \{a^m b^n c^k d^l \mid m + k \leq n + l\}$;
R (j) $L_j = \{a^m b^n c^m d^l \mid 2m \geq n + l\}$;
(k) $L_k = \{a^m b^n c^k d^l \mid m \cdot k \leq n \cdot l\}$;
R (l) $L_l = \{a^m b^n c^k d^l \mid m + l \not\equiv n + k \textbf{ mod } 3\}$;
R (m) $L_m = \{a^m b^n c^k d^l \mid m + l \geq n + k\}$;
R (n) $L_n = \{a^m b^n c^k d^m \mid 2m \geq n + k\}$;
(o) $L_o = \{a^m b^n c^k d^l \mid m \cdot l \geq n \cdot k\}$;
(p) $L_p = \{a,b\}^* - \{wbabav \mid w, v \in \{a,b\}^*\}$
(zbiór tych słów, w których nie występuje podsłowo *baba*);
(q) $L_q = \{a^n b^m a^k \mid n = m \text{ lub } m = k\}$;
(r) $L_r = \{a^n b^m a^k \mid n = m \text{ oraz } m = k\}$;
R (s) $L_s = \{(a^2b^3)^k \mid k \in \mathbb{N}\}$;
R (t) $L_t = \{a^{2k} b^{3k} \mid k \in \mathbb{N}\}$;
R (u) $L_u = \{a^{2^k} b^{3^k} \mid k \in \mathbb{N}\}$.

R 13. Który z języków wymienionych w zadaniu 12 jest (a) kontekstowy? (b) w klasie Logspace? (c) w klasie *co*-NP? (d) rekurencyjnie przeliczalny? (e) nierozstrzygalny?

14. Które z następujących języków są regularne? Które są bezkontekstowe?

R (a) $L_1 = \{a^m b^n a^k \mid m \equiv n + k \mod 2\}$;
R (b) $L_2 = \{a^m b^n a^k \mid m = n + k\}$;
R (c) $L_3 = \{a^m b^n a^k \mid m = n + k \wedge n = k\}$;
(d) $L_4 = \{a^m b^n a^\ell b^k \mid \text{liczba } k + m - n - \ell \text{ jest parzysta}\}$;
(e) $L_5 = \{a^m b^n a^\ell b^k \mid k + m = n + \ell\}$;
R (f) $L_6 = \{a^m b^n a^m \mid m, n \in \mathbb{N}\}$;
R (g) $L_7 = \{w \in \{a,b\}^* \mid \exists k \in \mathbb{Z}\, (\#(a,w) - \#(b,w) = 3k)\}$;
R (h) $L_8 = \{w \in \{a,b\}^* \mid \#(a,w) = 3 \cdot \#(b,w)\}$;
R (i) $L_9 = \{w \in \{a,b\}^* \mid \#(a,w) = 3^{\#(b,w)}\}$;
R (j) $L_{10} = \{a^i (ba)^j b^k \mid i, j, k \in \mathbb{N}\}$;
R (k) $L_{11} = \{a^i (ba)^j b^i \mid i, j \in \mathbb{N}\}$;
R (l) $L_{12} = \{a^i (ba)^i b^i \mid i \in \mathbb{N}\}$;
R (m) $L_{13} = \{a^n bw \mid n \in \mathbb{N} \wedge |w| = n\}$;
R (n) $L_{14} = \{a^n bw \mid n \in \mathbb{N} \wedge |w| = 4\}$;
R (o) $L_{15} = \{a^n bw \mid n \in \mathbb{N} \wedge |w| = n^2\}$;
R (p) $L_{16} = \{a^m b^n \mid m, n \in \mathbb{N}\}$;
R (q) $L_{17} = \{a^n b^m a^n b^m \mid n, m \in \mathbb{N}\}$.

R 15. Czy wszystkie języki z zadania 14 należą do klasy Logspace? A do klasy P?

[30] Przez $\#(x,w)$ oznaczamy liczbę wystąpień litery x w słowie w.

R 16. Które z następujących języków są regularne? Które są bezkontekstowe?

(a) $A = \{w\$v \mid w, v \in \{0,1\}^* \wedge \#(1,w) \bmod 3 = \#(1,v) \bmod 3\}$;
(b) $B = \{w\$v \mid w, v \in \{0,1\}^* \wedge \#(1,w) = \#(1,v)\}$;
(c) $C = \{w\$v \mid w, v \in \{0,1\}^* \wedge \#(1,w)^2 = \#(1,v)\}$.

R 17. Które z następujących języków są regularne? Które są bezkontekstowe?

(a) $X = \{a^n b^m c^k \mid n + k = m\}$;
(b) $Y = \{a^n b^m c^k \mid n \cdot k = m\}$;
(c) $Z = \{a^n b^m c^k \mid n \cdot k = 0\}$.

R 18. Jeśli $w \in \{a,b\}^*$, to przez $\overline{w}$ oznaczymy słowo powstające przez zamianę wszystkich liter a na b i wszystkich b na a. Na przykład jeśli $w = ababb$, to $\overline{w} = babaa$. Które z następujących języków są regularne, a które są bezkontekstowe?

(a) $L_1 = \{w^R \overline{w} \mid w \in \{a,b\}^*\}$;
(b) $L_2 = \{w\overline{w} \mid w \in \{a,b\}^*\}$;

R 19. Niech L_1 będzie jak w zadaniu 18a. Udowodnić, że $L_1 = \{w \mid \overline{w} = w^R\}$.

R 20. Czy języki L_1 i L_2 z zadania 18 są rozstrzygalne? Do jakich należą klas złożoności? Czy to możliwe, że któryś z nich jest P-zupełny? NP-zupełny? PSPACE-zupełny? Co by z tego wynikało?

21. Udowodnić, że jeśli L_1 i L_2 są językami bezkontekstowymi, to także $L_1 \cup L_2$, $L_1 \cdot L_2$ oraz L_1^* są językami bezkontekstowymi.

R 22. Narysować drzewo wywodu słowa *aabbaabaa* w gramatyce:

$$\xi_0 ::= a\eta\xi_0 \mid a \qquad \eta ::= \xi_0 b\eta \mid \xi_0\xi_0 \mid ba\,.$$

23. Narysować dwa różne drzewa wywodu słowa $a{+}b \cdot a$ w gramatyce:

$$\xi_0 ::= \xi_0{+}\xi_0 \mid \xi_0 \cdot \xi_0 \mid a \mid b\,.$$

24. Czy jedno słowo może mieć nieskończenie wiele drzew wywodu?

25.* Udowodnić, że dowolne przeplatanie słowa "()" daje w wyniku poprawne wyrażenie nawiasowe.

26. Rozpatrzmy następującą rekurencyjną definicję *wyrażenia*:

- *Wyrażenie* to suma skończenie wielu jednomianów.
- *Jednomian* to albo stała, albo iloczyn dwóch lub więcej czynników.
- *Czynnik* to stała, lub ujęta w nawiasy suma dwóch lub więcej wyrażeń.
- *Stała* to 0 lub 1.

Napisać gramatyki generujące: (a) wszystkie wyrażenia, (b) wyrażenia o wartości zero.

R 27. Rozpatrzmy następującą rekurencyjną definicję *wyrażenia*:

- *Wyrażenie* to jednomian albo suma dwóch lub więcej jednomianów.
- *Jednomian* to czynnik albo iloczyn dwóch lub więcej czynników.
- *Czynnik* to stała, lub suma dwóch lub więcej jednomianów ujęta w nawiasy.
- *Stała* to $0, 1$ lub 3.

Napisać gramatykę generującą wszystkie wyrażenia o wartościach nieparzystych.

R 28. Rozpatrzmy następującą rekurencyjną definicję *wyrażenia*:

- *Wyrażenie* to jednomian albo suma dwóch lub więcej jednomianów.
- *Jednomian* to albo stała, albo iloczyn dwóch lub więcej czynników.
- *Czynnik* to stała, lub wyrażenie ujęte w nawiasy.
- *Stała* to $0, 1$ lub 2.

Napisać gramatykę generującą wszystkie wyrażenia arytmetyczne o wartościach parzystych.

R 29. Napisać gramatykę bezkontekstową generującą wszystkie całkowicie nawiasowane wyrażenia arytmetyczne o wartości 3, zbudowane z pomocą symboli dodawania i mnożenia oraz stałych 0 i 1.

30. Napisać gramatykę bezkontekstową generującą wszystkie poprawnie zbudowane wyrażenia arytmetyczne nie zawierające zbędnych nawiasów. W wyrażeniach mogą występować symbole dodawania i mnożenia oraz dwie zmienne x i y.

31. Niech v będzie takim wartościowaniem zdaniowym, że $v(p) = v(q) = 1$ oraz $v(r) = 0$. Napisać gramatykę bezkontekstową, generującą wszystkie schematy zdaniowe, które są spełnione przy wartościowaniu v, i w których nie występują inne zmienne niż p, q, r.

32. Skonstruować automat ze stosem akceptujący język $\{a^k b a^m b a^\ell \mid m = k+\ell\}$ i gramatykę bezkontekstową generującą ten język.

33. Skonstruować automat ze stosem akceptujący język $\{w \in \{a,b\}^* \mid \#(w,a) = 2 \cdot \#(w,b)\}$, gdzie symbol $\#(w,x)$ oznacza liczbę wystąpień litery x w słowie w. Skonstruować gramatykę bezkontekstową generującą ten sam język.

34. Zdefiniować gramatykę bezkontekstową generującą dopełnienie języka $\{ww \mid w \in \{a,b\}^*\}$.

35.* Pokazać, że język $\{ww^R \mid w \in \{a,b\}^*\}$ nie jest deterministyczny. *Wskazówka: udowodnić, że wtedy język* $\{1^n 0^m 0^m 1^n 1^n 0^m 0^m 1^n \mid m,n \in \mathbb{N}\}$ *byłby bezkontekstowy, a nie jest.*

R 36. Udowodnić, że język $\{a^n b^m a^n b^n \mid n,m \in \mathbb{N}\}$ nie jest bezkontekstowy, ale jest kontekstowy. Skonstruować gramatykę typu zero generującą ten język.

37. Skonstruować gramatykę typu zero generującą język $\{a^{2^n} \mid n \in \mathbb{N}\}$.

38. Opisać maszynę Turinga akceptującą język $\{a^n b^n c^n \mid n \in \mathbb{N}\}$.

39. Opisać algorytmy rozwiązujące następujące zadania:
    (a) Dana gramatyka bezkontekstowa G i słowo w. Czy $w \in L(G)$?
    (b) Dana gramatyka bezkontekstowa G. Czy $L(G) = \varnothing$?
    (c) Dana gramatyka bezkontekstowa G. Czy $L(G)$ jest nieskończony?

40. Udowodnić, że następujący problem jest nierozstrzygalny:
    Czy dana maszyna Turinga akceptuje słowo puste?

R 41. Który z następujących problemów jest rozstrzygalny, a który nierozstrzygalny?
    (a) Dana maszyna Turinga M, czy język $L(M)$ jest częściowo obliczalny?
    (b) Dana maszyna Turinga M, czy $L(M) = (aabbb)^*$?

42. Który z następujących problemów jest (a) rozstrzygalny (b) nierozstrzygalny?
    P1: Dana maszyna Turinga M, czy $L(M)^* \subseteq L(M)$?
    P2: Dana maszyna Turinga M, czy $L(M)^* \supseteq L(M)$?

R 43. Które z następujących problemów są rozstrzygalne, a które nie? W przypadku pozytywnej odpowiedzi: do jakiej klasy złożoności należy ten problem?
    (a) Dana maszyna Turinga M, czy $L(M) \subseteq a^*$?
    (b) Dany automat skończony A, czy $L(A) \subseteq a^*$?

R 44. Rozpatrzmy następujący problem decyzyjny:
    Czy dana deterministyczna maszyna Turinga akceptuje wszystkie słowa długości 28?
    (a) Czy ten problem jest rozstrzygalny?
    (b) Czy ten problem jest częściowo rozstrzygalny?
    (c) Czy ten problem jest NP-zupełny?

R 45. Rozpatrzmy następujące problemy decyzyjne:

(a) Czy dana gramatyka bezkontekstowa generuje jakieś słowo złożone z samych liter a?

(b) Czy dana maszyna Turinga akceptuje jakieś słowo złożone z samych liter a?

Który z tych problemów jest rozstrzygalny? Który jest częściowo rozstrzygalny? Czy to możliwe, że któryś z nich jest NP-zupełny?

46. Udowodnić, że dla dowolnego zestawu kafelków równoważne są warunki:
(a) Istnieje pokrycie całej płaszczyzny;
(b) Dla każdego n, istnieje pokrycie kwadratu o boku n.

R 47. Zbiór $A \subseteq \mathbb{N}$ jest obliczalny (rekurencyjny), funkcja $f : \mathbb{N} \to \mathbb{N}$ jest też obliczalna. Czy obraz $f(A)$ musi być obliczalny? Czy musi być rekurencyjnie przeliczalny? Co się zmieni jeśli zamiast o obraz zapytamy o przeciwobraz $f^{-1}(A)$?

48. Udowodnić, że zbiór $Z \subseteq A^*$ jest rekurencyjnie przeliczalny wtedy i tylko wtedy gdy istnieje taki obliczalny zbiór $R \subseteq (A \cup \{\$\})^*$, że dla dowolnego $w \in A^*$:

$$w \in Z \text{ wtedy i tylko wtedy gdy } \exists v (w\$v \in R)\,.$$

49.* Czy istnieje taka liczba n, że **while**-programy z n zmiennymi obliczają wszystkie jednoargumentowe funkcje częściowo rekurencyjne? Jaka jest najmniejsza taka liczba? (Można też używać instrukcji **go to**.)

50. Załóżmy, że f_1 jest $\mathcal{O}(g_1)$ i że f_2 jest $\mathcal{O}(g_2)$. Czy funkcje $f_1 + f_2$, $f_1 \cdot f_2$, 2^{f_1} są, odpowiednio, $\mathcal{O}(g_1 + g_2)$, $\mathcal{O}(g_1 \cdot g_2)$, $\mathcal{O}(2^{g_1})$?

51. Dla prawie wszystkich $n \in \mathbb{N}$ zachodzi $F(n) \leq c \cdot G(n)$, gdzie $c > 0$. Czy funkcja F jest $\mathcal{O}(G)$?

R 52. Czy język $L = \{ww^R w \mid w \in \{a, b\}^*\}$ jest bezkontekstowy? Czy należy do klasy P?

R 53. Które z następujących języków są regularne? Które są bezkontekstowe, a które są nierozstrzygalne? Do jakich klas złożoności należą te języki?

(a) $L_1 = \{a^m b^n a^k b^\ell \mid m, n, k, \ell \in \mathbb{N}\}$;

(b) $L_2 = \{a^m b^n a^k b^\ell \mid m, n, k, \ell \in \mathbb{N} \wedge m \leq \ell \wedge n \leq k\}$;

(c) $L_3 = \{a^m b^n a^k b^\ell \mid m, n, k, \ell \in \mathbb{N} \wedge m \leq k \wedge n \leq \ell\}$.

54. Który z następujących języków jest (a) regularny (b) bezkontekstowy (c) w klasie LOG (d) w klasie NP (e) PSPACE-zupełny?

$$L_1 = \{a^m b^k b^m a^k \mid m, k \in \mathbb{N}\} \qquad L_2 = \{a^m b^k a^m b^k \mid m, k \in \mathbb{N}\}.$$

55. Opisać algorytm, który w pamięci logarytmicznej oblicza wartość danej formuły rachunku zdań (przy ustalonym wartościowaniu).

