Modele Obliczeń

Notatki

„Liczba nieparzysta to taka, że zawsze jest jeden, chyba że jest zero.”

Maciej Mikołajczak
Kraków, 2025

Spis treści

1. Języki

2025-02-18

Notacja. Będziemy rozważać alfabet Σ i słowa zbudowane z jego elementów. Zazwyczaj interesuje nas $\Sigma = \{0, 1\}$. Wprowadzamy oznaczenia:

- Σ^i – zbiór słów długości i.
- $\Sigma^* = \bigcup_{i=0}^{\infty} \Sigma^i$ – zbiór wszystkich słów nad Σ.
- $\Sigma^+ = \bigcup_{i=1}^{\infty} \Sigma^i$ – zbiór niepustych słów.
- ε – puste słowo (czyli $\Sigma^0 = \{\varepsilon\}$).

Definicja 1. Językiem nad Σ nazywamy dowolne $L \subseteq \Sigma^*$ – podzbiór słów. Będzie nas interesować, które nieskończone języki da się opisać w skończony sposób (to znaczy stworzyć program, który dla danego słowa stwierdza, czy należy ono do języka).

Uwaga. Programy są skończone, więc jest ich przeliczalnie wiele. Języków jest natomiast nieprzeliczalnie wiele. Zatem muszą istnieć języki, których nie da się opisać.

Przykład. Mamy problem decyzyjny: czy liczba x jest parzysta. Zapisując liczbę w postaci binarnej widzimy, że liczby naturalne są podzbiorem Σ^* (językiem), a nasze pytanie sprowadza się do sprawdzenia, czy dane słowo należy do tego języka. Podobnie możemy mówić np. o problemach grafowych kodując graf jako macierz incydencji.

Notacja. Słowa z Σ^n są właściwie (na co wskazuje notacja) funkcjami $n \to \Sigma$. Będziemy zatem często pisać $v(i)$ dla $v \in \Sigma^n$ i $i \in n$. Wprowadzamy też oznaczenie na rozmiar słowa $|v|$.

Definicja 2. Konkatenacją słów nazywamy funkcję $\cdot : \Sigma^* \times \Sigma^* \to \Sigma^*$ zadaną przez

$$(v \cdot w)(i) = \begin{cases} v(i), & i < |v| \\ w(i - |v|), & |v| \le i < |v| + |w| \end{cases}.$$

Jest to po prostu sklejenie dwóch słów. W zapisie często pomijamy kropkę i piszemy vw.

Definicja 3. Słowo x jest prefiksem słowa v, jeśli istnieje takie $y \in \Sigma^*$, że $xy = v$. Oznaczamy to $x \sqsubseteq v$. Prefiks jest właściwy, jeśli nie jest pełnym słowem – to oznaczamy $x \sqsubset v$. Analogicznie definiujemy pojęcie sufiksu.

Propozycja 1. Konkatenacja spełnia własności:

- łączność $(ab)c = a(bc)$.
- ε jest elementem neutralnym $\varepsilon a = a\varepsilon = a$.
- $|ab| = |a| + |b|$.

Zatem $(\Sigma^*, \cdot, \varepsilon)$ jest monoidem.

Definicja 4. Definiujemy potęgę słowa w jako

$$w^n = \begin{cases} \varepsilon, & n = 0 \\ w^{n-1}w, & n > 0 \end{cases}.$$

Notacja. Na językach L_1, L_2 możemy wprowadzić operacje teoriomnogościowe sumy $L_1 \cup L_2$, przecięcia $L_1 \cap L_2$ i dopełnienia $\Sigma^* \setminus L_1$.

Definicja 5. Katenacją języków L_1, L_2 nazywamy zbiór (język) $L_1 \cdot L_2 = \{w_1 w_2 : w_1 \in L_1, w_2 \in L_2\}$. Mamy $L\emptyset = \emptyset L = \emptyset$ i $L\{\varepsilon\} = \{\varepsilon\}L = L$.

Definicja 6. Definiujemy potęgę języka L jako

$$L^n = \begin{cases} \{\varepsilon\}, & n = 0 \\ L^{n-1}L, & n > 0 \end{cases}.$$

Definicja 7. Gwiazdką Kleene'ego nazywamy operację

$$L^* = \bigcup_{i=0}^{\infty} L^i,$$

gdzie L jest językiem. Definiujemy też operację

$$L^+ = \bigcup_{i=1}^{\infty} L^i.$$

Lemat 1 (Lemat Ardena). Niech $A, B, X \subseteq \Sigma^*$ będą związane równaniem $AX \cup B = X$ ($XA \cup B = X$). W takiej sytuacji język $X = A^*B$ ($X = BA^*$) jest najmniejszym (w sensie inkluzji) rozwiązaniem tego równania. Jeśli $\varepsilon \notin A$, to jest to jedyne rozwiązanie.

Dowód.

$$A\left(A^*B\right) \cup B = A\left(\left(A^+ \cup \{\varepsilon\}\right)B\right) \cup B = A\left(A^+B \cup B\right) \cup B = AA^+B \cup AB \cup B = A^*B.$$

Niech X będzie rozwiązaniem. Mamy $A^*B = \bigcup_{i=0}^{\infty} A^iB$. Indukcją po i pokażemy, że $A^iB \subseteq X$. Mamy $A^0B = B \subseteq X$. Jeśli $A^iB \subseteq X$, to z naszego równania mamy $AA^iB = A^{i+1}B \subseteq X$. Z tego wynika minimalność naszego rozwiązania.

Załóżmy, że $\varepsilon \notin A$ i istnieje $v \in X \setminus A^*B$ będące najkrótszym słowem w tym zbiorze. W szczególności $v \notin B$, a więc $v = uw$, gdzie $u \in A$, $w \in X$. Mamy $w \notin A^*B$, bo inaczej $v \in AA^*B \subseteq A^*B$. Zatem w spełnia te same założenia co v i jest ostro krótsze, bo $\varepsilon \notin A$. Uzyskana sprzeczność kończy dowód.

Drugą wersję lematu dowodzimy analogicznie. □

Notacja. Wprowadzamy oznaczenie $a \leq b$ oznaczające $a + b = b$.

Definicja 8. Algebrą Kleene'ego nazywamy krotkę $(A, +, \cdot, *, 0, 1)$ spełniającą własności:

1. $(a + b) + c = a + (b + c)$.
2. $a + b = b + a$.
3. $a + a = a$.
4. $a + 0 = a$.
5. $(ab)\,c = a\,(bc)$.
6. $a \cdot 1 = 1 \cdot a = a$.
7. $a \cdot 0 = 0 \cdot a = 0$.
8. $(b + c)\,a = ba + ca$.
9. $a\,(b + c) = ab + ac$.
10. $1 + aa^* \leq a^*$.
11. $1 + a^*a \leq a^*$.
12. $ax + b \leq x \implies a^*b \leq x$.
13. $xa + b \leq x \implies ba^* \leq x$.

Propozycja 2. Krotka $(\mathcal{P}(\Sigma^*), \cup, \cdot, *, \emptyset, \{\varepsilon\})$ jest algebrą Kleene'ego.

Twierdzenie 1 (Kozen)**.** Każde równanie prawdziwe w algebrze Kleene'ego jest dowodliwe z jej aksjomatów.

Propozycja 3. $\leq$ w algebrze Kleene'ego jest porządkiem częściowym. Ponadto wszystkie operacja są względem niego monotoniczne: dla dowolnych a, b, c będących elementami algebry Kleene'ego zachodzi

$$a \leq b \implies ac \leq bc,$$
$$a \leq b \implies ca \leq cb,$$
$$a \leq b \implies a + c \leq b + c,$$
$$a \leq b \implies a^* \leq b^*.$$

Dowód. Zwrotność daje nam aksjomat $a + a = a$. Zakładając $x \leq y \leq z$ mamy

$$x + z = x + (y + z) = (x + y) + z = y + z = z,$$

a więc $x \leq z$ – przechodniość. Zakładając $x \leq y, y \leq x$ mamy

$$y = x + y = y + x = x,$$

a to jest antysymetryczność. Teraz rozpisujemy

$$ac + bc = (a + b)\, c = bc,$$
$$ca + cb = c\,(a + b) = cb$$
$$a + c + b + c = (a + b) + (c + c) = b + c,$$

To są trzy pierwsze wynikania. Z nich mamy pierwszą nierówność w $1 + ab^* \leq 1 + bb^* \leq b^*$, a to daje nam $a^* = a^* \cdot 1 \leq b^*$. □

Lemat 2. W algebrze Kleene'ego mamy

$$ax \leq xb \implies a^*x \leq xb^*.$$

Dowód. Z założenia dostajemy $axb^* \leq xbb^*$, a więc

$$x + a\,(xb^*) \leq x + xbb^* = x\,(1 + bb^*) \leq xb^*.$$

Nierówność $x + a\,(xb^*) \leq xb^*$ daje nam $a^*x \leq xb^*$. □

2. Automaty skończone

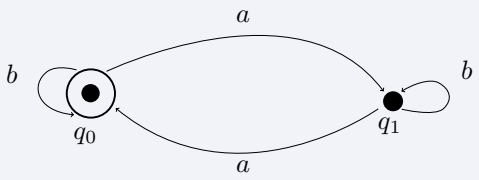
2025-02-20

Definicja 9. Deterministyczny automat skończony (DFA – deterministic finite automaton) to krotka $A = (Q, \Sigma, \delta, s, F)$, gdzie:

- Q – skończony zbiór stanów,
- Σ – skończony alfabet,
- $\delta : Q \times \Sigma \to Q$ – funkcja przejścia,
- $s \in Q$ – stan początkowy,
- $F \subseteq Q$ – stany końcowe (akceptujące).

Automat zaczyna od zadanego stanu s i dostaje kolejne litery alfabetu, na podstawie których przechodzi do nowych stanów w sposób zadany przez δ. Będą nas interesować sytuacje, w których ostatecznie znajdziemy się w stanie należącym do F.

Przykład. DFA łatwo przedstawia się za pomocą grafu, na przykład takiego jak na Rysunku 1.



Rysunek 1: Graf DFA o stanach $\{q_0, q_1\}$ i alfabecie $\{a, b\}$. Mamy stan początkowy q_0 i zbiór stanów końcowych $\{q_0\}$. Jest też $\delta = \{(q_0, b, q_0), (q_1, b, q_1), (q_0, a, q_1), (q_1, a, q_0)\}$.

Pomysł. Dla zadanego DFA będziemy rozważać słowa, które po przepuszczeniu przez DFA produkują stan końcowy. Będziemy zajmować się językami opisanymi w taki sposób.

Definicja 10. Definiujemy funkcję $\hat{\delta} : Q \times \Sigma^* \to Q$ w następujący sposób

$$\hat{\delta}(q, w) = \begin{cases} q, & w = \varepsilon \\ \delta\left(\hat{\delta}(q, x), a\right), & w = xa, a \in \Sigma, x \in \Sigma^* \end{cases}.$$

Jest to funkcja określająca stan automatu po wielu krokach (zadanych danym słowem).

Definicja 11. Niech $A = (Q, \Sigma, \delta, s, F)$ będzie DFA. Językiem akceptowanym przez A nazywamy język $L(A) = \left\{w \in \Sigma^* : \hat{\delta}(s, w) \in F\right\}$.

Przykład. Dla DFA z Rysunku 1 zachodzi $L(A) = \left\{w \in \{a + b\}^* : \#_a(w) = 0 \pmod 2\right\}$.

Dowód. Oznaczmy $L' = \left\{w \in \{a + b\}^* : \#_a(w) = 0 \pmod 2\right\}$. Indukcją po liczbie a w słowie $w \in L'$ dowodzimy, że $\hat{\delta}(q_0, w) = q_0$ oraz $\hat{\delta}(q_1, w) = q_1$. Z pierwszej równości wynika $L' \subseteq L(A)$. Z drugiej natomiast wynika, że $L'^c \cap L(A) = \emptyset$. To daje nam tezę.

Definicja 12. Niedeterministyczny automat skończony (NFA – nondeterministic finite automaton) to krotka $A = (Q, \Sigma, \delta, S, F)$, gdzie:

- Q – skończony zbiór stanów,
- Σ – skończony alfabet,
- $\delta \subseteq Q \times \Sigma \times Q$ – relacja przejścia,
- $S \subseteq Q$ – zbiór stanów początkowych,
- $F \subseteq Q$ – stany końcowe (akceptujące).

Automat niedeterministyczny różni się od deterministycznego tym, że zaczyna w wielu stanach początkowych i wykonuje przejścia zgodne z δ, która tym razem jest relacją – dla zadanego stanu i litery może być wiele poprawnych przejść (lub nie być żadnego).

Definicja 13. Definiujemy funkcję $\tilde{\delta} : \mathcal{P}(Q) \times \Sigma \to \mathcal{P}(Q)$ w następujący sposób

$$\tilde{\delta}(B, a) = \{q \in Q : \exists_{q_b \in B} \ (q_b, a, q) \in \delta\}.$$

Jest to funkcja określająca możliwe stany dla zadanych stanów początkowych i litery.

Definicja 14. Analogicznie jak przedtem definiujemy funkcję $\hat{\delta} : \mathcal{P}(Q) \times \Sigma^* \to \mathcal{P}(Q)$.

$$\hat{\delta}(B, w) = \begin{cases} B, & w = \varepsilon \\ \tilde{\delta}\left(\hat{\delta}(B, x), a\right), & w = xa, a \in \Sigma, x \in \Sigma^* \end{cases}.$$

Definicja 15. Niech $A = (Q, \Sigma, \delta, S, F)$ będzie NFA. Językiem akceptowanym przez A nazywamy język $L(A) = \left\{w \in \Sigma^* : \hat{\delta}(S, w) \cap F \neq \emptyset\right\}$.

Twierdzenie 2. Niech $L \subseteq \Sigma^*$. Mamy $L = L(A_D)$ dla pewnego DFA A_D wtedy i tylko wtedy, gdy $L = L(A_N)$ dla pewnego NFA A_N.

Dowód. ($\Longrightarrow$) Jeśli DFA $(Q, \Sigma, \delta, s, F)$ akceptuje L, to NFA $(Q, \Sigma, \delta, \{s\}, F)$ również (każdy DFA jest NFA z minimalną różnicą typu krotki).

($\Longleftarrow$) Konstrukcja potęgowa – jeśli $A_N = (Q, \Sigma, \delta, S, F)$ akceptuje L, to akceptuje je też DFA $(\mathcal{P}(Q), \Sigma, \delta_D, S, \mathcal{F})$, gdzie $\delta_D = \tilde{\delta}$ (czyli $\hat{\delta}_D = \hat{\delta}$) oraz $\mathcal{F} = \{B \in \mathcal{P}(Q) : B \cap F \neq \emptyset\}$. □

Twierdzenie 3. Dla każdego $n \in \mathbb{N}_+$ istnieje NFA A_n o $n+1$ stanach taki, że DFA akceptujący $L(A_n)$ ma co najmniej 2^n stanów.

Dowód. Ustalmy alfabet $\Sigma = \{0, 1\}$. Niech A_n będzie NFA akceptującym język, w którym n-ta litera od końca to 1. Jest on zadany grafem jak na Rysunku 2 i ma dokładnie $n+1$ stanów.

Rysunek 2: Graf omawianego NFA. Strzałka bez litery oznacza stan początkowy.

Załóżmy, że istnieje DFA A_D taki, że $L(A_D) = L(A_n)$ oraz liczba jego stanów jest ostro mniejsza od 2^n. n-bitowych ciągów jest 2^n, a więc muszą istnieć takie dwa różne ciągi $\overline{a}, \overline{b} \in \{0, 1\}^n$, którym A_D przypisuje ten sam stan. Niech i będzie pewnym indeksem takim, że $a_i \neq b_i$. Bez straty ogólności $a_i = 1$ i $b_i = 0$. Rozważmy ciągi $\overline{a'}, \overline{b'}$ powstałe z poprzednich ciągów poprzez dopisanie do nich i zer. Takim ciągom A_D również przypisze ten sam stan (bo przypisze ich prefiksom, a potem nie mogą się rozejść, bo automat jest deterministyczny). Do tego litera, na której się różnią, jest n-ta od końca. Zatem $\overline{a'} \in L(A_n)$ i $\overline{b'} \notin L(A_n)$ – sprzeczność, bo A_D daje im ten sam stan. □

Uwaga. Widzimy, że DFA i NFA są tak samo mocne – akceptują dokładnie takie same rodziny języków. Jednak DFA wymaga wykładniczo więcej stanów.

Definicja 16. Definiujemy ε-NFA tak samo jak NFA, z tym, że relacja przejścia ma sygnaturę $Q \times (\Sigma \cup \{\varepsilon\}) \times Q$ (zakładamy, że $\varepsilon \notin \Sigma$). Oznacza to, że ε-NFA może wykonywać przejścia na słowie pustym, czyli zmienić stan bez „zużycia" litery z wejścia.

Funkcję $\tilde{\delta}$ definiujemy identycznie jak dla NFA.

Definicja 17. Definiujemy ε-domknięcie zbioru stanów B w automacie A jako zbiór $B^{A,\varepsilon}$ taki, że

$$B^{A,\varepsilon} = \bigcup_{i=0}^{\infty} B_i^{A,\varepsilon},$$

gdzie

$$B_i^{A,\varepsilon} = \begin{cases} B, & i = 0 \\ \tilde{\delta}\left(B_{i-1}^{A,\varepsilon}, \varepsilon\right), & i > 0 \end{cases}.$$

Intuicyjnie, są to wszystkie stany, do których da się dojść z B za pomocą przejść ε.

Definicja 18. Analogicznie jak przedtem dla ε-NFA A definiujemy funkcję $\hat{\delta} : \mathcal{P}(Q) \times \Sigma^* \to \mathcal{P}(Q)$.

$$\hat{\delta}(B, w) = \begin{cases} B^{A,\varepsilon}, & w = \varepsilon \\ \tilde{\delta}\left(\hat{\delta}(B, x), a\right)^{A,\varepsilon}, & w = xa, a \in \Sigma, x \in \Sigma^* \end{cases}.$$

Jest to taka sama funkcja jak dla NFA z tym, że po zużyciu każdej litery dokonujemy ε-domknięcia.

