Politechnika Wrocławska
Wydział Informatyki i Telekomunikacji

Kierunek: **Informatyka Algorytmiczna (INA)**

PRACA DYPLOMOWA
INŻYNIERSKA

Kombinatoryka języków formalnych

Wojciech Gabryelski

Opiekun pracy
Profesor Jacek Cichoń

Słowa kluczowe: kombinatoryka analityczna, wyrażenia regularne, automaty skończone

WROCŁAW 2023

STRESZCZENIE

Praca ta jest poświęcona problemowi zliczania słów zadanej długości należących do języka regularnego generowanego przez wyrażenie regularne. Omówione tu metody pozwalają wyznaczyć zwarty wzór na liczbę słów długości n z danego języka regularnego dla wszystkich liczb naturalnych n przy użyciu technik z kombinatoryki analitycznej. Zaimplementowane i opisane tu narzędzia mogą być wykorzystane do badania wpływu ograniczeń narzuconych na hasła w różnych systemach informatycznych na liczbę możliwych do wprowadzenia haseł danej długości, a tym samym na trudność złamania hasła. W pracy tej opisane są wszystkie pojęcia z dziedzin kombinatoryki analitycznej i języków formalnych niezbędne do zrozumienia głównej części pracy oraz zawarte są odniesienia do literatury szczegółowo opisującej wspomniane tu zagadnienia.

ABSTRACT

This work is devoted to the problem of counting words of a given length that belong to a regular language defined by a regular expression. The methods discussed here allow us to derive a compact formula for the number of words of length n from a given regular language for all natural numbers n using techniques related to analytical combinatorics. The tools implemented and described here can be used to study the impact of restrictions imposed on passwords in various IT systems on the number of passwords of a given length that can be entered, and thus on the difficulty of cracking the password. This work describes all the concepts from the fields of analytical combinatorics and formal languages that are necessary to understand the main part of this work and includes references to the literature describing in detail the issues mentioned here.

SPIS TREŚCI

WPROWADZENIE

Praca dyplomowa poświęcona jest problemowi zliczania słów języka regularnego o ustalonym rozmiarze. Zauważyć bowiem można, że pewne klasy języków regularnych są łatwe do opisania (np. wyrażanie regularne $(a+b)^*aa(a+b)^*$ opisuje słowa w których występują pod rząd kolejne dwa wystąpienia litery a), jednak bezpośrednie tłumaczenie takiego wyrażenia na funkcje tworzące nie daje prawidłowej odpowiedzi. W pracy omówimy metodę dokładnego wyznaczania liczby słów ustalonej długości dla zadanego wyrażenia regularnego.

W rozdziale 1 wprowadzamy wykorzystywane w dalszej części pracy oznaczenia, fakty i wzory. W szczególności omawiamy pojęcie klasy kombinatorycznej oraz związanych z nimi pojęcie funkcje tworzące. Podamy również uogólniony wzór dwumianowy Newtona.

W rozdziale 2 przedstawiamy pojęcia wyrażeń regularnych, języków regularnych i automatów skończonych oraz omawiamy związki między nimi.

W rozdziale 3 przedstawiamy metodę wyznaczania funkcji tworzącej (wprowadzonej w rozdziale 1) dla danego języka regularnego, a następnie obliczania dokładnej liczby słów zadanej długości należących do tego języka.

W rozdziale 4 omawiamy działanie programu wyznaczającego funkcję tworzącą języka generowanego przez podane wyrażenie regularne oraz role poszczególnych klas i metod w całym tym procesie.

W rozdziale 5 prezentujemy funkcje tworzące kilku języków regularnych danych przez wybrane wyrażenia regularne oraz przeprowadzamy badania wpływu ograniczeń narzuconych przez wyrażenia regularne na liczbę haseł zadanej długości opisanych przez to wyrażenie przy wykorzystaniu zaimplementowanego oprogramowania.

Rozdział 6 zawiera krótkie podsumowanie pracy dyplomowej, możliwości rozbudowania zbudowanych narzędzi oraz możliwości ich wykorzystania do innych zagadnień.

Wszystkie zrealizowane i omawiane w pracy algorytmy zostały napisane w wersji 11 języka C++ (patrz np. [5]) i skompilowane na urządzeniu z systemem operacyjnym Windows 10 oraz na maszynie wirtualnej z systemem Linux (wymagana jest instalacja biblioteki GMP do obliczeń numerycznych dowolnej dokładności (patrz [3]), więc rekomendowanym systemem operacyjnym do testowania programu jest system Linux ze względu na łatwość instalacji tej biblioteki).

1. POJĘCIA KOMBINATORYCZNE

W rozdziale tym omówimy pojęcia kombinatoryczne, które wykorzystywać będziemy w pracy. Omówione tutaj własności uogólnionych współczynników dwumianowych znaleźć można w klasycznej książce [2], zaś pojęcie, podstawowe konstrukcje i podstawowe własności klas kombinatorycznych znaleźć można w książce [1].