56. Skonstruować maszyny Turinga rozpoznające następujące języki i określić ich złożoność czasową i pamięciową:
(a) $\{ww \mid w \in \{0, 1\}^*\}$;
(b) $\{w \in \{0, 1\}^* \mid$ w słowie w jest parzysta liczba jedynek$\}$;
(c) $\{w \in \{0, 1\}^* \mid$ w słowie w jest więcej jedynek niż zer$\}$;
(d) $\{w \in \{(,)\}^* \mid w$ jest poprawnym wyrażeniem nawiasowym$\}$;
(e) $\{w \in \{(,), [,], \langle, \rangle\}^* \mid w$ jest poprawnym wyrażeniem nawiasowym$\}$;
(f) $\{a^{2^n} \mid n \in \mathbb{N}\}$;
(g) $\{\varphi \mid \varphi$ jest poprawnie zbudowaną formułą rachunku zdań$\}$;
(h) $\{\varphi \mid \varphi$ jest formułą zdaniową oraz $[\![\varphi]\!]_\rho = 1\}$, gdzie ρ jest ustalonym wartościowaniem.

57. Ile czasu potrzeba na pełen cykl pracy licznika binarnego długości $\log n$ (od $0^{\log n}$ do $1^{\log n}$)?

58. Niech $L = L(\mathcal{M})$ dla pewnej maszyny wielotaśmowej o złożoności czasowej $T(n)$ i pamięciowej $S(n)$. Pokazać, że maszyna jednotaśmowa taka jak w dowodzie faktu 6.2 rozpoznaje ten sam język w pamięci $S(n)$ i czasie $\mathcal{O}(T(n)^2)$.

59. *Klasyczną* maszyną Turinga nazwijmy maszynę o jednej taśmie nieskończonej w prawo, służącej zarówno jako taśma wejściowa jak robocza. Pokazać, że dla każdej deterministycznej maszyny $\mathcal{M}$ o złożoności czasowej $T(n)$ istnieje klasyczna maszyna deterministyczna o złożoności czasowej $\mathcal{O}(T(n)^2)$ rozpoznająca ten sam język.

60. Niech $\mathcal{M}$ będzie niedeterministyczną maszyną wielotaśmową o złożoności czasowej $T(n)$. Pokazać, że istnieje dwutaśmowa niedeterministyczna maszyna $\mathcal{M}'$ o złożoności czasowej $\mathcal{O}(T(n))$, rozpoznająca ten sam język.[31]

61. Pokazać, że wielomiany i funkcje 2^n, $\lceil \log n \rceil$ $\lfloor \log n \rfloor$, $\lceil \sqrt{n} \rceil$ są konstruowalne.

62. Każde słowo $w \in L(\mathcal{M})$ jest akceptowane przez deterministyczną maszynę $\mathcal{M}$ w co najwyżej $T(|w|)$ krokach. Pokazać, że jeśli T jest konstruowalna, to $L(\mathcal{M}) \in \textsc{Dtime}(T(n))$.

63. Dla każdego słowa $w \in L(\mathcal{M})$ istnieje akceptujące obliczenie niedeterministycznej maszyny $\mathcal{M}$ o długości co najwyżej $T(n)$. Pokazać, że jeśli T jest konstruowalna, to $L(\mathcal{M}) \in \textsc{Ntime}(T(n)$.

64. Wiadomo, że deterministyczną maszynę wielotaśmową o złożoności czasowej $T(n)$ można symulować dwutaśmową maszyną o złożoności czasowej $T(n) \log T(n)$. Udowodnić twierdzenie 8.7(2).

R 65. Które z następujących zawierań: CSL $\subseteq$ $\textsc{Dspace}(n^2 \log n) \subseteq \textsc{Nspace}(n \log\log n) \subseteq$ P (gdzie CSL oznacza klasę języków kontekstowych) faktycznie zachodzą, które nie zachodzą, a które implikują równość $\textsc{Pspace} =$ P?

66. Które z następujacych inkluzji są prawdziwe i dlaczego?

(a) $\textsc{Dtime}(3n^2) \subseteq \textsc{Dtime}(n^2)$?
(b) $\textsc{Dtime}(2^{3n^2}) \subseteq \textsc{Dtime}(2^{n^2})$?
(c) $\textsc{Ntime}(3n^2) \subseteq \textsc{Dtime}(2^{n^3})$?
(d) $\textsc{Dtime}(3n^2) \subseteq \textsc{Dspace}(n^2)$?
(e) $\textsc{Ntime}(3n^2) \subseteq \textsc{Dspace}(n^2)$?
(f) $\textsc{Dspace}(3n^2) \subseteq \textsc{Dtime}(2^{64}n + 123)$?
R (g) $\textsc{Dspace}(3n) \subseteq \textsc{Nspace}(2n)$?
R (h) $\textsc{Dspace}(2^{3n}) \subseteq \textsc{Dspace}(2^{2n})$?
R (i) $\textsc{Dtime}(2^{3n}) \subseteq \textsc{Dtime}(2^{2n})$?
R (j) $\textsc{Dspace}(3n) \subseteq \textsc{Dtime}(n^n)$?
R (k) $\textsc{Nspace}(3n) \subseteq \textsc{Dtime}(n^n)$?
R (l) $\textsc{Dtime}(n^3) \subseteq \textsc{Dtime}(2n)$?
R (m) $\textsc{Ntime}(n \log n) \subseteq \textsc{Dtime}(2^{n^2})$?
R (n) $\textsc{Nspace}(\log n) \subseteq \textsc{Dspace}(n)$?
(o) $\textsc{Nspace}(n^2) \subseteq \textsc{Dspace}(n^5)$?
R (p) $\textsc{Ntime}(n) \subseteq \textsc{Dtime}(n^n)$?
(q) $\textsc{Dtime}(2^{4n}) \subseteq \textsc{Dtime}(2^n)$?
(r) $\textsc{Nspace}(3^n) \subseteq \textsc{Nspace}(2^n)$?

R 67. Która z następujących inkluzji jest prawdziwa:
$\textsc{Nspace}(n \log n) \subseteq \textsc{Dspace}(2n^3) \subseteq \textsc{Dspace}(n^3) \subseteq \textsc{Dtime}(2^{n^4}) \subseteq \textsc{Dtime}(2^n)$?

R 68. Czy prawdziwe są następujące zawierania:
(a) $\bigcup_{k\in\mathbb{N}} \textsc{Nspace}(\log^k n) \subseteq \bigcup_{k\in\mathbb{N}} \textsc{Dspace}(\log^k n)$?
(b) $\textsc{Nspace}(\log^2 n) \subseteq \textsc{Dtime}(n^{\log^2 n})$?
(c) $\textsc{Dtime}(n^2\sqrt{n}) \subseteq \textsc{Dtime}(n^2)$?
(d) $\textsc{Dtime}((\log n)^n) \subseteq \textsc{Dspace}(n)$?

R 69. Czy to możliwe, że:
(a) $\textsc{Dspace}(n^2) \subseteq \textsc{Nspace}(n)$ ale $\textsc{Nspace}(n^2) \neq \textsc{Dspace}(n^4)$?
(b) Pewien język bezkontekstowy jest NP-zupełny?
(c) $\textsc{Dtime}(n^{\log n}) \subseteq$ P?
(d) NP $=$ Pspace?

Jeśli możliwe, to co by z tego wynikało?

[31] Dla maszyn deterministycznych najlepsze dwutaśmowe ograniczenie to $T(n) \log T(n)$, por. ćwiczenie 75.

R 70. Która z następujących inkluzji zachodzi, ktora nie zachodzi, a która stanowi problem otwarty? Jeśli inkluzja zachodzi, to czy znak $\subseteq$ można zamienić na $\subsetneq$? Jeśli problem jest otwarty, to co wynika z pozytywnej odpowiedzi?

(a) $\textsc{Nspace}(\log n) \subseteq \textsc{Dspace}(\sqrt{n})$;

(b) $\textsc{Dspace}(\sqrt{n}) \subseteq \text{NP}$;

(c) $\text{NP} \subseteq \textsc{Dtime}(n^2)$.

R 71. Które z następujących inkluzji zachodzą? Które zawierania są właściwe?

(a) $\textsc{Ntime}(n \log n) \subseteq \textsc{Dspace}(n\sqrt{n})$;

(b) *co*-$\textsc{Nspace}(n\sqrt{n}) \subseteq \textsc{Dspace}(n^4)$;

(c) $\textsc{Nspace}(3n^2 + 2) \subseteq$ *co*-$\textsc{Nspace}(n^2)$;

(d) $\textsc{Dtime}(3^{n^2}) \subseteq \textsc{Ntime}(n\sqrt{n})$.

R 72. Które z następujacych inkluzji są prawdziwe?

(a) $\textsc{Atime}(n^2) \subseteq \textsc{Dspace}(n^2)$?
(b) $\text{NC}_2 \subseteq$ *co*-RP?
(c) $\text{NC}_1 \subseteq \textsc{Logspace}$?
(d) *co*-$\textsc{Nspace}(2n) \subseteq \textsc{Nspace}(n \log\ n)$
(e) *co*-$\textsc{Nspace}(\log n) \subseteq \textsc{Nspace}(\log^2 n)$?

73. Uzupełnić, podając jak najlepsze oszacowanie:

(a) *co*-$\textsc{Nspace}(\ldots) \subseteq \textsc{Nspace}(\log^2 n) \subseteq \textsc{Dspace}(\ldots) \subseteq \textsc{Dtime}(\ldots)$.
(b) *co*-$\textsc{Nspace}(\log^3 n) \subseteq \textsc{Dspace}(\ldots) \subseteq \textsc{Dtime}(\ldots) \subsetneq \textsc{Dspace}(2^n)$.
(c) *co*-$\textsc{Nspace}(3\log^2 n) \subseteq \textsc{Dspace}(\ldots) \subseteq \textsc{Dtime}(\ldots) \subsetneq \textsc{Dspace}(2^n)$.

74. Dla ustalonego k, niech $w_k = \#0\ldots00\#0\ldots01\#\ldots\#1\ldots11\#$ będzie słowem złożonym z binarnych reprezentacji liczb $0, 1, \ldots, 2^k - 1$ (każdy segment długości k). Pokazać, że język $L = \{w_k \mid k \in \mathbb{N}\}$ nie jest regularny, i że $L \in \textsc{Dspace}(\log\log n)$.

75.* Pokazać, że deterministyczne rozpoznawanie palindromów na jednej taśmie wymaga co najmniej czasu rzędu n^2.

R 76.* Czy $\textsc{Polylog} := \bigcup_k \textsc{Dspace}(\log^k(n)) = \text{P}$?

77. Udowodnić, że jeśli $\textsc{Dspace}(n) \subseteq \textsc{Ptime}$, to $\textsc{Ptime} = \textsc{Pspace}$.

78. Udowodnić, że $\textsc{Dspace}(n) \neq \textsc{Ptime}$.

R 79. Udowodnić, że jeśli $\textsc{Logspace} = \textsc{Ptime}$, to $\textsc{Pspace} = \textsc{Exptime}$.

R 80. Udowodnić, że jeśli $\textsc{Ptime} = \textsc{Pspace}$ to $\textsc{Exptime} = \textsc{Expspace}$.

R 81. Udowodnić, że jeśli $\textsc{Nlogspace} = \textsc{Logspace}$, to każdy język kontekstowy jest deterministyczny.

R 82. Udowodnić, że jeśli NP = *co*-NP, to $\textsc{Nexptime}=$ *co*-$\textsc{Nexptime}$.

R 83. Udowodnić, że jeśli $\textsc{Dspace}(n^2) \subseteq \text{NP}$, to $\textsc{Pspace} = \text{NP}$.

R 84. Udowodnić, że jeśli $\textsc{Dspace}(2^n) \subseteq \textsc{Exptime}$, to $\textsc{Expspace} \subseteq \textsc{Exptime}$.

R 85. Udowodnić, że jeśli $\text{CSL} \subseteq \text{NP}$, to $\text{NP} = \textsc{Pspace}$.

R 86. Czy jeśli $\textsc{Ntime}(n^2) \subseteq \text{P}$, to P = NP?

R 87. Które z następujących inkluzji zachodzą, a które nie? Które stanowią pytania otwarte? Czy z pozytywnych odpowiedzi na te otwarte pytania wynika równość $\textsc{Nlogspace} = \text{P}$?

$$\textsc{Dtime}(n^2) \subseteq \textit{co}\text{-}\textsc{Nlogspace} \subseteq \textsc{Dtime}(n^{\log n}) \subseteq \textsc{Dspace}(n^n) \subseteq \textsc{Pspace}?$$

88. Udowodnić, że relacja $\leq_{\log}$ jest przechodnia.

89. Udowodnić, że problem osiągalności pozostaje NLOGSPACE-zupełny jeśli ograniczymy go do grafów acyklicznych.

90. Pokazać jak każdą sieć boolowską można przekształcić w równoważną formułę zdaniową. Jakie są (w najgorszym przypadku) rozmiary tej formuły? Czy obliczanie wartości sieci jest tak samo trudne jak obliczanie wartości formuły (por. ćwiczenie 56h)?[32]

91. Opisać maszyny realizujące LOGSPACE-redukcje, o których mowa w dowodach lematu 10.7 i twierdzenia 10.8.

92.* Sieć *monotoniczna* to sieć bez negacji, która za to ma podwójną liczbę wejść. Każdemu wejściu x odpowiada dualny wierzchołek wejściowy $\overline{x}$, którego wartość jest negacją wartości x. Udowodnić, że problem wartości sieci monotonicznej też jest P-zupełny.

93. Pokazać, że problem wartości sieci pozostaje P-zupełny jesli ograniczymy się do bramek *binarnych* tj. bramek o co najwyżej dwóch wejściach (sieć na rysunku 6 nie ma tej własności).

94. Pokazać, że problem prawdziwości formuły zdaniowej w koniunkcyjnej postaci normalnej (równoważnie, problem spełnialności formuły w dyzjunkcyjnej postaci normalnej) należy do klasy P.

95. Pokazać, że problem spełnialności formuł w koniunkcyjnej postaci normalnej z dwoma literałami w każdym członie koniunkcji (2-CNF-SAT) jest zupełny w klasie NLOGSPACE. (*Wskazówka: pokazać zupełność w co-*NLOGSPACE *i użyć twierdzenia 9.4.*)

R 96. Udowodnić, że problem kliki jest NP-zupełny.

97. Udowodnić, że problem pakowania jest NP-zupełny.

R 98. Rozpatrzmy następujące problemy decyzyjne:

(a) Dana formuła zdaniowa φ. Czy φ ma co najmniej dwa wartościowania spełniające?

(b) Dany graf G i liczba k. Czy istnieje taki k-elementowy zbiór S wierzchołków grafu G, że żadne dwa wierzchołki ze zbioru S nie są połączone krawędzią?

(c) Dany graf G i liczby d, k. Czy istnieje taki k-elementowy zbiór S wierzchołków grafu G, że każdy z pozostałych wierzchołków jest osiągalny z jakiegoś wierzchołka należącego do S drogą składającą się z co najwyżej d krawędzi?

Czy któryś z nich jest rekurencyjnie przeliczalny, NP-zupełny, NL-zupełny, rozstrzygalny?

99. *Problem pustości dla sieci* polega na ustaleniu, czy dana sieć boole'owska akceptuje przynajmniej jedno słowo. Proszę udowodnić, że problem pustości dla sieci jest NP-zupełny. Czy zachodzi to także dla sieci o wysokości 3?

100. Pokazać, że we wniosku 11.4 można żądać, aby $L' \in$ LOGSPACE.

101. Czy waga skojarzenia S, o którym mowa w dowodzie faktu 12.3 może być większa od połowy wagi drzewa T?

102. Opisać wielomianowy algorytm znajdujący minimalne drzewo rozpinające grafu.

103. Jaką aproksymację otrzymujemy dla problemu pakowania, stosując następujący algorytm?
– Każdy element wkładamy do ostatniego pojemnika jeśli się zmieści;
– jeśli się nie zmieści, to bierzemy nastepny pojemnik.