Język akceptowany przez ε-NFA definiujemy identycznie jak dla zwykłego NFA.

Twierdzenie 4. Niech $L \subseteq \Sigma^*$. Mamy $L = L(A_D)$ dla pewnego DFA A_D wtedy i tylko wtedy, gdy $L = L(A_N)$ dla pewnego ε-NFA A_N.

Dowód. ($\implies$) Podobnie jak w przypadku NFA, każde DFA jest NFA, które z kolei jest ε-NFA.

($\impliedby$) Jeśli $A_N = (Q, \Sigma, \delta, S, F)$ akceptuje L, to akceptuje je też DFA $\left(\mathcal{P}(Q), \Sigma, \delta_D, S^{A_N,\varepsilon}, \mathcal{F}\right)$, gdzie $\delta_D = \tilde{\delta}^{A_N,\varepsilon}$ (czyli $\hat{\delta}_D = \hat{\delta}$) oraz $\mathcal{F} = \{B \in \mathcal{P}(Q) : B \cap F \neq \emptyset\}$. □

3. Wyrażenia regularne

2025-02-22

Definicja 19. Wyrażenia regularne nad Σ to najmniejszy język RE_Σ nad alfabetem $\Sigma' = \Sigma \cup \{+, \cdot, *, 0, 1, (,)\}$ taki, że

- $0, 1$ oraz wszystkie $a \in \Sigma$ są elementami RE_Σ.
- jeśli $\alpha, \alpha_1, \alpha_2 \in \mathrm{RE}_\Sigma$, to również $\alpha_1 + \alpha_2, \alpha_1 \cdot \alpha_2, \alpha^*, (\alpha) \in \mathrm{RE}_\Sigma$.

Definicja 20. Interpretacją wyrażeń regularnych jest funkcja $L : \mathrm{RE}_\Sigma \to \mathcal{P}(\Sigma^*)$ zadana przez

- $L(0) = \emptyset, L(1) = \{\varepsilon\}, \forall_{a \in \Sigma} L(a) = \{a\}$.
- $L(\alpha + \beta) = L(\alpha) \cup L(\beta)$.
- $L(\alpha \cdot \beta) = L(\alpha) \cdot L(\beta)$.
- $L(\alpha^*) = (L(\alpha))^*$.
- $L((\alpha)) = L(\alpha)$.

Intuicyjnie wyrażenia regularne określają język – budujemy słowa za pomocą konkatenowania, sumowania i wielokrotnego powtarzania bloków, gdzie blokami atomowymi są pojedyncze litery.

Kolejność wykonywania operacji ustalają nawiasy. W przeciwnym wypadku wygląda tak: najpierw gwiazdka Kleene'ego $*$, potem konkatenacja $\cdot$, na koniec suma $+$.

Twierdzenie 5. Niech $L \subseteq \Sigma^*$. Mamy $L = L(R)$ dla pewnego wyrażenia regularnego R wtedy i tylko wtedy, gdy $L = L(A)$ dla pewnego DFA A (lub równoważnie NFA, ε-NFA).

Dowód. ($\implies$) Stosując indukcję po długości wyrażenia regularnego wykażemy, że istnieje ε-NFA, które akceptuje ten sam język. Będziemy konstruować ε-NFA, które mają dokładnie jeden stan początkowy i końcowy takie, że nie istnieją żadne przejścia do stanu początkowego ani ze stanu końcowego. Do tego będą istniały co najwyżej 2 przejścia z każdego stanu, a rozmiar całego ε-NFA będzie liniowy od rozmiaru wyrażenia regularnego.

Zaczniemy od przypadków bazowych. Jeśli $R = 0, 1, a$, to konstruujemy automat, który odpowiednio: ma sam stan startowy, ma ε-przejście ze stanu startowego do końcowego, ma przejście ze stanu

startowego do końcowego z literą a. Teraz przechodzimy do kroku indukcyjnego:

- Jeśli $R = \alpha_1 + \alpha_2$, to istnieją ε-NFA A_1, A_2 takie, że $L(A_1) = L(\alpha_1), L(A_2) = L(\alpha_2)$. Tworzymy automat A_3, w którym mamy wszystkie stany z A_1 i A_2, stan startowy ma ε-przejścia do stanów startowych A_1 i A_2, a ich stany końcowe mają ε-przejścia do stanu końcowego.
- Jeśli $R = \alpha_1 \cdot \alpha_2$, to ponownie rozważamy automaty A_1, A_2 i tworzymy A_3 – stanem startowym jest stan startowy A_1, jego stan końcowy ma ε-przejście do stanu startowego A_2, a stanem końcowym jest stan końcowy A_2.
- Jeśli $R = \alpha^*$, to istnieje ε-NFA A takie, że $L(A) = L(\alpha)$. Tworzymy automat B, w którym stan startowy ma ε-przejścia do stanu startowego A i stanu końcowego, a stan końcowy A ma ε-przejścia do stanu końcowego i swojego stanu startowego.

($\Longleftarrow$) Niech DFA $A = (Q, \Sigma, \delta, s, F)$. Wprowadźmy porządek na stanach $Q = \{q_1, \ldots, q_n\}$. Zauważmy, że $w \in L(A)$ jest równoważne istnieniu ścieżki na stanach $s \rightsquigarrow q_j \in F$, którą automat przechodzi dla tego słowa. Skonstruujemy wyrażenie regularne, które generuje dokładnie takie słowa.

Niech $R_{i,j}^{(k)}$ będzie wyrażeniem regularnym, którego językiem są słowa, które generują w A ścieżkę od stanu q_i do q_j i korzystają pomiędzy tylko ze stanów $\{q_1, \ldots, q_k\}$. Zdefiniujemy takie wyrażenie indukcyjnie. Mamy

$$R_{i,j}^{(0)} = \beta_{i,j} + \sum_{a \in \Sigma : \delta(q_i, a) = q_j} a,$$

gdzie $\beta_{i,j} = \begin{cases} 0, & i \neq j \\ 1, & i = j \end{cases}$.

Teraz dla $0 < k \leq n$ możemy zdefiniować $R_{i,j}^{(k)} = R_{i,j}^{(k-1)} + R_{i,k}^{(k-1)} \left(R_{k,k}^{(k-1)}\right)^* R_{k,j}^{(k-1)}$ – mając do dyspozycji nowy stan q_k możemy albo znaleźć ścieżkę bez niego, albo dojść do niego korzystając tylko z mniejszych stanów, potem wrócić do niego dowolną liczbę razy i ostatecznie wyjść z niego i dojść do końca pozostałymi stanami.

Ostatecznie szukane wyrażenie to $R = \sum_{q_j \in F} R_{i,j}^{(n)}$, gdzie $q_i = s$. □

Definicja 21. Język, który jest akceptowany przez jakieś wyrażenie regularne (lub – równoważnie – skończony automat) nazywamy językiem regularnym.

Definicja 22. Wyrażenia regularne α_1, α_2 nazywamy równoważnymi, jeśli $L(\alpha_1) = L(\alpha_2)$. Oznaczamy to $\alpha_1 = \alpha_2$.

Propozycja 4. Dla wyrażeń regularnych L, M, N zachodzi:

1. $L + M = M + L$.
2. $(L + M) + N = L + (M + N)$.
3. $(LM)N = L(MN)$.
4. $0 + L = L$.
5. $1L = L1 = L$.
6. $0L = L0 = 0$.
7. $L(M + N) = LM + LN$.
8. $(M + N)L = ML + NL$.
9. $L + L = L$.

4. Języki regularne

2025-02-23

Lemat 3 (O pompowaniu). Jeśli L jest językiem regularnym, to istnieje takie $n > 0$ (zależne od L), że dla każdego $w \in L$ z $|w| \geq n$ istnieje postać $w = xyz$ taka, że $|xy| \leq n$, $|y| \geq 1$ i $\forall_{i \in \mathbb{N}}\ xy^i z \in L$. Inaczej mówiąc: musi istnieć takie niepuste y będące blisko początku słowa, że można je „napompować" – powtórzyć dowolną liczbę razy.

Dowód. Niech $A = (Q, \Sigma, \delta, s, F)$ będzie DFA akceptującym L. Ustalmy $n = |Q|$ i weźmy dowolne słowo $w \in L$ takie, że $m = |w| \geq n$. Niech $w = a_0 \dots a_{m-1}$.

Słowo jest akceptowane, a więc istnieje ciąg stanów $q_0, \dots, q_m$ taki, że $q_0 = s$ i $q_m \in F$. Jego długość jest większa niż n, a więc jakiś stan musi się powtarzać. Niech q_i będzie tym stanem, którego pierwsze powtórzenie występuje najwcześniej – niech będzie to q_j. Ustalamy

$$x = a_0 \dots a_{i-1},\ y = a_i \dots a_{j-1},\ z = a_j \dots a_{m-1}.$$

Mamy $|xy| \leq n$, bo inaczej q_i nie powtarzałby się najwcześniej. Do tego z $q_i = q_j$ wynika, że słowo y może wystąpić w akceptowanym słowie dowolną liczbę razy – mamy cykl stanów. Zatem $\forall_{i \in \mathbb{N}}\ xy^i z \in L$. Oczywiście y jest niepuste. □

Twierdzenie 6. Język

$$L = \{a^n b^n : n \in \mathbb{N}\}$$

nie jest regularny.

Dowód. Pokażemy, że L nie spełnia lematu o pompowaniu. Ustalmy dowolne $n > 0$ i rozważmy słowo $w = a^{2n} b^{2n}$. Dla każdego podziału $w = xyz$ takiego, że $|xy| \leq n$ zachodzi $xy = a^k$ dla pewnego k. Wobec tego mamy $y = a^l$ dla pewnego $l > 0$. Gdyby lemat o pompowaniu był spełniony, to $a^{2n-l} b^{2n}$ musiałoby należeć do języka, a tak nie jest. Zatem L nie spełnia lematu o pompowaniu i nie jest regularny. □

Definicja 23. Odwróceniem słowa $w = a_1 \dots a_n$ nazywamy słowo $w^R = a_n \dots a_1$. Odwróceniem języka L nazywamy język $L^R = \left\{w^R : w \in L\right\}$.

Definicja 24. Homomorfizmem słów nazywamy funkcję $h : \Sigma^* \to \Sigma'^*$, która dla zadanego słowa w zamienia każdą jego literę na ustalone słowo: dla $w = a_1 \dots a_n$ mamy $h(w) = h(a_1) \dots h(a_n)$.

Homomorfizm możemy też zastosować na języku: $h(L) = \{h(w) : w \in L\}$.

Twierdzenie 7. Języki regularne są zamknięte na

1. sumę, konkatenację i gwiazdkę Kleene'ego.
2. przecięcie, dopełnienie i różnicę.
3. odwrócenie.

Dowód. Pierwszy punkt wynika z definicji wyrażeń regularnych.

Rozważmy DFA $A_1 = (Q_1, \Sigma, \delta_1, s_1, F_1), A_2 = (Q_2, \Sigma, \delta_2, s_2, F_2)$. Zdefiniujmy DFA

$$B = (Q_1 \times Q_2, \Sigma, \delta', (s_1, s_2), F_1 \times F_2),$$

gdzie $\delta'((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a))$. Jest to automat, który symuluje chodzenie w obu automatach na raz. Widać, że akceptuje dokładnie te słowa, które akceptują oba automaty A_1, A_2. Zatem $L(B) = L(A_1) \cap L(A_2)$, czyli języki regularne są zamknięte na branie przecięć.

Dla DFA $A = (Q, \Sigma, \delta, s, F)$ automat $B = (Q, \Sigma, \delta, s, F^c)$ akceptuje dokładnie te słowa, których nie akceptuje A. Zatem $L(B) = L(A)^c$, czyli języki regularne są zamknięte na branie przecięć.

Zamknięcie na różnicę wynika z równości $L \setminus M = L \cap M^c$.

Zamknięcie na odwrócenie pokazujemy indukcją po długości wyrażenia regularnego. Jeśli język L jest generowany przez wyrażenie regularne E, to konstruujemy E^R spełniające $L\left(E^R\right) = L^R$:

- jeśli $E \in \{0, 1, a\}$, to $L = L^R$ i $E^R = E$.

- jeśli $E = E_1 + E_2$, to $E^R = E_1^R + E_2^R$.
- jeśli $E = E_1 E_2$, to $E^R = E_2^R E_1^R$.
- jeśli $E = E_1^*$, to $E^R = \left(E^R\right)^*$.

□

Twierdzenie 8. Jeśli $h : \Sigma^* \to \Sigma'^*$ jest homomorfizmem, to dla języka regularnego $L \subseteq \Sigma^*$ język $h(L) \subseteq \Sigma'^*$ też jest regularny. Podobnie dla regularnego $L' \subseteq \Sigma'^*$ język $h^{-1}(L') \subseteq \Sigma^*$ też jest regularny.

Dowód. Dla wyrażenia regularnego R oznaczmy przez $h(R)$ wyrażenie regularne, w którym każde wystąpienie $a \in \Sigma$ zostaje zastąpione przez $h(a) \in \mathrm{RE}_{\Sigma'}$ (jest to poprawne wyrażenie regularne, bo jest konkatenacją poprawnych wyrażeń regularnych). Mamy $L(h(R)) = h(L(R))$, co dowodzimy indukcją po długości wyrażenia regularnego. Jeśli $R \in \{0, 1, a\}$, to homomorfizm robi dokładnie to, co chcemy. Jeśli $R = R_1 + R_2$, to

$$L(h(R)) = L(h(R_1) + h(R_2)) = L(h(R_1)) \cup L(h(R_2)) = h(L(R_1)) \cup h(L(R_2)) = h(L(R)),$$

gdzie pierwsze przejście wynika z tego, że h zmienia tylko litery. Pozostałe przypadki wyglądają podobnie. Zatem dla $L = L(R)$ znaleźliśmy wyrażenie regularne akceptujące język $h(L)$.

Dla dowodu drugiej części rozważmy DFA $A = (Q, \Sigma', \delta, s, F)$ akceptujące język L'. Zdefiniujmy DFA $B = (Q, \Sigma, \gamma, s, F)$, gdzie $\gamma(q, a) = \hat{\delta}(q, h(a))$. B symuluje zachowanie A na literach z Σ – przepuszcza je przez homomorfizm i zachowuje się jak A na otrzymanym słowie. Łatwo zauważyć, że $\hat{\gamma}(s, w) = \hat{\delta}(s, h(w))$ (indukcja po długości słowa). Zatem B akceptuje słowo w wtedy i tylko wtedy, gdy A akceptuje $h(w)$ – zatem $L(B) = h^{-1}(L')$. □

Definicja 25. Dla ustalonego DFA $A = (Q, \Sigma, \delta, s, F)$ mówimy, że jego stany q_1, q_2 są A-równoważne, jeśli

$$\forall_{w \in \Sigma^*}\ \hat{\delta}(q_1, w) \in F \iff \hat{\delta}(q_2, w) \in F.$$

Jeśli stany nie są A-równoważne, to są A-rozróżnialne, czyli

$$\exists_{w \in \Sigma^*}\ \hat{\delta}(q_1, w) \in F \iff \hat{\delta}(q_2, w) \notin F.$$

Propozycja 5. A-równoważność jest relacją równoważności.

Notacja. Dla DFA A i stanu q przez $[q]_A$ oznaczamy klasę A-równoważności, do której należy q.

Algorytm 1. Dla zadanego DFA A rozważmy następujący algorytm: tworzymy macierz Boolowską M, w której $M_{ij} = 1 \iff q_i, q_j$ są A-rozróżnialne. Robimy to zaczynając od macierzy zerowej i ustalając $M_{ij} = 1$ dla $q_i \in F, q_j \notin F$.

Krok algorytmu wygląda tak: dopóki macierz się zmienia dla każdych dwóch stanów q_i, q_j i każdej litery a, jeśli $\delta(q_i, a) = q_k$ i $\delta(q_j, a) = q_l$ oraz $M_{kl} = 1$, to ustalamy $M_{ij} = 1$.

Tak zadany algorytm zwraca poprawny wynik.

Dowód. Ten algorytm działa mniej więcej jak Floyd-Warshall, zaczynamy od stanów które rozróżnia puste słowo i budujemy dalej – jeśli jakaś litera prowadzi do stanów, które są rozróżniane przez krótsze słowo, to można do niego dopisać tę literę i znaleźć słowo rozróżniające nasze stany.

Jeśli mamy $M_{ij} = 1$, to q_i i q_j są A-rozróżnialne, bo algorytm skonstruował słowo, które je rozróżnia. Załóżmy nie wprost, że stany q_i, q_j są rozróżnialne, a algorytm stwierdził $M_{ij} = 0$. Do tego słowo w świadczące o rozróżnialności jest najkrótszym, dla którego ta sytuacja może zajść. Jeśli $w = \varepsilon$, to z bazy algorytmu te stany będą rozróżnione. Zatem $w = av$ i mamy

$$\left(\hat{\delta}(q_1, w) \in F \iff \hat{\delta}(q_2, w) \notin F\right) \iff \left(\hat{\delta}(\delta(q_1, a), v) \in F \iff \hat{\delta}(\delta(q_2, a), v) \notin F\right).$$

Stany $q_k = \delta(q_1, a)$ i $q_l = \delta(q_2, a)$ są rozróżniane przez v, a więc $M_{kl} = 1$, bo $|v| < |w|$. Widać, że

gdy algorytm ustali $M_{kl} = 1$, to w następnym kroku stwierdzi $M_{ij} = 1$ – sprzeczność. □

Twierdzenie 9. Rozważmy DFA $A = (Q, \Sigma, \delta, s, F)$. Skonstruujmy automat D kolapsując stany A, które są A-równoważne – mamy $D = (\mathcal{Q}, \Sigma, \delta_D, [s]_A, \mathcal{F})$, gdzie $\mathcal{Q}$ i $\mathcal{F}$ oznaczają rodziny klas A-równoważności, do których należą elementy odpowiednio Q i F, a δ_D jest relacją $\mathcal{Q} \times \Sigma \times \mathcal{Q}$, w której

$$([q]_A, a, [g]_A) \in \delta_D \iff \exists_{q_1, q_1 \in Q}\ q_1 \in [q]_A \wedge q_2 \in [g]_A \wedge \delta(q_1, a) = q_2.$$

Następnie zmodyfikujmy D, usuwając stany, do których nie da się dojść ze stanu startowego. W takiej sytuacji D jest DFA i do tego jest najmniejszym na liczbę stanów DFA akceptującym ten sam język, co A.

