Języki, automaty i obliczenia

Wykład dla III roku informatyki, zima 2002-03

Paweł Urzyczyn
http://www.mimuw.edu.pl/∼urzy/jaio.html

Wykład 1

Słowa i języki

Definicja 1.1 *Słowo* nad alfabetem $\mathcal{A}$ to dowolny skończony ciąg elementów zbioru $\mathcal{A}$. Dokładniej, jest to dowolna funkcja $w : n \to \mathcal{A}$, gdzie n jest pewną liczbą naturalną. Liczbę tę nazywamy *długością* słowa w, i zapisujemy to tak: $n = |w|$. A więc 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.*

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. A zatem $w \cdot v : n + m \to \mathcal{A}$, a 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 *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 zbioru $\mathcal{A}^*$).

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

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

Oczywiście 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

- $\emptyset \in \mathcal{R}$;
- $\{\varepsilon\} \in \mathcal{R}$;
- $\{a\} \in \mathcal{R}$, 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:

- „$\emptyset$" 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: $L_\emptyset = \emptyset$, $L_\varepsilon = \{\varepsilon\}$, $L_a = \{a\}$, $L_{\alpha\cup\beta} = L_\alpha \cup L_\beta$, $L_{\alpha\cdot\beta} = L_\alpha \cdot L_\beta$. 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 więc 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:

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

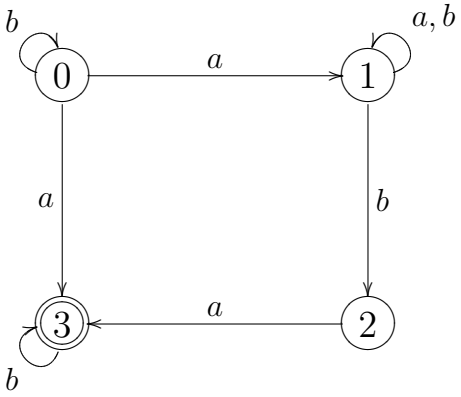
gdzie:

- $\mathcal{A}$ 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.



Obrazek 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 $q_0 \xrightarrow{w} q$, dla pewnego $q \in F$. Uwaga: w ogólności może być wiele obliczeń realizujących relację $q_0 \xrightarrow{w} q$, ale jeśli automat jest deterministyczny, to co najwyżej jedno. *Językiem akceptowanym* przez $\mathcal{M}$ nazywamy język:

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

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

Wykład 2

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 $A_{ij} = \{a \in \mathcal{A} \mid q_i \xrightarrow{a} q_j\}$. 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 2.1 *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. ∎

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

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

i wszystkie te języki są regularne.

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

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

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 2.1, 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 2.3 *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 2.2. ∎

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

Przykład 2.4 Dla automatu z obrazka mamy układ równań:

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

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

I żadnych innych

Twierdzenie 2.5 *Dla dowolnego języka regularnego L istnieje automat skończony, który go akceptuje.*

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

- jest równy $\emptyset$ 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 $q_0 \xrightarrow{u} f \in F_1$ w automacie $\mathcal{M}_1$, oraz przejścia $q_0 \xrightarrow{v} f' \in F_2$ 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. ∎

2.1 Pompowanie

Twierdzenie 2.6 (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 $j \leq m$. Trzeci warunek wynika stąd, że zarówno ciąg

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

jak i 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 2.7 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 $0^{N+ik}110^N \in L$, dla dowolnego $k = -1, 0, 1, 2, \ldots$

Wykład 3

Determinizacja

Twierdzenie 3.1 *Każdy język regularny jest akceptowany przez pewien 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 $\mathbf{P}(Q)$. Stan początkowy to zbiór jednoelementowy $q_0' = \{q_0\}$. Zbiorem stanów końcowych jest rodzina tych podzbiorow Q, które mają niepuste przecięcia z F:

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

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 $\mathcal{M}' \vdash \{q_0\} \xrightarrow{w} Z$ oznacza, że zbiór Z składa się ze wszystkich stanów osiągalnych z q_0 za pomocą relacji $\xrightarrow{w}_{\mathcal{M}}$. Dokładniej:

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

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

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

∎

Wniosek 3.2 *Następujące warunki są równoważne:*

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

Wniosek 3.3

a) Jeśli języki L_1 i L_2 są regularne to $L_1 \cap L_2$ jest regularny.

b) Jeśli język L jest regularny to $-L$ też jest regularny[3]*.*

Dowód: (a) 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 deterministycznymi akceptującymi odpowiednio L_1 i L_2. Automat akceptujący iloczyn jest taki:

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

Funkcja przejścia jest określona tak:

$$\delta(\langle q_1, q_2\rangle, a) = \langle \delta_1(q_1, a), \delta_2(q_2, a)\rangle.$$

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

(b) 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 $-L$. ∎

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

Ilorazy

Zdefiniujemy teraz (tym razem poprawnie) pojęcie lewo- i prawostronnego *ilorazu* języków.

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

Lemat 3.4

- $L\backslash\varepsilon = L$;
- $(L\backslash w)\backslash a = L\backslash wa$;
- *Warunki* $\varepsilon \in L\backslash w$ *i* $w \in L$ *są równoważne.*

Dowód: Łatwy, jeśli się nie pomyli ilorazów... ∎

Twierdzenie 3.5 *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 automatu $\mathcal{M} = \langle Q, \mathcal{A}, \delta, q_0, F\rangle$. Przyjmując $Q = \{q_0, \ldots, q_n\}$ możemy, jak już robiliśmy to wcześniej, rozważać maszyny $\mathcal{M}_i = \langle Q, \mathcal{A}, \delta, q_i, F\rangle$ i języki $L_i = L(\mathcal{M}_i)$, dla $i = 1, \ldots, n$.

Jeśli teraz $q_0 \xrightarrow{w} q_i$ 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 $a \in \mathcal{A}\}$.
- $q_0 = L$.
- $F = \{L' \in Q \mid \varepsilon \in L'\}$.

Zauważmy, że automat jest deterministyczny. Łatwo widzieć, że dla dowolnego słowa w zachodzi $\mathcal{M} \vdash L \xrightarrow{w} L\backslash w$.

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:

- Jeśli L jest regularny to $L^R = \{w^R \mid w \in L\}$ też.
- Zawsze $L/w = (L^R\backslash w^R)^R$.

Przykład 3.6 Język $L = \{0^n 1^n \mid n \in \mathbb{N}\}$ nie jest regularny, bo każdy iloraz $L\backslash 0^n$ jest inny.

Języki bezkontekstowe

Przypomnijmy, że *gramatyką bezkontekstową* nad alfabetem $\mathcal{A}$ nazywamy twór postaci

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

w którym *produkcje* (czyli *reguły*) ze zbioru P mają kształt

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

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. 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*.

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, generującą język wszystkich palindromów. A ta gramatyka opisuje mały kawałeczek języka polskiego:

$\langle$zdanie$\rangle$ ::= $\langle$grupa podmiotu$\rangle\langle$grupa orzeczenia$\rangle$;
$\langle$grupa podmiotu$\rangle$ ::= $\langle$przydawka$\rangle\langle$grupa podmiotu$\rangle$ | $\langle$podmiot$\rangle$;
$\langle$grupa orzeczenia$\rangle$ ::= $\langle$okolicznik$\rangle\langle$orzeczenie$\rangle$;
$\langle$orzeczenie$\rangle$ ::= $\langle$czasownik$\rangle$;
$\langle$podmiot$\rangle$ ::= $\langle$rzeczownik$\rangle$;
$\langle$przydawka$\rangle$ ::= $\langle$przymiotnik$\rangle$;
$\langle$okolicznik$\rangle$ ::= $\langle$przysłówek$\rangle$;
$\langle$rzeczownik$\rangle$ ::= *pies* | *kot*;
$\langle$czasownik$\rangle$ ::= *szczeka* | *śpi*;
$\langle$przymiotnik$\rangle$ ::= *czarny* | *mały*;
$\langle$przysłówek$\rangle$ ::= *głośno* | *smacznie*.