104. Jaką aproksymację otrzymujemy dla problemu pokrycia wierzchołkowego, jeśli wyszukamy maksymalne skojarzenie w grafie i jako rozwiązanie wskażemy jego wierzchołki?

[32] Sieć tym się różni od formuły zdaniowej, że z jednego wierzchołka może wychodzić dowolnie wiele krawędzi.

105. Udowodnić, że jeśli PH = PSPACE to zachodzi *kolaps hierarchii wielomianowej*, tj. dla pewnego n mamy $\mathrm{PH} = \Sigma_n^p$.

106. Udowodnić, że każda równość postaci $\Sigma_k^p = \Pi_k^p$ implikuje kolaps hierarchii wielomianowej.

107. Udowodnić, że istnienie problemu zupełnego w PH implikuje kolaps hierarchii wielomianowej.

108. Opisać alternujący algorytm obliczający wartość sieci boolowskiej w pamięci logarytmicznej.

109. Opisać alternujący algorytm dla QBF, który działa w czasie wielomianowym. Czy można go poprawić do ALOGSPACE (por. ćwiczenie 108)?

110. Udowodnić, że APSPACE = EXPTIME (twierdzenie 15.6(3)).

111. Pokazać, że jeśli funkcja $S(n) \geq \log n$ jest konstruowalna, to:
   (a) $\mathrm{ASPACE}(S(n)) \subseteq \mathrm{DTIME}(2^{\mathcal{O}(S(n))})$;
   (b) $\mathrm{NSPACE}(S(n)) \subseteq \mathrm{ATIME}(\mathcal{O}(S(n)^2))$.

112. Jeśli $T(n) \geq n$ jest konstruowalna to:
   (a) $\mathrm{DTIME}(T(n)) \subseteq \mathrm{ASPACE}(\mathcal{O}(\log T(n)))$;
   (b) $\mathrm{ATIME}(T(n)) \subseteq \mathrm{DSPACE}(T(n))$.

113. Pokazać, że dowolny język nad alfabetem jednoliterowym (także nierozstrzygalny) należy do klasy P/poly.

114.* Pokazać, że dowolny język jest w P wtedy i tylko wtedy, gdy jest rozpoznawany przez jednostajną rodzinę sieci rozmiaru wielomianowego z binarnymi bramkami. Czy stąd wynika PTIME ⊆ NC?

115. Pokazać, że PARITY $\in \mathrm{NC}_1$. Wywnioskować stąd, że $\mathrm{AC}_0 \neq \mathrm{NC}_1$.

116. Czy dla rozmiaru sieci zachodzi twierdzenie o hierarchii podobne do tw. 8.6?

R 117. Które z następujących terminów oznaczają to samo, a które co innego? Proszę objaśnić na przykładach.
   (a) Język obliczalny;
   (b) Język rekurencyjnie przeliczalny;
   (c) Język rozstrzygalny;
   (d) Język częściowo obliczalny;
   (e) Język rekurencyjny;
   (f) Język typu zero;
   (g) Język kontekstowy;
   (h) Język monotoniczny.

R 118. Które z następujących stwierdzeń są prawdziwe?
   (a) Problem prawdziwości formuły zdaniowej jest NP-zupełny.
   (b) Problem odpowiedniości Posta jest NP-trudny.
   (c) Klasa języków bezkontekstowych jest zamknięta ze względu na dopełnienie.
   (d) Dopełnienie języka rekurencyjnie przeliczalnego jest rekurencyjnie przeliczalne.
   (e) Problem pustości języka bezkontekstowego jest nierozstrzygalny.
   (f) Jeśli $L_1 \leq_{\log} L_2$, to $-L_1 \leq_{\log} -L_2$.
   (g) Jeśli L_1 jest PSPACE-zupełny, to $-L_1$ też.
   (h) Jeśli L_1 jest PSPACE-zupełny i $L_1 \subseteq L_2$, to L_2 też jest PSPACE-zupełny.
   (i) Jeśli L_1 i L_2 są PSPACE-zupełne, to $L_1 \cup L_2$ lub $L_1 \cap L_2$ jest PSPACE-zupełny.
   (j) Jeśli L jest PSPACE-zupełny, to L jest NP-trudny.
   (k) Każdy język kontekstowy jest w klasie LOGSPACE.
   (l) Klasa $\mathrm{DTIME}(n^{\log n})$ jest zawarta w P.
   (m) Iloczyn NP ∩ *co*-NP jest pusty?
   (n) Problem kolorowania grafu jest w klasie P.
   (o) Jeśli $L_1 \leq_{\log} L_2$, to $L_1 \in \mathrm{P}^{L_2}$.

R 119. Niech $L_1 = \{a^n b^m \mid n \neq m\}$ i $L_2 = \{ww^R w^R \mid w \in \{a,b\}^*\}$.

(a) Który z języków L_1, L_2 jest regularny?
(b) Który z nich jest bezkontekstowy?
(c) Czy któryś z nich należy do klasy P lub PSPACE?
(d) Czy następujący problem jest rozstrzygalny:
Dana deterministyczna maszyna Turinga M, czy $L(M) = L_1$?

Klasówki i egzaminy z ostatnich kilku lat:

- Klasówka 2018-19: zadania 10, 11.
- Klasówka 2019-20: zadania: 52, 67, 84.
- Klasówka 2020-21: zadanie 119.
- Klasówka 2022-23: zadania 42, 54.
- Egzamin 2018-19: zadania 14p–14q, 44, 83, 118k–118n.
- Egzamin 2019-20: zadania 53, 98, 86.
- Egzamin 2020-21: zadania 18, 20, 65.
- Egzamin 2021-22: zadania 12s–12u, 41, 87.
- Egzamin 2022-23: zadania 16, 43, 70.
- Egzamin 2023-24: zadania 17, 45, 85, 71.

Rozwiązania

2: Automat składa się z sześciu stanów: $0, 1, 2, 3, 4, 5$. Wszystkie stany, oprócz 5, są akceptujące. Stanem początkowym jest 0. Funkcja przejścia jest określona tabelką:

	0	1
0	0	1
1	2	1
2	0	3
3	0	4
4	5	5
5	5	5

Poszukiwane wyrażenie regularne jest np. takie:

$$(0 \cup 11^*0(0 \cup 10))^*(\varepsilon \cup 11^* \cup 11^*0 \cup 11^*01 \cup 11^*011)\,.$$

(Zauważmy, że język $(0 \cup 11^*0(0 \cup 10))^*$ byłby językiem akceptowanym przez ten automat, gdyby 0 było jedynym stanem akceptującym.)

10a: Język L_1 jest regularny, bo można go zdefiniować wyrażeniem regularnym:

$$L_1 = (aabb \cup abab \cup abba \cup baab \cup baba \cup bbaa)^+.$$

10b: Język L_2 jest bezkontekstowy, bo $L_2 = L(G)$, gdzie G jest gramatyką o produkcjach:

$$\xi_0 ::= \xi\xi\xi\xi; \quad \xi ::= a\xi b \mid b\xi a \mid ab \mid ba \mid \xi\xi\,.$$

Język L_2 nie jest regularny. Na przykład dlatego, że ma nieskończenie wiele różnych ilorazów: wszystkie języki $L_2 \setminus a^n$ są różne. Mamy bowiem $b^n ababab \in L_2 \setminus a^n$, ponieważ $a^n b^n ababab \in L_2$, ale $b^n ababab \in L_2 \setminus a^m$, dla $m \neq n$, ponieważ $a^m b^n ababab \notin L_2$.

10c: Język L_3 nie jest bezkontekstowy. W przeciwnym razie język $L = L_3 \cap a^+b^+a^+b^+a^+b^+a^+b^+$ byłby bezkontekstowy, jako iloczyn bezkontekstowego i regularnego. Niech N będzie stałą z lematu

o pompowaniu dla języka L. Rozpatrzmy słowo $w = a^N b^N a^N b^N a^N b^N a^N b^N \in L$. Na mocy lematu o pompowaniu możliwy jest taki podział słowa w na 5 części, że $w = xyzuv$, gdzie $|yzu| \leq N$ i $yu \neq \varepsilon$ oraz $xzv \in L$. Zatem długość słowa xzv jest postaci $8n$ gdzie $n < N$ i słowo to składa się z czterech zrównoważonych segmentów długości $2n < 2N$. Ale część yzu słowa w jest krótka i mieści się w co najwyżej dwóch sąsiadujących segmentach postaci a^N lub b^N. A zatem na początku lub na końcu słowa xzv mamy nadal $a^N b^N$. Czyli początkowy albo końcowy segment długości $2n$ nie może być zrównoważony.

11: Skoro L_1 jest regularny, to $L_1 \in \textsc{Dspace}(1)$. Natomiast język L_2 należy do klasy $\textsc{Dspace}(\log n)$. Do rozpoznania czy słowo w należy do L_2 potrzebny jest licznik zliczający nadmiar liter a nad b, lub nadmiar b nad a. Wartość licznika nie przekracza n zatem jego rozmiar jest logarytmiczny. Licznik powinien co najmniej cztery razy się wyzerować, w tym na końcu słowa.

Na koniec zauważmy, że $L_3 \in \textsc{Dspace}(\log n)$. Deterministyczna maszyna Turinga najpierw sprawdza czy długość danego słówa jest postaci $4k$ i zapamiętuje liczbę k (pierwszy licznik). Potem cztery razy sprawdza, czy kolejne segmenty długości k są zrównoważone (drugi licznik). Potrzebne są więc dwa liczniki rozmiaru $\log n$.

12a: Język L_a nie jest regularny, na przykład dlatego, że ma nieskończenie wiele ilorazów. Faktycznie, wszystkie języki $L_a \backslash b^k = \{w \in \{a,b\}^* \mid a_w = 2b_w + k + 1\}$ są różne.

To jest natomiast język bezkontekstowy, a nawet deterministyczny język bezkontekstowy. Automat rozpoznający ten język ma na stosie zawsze $|a_v - 2b_v - 1|$ piłeczek, gdzie v jest dotychczas przeczytanym słowem. Jeśli różnica $a_v - 2b_v - 1$ jest dodatnia, to piłeczki na stosie są zielone, w przeciwnym razie są czerwone. Po przeczytaniu każdego a, automat umieszcza na stosie nastepną piłeczkę zieloną albo zdejmuje jedną czerwoną. Po przeczytaniu każdego b ze stosu zdejmuje się dwie piłeczki zielone, albo wkłada dwie czerwone, nigdy nie mieszając kolorów. Sprawdzając każdorazowo czy stos jest pusty, automat potrafi rozpoznać te słowa dla których różnica jest zerowa.

12b: Język L_b jest bezkontekstowy. Generuje go gramatyka z symbolem początkowym ξ_0:

$$\xi_0 \Rightarrow \xi_1 \mid a\xi_0 b, \qquad \xi_1 \Rightarrow \varepsilon \mid b\xi_1 a.$$

Ten język nie jest regularny, bo ma nieskończenie wiele ilorazów. Języki $L_b \setminus a^k$ są różne dla różnych k, mamy np. $a^k \in L_b \setminus a^k$, ale $a^k \notin L_b \setminus a^n$, gdy $n \neq k$.

12d: Język L_d nie jest bezkontekstowy. Można się o tym przekonać stosując lemat o pompowaniu. Przypuśćmy, że język L_d jest bezkontekstowy i niech n będzie stałą z lematu o pompowaniu. Rozpatrzmy słowo $a^{n^2+1}b^n$. Jeśli rozbijemy to słowo na pięć części $xyzuv$, w ten sposób, że zarówno xzv jak xy^2zu^2v należą do L_3, to będą spełnione równania:

$$n^2 + 1 - m = (n-k)^2 + 1 \qquad \text{oraz} \qquad n^2 + 1 + m = (n+k)^2 + 1,$$

gdzie $m = \#(a,y) + \#(a,u)$ i $k = \#(b,y) + \#(b,u)$. Proste rachunki dają w wyniku $m = k = 0$, czyli słowo yu musiałoby być puste – sprzeczność.

12f: Język L_f jest regularny, bo $L_f = \{a,b\}^*$.

12h: Język L_h jest regularny. Automat rozpoznający ten język zlicza resztę liczby $x + z - (y + u)$ z dzielenia przez 3. Ma on 13 stanów. Stanem początkowym jest q_0, a stany końcowe to p_0, q_0, r_0 i s_0.

	q_0	q_1	q_2	p_0	p_1	p_2	r_0	r_1	r_2	s_0	s_1	s_2	$\bullet$
a	q_1	q_2	q_0	$\bullet$	$\bullet$	$\bullet$	$\bullet$	$\bullet$	$\bullet$	$\bullet$	$\bullet$	$\bullet$	$\bullet$
b	p_2	p_0	p_1	p_2	p_0	p_1	$\bullet$	$\bullet$	$\bullet$	$\bullet$	$\bullet$	$\bullet$	$\bullet$
c	r_1	r_2	r_0	r_1	r_2	r_0	r_1	r_2	r_0	$\bullet$	$\bullet$	$\bullet$	$\bullet$
d	s_2	s_0	s_1	s_2	s_0	s_1	s_2	s_0	s_1	s_2	s_0	s_1	$\bullet$

12i: Język L_i nie jest regularny, bo ma nieskończenie wiele ilorazów lewostronnych. Istotnie, wszystkie języki $L_2 \backslash a^k = \{a^x b^y c^z d^u \mid x + z + k \geq y + u\}$ są różne.

Język L_i jest natomiast bezkontekstowy. Automat rozpoznający ten język zapamiętuje na stosie liczbę $|x + z - (y + u)|$ w postaci ciągu gwiazdek. Stany automatu są postaci q_x°, gdzie $x \in \{a,b,c,d\}$ a $\circ$ to plus albo minus. Znak $\circ$ wskazuje na to, czy liczba $x + z - (y + u)$ jest dodatnia czy ujemna. Dodatkowo automat ma stan pomocniczy *fajrant*. Stanem początkowym jest q_a^+, a stanami akceptującymi są wszystkie q_x^-. Automat działa tak:

- W stanie q_x° automat czyta kolejne litery x, każdorazowo umieszczając gwiazdkę na stosie, lub zdejmując gwiazdkę ze stosu, według opisu poniżej. Jeśli na wejściu pojawi się nowa literę y, alfabetycznie dalsza od x, to automat przechodzi do stanu q_y°. Jeśli y jest w alfabecie wcześniej niż x to automat przechodzi do stanu *fajrant*.
- W stanach $q_a^+, q_b^-, q_c^+, q_d^-$ automat wkłada gwiazdki na stos.
- W pozostałych stanach q_x° automat zdejmuje gwiazdki ze stosu. Jeśli gwiazdek zabraknie, to automat przechodzi do stanu $q_x^\bullet$, gdzie $\bullet$ oznacza znak przeciwny do $\circ$.
- W stanie *fajrant* automat pozostaje na dobre, czegokolwiek by nie przeczytał.

12j: Język L_j nie jest bezkontekstowy. Przypuśćmy, że jest i niech N będzie stałą z lematu o pompowaniu. Rozpatrzmy słowo $w = a^N b^N c^N d^N$. Powinno istnieć takie rozbicie tego słowa $w = xyzuv$, gdzie yu jest niepuste, oraz wszystkie słowa $xy^i zu^i v$ należą do naszego języka. Przy tym długość podsłowa yzu nie przekracza N.

Przypadek 1: Słowo yzu zawiera jakieś litery a. Wtedy nie ma w nim ani jednego c, bo słowo yzu jest na to za krótkie. Zatem w słowie xy^2zu^2v nie zgadza się liczba liter a i c.