1.1. OZNACZENIA

W pracy będziemy stosowali standardową notację matematyczną. W szczególności korzystać będziemy z następujących oznaczeń:

1. $\mathbb{N}$ - zbiór liczb naturalnych (do zbioru tego zaliczamy również liczbę 0)
2. $\mathbb{C}$ - zbiór liczb zespolonych
3. $[n]$ - zbiór liczb $\{1, \ldots, n\}$ dla $n \in \mathbb{N}$ ($[0] = \emptyset$)
4. $[\![p]\!] = \begin{cases} 1, & \text{zdanie } p \text{ jest prawdziwe} \\ 0, & \text{zdanie } p \text{ jest fałszywe} \end{cases}$
5. $[x^n]A(x)$ - współczynnik przy x^n w szeregu $A(x) = \sum\limits_{n=0}^{\infty} a_n x^n$

1.2. ELEMENTY KOMBINATORYKI

Klasyczny symbol dwumianowy $\binom{n}{k}$ określony jest dla liczb naturalnych n oraz k. Można go uogólnić do funkcji $\binom{x}{k}$ dla dowolnych $x \in \mathbb{C}$:

$$\binom{x}{k} = \frac{\prod_{a=0}^{k-1}(x-a)}{k!}$$

Dla tak określonego uogólnienia współczynników dwumianowych zachodzi następujące uogólnienie wzoru dwumianowego Newtona

$$(1+x)^{\alpha} = \sum_{k \geq 0} \binom{\alpha}{k} x^k \,, \quad |x| < 1 \,.$$

Korzystając z następującej tożsamości (zwanej "górną negacją"): $\binom{x}{k} = (-1)^k \binom{k-x-1}{k}$ otrzymujemy następujący ważny dla nas wzór:

$$\frac{1}{(1-x)^n} = \sum_{k\geq 0} \binom{n+k-1}{k} x^n, \quad |x| < 1 \,. \tag{1.1}$$

Liczbami Fibonacciego nazywamy liczby F_n zdefiniowane w następujący sposób: $F_0 = 0$, $F_1 = 1$ oraz $F_n = F_{n-1} + F_{n-2}$ dla $n \geq 2$. Funkcja tworząca ciągu liczb Fibonacciego wyraża się wzorem

$$\sum_{n=0}^{\infty} F_n x^n = \frac{x}{1-x-x^2} \,. \tag{1.2}$$

1.3. KLASY KOMBINATORYCZNE

Definicja 1.1. Klasą kombinatoryczną *nazywamy parę* $\mathcal{C} = (C, |\cdot|)$ *taką, że* C *jest zbiorem niepustym, a* $|\cdot| : C \to \mathbb{N}$ *jest funkcją ze zbioru* C *w zbiór liczb naturalnych taką, że dla każdego* $n \in \mathbb{N}$ *zbiór* $\{c \in C : |c| = n\}$ *jest skończony. Liczbę* $|c|$ *nazywamy wagą lub rozmiarem elementu* c.

Klasę kombinatoryczną będziemy oznaczali kręconą literą, natomiast zbiór tej klasy będziemy oznaczali prostą literą. Dla klasy kombinatorycznej $\mathcal{C} = (C, |\cdot|)$ definiujmy

1. $C_n = \{c \in C : |c| = n\}$
2. $c_n = card(C_n)$

Analogiczne oznaczenia będziemy stosować dla innych liter alfabetu.

Definicja 1.2. Funkcją tworzącą *klasy* $\mathcal{C} = (C, |\cdot|)$ *będziemy nazywać (formalny) szereg potęgowy*

$$\mathcal{C}(x) = \sum_{n=0}^{\infty} c_n x^n \,.$$

Zauważmy, że

$$\begin{aligned} \mathcal{C}(x) &= \sum_{n=0}^{\infty} c_n x^n = \sum_{n=0}^{\infty} card(\{c \in C : |c| = n\}) x^n = \sum_{n=0}^{\infty} x^n \sum_{c\in C} [\![|c| = n]\!] \\ &= \sum_{c\in C} \sum_{n=0}^{\infty} [\![|c| = n]\!] x^n = \sum_{c\in C} x^{|c|}. \end{aligned}$$

W naszych rozważaniach często będziemy mieli do czynienia ze zwartą postacią szeregu $\mathcal{C}(x)$. Wówczas przyda się oznaczenie $[x^n]\mathcal{C}(x) = c_n$.

Definicja 1.3. *Dwie klasy* $\mathcal{A}$ *i* $\mathcal{B}$ *nazywamy* izomorficznymi *(*$\mathcal{A} \cong \mathcal{B}$*), gdy ich funkcje tworzące są identyczne.*

Łatwo można zauważyć, że klasy kombinatoryczne $\mathcal{A} = (A, |\cdot|_{\mathcal{A}})$ oraz $\mathcal{B} = (B, |\cdot|_{\mathcal{B}})$ są izomorficzne wtedy i tylko wtedy, gdy istnieje bijekcja $f : A \to B$ taka, że dla każdego $x \in A$ mamy $|f(x)|_{\mathcal{B}} = |x|_{\mathcal{A}}$.

1.3.1. Podstawowe konstrukcje klas kombinatorycznych

W części tej omówimy podstawowe metody konstruowania klas kombinatorycznych oraz kilka bazowych klas. Niech

$$\mathcal{E} = (\{\varepsilon\}, |\cdot|_{\mathcal{E}}) ,$$

gdzie $|\varepsilon|_{\mathcal{E}} = 0$ oraz

$$\mathcal{X} = (\{\bullet\}, |\cdot|_{\mathcal{X}}) ,$$

gdzie $|\bullet|_{\mathcal{X}} = 1$. Funkcje tworzące tych klas to $\mathcal{E}(x) = 1$ oraz $\mathcal{X}(x) = x$.