Dowód. D mógłby nie być DFA, gdyby istniały jakieś A-równoważne stany q_1, q_2 i litera a takie, że q_1 przechodzi z nią do q_3 a q_2 do q_4 i q_3 oraz q_4 są A-rozróżnialne. Wtedy jednak słowo a świadczyłoby o A-rozróżnialności q_1 i q_2, co daje sprzeczność. Jasnym jest, że D akceptuje ten sam język, co A (mamy $\hat{\delta}_D([q]_A, w) = [\hat{\delta}(q, w)]_A$). Pozostało wykazać jego minimalność.

Załóżmy, że istnieje DFA $B = (Q_B, \Sigma, \delta_B, s_B, F_B)$ takie, że $L(B) = L(A)$ i B ma mniej stanów niż D. Zdefiniujmy DFA $C = (\mathcal{Q} \cup Q_B, \Sigma, \delta_D \cup \delta_B, s_B, \mathcal{F} \cup F_B)$. Zauważmy, że dla każdego stanu $q_D \in \mathcal{Q}$ istnieje $q_B \in Q_B$ taki, że q_D i q_B są C-równoważne – D i B akceptują te same języki, a więc na pewno $[s]_A$ i s_B są C-równoważne (gdyby nie były, to jakieś słowo różniłoby te stany, a więc i języki). Z tego wynika, że dla każdego $a \in \Sigma$ stany $\delta([s]_A, a)$ i $\delta(s_B, a)$ są C-równoważne (i tak dalej dla dłuższych słów).

Zatem każdy stan D (jako osiągalny ze stanu startowego) ma stan B, któremu jest C-równoważny. Stanów B jest mniej, a więc istnieją takie stany $q_1, q_2 \in \mathcal{Q}$, że oba są równoważne pewnemu $q_3 \in Q_B$. C-równoważność jest przechodnia, a więc q_1 i q_2 są C-równoważne, czyli też D-równoważne. To jednak jest niemożliwe, bo w naszej konstrukcji skolapsowaliśmy wszystkie równoważne stany w jeden. Uzyskana sprzeczność kończy dowód. □

Uwaga. Powyższe twierdzenie i algorytm dają nam metodę konstrukcji najmniejszego DFA akceptującego dany język.

Definicja 26. Mówimy, że relacja $\equiv$ jest L-kongruentna dla pewnego języka $L \subseteq \Sigma^*$, jeśli

1. $\forall_{v \in \Sigma^*}\ x \equiv y \implies xv \equiv yv$.
2. $x \equiv y \implies (x \in L \iff y \in L)$.

Twierdzenie 10 (Myhill, Nerode)**.** Język L jest regularny wtedy i tylko wtedy, gdy istnieje relacja równoważności $\equiv \subseteq \Sigma^* \times \Sigma^*$ o skończenie wielu klasach równoważności, która jest L-kongruentna.

Dowód. ($\implies$) Regularny język L ma swoje DFA A, na podstawie którego definiujemy

$$x \equiv y \iff \hat{\delta}(s, x) = \hat{\delta}(s, y).$$

Ta relacja jest równoważnością, a do tego dla każdego $v \in \Sigma^*$ i dowolnych $x, y \in \Sigma^*$ zachodzi

$$x \equiv y \iff \hat{\delta}(s, x) = \hat{\delta}(s, y) \implies \hat{\delta}(s, xv) = \hat{\delta}(s, yv) \iff xv \equiv yv,$$

$$x \equiv y \iff \hat{\delta}(s, x) = \hat{\delta}(s, y) \implies (x \in L \iff y \in L).$$

Zatem nasza relacja jest L-kongruentna. Do tego ma skończenie wiele klas równoważności, bo co najwyżej tyle, ile A ma stanów.

($\impliedby$) Tworzymy automat, w którym stany to klasy równoważności naszej relacji $\equiv$ (tutaj ważna jest ich skończona ilość). Jako stany końcowe przyjmujemy te klasy, których elementy należą do języka (drugi warunek z definicji L-kongruentności). Stanem początkowym jest klasa, do której należy ε. Funkcję przejścia definiujemy w następujący sposób: dla zadanej klasy i litery konkatenujemy tę literę do dowolnego słowa z tej klasy i tworzymy przejście do klasy, do której należy wynik. Z pierwszego warunku z definicji L-kongruentności będzie to poprawna funkcja. W ten sposób stworzyliśmy DFA.

Zauważmy, że $\hat{\delta}(s, w)$ jest klasą równoważności, do której należy w – dowód indukcją po długości w, dla słowa pustego wszystko się zgadza z definicji stanu początkowego. Dla $w' = wa$ wiemy z założenia indukcyjnego, że $\hat{\delta}(s, w)$ jest klasą, do której należy w. Przejście po literze a powoduje przejście do klasy wa, bo tak zdefiniowaliśmy δ.

Z tego mamy $L(A) = L$, bo A akceptuje dokładnie te słowa, których klasy równoważności zawierają się w L. □

5. Problemy decyzyjne

2025-02-24

Pomysł. Dla zadanej reprezentacji języka chcemy znaleźć algorytmy, które stwierdzają różne rzeczy o tym języku.

Uwaga. Rozwiązując jakiś problem decyzyjny dla języka regularnego najlepiej mówić o postaci ε-NFA: NFA i DFA to jej szczególne przypadki, a wyrażenie regularne da się liniowo przekształcić do ε-NFA.

Definicja 27. Mając zadany język L i słowo v problem MEMBERSHIP polega na stwierdzeniu, czy $v \in L$.

Twierdzenie 11. Dla języków regularnych problem MEMBERSHIP jest wielomianowy.

Dowód. Pamiętamy zbiór stanów osiągalnych dla kolejnych prefiksów słowa – zaczynamy od ε-dopełnionych stanów początkowych, następnie przechodzimy kolejnymi literami do następnych stanów (i bierzemy ε-dopełnienia). Na koniec sprawdzamy, czy któryś z ostatecznych stanów jest stanem końcowym. Każdy z tych kroków działa wielomianowo. □

Definicja 28. Mając zadany język L problem EMPTINESS polega na stwierdzeniu, czy $L = \emptyset$.

Twierdzenie 12. Dla języków regularnych problem EMPTINESS jest wielomianowy.

Dowód. Wystarczy przejść po krawędziach grafu automatu i sprawdzić, czy da się dojść ze stanów początkowych do jakiegokolwiek stanu końcowego.

Dla wyrażenia regularnego można działać indukcyjnie: z bazowych wyrażeń tylko 0 jest puste, $\alpha + \beta$ jest puste jak oba są puste, $\alpha\beta$ jest puste jak któreś jest puste, α^* nigdy nie jest puste. □

Definicja 29. Mając zadany język L problem INFINITENESS polega na stwierdzeniu, czy $|L| = \aleph_0$.

Twierdzenie 13. Dla języków regularnych problem INFINITENESS jest wielomianowy.

Dowód. Zauważmy, że język jest nieskończony wtedy i tylko wtedy, gdy w grafie jego automatu istnieje cykl (nie składający się z samych ε-przejść), do którego da się dojść ze stanu początkowego i z którego da się dojść do stanu końcowego. Możemy zatem usunąć wszystkie nieosiągalne stany (znajdujemy, np. BFS'em ze stanów startowych), potem wszystkie stany, z których nie da się dojść do stanu końcowego i na koniec sprawdzić, czy w grafie istnieje odpowiedni cykl – jeśli ε-krawędzie dostaną wagę 0 a pozostałe -1, to algorytm Bellmana-Forda stwierdzi, czy istnieje ujemny cykl, a tego szukamy.

Dla wyrażenia regularnego można działać indukcyjnie: aby język był nieskończony na pewno potrzebna jest gwiazdka Kleene'ego, ale mamy $0^* = 1$ i $1^* = 1$, zatem z przypadków bazowych tylko a^* jest nieskończony. Dla $\alpha + \beta$ jedno musi być nieskończone, dla $\alpha\beta$ jedno musi być nieskończone a drugie niepuste, dla α^* pod gwiazdką musi być coś nierównego $0, 1$. □

Definicja 30. Mając zadane języki L_1, L_2 problem EQUIVALENCE polega na stwierdzeniu, czy $L_1 = L_2$.

Twierdzenie 14. Dla języków regularnych problem EQUIVALENCE jest rozstrzygalny, a dla reprezentacji przez DFA jest do tego wielomianowy.

Dowód. Mając zadane DFA A_1, A_2 takie, że $L_1 = L(A_1), L_2 = L(A_2)$ można zauważyć, że

$$L(A_1) = L(A_2) \iff (L(A_1) \cap L(A_2)^c) \cup (L(A_1)^c \cap L(A_2)) = \emptyset.$$

Zatem wystarczy skonstruować dwa automaty dla dopełnienia i dwa dla przecięć (wszystko wielomianowo za pomocą wcześniej wprowadzonych konstrukcji), a następnie sprawdzić pustość obu języków generowanych przez automaty przecięć.

Dla innych reprezentacji wystarczy przekształcić je na DFA (być może wykładniczo) i zastosować ten algorytm. □

6. Gramatyki bezkontekstowe

2025-02-24

Definicja 31. Gramatyka bezkontekstowa (CFG, context-free grammar) to krotka $G = (N, \Sigma, P, S)$, gdzie:

- N - skończony zbiór zmiennych (nieterminale),
- Σ - skończony alfabet (terminale),
- $P \subseteq N \times (N \cup \Sigma)^*$ - skończony zbiór produkcji,
- $S \in N$ - symbol startowy.

Notacja. Produkcję (s, w) będziemy zapisywać $s \to w$. Kilka produkcji $s \to w_1, s \to w_2$ możemy zapisać łącznie: $s \to w_1 \mid w_2$.

Definicja 32. Forma zdaniowa to dowolne słowo z $(N \cup \Sigma)^*$.

Definicja 33. Dla gramatyki bezkontekstowej $G = (N, \Sigma, P, S)$ definiujemy relację $\to_G$ na formach zdaniowych. Mówimy, że $\alpha \to_G \beta$ (możemy uzyskać β z α w jednym kroku), jeśli

$$\exists_{\alpha_1, \alpha_2, \gamma \in (N \cup \Sigma)^*} \exists_{A \in N} \ (A, \gamma) \in P \wedge \alpha = \alpha_1 A \alpha_2 \wedge \beta = \alpha_1 \gamma \alpha_2,$$

czyli β powstaje z α przez zamianę nieterminala na formę zdaniową zgodnie z jakąś produkcją.

Przykład. Język palindromów nad $\{0, 1\}$ można zdefiniować rekurencyjnie: $0, 1, \varepsilon$ są palindromami, dla palindromu P palindromami są $0P0, 1P1$.

Można to wypowiedzieć w języku gramatyk bezkontekstowych: dla $G = (\{S\}, \{0, 1\}, P, S)$ do P należą produkcje $S \to \varepsilon \mid 0 \mid 1$ i $S \to 0S0 \mid 1S1$.

Definicja 34. Przez $\to_G^*$ oznaczamy zwrotne i przechodnie domknięcie $\to_G$, czyli $\alpha \to_G^* \beta$, jeśli możemy uzyskać β z α w pewnej liczbie kroków.

Definicja 35. Język $L(G)$ generowany przez gramatykę $G = (N, \Sigma, P, S)$ to język

$$L(G) = \{w \in \Sigma^* : S \to_G^* w\}.$$

Ogólniej można zdefiniować

$$L(G, A) = \{w \in \Sigma^* : A \to_G^* w\}.$$

Takie języki nazywamy językami bezkontekstowymi (CFL).

Definicja 36. Wywód (derivation) w gramatyce G to ciąg form zdaniowych $\alpha_0, \ldots, \alpha_n$ takich, że $\forall_{i<n} \ \alpha_i \to_G \alpha_{i+1}$.

Uwaga. $v \in L(G)$ wtedy i tylko wtedy, gdy istnieje wywód $\alpha_0, \ldots, \alpha_n$ w G taki, że $\alpha_0 = S$ i $\alpha_n = v$.

Definicja 37. Wywód lewostronny $\alpha_0, \ldots, \alpha_n$ to taki wywód, że dla każdego $i < n$ mamy

$$\alpha_i = xAB_i,$$

gdzie $x \in \Sigma^*$, $A \in N$, $B_i \in (N \cup \Sigma)^*$. Przez A rozumiemy nieterminal, który jest zastępowany w danym kroku wywodu. Zatem w wywodzie lewostronnym zawsze zamieniamy najbardziej lewy nieterminal.

Definicja 38. Drzewo wywodu (parse tree) formy zdaniowej α w gramatyce $G = (N, \Sigma, P, S)$ to ukorzenione drzewo z porządkiem na dzieciach, w którym każdy wierzchołek ma etykietę z $N \cup \Sigma \cup \{\varepsilon\}$. Etykieta korzenia to S. Jeśli wierzchołek ma etykietę A i dzieci $X_1, \ldots, X_n$, to $A \to X_1 \ldots X_n$ jest produkcją w P. Wierzchołki o etykiecie ε są liśćmi i nie mają rodzeństwa. W wyniku konkatenacji etykiet liści otrzymujemy α.

Propozycja 6. Każde drzewo wywodu można jednoznacznie przypisać do wywodu lewostronnego.

Dowód. Mając zadane drzewo wywodu dla formy zdaniowej α można przejść po nim zgodnie z kolejnością dzieci. W ten sposób z symbolu startowego dostajemy α, a w kolejnych formach zdaniowych zmienia się najbardziej lewy nieterminal. Podobnie wywód lewostronny zadaje drzewo wywodu – budujemy drzewo zgodnie z kolejnymi przejściami w wywodzie. □

Definicja 39. Gramatyka jest jednoznaczna (unambiguous), jeśli każde słowo $v \in L(G)$ ma dokładnie jedno drzewo wywodu (wywód lewostronny) w tej gramatyce.

Definicja 40. Język bezkontekstowy jest wewnętrznie niejednoznaczny (inherently ambiguous), jeśli każda generująca go gramatyka jest niejednoznaczna.

Przykład. Rozważmy język L_{BAL} nad $\{(,)\}$ składający się z poprawnych nawiasowań. Generuje go gramatyka $G_{\mathrm{BAL}} = (\{B\}, \{(,)\}, P, B)$, gdzie P zawiera $B \to \varepsilon \mid (B) \mid BB$. Nie jest ona jednoznaczna: dla słowa $()()()$ mamy wywody

$$B \to BB \to (B)\,B \to ()\,B \to ()\,BB \to ()\,(B)\,B \to ()\,()\,B \to ()\,()\,(B) \to ()\,()\,()$$

$$B \to BB \to BBB \to (B)\,BB \to ()\,BB \to ()\,(B)\,B \to ()\,()\,B \to ()\,()\,(B) \to ()\,()\,().$$

Sam język jest jednak jednoznaczny, o czym świadczy gramatyka $G'_{\mathrm{BAL}} = (\{S\}, \{(,)\}, P, S)$, gdzie P zawiera $S \to S(S) \mid \varepsilon$.

7. Postać normalna Chomsky'ego

2025-03-01

Definicja 41. Mówimy, że symbol $X \in N \cup \Sigma$ jest generujący, jeśli $X \to_G^* w$ dla pewnego $w \in \Sigma^*$.

Definicja 42. Mówimy, że symbol $X \in N \cup \Sigma$ jest osiągalny, jeśli $S \to_G^* \alpha X \beta$ dla pewnych form zdaniowych α, β.

Definicja 43. Mówimy, że symbol $X \in N \cup \Sigma$ jest użyteczny, jeśli jest osiągalny i generujący.

Twierdzenie 15. Niech $G = (N, \Sigma, P, S)$ będzie CFG, dla której $L(G) \neq \emptyset$. Zdefiniujmy nową gramatykę $G_1 = (N_1, \Sigma_1, P_1, S)$ powstałą w następujący sposób.

1. Usuwamy z G wszystkie symbole niegenerujące i produkcje je zawierające, otrzymując gramatykę $G_2 = (N_2, \Sigma_2, P_2, S)$.
2. W podobny sposób usuwamy symbole nieosiągalne w G_2.

Gramatyka G_1 nie zawiera symboli bezużytecznych i jest $L(G_1) = L(G)$.

Dowód. Niech $X \in N_1 \cup \Sigma_1$. Wiemy, że $X \to_G^* w$ dla pewnego $w \in \Sigma^*$ (inaczej zostałby usunięty w pierwszym kroku jako niegenerujący). Wszystkie symbole użyte w tym wywodzie również są

generujące, a więc mamy $X \to^*_{G_2} w$. Podobnie X nie został usunięty w drugim kroku, a więc $S \to^*_{G_2} \alpha X \beta$ dla pewnych form zdaniowych α, β. Wszystkie symbole w tym wywodzie są osiągalne, a więc $S \to^*_{G_1} \alpha X \beta$.

Każdy symbol w $\alpha X \beta$ jest osiągalny i jest w $N_2 \cup \Sigma_2$, a więc $\alpha X \beta \to^*_{G_2} xwy$. Wszystkie symbole w tym wywodzie są osiągalne, a więc również w G_1 zachodzi $S \to^*_{G_1} \alpha X \beta \to^*_{G_1} xwy$, czyli X jest użyteczny. Zatem G_1 nie ma bezużytecznych symboli.