Przypadek 2: Podobnie jest, gdy w słowie yzu znajdzie się jakieś c. (Wtedy nie ma tam żadnego a.)

Przypadek 3: Słowo yzu jest w całości zawarte we fragmencie b^N lub we fragmencie d^N słowa w. Wtedy w słowie xy^2zu^2v jest za dużo liter b lub d.

12l: Język L_l jest regularny. Automat rozpoznający ten język zlicza resztę liczby $x + u - (y + z)$ z dzielenia przez 3. Stanem początkowym jest q_0, a stanami końcowymi są p_1, p_2, q_1, q_2, r_1, r_2,s_1 i s_2.

	q_0	q_1	q_2	p_0	p_1	p_2	r_0	r_1	r_2	s_0	s_1	s_2	$\bullet$
a	q_1	q_2	q_0	$\bullet$	$\bullet$	$\bullet$	$\bullet$	$\bullet$	$\bullet$	$\bullet$	$\bullet$	$\bullet$	$\bullet$
b	p_2	p_0	p_1	p_2	p_0	p_1	$\bullet$	$\bullet$	$\bullet$	$\bullet$	$\bullet$	$\bullet$	$\bullet$
c	r_2	r_0	r_1	r_2	r_0	r_1	r_2	r_0	r_1	$\bullet$	$\bullet$	$\bullet$	$\bullet$
d	s_1	s_2	s_0	s_1	s_2	s_0	s_1	s_2	s_0	s_1	s_2	s_0	$\bullet$

12m: Język L_m nie jest regularny, bo ma nieskończenie wiele ilorazów lewostronnych. Istotnie, wszystkie języki $L_m \backslash a^k = \{a^x b^y c^z d^u \mid x + u + k \geq y + z\}$ są różne.

Język L_m jest natomiast bezkontekstowy. Automat rozpoznający ten język zapamiętuje na stosie liczbę $|x + u - (y + z)|$ w postaci ciągu gwiazdek. Stany automatu są postaci q_x°, gdzie $x \in \{a, b, c, d\}$ a $\circ$ to plus albo minus. Znak $\circ$ wskazuje na to, czy liczba $x + u - (y + z)$ jest dodatnia czy ujemna. Dodatkowo automat ma stan pomocniczy *fajrant*. Stanem początkowym jest q_a^+, a stanami akceptującymi są wszystkie q_x^+. Automat działa tak:

- W stanie q_x° automat czyta kolejne litery x, każdorazowo umieszczając gwiazdkę na stosie, lub zdejmując gwiazdkę ze stosu, według opisu poniżej. Jeśli na wejściu pojawi się nowa literę y, alfabetycznie dalsza od x, to automat przechodzi do stanu q_y°. Jeśli y jest w alfabecie wcześniej niż x to automat przechodzi do stanu *fajrant*.
- W stanach $q_a^+, q_b^-, q_c^-, q_d^+$ automat wkłada gwiazdki na stos.
- W pozostałych stanach q_x° automat zdejmuje gwiazdki ze stosu. Jeśli gwiazdek zabraknie, to automat przechodzi do stanu $q_x^\bullet$, gdzie $\bullet$ oznacza znak przeciwny do $\circ$.
- W stanie *fajrant* automat pozostaje na dobre, czegokolwiek by nie przeczytał.

12n: Język L_n nie jest bezkontekstowy. Przypuśćmy, że jest i niech N będzie stałą z lematu o pompowaniu. Rozpatrzmy słowo $w = a^N b^N c^N d^N$. Powinno istnieć takie rozbicie tego słowa $w = xyzuv$, gdzie yu jest niepuste, oraz wszystkie słowa $xy^i zu^i v$ należą do naszego języka. Przy tym długość podsłowa yzu nie przekracza N.
Przypadek 1: Słowo yzu zawiera jakieś litery a. Wtedy nie ma w nim ani jednego d, bo słowo yzu jest na to za krótkie. Zatem w słowie xy^2zu^2v nie zgadza się liczba liter a i d.
Przypadek 2: Podobnie jest, gdy w słowie yzu znajdzie się jakieś d. (Wtedy nie ma tam żadnego a.)
Przypadek 3: Słowo yzu jest w całości zawarte we fragmencie $b^N c^N$ słowa w. Wtedy w słowie xy^2zu^2v jest za dużo liter b i c.

12s: Język L_s jest regularny, bo $L_1 = (aabbb)^*$.

12t: Język L_t nie jest regularny, bo ma nieskończenie wiele różnych ilorazów (wszystkie języki $L_2\backslash a^{2r} = \{a^{2m}b^{3r+3m} \mid m \in \mathbb{N}\}$ są różne). Jest to język bezkontekstowy, generowany przez gramatykę o produkcjach $\xi_0 ::= aa\xi_0bbb \mid \varepsilon$.

12u: Język L_u nie jest nawet bezkontekstowy, bo nie spełnia lematu o pompowaniu. Przypuśćmy bowiem, że N jest stałą z lematu. Jeśli $a^{2^N}b^{3^N} = xyzuv$, gdzie $0 < |yu| \leq N$, to słowo xy^2zu^2v nie należy do L_3, bo ma złą długość. Ponieważ $2^N + 3^N < |xy^2zu^2v| \leq 2^N + 3^N + N < 2^{N+1} + 3^{N+1}$, więc słowo xy^2zu^2v jest dłuższe niż każde słowo $a^{2^k}b^{2^k}$ dla $k \leq N$ i krótsze niż wszystkie słowa $a^{2^k}b^{2^k}$, dla dowolnego $k > N$.

13: Wszystkie te języki są rozpoznawalne w deterministycznej pamięci logarytmicznej, tj. należą do klasy LOGSPACE. Na przykład maszyna Turinga rozpoznająca język L_c odszukuje najpierw środek słowa wejściowego, a następnie porównuje ze sobą klatka po klatce obie połowy. Używa w tym celu tyle pamięci ile potrzeba do zapamiętania kilku pozycji na taśmie wejściowej, czyli $\mathcal{O}(\log n)$. Podobnie działają maszyny rozpoznające inne języki.

Skoro nasze języki są w LOGSPACE, to są rozstrzygalne, więc na pytanie (e) odpowiedź jest „nie". Na pozostałe pytania należy odpowiedzieć „tak", bo klasa LOGSPACEjest zawarta zarówno w *co*-NP jak i w klasie języków kontekstowych, a tym bardziej w klasie języków rekurencyjnie przeliczalnych.

14a: Ten język jest regularny, określony np. wyrażeniem regularnym $(a^2)^*(ab \cup \varepsilon)(b^2)^*(ba \cup \varepsilon)(a^2)^*$.

14b: Ten język jest bezkontekstowy, generowany przez gramatykę o produkcjach:

$$\xi_0 \Rightarrow \xi \mid a\xi_0 a; \qquad \xi \Rightarrow \varepsilon \mid a\xi b,$$

gdzie ξ_0 jest symbolem początkowym. Ale nie jest to język regularny, bo ma nieskończenie wiele różnych ilorazów. Na przykład, dla różnych $r \in \mathbb{N}$ różne są języki

$$L_2 \setminus a^r = \{w \mid a^r w \in L_2\} = \{a^m b^n a^k \mid r + m = n + k\}.$$

14c: Ten język nie jest bezkontekstowy. Przypuśćmy przeciwnie i niech N będzie stałą z lematu o pompowaniu. Słowo $w = a^{2N}b^Na^N$ można przedstawić w postaci $w = uvzxy$, gdzie $uv^izx^iy \in L_3$ dla każdego $i \in \mathbb{N}$. Przy tym $vx \neq \varepsilon$, a słowo vzx jest długości co najwyżej N.

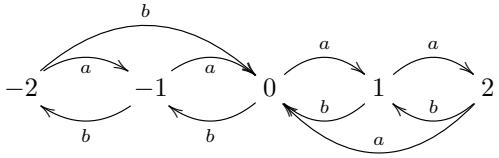
W szczególności, słowo $w' = uv^2zx^2y$ musi być postaci $a^{2M}b^Ma^M$, gdzie $M > N$, bo w' jest dłuższe niż w. Słowo vzx jest na tyle krótkie, że musi się zawierać w części $a^{2N}b^N$ albo w części b^Na^N. W pierwszym przypadku otrzymamy $w' = w''ba^N$ (nie zmienia się końcowa część od ostatniego b włącznie) w drugim $w' = a^{2N}bw''$ (nie zmieni się część początkowa do pierwszego b włącznie). Stąd $M = N$, sprzeczność.

14f: Język L_6 jest bezkontekstowy, bo można go zdefiniować gramatyką:

$$\xi_0 := a\xi_0 a \mid \xi_1, \qquad \xi_1 := \varepsilon \mid b\xi_1\,.$$

Nie jest regularny, bo ma nieskończenie wiele różnych ilorazów, np. $L_6\backslash a^k = \{a^pb^na^{p+k} \mid p, k, n \in \mathbb{N}\}$.

14g: Ten język jest regularny, rozpoznaje go poniższy automat skończony, w którym 0 jest stanem początkowym i jedynym stanem akceptującym.



14h: Ten język jest bezkontekstowy. Rozpoznaje go automat ze stosem, który po przeczytaniu każdego prefiksu v słowa w:
- przechowuje na stosie słowo postaci $x^{|N|}$, gdzie $N = \#(a, v) - 3 \cdot \#(b, v)$;
- znajduje się w stanie „plus" gdy $N > 0$, w stanie „zero", gdy $N = 0$, a w stanie „minus", gdy $N < 0$.

Po przeczytaniu kolejnej litery, automat aktualizuje swój stan i zawartość stosu. Automat akceptuje w stanie „zero" (wtedy stos powinien być pusty).

Język L_8 nie jest regularny, bo ma nieskończenie wiele różnych ilorazów. Na przykład, dla różnych $r \in \mathbb{N}$ różne są języki $L_2 \setminus a^r = \{w \mid a^r w \in L_2\} = \{w \in \{a, b\}^* \mid \#(a, w) + r = 3 \cdot \#(b, w)\}$.

14i: Ten język nie jest bezkontekstowy. Przypuśćmy przeciwnie i niech N będzie stałą z lematu o pompowaniu. Słowo $w = a^{3^N}b^N$ można przedstawić w postaci $w = uvzxy$, gdzie $uv^izx^iy \in L_9$ dla

każdego $i \in \mathbb{N}$. Przy tym $vx \neq \varepsilon$, a słowo vzx jest długości co najwyżej N.

Niech teraz $m = \#(a, vx)$ oraz $k = \#(b, vx)$. Ponieważ $uv^2zx^2y \in L_9$ i $uv^3zx^3y \in L_9$, więc muszą zachodzić równości $3^N + m = 3^{N+k}$ i $3^N + 2m = 3^{N+2k}$. Stąd mamy $m = 3^{N+k} - 3^N = 3^N(3^k - 1)$ oraz $m = 3^{N+2k} - 3^k - m = 3^{N+2k} - 3^{N+k} = 3^N(3^{2k} - 3^k)$, a więc $3^k - 1 = 3^{2k} - 3^k$. To zachodzi tylko dla $k = 0$, ale wtedy $m \neq 0$, więc $3^N + m \neq 3^{N+k}$, sprzeczność.[33]

14j: Język L_{10} jest regularny, bo to po prostu $a^*(ba)^*b^*$.

14k: Język L_{11} jest bezkontekstowy. Generuje go gramatyka o produkcjach postaci $\xi_0 ::= a\xi_0 b \mid \xi_1$ oraz $\xi_1 ::= \varepsilon \mid ba\xi_1$, gdzie ξ_0 jest symbolem początkowym. Nie jest to jezyk regularny, bo ma nieskończenie wiele różnych ilorazów postaci $L_2 \backslash a^n = \{a^i(ba)^j b^{i+n} \mid i, j \in \mathbb{N}\}$.

14l: Nie jest to język bezkontekstowy, bo nie spełnia lematu o pompowaniu. W przeciwnym razie niech N będzie stałą z lematu o pompowaniu i niech $w = a^N(ba)^N b^N$. Wtedy $w = xyzuv$, gdzie $|yzu| \leq N$, $yu \neq \varepsilon$, oraz $xy^r zu^r v \in L_{12}$ dla dowolnego r. Słowo yzu jest tak krótkie, że albo część początkowa a^N słowa w jest prefiksem słowa x albo część końcowa b^N jest sufiksem słowa v. W pierwszym przypadku słowo $xy^{2017}zu^{2017}v$ ma postać $a^N bW$ przy czym słowo bW ma albo za dużo liter b w części końcowej, albo za dużo segmentów ba w części środkowej, albo nie należy nawet do L_{10}. Drugi przypadek jest analogiczny i mamy sprzeczność.

14m: Język L_{13} jest bezkontekstowy. Generuje go gramatyka o produkcjach $\xi_0 ::= b \mid a\xi_0 b \mid a\xi_0 a$.

Nie jest to język regularny, bo ma nieskończenie wiele ilorazów. Istotnie, wszystkie języki postaci $L_{13} \backslash a^n = \{a^m bw \mid m \in \mathbb{N} \wedge |w| = n + m\}$ są różne.

14n: Język L_{14} jest regularny, bo $L_{14} = a^*b(a \cup b)^4$. W szczególności L_{14} jest bezkontekstowy.

14o: Język L_{15} nie jest bezkontekstowy. Wystarczy pokazać, że $L' = L_{15} \cap a^*b^* = \{a^n b^{n^2+1} \mid n \in \mathbb{N}\}$ nie jest bezkontekstowy (bo iloczyn języka bezkontekstowego i regularnego jest bezkontekstowy). Załóżmy przeciwnie. Niech N będzie stałą dla języka L' z lematu o pompowaniu i niech $w = a^N b^{N^2+1}$. Wtedy $w = xyzuv$, gdzie $|yzu| \leq N$, $yu \neq \varepsilon$, oraz $w_r = xy^r zu^r v \in L'$ dla dowolnego r. Niech k i m będzie odpowiednio liczbą liter a i b w słowie yu. Dla $r = 2$ otrzymujemy, że słowo w_2 ma $N + k$ liter a i $N^2 + 1 + m$ liter b. Natomiast przy $r = 3$ mamy $N + 2k$ liter a i $N^2 + 1 + 2m$ liter b w słowie w_3. Ponieważ $w_2, w_3 \in L'_3$, więc $(N + k)^2 + 1 = N^2 + 1 + m$ oraz $(N + 2k)^2 + 1 = N^2 + 1 + 2m$. Z tych równań łatwo wynika, że $m = k = 0$, czyli $yu = \varepsilon$ i mamy sprzeczność.

14p: Język L_{16} jest regularny, bo to po prostu język a^*b^*. Skoro regularny, to i bezkontekstowy.

14q: Język L_c nie jest bezkontekstowy. Przypuśćmy, że jest i niech N będzie stałą z lematu o pompowaniu. Rozpatrzmy słowo $a^n b^m a^n b^m$, gdzie $n, m > N$. Każdą z części postaci a^n, b^m nazwijmy *segmentem*. Jeśli rozbijemy nasze słowo na pięć części $xyzuv$, w ten sposób, że $|yzu| \leq N$ to podsłowo yzu zawiera się w całości w co najwyżej dwóch sąsiednich segmentach. W każdym przypadku usunięcie części y i u spowoduje zmniejszenie długości jednego segmentu postaci a^n lub jednego segmentu b^m, ale nie drugiego takiego segmentu. Słowo xzv nie należy więc do języka L_3.