Przedstawimy teraz trzy podstawowe konstrukcje klas kombinatorycznych. Niech $\mathcal{A} = (A, |\cdot|_{\mathcal{A}})$ i $\mathcal{B} = (B, |\cdot|_{\mathcal{B}})$ będą klasami kombinatorycznymi.

1. Sumą klas $\mathcal{A}$ i $\mathcal{B}$ nazywamy klasę

   $$\mathcal{A} + \mathcal{B} = ((A \times \{\bullet\}) \cup (B \times \{\circ\}), |\cdot|_{\mathcal{A}+\mathcal{B}}) ,$$

   gdzie $\bullet$ oraz $\circ$ są dowolnymi różnymi elementami (suma klas kombinatorycznych $\mathcal{A}$ i $\mathcal{B}$ wymaga urozłącznienia zbiorów A i B), a funkcja $|\cdot|_{\mathcal{A}+\mathcal{B}} : (A \times \{\bullet\}) \cup (B \times \{\circ\}) \to \mathbb{N}$ wyraża się wzorem

   $$|c|_{\mathcal{A}+\mathcal{B}} = \begin{cases} |a|_{\mathcal{A}}, & c = (a, \bullet) \text{ dla } a \in A \\ |b|_{\mathcal{B}}, & c = (b, \circ) \text{ dla } b \in B \end{cases}$$

2. Produktem klas $\mathcal{A}$ i $\mathcal{B}$ nazywamy klasę

   $$\mathcal{A} \times \mathcal{B} = (A \times B, |\cdot|_{\mathcal{A}\times\mathcal{B}}),$$

   gdzie funkcja $|\cdot|_{\mathcal{A}\times\mathcal{B}} : A \times B \to \mathbb{N}$ wyraża się wzorem

   $$|(a, b)|_{\mathcal{A}\times\mathcal{B}} = |a|_{\mathcal{A}} + |b|_{\mathcal{B}}$$

3. Ciągami elementów klasy $\mathcal{A}$ takiej, że $a_0 = 0$ nazywamy klasę

   $$\mathbf{SEQ}(\mathcal{A}) = \mathcal{E} + \mathcal{A} + (\mathcal{A} \times \mathcal{A}) + (\mathcal{A} \times \mathcal{A} \times \mathcal{A}) + \ldots$$

Operacja $\mathbf{SEQ}$ jest określona tylko dla takich klas kombinatorycznych $\mathcal{A}$, że $a_0 = 0$. Gdyby w klasie $\mathcal{A}$ występował element e taki, że $|e|_{\mathcal{A}} = 0$, to do sumy $\bigcup_{n\geq 0} A^n$ należałyby

elementy (e), (e,e), (e,e,e), ... więc w sumie tej występowałoby nieskończenie wiele elementów rozmiaru 0.

Twierdzenie 1.1. *Dla dowolnych klas kombinatorycznych $\mathcal{A}$ oraz $\mathcal{B}$ mamy*

1. $(\mathcal{A}+\mathcal{B})(x) = \mathcal{A}(x) + \mathcal{B}(x)$
2. $(\mathcal{A}\times\mathcal{B})(x) = \mathcal{A}(x)\cdot\mathcal{B}(x)$
3. $(\mathbf{SEQ}(\mathcal{A}))(x) = (1-\mathcal{A}(x))^{-1}$, *o ile* $a_0 = 0$

Operacje arytmetyczne (dodawanie, mnożenie, odwrotność) występujące w powyższym twierdzeniu są operacjami na szeregach formalnych. W szczególności, produkt jest interpretowany jako iloczyn Cauchy'ego szeregów. Jeśli rozważane szeregi mają niezerowe promienie zbieżności, to mogą być interpretowane jako szeregi potęgowe i wtedy owe operacje formalne pokrywają się z analitycznymi operacjami na szeregach potęgowych. W szczególności, ostatni wzór można zapisać w postaci

$$(\mathbf{SEQ}(\mathcal{A}))(x) = \frac{1}{1-\mathcal{A}(x)}$$

Dowód. (1) Pierwsza część twierdzenia wynika z następującego ciągu równości :

$$(\mathcal{A}+\mathcal{B})(x) = \sum_{c\in(A\times\{\bullet\})\cup(B\times\{\circ\})} x^{|c|_{\mathcal{A}+\mathcal{B}}} = \sum_{a\in A} x^{|a|_{\mathcal{A}}} + \sum_{b\in B} x^{|b|_{\mathcal{B}}} = \mathcal{A}(x)+\mathcal{B}(x)\,.$$

(2) Druga część twierdzenia wynika z następujących równości:

$$(\mathcal{A}\times\mathcal{B})(x) = \sum_{c\in A\times B} x^{|c|_{\mathcal{A}\times\mathcal{B}}} = \sum_{(a,b)\in A\times B} x^{|a|_{\mathcal{A}}+|b|_{\mathcal{B}}} = \sum_{(a,b)\in A\times B} x^{|a|_{\mathcal{A}}} x^{|b|_{\mathcal{B}}} = \left(\sum_{a\in A} x^{|a|_{\mathcal{A}}}\right)\left(\sum_{b\in B} x^{|b|_{\mathcal{B}}}\right) = \mathcal{A}(x)\cdot\mathcal{B}(x)\,.$$