Język generowany przez gramatykę bezkontekstową można uważać za najmniejszy punkt stały pewnego operatora (dokładniej: jedną ze współrzędnych najmniejszego punktu stałego pewnego wielowymiarowego operatora na językach). Zilustrujemy to na przykładzie. Niech gramatyka 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 $R' = R$. Jeśli zdefiniujemy operator

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

wzorem $\Phi(\langle X, Y\rangle) = \langle YX \cup a, \varepsilon \cup aYb\rangle$, to punktami stałymi Φ są dokładnie rozwiązania naszego układu.

Fakt 3.7 *Niech $L_0 = R_0 = \emptyset$ i niech $\langle L_{n+1}, R_{n+1}\rangle = \Phi(\langle L_n, R_n\rangle)$ dla $n \in \mathbb{N}$. Inaczej, $\langle L_n, R_n\rangle = \Phi^n(\langle\emptyset, \emptyset\rangle)$.*

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 $w \in R_k$.*

[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.

Wykład 4

Języki regularne są bezkontekstowe

Fakt 4.1 *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

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

Istotnie, każdemu wyprowadzeniu w G ospowiada 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 4.1 jest prawostronna.

Fakt 4.2 *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 4.3 *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 4.3 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 dowolnego $i = 0, \ldots, k-1$:

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

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

Lemat 4.4 *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 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) 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.

Oczywiście drzewo wywodu dla w istnieje wtedy i tylko wtedy, gdy $w \in L(G)$. Drzewo wywodu może odpowiadać różnym wyprowadzeniom tego samego słowa, ale tylko jednemu wyprowadzeniu lewostronnemu. Mimo to dla jednego słowa może istnieć więcej niż jedno drzewo wywodu.

Lemat 4.5 *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 4.6 (Lemat o pompowaniu)

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

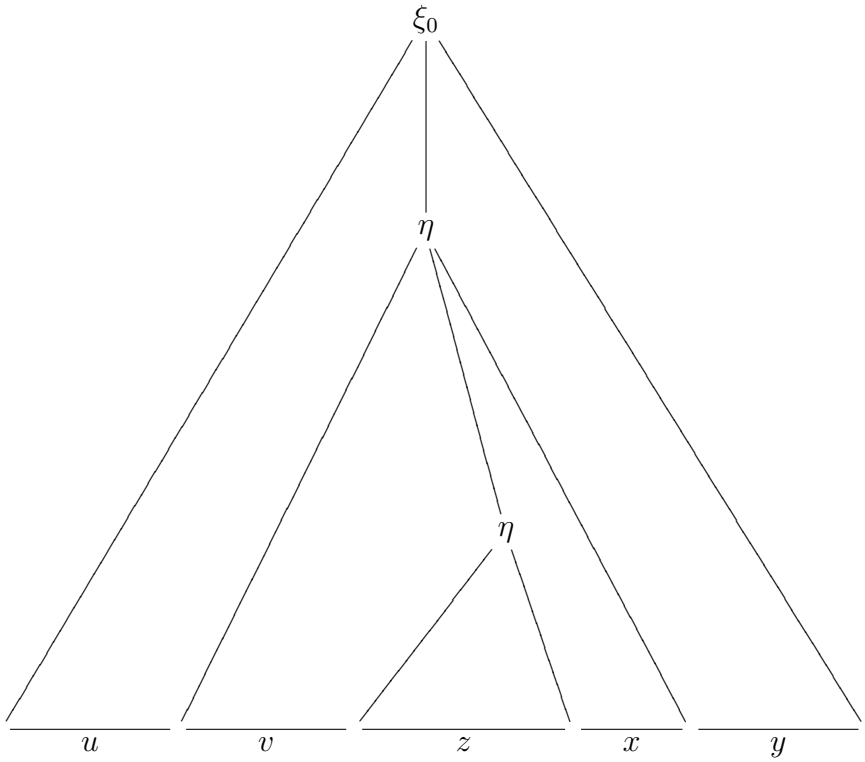
Jeżeli $w \in L$ oraz $|w| \geq n$ to w można przedstawić w postaci $w = uvzxy$, gdzie

- $vx \neq \varepsilon$;
- $|vzx| \leq n$;
- $uv^i zx^i y \in L$ *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 4.5) 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. Na mocy Lematu 4.5, 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:

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

w którym dwa wyróżnione wystąpienia η dotyczą wierzchołków `a` i `b`. Można to wyprowadzenie modyfikować, usuwając lub powielając fragment drzewa odpowiadający wyprowadzeniu $\eta \twoheadrightarrow v\eta x$. 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 wyprowadzenie $\xi_0 \twoheadrightarrow u\eta y \twoheadrightarrow uzy$ ma mniejsze drzewo wywodu. ∎

Przykład 4.7 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 wystepowały w niej zarówno litery a jak i c. Zatem w słowie uv^2zx^2y zabraknie albo liter a albo c.

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*;
- $\delta \subseteq (Q \times \mathcal{A} \cup \{\varepsilon\} \times \Sigma) \times (Q \times \Sigma^*)$ 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 a $V \in \Sigma^*$ nazywamy *konfiguracją* automatu. (Interpretacja: automat jest w stanie q a zawartością stosu jest słowo v, przy czym wierzchołek stosu jest z lewej strony.) Działanie automatu opisujemy za pomocą relacji $\mathcal{M} \vdash \mathcal{C}_1 \xrightarrow{w} \mathcal{C}_2$, pomiędzy konfiguracjami:

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

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

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

Przez *obliczenie* rozumiemy oczywiście odpowiedni ciąg konfiguracji.

Przykład 4.8 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 takich piątek:

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

Wykład 5

W Przykładzie 4.8 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:

Lemat 5.1 *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 automat $\mathcal{M} = \langle Q, \mathcal{A}, \Sigma, \delta, q_0, \sigma_0, F \rangle$ 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:*
$L = L(\mathcal{M}) = \{w \in \mathcal{A}^* \mid \mathcal{M} \vdash \langle q_0, \sigma_0 \rangle \xrightarrow{w} \langle f, \varepsilon \rangle\}$.

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

4) *Nie ma w δ piątki postaci $\langle q, a, \sigma, p, \sigma_0\sigma \rangle$, a jeśli $\langle q, a, \sigma_0, p, \varepsilon \rangle \in \delta$ 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 ustalamy 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 $\langle q, a, \sigma, p, \tau_1 \dots \tau_k\rangle$, która nie jest ani postaci *push* ani *pop* zastąpimy mianowicie przez piątki:

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

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

Akceptowanie języków bezkontekstowych

Twierdzenie 5.2 *Każdy język bezkontekstowy jest akceptowany przez pewien 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.

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

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

- $\langle q_1, \varepsilon, \sigma_0, q_2, \varepsilon\rangle$.

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 $x, y \in (\mathcal{A} \cup \mathcal{N})^*$:

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

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 $L(\mathcal{M}) = L(G)$. ∎

Twierdzenie 5.3 *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 5.1, 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:

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

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 $L(G) = L(G_{0n}^{\sigma_0}) = L(\mathcal{M})$.

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

- $\langle q_i, \sigma\rangle \xrightarrow{a} \langle q_k, \varepsilon\rangle$ (pojedyncza operacja typu „pop")
- $\langle q_i, \sigma\rangle \xrightarrow{a} \langle q_k, \tau\sigma\rangle \xrightarrow{w'} \langle q_\ell, \sigma\rangle \xrightarrow{w''} \langle q_j, \varepsilon\rangle$,
  przy czym obliczenie $\langle q_k, \tau\sigma\rangle \xrightarrow{w'} \langle q_\ell, \sigma\rangle$ odbywa się bez usuwania σ ze stosu.

(Zauważmy, że warunki Lematu 5.1 uniemożliwiają jakikolwiek krok przy pustym stosie.) Dla $w \in L_{ij}^{\sigma}$, w drugim przypadku mamy $w' \in L_{k\ell}^{\tau}$ iraz $w'' \in L_{\ell j}^{\sigma}$. Dlatego gramatyka G ma takie produkcje:

- $\xi_{ij}^{\sigma} \Rightarrow a$, dla dowolnego przejścia $\langle q_i, a, \sigma, q_k, \varepsilon \rangle \in \delta$;
- $\xi_{ij}^{\sigma} \Rightarrow a\xi_{k\ell}^{\tau}\xi_{\ell j}^{\sigma}$, dla dowolnego przejścia $\langle q_i, a, \sigma, q_k, \tau\sigma \rangle \in \delta$.