15: Tak. Na przykład do sprawdzenia, czy dane słowo w długości n należy do L_2 wystarczy licznik rozmiaru n (zajmujący na taśmie roboczej $\log n$ komórek). Maszyna Turinga przesuwa głowicę wejściową w prawo, zwiększając w każdym kroku licznik o 1, aż do pierwszej litery b (jeśli jej nie ma, to jest jeszcze łatwiej, bo wystarczy sprawdzić czy słowo ma parzystą długość), a następnie zmniejszając licznik, który powinien osiągnąć wartość 0 na końcu słowa. (Oczywiście maszyna sprawdza też, że słowo należy do $a^*b^*a^*$, ale do tego wystarczą stany wewnętrzne.)

Język L_3 rozpoznajemy podobnie, używając dwóch liczników, a język L_1 w ogóle nie wymaga licznika, bo wystarczy automat skończony.

Język L_9 jest w klasie LOGSPACE, bo teraz też potrzebny jest tylko licznik, przyjmujący wartości nie większe niż długość słowa wejściowego. (Do policzenia liter a można użyć licznika ternarnego, który powinien być postaci „100 . . . 0", gdzie liczba zer jest równa liczbie wystąpień litery b.)

Ponieważ klasa P zawiera klasę LOGSPACE, więc nasze języki są też w P. W przypadku języków $L_1, L_2, L_4, L_5, L_6, L_7, L_8, L_{10}, L_{11}, L_{13}, L_{14} i L_{16}$, można to uzasadnić jeszcze łatwiej, bo wszystkie języki

[33] Jeszcze prościej: ponieważ $|vzx| \leq N$, więc $m, k \leq N$. Równość $m = 3^N(3^k - 1)$ może więc zajść tylko dla $m = k = 0$, co oznacza, że $vx = \varepsilon$.

bezkontekstowe, w tym regularne, są w P.

16a: Język A jest regularny (a zatem także bezkontekstowy), bo $A = A_0\$A_0 \cup A_1\$A_1 \cup A_2\$A_2$, gdzie $A_i = \{w \in \{0,1\}^* \mid \#(1,w) \bmod 3 = i\}$, a każdy z tych trzech składników jest regularny, bo: $A_0 = (0^*10^*10^*1)^*0^*$, $A_1 = (0^*10^*10^*1)^*0^*10^*$, $A_2 = (0^*10^*10^*1)^*0^*10^*10^*$.

16b: Język B nie jest regularny, bo nie spełnia lematu o pompowaniu dla języków regularnych. Przypuśćmy bowiem, że N jest stałą dobraną dla tego języka z lematu o pompowaniu. Jeśli słowo $1^N\$1^N$ przedstawimy w postaci xyz, gdzie $|xy| \le N$, $y \neq \varepsilon$, to słowo $xz = 1^{N-|y|}\$1^N$ powinno należeć do B, ale tak nie jest. Ale B jest bezkontekstowy, bo jest generowany przez gramatykę $\xi_0 ::= \$ \mid \xi_0 0 \mid 0\xi_0 \mid 1\xi_0 1$.

16c: Język C nie jest bezkontekstowy, bo nie spełnia lematu o pompowaniu dla języków bezkontekstowych. Jeśli N jest stałą z lematu, to słowo $1^N\$1^{N^2}$ możemy zapisać jako $xyzuv$, gdzie $|yzu| \le N$ i $yu \neq \varepsilon$. Jeśli symbol \$ występuje w yu, to oczywiście $xzv \notin C$, więc słowa y i u składają się z samych jedynek. Niech k będzie liczbą jedynek występujących w yu przed dolarem, a ℓ liczbą jedynek występujących w yu za dolarem. Wtedy słowo xzv ma postać $1^{N-k}\$1^{N^2-\ell}$, a słowo xy^2zu^2v ma postać $1^{N+k}\$1^{N^2+\ell}$. Jeśli oba te słowa mają należeć do C, to muszą zachodzić równości $(N-k)^2 = N^2 - \ell$ i $(N+k)^2 = N^2 + \ell$. Dodając je stronami, otrzymujemy $2N^2 + 2k^2 = 2N^2$, czyli $2k^2 = 0$. A więc $k = 0$, skąd także $\ell = 0$. Ale $yu \neq \varepsilon$, więc $k + \ell > 0$ i mamy sprzeczność.

17a: Język X jest bezkontekstowy. Generuje go na przykład taka gramatyka:

$$\xi_0 ::= \xi_1\xi_2, \quad \xi_1 ::= \varepsilon \mid a\xi_1 b, \quad \xi_2 ::= \varepsilon \mid b\xi_1 c.$$

Ale nie jest regularny, bo ma nieskończenie wiele różnych ilorazów, np. $X \setminus b^i = \{b^x c^{x+i} \mid x \in \mathbb{N}\}$.

17b: Język Y nie jest bezkontekstowy, bo nie spełnia lematu o pompowaniu. W przeciwnym razie niech $w = a^N b^{N^2} c^N$, gdzie N jest stałą z lematu. Jeśli $w = xyzuv$, gdzie $|yzu| \le N$, to obie części pompowane y i u zawierają się albo w prefiksie $a^N b^{N^2}$ albo w sufiksie $b^{N^2} c^N$. Przypuśćmy, że zachodzi pierwszy przypadek (drugi jest analogiczny) i załóżmy, że w słowach y i u jest łącznie p liter a oraz łącznie q liter b. Ponieważ słowo xy^2zu^2v powinno należeć do Y, więc mamy równość $(N+p)N = N^2 + q$, skąd $pN = q$. To niemożliwe, bo $p + q \le N$.

17c: Język Z jest regularny, bo $Z = a^*b^* \cup b^*c^*$.

18a: Elementy L_1 to wszystkie słowa postaci $x_1x_2\ldots x_n\overline{x}_n\ldots\overline{x}_2\overline{x}_1$, gdzie $x_1, x_2, \ldots, x_n \in \{a,b\}$. Ten język jest bezkontekstowy. Generuje go taka gramatyka: $\xi_0 ::= \varepsilon \mid a\xi_0 b \mid b\xi_0 a$. Ale nie jest regularny. W przeciwnym razie iloczyn $L_1' = L_1 \cap a^+bab^+ = \{a^nbab^n \mid n,k > 0\}$ też byłby regularny. A ten jezyk nie da się pompować: jeśli N jest stałą z lematu o pompowaniu, to trzeba podzielić słowo a^Nbab^N na trzy części x, y, z i to tak że $|xy| \le N$, czyli segment y ma postać a^d, dla pewnego $d > 0$. Wtedy $xy^2z = a^{N+d}bab^N \notin L_1'$, bo liczba liter w pierwszej i ostatniej części się nie zgadza.

18b: Ten język nie jest bezkontekstowy. Gdyby był, to język

$$L_2' = L_1 \cap a^+b^+a^+b^+a^+b^+ = \{a^nb^ma^kb^na^mb^k \mid n,m,k > 0\}$$

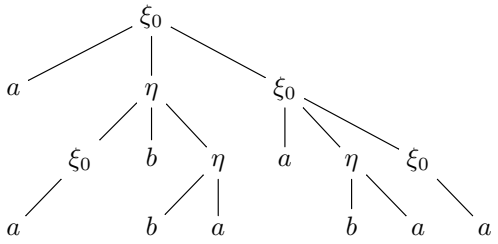
też byłby bezkontekstowy, a nie jest. Użyjemy lematu o pompowaniu: niech N będzie stałą z lematu i niech $a^Nb^Na^Nb^Na^Nb^N = xyzuv$, gdzie $|yzu| \le N$, $yu \neq \varepsilon$. Słowo yzu mieści w jednym z segmentów postaci a^Nb^N lub postaci b^Na^N. Przy tym słowo y zbudowane jest z samych a lub samych b i tak samo słowo u – inaczej słowo xy^2zu^2v ma za dużo alternacji liter i nie należy do L_2'. Ale wtedy $xzv \notin L_2'$, bo któraś z sześciu części jest w nim za krótka.

19: Inkluzja z lewej do prawej wynika z równości: $\overline{w^R\overline{w}} = \overline{w^R}\overline{\overline{w}} = (\overline{w})^R w = \overline{w}^R w^{RR} = (w^R\overline{w})^R$. Aby uzasadnić inkluzję z prawej do lewej, udowodnimy przez indukcję ze względu na $|w|$, że jeśli $\overline{w} = w^R$, to $w \in L_1$. Jeśli $w = \varepsilon$, to $w = \varepsilon^R\overline{\varepsilon} \in L_1$, więc załóżmy, że $|w| > 0$, na przykład $w = va$. Wtedy $w^R = av^R = \overline{w} = \overline{v}b$. Zatem w^R kończy się na literę b, skąd w zaczyna się od tej litery. To znaczy, że słowo w jest postaci bxa i równanie $\overline{w} = w^R$ można przepisać jako $a\overline{x}b = ax^Rb$. Stąd $\overline{x} = x^R$, więc z założenia indukcyjnego $x \in L_1$, czyli $x = y^R\overline{y}$, dla pewnego słowa y. Ale wtedy $w = bxa = by^R\overline{y}a = (yb)^R\overline{yb} \in L_1$.

20: Język L_1 należy do klasy $\textsc{Dtime}(\mathcal{O}(n))$, bo można go rozpoznawać maszyną, która czytając słowo wejściowe w zapisuje $\overline{w}$ na taśmie roboczej, a następnie porównuje taśmy czytając je w przeciwne strony (z zadania 19 wiemy, że $L_1 = \{w \mid \overline{w} = w^R\}$). Możemy też zauważyć, że $L_1 \in \textsc{Logspace}$, bo zamiast słowo kopiować (co wymaga pamięci liniowej) można porównywać poszczególne litery, posługując się

licznikami rozmiaru $\log n$. Podobnie jak L_1, także język L_2 należy do klas $\textsc{Dtime}(\mathcal{O}(n))$ i $\textsc{Logspace}$. Tym razem algorytmy są nieco bardziej kłopotliwe, trzeba bowiem (deterministycznie) odnaleźć środek słowa (czyli obliczyć $\frac{1}{2}n$) i porównywać połówki czytając obie od lewej do prawej. Liczbę $\frac{1}{2}n$ można jednak ustalić z pomocą jednego licznika w czasie liniowym: czytamy słowo wejściowe, dodając jedynkę do licznika co dwa kroki. Skoro języki L_1 i L_2 mają określoną złożoność, to oczywiście są rozstrzygalne. Gdyby któryś z nich okazał się zupełny w klasie P lub NP (czego nie wiadomo), to wtedy cała klasa P (odpowiednio NP) byłaby równa klasie $\textsc{Logspace}$. Ale nasze języki na pewno nie są zupełne w $\textsc{Pspace}$, bo wiadomo, że $\textsc{Logspace} \neq \textsc{Pspace}$.

22: Drzewo dla słowa *aabbaabaa*:



27: Symbolem początkowym gramatyki jest WN. Produkcje są następujące:

$WN \Rightarrow JN \mid WP + JN \mid WN + JP$;
$WP \Rightarrow JP \mid WP + JP \mid WN + JN$;
$JP \Rightarrow CP \mid J \cdot CP \mid JP \cdot C$;
$JN \Rightarrow CN \mid JN \cdot CN$;
$C \Rightarrow CP \mid CN$;
$J \Rightarrow JP \mid JN$;
$CP \Rightarrow SP \mid (WP + JP) \mid (WN + JN)$;
$CN \Rightarrow SN \mid (WN + JP) \mid (WP + JN)$;
$SN \Rightarrow 1 \mid 3$;
$SP \Rightarrow 0$.

28: Symbolem początkowym gramatyki jest WP. Produkcje są następujące:

$WP \Rightarrow JP \mid WP + JP \mid WN + JN$;
$WN \Rightarrow JN \mid WP + JN \mid WN + JP$;
$JP \Rightarrow SP \mid CP \cdot C \mid C \cdot CP \mid J \cdot CP \mid JP \cdot C$;
$C \Rightarrow CP \mid CN$;
$J \Rightarrow JP \mid JN$;
$JN \Rightarrow SN \mid CN \cdot CN \mid JN \cdot CN$;
$CP \Rightarrow SP \mid (WP)$;
$CN \Rightarrow SN \mid (WN)$;
$SP \Rightarrow 0 \mid 2$;
$SN \Rightarrow 1$.

29: Gramatyka ma nieterminały **0**, **1**, **2**, **3**, **X**, a symbolem początkowym jest **X**. Produkcje są takie:

$\mathbf{3} \Rightarrow (\mathbf{0} + \mathbf{3}) \mid (\mathbf{3} + \mathbf{0}) \mid (\mathbf{1} + \mathbf{2}) \mid (\mathbf{2} + \mathbf{1}) \mid (\mathbf{1} \cdot \mathbf{3}) \mid (\mathbf{3} \cdot \mathbf{1})$
$\mathbf{2} \Rightarrow (\mathbf{0} + \mathbf{2}) \mid (\mathbf{2} + \mathbf{0}) \mid (\mathbf{1} + \mathbf{1}) \mid (\mathbf{1} + \mathbf{1}) \mid (\mathbf{1} \cdot \mathbf{2}) \mid (\mathbf{2} \cdot \mathbf{1})$
$\mathbf{1} \Rightarrow (\mathbf{0} + \mathbf{1}) \mid (\mathbf{1} + \mathbf{0}) \mid (\mathbf{1} \cdot \mathbf{1}) \mid (\mathbf{1} \cdot \mathbf{1}) \mid 1$
$\mathbf{0} \Rightarrow (\mathbf{0} + \mathbf{0}) \mid (\mathbf{0} \cdot \mathbf{X}) \mid (\mathbf{X} \cdot \mathbf{0}) \mid 0$
$\mathbf{X} \Rightarrow (\mathbf{X} + \mathbf{X}) \mid (\mathbf{X} \cdot \mathbf{X}) \mid 0 \mid 1$

36: Przypuśćmy, że ten język jest bezkontekstowy i niech N będzie stałą z lematu o pompowaniu. Zastosujemy lemat do słowa $w = a^N b^N a^N b^N$. Powinno istnieć takie rozbicie tego słowa $w = xyzuv$, gdzie yu jest niepuste, oraz wszystkie słowa $xy^i zu^i v$ należą do naszego języka. Przy tym długość podsłowa yzu nie przekracza N.

Przypadek 1: Słowo yzu jest w całości zawarte w początkowym fragmencie a^N słowa w. Wtedy $xy^2zu^2v = a^M b a^N b^N$, gdzie $M > N$, a zatem $xy^2zu^2v \notin L$, sprzeczność.

Przypadek 2: Słowo yzu zawiera pierwsze wystąpienie litery b i jest w całości zawarte we fragmencie $a^N ba^N$. Zależnie od tego czy litera b występuje w słowie z czy w yu, mamy albo $xy^2zu^2v = a^M ba^K b^N$, gdzie $M, K > N$, albo $xy^2zu^2v = a^M ba^K ba^L b^N$, dla pewnych M, K, L, gdzie na pewno $L \neq 0$. Tak czy owak, znowu $xy^2zu^2v \notin L$.

Przypadek 3: Słowo yzu jest w całości zawarte w końcowym odcinku postaci $a^N b^N$. Wtedy łączna liczba liter a i b występujących w słowie xy^2zu^2v na prawo od pierwszego b jest większa niż $2N$, zatem znowu $xy^2zu^2v \notin L$.

Następująca gramatyka typu zero generuje język L:

$$\xi_0 \Rightarrow a\eta b\xi_0, \quad \xi_0 \Rightarrow \varepsilon, \quad \eta a \Rightarrow a\eta, \quad ba \Rightarrow ab, \quad b\eta \Rightarrow \eta b, \quad a\eta \Rightarrow ab\nu, \quad \nu\eta \Rightarrow a\nu, \quad \nu b \Rightarrow ab.$$

Gramatyka ta nie jest kontekstowa ani nawet monotoniczna. Język L jest jednak kontekstowy, bo jest akceptowany w pamięci liniowej (a nawet logarytmicznej) przez maszynę Turinga, która porównuje liczbę liter w poszczególnych segmentach za pomocą 2 liczników.