(3) Zauważmy, że

$$(\mathbf{SEQ}(\mathcal{A}))(x) = 1 + \mathcal{A} + (\mathcal{A}\times\mathcal{A})(x) + (\mathcal{A}\times\mathcal{A}\times\mathcal{A}))(x) + \ldots = \sum_{n\geq 0}\mathcal{A}(x)^n\,.$$

Następnie

$$\left(\sum_{n\geq 0}\mathcal{A}(x)^n\right)(1-\mathcal{A}(x)) = \left(\sum_{n\geq 0}\mathcal{A}(x)^n\right) - \left(\sum_{n\geq 0}\mathcal{A}(x)^{n+1}\right) = 1\,,$$

więc szereg $1-\mathcal{A}(x)$ jest formalną odwrotnością szeregu $(\mathbf{SEQ}(\mathcal{A}))(x)$ w pierścieniu szeregów formalnych.

□

W książce [1] znaleźć można wiele innych konstrukcji klas kombinatorycznych (np. operacje zbioru potęgowego, operację cykli). Jednak w tej pracy nie będziemy z nich korzystać, więc nie omawiamy ich.

Warto zaznaczyć, że pojęcie klasy kombinatorycznych należy do dziedziny zwanej kombinatoryką symboliczną. Umożliwia ona wyznaczanie funkcji tworzących opisujących rozważane obiekty kombinatoryczne za pomocą metod algebraicznych.

W dalszych częściach pracy stosować będziemy klasy kombinatoryczne do badania własności języków regularnych. W takich przypadkach w rozważanych przez nas przypadkach dla ustalonego skończonego alfabetu $\Sigma = \{a_1, \ldots, a_n\}$ bazową klasą kombinatoryczną będzie klasa $\mathcal{A} = (\{a_1, \ldots, a_n\}, |\cdot|\}$ dla której określamy $|a_n| = 1$, dla wszystkich $i \in [n]$. Inaczej mówiąc, ustalamy, że wszystkie elementy języka mają tę samą wagę równą 1. Zauważmy, że dla takiej klasy $\mathcal{A}$ mamy

$$\mathcal{A}(x) = n \cdot x$$

oraz

$$\mathbf{SEQ}(\mathcal{A})(x) = \frac{1}{1 - n \cdot x} \, .$$

2. JĘZYKI REGULARNE

W rozdziale tym omówimy krótko pojęcie wyrażenia regularnego, języka regularnego, automatu skończonego oraz związków między tymi pojęciami. Wszystkie pojęcia oraz wyniki omawiane w tym rozdziale znajdują się w pierwszych rozdziałach klasycznej książki [4].

Wyrażenia regularne nad alfabetem Σ definiowane są za pomocą następujących rekurencyjnych reguł:

1. $\emptyset$, ε (słowo puste), a (dla dowolnego $a \in \Sigma$) są wyrażeniami regularnymi,
2. jeśli r_1 oraz r_2 są wyrażeniami regularnymi, to
   - $r_1 + r_2$ (suma)
   - $r_1 r_2$ (konkatenacja)
   - r_1^* (domknięcie Kleene'ego)
   - (r_1) (grupowanie)

   są wyrażeniami regularnymi.

Z każdym wyrażeniem regularnym r związany jest język $L(r)$ reprezentowany przez r, który definiowany jest za pomocą następujących reguł:

1. $L(\emptyset) = \emptyset$
2. $L(\varepsilon) = \{\varepsilon\}$
3. $L(a) = \{a\}$ dla $a \in \Sigma$
4. $L(r_1 + r_2) = L(r_1) \cup L(r_2)$
5. $L(r_1 r_2) = \{xy : x \in L(r_1) \wedge y \in L(r_2)\}$, gdzie xy oznacza konkatenację ciągów x oraz y
6. $L(r^*) = L(r)^*$, gdzie L^* definiujemy następująco:
   $L^0 = \{\varepsilon\}$
   $L^{n+1} = \{vw : v \in L^n \wedge w \in L\}$
   $L^* = \bigcup_{n=0}^{\infty} L^n$

przy czym gwiazdka wiąże najsilniej, konkatenacja słabiej, a suma najsłabiej (przykładowo nie możemy zastosować reguły $L(aa + bb) = \{xy : x \in L(a) \wedge y \in L(a + bb)\}$, gdyż konkatenacja wiąże silniej niż suma, natomiast zachodzi $L(aa + bb) = L(aa) \cup L(bb)$). Języki postaci $L(r)$ dla wyrażania regularnego nazywamy językami regularnymi.

2.1. JĘZYKI REGULARNE I AUTOMATY SKOŃCZONE

Najczęstszym problemem dotyczącym wyrażeń regularnych jest sprawdzanie, czy dane słowo należy języka generowanego przez to wyrażenie oraz wyszukiwanie słów z tego języka w tekście. W celu zautomatyzowania tego procesu konstruuje się automaty skończone.