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. ∎

Wniosek 5.4 *Każdy język regularny jest bezkontekstowy.*

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

5.2 Deterministyczne języki bezkontekstowe

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

Klasa deterministycznych języków bezkontekstowych, jak się niebawem okaże, jest zamknięta ze względu na dopełnienie. Tymczasem:

Fakt 5.5 *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. Język ten nie spełnia jednak warunków lematu o pompowaniu. Niech bowiem n będzie stałą z lematu o pompowaniu. Jeśli rozbijemy słowo $a^n b^n c^n$ na pięć części $uvzxy$, tak że $|vzx| \leq n$, to słowo vx albo wcale nie zawiera litery a albo wcale nie zawiera litery c. W obu przypadkach, którejś litery będzie za mało w słowie uv^2zx^2y. ∎

Wniosek 5.6 *Nie każdy język bezkontekstowy jest deterministyczny.*

Wykład 6

Deterministyczne języki bezkontekstowe

Lemat 6.1 *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 $\delta(q, a, \sigma_0) = (p, W)$, to $W = W'\sigma_0$, gdzie $W' \in (\Sigma - \{\sigma_0\})^*$, jeśli zaś $\sigma \neq \sigma_0$ i $\delta(q, a, \sigma) = (p, W)$, to $W \in (\Sigma - \{\sigma_0\})^*$. (Symbol σ_0 nie jest nigdy usuwany ze stosu ani nie jest tam dokładany.)*

2) *Dla dowolnej $q \in Q$, $\sigma \in \Sigma$ albo określona jest wartość $\delta(q, \varepsilon, \sigma)$, albo wszystkie wartości $\delta(q, a, \sigma)$, dla $a \in \mathcal{A}$. (Automat $\mathcal{M}$ w każdej konfiguracji, przy każdym wejściu, może wykonać dokładnie jeden ruch.)*

3) *Dla dowolnego słowa w istnieje obliczenie postaci $\langle q_0, \sigma_0\rangle \xrightarrow{w} \langle q, X\rangle$, 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 $\delta(q_0, \varepsilon, \sigma_0) = (q_0', \sigma_0'\sigma_0)$. W ten sposób uzyskamy warunek (1).

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

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

Dla dowolnych a i σ definiujemy też $\delta(r, a, \sigma) = (r, \sigma)$. 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 $\delta(f, a, \sigma) = (r, \sigma)$, dla wszystkich $a \in \mathcal{A}$.

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

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

Jeśli w takiej sytuacji, któryś ze stanów p_i jest końcowy, to poprawiamy funkcję δ' tak: $\delta(q, \varepsilon, \sigma) = (f, \sigma)$. Jeśli żaden nie jest końcowy, to definiujemy $\delta(q, \varepsilon, \sigma) = (r, \sigma)$. 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

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

Z ciągu wszystkich 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: wszystkie słowa W_m, dla $m \geq k$ są postaci $W_m = W'_m W$, istnieje więc nieskończone obliczenie

$$\langle p_k, \tau \rangle \xrightarrow{\varepsilon} \langle p_{k+1}, W'_{k+1} \rangle \xrightarrow{\varepsilon} \langle p_{k+2}, W'_{k+2} \rangle \xrightarrow{\varepsilon} \cdots$$

W automacie $\mathcal{M}$ mamy teraz $\delta(p_k, \varepsilon, \tau) = (r, \tau)$ lub $\delta(p_k, \varepsilon, \tau) = (f, \tau)$. A zatem $\mathcal{M} \vdash \langle q_0, \sigma_0 \rangle \xrightarrow{w'} \langle p_k, \tau \cdot W \rangle \xrightarrow{\varepsilon} \langle p, \tau \cdot W \rangle \xrightarrow{w''} \langle r, \tau \cdot W \rangle$, gdzie p to albo r albo f. Tak czy owak, obliczenie kończy się w stanie r, bo słowo w'' jest niepuste. ∎

Fakt 6.2 *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 6.1. 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 $q'_0 = \langle q_0, 2 \rangle$. 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 osiągnął (1) czy nie (2) jakiś stan akceptujący przeczytane dotychczas słowo. Wartość zero oznacza, że stanu akceptującego nie było i już nie będzie, bo właśnie zamierzamy przeczytać następną literę. Funkcja przejścia automatu $\mathcal{M}'$ jest więc określona tak:

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

Definiujemy $F' = \{\langle q, 0\rangle \mid q \in Q\}$. Wówczas 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. ∎

Wniosek 6.3 *Nie każdy język bezkontekstowy jest deterministyczny.*

Dowód: Z poprzedniego wykładu wiadomo, że klasa wszystkich języków bezkontekstowych nie jest zamknięta ze względu na dopełnienie. ∎

Przykład 6.4 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.

Języki kontekstowe

Przez *gramatykę typu zero* rozumiemy krotkę $G = \langle \mathcal{A}, \mathcal{N}, P, \xi_0\rangle$, w którym *produkcje* są postaci $u \Rightarrow v$, gdzie $u, v \in (\mathcal{A} \cup \mathcal{N})^*$ są zupełnie dowolnymi słowami, byle tylko $u \neq \varepsilon$. Relacja redukcji $\rightarrow_G$ jest zdefiniowana podobnie jak dla gramatyk bezkontekstowych, mianowicie $x \rightarrow_G y$ (zapisywane też tak: $G \vdash x \rightarrow y$) 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:

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

Na przykład język $\{a^n b^n c^n \mid n \in \mathbb{N}\}$ jest generowany przez gramatykę typu zero, która ma takie 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* jeśli jej produkcje są tylko postaci

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

Jeśli język jest generowany przez gramatykę typu zero (monotoniczną) to mówimy, że jest to język *typu zero* (*monotoniczny*).

Gramatyka z przykładu powyżej jest monotoniczna. Natomiast nie każda gramatyka bezkontekstowa jest monotoniczna, bo po prawej stronie produkcji może być słowo puste. Ale chodzi nam o języki, a nie o gramatyki.

Fakt 6.5 *Każdy język bezkontekstowy jest monotoniczny.*

Dowód: Przerabiamy daną gramatykę bezkontekstową $G = \langle \mathcal{A}, \mathcal{N}, P, \xi_0 \rangle$, generującą język L, na gramatykę monotoniczną. 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 $\xi_0 \Rightarrow \xi_0'$.

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 $\xi_0 \Rightarrow \varepsilon$.

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. ∎

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. W istocie mamy:

Twierdzenie 6.6 *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 zadne 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. ∎

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 $n = 1, 2$ takie przykłady już znamy, przypadek $n = 0$ wyniknie z pewnych ogólniejszych faktów.

Wykład 7

Maszyny Turinga

Maszynę Turinga nad alfabetem $\mathcal{A}$ (niedeterministyczną, jednotaśmową) definiujemy jako krotkę

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

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 sumę $F = A \cup R$ nazywamy zbiorem 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 zazwyczaj 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 glowica 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 prawdziwość 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 $wcqav \to_{\mathcal{M}} wpcbv$;

Symbolem $\twoheadrightarrow_{\mathcal{M}}$ oznaczamy przechodnio-zwrotne domknięcie relacji $\to_{\mathcal{M}}$. Mówimy, że maszyna *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* słowo 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 $\mathcal{C} = \mathcal{C}_0 \to_{\mathcal{M}} \mathcal{C}_1 \to_{\mathcal{M}} \mathcal{C}_2 \to_M \cdots$. Obliczenie może być skończone (akceptujące lub odrzucające[5]) 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 7.1 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.

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

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. Następnie 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. Wtedy 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 na znowu na blank to akceptuje.

[5]Przy naszej definicji nie ma innej możliwości.