Oczywiście mamy $L(G_1) \subseteq L(G)$, bo tylko usuwaliśmy symbole i produkcje. $L(G) \subseteq L(G_1)$ wynika z tego, że jeśli $S \to^*_G w$, to każdy symbol w tym wywodzie jest użyteczny, a więc $S \to^*_{G_1} w$. □

Przykład. Usuwania symboli bezużytecznych nie można wykonać w odwrotnej kolejności. Dla gramatyki z produkcjami $S \to AB|a$, $A \to b$ wszystkie symbole są osiągalne, a po usunięciu symboli niegenerujących dostajemy $S \to a$, $A \to b$. Widzimy, że A, b są bezużyteczne.

Algorytm 2. Dla zadanej gramatyki $G = (N, \Sigma, P, S)$ można wyznaczyć symbole generujące dynamicznie: bazowo uznajemy za generujące symbole z Σ, potem przechodzimy po produkcjach i uznajemy za generujące te nieterminale, dla których istnieje produkcja tworząca formę zdaniową składającą się tylko z symboli generujących (w szczególności pustą). Powtarzamy takie przejście dopóki coś się zmienia.

Symbole osiągalne wyznaczamy algorytmem podobnym do przejścia grafu: zaczynamy odwiedzając S, odwiedzamy wszystkie symbole, które da się wyprodukować z odwiedzonych symboli, powtarzamy, dopóki kolejne iteracje odwiedzają nowe symbole.

Pierwszy z tych algorytmów działa w czasie $|\Sigma| + |P|\,|N|$, a drugi w $|P|\,|N \cup \Sigma|$.

Definicja 44. Nieterminal A jest wymazujący (nullable), jeśli $A \to^*_G \varepsilon$. Symbol $a \in \Sigma$ zawsze jest niewymazujący.

Twierdzenie 16. Niech $G = (N, \Sigma, P, S)$ będzie CFG. Niech $G' = (N', \Sigma, P', S)$ będzie gramatyką otrzymaną z G w następujący sposób. Dla każdej produkcji $A \to X_1 \dots X_k$ w P do P' należą wszystkie takie produkcje $A \to Y_1 \dots Y_l$, że $Y_1 \dots Y_l$ jest pewnym niepustym podciągiem $X_1 \dots X_k$, który zawiera wszystkie symbole niewymazujące i pewne wymazujące.

Gramatyka G' nie zawiera symboli wymazujących i jest $L(G') = L(G) \setminus \{\varepsilon\}$.

Dowód. W P' nie ma produkcji postaci $A \to \varepsilon$, a więc nie ma symboli wymazujących.

Zauważmy, że gramatyka G' „modeluje" symbole wymazujące poprzez dodanie wszystkich produkcji powstałych poprzez usunięcie dowolnego ich podzbioru z pewnej produkcji w P. Zatem jeśli $S \to^*_G w$, to dostajemy $S \to^*_{G'} w$ przechodząc tymi produkcjami, dla których odpowiednie symbole wymazujące znikają. Podobnie w drugą stronę – wystarczy wybierać produkcje, z których powstały te użyte. Formalny dowód przebiega indukcją po długości wywodu. □

Algorytm 3. Nieterminale wymazujące łatwo wyznaczyć algorytmem dynamicznym: bazowo generujące są takie A, że $A \to \varepsilon$ jest produkcją, potem przechodzimy po produkcjach i uznajemy za wymazujące te nieterminale, dla których istnieje produkcja tworząca formę zdaniową składającą się tylko z nieterminali wymazujących. Powtarzamy takie przejście dopóki coś się zmienia. Taki algorytm działa w czasie $|P|\,|N|$.

Definicja 45. Produkcję nazywamy jednostkową, jeśli jest postaci $A \to B$ dla $A, B \in N$.

Twierdzenie 17. Niech $G = (N, \Sigma, P, S)$ będzie CFG. Niech $G' = (N', \Sigma, P', S)$ będzie gramatyką otrzymaną z G w następujący sposób. Rozważmy zbiór $J = \{(A, B) \in N^2 : A \to B \in P\}$ i domknijmy go zwrotnie i przechodnio do J^*. Następnie dla każdego $(A, B) \in J^*$ umieszczamy w P' wszystkie produkcje $A \to \alpha$, gdzie $B \to \alpha$ jest niejednostkową produkcją w P.

Gramatyka G' nie ma produkcji jednostkowych i jest $L(G') = L(G)$.

Dowód. Umieszczamy w P' tylko produkcje niejednostkowe, a więc nie ma żadnych jednostkowych. Języki się zgadzają, bo wywody w G' są tylko skróconymi wywodami z G – wykonujemy produkcje jednostkowe „w miejscu" poprzez dodanie ich ciągów jako osobnych produkcji. □

Wniosek. Dla każdej CFG G generującej coś więcej niż ε istnieje CFG G_1 taka, że $L(G_1) = L(G) \setminus \{\varepsilon\}$ oraz G_1 nie zawiera produkcji jednostkowych, symboli wymazujących oraz symboli bezużytecznych.

Dowód. Korzystając z przedstawionych wcześniej metod możemy usunąć symbole wymazujące, następnie usunąć produkcje jednostkowe (co nie wprowadza nowych symboli wymazujących), a następnie usunąć symbole bezużyteczne (co tylko usuwa produkcje i symbole, więc nie wprowadzi rzeczy już usuniętych). □

Definicja 46. Gramatyka $G = (N, \Sigma, P, S)$ jest w postaci normalnej Chomsky'ego (CNF), jeśli każda produkcja jest postaci $A \to BC$ dla $A, B, C \in N$ lub $A \to a$ dla $a \in \Sigma$. Do tego wszystkie symbole są użyteczne.

Twierdzenie 18. Dla każdej CFG G istnieje CFG G' w postaci normalnej Chomsky'ego taka, że $L(G) \setminus \{\varepsilon\} = L(G')$.

Dowód. Jeśli G generuje $\emptyset$ lub $\{\varepsilon\}$ to twierdzenie jest oczywiste. Dalej zakładamy, że G nie zawiera produkcji jednostkowych, symboli wymazujących oraz symboli bezużytecznych i nie generuje ε, czyli każda produkcja jest postaci $A \to \alpha_1 \dots \alpha_k$ dla $k \geq 2$ i $\alpha_i \in \Sigma \cup N$ lub $A \to \alpha$ dla $\alpha \in \Sigma$.

Jeśli $\alpha_i \in \Sigma$, to wprowadzamy nieterminal B_i z produkcją $B_i \to \alpha_i$, a następnie zamieniamy wszystkie wystąpienia α_i na B_i. Teraz problemem są tylko produkcje postaci $A \to B_1 \dots B_k$ dla $k \geq 3$. Możemy wprowadzić nowe nieterminale $C_1, \dots, C_{k-2}$ oraz produkcje $C_i \to B_{i+1}C_{i+1}$ dla $i \in [k-3]$ oraz $C_{k-2} \to B_{k-1}B_k$. Teraz wystarczy zastąpić problematyczną produkcję przez $A \to B_1C_1$. □

Algorytm 4 (Cocke, Younger, Kasami (CYK)). Mając zadaną gramatykę $G = (N, \Sigma, P, S)$ w CNF i słowo $w \in \Sigma^*$ chcemy stwierdzić, czy $w \in L(G)$.

Robimy to dynamicznie: niech $dp[i, j]$ oznacza listę wszystkich nieterminali, które generują podsłowo w długości i o początku w j. Bazą jest $i = 1$, do list wsadzamy nieterminale generujące daną literkę (łatwo znaleźć z postaci Chomsky'ego). Następnie iterujemy się po rosnących i, obliczając kolejne listy – dzielimy interesujące nas podsłowo na dwa podsłowa (na wszystkie sposoby), znając nieterminale generujące pierwsze i drugie podsłowo znajdujemy te nieterminale, które generują takie dwa nieterminale w odpowiedniej kolejności (znów postać Chomsky'ego).

Na koniec sprawdzamy, czy całe słowo jest generowane przez nieterminal startowy. Taki algorytm działa w czasie $O\left(|P|\,|v|^3\right)$.

8. Automaty ze stosem

2025-03-17

Definicja 47. Automat ze stosem (PDA – pushdown automaton) to krotka

$$P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F),$$

gdzie:

- Q – skończony zbiór stanów,
- Σ – skończony alfabet,
- Γ – skończony alfabet stosowy,
- $\delta : (Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma) \to \mathcal{P}(Q \times \Gamma^*)$ – skończona niedeterministyczna funkcja przejścia,
- $q_0 \in Q$ – stan startowy,

- $Z_0 \in \Gamma$ – stosowy symbol początkowy (sygnalizujący pusty stos),
- $F \subseteq Q$ – zbiór stanów końcowych.

Taki automat działa podobnie jak ε-NFA, ale tym razem ma stos, na który może odkładać różne wartości, które potem są z niego ściągane i uczestniczą w podejmowaniu decyzji.

Przykład. Zdefiniujemy PDA generujące język parzystych palindromów $L = \left\{ww^R : w \in \{0,1\}^*\right\}$. Ustalamy $P = (\{q_0, q_1, q_2\}, \{0,1\}, \{0,1,Z_0\}, \delta, q_0, Z_0, \{q_2\})$, gdzie δ zawiera przejścia:

- $\delta(q_0, 0, Z_0) = \{(q_0, 0Z_0)\}$, $\delta(q_0, 1, Z_0) = \{(q_0, 1Z_0)\}$ – przejścia początkowe
- $\delta(q_0, 0, 0) = \{(q_0, 00)\}$, $\delta(q_0, 1, 0) = \{(q_0, 10)\}$, $\delta(q_0, 0, 1) = \{(q_0, 01)\}$, $\delta(q_0, 1, 1) = \{(q_0, 11)\}$ – dokładanie liter na stos
- $\delta(q_0, \varepsilon, Z_0) = \{(q_1, Z_0)\}$, $\delta(q_0, \varepsilon, 0) = \{(q_1, 0)\}$, $\delta(q_0, \varepsilon, 1) = \{(q_1, 1)\}$ – przejście do drugiej części słowa
- $\delta(q_1, 0, 0) = \{(q_1, \varepsilon)\}$, $\delta(q_1, 1, 1) = \{(q_1, \varepsilon)\}$ – ściąganie liter, jeśli się zgadzają
- $\delta(q_1, \varepsilon, Z_0) = \{(q_2, Z_0)\}$ – akceptacja słowa.

Taki automat można też prezentować graficznie podobnie jak automaty skończone, ale trzeba dopisywać przy strzałkach jak wyglądają zmiany stosu.

Definicja 48. Opis chwilowy (konfiguracja) PDA to trójka (q, w, γ), gdzie q to aktualny stan, w to nieprzetworzona jeszcze część słowa a γ to wszystkie symbole na stosie (czytane od góry).

Definicja 49. Dla PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ definiujemy $\vdash_P$ jako relację przejścia między konfiguracjami, to znaczy

$$(q, aw, X\beta) \vdash_P (p, w, \alpha\beta) \iff (p, \alpha) \in \delta(q, a, X),$$

gdzie $a \in \Sigma \cup \{\varepsilon\}$, $X \in \Gamma$ a pozostałe symbole mają typy wynikające z definicji.

Definiujemy $\vdash_P^*$ jako zwrotne i przechodnie domknięcie $\vdash_P$.

Propozycja 7. Dla PDA P jeśli $(q, x, \alpha) \vdash_P^* (p, y, \beta)$, to również $(q, xw, \alpha\gamma) \vdash_P^* (p, yw, \beta\gamma)$.

Dowód. Założenie oznacza, że konsumując znaki z x i α możemy dojść do odpowiedniej konfiguracji. Mając dołożone na ich koniec jakieś znaki w i γ możemy zastosować te same przejścia, by otrzymać odpowiednią konfigurację. Te przejścia oczywiście nie dotkną dołożonych znaków. □

Propozycja 8. Dla PDA P jeśli $(q, xw, \alpha) \vdash_P^* (p, yw, \beta)$, to również $(q, x, \alpha) \vdash_P^* (p, y, \beta)$.

Dowód. Jeśli w odpowiednich przejściach nie zaczęliśmy konsumować w, to równie dobrze można je usunąć z naszego słowa. □

Definicja 50. Niech $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ będzie PDA. Językiem akceptowanym stanem końcowym tego PDA nazywamy język $L(P) = \{w \in \Sigma^* : (q_0, w, Z_0) \vdash_P^* (q, \varepsilon, \alpha) \wedge q \in F\}$.

Definicja 51. Niech $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ będzie PDA. Językiem akceptowanym pustym stosem tego PDA nazywamy język $N(P) = \{w \in \Sigma^* : (q_0, w, Z_0) \vdash_P^* (q, \varepsilon, \varepsilon)\}$.

Twierdzenie 19. Mamy $L = L(P)$ dla pewnego PDA P wtedy i tylko wtedy, gdy $L = N(Q)$ dla pewnego PDA Q.

Dowód. ($\Longrightarrow$) Dla $P = (Q, \Sigma, \Gamma, \delta, p_0, Z_0, F)$ niech $Q = (Q \cup \{q_0, q_1\}, \Sigma, \Gamma \cup \{X_0\}, \delta_Q, q_0, X_0, F)$. W δ_Q mamy takie same przejścia jak w δ, a do tego $\delta_Q(q_0, \varepsilon, X_0) = \{(p_0, Z_0X_0)\}$, $\delta_Q(q_1, \varepsilon, A) = \{(q_1, \varepsilon)\}$ dla każdego $A \in \Gamma \cup \{X_0\}$ oraz $(q_1, \varepsilon) \in \delta_Q(p_F, \varepsilon, A)$ dla każdego $A \in \Gamma$ i $p_F \in F$. Ten automat po przejściu do stanu akceptującego opróżni stos i jednocześnie dzięki symbolowi X_0 nie opróżni go w żaden inny sposób.

($\Longleftarrow$) Dla $Q = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ definiujemy $P = (Q \cup \{p_0, p_1\}, \Sigma, \Gamma \cup \{X_0\}, \delta_P, p_0, X_0, \{p_1\})$. W δ_P mamy takie same przejścia jak w δ, a do tego $\delta_P(p_0, \varepsilon, X_0) = \{(q_0, Z_0X_0)\}$ i $(p_1, \varepsilon) \in \delta_P(q, \varepsilon, X_0)$ dla każdego $q \in Q$. W ten sposób nasz automat włoży Z_0 na stos, zachowa się dokładnie jak Q (czyli dokładnie słowa akceptowane przez Q skończą ze stosem zawierającym samo X_0), a następnie przejdzie do stanu akceptującego. □

Twierdzenie 20. Język L jest bezkontekstowy wtedy i tylko wtedy, gdy jest akceptowany przez pewien automat ze stosem.

Dowód. ($\Longrightarrow$) Język L jest generowany przez pewną gramatykę G'. Modyfikujemy ją do gramatyki $G = (N, \Sigma, P, S)$, w której każda produkcja jest postaci $A \to a$ dla $a \in \Sigma \cup \{\varepsilon\}$ lub $A \to B_1 \dots B_n$ dla $A, B_1, \dots, B_n \in N$. Robimy to dodając produkcję $A \to a$ dla każdego terminala a i zastępując go w każdej produkcji z G' przez odpowiedni nieterminal. Widać, że G akceptuje L. Konstruujemy PDA $P = (\{q\}, \Sigma, N, \delta, q, S, \{q\})$, które będzie akceptować stosem pustym. Dla produkcji $A \to a$ dodajemy $(q, \varepsilon) \in \delta(q, a, A)$, a dla produkcji $A \to B_1 \dots B_n$ dodajemy $(q, B_1 \dots B_n) \in \delta(q, \varepsilon, A)$. W ten sposób nasz stos określa kolejne nieterminale, jakie mamy do dyspozycji.

Pokażemy, że $A \to_G^* w$ wtedy i tylko wtedy, gdy $(q, w, A) \vdash_P^* (q, \varepsilon, \varepsilon)$. Jeśli $A \to_G w$, to teza zachodzi z konstrukcji. Dla dłuższego wywodu mamy $A \to_G B_1 \dots B_n \to_G^* w_1 \dots w_n$ dla $w = w_1 \dots w_n$. Z założenia indukcyjnego mamy $(q, w_i, B_i) \vdash_P^* (q, \varepsilon, \varepsilon)$, a więc $(q, w, B_1 \dots B_n) \vdash_P^* (q, \varepsilon, \varepsilon)$. Wraz z $(q, w, A) \vdash_P (q, w, B_1 \dots B_n)$ daje to tezę. W drugą stronę indukujemy się po ilości przejść automatu. Jeśli $(q, w, A) \vdash_P (q, \varepsilon, \varepsilon)$, to mamy $A \to w$. Dla większej ilości przejść jest $(q, w, A) \vdash_P (q, w, B_1 \dots B_n) \vdash_P^* (q, \varepsilon, \varepsilon)$. Istnieje podział $w = w_1 \dots w_n$ taki, że $(q, w_i, B_i) \vdash_P^* (q, \varepsilon, \varepsilon)$. Zatem z założenia indukcyjnego $B_1 \dots B_n \to_G^* w_1 \dots w_n$, co kończy dowód.

($\Longleftarrow$) Niech $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ będzie PDA akceptującym L stosem pustym. Skonstruujemy odpowiednią gramatykę G. Będą do niej należeć stan startowy S oraz stany postaci N_{pXq} dla każdych $p, q \in Q$ oraz $X \in \Gamma$. Do tego dla każdego stanu $p \in Q$ mamy produkcję $S \to N_{q_0 Z_0 p}$ oraz jeśli $(r, Y_1 \dots Y_k) \in \delta(q, a, X)$, to dla każdej k-elementowej listy stanów $r_1 \dots r_k$ tworzymy produkcję $N_{qXr_k} \to aN_{rY_1r_1}N_{r_1Y_2r_2} \dots N_{r_{k-1}Y_kr_k}$. Jeśli $(r, \varepsilon) \in \delta(q, a, X)$, to $N_{qXr} \to a$.