41a: Ten problem jest oczywiście rozstrzygalny, bo każdy język akceptowany przez maszynę Turinga jest z definicji częściowo obliczalny. Odpowiedź na pytanie 41a jest zawsze „tak".

41b: Ten problem jest nierozstrzygalny, bo redukuje się do niego problem stopu. Dla danej maszyny M i słowa w można bowiem skonstruować maszynę $N_{M,w}$, która dla dowolnego wejścia v:

1. uruchamia maszynę M na wejściu w i jeśli to obliczenie się zakończy, to
2. uruchamia automat skończony sprawdzający, czy $v \in (aabbb)^*$.

Maszyna M akceptuje w wtedy i tylko wtedy, gdy $L(M) = (aabbb)^*$, (w przeciwnym razie $L(M) = \varnothing$). Gdyby problem 41b był rozstrzygalny, to problem stopu też byłby rozstrzygalny, a to nieprawda.

43a: Ten problem jest nierozstrzygalny, bo redukuje się do niego dopełnienie problemu stopu. Dla danej maszyny M i słowa w można bowiem skonstruować maszynę $N_{M,w}$, która dla każdego wejścia v:

1. uruchamia maszynę M na wejściu w i jeśli to obliczenie się zakończy, to:
2. sprawdza, czy $v = b$.

Jeśli maszyna M akceptuje w, to $L(N_{M,w}) = \{b\}$, w przeciwnym razie $L(N_{M,w}) = \varnothing$. Zatem:

$$w \in L(M) \quad \text{wtedy i tylko wtedy, gdy} \quad L(N_{M,w}) \not\subseteq a^*.$$

Gdyby problem 43a był rozstrzygalny, to problem stopu też byłby rozstrzygalny, a to nieprawda.

43b: Ten problem jest rozstrzygalny. Wystarczy sprawdzić, czy w grafie automatu A wszystkie drogi od stanu początkowego do końcowego składają się wyłącznie z krawędzi oznaczonych literą a (i żadną inną literą). Inaczej: po usunięciu przejść typu a stan końcowy nie jest osiągalny z początkowego. To zadanie jest dualne do problemu osiągalności, więc należy do klasy *co*-NLOGSPACE= NLOGSPACE.

44a: Ten problem jest nierozstrzygalny, bo można do niego sprowadzić problem stopu. Dla danej maszyny Turinga M i słowa w, konstruujemy maszynę $T_{M,w}$, która (ignorując swoje słowo wejściowe x) symuluje zachowanie maszyny M dla wejścia w. Język $L(T_{M,w})$ rozpoznawany przez maszynę $T_{M,w}$ jest wtedy albo pusty (gdy M nie akceptuje słowa w) albo pełny (gdy M akceptuje słowo w). W szczególności mamy taką równoważność: M akceptuje słowo w wtedy i tylko wtedy, gdy $T_{M,w}$ akceptuje wszystkie słowa długości 28.

44b: Alfabet wejściowy każdej maszyny Turinga jest skończony, zatem istnieje tylko skończenie wiele słów długości 28 nad tym alfabetem. Mamy więc częściowy algorytm dla naszego problemu: trzeba po prostu uruchamiać maszynę kolejno dla każdego takiego słowa.

44c: Nie, bo języki z klasy NP są rozstrzygalne.

45a: Jeśli L jest językiem bezkontekstowym, to $L \cap a^*$ też jest językiem bezkontekstowym. Co więcej, daną gramatykę dla języka L można łatwo (w pamieci logarytmicznej) przerobić na gramatykę dla języka $L \cap a^*$. Wystarczy każdą literę terminalną inną niż a zastąpić przez nowy symbol nieterminalny $\perp$, dla którego nie ma żadnej produkcji. Nasze zadanie sprowadza się więc do problemu niepustości dla języków bezkontekstowych, o którym wiadomo, że jest rozstrzygalny.
Problem niepustości języka bezkontekstowego należy do klasy P. A zatem możliwe, że ten problem jest NP-zupełny, ale wtedy P = NP. Wielomianowy algorytm dla niepustości można opisać tak: dla danej gramatyki $G = \langle \mathcal{A}, \mathcal{N}, P, \xi_0 \rangle$ konstruujemy zbiór $D = \{\xi \in \mathcal{N} \mid \exists k \in \mathbb{N}(\xi \twoheadrightarrow a^k)\}$, jako sumę wstępującego ciągu $D_0 \subseteq D_1 \subseteq D_2 \ldots$. Zaczynamy od $D_0 = \varnothing$, a potem ze zbioru D_i tworzymy $D_{i+1} = D_i \cup \{\xi \in \mathcal{N} \mid \exists w(w \in (D_n \cup a)^* \wedge (\xi \Rightarrow w) \in P)\}$. Każdy taki krok wymaga czasu

proporcjonalnego do rozmiaru gramatyki, a liczba kroków też jest tego rzędu, bo dla pewnego $i \leq |\mathcal{N}|$ dostaniemy $D = D_i = D_{i+1}$. Na koniec sprawdzamy, czy $\xi_0 \in D$ i już.

45b: Z tym gorzej: to jest problem nierozstrzygalny. Można do niego sprowadzić problem stopu. Dla danej deterministycznej maszyny $\mathcal{M}$ i słowa wejściowego w skonstruujemy bowiem maszynę $\mathcal{N}_{M,w}$ w ten sposób, że $w \in L(\mathcal{M})$ wtedy i tylko wtedy, gdy maszyna $\mathcal{N}_{M,w}$ akceptuje jakieś słowo złożone z samych liter a. Maszyna $\mathcal{N}_{M,w}$ dla dowolnego wejścia $v \in \Sigma^*$ działa tak: uruchamia $\mathcal{M}$ na w i jeśli obliczenie się zakończy, to akceptuje. A zatem:
– Jeśli $w \in L(\mathcal{M})$, to $L(\mathcal{N}_{M,w}) = \Sigma^*$ (wtedy np. $aa \in L(\mathcal{N}_{M,w})$);
– Jeśli $w \notin L(\mathcal{M})$, to $L(\mathcal{N}_{M,w}) = \varnothing$ (wtedy do $L(\mathcal{N}_{M,w})$ nie należy żadne słowo $aa\ldots a$).
Skoro problem 45b jest nierozstrzygalny, to w szczególności nie należy do klasy NP i nie może być NP-zupełny. Ale jest częściowo rozstrzygalny, bo jeśli dana maszyna akceptuje jakieś słowo postaci $aa\ldots a$, to można to wykryć, systematycznie przeglądając początkowe fragmenty wszystkich możliwych obliczeń dla takich słów wejściowych, aż nie natrafi się na obliczenie akceptujące.

47: Obraz $f(A) = \{f(a) \mid a \in A\}$ jest zbiorem rekurencyjnie przeliczalnym (częściowo obliczalnym), bo istnieje częściowy algorytm sprawdzający, że dana liczba $n \in \mathbb{N}$ należy do $f(A)$. Dla $a = 0, 1, 2, \ldots$ wystarczy sprawdzać, czy $a \in A$ i czy $f(a) = n$, aż się takie a znajdzie. Obraz zbioru obliczalnego nie musi być obliczalny. (W istocie, każdy niepusty zbiór rekurencyjnie przeliczalny jest obrazem całego zbioru $\mathbb{N}$ przy pewnej funkcji obliczalnej.) Na przykład niech $k \in K = \{m \mid M_m \text{ akceptuje } m\}$ i niech $f(m,n) = m$, jeśli maszyna M_m akceptuje m w co najwyżej n krokach, a w przeciwnym razie niech $f(m,n) = k$. Wtedy $f(\mathbb{N} \times \mathbb{N}) = K$, a przecież zbiór K to nic innego, tylko problem stopu. Dwuargumentową funkcję f można zastąpić przez jednoargumentową funkcję $f \circ g$, gdzie $g : \mathbb{N} \xrightarrow[\text{na}]{1-1} \mathbb{N} \times \mathbb{N}$ jest obliczalna, np. taka że $g(2^m(2n+1)-1) = \langle m, n\rangle$.

Przeciwobraz $f^{-1}(A) = \{x \in \mathbb{N} \mid f(x) \in A\}$ jest obliczalny, bo aby ustalić czy $x \in f^{-1}(A)$ wystarczy obliczyć $f(x)$ i sprawdzić, czy $f(x) \in A$.

52: Ten język należy do klasy P a nawet do klasy Log. Żeby sprawdzić, czy słowo jest postaci ww^Rw można policzyć jego długość (musi to być liczba postaci $3k$) a następnie użyć 3 liczników zmieniających się odpowiednio od 1 do k, od $2k$ do $k+1$ i od $2k+1$ do $3k$, żeby sprawdzić czy symbole o odpowiednich numerach są takie same. Całość wymaga czasu $\mathcal{O}(n^2)$ i pamięci logarytmicznej.

Ale to nie jest język bezkontekstowy. Można to pokazać tak: gdyby L był bezkontekstowy, to także język $L_1 = L \cap a^*b^*a^*b^* = \{a^n b^{2n} a^{2n} b^n \mid n \in \mathbb{N}\}$ byłby bezkontekstowy. Do języka L_1 zastosujemy lemat o pompowaniu: niech N będzie odpowiednią stałą. Rozpatrzmy słowo $a^{2N}b^{4N}a^{4N}b^{2N} \in L_1$. Powinniśmy je podzielić na 5 cześci $uwvxy$, w ten sposób, że $wx \neq \varepsilon$, $|wvx| \leq N$ oraz (między innymi) słowo uvy należy do L_1.

Słowo wvx jest krótkie, więc części w i x muszą zawierać się w sąsiednich segmentach złożonych z liter a i b (lub odwrotnie – z liter b i a). Po usunięciu w i x liczba liter a i b w tych segmentach będzie za mała w stosunku do pozostałych segmentów, słowo uvy nie może więc być elementem języka L_1.

53: Wszystkie trzy języki są oczywiście rozstrzygalne i mieszczą się w klasie Logspace. Do ich rozpoznawania potrzebne są dwa liczniki służące do zliczania liter a i b. Wartości liczników nie przekraczają długości słowa wejściowego, wystarczy więc logarytmiczna pamięć.

53a: Język L_1 jest regularny, bo to po prostu język $a^*b^*a^*b^*$.

53b: Język L_2 nie jest regularny, bo ma nieskończenie wiele różnych ilorazów, w tym np. języki:

$$L_2 \backslash a^d = \{a^m b^n a^k b^\ell \mid m, n, k, \ell \in \mathbb{N} \wedge m + d \leq \ell \wedge n \leq k\},$$

dla dowolnego $d \in \mathbb{N}$. Jest to język bezkontekstowy generowany przez gramatykę

$$\xi_0 ::= a\xi_0 b \mid \xi_1, \quad \xi_1 ::= \xi_1 b \mid \xi_2, \quad \xi_2 ::= b\xi_2 a \mid \xi_3, \quad \xi_3 ::= \xi_3 a \mid \varepsilon$$

53c: Ten język nie jest bezkontekstowy. W przeciwnym razie niech N będzie stałą z lematu o pompowaniu i niech $w = a^N b^N a^N b^N$. Ponieważ $w \in L_3$ i $|w| \geq N$, więc $w = xyzuv$ dla pewnych słów x, y, z, u, v, gdzie $|yzu| \leq N$, $yu \neq \varepsilon$ oraz każde słowo $xy^k zu^k v$ należy do L_3. Przypuśćmy, że w słowie yu jest choć jedna litera b (w przeciwnym razie jest jakieś a i rozumujemy podobnie). Ponieważ fragment yzu jest krótki, więc wszystkie litery b w słowie yu pochodzą z tego samego segmentu złożonego z N liter b. Jeśli jest to pierwszy segment, to słowo xy^2zu^2v ma za dużo liter b w pierwszej części aby należało do L_3. A jeśli jest to drugi segment, to słowo xy^0zu^0v, czyli słowo xzv ma w drugiej części

za mało liter b. W obu przypadkach mamy sprzeczność z lematem o pompowaniu.

65: Ponieważ CSL = $\textsc{Nspace}(n) \subseteq \textsc{Dspace}(n^2)$ z twierdzenia Savitcha, więc tym bardziej zachodzi pierwsza inkluzja w zadaniu. Druga nie zachodzi, bo z tegoż twierdzenia Savitcha i z tego, że $\lim_{n\to\infty} \frac{(\log\log n)^2}{\log n} = 0$ wynika, że $\textsc{Nspace}(n\log\log n) \subseteq \textsc{Dspace}(n^2(\log\log n)^2) \subsetneq \textsc{Dspace}(n^2\log n)$.

Pozostaje pokazać, że jeśli $\textsc{Nspace}(n\log\log n) \subseteq \mathrm{P}$, to $\textsc{Pspace} = \mathrm{P}$. Niech więc $L \in \textsc{Pspace}$, powiedzmy, że $L \in \textsc{Nspace}(n^k)$. „Wypychamy" język L, aby dostać język L' należący do klasy $\textsc{Nspace}(n\log\log n)$. Można to zrobić tak: $L' = \{w\$...\$ \mid w \in L \text{ oraz } |w\$...\$| = |w|^k\}$. Maszynę, która rozpoznawała język L w pamięci n^k można łatwo przerobić na maszynę rozpoznającą L' w pamięci liniowej (liczonej od $N = n^k$). Zatem $L' \in \textsc{Nspace}(n) \subseteq \textsc{Nspace}(n\log\log n) \subseteq \mathrm{P}$, czyli mamy deterministyczną maszynę rozpoznającą język L' w czasie wielomianowym, powiedzmy n^r. Pozostaje ją przerobić na taką maszynę M, że $L = L(M)$, która działa w wielomianowym czasie $N^r = n^{kr}$.

66g: Tak, bo $\textsc{Dspace}(3n) = \textsc{Dspace}(2n) \subseteq \textsc{Nspace}(2n)$.

66h: Nie, bo iloraz $2^{2n}/2^{3n} = 1/2^n$ dąży do zera[34] przy $n \to \infty$, skąd $\textsc{Dspace}(2^{2n}) \subsetneq \textsc{Dspace}(2^{3n})$.

66i: Nie, bo iloraz $2^{2n}\cdot\log(2^{2n})/2^{3n} = 2^{2n}\cdot 2n/2^{3n} = 2n/2^n$ dąży do zera przy $n \to \infty$, więc $\textsc{Dtime}(2^{2n}) \subsetneq \textsc{Dtime}(2^{3n})$.

66j: Tak, bo jeśli $L \in \textsc{Dspace}(3n)$, to $L \in \textsc{Dtime}(2^{c\cdot 3n})$ dla pewnego c, oraz $2^{c\cdot 3n} < 2^{n\log n} = n^n$ pw.

66k: Tak, ponieważ $\textsc{Nspace}(3n) \subseteq \textsc{Dtime}(2^{c\cdot 3n})$ też zachodzi.

66l: Nie, bo iloraz $2n\cdot\log 2n/n^3 < 2n(2+\log n)/n^3 < 2n^2/n^3 = 2/n$ dąży do zera przy $n \to \infty$, skąd $\textsc{Dtime}(2n) \subsetneq \textsc{Dtime}(n^3)$.

66m: Tak, ponieważ $\textsc{Ntime}(n\log n) \subseteq \bigcup_{c>0} \textsc{Dtime}(2^{c\cdot n\log n}) \subseteq \textsc{Dtime}(2^{n^2})$.

66n: Tak, ponieważ $\textsc{Nspace}(\log n) \subseteq \textsc{Dspace}(\log^2 n) \subseteq \textsc{Dspace}(n)$ na mocy tw. Savitcha.