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

Maszyny Turinga stanowią bardzo prosty formalny model obliczenia. Ze względu na tę prostotę, opisy różnych algorytmów za pomocą zwykłych maszyn Turinga bywają skomplikowane. Dlatego posługujemy się różnymi uogólnieniami, np. rozważamy maszyny, które mają kilka taśm roboczych. Formalnie, maszyna z k taśmami roboczymi (i jedną wejściową) definiowana jest podobnie do zwykłej, ale ma relację przejścia

$$\delta \subseteq (Q - F) \times \mathcal{A} \times \Sigma^k \times \Sigma^k \times Q \times \{-1, 0, +1\}^{k+1}$$

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.) Konfiguracja początkowa ma postać:

$$\langle q_0, \langle \varepsilon, w\rangle, \langle \varepsilon, \varepsilon\rangle, \ldots, \langle \varepsilon, \varepsilon\rangle\rangle$$

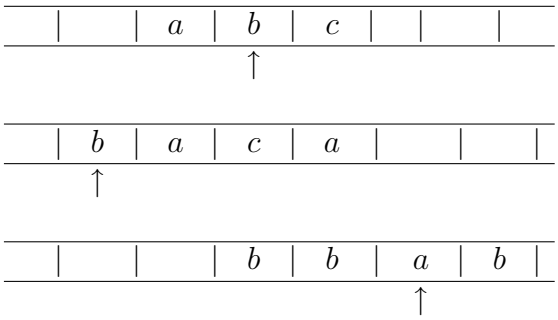
Zmiana konfiguracji polega na zmianie stanu, poprawieniu pozycji wszystkich głowic, i wpisaniu odpowiednich symboli na taśmach roboczych. Zawartość taśmy wejściowej nie ulega zmianie. Oczywiście maszyna wielotaśmowa może być deterministyczna lub nie.

Maszyny wielotaśmowe potrafią tyle samo ile zwykłe, tylko czasem wygodniej o nich mówić.

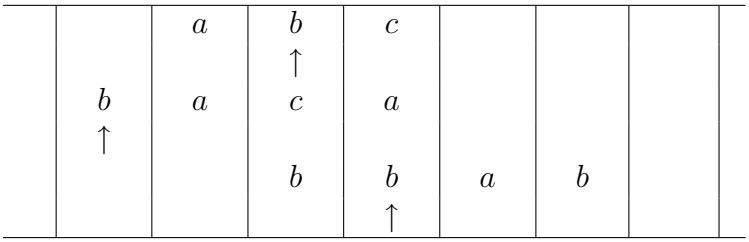
Fakt 7.2 *Jeśli $L = L(\mathcal{M})$, dla pewnej maszyny wielotaśmowej $\mathcal{M}$, to także $L = L(\mathcal{M}')$, dla pewnej maszyny jednotaśmowej $\mathcal{M}'$. Co więcej, jeśli $\mathcal{M}$ jest deterministyczna to $\mathcal{M}'$ też jest deterministyczna.*

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 znajdująca się na $2i$-tej współrzędnej oznacza, że głowica i-tej taśmy ogląda odpowiednią klatkę.

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



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



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. ∎

Wykład 8

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 *rekurencyjnie przeliczalny* (także: *częściowo obliczalny, częściowo rozstrzygalny*) jeśli $Z = L(\mathcal{M})$ dla jakiejkolwiek maszyny Turinga $\mathcal{M}$.

Fakt 8.1 *Jeśli Z jest rekurencyjnie przeliczalny, to $Z = L(\mathcal{M})$ dla pewnej deterministycznej maszyny Turinga $\mathcal{M}$.*

Dowód: Niech $Z = L(\mathcal{M}')$, gdzie $\mathcal{M}' = \langle \Sigma, Q, \delta, q_0, A, R \rangle$. 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 pewiem 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. ∎

Twierdzenie 8.2 *Język L jest rekurencyjnie przeliczalny wtedy i tylko wtedy, gdy $L = L(G)$ 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 $w\,[q_0 w]$ (ćwiczenie: jak to zrobić?). 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 $\delta(q, a) = (b, p, 0)$;

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

Zauważmy, że wówczas zachodzi równoważność:

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

bo każdy ciąg postaci $\texttt{[}q_0 w\texttt{]} \rightarrow_G x_1 \rightarrow_G x_2 \rightarrow_G \cdots \rightarrow_G \texttt{[}q_f\texttt{]}$ 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 $\xi_0 \twoheadrightarrow_G w\texttt{[}q_0 w\texttt{]}$, dla pewnego w.

A zatem wyprowadzenie w gramatyce G, kończące się słowem terminalnym może wyglądać tylko tak:

$$\xi_0 \twoheadrightarrow w\texttt{[}q_0 w\texttt{]} \twoheadrightarrow w\texttt{[}q_f\texttt{]} \rightarrow w,$$

co oznacza, że $L(G) = L(\mathcal{M})$. ∎

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 $\mathcal{A} \cup \{\$\}$. W ten sposób możemy też mówić o rekurencyjnych i rekurencyjnie przeliczalnych relacjach.

Jeśli f jest k-argumentową funkcją częściową na słowach, to mówimy, że f jest *częściowo rekurencyjna* (także *częściowo obliczalna*) jeśli jest to rekurencyjnie przeliczalna relacja $k+1$-argumentowa. Jeśli f jest na dodatek wszędzie określona, to mówimy, że jest rekurencyjna (także *obliczalna*).

Ćwiczenie: Pokazać, że jeśli f jest k-argumentową funkcją rekurencyjną to jest też rekurencyjną relacją $k+1$-argumentową. Uwaga: niekoniecznie na odwrót!

Uwaga: Jeśli zbiór A jest rekurencyjnie przeliczalny ale nie rekurencyjny, to jego *częściowa funkcja charakterystyczna* $\chi_A = \{\langle x, 0\rangle \mid x \in A\}$ też nie jest zbiorem rekurencyjnym. Zatem w defnicji funkcji częściowo rekurencyjnej nie można zamienić słów „rekurencyjnie przeliczalna" na „rekurencyjna".

Uwaga: Zwykle definiuje się funkcje (częściowo) rekurencyjne za pomocą maszyn, które generują na taśmie wartość wynikową. Definicja powyżej jest równoważna.

Fakt 8.3 *Następujące warunki są równoważne:*

1. *Z jest zbiorem rekurencyjnym;*
2. *$-Z$ jest zbiorem rekurencyjnym;*
3. *Z i $-Z$ są rekurencyjnie przeliczalne.*

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 łatwa. ∎

Fakt 8.4 *Następujące warunki są równoważne:*

1. *Z jest rekurencyjnie przeliczalny;*
2. *Z jest dziedziną pewnej funkcji częściowo rekurencyjnej;*
3. *Z jest zbiorem wartości pewnej funkcji częściowo rekurencyjnej;*
4. *Z jest pusty lub jest zbiorem wartości pewnej funkcji rekurencyjnej.*[6]

Dowód: Opuszczamy. ∎

Uniwersalna maszyna Turinga

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 sam algorytm przedstawiony jako słowo, zgodnie z ustaloną składnią, może stanowic daną wejściową dla innego algorytmu. Nie ma wiec istotnej róznicy miedzy danymi i programem. Maszyna, która wykonuje algorytm określony słowem wejściowym, to nic innego jak interpreter pewnego języka programowania.

[6]To uzasadnia nazwę „rekurencyjnie przeliczalny".

Aby określić taką *uniwersalną maszynę Turinga*, musimy najpierw przedstawić każdą maszynę $\mathcal{M}$ za pomocą pewnego słowa $kod_{\mathcal{M}}$. Można to zrobić rutynowymi metodami (kodując wszystkie informacje o maszynie, w tym relację (funkcję) przejścia, za pomocą odpowiednich słów) i to tak, że $kod_{\mathcal{M}} \in \mathcal{A}^*$. Rozpatrzmy teraz język

$$S = \{(kod_{\mathcal{M}}, w) \mid w \in L(M)\ \}.$$

Uniwersalna maszyna Turinga, to maszyna akceptująca język S. Maszyna ta, dla danych słów u, v, interpretuje słowo u jako kod pewnej maszyny $\mathcal{M}$ i symuluje jej zachowanie dla słowa wejściowego w. W konsekwencji otrzymujemy:

Wniosek 8.5 *Język uniwersalny S jest rekurencyjnie przeliczalny.*

Wykład 9

Problemy decyzyjne

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?"

gdzie daną wejściową, czyli *instancją* problemu nie jest słowo, ale jakiś obiekt kombinatoryczny. Oczywiście skończony graf można łatwo przedstawić za pomocą słowa (trzeba po prostu wypisać jego wierzchołki i krawędzie). Podobnie można też postąpić z innymi skończonymi obiektami. Ale trzeba tu zrobić dwie uwagi:

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?"