Definicja 2.1. Deterministycznym automatem skończonym (DFA) *nazywamy strukturę* $(\Sigma, Q, q_0, \delta, F)$*, gdzie*

1. Σ *jest skończonym zbiorem symboli nazywanym alfabetem,*
2. Q *jest skończonym zbiorem stanów automatu,*
3. $q_0 \in Q$ *jest stanem początkowym automatu,*
4. $\delta : Q \times \Sigma \to Q$ *jest funkcją przejścia, która parze* (q, a) *przypisuje stan, w jakim znajdzie się automat po przeczytaniu symbolu* a *w stanie* q
5. F *jest zbiorem stanów akceptujących (końcowych).*

Funkcja δ może być również określona częściowo. Wówczas z niektórych stanów q nie ma przejścia do kolejnego stanu po przeczytaniu pewnego symbolu a, dla którego wartość $\delta(q, a)$ nie jest określona. Mówimy, że słowo $w = a_1 \cdots a_n$ jest akceptowalne przez deterministyczny automat skończony $M = (Q, \Sigma, q_0, \delta, F)$, jeśli istnieją takie stany $q_1, \ldots, q_n \in Q$, że $\delta(q_{k-1}, a_k) = q_k$ dla $k = 1, \ldots, n$ oraz stan q_n jest akceptowalny.

Definicja 2.2. Niedeterministycznym automatem skończonym (NFA) *nazywamy piątkę* $(\Sigma, Q, q_0, \delta, F)$*, gdzie*

1. Σ *jest skończonym alfabetem,*
2. Q *jest skończonym zbiorem stanów automatu,*
3. $q_0 \in Q$ *jest stanem początkowym automatu,*
4. $\delta : Q \times \Sigma \to P(Q)$ *jest funkcją przejścia, która parze* (q, a) *przypisuje zbiór stanów, w jakich może znaleźć się automat po przeczytaniu symbolu* a *w stanie* q *(*$P(Q)$ *jest zbiorem potęgowym zbioru* Q*),*
5. F *jest zbiorem stanów akceptujących.*

Definicja 2.3. Niedeterministyczny automat skończony z ε-przejściami (ε-NFA) *definiujemy analogicznie do NFA bez* ε*-przejść z tą różnicą, że dodajemy możliwość przejścia z jednego stanu automatu do drugiego po słowie pustym, czyli* $\delta : Q \times (\{\varepsilon\} \cup \Sigma) \to P(Q)$*.*

Mówimy, że słowo $w = a_1 \cdots a_n$ jest akceptowalne przez niedeterministyczny automat skończony $M = (Q, \Sigma, q_0, \delta, F)$, jeśli istnieją takie stany $q_1, \ldots, q_n \in Q$, że $q_k \in \delta(q_{k-1}, a_k)$ dla $k = 1, \ldots, n$ oraz stan q_n jest akceptowalny.

Mówimy, że automat skończony $M = (Q, \Sigma, q_0, \delta, F)$ (deterministyczny lub niedeterministyczny) akceptuje język L, jeśli L jest zbiorem słów akceptowalnych przez M. Język akceptowany przez M oznaczamy $L(M)$.

Przedstawimy teraz kilka twierdzeń dotyczących wyrażeń regularnych i automatów skończonych.

Twierdzenie 2.1. *Niech r będzie wyrażeniem regularnym. Wtedy istnieje ε-NFA akceptujący język $L(r)$.*

Twierdzenie 2.2. *Niech L będzie zbiorem akceptowalnym przez ε-NFA. Wtedy istnieje NFA bez ε-przejść akceptujący L.*

Twierdzenie 2.3. *Niech L będzie zbiorem akceptowalnym przez NFA. Wtedy istnieje DFA akceptujący L.*

Twierdzenie 2.4. *Niech L będzie zbiorem akceptowalnym przez DFA. Wtedy L jest reprezentowany przez wyrażenie regularne.*

Z powyższych twierdzeń wynika, że wszystkie przedstawione tu pojęcia są równoważne. W szczególności dla każdego wyrażenia regularnego możemy stworzyć deterministyczny automat skończony akceptujący język generowany przez to wyrażenie.

Omówmy teraz pokrótce metodę konstrukcji DFA na podstawie danego wyrażenia regularnego. Jest ona podzielona na kilka kroków odpowiadających powyżej wymienionym twierdzeniom.

W pierwszym kroku konstruujemy ε-NFA odpowiadający danemu wyrażeniu regularnemu według następujących reguł:

1. Dla wyrażenia $\emptyset$ tworzymy automat składający się z jednego stanu nieakceptującego, a wyrażenie ε odpowiada automatowi złożonemu z jednego stanu akceptującego. Język generowany przez wyrażenie a dla $a \in \Sigma$ jest akceptowalny przez automat, który zawiera stan początkowy, który nie jest akceptujący oraz drugi stan, który jest akceptujący, przy czym istnieje jedno przejście ze stanu początkowego do drugiego stanu po symbolu a.
2. Dla wyrażenia regularnego $r = r_1 + r_2$ i automatów $M_1 = (Q_1, \Sigma_1, q_1, \delta_1, F_1)$ i $M_2 = (Q_2, \Sigma_2, q_2, \delta_2, F_2)$ takich, że $L(M_1) = L(r_1)$ i $L(M_2) = L(r_2)$, automat M akceptujący język $L(r)$ tworzymy w ten sposób, że tworzymy nowy stan nieakceptowalny q_0 i łączymy ten stan ze stanami początkowymi automatów M_1 i M_2 ε-przejściami. Otrzymujemy w ten sposób automat

$$M = (Q_1 \cup Q_2 \cup \{q_0\}, \Sigma_1 \cup \Sigma_2, q_0, \delta_1 \cup \delta_2 \cup \{((q_0, \varepsilon), \{q_1, q_2\})\}, F_1 \cup F_2) \,.$$