Pokażemy, że $N_{qXp} \to_G^* w$ wtedy i tylko wtedy, gdy $(q, w, X) \vdash_P^* (p, \varepsilon, \varepsilon)$. To w szczególności da nam tezę. Jeśli $(q, w, X) \vdash_P (p, \varepsilon, \varepsilon)$ (w jednym kroku), to mamy $N_{qXp} \to_G w$. Jeśli liczba kroków automatu jest większa niż jeden, to mamy $(q, w, X) \vdash_P (r_0, v, Y_1 \dots Y_k) \vdash_P^* (p, \varepsilon, \varepsilon)$ dla $w = av$ i istnieje podział $v = v_1 \dots v_k$ i stany $r_1, \dots, r_k$ takie, że $r_k = p$, $(r_{i-1}, v_i \dots v_k, Y_i \dots Y_k) \vdash_P^* (r_i, v_{i+1} \dots v_k, Y_{i+1} \dots Y_k)$, czyli $(r_{i-1}, v_i, Y_i) \vdash_P^* (r_i, \varepsilon, \varepsilon)$. Zatem w G mamy wywód $N_{qXp} \to_G aN_{r_0Y_1r_1} \dots N_{r_{k-1}Y_kp} \to_G^* av_1 \dots v_k$.

Podobnie jeśli mamy $N_{qXp} \to_G w$, to mamy $(p, \varepsilon) \in \delta(q, w, X)$. Dla dłuższego wywodu mamy $N_{qXp} \to_G aN_{r_0Y_1r_1} \dots N_{r_{k-1}Y_kr_k} \to_G^* w$ dla $r_k = p$. Możemy zapisać $w = av_1 \dots v_k$ dla $N_{r_{i-1}Y_ir_i} \to_G^* v_i$. Z założenia indukcyjnego $(r_{i-1}, v_i, Y_i) \vdash_P^* (r_i, \varepsilon, \varepsilon)$, a więc $(r_0, v_1 \dots v_k, Y_1 \dots Y_k) \vdash_P^* (r_k, \varepsilon, \varepsilon)$. Jednocześnie mamy $(q, w, X) \vdash_P (r_0, v_1 \dots v_k, Y_1 \dots Y_k)$, co kończy dowód. □

9. Języki bezkontekstowe

2025-03-23

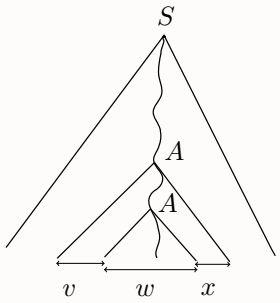
Propozycja 9. Niech G będzie gramatyką bezkontekstową w postaci normalnej Chomsky'ego. Jeśli w drzewie wywodu słowa w każda ścieżka ma długość co najwyżej n, to zachodzi $|w| \leq 2^{n-1}$.

Dowód. Dowód indukcją po długości najdłuższej ścieżki w drzewie. Jeśli ma ona długość 1, to drzewo składa się z korzenia i jego jednego dziecka etykietowanego terminalem, mamy $|w| = 1$. Jeśli jest dłuższa, to korzeń ma dokładnie dwójkę dzieci (bo G jest w CNF). Oba poddrzewa ukorzenione w tych dzieciach mają długość najdłuższej ścieżki co najwyżej $n-1$, a więc z założenia indukcyjnego mamy $|w| \leq 2^{n-2} + 2^{n-2} = 2^{n-1}$. □

Lemat 4 (O pompowaniu). Niech L będzie językiem bezkontekstowym. Wtedy istnieje takie $n \in \mathbb{N}$, że dla każdego $z \in L$ z $|z| \geq n$ istnieje postać $z = uvwxy$ taka, że $|vwx| \leq n$, $|vx| \geq 1$ i $\forall_{i \in \mathbb{N}}\ uv^iwx^iy \in L$.

Dowód. Załóżmy, że gramatyka $G = (V, \Sigma, P, S)$ języka L jest w CNF (dla języków $\emptyset, \{\varepsilon\}$ teza oczywiście zachodzi, a pozostałe języki mają taką gramatykę). Wybieramy $n = 2^m$, gdzie $m = |V|$.

Wtedy w drzewie wywodu słowa z istnieje ścieżka długości co najmniej $m+1$, a więc pewien nieterminal występuje na niej co najmniej dwa razy. Niech A będzie pierwszym powtarzającym się nieterminalem (patrząc od dołu ścieżki).



Rysunek 3: Drzewo rozważanego słowa.

Niech w będzie częścią słowa generowaną przez dolne wystąpienie A. Niech vwx będzie częścią słowa generowaną przez górne wystąpienie A, a $uvwxy$ całym słowem. Mamy $|vwx| \leq n$, bo inaczej A nie byłby pierwszym powtarzającym się nieterminalem (ścieżka byłaby za długa). Do tego $vx \neq \varepsilon$ – górne wystąpienie A ma dwójkę dzieci, z których co najmniej jedno nie jest dolnym A i generuje jakieś litery. Zauważmy, że poddrzewo dolnego A można zastąpić poddrzewem górnego A i można to zrobić dowolną liczbę razy (lub zastąpić poddrzewo górnego A poddrzewem dolnego A), a więc $uv^iwx^iy \in L$ dla każdego i, bo ma odpowiednie drzewo wywodu. □

Twierdzenie 21. Język

$$L = \{a^n b^n c^n : n \in \mathbb{N}\}$$

nie jest bezkontekstowy.

Dowód. Niech n będzie stałą z lematu o pompowaniu. Rozważmy słowo $a^n b^n c^n$ i załóżmy nie wprost, że ma ono postać $uvwxy$ jak w lemacie o pompowaniu. Jeśli $vwx = a^p$, to do języka będzie należeć słowo $uwy = a^{n-|vx|}b^n c^n$ – sprzeczność. Podobnie, gdy vwx składa się w całości z dwóch pozostałych liter. Jeśli $vwx = a^p b^q$, to do języka będzie należeć słowo $a^{n-l}b^{n-k}c^n$ dla pewnych l, k. Podobnie dla kombinacji $b^p c^q$. Ograniczenie na wielkość vwx powoduje, że są to wszystkie możliwości. □

Lemat 5 (Ogdena). Niech L będzie językiem bezkontekstowym. Wtedy istnieje takie $n \in \mathbb{N}$, że dla każdego $z \in L$ jeśli wyróżnimy w z co najmniej n pozycji, to istnieje postać $z = uvwxy$ taka, że vwx zawiera co najwyżej n wyróżnionych pozycji, vx zawiera co najmniej jedną i $\forall_{i\in\mathbb{N}}\ uv^iwx^iy \in L$.

Dowód. Załóżmy, że gramatyka $G = (V, \Sigma, P, S)$ języka L jest w CNF (dla języków $\emptyset, \{\varepsilon\}$ teza oczywiście zachodzi, a pozostałe języki mają taką gramatykę). Wybieramy $n = 2^{m+1}$, gdzie $m = |V|$. Rozważmy pewne drzewo wywodu słowa z. Węzeł drzewa będziemy nazywać rozwidleniem, jeśli oba jego dzieci generują niezerową liczbę wyróżnionych pozycji. Skonstruujemy ścieżkę od korzenia do liścia w drzewie z co najmniej $m+1$ rozwidleniami. Robimy to w następujący sposób: zaczynamy z korzenia i jeśli aktualny wierzchołek nie jest rozwidleniem, to idziemy do jedynego dziecka generującego wyróżnione pozycje, a w przeciwnym wypadku idziemy do tego dziecka, które generuje ich więcej. W pierwszym wypadku liczba wyróżnionych pozycji w poddrzewie aktualnego wierzchołka nie zmienia się, a w drugim maleje co najwyżej dwukrotnie. Zatem na drodze do liścia musimy napotkać co najmniej $m+1$ rozwidleń.

Dalej postępujemy jak w dowodzie lematu o pompowaniu: pewien nieterminal występuje wśród rozwidleń na naszej ścieżce co najmniej dwa razy. Niech A będzie pierwszym powtarzającym się nieterminalem (patrząc od dołu ścieżki). Niech w będzie częścią słowa generowaną przez dolne wystąpienie A. Niech vwx będzie częścią słowa generowaną przez górne wystąpienie A, a $uvwxy$ całym słowem. vwx ma co najwyżej n wyróżnionych pozycji, bo inaczej A nie byłby pierwszym powtarzającym się nieterminalem (ścieżka byłaby za długa). Do tego vx zawiera wyróżnioną pozycję – górne wystąpienie A ma dwójkę dzieci generujących wyróżnione pozycje, z których co najmniej jedno nie jest dolnym A. Zauważmy, że poddrzewo dolnego A można zastąpić poddrzewem górnego

A i można to zrobić dowolną liczbę razy (lub zastąpić poddrzewo górnego A poddrzewem dolnego A), a więc $uv^iwx^iy \in L$ dla każdego i, bo ma odpowiednie drzewo wywodu. □

Propozycja 10. Języki bezkontekstowe są zamknięte na sumę, konkatenację i gwiazdkę Kleene'ego, ale nie na przecięcie i dopełnienie.

Dowód. Mając dwa języki bezkontekstowe L_1, L_2 o gramatykach G_1, G_2 tworzymy nową gramatykę G. Niech S, S_1, S_2 będą symbolami startowymi odpowiednio G, G_1, G_2. W G mamy wszystkie produkcje z G_1, G_2 (zakładamy, że mają rozłączne zbiory nieterminali), a do tego:

- jeśli chcemy $L(G) = L_1 \cup L_2$, to ustalamy $S \to S_1 \mid S_2$.
- jeśli chcemy $L(G) = L_1 L_2$, to ustalamy $S \to S_1 S_2$.

Jeśli chcemy $L(G) = L_1^*$, to możemy ustalić $S \to SS_1 \mid \varepsilon$.

Język $L_1 = \{a^n b^n c^k : n, k \in \mathbb{N}\}$ jest bezkontekstowy jako konkatenacja języków $\{a^n b^n : n \in \mathbb{N}\}$ i $\{c^k : k \in \mathbb{N}\}$. Podobnie język $L_2 = \{a^k b^n c^n : n, k \in \mathbb{N}\}$ jest bezkontekstowy. Mamy $L_1 \cap L_2 = \{a^n b^n c^n : n \in \mathbb{N}\}$, co nie jest bezkontekstowe. Zatem języki bezkontekstowe nie są zamknięte na przecięcie. Z zamknięcia na dopełnienie i równości $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$ wynikałoby zamknięcie na przecięcie. Zatem języki bezkontekstowe nie są zamknięte na dopełnienie. □

Twierdzenie 22. Dla gramatyk bezkontekstowych problem MEMBERSHIP jest wielomianowy.

Dowód. Wystarczy zastosować algorytm CYK. □

Twierdzenie 23. Dla gramatyk bezkontekstowych problem EMPTINESS jest wielomianowy.

Dowód. Umiemy wyznaczyć symbole niegenerujące w gramatyce. Wystarczy sprawdzić, czy symbol startowy jest generujący. □

Twierdzenie 24. Dla gramatyk bezkontekstowych problem INFINITENESS jest wielomianowy.

Dowód. Przekształcamy gramatykę do CNF. Następnie szukamy cyklu w grafie produkcji (grafie skierowanym na nieterminalach, w którym mamy krawędź, jeśli można wyprodukować wyrażenie zawierające dany nieterminal z innego nieterminala). Jeśli taki cykl istnieje, to można przechodzić po nim dowolnie dużo razy, generując za każdym razem nowe słowa (brak produkcji jednostkowych i wymazujących gwarantuje powstawanie nowych liter). Jeśli nie ma takiego cyklu, to maksymalna długość wywodu jest ograniczona, zatem długość wyprodukowanego słowa również jest ograniczona. To implikuje skończoność języka. □

10. Maszyny Turinga

2025-04-30

Definicja 52. Maszyna Turinga (Turing Machine) to krotka

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F),$$

gdzie:

- Q – skończony zbiór stanów,
- Σ – skończony alfabet,
- $\Gamma \supseteq \Sigma$ – skończony alfabet taśmowy,
- $\delta : (Q \times \Gamma) \to (Q \times \Gamma \times \{-1, 1\})$ - częściowa funkcja przejścia,
- $q_0 \in Q$ – stan startowy,
- $B \in \Gamma \setminus \Sigma$ – blank,
- $F \subseteq Q$ – zbiór stanów akceptujących.

Maszyna Turinga składa się z taśmy o nieskończenie wielu polach rozciągającej się w nieskończoność w obie strony. Początkowo na taśmie zapisane jest słowo początkowe (w pozostałych polach są blanki B), a głowica maszyny patrzy na pierwszą literę słowa. Głowica będzie się przesuwać w prawo lub lewo i nadpisywać symbole na taśmie oraz zmieniać stany.

Pomysł. W 1933 Gödel i Herbrand wprowadzili funkcje μ-rekurencyjne. W 1936 Church wprowadził λ rachunek. Turing z kolei wprowadził maszyny Turinga. Te modele obliczeń są sobie nawzajem równoważne, co sugeruje, że nie istnieje mocniejszy od nich model obliczeń.

Twierdzenie 25 (Teza Churcha 1936)**.** Funkcja (częściowa) $\mathbb{N} \to \mathbb{N}$ jest obliczalna przez człowieka wykonującego pewien algorytm przy użyciu nieograniczonych zasobów wtedy i tylko wtedy, gdy jest obliczalna przez maszynę Turinga.

Twierdzenie 26 (Church, Turing)**.** Funkcja jest obliczalna przez maszynę Turinga wtedy i tylko wtedy, gdy jest μ-obliczalna wtedy i tylko wtedy, gdy jest λ-obliczalna.

Definicja 53. Konfiguracja (opis chwilowy) maszyny Turinga to ciąg postaci $X_1 \dots X_{i-1} q X_i \dots X_n$, gdzie $X_k \in \Gamma$ i $q \in Q$. Odpowiada on sytuacji, w której maszyna jest w stanie q i odwiedziła kiedyś (być może zmieniła) pola od X_1 do X_n (pozostałe to blanki), a aktualnie znajduje się na polu X_i.

Definicja 54. Dla Maszyny Turinga $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ definiujemy $\vdash_M$ jako relację przejścia między konfiguracjami, w której mamy:

- środkowy ruch w lewo $X_1 \dots X_{i-1} q X_i \dots X_n \vdash_M X_1 \dots X_{i-2} p X_{i-1} Y \dots X_n$, jeśli $\delta(q, X_i) = (p, Y, -1)$ dla $i > 1$.
- lewy ruch w lewo $q X_1 \dots X_n \vdash_M pBY \dots X_n$, jeśli $\delta(q, X_1) = (p, Y, -1)$.
- środkowy ruch w prawo $X_1 \dots X_{i-1} q X_i \dots X_n \vdash_M X_1 \dots X_{i-1} Y p X_{i+1} \dots X_n$, jeśli $\delta(q, X_i) = (p, Y, 1)$ dla $i < n$.
- prawy ruch w prawo $X_1 \dots q X_n \vdash_M X_1 \dots X_{n-1} Y p B$, jeśli $\delta(q, X_n) = (p, Y, 1)$.

Przez $\vdash_M^*$ oznaczamy zwrotne i przechodnie domknięcie $\vdash_M$.

Definicja 55. Konfiguracja początkowa maszyny Turinga M to $q_0 w$. Język akceptowany przez nią to

$$L(M) = \{w \in \Sigma^* : q_0 w \vdash_M^* \alpha q_F \beta \wedge \alpha, \beta \in \Gamma^*, q_F \in F\}.$$

Definicja 56. Mówimy, że język jest rekurencyjnie przeliczalny, jeśli jest akceptowany przez pewną maszynę Turinga. Zbiór takich języków oznaczamy RE.

Definicja 57. Maszyna Turinga M ma własność stopu, jeśli zatrzymuje się na każdym wejściu w. Zatrzymanie się polega na tym, że znajdziemy się w konfiguracji, dla której funkcja przejścia δ nie jest zdefiniowana. Możemy założyć, że maszyna Turinga z własnością stopu ma taki stan akceptujący q_{acc} i nieakceptujący q_{rej}, że dla żadnego symbolu na taśmie δ nie jest dla nich zdefiniowana i są to jedyne takie stany. Do tego dla każdej Maszyny Turinga można założyć, że ma jeden stan akceptujący, na którym się zatrzymuje.

Definicja 58. Mówimy, że $L \subseteq \Sigma^*$ jest rekurencyjny, jeśli istnieje maszyna Turinga M z własnością stopu taka, że $L = L(M)$. Takie języki nazywamy też rozstrzygalnymi, a ich zbiór oznaczamy R.

Definicja 59. k-taśmowa maszyna Turinga to maszyna Turinga, w której funkcja przejścia ma sygnaturę $\delta : \left(Q \times \Gamma^k\right) \to \left(Q \times \Gamma^k \times \{-1, 1\}^k\right)$. Oznacza to, że mamy k taśm, na każdej piszemy i chodzimy niezależnie.

Twierdzenie 27. Dla każdego $k \geq 1$ język L jest rekurencyjnie przeliczalny wtedy i tylko wtedy, gdy $L = L(M_k)$ dla pewnej k-taśmowej maszyny Turinga $M_k = (Q, \Sigma, \Gamma, \delta_k, q_0, B, F)$.

Dowód. ($\Longrightarrow$) Oczywiste, można nie używać innych taśm niż pierwsza.