O ile powyższa obserwacje jest dość oczywista, następna uwaga dotyczy pułapki, w którą wpadają nawet doświadczeni badacze. Otóż np. w zadaniu dotyczącym formuł rachunku

predykatów[7] instancją problemu nie jest dowolne słowo, ale poprawnie zbudowana formuła. Zatem nie jest to ściśle biorąc problem decyzyjny. Ale oczywiście możemy łatwo wybrnąć z sytuacji zadając nieco zmodyfikowane pytanie:

– „Czy dane słowo jest tautologią rachunku predykatów?".

Łatwo, bo istnieje prosty algorytm sprawdzający czy dane słowo jest formułą, i w istocie nie ma znaczenia, które pytanie postawimy.

W praktyce nikt nie formułuje problemów decyzyjnych w taki sposób. Ważne tylko, żeby pytanie:

– „Czy dane słowo jest poprawną instancją problemu?"

(tj. formułą, reprezentacją grafu itp.) samo było rozstrzygalne.[8] Zwykle tak jest i to w oczywisty sposób. Ale może tak nie być. Na przykład to pytanie nie stanowi poprawnego problemu decyzyjnego:

– „Czy dana tautologia (klasycznego) rachunku predykatów jest także tautologią intuicjonistyczną?"

Albo to:

– „Dana jest maszyna $\mathcal{M}$, zatrzymująca sie dla słowa pustego. Czy $\mathcal{M}$ akceptuje słowo puste?"

Nierozstrzygalność

Teraz będzie o tym, że w ogóle istnieją problemy nierozstrzygalne. *Problemem stopu* dla maszyn Turinga nazywamy pytanie:

– „Czy dana maszyna $\mathcal{M}$ akceptuje dane słowo w?"

Aby to pytanie stało się poprawnym problemem decyzyjnym, musimy przedstawić każdą maszynę $\mathcal{M}$ za pomocą pewnego słowa $kod_{\mathcal{M}}$. Można to zrobić rutynowymi metodami (kodując wszystkie informacje o maszynie, w tym relację (funkcję) przejścia za pomocą odpowiednich słów) i to tak, że $kod_{\mathcal{M}} \in \mathcal{A}^*$. A zatem nasz *problem stopu* to zbiór par:

$$S = \{(kod_{\mathcal{M}}, w) \mid w \in L(M)\ \}.$$

Dla danego słowa v, niech $\mathcal{M}(v)$ oznacza maszynę spełniającą warunek $kod_{\mathcal{M}(v)} = v$. Jeśli takiej maszyny nie ma, to przyjmujemy za $\mathcal{M}(v)$ jakąkolwiek ustaloną maszynę. Zdefini-

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

[8] Jeśli interesuje nas nie tylko rozstrzygalność ale także złożoność problemu, to sprawdzenie poprawności instancji powinno być zadaniem o pomijalnej złożoności.

ujemy teraz zbiór:

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

Lemat 9.1 *Zbiór $-K$ nie jest rekurencyjnie przeliczalny, a zatem zbiór K nie jest rekurencyjny.*

Dowód: Przypuśćmy, że istnieje taka maszyna $\mathcal{M}$, że $L(\mathcal{M}) = -K$. Niech v będzie takie, że $\mathcal{M} = \mathcal{M}(v)$. Wtedy $L(\mathcal{M}(v)) = L(\mathcal{M})$.

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 9.2 *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 1: Powyższe dotyczy także maszyn deterministycznych.

Uwaga 2: Problem stopu może być też sformułowany tak: „Czy dana maszyna $\mathcal{M}$ zatrzymuje się dla danego słowa w?" i nadal będzie nierozstrzygalny.

Uwaga 3: 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 $R = \emptyset$ 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.

Metoda użyta w dowodzie Twierdzenia 9.2 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 oznaczo 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 9.3 *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. Najprostsze jej zastosowanie jest na przykład takie. 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 9.4 *Problem 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 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 pażdej parze $(\mathcal{M}, w)$ (instancji problemu stopu) trójkę (G, ξ_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 reprezentuje odpowiednią trójkę (w jakiś ustalony sposób). ∎

Problem odpowiedniości Posta

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

Dany jest skończony zbiór par słów $\{(x_i, y_i) \mid i = 1, \ldots, n\}$ (system Posta).
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* systemu Posta. Także słowo

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

bywa wtedy nazywane rozwiązaniem.

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

$$\frac{x_1|x_2|\ldots|x_n}{y_1|y_2|\ldots|y_n}$$

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 rozwiązania.

Twierdzenie 9.5 *Problem odpowiedniości Posta jest nierozstrzygalny.*

Dowód: Redukcja problemu 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[9]) system Posta P, który będzie miał rozwiązanie wtedy i tylko wtedy, gdy $G \vdash w \twoheadrightarrow v$.

Alfabet naszego systemu Posta 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. Należy to rozumieć tak, że do systemu P należą wszystkie takie pary.

$\texttt{[}w*$	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 mamy taki system Posta:

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

[9]Tj. opiszemy w istocie algorytm realizujący tę konstrukcję.

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 system Posta można rozwiązać tak:

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

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 systemu P, to takie rozwiązanie musi wyznaczać pewne wyprowadzenie $w \to\!\!\!\!\to_G v$. Rozwiązanie musi się zaczynać od pary ($[w*$, $[$), 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 $[$ słowa $w*$. To jednak oznacza, że do słowa $[w*$ musimy dopisać słowo postaci $\overline{u}_1\overline{*}$ gdzie $w \to\!\!\!\!\to_G u_1$. Zatem mamy teraz słowa

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

i musimy do drugiego z nich dopisać $\overline{u}_1\overline{*}$. Ale wtedy do pierwszego dopiszemy u_2*, gdzie $u_1 \to\!\!\!\!\to_G u_2$ i tak dalej. To sie może skończyć tylko użyciem jednej z par zawierających $]$, co oznacza, że $w \to\!\!\!\!\to v$.

Uwaga: rozwiązanie naszego systemu Posta nie musi być dokładnie takiej postaci jak podaliśmy wyżej. Na przykład przekształcenie bacy w owada może wyglądać nie tylko tak:

$$[\texttt{baca}*\overline{\texttt{owca}*}\texttt{owad}]$$

ale też na przykład tak:

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

∎

Przykładem zastosowania PCP jest taki fakt:

Twierdzenie 9.6 *Problem pustości dla języków kontekstowych jest nierozstrzygalny.*

Dowód: **(szkic)**

Dla danego systemu Posta P, 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 8.2, 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. ∎

Wykład 10

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 dowolnego 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 każde jej obliczenie się zatrzymuje, i na dodatek, 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 $\max\{S(n), 1\}$.

Fakt 10.1 *(1) Jeśli maszyna ma określoną złożoność czasową lub pamięciową, to język akceptowany przez tę maszynę jest rekurencyjny.*

(2) Każdy język rekurencyjny jest akceptowany przez pewną 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 oczywista, bo wtedy maszyna jest totalna. Jeśli maszyna niedeterministyczna ma określoną złożoność czasową lub pamięciową to dla dowolnego słowa wejściowego liczba jej obliczeń jest skończona. Można więc ją przerobić na totalną maszynę deterministyczną, która wykonuje kolejno wszystkie te obliczenia.