3. W przypadku, gdy $r = r_1 r_2$, automat M akceptujący język $L(r)$ tworzymy w ten sposób, że każdy stan akceptujący automatu M_1 łączymy ε-przejściem ze stanem akceptującym automatu M_2. Ponadto wszystkie stany akceptujące automatu M_1 zamieniamy na stany

nieakceptujące w nowym automacie M, a stanem początkowym automatu M jest stan początkowy automatu M_2. Otrzymujemy w ten sposób automat

$$M = (Q_1 \cup Q_2, \Sigma_1 \cup \Sigma_2, q_1, \delta, F_2) \ ,$$

gdzie

- $\delta(q, a) = \delta_1(q, a)$ dla $q \in Q_1 \setminus F_1$ i $a \in \Sigma_1 \cup \{\varepsilon\}$,
- $\delta(q, a) = \delta_1(q, a) \cup \{q_2\}$ dla $q \in Q_1 \cap F_1$ i $a \in \Sigma_1 \cup \{\varepsilon\}$,
- $\delta(q, a) = \delta_2(q, a)$ dla $q \in Q_2$ i $a \in \Sigma_2 \cup \{\varepsilon\}$.

4. W przypadku, gdy $r = r_1^*$, automat M akceptujący język $L(r)$ tworzymy z automatu M_1 w ten sposób, że każdy stan akceptujący automatu M_2 łączymy ε-przejściem ze stanem początkowym tego automatu oraz ustawiamy ten stan początkowy na stan akceptujący. W ten sposób otrzymujemy automat

$$M = (Q_1, \Sigma_1, q_1, \delta, F_1 \cup \{q_1\}) \ ,$$

gdzie

- $\delta(q, a) = \delta_1(q, a)$ dla $q \in Q_1 \setminus F_1$ i $a \in \Sigma_1 \cup \{\varepsilon\}$,
- $\delta(q, a) = \delta_1(q, a) \cup \{q_1\}$ dla $q \in Q_1 \cap F_1$ i $a \in \Sigma_1 \cup \{\varepsilon\}$.

Usuwanie epsilon przejść jest nieco bardziej skomplikowanym procesem. Niech $M_0 = (Q_0, \Sigma_0, q_0, \delta_0, F_0)$ będzie niedeterministycznym automatem skończonym z ε przejściami. W pierwszym kroku, w celu ograniczenia liczby stanów, możemy wyznaczyć silnie spójne składowe podgrafu zawierającego tylko ε-przejścia, a następnie złączyć ze sobą wszystkie stany należące do jednej silnie spójnej składowej, gdyż w takiej składowej z każdego stanu jesteśmy w stanie przejść do innego stanu wyłącznie po ε-przejściach. Niech $V_1, \ldots, V_n$ będą takimi silnie spójnymi składowymi w automacie M_0 i niech q'_k będzie stanem, na który zamieniamy składową V_k dla $k = 1, \ldots, n$. Niech $M_k = (Q_k, \Sigma_k, q_k, \delta_k, F_k)$ będzie automatem po zamianie składowych $V_1, \ldots, V_k$ na pojedyncze stany. Automat M_k otrzymujemy w następujący sposób:

1. $Q_k = Q_{k-1} \cup \{q'_k\} \setminus V_k$
2. $\Sigma_k = \Sigma_{k-1}$
3. $q_k = \begin{cases} q_{k-1}, & q_{k-1} \notin V_k \\ q'_k, & q_{k-1} \in V_k \end{cases}$
4. Na początek każde przejście z dowolnego stanu z V_k chcemy zamienić na przejście ze stanu q'_k. Następnie każde przejście z dowolnego stanu do stanu z V_k chcemy zamienić na przejście do stanu q'_k, za wyjątkiem ε-przejść między dwoma stanami z V_k (wówczas

stworzylibyśmy ε-przejście z q'_k do q'_k, które jest niepotrzebne). Zdefiniujmy pomocniczo

$$\begin{cases} \delta_{k-1}(q'_k, a) &= \bigcup_{q \in V_k} \delta_{k-1}(q, a), & a \in \Sigma_k \\ \delta_{k-1}(q'_k, \varepsilon) &= \left(\bigcup_{q \in V_k} \delta_{k-1}(q, \varepsilon) \right) \setminus V_k \end{cases} .$$

Dla $q \in Q_k$ i $a \in \Sigma_k \cup \varepsilon$ definiujemy

$$\delta_k(q, a) = \begin{cases} \delta_{k-1}(q, a), & \delta_{k-1}(q, a) \cap V_k = \emptyset \\ \delta_{k-1}(q, a) \cup \{q'_k\} \setminus V_k, & \delta_{k-1}(q, a) \cap V_k \neq \emptyset \end{cases} .$$