($\Longleftarrow$) Zdefiniujemy maszynę Turinga M, która będzie symulować M_k. Jej symbolami taśmowymi będą $2k$-krotki, w których na parzystych indeksach są symbole taśmowe z M_k, a na nieparzystych liczby 0 lub 1. Parzyste indeksy będą oznaczały, co znajduje się na danej taśmie w maszynie k-taśmowej, a indeksy nieparzyste, czy M_k aktualnie patrzy na to pole. Blankiem jest krotka postaci $(B, 0 \ldots, B, 0)$. Na początku w taśmie będzie słowo (niebędące krotką). M zaczyna działanie od zamienienia liter na odpowiednie krotki, to znaczy dla słowa $w_1 w_2 \ldots w_n$ zamienia kolejne litery na $(w_1, 1, B, 1, \ldots, B, 1)$, $(w_2, 0, B, 0, \ldots, B, 0)$ i tak aż do $(w_n, 0, \ldots, B, 0)$. Można to zrobić definiując stan, który zamieni w_1 na odpowiednią krotkę i przejdzie w prawo zmieniając się w inny stan. On będzie zamieniał litery w_i na odpowiednie krotki i pozostawał w tym samym stanie. Po dojściu do blanka zostanie zmieniony stan. On będzie szedł w lewo aż do napotkania blanka. Wtedy pójdzie w prawo i przejdzie do stanu, który zacznie wykonanie dalszej części programu.

M będzie symulować jeden krok M_k za pomocą sekwencji kroków. Sekwencja zaczyna się w stanie s_q, gdzie $q \in Q$ jest stanem, w którym M_k jest przed symulowanym krokiem. M przesuwa się w prawo i jeśli widzi, że któraś taśma M_k patrzy na dany symbol, to odnotowuje to, zmieniając swój stan. Ostatecznie będziemy na prawo od wszystkich oglądanych przez M_k symboli i będziemy w stanie $s_{q,a_1,\ldots,a_k}$, gdzie $a_1, \ldots, a_k$ to oglądane symbole. Wtedy wracamy w lewo i przechodzimy do stanu $s_{p,b_1,\ldots,b_k,d_1,\ldots,d_k}$ takiego, że $\delta_k(q, a_1, \ldots, a_k) = (p, b_1, \ldots, b_k, d_1, \ldots, d_k)$. Będziemy szli w w lewo i jeśli napotkamy oglądane symbole, to zastępujemy je zgodnie z funkcją przejścia (a_i na b_i), a następnie zmieniamy to, które pole jest oglądane zgodnie z wartością d_i (jeśli $d_i = 1$, to cofamy się w prawo). To, które taśmy zostały nam do zmienienia i czy powinniśmy uznać któryś symbol w aktualnej krotce za oglądany trzymamy w stanie, w którym jesteśmy. Ostatecznie znajdziemy się na lewo od wszystkich oglądanych przez M_k pól i będziemy w stanie s_p, z którego zaczynamy nową iterację.

Uznajemy za akceptujące takie stany s_q, że $q \in F$. Wtedy nasza maszyna znajdzie się w stanie akceptującym dokładnie wtedy, gdy znajdzie się w nim M_k. □

Definicja 60. Niedeterministyczna maszyna Turinga to maszyna Turinga, w której funkcja przejścia została zastąpiona relacją przejścia.

Twierdzenie 28. Język L jest rekurencyjnie przeliczalny wtedy i tylko wtedy, gdy $L = L(M_N)$ dla pewnej niedeterministycznej maszyny Turinga M_N.

Dowód. ($\Longrightarrow$) Oczywiste.

($\Longleftarrow$) Skonstruujemy 3-taśmową maszynę Turinga M. Na pierwszej taśmie będzie ona miała słowo wejściowe w, na drugiej będzie trzymała kolejkę konfiguracji M_N, które będzie po kolei rozpatrywać, a trzecia taśma będzie taśmą roboczą. Na początku M przechodzi po słowie w i zapisuje odpowiadającą mu konfigurację (czyli też stan) na drugiej taśmie. Następnie zaczyna się właściwa praca maszyny. Na drugiej taśmie maszyna ma ciąg (kolejkę) konfiguracji M_N. Początkowo jest tam tylko jedna konfiguracja. Konfiguracje są oddzielone specjalnymi znakami. Maszyna patrzy na początek aktualnej konfiguracji. Przechodzi przez nią (od lewej do prawej) i zapisuje ją na trzeciej taśmie. Przy okazji zapisuje w swoim stanie informację o tym, w jakim stanie jest rozważana konfiguracja i na jaki znak patrzy.

W tej chwili M zna skończoną liczbę konfiguracji, do których może przejść M_N. Jeśli istnieje przejście do stanu akceptującego, to M akceptuje i kończy wykonanie. W przeciwnym wypadku rozważa wszystkie (skończenie wiele) przejścia, jakie może zrobić M_N i zapisuje wynikającą z nich konfigurację na końcu kolejki – przechodzi po konfiguracji zapisanej na trzeciej taśmie i przepisuje ją z odpowiednią zmianą. Następnie wraca na taśmie będącej kolejką do właśnie rozważonej konfiguracji (przy okazji czyści ostatnią taśmę) i przechodzi do następnej, oznaczając poprzednią konfigurację jako rozważoną (inny specjalny znak pomiędzy konfiguracjami).

Jeśli M_N akceptuje jakieś słowo, to w związku z istnieniem ciągu skończenie wielu kroków. Zatem M rozważając kolejne kroki w końcu znajdzie się na końcu odpowiedniego ich ciągu. □

11. Rozstrzygalność

2025-05-11

Uwaga. Istnieją języki, które nie są rekurencyjnie przeliczalne. Wynika to z faktu, że języków jest continuum, a Maszyn Turinga przeliczalnie wiele.

Definicja 61. Kodowaniem Maszyny Turinga $M = (Q, \{0,1\}, \Gamma, \delta, q_1, B, F)$ nazywamy słowo, które koduje jego funkcję przejścia δ. Powstaje ono w następujący sposób: ustalamy porządek na stanach $Q = \{q_1, \ldots, q_k\}$, symbolach $\Gamma = \{X_1, \ldots, X_n\}$ oraz oznaczamy -1 i 1 jako D_1 i D_2. Do tego zakładamy, że q_1 jest stanem startowym, q_2 jedynym stanem akceptującym, $X_1 = 0$, $X_2 = 1$ i $X_3 = B$. Kodujemy krotkę $(q_i, X_j, q_k, X_\ell, D_t) \in \delta$ jako $0^i10^j10^k10^\ell10^t$. Całą Maszynę Turinga kodujemy jako $C_1 11 C_2 11 \ldots 11 C_s$, gdzie $C_1, \ldots, C_s$ są kodowaniami wszystkich krotek w δ (w dowolnej kolejności).

Uwaga. Zauważmy, że istnieje wiele słów, które odpowiadają tej samej Maszynie Turinga. Istnieją też słowa, które nie są poprawnym kodowaniem żadnej Maszyny Turinga (np. słowa kończące się na 1). Takim słowom możemy przypisać Maszynę Turinga z jednym stanem, który jest nieakceptujący. Z ten sposób możemy założyć, że każde słowo odpowiada pewnej Maszynie Turinga.

Definicja 62. Na słowach $u, v \in \{0,1\}^*$ definiujemy porządek liniowy

$$u \preccurlyeq v \iff (|u| < |v| \vee (|u| = |v| \wedge u \leqslant_{\text{lex}} v)),$$

w którym najpierw porównujmy słowa po długości, a potem leksykograficznie. Przez $w_1, w_2, \ldots$ rozumiemy listę wszystkich słów wypisanych w tym porządku, a przez $M_1, M_2, \ldots$ listę odpowiadających im Maszyn Turinga.

Definicja 63. Językiem przekątniowym nazywamy język $L_d = \{w_i : w_i \notin L(M_i)\}$.

Lemat 6. Język przekątniowy nie jest rekurencyjnie przeliczalny.

Dowód. Załóżmy, nie wprost, że $L_d = L(M)$ dla pewnej Maszyny Turinga M. Mamy $M = M_i$ dla pewnego i. Zatem $w_i \in L(M) \iff w_i \notin L(M_i)$ – sprzeczność, bo $M = M_i$. □

Propozycja 11. Języki rekurencyjne są zamknięte na branie dopełnienia. To znaczy, że jeśli $L \in \mathrm{R}$, to również $\overline{L} \in \mathrm{R}$.

Dowód. Mając Maszynę Turinga z własnością stopu dla L można zamienić miejscami jej stan akceptujący q_{acc} i odrzucający q_{rej}. □

Wniosek. Dla $\overline{L_d} = \{w_i : w_i \in L(M_i)\}$ zachodzi $\overline{L_d} \notin \mathrm{R}$.

Dowód. Gdyby było inaczej, to mielibyśmy $L_d \in \mathrm{R}$. □

Lemat 7. Jeśli $L, \overline{L} \in \mathrm{RE}$, to zachodzi $L, \overline{L} \in \mathrm{R}$.

Dowód. Jeśli $L = L(M)$ i $\overline{L} = L(\overline{M})$ dla pewnych Maszyn Turinga $M, \overline{M}$, to możemy symulować ich działanie równolegle. Na jednej taśmie trzymamy stan M, a na drugiej $\overline{M}$ i wykonujemy ich kroki na zmianę. W końcu jedna z tych maszyn zaakceptuje i na podstawie tego będziemy wiedzieć, czy odrzucić, czy zaakceptować. Tak skonstruowana Maszyna Turinga zatrzyma się na każdym wejściu, a więc ma własność stopu. □

Uwaga. Z tego wynika, że RE nie jest zamknięte na branie dopełnień, bo, jak pokażemy zaraz, istnieją języki należące do $\mathrm{RE} \setminus \mathrm{R}$.

Definicja 64. Językiem uniwersalnym nazywamy język $L_u = \{\langle M_i, w_j \rangle : w_j \in L(M_i)\}$, gdzie Maszyna Turinga M_i i słowo w_j są zakodowane, np. standardowym kodowaniem Maszyny Turinga, które jest oddzielone od słowa za pomocą 111 – taki ciąg nie występuje w kodowaniu maszyny.

Lemat 8. Zachodzi $L_u \in \mathrm{RE}$.

Dowód. Skonstruujemy tak zwaną Uniwersalną Maszynę Turinga, to znaczy Maszynę Turinga U, która symuluje daną jej maszynę M_i na zadanym słowie w_j. Będzie to maszyna 4-taśmowa. Na pierwszej taśmie jest wejście, na drugiej taśma M_i (symbol taśmowy reprezentujemy odpowiednią ilością zer, które są oddzielane jedynkami, podwójna jedynka oznacza, że na następny symbol patrzy głowica), na trzeciej stan M_i (zakodowany tak samo), a czwarta będzie służyć do naszych obliczeń.

Na początku sprawdzamy, czy M_i jest poprawnym kodowaniem Maszyny Turinga (w przeciwnym wypadku możemy od razu odrzucić). Następnie przepisujemy w_j na drugą taśmę (w odpowiednim kodowaniu), a na trzeciej taśmie piszemy 0 (stan startowy). Następnie symulujemy ruch M_i – znajdujemy na pierwszej taśmie przejście dla aktualnego stanu i aktualizujemy zapisany stan na trzeciej taśmie i stan taśmy M_i na drugiej taśmie (często będzie to wymagało przesuwania całego zakodowanego napisu – do tego możemy wykorzystać ostatnią taśmę, traktując ją jako pamięć podręczną). Akceptujemy lub odrzucamy dokładnie wtedy, gdy symulowana maszyna to zrobi. □

Lemat 9. Zachodzi $L_u \notin \mathrm{R}$.

Dowód. Jeśli L_u jest rekurencyjny, to $\overline{L_u}$ również. Zakładamy nie wprost, że istnieje maszyna $\overline{M_u}$, która rozpoznaje $\overline{L_u}$. Za jej pomocą skonstruujemy maszynę dla języka L_d, co da sprzeczność.

Mając w_i na wejściu chcemy zaakceptować, jeśli $w_i \notin L(M_i)$. Zapisujemy na taśmie $w_i 111 w_i$ – jest to kodowanie maszyny M_i i słowa w_i dla maszyny rozpoznającej $\overline{L_u}$. Dalej podążamy za regułami ustalonymi przez przejścia maszyny $\overline{M_u}$. Zaakceptujemy dokładnie wtedy, gdy $w_i \notin L(M_i)$, a więc będziemy akceptować język L_d – sprzeczność. □

Definicja 65. Mówimy, że język $L_1 \subseteq \Sigma^*$ redukuje się do języka $L_2 \subseteq \Sigma'^*$ (co oznaczamy $L_1 \leq L_2$), jeśli istnieje algorytm (Maszyna Turinga z własnością stopu), który dla $x \in \Sigma^*$ oblicza $f(x) \in \Sigma'^*$ takie, że $x \in L_1 \iff f(x) \in L_2$.

Lemat 10. Niech $L_1, L_2 \subseteq \Sigma^*$ będą takie, że $L_1 \leq L_2$. Jeśli L_1 jest nierozstrzygalny, to L_2 też jest nierozstrzygalny. Podobnie jeśli L_1 nie jest rekurencyjnie przeliczalny, to L_2 też nie jest.

Dowód. Załóżmy, że L_2 jest rozstrzygalny, a L_1 nie. Istnieje maszyna M_r obliczająca dla wejścia x funkcję $f(x)$ taką, że $x \in L_1 \iff f(x) \in L_2$. Z założenia istnieje Maszyna Turinga M_l z własnością stopu dla L_2. Możemy przekształcić wejście x na $f(x)$ za pomocą M_r, a następnie zasymulować M_l i zwrócić taką odpowiedź, jak ona. W ten sposób skonstruowaliśmy Maszynę Turinga dla języka L_1 – sprzeczność.

Identyczny dowód działa w przypadku rekurencyjnej przeliczalności, tylko maszyna wyjściowa (i M_l) nie ma własności stopu. □

Definicja 66. Własnością języków rekurencyjnie przeliczalnych nazywamy pewien podzbiór języków rekurencyjnie przeliczalnych (np. własność bycia językiem regularnym, własność bycia językiem pustym). Własność jest nietrywialna, jeśli jest niepusta i niepełna. Kodowaniem (językiem) własności $\mathcal{P}$ nazywamy język $L_{\mathcal{P}} = \{M : L(M) \in \mathcal{P}\}$.

Twierdzenie 29 (Rice). Niech $\mathcal{P}$ będzie nietrywialną własnością języków rekurencyjnie przeliczalnych. Wtedy $L_{\mathcal{P}}$ jest nierozstrzygalny.

Dowód. Załóżmy, że $\emptyset \notin \mathcal{P}$. Z nietrywialności $\mathcal{P}$ istnieje $L \in \mathcal{P}$, który jest akceptowany przez Maszynę Turinga M_l. Zredukujemy język uniwersalny L_u do $L_{\mathcal{P}}$, co udowodni, że $L_{\mathcal{P}}$ nie jest rozstrzygalny.

Dla wejścia $\langle M_i, w_j \rangle$ chcemy stwierdzić, czy $w_j \in L(M_i)$. Konstruujemy maszynę M_{M_i,w_j}, która po wczytaniu wejścia x ignoruje je, zapisuje na taśmie w_j a następnie uruchamia M_i na tym słowie. Jeśli M_i akceptuje w_j, to uruchamia M_l na x i akceptuje wtedy, kiedy ona.

Jeśli $\langle M_i, w_j \rangle \in L_u$, to M_i akceptuje w_j, a więc $L\left(M_{M_i,w_j}\right) = L(M_l) = L$, czyli $M_{M_i,w_j} \in L_{\mathcal{P}}$. Jeśli natomiast $\langle M_i, w_j \rangle \notin L_u$, to M_i nie akceptuje w_j, a więc $L\left(M_{M_i,w_j}\right) = \emptyset \notin \mathcal{P}$.

Zatem $\langle M_i, w_j \rangle \in L_u \iff M_{M_i,w_j} \in L_{\mathcal{P}}$, czyli redukcja jest zakończona. Pozostaje rozważyć

przypadek $\emptyset \in \mathcal{P}$. Wtedy $\overline{\mathcal{P}}$ jest nietrywialną własnością, dla której można powtórzyć powyższy dowód. $\overline{\mathcal{P}}$ jest nierozstrzygalny, a więc $\mathcal{P}$ również. □

Wniosek. Nierozstrzygalne są np. własności bycia językiem pustym, niepustym, regularnym, bezkontekstowym.

Twierdzenie 30. Problem INTERSECTION-NON-EMPTINESS dla CFG polega na stwierdzeniu, czy dla gramatyk bezkontekstowych G_1, G_2 zachodzi $L(G_1) \cap L(G_2) \neq \emptyset$. Ten problem jest nierozstrzygalny.

Dowód. Oznaczmy $L_{ne} = \{M : L(M) \neq \emptyset\}$. Z twierdzenia Rice'a ten język jest nierozstrzygalny. Wskażemy redukcję $L_{ne} \leq$ INTERSECTION-NON-EMPTINESS, co zakończy dowód. Dla Maszyny Turinga $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ znajdziemy takie dwie gramatyki bezkontekstowe G_1, G_2, że zachodzi $L(M) \neq \emptyset \iff L(G_1) \cap L(G_2) \neq \emptyset$.

Obliczeniem Maszyny Turinga nazywamy ciąg konfiguracji $w_0 \vdash_M w_1 \vdash_M \ldots \vdash_M w_{2k-1}(w_{2k})$, gdzie w_0 jest konfiguracją startową. Obliczenie jest akceptujące, jeśli konfiguracja $w_{2k-1}(w_{2k})$ jest w stanie akceptującym. Przez $\text{accept}(M)$ oznaczamy zbiór obliczeń akceptujących. Obliczenie będziemy kodować jako słowo $w_0 \# w_1^R \# w_2 \# w_3^R \# \ldots \# w_{2k-1}^R (w_{2k}) \#$. Skonstruujemy dwie gramatyki tak, że przecięcie ich języków będzie dokładnie zbiorem kodowań akceptujących obliczeń. To zakończy dowód.