(2) Mamy deterministyczną, totalną maszynę $\mathcal{M}$ akceptującą dany język. Nietrudno ją przerobić na maszynę obliczającą czas pracy $T(n)$ maszyny $\mathcal{M}$ i użytą pamięć $S(n)$. ∎

Klasy złożoności

Języki obliczalne klasyfikujemy ze względu na ich złożoność czasową i pamięciową:

$\mathrm{DTIME}(T(n)) = \{L \mid L$ jest akceptowany przez deterministyczną maszynę Turinga o złożoności czasowej $T(n)\}$;

$\mathrm{NTIME}(T(n)) = \{L \mid L$ jest akceptowany przez niedeterministyczną maszynę Turinga o złożoności czasowej $T(n)\}$;

$\mathrm{DSPACE}(S(n)) = \{L \mid L$ jest akceptowany przez deterministyczną maszynę Turinga o złożoności pamięciowej $S(n)\}$;

$\mathrm{NSPACE}(S(n)) = \{L \mid L$ jest akceptowany przez niedeterministyczną maszynę Turinga o złożoności pamięciowej $S(n)\}$;

Złożoność obliczeniową rozważamy z dokładnością do stałej:

Fakt 10.2 *(1) Dla dowolnej stałej $c > 0$ zachodzą równości:*

$\mathrm{DSPACE}(S(n)) = \mathrm{DSPACE}(c \cdot S(n))$;
$\mathrm{NSPACE}(S(n)) = \mathrm{NSPACE}(c \cdot S(n))$.

(2) Jeśli $\lim_{n\to\infty} \frac{T(n)}{n} = \infty$, to dla dowolnej stałej $c > 0$ zachodzą równości:

$\mathrm{DTIME}(T(n)) = \mathrm{DTIME}(c \cdot T(n))$;
$\mathrm{NTIME}(T(n)) = \mathrm{NTIME}(c \cdot T(n))$;

Dowód: (1) Kodowanie (skomplikowane) zawartości kilku klatek w jednej.
(2) Symulacja kilku kroków maszyny w jednym (jeszcze bardziej skomplikowana). ∎

Twierdzenie 10.3 (o hierarchii)

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

(a) $\mathrm{DTIME}(T(n)) \subsetneq \mathrm{DTIME}(T_1(n))$ *oraz*
(b) $\mathrm{NTIME}(T(n)) \subsetneq \mathrm{NTIME}(T_1(n))$.

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

(a) $\mathrm{DSPACE}(S(n)) \subsetneq \mathrm{DSPACE}(S_1(n))$ *oraz*
(b) $\mathrm{NSPACE}(S(n)) \subsetneq \mathrm{NSPACE}(S_1(n))$.

Dowód: (1a) Niech $\mathcal{M}_w$ oznacza deterministyczną maszynę o kodzie w. Istnieje maszyna $\mathcal{M}$, która akceptuje słowo w wtedy i tylko wtedy, gdy maszyna $\mathcal{M}_w$ nie akceptuje w w ciągu pierwszych $T(n)$ kroków obliczenia. Sama maszyna M wykona w tym celu przynajmniej $T(n)$ kroków. Oczywiście $L(\mathcal{M}) \notin \mathrm{DTIME}(T(n))$. Jako T_1 przyjmujemy złożoność maszyny $\mathcal{M}$.

W pozostałych przypadkach stosujemy podobne rozumowanie przekątniowe. ∎

Powiemy, że funkcja $S(n)$ jest *konstruowalna* wtedy i tylko wtedy, gdy istnieje maszyna Turinga, która dla dowolnego słowa długości n, wypisuje na taśmie słowo $a^{S(n)}$ i zatrzymuje się, a przy tym ma złożoność pamięciową $S(n)$. Większość funkcji, które są rozważane w teorii złożoności jest konstruowalna (np. n^2, n^3, 2^n, $\lfloor \log n \rfloor$, itp.[10]).

Następujące twierdzenie wyraża podstawowe związki między klasami złożoności.

Twierdzenie 10.4

(1) $\mathrm{DTIME}(T(n)) \subseteq \mathrm{NTIME}(T(n))$.

(2) $\mathrm{NTIME}(T(n)) \subseteq \mathrm{DSPACE}(T(n))$.

(3) $\mathrm{DSPACE}(S(n)) \subseteq \mathrm{NSPACE}(S(n))$.

(4) $\mathrm{DSPACE}(S(n)) \subseteq \bigcup_{c>0} \mathrm{DTIME}(2^{c \cdot S(n)})$, *o ile* $S(n) \geq \log n$.

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

$\mathrm{NSPACE}(S(n)) \subseteq \bigcup_{c>0} \mathrm{DTIME}(2^{c \cdot S(n)})$.

Dowód: Zawierania (1) i (3) są oczywiste. Inkluzja (2) wynika stąd, że maszyna niedeterministyczna $\mathcal{M}$ może zostać zastąpiona maszyną deterministyczną przeglądającą wszystkie możliwe obliczenia maszyny $\mathcal{M}$. Postępujemy tak, jak w dowodzie Faktu 8.1. Potrzebna do tego jest pamięć rozmiaru $T(n)$, bo $T(n)$ jest także złożonością pamięciową $\mathcal{M}$ (w czasie $T(n)$ można użyć co najwyżej $T(n)$ klatek taśmy). Dodatkowo potrzeba pamięci $\mathcal{O}(T(n))$ na sterowanie symulacją (ciąg $m_1, m_2, \ldots, m_k$ na taśmie pierwszej).

Aby wykazać (4), rozpatrzmy maszynę deterministyczną $\mathcal{M}$ o złożoności pamięciowej $S(n)$, która ma k stanów wewnętrznych, r taśm roboczych i m-literowy alfabet. Wtedy łączna liczba wszystkich konfiguracji maszyny $\mathcal{M}$ nie przekracza $k \cdot S(n)^r \cdot n \cdot (m^{S(n)})^r$ (liczba stanów razy liczba położeń głowicy na r taśmach roboczych razy liczba położeń głowicy na taśmie wejściowej razy liczba możliwych sposobów wypełnienia $S(n)$ klatek na r taśmach za pomocą m symboli). Oczywiście

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

dla odpowiednio dużej stałej c. 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.

Załóżmy, że mamy niedeterministyczną maszynę $\mathcal{M}$ o złożoności pamięciowej $S(n)$. Niech $\mathbf{C}$ oznacza zbiór wszystkich konfiguracji maszyny $\mathcal{M}$ rozmiaru co najwyżej $S(n)$. Ponieważ

[10] Symbol log oznacza oczywiście logarytm dwójkowy.

oszacowanie z części (4) pozostaje poprawne, więc zbiór $\mathbf{C}$ ma co najwyżej $f(n) = 2^{c \cdot S(n)}$ elementów.

Rozpatrzmy następujący deterministyczny algorytm, który dla konfiguracji $\mathcal{C}_1, \mathcal{C}_2 \in \mathbf{C}$, stwierdza, czy $\mathcal{M} \vdash C_1 \twoheadrightarrow \mathcal{C}_2$. (W istocie jest to najprostszy algorytm sprawdzający czy w danym grafie istnieje ścieżka łącząca dwa wierzchołki.) Na początku zapisujemy na taśmie konfigurację $\mathcal{C}_1$. Następnie powtarzamy taką operację: dla wszystkich dotychczas zapisanych konfiguracji $\mathcal{C} \in \mathbf{C}$ wyszukujemy wszystkie $\mathcal{C}' \in \mathbf{C}$, że $\mathcal{M} \vdash \mathcal{C} \to \mathcal{C}'$ i też je zapisujemy (jeśli dotąd nie były zapisane). Postępujemy tak, aż znajdziemy konfigurację $\mathcal{C}_2$ albo wyczerpiemy wszystkie możliwości (stwierdzimy, że nic nowego nie da się już dopisać).

Ten algorytm wymaga liczby kroków rzędu $\mathcal{O}(f(n)^2 \cdot S(n))$, a to można oszacować przez $2^{c_1 \cdot S(n)}$ dla pewnej stałej c_1. Można go zrealizować za pomocą deterministycznej maszyny Turinga, której złożoność będzie ograniczona przez $2^{c_1 \cdot S(n)}$. ∎

Wykład 11

Związki między klasami złożoności

O związkach pomiędzy klasami złożoności wciąż niewiele wiadomo. Poniższe twierdzenia należą do nielicznych wyników w tej dziedzinie.

Twierdzenie 11.1 (Savitch, 1970) *Jeśli funkcja $S(n)$ jest konstruowalna, a przy tym*[11] $S(n) \geq \log n$, *to* $\mathrm{NSPACE}(S(n)) \subseteq \mathrm{DSPACE}(S(n)^2)$.