66p: Tak, ponieważ $\textsc{Ntime}(n) \subseteq \textsc{Dtime}(2^{\mathcal{O}(n)}) \subseteq \textsc{Dtime}(2^{n\log\log n}) \subsetneq \textsc{Dtime}(n^n)$. Ostatnie zawieranie jest ostre, bo iloraz $2^{n\log\log n}\cdot n\log\log n/n^n$ dąży (dlaczego?) do zera przy $n \to \infty$.

67: Pierwsza tak, bo $\textsc{Nspace}(n\log n) \subseteq \textsc{Dspace}((n\log n)^2)$ oraz $(n\log n)^2 \le n^3$. Druga oczywiście tak, bo stała nie ma znaczenia. Trzecia też tak, bo $\textsc{Dspace}(n^3) \subseteq \bigcup\{\textsc{Dtime}(2^{cn^3}) \mid c > 0\}$, oraz $2^{cn^3} \le 2^{n^4}$ prawie wszędzie. Czwarta nie, bo iloraz wyrażenia $2^n\log(2^n) = n2^n$ przez 2^{n^4} zbiega w nieskończoności do zera.

68a: Tak, bo $\textsc{Nspace}(\log^k n) \subseteq \textsc{Dspace}(\log^{2k} n)$ z twierdzenia Savitcha.

68b: Tak, ponieważ $\textsc{Nspace}(\log^2 n) \subseteq \textsc{Dtime}(2^{\mathcal{O}(\log^2 n)}) = \textsc{Dtime}(n^{\mathcal{O}(\log n)}) \subseteq \textsc{Dtime}(n^{\log^2 n})$.

68c: Nie. Ponieważ $n^2 \le n^2\sqrt{n}$ oraz $\lim\limits_{n\to\infty} \dfrac{n^2\log n}{n^2\sqrt{n}} = 0$, więc $\textsc{Dtime}(n^2) \subsetneq \textsc{Dtime}(n^2\sqrt{n})$.

68d: Nie. Klasa $\textsc{Dspace}(n)$ zawiera się w klasie $\textsc{Dtime}(2^{\mathcal{O}(n)}) \subseteq \textsc{Dtime}(2^{n\cdot\log\log\log n})$. Tymczasem $\textsc{Dtime}((\log n)^n)$ to to samo co $\textsc{Dtime}(2^{n\log\log n})$. Pozostaje podzielić $2^{n\cdot\log\log\log n}\cdot n\cdot\log\log\log n$ przez $2^{n\log\log n}$) i zobaczyć, że iloraz maleje do zera.

69a: Niemożliwe. Przypuśćmy, że $\textsc{Dspace}(n^2) \subseteq \textsc{Nspace}(n)$. Pokażemy, że wtedy zachodzi równość $\textsc{Nspace}(n^2) = \textsc{Dspace}(n^4)$. Inkluzja $\subseteq$ wynika z twierdzenia Savitcha. Niech więc $L \in \textsc{Dspace}(n^4)$ i niech $L' = \{w\$^{|w|^2-|w|} \mid w \in L\}$. Zauważmy, że słowo $w\$^{|w|^2-|w|}$ ma długość $|w|^2$, a ponieważ L jest w klasie $\textsc{Dspace}(n^4)$, więc $L' \in \textsc{Dspace}(n^2)$. Z założenia wynika $L' \in \textsc{Nspace}(n)$, a stąd dostajemy $L \in \textsc{Nspace}(n^2)$.

69b: Możliwe, ale wtedy NP = P, bo języki bezkontekstowe są w klasie P.

69c: To nieprawda, bo zachodzi ostre zawieranie $\mathrm{P} \subsetneq \textsc{Dtime}(n^{\log n})$. Istotnie, każdy język z klasy P jest w pewnej klasie $\textsc{Dtime}(n^k) \subseteq \textsc{Dtime}(n^{\log\log n})$. Zatem $\mathrm{P} \subseteq \textsc{Dtime}(n^{\log\log n})$. Ale wyrażenie

$$\frac{n^{\log\log n}\cdot\log(n^{\log\log n})}{n^{\log n}} = \frac{2^{\log n\cdot\log\log n}\cdot\log n\cdot\log\log n}{2^{\log^2 n}} = 2^{\log n\cdot\log\log n+\log\log n+\log\log\log n-\log^2 n}$$

zbiega do zera przy $n \to \infty$, więc $\textsc{Dtime}(n^{\log\log n}) \subsetneq \textsc{Dtime}(n^{\log n})$.

69d: To możliwe. Jeśli to prawda, to wtedy na przykład $\textsc{NExptime} = \textsc{NExpspace}$.

[34] W tym zadaniu korzystamy z tego, że wszystkie rozważane funkcje są konstruowalne.

70a: To jest zawieranie właściwe. Mamy $\text{NSPACE}(\log n) \subseteq \text{DSPACE}(\log^2 n) \subseteq \text{DSPACE}(\sqrt{n})$, a ponieważ iloraz $\log^2 n/\sqrt{n}$ zbiega do zera, więc drugie zawieranie jest ostre.

70b: Tego nie wiadomo. Gdyby tak było, to PSPACE = NP. Weźmy bowiem jakiś język $L \in \text{PSPACE}$. Istnieje deterministyczna maszyna Turinga, rozpoznająca język L w pamięci n^k. Rozpatrzmy język $L' = \{w\$^{|w|^{2k}-|w|} \mid w \in L\}$. Elementami L' są słowa postaci $w\$...\$$, długości $|w|^{2k}$. Ten język należy do klasy $\text{DSPACE}(\sqrt{n})$, bo do rozpoznania czy dane słowo długości N ma postać $w\$...\$$, gdzie $w \in L$ oraz $N = |w|^{2k}$, maszyna Turinga użyje pamięci $|w|^k = \sqrt{N}$. Skoro jednak $L' \in \text{DSPACE}(\sqrt{n}) \subseteq \text{NP}$, to istnieje niedeterministyczna maszyna M rozpoznająca L' w wielomianowym czasie n^d, dla pewnego d. Tę maszynę M potrafi naśladować maszyna M', rozpoznająca L w wielomianowym czasie $(n^{2k})^d$. Zatem $L \in \text{NP}$.

70c: To nieprawda. Gdyby tak było, to $\text{NP} \subseteq \text{DTIME}(n^2) \subsetneq \text{DTIME}(n^3) \subseteq \text{P} \subseteq \text{NP}$, skąd wynika nonsensowne $\text{NP} \neq \text{NP}$. Inkluzję $\text{DTIME}(n^2) \subsetneq \text{DTIME}(n^3)$ uzasadniamy tym, że iloraz $n^2 \cdot \log(n^2)/n^3$ zbiega do zera przy $n \to \infty$.

71a: Wiadomo, że $\text{NTIME}(n \log n) \subseteq \text{DSPACE}(n \log n)$. A zatem inkluzja zachodzi, bo $n \log n \leq n\sqrt{n}$ i jest właściwa, bo $\lim\limits_{n\to\infty} \dfrac{n \log n}{n\sqrt{n}} = 0$.

71b: Wiadomo , że *co*-$\text{NSPACE}(n\sqrt{n}) = \text{NSPACE}(n\sqrt{n}) \subseteq \text{DSPACE}(n^3)$. Ponieważ $n^3 \leq n^4$, a granica ilorazu jest zerem, więc inkluzja zachodzi i jest właściwa.

71c: Ta inkluzja też zachodzi, ale jest niewłaściwa, bo klasy $\text{NSPACE}(3n^2 + 2)$ i $\text{NSPACE}(n^2)$ są identyczne, a na dodatek zamknięte ze względu na dopełnienie.

71d: Ostatnie zawieranie nie zachodzi: faktycznie mamy właściwą inkluzję w przeciwną stronę. A to dlatego, że $\text{NTIME}(n\sqrt{n}) \subseteq \text{DTIME}(2^{\mathcal{O}(n\sqrt{n})}) \subseteq \text{DTIME}(2^{n \log n\sqrt{n}})$. Ta ostatnia klasa jest ostro zawarta w $\text{DTIME}(3^{n^2})$ bo iloraz $\dfrac{2^{n \log n \cdot \sqrt{n}} \cdot n \log n\sqrt{n}}{3^{n^2}}$ dąży do zera.

72a: Tak, bo alternujące obliczenie długości n^2 może być implementowane sekwencyjnie z pomocą stosu o wysokości n^2.

72b: Tak, bo $\text{NC}_2 \subseteq \text{NC} \subseteq \text{P} \subseteq$ *co*-RP.

72c: Tak, bo wartość sieci o wysokości $\log n$ można obliczać rekurencyjnie, używając binarnego stosu o wysokości $\log n$.

72d: Nie, zob. twierdzenie 8.8.

72e: Tak, bo *co*-$\text{NSPACE}(\log n) = \text{NSPACE}(\log n)$, oraz $\log n \leq \log^2 n$.

76: Wskazówka: czy w klasie POLYLOG są problemy zupełne?

79: Przypuśćmy, że LOGSPACE = P i niech $L \in \text{EXPTIME}$. Istnieje deterministyczna maszyna Turinga, rozpoznającą język L w czasie wykładniczym, tj. w czasie $2^{p(n)}$, gdzie $p(n)$ jest pewnym wielomianem. Rozpatrzmy język $L_1 = \{w\$^{2^{p(|w|)}-|w|} \mid w \in L\}$. Jeśli teraz przyjmiemy $n = |w\$^{2^{p(|w|)}-|w|}|$, to $2^{p(|w|)} = n$, zatem język L_1 jest rozpoznawalny w deterministycznym czasie liniowym. Mamy więc $L_1 \in \text{P}$, a zatem także $L_1 \in \text{LOGSPACE}$. Istnieje więc maszyna Turinga $\mathcal{M}_1$, rozpoznająca język L_1 w pamięci $\mathcal{O}(\log n)$. Można ją łatwo przerobić na maszynę $\mathcal{M}$, rozpoznającą L w pamięci $\mathcal{O}(\log 2^{p(n)})$ czyli w pamieci wielomianowej. Maszyna $\mathcal{M}$ wykonuje te same czynności co maszyna $\mathcal{M}_1$. Jedyna różnica polega na tym, że inne jest słowo wejściowe. Zamiast czytać dolary ze słowa wejściowego, maszyna $\mathcal{M}$ pamięta więc tylko pozycję, w jakiej powinna się znajdować głowica maszyny $\mathcal{M}_1$.

80: Przypuśćmy, że P = PSPACE i niech $L \in \text{EXPSPACE}$. Mamy deterministyczną maszynę Turinga, rozpoznającą język L w pamięci wykładniczej, tj. w pamięci $2^{p(n)}$, gdzie $p(n)$ jest pewnym wielomianem. Rozpatrzmy język $L_1 = \{w\$^{2^{p(|w|)}-|w|} \mid w \in L\}$. Słowo postaci $w\$^{2^{p(|w|)}-|w|}$ ma długość $n = 2^{p(|w|)}$, zatem język L_1 jest rozpoznawalny w pamięci liniowej. Mamy więc $L_1 \in \text{PSPACE}$, a zatem także $L_1 \in \text{P}$. Istnieje więc maszyna Turinga $\mathcal{M}$, rozpoznająca język L_1 w czasie n^k. Można ją łatwo przerobić na maszynę $\mathcal{M}_1$, rozpoznającą L w czasie $(2^{p(n)})^k = 2^{k \cdot p(n)}$.

81: Załóżmy, że NLOGSPACE = LOGSPACE, i niech L będzie językiem kontekstowym, tj. niech $L \in \text{NSPACE}(n)$. Należy pokazać, że $L \in \text{DSPACE}(n)$.

Mamy niedeterministyczną maszynę Turinga M, rozpoznającą L w pamięci n. Łatwo ją przerobić na maszynę M' (nadal niedeterministyczną), która rozpoznaje język $L' = \{w\$^{2^{|w|}-|w|} \mid w \in L\}$ w pamięci logarytmicznej, bo długość słowa $|w|$ to logarytm z długości słowa $w\$^{2^{|w|}-|w|}$.

Zatem $L' \in$ NLOGSPACE, i z naszego założenia wynika, że $L' \in$ LOGSPACE. A więc istnieje też deterministyczna maszyna M'', rozpoznająca język L' w pamięci logarytmicznej. Tę maszynę przerabiamy na deterministyczną maszynę M''', która w pamięci liniowej rozpoznaje L. Maszyna M''' pracuje na słowie wejściowym w, naśladując działanie M'' na słowie $w\$^{2^{|w|}-|w|}$. Położenie głowicy M'' na taśmie wejściowej jest reprezentowane w maszynie M' za pomocą $|w|$ klatek taśmy roboczej.

82: Załóżmy, że NP = *co*-NP. Pokażemy, że NEXPTIME $\subseteq$ *co*-NEXPTIME. Niech $L \in$ NEXPTIME, tj. $L \in$ NTIME(2^{n^k}) dla pewnego k. Wtedy $L' = \{w\$^{2^{|w|^k}-|w|} \mid w \in L\}$ należy do NTIME$(\mathcal{O}(n)) \subseteq$ NP, ponieważ słowa postaci $w\$^{2^{|w|^k}-|w|}$ o długości n można rozpoznawać niedeterministycznie w czasie proporcjonalnym do $2^{|w|^k} = n$. Z naszego założenia wynika, że $L' \in$ *co*-NP, czyli $-L' \in$ NP, powiedzmy $-L' \in$ NTIME(n^r). Ale wtedy język $-L$ jest w klasie NEXPTIME. Istotnie, aby sprawdzić, że dane słowo w długości n należy do $-L$ wystarczy ustalić, że $w\$^{2^{|w|^k}-|w|} \in -L'$. To można zrobić niedeterministycznie w czasie wielomianowym od długości słowa $w\$^{2^{|w|^k}-|w|}$, dokładniej w czasie $(2^{n^k})^r = 2^{rn^k} \leq 2^{n^{k+1}}$ (prawie wszędzie). Zatem $-L \in$ NTIME$(2^{n^{k+1}})$, więc $L \in$ *co*-NEXPTIME.

Inkluzja *co*-NEXPTIME $\subseteq$ NEXPTIME wynika natychmiast z NEXPTIME $\subseteq$ *co*-NEXPTIME. Jeśli bowiem $L \in$ *co*-NEXPTIME, to $-L \in$ NEXPTIME, skąd $-L \in$ *co*-NEXPTIME i wreszcie $L \in$ NEXPTIME.

83: Przypuśćmy, że DSPACE$(n^2) \subseteq$ NP i niech $L \in$ PSPACE. Wtedy $L \in$ DSPACE(n^{2k}) dla pewnego k. Rozpatrzmy język $L' = \{w\$^d \mid d = |w|^k - |w| \wedge w \in L\}$. Ten język należy do klasy DSPACE(n^2), bo słowo $w\$^d$ o długości $|w|^k$ jest rozpoznawane w pamięci $|w|^{2k} = |w\$^d|^2$. A zatem $L' \in$ NP, czyli istnieje maszyna niedeterministyczna rozpoznająca L' w czasie n^ℓ dla pewnego ℓ. Tę maszynę łatwo przerobić na maszynę rozpoznającą język L w czasie $(n^k)^\ell = n^{k\cdot\ell}$.