Oznaczmy $L_{\text{even}} = \left\{u \# v^R \# : u, v \in \Gamma^* Q \Gamma \Gamma^*, u \vdash_M v\right\}$. Skonstruujemy dla niego gramatykę G_{even}. Mamy produkcje:

- $S \to \text{Right}\# \mid \text{Left}\#$, gdzie Right, Left będą odpowiadać za przejścia w prawo i lewo.
- $C \to aCa \mid \#$ dla każdego $a \in \Gamma$. Ta produkcja będzie generować $w \# w^R$.
- $\text{Left} \to a\text{Left}a$ i $\text{Right} \to a\text{Right}a$ dla $a \in \Gamma$. Te produkcje tworzą symbole stojące na lewo od głowicy.
- $\text{Left} \to yqxCzyq_1$ dla $\delta(q, x) = (q_1, z, -1)$ i $y \in \Gamma$. To jest okolica, w której następuje środkowe lewe przejście.
- $\text{Right} \to qxyCyq_1 z$ dla $\delta(q, x) = (q_1, z, 1)$. Analogiczne przejście prawe.
- $S \to qxCzBq_1\#$ dla $\delta(q, x) = (q_1, z, -1)$. Jest to lewe lewe przejście. Ono nie może przechodzić przez Left, bo nie możemy dołożyć żadnych symboli po lewej stronie konfiguracji.
- $\text{Right} \to qx\#Bq_1 z$ dla $\delta(q, x) = (q_1, z, 1)$. Jest to prawe prawe przejście.

Mamy $L(G_{\text{even}}) = L_{\text{even}}$. Analogicznie konstruujemy gramatykę G_{odd} dla podobnego języka $L_{\text{odd}} = \left\{v^R \# u \# : u, v \in \Gamma^* Q \Gamma \Gamma^*, v \vdash_M u\right\}$. Niech G_1 generuje $(L_{\text{even}})^* (\{\varepsilon\} \cup \Gamma^* F \Gamma \Gamma^* \#)$ i niech G_2 generuje $q_0 (\{B\} \cup \Sigma^+) \# (L_{\text{odd}})^* (\{\varepsilon\} \cup \Gamma^* F \Gamma \Gamma^* \#)$.

Rozważmy akceptujące obliczenie o kodowaniu $w_0 \# w_1^R \# \ldots \# w_{2k-1}^R (w_{2k}) \#$. Mamy $w_0 = q_0 w$ dla słowa w i $w_{2k-1}(w_{2k}) \in \Gamma^* F \Gamma \Gamma^*$. Dla nieparzystej długości obliczenia G_1 może najpierw wziąć kilka słów z L_{even}, a potem zakończyć konfiguracją akceptującą. G_2 bierze konfigurację początkową (bo musi), a potem kilka słów z L_{odd}. Dla nieparzystej długości obliczenia jest analogicznie (na odwrót).

Jeśli słowo jest akceptowane przez obie gramatyki, to jest postaci $w_0 \# w_1^R \# \ldots \# w_{2k-1}^R (w_{2k}) \#$, gdzie $w_{2i} \vdash_M w_{2i+1}$ (należenie części słowa do L_{even}), $w_{2i+1} \vdash_M w_{2i+2}$ (to samo dla L_{odd}), w_0 jest konfiguracją początkową (z G_2), a $w_{2k-1}(w_{2k})$ jest konfiguracją akceptującą (odpowiednio z G_2 lub G_1).

Na koniec zauważmy, że konstrukcja tych gramatyk jest wykonalna za pomocą Maszyny Turinga – przechodzimy po wszystkich przejściach w wejściowej maszynie i po wszystkich symbolach taśmowych i dodajemy do gramatyki odpowiednie produkcje. To ostatecznie kończy dowód. □

Uwaga. Zauważmy, że powyższy problem jest w RE – Maszyna Turinga może generować kolejne słowa, a następnie sprawdzać, czy należą do obu gramatyk. Zatem również L_{ne} jest w RE.

Twierdzenie 31. Problem EQUIVALENCE dla gramatyk bezkontekstowych nie jest rekurencyjnie przeliczalny.

Dowód. Pokażemy, że problem uniwersalności (czy język gramatyki jest pełny) nie jest w RE. Da nam to tezę, bo można ten problem zredukować do problemu równoważności – konstruujemy trywialną gramatykę akceptującą język pełny i pytamy, czy jej język jest taki sam jak język gramatyki z wejścia. Zrobimy to za pomocą redukcji z $L_e = \{M : L(M) = \emptyset\}$. Ten język nie jest w RE, bo jego dopełnienie L_{ne} jest w RE $\setminus$ R.

Dla Maszyny Turinga M niech invalid $(M) = (\Sigma \cup \Gamma \cup \{\#\})^* \setminus \text{accept}(M)$ będzie zbiorem słów, które nie kodują akceptującego obliczenia M (odpowiednie kodowanie zdefiniowaliśmy w poprzednim dowodzie). Do tego języka należą słowa, które:

1. mają dwa # z rzędu lub nie mają żadnych lub nie mają # na końcu.
2. są w postaci $x_1\#x_2\#\dots x_n\#$, ale x_i nie jest poprawną konfiguracją (lub jej odbiciem), czyli nie ma w sobie dokładnie jednego stanu.
3. x_1 nie jest początkowa.
4. x_n nie jest akceptująca.
5. nie zachodzi $x_i \vdash_M x_{i+1}^R$ dla i parzystego.
6. nie zachodzi $x_i^R \vdash_M x_{i+1}$ dla i nieparzystego.

Dla każdego z tych przypadków da się skonstruować gramatykę bezkontekstową. W pierwszym generujemy wszystkie słowa bez # lub dwa z rzędu i dowolne litery dookoła nich lub nie-# i dowolne litery na lewo. W drugim generujemy niepoprawną konfigurację i dokładamy po obu stronach dowolne litery. W trzecim i czwartym analogicznie. W piątym i szóstym możemy postępować identycznie jak przy generowaniu poprawnych przejść, ale zamiast zmieniać stan i symbol zgodnie z przejściem zmieniamy je na każdy inny sposób (sposobów jest skończenie wiele). Musimy jeszcze uwzględnić niepoprawne przejścia, w których zmieniają się stany poza okolicą głowicy (podczas generowania konfiguracji dodajemy produkcje, które stawiają różne symbole, a następnie oddzielają słowa za pomocą # i po obu jego stronach dają dowolne znaki).

Składamy te wszystkie przypadki w jedną gramatykę, która generuje język invalid (M). Teraz pełność jej języka jest równoważna nieistnieniu żadnego obliczenia akceptującego, a więc pustości języka $L(M)$. To kończy redukcję, a więc i dowód. □

Definicja 67. W problemie odpowiedniości Posta (POST, PCP – Post Correspondence Problem) dostajemy na wejściu dwa skończone ciągi słów nad alfabetem Σ: $S = (s_1, \dots, s_n)$ i $T = (t_1, \dots, t_n)$. Pytamy o to, czy istnieje skończony ciąg indeksów $j_1, \dots, j_m$ taki, że $j_i \in [n]$ dla każdego $i \in [m]$ oraz $s_{j_1} \dots s_{j_m} = t_{j_1} t_{j_m}$.

Twierdzenie 32. Problem PCP jest nierozstrzygalny.

12. Rekurencyjna przeliczalność

2025-05-18

Definicja 68. Automat z k stosami (k-PDA) $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ definiujemy podobnie jak zwykłe PDA z tym, że δ ma typ $\left(Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma^k\right) \to \mathcal{P}\left(Q \times (\Gamma^*)^k\right)$. Opis chwilowy i relację przejścia dla k-PDA definiujemy podobnie jak dla zwykłego PDA (mamy k różnych stosów zamiast jednego). Język akceptowany definiujemy identycznie (poprzez stany akceptujące).

Twierdzenie 33. Język L jest akceptowany przez pewne k-PDA dla $k \geq 2$ wtedy i tylko wtedy, gdy jest akceptowany przez pewne 2-PDA.

Dowód. ($\Longrightarrow$) Pokażemy, że jesteśmy w stanie symulować 3-PDA za pomocą 2-PDA. Na pierwszym stosie będziemy odkładać symbole ze wszystkich trzech stosów, a drugi będzie stosem roboczym. Dla każdego symbolu stosowego $A \in \Gamma$ wprowadzamy trzy symbole A^1, A^2, A^3. Numer oznacza z jakiego stosu pochodzi symbol. Oba nasze stosy zaczynają z nowym symbolem startowym $\overline{Z_0}$.

Wrzucamy ε-przejściami symbole Z_0^1, Z_0^2, Z_0^3 na pierwszy stos. Następnie zaczynamy symulować pracę 3-PDA. Zaczynamy ze stanu s_q, gdzie q jest stanem w 3-PDA (początkowo stanem startowym). Za pomocą ε-przejść przerzucamy kolejne symbole z pierwszego stosu na drugi. Pierwszy odczytany symbol pochodzący z danego stosu 3-PDA nie zostaje odłożony na drugi stos. Jego wartość zostaje zapisana poprzez zmianę stanu naszego 2-PDA. W końcu odczytamy po co najmniej jednym symbolu z każdego z trzech stosów i będziemy w stanie s_{q,A^1,B^2,C^3}. Przechodzimy do stanu $w_{p,\alpha^1,\beta^2,\gamma^3}$, gdzie p jest stanem a α, β, γ są ciągami symboli stosowych. Robimy to zgodnie z funkcją przejścia symulowanego 3-PDA (konsumujemy przy tym odpowiednią literę ze słowa). Następnie przekładamy wszystko z drugiego stosu na pierwszy i dopisujemy do niego $\alpha^1\beta^2\gamma^3$. Przechodzimy do stanu s_p. W ten sposób przeprowadziliśmy symulację jednego kroku 3-PDA. Jeśli za akceptujące uznamy takie stany s_q, że q jest stanem akceptującym 3-PDA, to będziemy akceptować dokładnie te słowa, co ono.

W analogiczny sposób można symulować k-PDA za pomocą $(k-1)$-PDA dla $k \geq 3$ – trzy ostatnie stosy zostają zastąpione dwoma, a pozostałe nic nie robią poza przejściem, w którym zmienia się stan symulowanego automatu. Wtedy zachowują się zgodnie z oryginalną funkcją przejścia. Teza wynika z indukcji.

($\Longleftarrow$) Oczywiste. □

Twierdzenie 34. Język L jest rekurencyjnie przeliczalny wtedy i tylko wtedy, gdy jest akceptowany przez pewne 2-PDA (równoważnie k-PDA dla $k \geq 2$).

Dowód. ($\Longrightarrow$) 3-taśmowa Maszyna Turinga jest w stanie symulować 2-PDA. Na pierwszej taśmie trzyma słowo wejściowe, a na dwóch pozostałych symuluje stosy.

($\Longleftarrow$) Możemy symulować Maszynę Turinga za pomocą 2-PDA. Konfiguracji Maszyny Turinga $X_1 \dots X_{i-1} q X_i \dots X_n$ będzie odpowiadać $X_1 \dots X_{i-1}$ leżące na pierwszym stosie (czytając od dołu) i $X_n \dots X_i$ na drugim oraz stan s_q. Symulując krok Maszyny Turinga zmieniamy wartość X_i na inną i „przesuwamy głowicę" – przerzucamy X_i na lewy stos lub X_{i-1} na prawy. W razie potrzeby dopisujemy blanki. W ten sposób jesteśmy w stanie symulować krok Maszyny Turinga. Nasze 2-PDA zacznie działanie od wczytania całego słowa na pierwszy stos i przerzucenia go na drugi (w odwrotnej kolejności), a następnie będzie symulować Maszynę Turinga i akceptować, gdy ona zaakceptuje. □

Definicja 69. Graf funkcji częściowej $f : \mathbb{N} \to \mathbb{N}$ to zbiór słów postaci $[x, f(x)]$, gdzie x oraz $f(x)$ są zapisane binarnie (w takim kodowaniu, że wiemy gdzie zaczynają się i kończą odpowiednie wartości).

Definicja 70. Maszyna Turinga oblicza funkcję częściową f, jeśli dostając słowo wejściowe x kończy działanie z zapisanym na taśmie jedynie słowem $f(x)$ zawsze, gdy $f(x)$ jest określone.

Twierdzenie 35. Istnieje Maszyna Turinga obliczająca (częściową) funkcję f wtedy i tylko wtedy, gdy istnieje Maszyna Turinga akceptująca graf f.

Dowód. ($\Longrightarrow$) Mając zadane $[x, y]$ możemy odczytać x, zasymulować maszynę obliczającą $f(x)$ i zaakceptować, jeśli $f(x) = y$ (możemy się zapętlić, gdy funkcja nie jest określona na x, ale wtedy zadane słowo na pewno nie należy do grafu funkcji).

($\Longleftarrow$) Mając zadane x możemy brać kolejne słowa y w jakimś porządku (np. $\preccurlyeq$) i sprawdzać, czy $[x, y]$ należy do grafu f. Jeśli tak, to zapisujemy na taśmie y i kończymy. Maszyna akceptująca graf funkcji może się jednak zapętlić dla $y \neq f(x)$. Radzimy sobie z tym w następujący sposób: będziemy symulować naraz wiele obliczeń tej maszyny. Trzymamy na którejś taśmie kolejkę wszystkich konfiguracji (każda dla różnej wartości y). Przechodzimy przez nie wszystkie i wykonujemy jeden krok w każdej symulacji. Następnie zaczynamy symulację dla nowej wartości y i również wykonujemy jej jeden krok (i dodajemy konfigurację do kolejki), a następnie zaczynamy taką procedurę od nowa. Jeśli $f(x)$ jest określona, to w końcu znajdziemy dobrego y i doprowadzimy do końca całe obliczenie dla niego, a więc będziemy mogli zapisać go na taśmie i zakończyć. □

Propozycja 12. Języki rekurencyjnie przeliczalne (rekurencyjne) są zamknięte na sumę, przecięcie, konkatenację i gwiazdkę Kleene'ego.

Dowód. Rozważmy języki L_1, L_2 i Maszyny Turinga M_1, M_2 takie, że $L(M_1) = L_1$ i $L(M_2) = L_2$. Maszyna dla $L_1 \cup L_2$ może symulować M_1 i M_2 jednocześnie i akceptować wtedy, gdy jedna z nich zaakceptuje. Maszyna dla $L_1 \cap L_2$ może najpierw zasymulować M_1, a potem M_2 i zaakceptować wtedy, gdy obie zaakceptują.

Dla $L_1 L_2$ możemy skonstruować niedeterministyczną maszynę, która dla słowa w zgadnie podział $w = vu$, a następnie zasymuluje M_1 na v i M_2 na u. Dla L_1^* możemy ponownie niedeterministycznie zgadnąć liczbę n i podział $w = v_1 \dots v_n$, a następnie zasymulować M_1 na każdej części podziału. □

Definicja 71. Enumeratorem dla zbioru $S \subseteq \mathbb{N}$ nazywamy Maszynę Turinga, która nigdy nie kończy działania i wypisuje na jednej z taśm słowo $10^{i_1}10^{i_1}1\dots$ takie, że $i_1, i_2 \in S$ oraz kolejność występowania $i_1, i_2, \dots$ jest dowolna. Do tego dla każdego $n \in S$ napis 0^n pojawia się na taśmie w pewnym momencie działania maszyny.

Twierdzenie 36. Zbiór $S \subseteq \mathbb{N}$ jest rekurencyjnie przeliczalny wtedy i tylko wtedy, gdy istnieje dla niego enumerator.

Dowód. ($\Longrightarrow$) Podobnie jak przy obliczaniu funkcji częściowych będziemy brać kolejne $a \in \mathbb{N}$ i sprawdzać, czy należą do S. Będziemy utrzymywać kolejkę z konfiguracjami maszyny akceptującej S (dla różnych wartości a), wykonywać po jednym kroku w każdej i rozpoczynać obliczenie dla kolejnej w $\preccurlyeq$ liczby a. Gdy któreś obliczenie zaakceptuje możemy wypisać odpowiadającą mu liczbę na pierwszą taśmę w odpowiednim formacie.

($\Longleftarrow$) Aby sprawdzić, czy $a \in S$, będziemy symulować enumerator. Akceptujemy, jeśli w pewnym momencie wypisze 0^a. □

Twierdzenie 37. Zbiór $S \subseteq \mathbb{N}$ jest rekurencyjny wtedy i tylko wtedy, gdy istnieje dla niego enumerator wypisujący elementy w porządku $\preccurlyeq$.

Dowód. ($\Longrightarrow$) Symulujemy maszynę dla S na kolejnych $a \in \mathbb{N}$ zgodnie z $\preccurlyeq$ i wypisujemy, jeśli maszyna zaakceptuje. Nie musimy obawiać się o zapętlenie się maszyny, więc nie trzeba nic robić równolegle.

($\Longleftarrow$) Aby sprawdzić, czy $a \in S$, będziemy symulować enumerator. Akceptujemy, jeśli w pewnym momencie wypisze 0^a. Jeśli zacznie wypisywać liczby większe w $\preccurlyeq$ od a, to możemy odrzucić. □

13. Złożoność obliczeniowa

2025-05-28

Definicja 72. Obliczeniem Maszyny Turinga M na słowie w nazywamy ciąg konfiguracji $q_0 w \vdash_M \dots \vdash_M \Gamma^* \{q_{\text{acc}}, q_{\text{rej}}\} \Gamma\Gamma^*$.

Definicja 73. Niedeterministyczna Maszyna Turinga M działa w czasie $T : \mathbb{N} \to \mathbb{N}$, jeśli każde obliczenie na słowie $w \in \Sigma^*$ składa się z co najwyżej $T(|w|)$ kroków.

Definicja 74. Język $L \subseteq \Sigma^*$ należy do PTIME (oznaczane też P), jeśli istnieje wielomian p i deterministyczna Maszyna Turinga M działająca w czasie p taka, że $L(M) = L$.

Definicja 75. Język $L \subseteq \Sigma^*$ należy do NPTIME (oznaczane też NP), jeśli istnieje wielomian p i niedeterministyczna Maszyna Turinga M działająca w czasie p taka, że $L(M) = L$.

Definicja 76. Mówimy, że język $L_1 \subseteq \Sigma^*$ redukuje się wielomianowo (w czasie wielomianowym) do języka $L_2 \subseteq \Sigma'^*$ (co oznaczamy $L_1 \leq_P L_2$), jeśli $L_1 \leq L_2$ oraz świadczący o tym algorytm (deterministyczna Maszyna Turinga) działa w czasie p, gdzie p jest pewnym wielomianem.