Dowód: Niech $\mathcal{M}$ będzie niedeterministyczną maszyną o złożoności pamięciowej $S(n)$. Rozpatrzmy znów zbiór $\mathbf{C}$ złożony ze wszystkich konfiguracji maszyny $\mathcal{M}$ rozmiaru co najwyżej $S(n)$. Jak już zauważyliśmy w dowodzie poprzedniego twierdzenia, moc tego zbioru jest ograniczona z góry przez $2^{c \cdot S(n)}$, a zatem liczba kroków maszyny jest też ograniczona przez $2^{c \cdot S(n)}$.

Dla $\mathcal{C}_1, \mathcal{C}_2 \in \mathbf{C}$, oraz $i \leq c \cdot S(n)$, piszemy $\mathcal{C}_1 \to^i \mathcal{C}_2$ wtedy i tylko wtedy, 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 $\mathcal{C} \in \mathbf{C}$ 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

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

[11] Założenie $S(n) \geq \log n$ jest bardzo naturalnym wymaganiem. Pamięć $\log n$ jest konieczna do tego aby maszyna mogła znać położenie własnej głowicy wejściowej.

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

A zatem w pamięci $\varphi(c \cdot S(n)) \leq c \cdot S(n)^2$ można deterministycznie sprawdzić, czy z konfiguracji początkowej maszyny $\mathcal{M}$ da się otrzymać konfigurację końcową. A nic więcej nam nie potrzeba. ∎

Twierdzenie 11.2 (Immerman, Szelepcsenyi, 1987) *Jeśli funkcja $S(n) \geq \log n$ jest konstruowalna, to klasa* NSPACE($S(n)$ *jest zamknięta ze względu na dopełnienie.*

Twierdzenie 11.3 (Paul, Pippenger, Szemeredi, Trotter, 1983)

$$\bigcup_{c>1} \mathrm{NTIME}(c \cdot n) \neq \bigcup_{c>1} \mathrm{DTIME}(c \cdot n).$$

Mamy następujące relacje pomiędzy klasami złożoności i hierarchią Chomsky'ego.

Fakt 11.4 *(1) Klasa języków regularnych pokrywa się z klasą* $\mathrm{DSPACE}(1) = \mathrm{NSPACE}(1)$.

(2) Klasa języków bezkontekstowych zawiera się w klasie $\mathrm{PTIME} = \bigcup_{k\in\mathbb{N}} \mathrm{DTIME}(n^k)$.

(3) Klasa języków kontekstowych pokrywa się z klasą $\mathrm{NSPACE}(n)$.

A zatem z twierdzenia Immermana wynika zamkniętość klasy języków kontekstowych ze względu na dopełnienie.

Następujące skróty oznaczają często spotykane klasy złożoności.

$$\begin{aligned}
\mathrm{LOGSPACE} &= \mathrm{DSPACE}(\log n);\\
\mathrm{NLOGSPACE} &= \mathrm{NSPACE}(\log n);\\
\mathrm{P} = \mathrm{PTIME} &= \textstyle\bigcup_{k\in\mathbb{N}} \mathrm{DTIME}(n^k);\\
\mathrm{NP} &= \textstyle\bigcup_{k\in\mathbb{N}} \mathrm{NTIME}(n^k);\\
\mathrm{PSPACE} &= \textstyle\bigcup_{k\in\mathbb{N}} \mathrm{DSPACE}(n^k) = \bigcup_{k\in\mathbb{N}} \mathrm{NSPACE}(n^k);\\
\mathrm{EXPTIME} &= \textstyle\bigcup_{k\in\mathbb{N}} \mathrm{DTIME}(2^{n^k});\\
\mathrm{NEXPTIME} &= \textstyle\bigcup_{k\in\mathbb{N}} \mathrm{NTIME}(2^{n^k});\\
\mathrm{EXPSPACE} &= \textstyle\bigcup_{k\in\mathbb{N}} \mathrm{DSPACE}(2^{n^k}) = \bigcup_{k\in\mathbb{N}} \mathrm{NSPACE}(2^{n^k}).
\end{aligned}$$

Zauważmy, że w definicji klas PSPACE i EXPSPACE nie ma znaczenia (nie)determinizm, a to z powodu twierdzenia Savitcha. Z Twierdzenia 10.4 wynikają następujące inkluzje:

$$\begin{aligned}\text{LOGSPACE} \subseteq \text{NLOGSPACE} \subseteq \text{P} \subseteq \text{NP} \\ \subseteq \text{PSPACE} \subseteq \text{EXPTIME} \subseteq \text{NEXPTIME} \subseteq \text{EXPSPACE}.\end{aligned}$$

O żadnej z tych inkluzji nie wiadomo, czy jest właściwa. Wiadomo natomiast, że Twierdzenie 10.3 można uściślić tak: NLOGSPACE $\neq$ PSPACE $\neq$ EXPSPACE i podobnie dla czasu: P $\neq$ EXPTIME i NP $\neq$ NEXPTIME.

Dzięki technice zwanej „paddingiem" 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 11.5 *Jeśli* P = NP *to* EXPTIME = NEXPTIME.

Dowód: Przypuśćmy, że P = NP i niech $L \in$ 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 $L_1 \in$ NP, a zatem także $L_1 \in$ 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 $\mathcal{M}$. ∎

Czas wielomianowy

Mówimy, że język $L_1 \subseteq \mathcal{A}_1^*$ jest *wielomianowo sprowadzalny (redukowalny)* do innego języka $L_2 \subseteq \mathcal{A}_2^*$, gdy istnieje funkcja $f : \mathcal{A}_1 \to \mathcal{A}_2$, obliczalna w czasie wielomianowym, i spełniająca warunek

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

dla dowolnego słowa $w \in \mathcal{A}_1^*$. Piszemy wtedy $L_1 \leq_p L_2$. (Czasem żąda się aby funkcja f była obliczalna w pamięci logarytmicznej. Mówi się wtedy o sprowadzalności logarytmicznej i używa symbolu $\leq_{\log}$. Najważniejsze wyniki są takie same, niezależnie od tego, czy mowa o sprowadzalności wielomianowej czy logarytmicznej.)

Mówimy, że język L jest *trudny* dla klasy złożoności $\mathcal{K}$, 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).

Następujący oczywisty fakt wskazuje na znaczenie problemów NP-zupełnych dla hipotezy P = NP.

Fakt 11.6

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 11.7 *Problem* SAT *jest* NP*-zupełny.*

Dowód: Dla dowolnej niedeterministycznej maszyny Turinga $\mathcal{M}$, działającej w czasie wielomianowym i dowolnego słowa w buduje 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 czasie wielomianowym ze względu na długość słowa w. Zatem dowolny język $L \in NP$ sprowadzamy w ten sposób do problemu SAT.

Pozosteje opisać budowę formuły $\Phi_{\mathcal{M},w}$. Załóżmy, że nasza maszyna działa w czasie $w(n)$, ma k stanów, r taśm i używa alfabetu Σ. Niech $|w| = n$. Formuła $\Phi_{\mathcal{M},w}$ jest zbudowana ze zmiennych zdaniowych

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

Zamierzone znaczenie tych zmiennych zdaniowych jest następujące. Interesują nas te wartościowania ρ, które dla pewnego obliczenia maszyny spełniają warunki:

1. $\rho(p_{i,j,\ell,a}) = 1$, gdy po i krokach obliczenia, na j-tym miejscu ℓ-tej taśmy stoi symbol a.

2. $\rho(q_{i,s}) = 1$, gdy po i krokach obliczenia maszyna jest w stanie s.

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.

Nasza formuła jest długą koniunkcją takich członów, których spełnienie jest na raz 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);

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 $i, j_1, j_2 \leq w(n)$ 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 (a zatem wielomianowego czasu). ∎

Wykład 12

NP-zupełność

Zacznijmy od następującego oczywistego wniosku z Twierdzenia 11.7.

Wniosek 12.1 *Jeśli* SAT $\in$ P *to* P = NP.

Problem spełnialności pozostaje NP-zupełny nawet w dość szczególnym przypadku. Wystarczy rozważać formuły w koniunkcyjnej postaci normalnej, a nawet tylko takie:

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

przy czym każde α_{ij} jest albo zmienną zdaniową albo jest postaci $\neg p$, gdzie p jest zmienną zdaniową. Ten wariant problemu SAT oznaczamy przez 3-SAT.