84: Załóżmy, że DSPACE$(2^n) \subseteq$ EXPTIME i niech $L \in$ EXPSPACE. To znaczy, że dla pewnego $k \in \mathbb{N}$ istnieje deterministyczna maszyna Turinga, rozpoznająca język L w pamięci 2^{n^k}. Rozpatrzmy język $L' = \{w\$^{|w|^k-|w|} \mid w \in L\}$. Elementami L' są słowa postaci $w\$...\$$, długości $|w|^k$. Ten język należy do klasy DSPACE(2^n), bo do rozpoznania czy dane słowo długości n ma postać $w\$...\$$, gdzie $w \in L$ oraz $n = |w|^k$, maszyna Turinga potrzebuje pamięci $2^{|w^k|} = 2^n$. A zatem $L' \in$ EXPTIME, czyli istnieje deterministyczna maszyna M rozpoznająca L' w czasie postaci 2^{n^ℓ}, gdzie $\ell \in \mathbb{N}$. Skoro tak, to potrafimy zdefiniować maszynę N, która rozpozna język L w czasie $2^{n^{k\cdot\ell}}$. Dla danego w maszyna N naśladuje maszynę M działającą na słowie wejściowym $w\$...\$$ w czasie $2^{(|w|^k)^\ell)} = 2^{|w|^{k\cdot\ell}}$. A zatem maszyna N działa w czasie $2^{n^{k\cdot\ell}}$.

85: Przypuśćmy, że CSL $\subseteq$ NP, pokażemy, że NP = PSPACE. Wiadomo, że zachodzi inkluzja w prawo, więc niech $L \in$ PSPACE, powiedzmy $L \in$ DSPACE(n^k). Jeśli $L' = \{w\$^{|w|^k-|w|} \mid w \in L\}$, to $L' \in$ DSPACE$(n) \subseteq$ CSL. A skoro CSL $\subseteq$ NP, to $L' \in$ NTIME(n^ℓ) dla pewnego ℓ. Stąd wynika, że $L \in$ NTIME$((n^k)^\ell) =$ NTIME$(n^{k\cdot\ell}) \subseteq$ NP.

86: Tak. Przypuśćmy, że NTIME$(n^2) \subseteq$ P; udowodnimy, że P=NP. Weźmy dowolny język $L \in$ NP, powiedzmy $L \in$ NTIME(n^{2k}). Wtedy język $L' = \{w\$^{|w|^k-|w|} \mid w \in L\}$ należy do NTIME(n^2), bo słowa postaci $w\$^{|w|^k-|w|}$ długości $n = |w|^k$, można rozpoznawać w czasie $|w|^{2k} = n^2$. Skoro NTIME$(n^2) \subseteq$ P, to także $L' \in$ P, tj. $L' \in$ DTIME(n^r), dla pewnego r. Wtedy jednak $L \in$ DTIME$(n^{rk}) \subseteq$ P, bo słowo $w \in L$ o długości n można rozpoznawać tak samo jak słowo $w\$...\$$ długości n^k, czyli w czasie $(n^k)^r$.

87: Ponieważ klasa *co*-NLOGSPACE jest równa klasie NLOGSPACE, więc pierwsza inkluzja oznacza to samo, co DTIME$(n^2) \subseteq$ NLOGSPACE. Tego oczywiście nie wiadomo. Jeśli tak jest, to NLOGSPACE = P. Udowodnimy zawieranie P $\subseteq$ NLOGSPACE, bo zawieranie w przeciwną stronę i tak zachodzi. Niech więc $L \in$ P, powiedzmy, że $L \in$ DTIME(n^k). Definiujemy język $L' = \{w\$^{|w|^k-|w|} \mid w \in L\}$, gdzie symbol \$ jest nowy. Ten język można rozpoznawać w czasie liniowym (słowo wejściowe $w\$^{|w|^k-|w|}$ ma długość $n = |w|^k$), a tym bardziej w czasie n^2. Z założenia, że DTIME$(n^2) \subseteq$ NLOGSPACE wynika więc, że $L' \in$ NLOGSPACE. Maszynę M' rozpoznającą L' w logarytmicznej pamięci można przerobić na maszynę M, która rozpoznaje język L. Maszyna M dla wejścia w naśladuje działanie M' dla wejścia

$w\$^{|w|^k-|w|}$ w ten sposób, że pozycja głowicy wejściowej maszyny M' jest reprezentowana przez licznik rozmiaru logarytmicznego (nie przepisuje się dolarów na taśmę roboczą). A zatem maszyna M na wejściu długości n używa pamięci $\mathcal{O}(\log(n^k)) = \mathcal{O}(k \log n) = \mathcal{O}(\log n)$.
Druga inkluzja zachodzi. Klasa *co*-NLOGSPACE = NLOGSPACE, czyli klasa NSPACE($\log n$), jest zawarta w klasie P = DTIME($n^{\mathcal{O}(1)}$), co oczywiście mieści się w DTIME($n^{\log n}$). Trzecia inkluzja też zachodzi, bo klasa DTIME($n^{\log n}$) jest zawarta nawet w klasie DSPACE($n^{\log n}$), a przecież $n^{\log n} \leq n^n$.
Czwarta inkluzja jest fałszywa, bo PSPACE = DSPACE($n^{\mathcal{O}(1)}$) $\subseteq$ DSPACE($n^{\log n}$) $\subsetneq$ DSPACE(n^n). Ostatnie zawieranie jest ostre, ponieważ iloraz $n^{\log n}/n^n$ maleje do zera.

96: Wskazówka: Rozpatrzmy graf $\overline{G}$, który ma te same wierzchołki, co graf G, ale zbiór jego krawędzi jest dopełnieniem zbioru krawędzi grafu G. (Tj. w grafie $\overline{G}$ wierzchołki są połączone krawędzią wtedy i tylko wtedy, gdy *nie są połączone* w grafie G.) Zbiór K jest kliką w $\overline{G}$ wtedy i tylko wtedy, gdy jego dopełnienie jest pokryciem wierzchołkowym w G. A zatem pytanie o istnienie k-elementowego pokrycia wierzchołkowego w m-elementowym grafie G sprowadza się (w logarytmicznej pamięci) do pytania o istnienie kliki w $\overline{G}$, która ma $m - k$ elementów. Skoro problem pokrycia wierzchołkowego jest NP-trudny, to także problem kliki też jest NP-trudny. A jest to oczywiście problem z klasy NP, bo niedeterministyczna maszyna może „zgadnąć", które elementy mają być w klice i sprawdzić, że są połączone krawędziami.

98: Wszystkie trzy problemy są NP-zupełne, a zatem oczywiście rozstrzygalne a tym bardziej rekurencyjnie przeliczalne. Jeśli któryś z nich jest NL-zupełny, to NL = P = NP.

98a: Do tego problemu sprowadza się problem SAT. Jeśli ψ jest dowolną formułą i zmienna zdaniowa r nie występuje w ψ, to ψ jest spełnialna wtedy i tylko wtedy, gdy formuła $\psi \wedge (\neg r \vee r)$ ma co najmniej dwa wartościowania spełniające.

98b: Do tego problemu sprowadza się problem kliki. Zbiór S wierzchołków grafu G jest kliką wtedy i tylko wtedy, gdy w grafie dualnym $\overline{G}$ żadne dwa jego wierzchołki nie są połączone krawędzią.

98c: Szczególnym przypadkiem tego problemu (dla $d = 1$) jest problem pokrycia wierzchołkowego.

117: Terminy 117b, 117d, 117f oznaczają to samo. Należy do tej klasy na przykład problem stopu. Ale problem stopu jest nierozstrzygalny, czyli nie jest rekurencyjny ani obliczalny, bo nazwy 117a, 117c i 117e mają to samo znaczenie. Natomiast języki kontekstowe (117g), zwane też monotonicznymi (117h), stanowią właściwą podklasę języków rekurencyjnych. Kontrprzykładem jest np. każdy język należący do różnicy DSPACE(2^n) − DSPACE(n^2), bo wiadomo, że

$$\text{CSL} = \text{NSPACE}(n) \subseteq \text{DSPACE}(n^2) \subsetneq \text{DSPACE}(2^n).$$

118a: Problem prawdziwości formuły zdaniowej jest *co*-NP-zupełny. Jest więc NP-zupełny tylko w przypadku gdy *co*-NP = NP.

118b: Problem odpowiedniości Posta jest NP-trudny. Każdy język z klasy NP jest językiem obliczalnym, w szczególności rekurencyjnie przeliczalnym. Pytanie o przynależność danego słowa do takiego języka jest więc szczególnym przypadkiem problemu stopu, który redukuje się *w pamięci logarytmicznej* do problemu odpowiedniości Posta.

118c: Klasa języków bezkontekstowych nie jest zamknięta ze względu na dopełnienie. Przykładem jest język $\{a^n b^m c^k \mid n \neq m \vee m \neq k\}$.

118d: Dopełnienie języka rekurencyjnie przeliczalnego nie jest rekurencyjnie przeliczalne, gdy język nie jest rekurencyjny.

118e: Problem pustości języka bezkontekstowego jest rozstrzygalny. Wystarczy w tym celu zauważyć, że jeśli dany język jest niepusty to zawiera przynajmniej jedno słowo o długości nie wiekszej niż stała z lematu o pompowaniu (którą można efektywnie wyznaczyć).

118f: Tak. Jeśli $L_1 \leq_{\log} L_2$, to dla każdego słowa x zachodzi równoważność $x \in L_1 \Leftrightarrow f(x) \in L_2$, gdzie f jest odpowiednią funkcją obliczalną w pamięci logarytmicznej. Wtedy $x \notin L_1 \Leftrightarrow f(x) \notin L_2$.

118g: Tak. Skoro L_1 jest PSPACE-zupełny, to należy do PSPACE, zamkniętego ze względu na dopełnienie. Stąd $-L_1 \in$ PSPACE. A jeśli $L \in$ PSPACE, to $-L \in$ PSPACE, więc $-L \leq_{\log} L_1$, skąd także $L \leq_{\log} -L_1$.

118h: Nie. Język QBF jest zawarty w zbiorze Σ^* wszystkich słów, który nie jest PSPACE-zupełny, bo łatwo widzieć, że jeśli $L \leq_{\log} \Sigma^*$, to $L = \Sigma^*$.

118i: Nie. Z części 118g wynika, że język $-$QBF jest PSPACE-zupełny, tymczasem QBF $\cup -$QBF $= \Sigma^*$, oraz QBF $\cap -$QBF $= \varnothing$. Tymczasem ani Σ^* ani $\varnothing = -\Sigma^*$ nie są PSPACE-zupełne.

118j: Tak. Wprost z definicji, bo NP $\subseteq$ PSPACE.

118k: Nie, bo klasa języków kontekstowych to klasa NSPACE(n). A ponieważ iloraz $\log n/n$ dąży do zera przy n rosnącym do nieskończoności, to nawet klasa DSPACE(n) nie zawiera się w LOGSPACE.

118l: Nie, ponieważ klasa P, czyli klasa DTIME$(n^{\mathcal{O}(1)})$ zawiera się w DTIME$(n^{\log\log n})$, a iloraz

$$\frac{n^{\log\log n} \cdot \log(n^{\log\log n})}{n^{\log n}} < \frac{n^{\log\log n} \cdot n^{\log\log n}}{n^{\log n}} = \frac{n^{2\log\log n}}{n^{\log n}}$$

dąży do zera, gdy n dąży do nieskończoności. Zatem P $\subseteq$ DTIME$(n^{\log\log n}) \subsetneq$ DTIME$(n^{\log n})$.

118m: Nie, bo klasa P zawiera się w tym iloczynie.

118n: Tego nie wiadomo. Jeśli tak jest, to P = NP.

118o: Tak, relacja $\leq_{\log}$ to szczególnie prosty sposób odwołania się do wyroczni.

119a,119b: Język L_1 nie jest regularny. W przeciwnym razie język $L_3 = -L_1 \cap a^*b^* = \{a^n b^n \mid n \in \mathbb{N}\}$ byłby regularny. A ten język ma nieskończenie wiele różnych ilorazów, np. $L_3 \setminus a^k = \{a^n b^{n+k} \mid n \in \mathbb{N}\}$.

Język L_1 jest bezkontekstowy. Generuje go taka gramatyka:

$$\xi_0 ::= a\xi_a \mid \xi_b b, \qquad \xi_a := a\xi_a \mid \eta, \qquad \xi_b := \xi_b b \mid \eta, \qquad \eta ::= a\eta b \mid \varepsilon$$

Język L_2 nie jest nawet bezkontekstowy. Gdyby był, to także język

$$L_4 = L_2 \cap ba^*bba^*bba^*b = \{ba^n bba^n bba^n b \mid n \in \mathbb{N}\}$$

byłby bezkontekstowy. A do języka L_4 łatwo można zastosować metodę lematu o pompowaniu. Przypuśćmy, że n jest stałą dobraną z tego lematu dla języka L_4 i rozpatrzmy słowo $w = ba^n bba^n bba^n b \in L_4$. Jeśli $w = xyzuv$, gdzie $|yzu| \leq n$, oraz litera b występuje w części y lub u, to słowo $xy^0zu^0v = xzv$ nie należy do L_4, bo ma za mało liter b. Zatem części y i u składają się wyłącznie z liter a, a skoro fragment yzu jest „krótki”, to albo pierwszy albo ostatni segment a^n pozostaje bez zmian w słowie $xy^0zu^0v = xzv$ i ma wtedy za dużo liter a. A zatem w każdym przypadku $xy^0zu^0v \notin L_4$, co przeczy lematowi o pompowaniu.

119c: Oba języki należą do klasy P, bo można je rozpoznawać deterministyczną maszyną Turinga w czasie wielomianowym, a w istocie nawet liniowym. Dla języka L_1 wystarczy nawet deterministyczny automat ze stosem.

Maszyna dla języka L_2 przepisuje całe słowo wejściowe na taśmę roboczą i przy tej okazji sprawdza, czy jego długość jest podzielna przez 3. Następnie przesuwa głowicę wejściową w lewo do początku, a roboczą w lewo tylko co trzeci krok. Teraz pierwsza głowica jest na początku taśmy, a robocza w $2/3$ długości słowa. To miejsce zostaje zaznaczone.

Następnie głowica wejściowa idzie w prawo a robocza w lewo aż ta robocza nie dojdzie do początku słowa. Czytane litery muszą być zawsze takie same (to będzie słowo ww^R).

Teraz głowicę wejściową przesuwamy na początek słowa, a roboczą na koniec. W ostatniej fazie czytamy taśmę roboczą w lewo, aż do zaznaczonego miejsca, i porównujemy z tym, co widzi przesuwająca się w prawo głowica wejściowa.

119d: Ten problem jest nierozstrzygalny. Sprowadzimy do niego problem stopu w wersji „*Czy dana deterministyczna maszyna Turinga M akceptuje słowo puste?*”

Dla danej maszymy M skonstruujemy maszynę M' działającą tak: dla dowolnego słowa wejściowego x maszyna M' uruchamia symulację M na słowie pustym. Jeśli to się uda (maszyna M zaakceptowała słowo puste), to maszyna M' uruchamia maszynę dla L_1 na wejściu x. W rezultacie:

$$M \text{ akceptuje słowo puste} \quad \text{wtedy i tylko wtedy, gdy} \quad L(M') = L_1.$$

Gdyby więc nasz problem był rozstrzygalny, to i problem stopu też, a nie jest.

Podziękowanie

Dziekuję Paniom: Marii Fronczak, Paulinie Knut i Natalii Ruteckiej, oraz Panom: Piotrowi Achingerowi, Jakubowi Katarzyńskiemu i Mateuszowi Legięckiemu za wykrycie pomyłek we wcześniejszych wersjach tego tekstu.

Literatura

- Kozen, D.C., *Theory of Computation*, Springer, 2006.
- Papadimitriou, Ch. H., *Złożoność obliczeniowa*, WNT, Warszawa 2002.
- **Sipser, M., *Wprowadzenie do teorii obliczeń*, WNT, Warszawa 2009.**
- Ślusarek, M. i inni, *Złożoność obliczeniowa*, `http://osilek.mimuw.edu.pl/`