Definicja 77. Język (problem) $L \subseteq \Sigma^*$ jest NP-trudny, jeśli dla każdego $L' \in \text{NP}$ zachodzi $L' \leq_P L$.

Definicja 78. Język $L \subseteq \Sigma^*$ jest NP-zupełny, jeśli jest NP-trudny i należy do NP.

Propozycja 13. Jeśli L jest NP-trudny i $L \leq_P L'$, to L' jest NP-trudny.

Dowód. Dla każdego $L'' \in \mathrm{NP}$ i $x \in \Sigma^*$ istnieje $f(x) : x \in L'' \iff f(x) \in L$. Do tego $|f(x)| \leq p(x)$ dla pewnego wielomianu p, bo algorytm redukcji działa w czasie wielomianowym. Do tego istnieje $g(x) : x \in L \iff g(x) \in L'$.

Łącząc te fakty dostajemy $x \in L'' \iff g(f(x)) \in L'$. Obie funkcje f, g obliczamy w czasie wielomianowym, a rozmiar $f(x)$ jest ograniczony przez wielomian, zatem obliczenie $g \circ f$ również odbywa się wielomianowo. Zatem istnieje odpowiednia redukcja i L' jest NP-trudny. □

Propozycja 14. Jeśli pewien problem NP-trudny jest w P, to P = NP.

Dowód. Mając taki problem NP-trudny możemy zredukować dowolny problem z NP do niego, a następnie rozwiązać go wielomianowo (bo jest w P, a rozmiar wejścia po redukcji jest wielomianowo zależny od początkowego wejścia). □

Definicja 79. Problem spełnialności Boolowskiej (SAT) to problem stwierdzenia, czy zadana formuła zdaniowa φ jest spełnialna czy nie.

Lemat 11. SAT jest w NP.

Dowód. Można skonstruować niedeterministyczną Maszynę Turinga, która zgaduje wartościowanie (zapisuje na taśmie wartości kolejnych zmiennych, mając niedeterministyczne przejścia na obie opcje), a następnie sprawdza, czy wartościowanie spełnia zadaną formułę. □

Twierdzenie 38 (Cooke, Levin)**.** SAT jest NP-zupełny.

Dowód. Już pokazaliśmy, że ten problem jest w NP. Teraz wskażemy redukcję do SAT-a z dowolnego $L \in \mathrm{NP}$. Rozważmy takie L. Istnieje wielomian p i niedeterministyczna Maszyna Turinga działająca w czasie p taka, że $w \in L$ wtedy i tylko wtedy, gdy istnieje obliczenie akceptujące dla w długości co najwyżej $p(|w|)$. Będziemy też zakładać, że maszyna po dojściu do stanu akceptującego pozostaje w nim i zaczyna przesuwać głowicę w prawo, nie zmieniając taśmy. Dzięki temu jeśli istnieje akceptujące obliczenie, to jest ono w stanie akceptującym po dokładnie $p(|w|)$ krokach.

Takie obliczenie możemy reprezentować tabelką, w której w wierszu trzymamy jedną konfigurację, a w kolejnych wierszach są kolejne konfiguracje obliczenia. Każdy wiersz będzie indeksowany wartościami od $-p(|w|)$ do $p(|w|)$ (jeśli głowica będzie początkowo w 0, to żadna konfiguracja nie będzie wymagała sięgnięcia poza te indeksy, bo M może wykonać co najwyżej $p(|w|)$ kroków). Wszystkich konfiguracji jest najwyżej $p(|w|) + 1$, więc tyle będzie wierszy. Oznaczamy $C = \{-p(|w|), \ldots, p(|w|)\}$ i $R = \{0, \ldots, p(|w|)\}$.

Dla każdego $w \in L$ skonstruujemy formułę ψ_w taką, że $w \in L \iff \psi_w$ jest spełnialna. W ψ_w będą zmienne $x_{i,j,a}$ dla $i \in R, j \in C, a \in \Gamma \cup Q$. Prawdziwość tej zmiennej będzie oznaczać, że w tabelce obliczenia na polu (i, j) występuje znak a. Do tego wprowadzimy ograniczenia które spowodują, że wartościowanie spełniajace ϕ_w będzie zadawać obliczenie akceptujące i na odwrót.

Zaczynamy od formuł

$$A_{ij} = \bigvee_{a \in \Gamma \cup Q} x_{i,j,a}, \quad B_{ij} = \bigwedge_{\substack{a,b \in \Gamma \cup Q, \\ a \neq b}} \neg(x_{i,j,a} \wedge x_{i,j,b}), \quad \phi_1 = \bigwedge_{\substack{i \in R \\ j \in C}} A_{ij} \wedge B_{ij}.$$

ϕ_1 wymusza, aby w każdym polu był dokładnie jeden znak. Teraz wymusimy konfigurację startową i końcową

$$\phi_2 = \bigwedge_{i \in [|w|]} x_{0,i,w_i} \wedge x_{0,0,q_0} \wedge \bigwedge_{i \in C \setminus (\{0\} \cup [|w|])} x_{0,i,B},$$

$$\phi_3 = \bigvee_{q_f \in F} \bigvee_{j \in C} x_{p(|w|),j,q_f}.$$

Pozostało nam wymusić poprawność przejść. Zdefiniujemy formułę oznaczającą, że symbole na

(i_1, j_1) i (i_2, j_2) są sobie równe:

$$E(i_1, j_1, i_2, j_2) = \bigwedge_{a \in \Gamma \cup Q} ((x_{i_1,j_1,a} \wedge x_{i_2,j_2,a}) \vee (\neg x_{i_1,j_1,a} \wedge \neg x_{i_2,j_2,a})).$$

A teraz formuły oznaczające przejście dla stanu q i symbolu a na koordynatach (i, j) w prawo/lewo:

$$T_r(q, a, i, j) = \bigvee_{(q',a',1) \in \delta(q,a)} ((x_{i,j,q} \wedge x_{i,j+1,a}) \implies (E(i, j-1, i+1, j-1) \wedge x_{i+1,j,a'} \wedge x_{i+1,j+1,q'})),$$

$$T_l(q, a, i, j) = \bigvee_{(q',a',-1) \in \delta(q,a)} ((x_{i,j,q} \wedge x_{i,j+1,a}) \implies (x_{i+1,j-1,q'} \wedge E(i, j-1, i+1, j) \wedge x_{i+1,j+1,a'})).$$

Obejmują one przejście dla pola w którym jest stan i obu pól obok. Wykorzystamy to w następnym kroku. Zdefiniujemy formułę stwierdzającą, czy w danym polu nie znajduje się stan i formułę, która przepisuje symbol w konfiguracji, jeśli nie sąsiaduje on ze stanem.

$$NQ(i, j) = \bigwedge_{q \in Q} \neg x_{i,j,q},$$

$$K(i, j) = (NQ(i, j-1) \wedge NQ(i, j) \wedge NQ(i, j+1)) \implies E(i, j, i+1, j).$$

Pozostało nam wymusić poprawne przejścia za pomocą formuły

$$\phi_4 = \bigwedge_{\substack{q \in Q, a \in \Gamma \\ i \in R, j \in C}} (T_r(q, a, i, j) \vee T_l(q, a, i, j)) \wedge \bigwedge_{i \in R, j \in C} K(i, j).$$

Ustalamy $\psi_w = \phi_1 \wedge \phi_2 \wedge \phi_3 \wedge \phi_4$ i kończymy dowód. Zauważmy jeszcze, że każda z tych formuł ma długość wielomianową względem $|w|$ – rozmiar maszyny jest stałą, a więc każda pomocnicza formuła ma stałą długość i wykorzystaliśmy takich formuł wielomianowo wiele. □

Uwaga. Mówiąc o dopełnieniu jakiegoś problemu zazwyczaj pomijamy słowa niebędące poprawnymi kodowaniami instancji problemu i myślimy tylko o instancjach problemu, dla których odpowiedź jest przecząca.

Propozycja 15. Klasa P jest zamknięta na branie dopełnień.

Dowód. Możemy wziąć deterministyczną Maszynę Turinga dla naszego problemu i skonstruować identyczną maszynę z zamienionymi stanami akceptującymi i odrzucającymi. □

Uwaga. Pokazaliśmy, że P = coP – problemy z P to dokładnie te problemy, których dopełnienia są w P.

Definicja 80. Język $L \subseteq \Sigma^*$ należy do coNP, jeśli jego dopełnienie należy do NP, czyli istnieje wielomianowa niedeterministyczna Maszyna Turinga, która ma akceptujące obliczenie dokładnie dla $x \notin L$. Pojęcia coNP-trudności i zupełności wprowadzamy analogicznie jak dla NP.

Uwaga. W przypadku NP i maszyn niedeterministycznych nie możemy rozstrzygnąć dopełnienia tak samo jak dla P – niedeterminizm pozwala nam wybrać dowolne akceptujące obliczenie, ale nie mamy łatwego sposobu na sprawdzenie, czy żadne obliczenie nie jest akceptujące.

Propozycja 16. L jest NP-trudny wtedy i tylko wtedy, gdy $\overline{L}$ jest coNP-trudny.

Dowód. Jeśli $T \in$ NP, to istnieje $f(x) : x \in T \iff f(x) \in L$ dla NP-trudnego L, a więc $x \notin T \iff f(x) \notin L$, czyli mamy redukcję z $\overline{T}$ do $\overline{L}$. Analogicznie w drugą stronę. □

Wniosek. Problem UNSAT (dopełnienie SAT-a) jest coNP-zupełny.

Definicja 81. Problem TAUTOLOGY dla zadanej formuły zdaniowej polega na stwierdzeniu, czy jest tautologią.

Lemat 12. Problem TAUTOLOGY jest coNP-zupełny.

Dowód. Problem oczywiście jest w coNP (dopełnienie polega na stwierdzeniu, czy formuła ma niespełniające wartościowanie, co właściwie jest SAT-em). Do tego mamy redukcję UNSAT $\leq_P$ TAUTOLOGY, bo aby sprawdzić niespełnialność formuły wystarczy sprawdzić, czy jej dopełnienie jest tautologią. □

Propozycja 17. NP = coNP wtedy i tylko wtedy, gdy istnieje pewien NP-zupełny problem L taki, że $\overline{L}$ jest w NP.

Dowód. Wynikanie w prawo jest oczywiste, w drugą stronę z coNP-zupełności $\overline{L}$ dostajemy coNP $\subseteq$ NP, teraz jeśli $L' \in$ NP, to $\overline{L'} \in$ coNP $\subseteq$ NP, a więc $L' \in$ coNP. □

Definicja 82. Deterministyczna Maszyna Turinga M działa w pamięci $T : \mathbb{N} \to \mathbb{N}$, jeśli długość konfiguracji występujących w obliczeniach tej maszyny na słowie $w \in \Sigma^*$ nie przekracza $T(|w|)$.

Definicja 83. Język $L \subseteq \Sigma^*$ należy do PSPACE, jeśli istnieje wielomian p i deterministyczna Maszyna Turinga M działająca w pamięci p taka, że $L(M) = L$. Pojęcie zupełności i trudności definiujemy analogicznie jak dla innych klas.

Propozycja 18. Zachodzi P $\subseteq$ PSPACE, NP $\subseteq$ PSPACE i coNP $\subseteq$ PSPACE.

Dowód. Pierwsze zawieranie wynika z tego, że w wielomianowej liczbie kroków można wydłużyć konfigurację co najwyżej o tyle kroków. Niedeterministyczną Maszynę Turinga dla problemu z NP można symulować standardowo (na kolejce), albo skorzystać z jej własności stopu i ograniczyć się do symulowania jednego obliczenia na raz (DFS po drzewie obliczeń). W ten sposób możemy przejść po wszystkich obliczeniach maszyny wykorzystując naraz tylko tyle pamięci, co jedno obliczenie. Dla dowodu ostatniego zawierania zauważmy, że możemy symulować niedeterministyczną maszynę dla dopełnienia (które jest w NP) i zwrócić odwrotny wynik – zaakceptować tylko wtedy, gdy żadne zasymulowane obliczenie nie zaakceptuje. □

Definicja 84. QBF (Quantified Boolean Formula) to formuła zdaniowa, w której występują kwantyfikatory $\forall$ i $\exists$. Problem TQBF (True Quantified Boolean Formula) polega na stwierdzeniu, czy zadana formuła jest prawdziwa.

Lemat 13. TQBF jest w PSPACE.

Dowód. Możemy ustalić wartość jednej zmiennej i wstawić ją do formuły, a następnie wywołać się rekurencyjnie (formalnie wstawienie wartości do formuły nie jest takie proste, ale można po prostu pamiętać ustalane wartościowanie i po ustaleniu ostatniej zmiennej wyewaluować formułę). Robimy to dla obu wartości zmiennej. Zależnie od kwantyfikatora przy zmiennej formuła jest prawdziwa, jeśli oba wywołania rekurencyjne zwróciły prawdę lub jeśli co najmniej jedno zwróciło prawdę. □

Twierdzenie 39. TQBF jest PSPACE-zupełny.

Dowód. Pokażemy, że dla każdego $L \in$ PSPACE istnieje redukcja z L do TQBF. Zauważmy, że jeśli Maszyna Turinga dla języka L działa w pamięci p dla pewnego wielomianu p, to możliwych konfiguracji tej maszyny na słowie wejściowym w jest co najwyżej $p(|w|)\, n m^{p(|w|)}$, gdzie n to liczba stanów a m to liczba symboli taśmowych maszyny – najpierw wybieramy miejsce na stan, a potem wypełniamy pozostałe miejsca symbolami. Podczas swojego działania maszyna nie może dwa razy wejść do tej samej konfiguracji (bo się zapętli), więc mamy ograniczenie na czas działania maszyny. Oznaczmy je przez T i zauważmy, że T jest wykładnicze od $|w|$.

Podobnie jak dla SAT-a będziemy tworzyć zmienne boolowskie, które będą odpowiadać umieszczeniu danego symbolu w danym miejscu w konfiguracji. Oznaczmy $C = \{-p(|w|), \ldots, p(|w|)\}$. Zmienną konfiguracyjną I nazywamy zbiór zmiennych boolowskich $\{x_{i,A}\}_{i \in C, A \in \Gamma \cup Q}$. Poprzez $\exists_I$ i $\forall_I$ rozumiemy odpowiednie kwalifikowanie wszystkich zmiennych z I. Różne zmienne konfiguracyjne

będą rozłączne i będzie ich wielomianowo wiele, a więc wszystkich zmiennych boolowskich również będzie wielomianowo wiele.

Stworzymy formułę postaci

$$\exists_{I_0}\ \exists_{I_f}\ \ (S \wedge N \wedge F)\,,$$

gdzie I_0 i I_f to zmienne konfiguracyjne, które mają odpowiadać odpowiednio konfiguracji startowej i końcowej. Formuła S składa się ze zmiennych z I_0 i zapewnia, że tworzą one konfigurację startową. Analogicznie F dla I_f. Konstrukcja takich formuł została podana w dowodzie NP-zupełności SAT-a. N będzie formułą wymuszającą istnienie obliczenia przechodzącego z I_0 do I_f (to znaczy $I_0 \vdash_M^* I_f$ dla maszyny M), którą teraz skonstruujemy. Przez $N_i\,(I, J)$ będziemy oznaczać formułę wymuszającą, że ciąg obliczeń świadczący o $I \vdash_M^* J$ składa się z co najwyżej 2^i przejść. Skonstruujemy ją indukcyjnie.

Dla $i = 0$ wystarczy sprawdzić, czy $I \vdash_M J$ lub $I = J$ (tę równość rozumiemy jako równość konfiguracji, ale możemy też ją rozumieć jako formułę logiczną mówiącą, że odpowiadające sobie zmienne tych zmiennych konfiguracyjnych przyjmują tę samą wartość – w ten właśnie sposób będziemy jej używać w formułach). Formułę dla pierwszego przypadku przedstawiliśmy w dowodzie NP-zupełności SAT-a, a drugą właśnie skomentowaliśmy. Wystarczy wziąć alternatywę tych formuł, co daje nam formułę wielomianową od $|w|$. Dla $i \geq 1$ kładziemy

$$N_i\,(I, J) = \exists_K\ \forall_P\ \forall_Q\ \left(((I = P \wedge K = Q) \vee (K = P \wedge J = Q)) \implies N_{i-1}\,(P, Q)\right).$$

Formuła ta mówi, że istnieje taka konfiguracja K, że $N_{i-1}\,(I, K)$ i $N_{i-1}\,(K, J)$, co daje nam $N_i\,(I, J)$. W jej konstrukcji wykorzystaliśmy wielomianową od $|w|$ liczbę dodatkowych znaków (nieskładających się na $N_{i-1}\,(P, Q)$).

Niech $k = \lceil \log T \rceil$. Możemy przyjąć $N = N_k\,(I_0, I_f)$. k jest wielomianowe od $|w|$, a więc N również jest. To kończy naszą konstrukcję. Zauważmy jeszcze, że dla każdej zmiennej konfiguracyjnej powinniśmy mieć formułę która zapewnia, że w każdym polu jest dokładnie jeden znak. Możemy to zrobić identycznie jak w dowodzie NP-zupełności SAT-a. □

Uwaga. Istnieje sporo problemów otwartych związanych ze złożonością obliczeniową. Najważniejsze z nich to $\mathrm{P} = \mathrm{NP}$, $\mathrm{NP} = \mathrm{coNP}$ i $\mathrm{P} = \mathrm{NP} \cap \mathrm{coNP}$.

Uwaga. Mamy $\mathrm{REG}, \mathrm{CFL} \subseteq \mathrm{P}$, bo przynależność do języków tych klas można sprawdzać wielomianowo. Do tego $\mathrm{P}, \mathrm{NP}, \mathrm{coNP}, \mathrm{PSPACE} \subseteq \mathrm{R}$. Dla RE i dalej nie ma sensu mówić o złożoności, bo nie mamy nawet własności stopu.