Twierdzenie 12.2 *Problem* 3-SAT *jest* NP*-zupełny.*

Dowód: Ćwiczenie. ∎

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

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.

Twierdzenie 12.3 *Problem kolorowania grafu jest* NP*-zupełny.*

Dowód: Redukcja problemu 3-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:

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

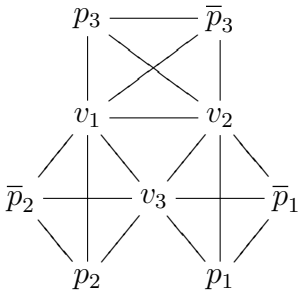
Krawędzie grafu będą takie:

- (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 $\ell = 1, 2, 3$.

Na przykład, 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 w naszym grafie będzie podgraf takiego kształtu:



Wierzchołek f_1 reprezentujący pierwszy człon koniunkcji jest połączony m.in. 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 kolory wierzchołków $p_3, p_5, \overline{p}_1$ są specjalne, a kolory wierzchołków $\overline{p}_3, \overline{p}_5, p_1$ są zwykłe. Wierzchołek f_k jest z nimi połączony, więc nie może przyjąć żadnego z tych 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. ∎

Kilka innych przykładów zadań NP-zupełnych:

- Problem pokrycia wierzchołkowego: *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?*
- Problem kliki: *Dany graf $G = (V, E)$ i liczba k. Czy G ma k-elementowy podgraf pełny (klikę)?*
- Problem cyklu Hamiltona: *Czy dany graf ma cykl Hamiltona?*
- Problem plecaka: *Dane są liczby $i_1, \ldots, i_k$, rozmiar plecaka b i ograniczenie dolne k. Czy istnieje taki podzbiór $A \subseteq \{1, \ldots, k\}$, że $k \leq \sum_{j \in A} i_j \leq b$?*

Klasy co-NP i PSPACE

Nie wiadomo, czy klasa NP jest zamknięta ze względu na dopełnienie. Przez co-NP oznaczamy klasę dopełnień języków z klasy NP.

Fakt 12.4 *Jeśli L jest* NP*-zupełny, oraz $L \in$* co-NP, to NP = co-NP.

Dowód: Dla dowolnego $L' \in$ NP mamy $L' \leq_p L$, więc także $-L' \leq_p -L$. Stąd łatwo widzieć, że $-L' \in$ co-NP. ∎

Klasycznym przykładem języka należącego do iloczynu NP ∩ co-NP, jest zbiór PRIMES binarnych reprezentacji liczb pierwszych. Długo przypuszczano, że może to być problem NP-zupełny. Ostatnio okazało się jednak, że rozpoznawanie liczb pierwszych jest możliwe w deterministycznym czasie wielomianowym.

Twierdzenie 12.5 (Agrawal, Kayal, Saxena, 2002) PRIMES $\in$ P.

Na koniec jeszcze przykład problemu, który jest PSPACE-zupełny. *Formuły zdaniowe drugiego rzędu* definiujemy przez indukcję tak:

- Zmienne zdaniowe $p, q, r, \ldots$ są formułami;
- 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 formuły przy danym wartościowaniu definiuje się jak zwykle, przyjmując, że wartość formuł $\forall p\, \alpha$ i $\exists p\, \alpha$ jest odpowiednio minimum i maksimum z wartości $\alpha[p := 0]$ i $\alpha[p := 1]$. *Tautologią* nazywamy formułę prawdziwą przy każdym wartościowaniu. Na przykład $\exists p(\neg q \vee p)$ jest tautologią.

Twierdzenie 12.6 *Problem „Czy dana zdaniowa formuła drugiego rzędu jest tautologią" jest* PSPACE*-zupełny.*

Inne klasy złożoności

Oprócz „klasycznych" podejść do teorii złożoności (badanie czasu i pamięci potrzebnych dla maszyny Turinga) rozważane są także inne modele, zwłaszcza probabilistyczne i równoległe. Oto krótka informacja o często spotykanych klasach złożoności.

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. Wystarczy aby

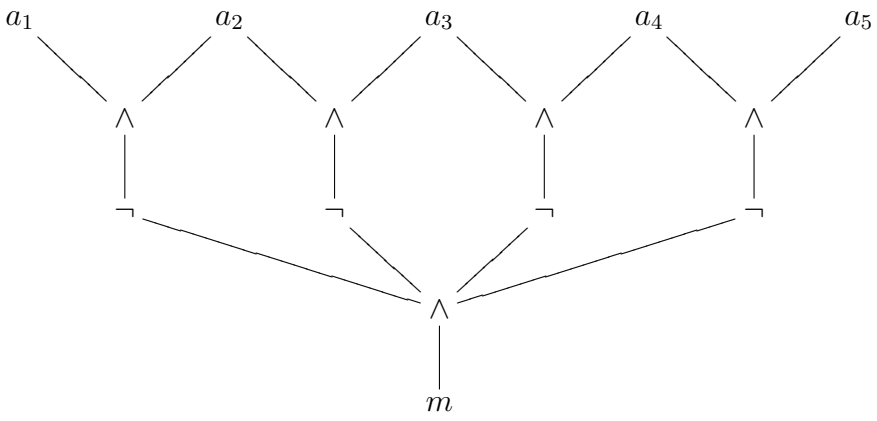
$$P(w \in L(\mathcal{M})) = 0, \quad \text{gdy } w \notin L;$$
$$P(w \in L(\mathcal{M})) \geq \frac{1}{2}, \quad \text{gdy } w \in L,$$

i (powtarzając wielokrotnie próby) można ustalić z dowolnie dużym prawdopodobieństwem, czy $w \in L$. 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*. Klasę RP $\cap$ $-$RP oznaczamy przez ZPP, i mówimy że języki $L \in$ ZPP mają wielomianowe algorytmy typu *Las Vegas*.

Alternatywne podejście polega na rozważaniu zwykłych niedeterministycznych maszyn Turinga i badaniu proporcji liczby obliczeń akceptujących do obliczeń odrzucających (klasy PP i BPP).

Sieci boole'owskie

Zamiast definicji sieci narysujmy przykład. W grafie poniżej, wierzchołki a_i, są *wejściowe* a wierzchołek m jest końcowy. Podając na wejście wartości logiczne zmiennych a_i otrzymujemy w wyniku wartość logiczną pewnej formuły. Powiemy, że sieć *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 naszym przykładzie $n = 5$, 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 akceptowaniu języka potrzebujemy nieskończonej rodziny sieci $C_0, C_1, C_2, \dots$ w której sieć C_n ma n wejść. Na dodatek te sieci powinny być efektywnie konstruowalne — zwykle żąda się aby konstrukcja sieci C_n była możliwa w pamięci $\log n$.

Interesuje nas rozmiar sieci (liczba wierzchołków) i jej wysokość (maksymalna długość drogi od wejścia do wyjscia), oczywiście jako funkcje zmiennej 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 akceptowane przez wielomianowe sieci o stałej wysokości. Sławne (i trudne) twierdzenie jest takie:

Twierdzenie 12.7 (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.

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).

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. Ma ona stany egzystencjalne i uniwersalne. *Obliczeniem* takiej maszyny jest drzewo konfiguracji. Jego korzeniem jest konfiguracja początkowa. Wierzchołki odpowiadające konfiguracjom ze stanem 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 użycia rekursji.) Klasy złożoności dla obliczeń alternujących oznaczamy przez ASPACE($S(n)$) i ATIME($S(n)$). Używamy też oznaczeń ALOGSPACE, APTIME, itp.

Twierdzenie 12.8

- ALOGSPACE = PTIME*;*
- APTIME = PSPACE*;*
- APSPACE = EXPTIME.

Podobne związki zachodzą dla innych klas: alternacja zawsze powoduje przejście w hierarchii „o jeden szczebel".

Podziękowanie

Dziekuję Pani Marii Fronczak i Panu Piotrowi Achingerowi za wykrycie licznych pomyłek w treści tych wykładów.

Ostatnia zmiana 23 marca 2006 o godzinie 11:43.