5. $F_k = \begin{cases} F_{k-1}, & F_{k-1} \cap V_k = \emptyset \\ F_{k-1} \cup \{q'_k\} \setminus V_k, & F_{k-1} \cap V_k \neq \emptyset \end{cases}$

Po tym kroku w automacie M_n nie istnieją cykle złożone z samych ε-przejść (inaczej stany należące do takiego cyklu należałyby do jednej silnie spójnej składowej). W tym momencie wyznaczamy ε-domknięcia każdego stanu, czyli zbiór stanów, do których możemy dotrzeć z danego stanu po ε-przejściach. Dla $q \in Q_n$ ε-domknięcia możemy wyznaczyć rekurencyjnie w następujący sposób

$$\varepsilon\text{-DOMKN}(q) = \begin{cases} q & \delta_n(q, \varepsilon) \subseteq \{q\} \\ q \cup \bigcup_{p \in \delta_n(q, \varepsilon)} \varepsilon\text{-DOMKN}(p), & \delta_n(q, \varepsilon) \nsubseteq \{q\} \end{cases} .$$

Dzięki temu, że w automacie M_n nie ma cykli złożonych z samych ε-przejść, przy obliczaniu ε-domknięć kolejnych stanów unikniemy zapętleń. Spróbujmy teraz NFA bez ε-przejść $\hat{M} = (\hat{Q}, \hat{\Sigma}, \hat{q}_0, \hat{\delta}, \hat{F})$. Dla $P \subseteq \hat{Q} = Q_n$ zdefiniujmy ε-DOMKN$(P) = \bigcup_{q \in P} \varepsilon$-DOMKN$(q)$. Dla $q \in \hat{Q}$ i $w \in \hat{\Sigma} = \Sigma_n$ niech $\hat{\delta}(q, w)$ oznacza zbiór stanów, w których możemy się znaleźć po przeczytaniu słowa w w stanie q. Wówczas

$$\begin{cases} \hat{\delta}(q, \varepsilon) &= \varepsilon\text{-DOMKN}(q) \\ \hat{\delta}(q, wa) &= \varepsilon\text{-DOMKN}\left(\bigcup_{p \in \hat{\delta}(q, w)} \delta_n(p, a) \right), & w \in \Sigma^*, a \in \Sigma \end{cases} .$$

W celu zbudowania NFA bez ε-przejść musimy wyznaczyć $\hat{\delta}(q, a)$ dla wszystkich $q \in \hat{Q}$ i $a \in \hat{\Sigma}$. Korzystając z powyższego wzoru otrzymujemy

$$\hat{\delta}(q, a) = \varepsilon\text{-DOMKN}\left(\bigcup_{p \in \varepsilon\text{-DOMKN}(q)} \delta_n(p, a) \right) .$$

Elementem początkowym w automacie $\hat{M}$ jest oczywiście element początkowy w automacie M_n, czyli $\hat{q}_0 = q_n$. Pozostało nam wyznaczenie stanów akceptowalnych w automacie $\hat{M}$. Do każdego stanu, do którego mogliśmy dotrzeć w automacie M_n po pewnym niepustym słowie, jesteśmy w stanie dotrzeć również w automacie $\hat{M}$. Jedynymi stanami osiągalnymi

w automacie M_n, do których nie jesteśmy w stanie dotrzeć w automacie $\hat{M}$, są te stany należące do ε-domknięcia stanu początkowego $\hat{q_0}$, do których w automacie M_n możemy się dostać jedynie po słowie pustym, za wyjątkiem stanu początkowego $\hat{q_0}$. W związku z tym, jeśli któryś z tych stanów jest akceptowalny, a stan początkowy nie jest akceptowalny w automacie $\hat{M}$, to automat $\hat{M}$ nie akceptowałby słowa pustego, a powinien je akceptować. Stąd stan początkowy $\hat{q_0}$ jest akceptujący w automacie $\hat{M}$ wtedy i tylko wtedy, gdy istnieje stan akceptujący należący do ε-DOMKN$(\hat{q_0})$. Ponadto, jeśli nie zmienimy akceptowalności pozostałych stanów, to $\hat{M}$ będzie akceptował te same słowa, które akceptuje automat M_n, gdyż do każdego stanu akceptującego, do którego możemy dojść po niepustym słowie w automacie M_n, możemy również dojść w automacie $\hat{M}$. Stąd

$$\hat{F} = \begin{cases} F_n, & F_n \cap \varepsilon\text{-DOMKN}(q_n) = \emptyset \\ F_n \cup \{q_n\}, & F_n \cap \varepsilon\text{-DOMKN}(q_n) \neq \emptyset \end{cases}.$$

Wiemy już, jak zbudować NFA na podstawie wyrażenia regularnego. Ostatnim krokiem, jaki musimy podjąć, jest zamiana NFA na DFA. Niech $M = (Q, \Sigma, q_0, \delta, F)$ będzie niedeterministycznym automatem skończonym bez ε-przejść. Deterministyczny automat skończony $M' = (Q', \Sigma', q_0', \delta', F')$ budujemy w następujący sposób:

1. $\Sigma' = \Sigma$, $q_0' = \{q_0\}$ jest stanem początkowym automatu M'.
2. Zbiór stanów Q' i funkcję przejścia δ' wyznaczamy jak poniżej.
   a) Początkowym zbiorem stanów jest $Q_0' = \{q_0'\} = \{\{q_0\}\}$. Podstawiamy $k = 0$ i dla kolejnych k wykonujemy poniższy krok, dopóki $Q_{k+1}' \neq Q_k'$.
   b) Zdefiniujmy pomocniczo $Q_{-1}' = \emptyset$. Niech P_k oznacza zbiór nieodwiedzonych stanów, czyli $P_k = Q_k' \setminus Q_{k-1}'$. Dla każdego stanu $q' \in P_k$ i każdego symbolu $a \in \Sigma$ wyznaczamy $\delta'(q', a) = \bigcup_{q \in q'} \delta(q, a)$. Jeśli $\delta'(q', a)$ nie należy do Q_k', tworzymy nowy stan $p = \delta'(q', a)$ i dodajemy go do zbioru P_{k+1}. Otrzymujemy

$$P_{k+1} = \left\{ \bigcup_{q \in q'} \delta(q, a) : q' \in Q_k' \right\} \setminus Q_k'$$

   oraz $Q_{k+1}' = Q_k' \cup P_{k+1}$.
   c) Niech n będzie największą liczbą k, dla której wykonujemy powyższy krok. Finalnym zbiorem stanów jest zbiór $Q' = Q_n'$.
3. $F' = \{q' \in Q' : F \cap q' \neq \emptyset\}$.

Niech $w = a_1 \cdots a_m$ będzie słowem akceptowalnym przez M. Pokażemy, że słowo w jest akceptowalne w M'. Niech $q_0, q_1, \ldots, q_m$ będą kolejnymi stanami, w których znajdziemy się odczytując słowo w, takimi, że $q_m \in F$. Niech $q_k' = \delta'(q_{k-1}', a_k)$ dla $k \geq 1$. Mamy

$q_0 \in q'_0$ oraz przy założeniu, że $q_{k-1} \in q'_{k-1}$, zachodzi

$$q_k \in \delta(q_{k-1}, a_k) \subseteq \bigcup_{q \in q'_{k-1}} \delta(q, a_k) = \delta'(q'_{k-1}, a_k) = q'_k \, .$$

Wobec tego $q_k \in q'_k$ dla $k = 1, \ldots, m$, w szczególności $q_m \in q'_m$, co w połączeniu z tym, że $q_m \in F$, daje $q'_m \cap F \neq \emptyset$, czyli q'_m jest stanem akceptowalnym, zatem słowo w jest akceptowalne w M'.
Przeprowadźmy dowód w drugą stronę. Niech $w = a_1 \cdots a_m$ będzie słowem akceptowalnym przez M'. Niech $q'_k = \delta'(q'_{k-1}, a_k)$ dla $k = 1, \ldots, m$. Stan q'_m jest akceptowalny, zatem musi istnieć taki stan $q_m \in F$, że $q_m \in q'_m$. Dla $k = 1, \ldots, m$ mamy

$$q'_k = \delta'(q'_{k-1}, a_k) = \bigcup_{q \in q'_{k-1}} \delta(q, a_k) \, ,$$

zatem dla każdego stanu $p \in q'_k$ istnieje stan $q \in q'_{k-1}$ taki, że $p \in \delta(q, a_k)$. Dla $k = m, m-1, \ldots, 1$ niech $q_{k-1} \in \{q'_{k-1}\}$ będzie dowolnym takim stanem, że $q_k \in \delta(q_k, a_k)$. Oznaczenie to nie jest sprzeczne z tym, że q_0 jest elementem początkowym automatu M, gdyż jest to jedyny element należący do q'_0. Wówczas $q_0, q_1, \cdots, q_m$ jest jedną z możliwych ścieżek, które możemy obrać przechodząc po kolejnych symbolach słowa w w automacie M. Ponadto stan q_m jest akceptowalny, zatem słowo w jest akceptowalne w automacie M. Wobec powyższych wniosków automaty M i M' są równoważne.

Zauważmy, że $F' \subseteq P(Q)$. W najgorszym przypadku przy zamianie NFA na DFA następuje więc wykładnicza eksplozja liczby stanów. W celu osiągnięcia jak najmniejszej możliwej liczby stanów automatu, automat ten możemy poddać minimalizacji. Algorytm minimalizacji deterministycznego automatu skończonego $M = (Q, \Sigma, q_0, \delta, F)$ przebiega następująco:

1. Będziemy oznaczać te pary stanów, które nie są równoważne i których nie możemy złączyć w jeden stan.
2. Dla każdych dwóch stanów $p \in F$ oraz $q \in Q \setminus F$ oznaczamy pary (p, q) i (q, p).
3. Oznaczmy przez L_q język słów, które byłyby akceptowane przez M, gdyby stanem początkowym tego automatu był stan q. Dwa stany p i q są równoważne wtedy i tylko wtedy, gdy $L_p = L_q$. Jeżeli istnieje słowo $w \in \Sigma^*$ takie, że po przeczytaniu słowa w w stanie p znajdziemy się w stanie akceptowalnym, a po przeczytaniu tego słowa w stanie q znajdziemy się w stanie nieakceptowalnym (lub na odwrót), to stany p i q nie są równoważne. Dla każdej pary różnych stanów $(p, q) \in (F \times F) \cup (Q \setminus F \times Q \setminus F)$ sprawdzamy, czy istnieje symbol $a \in \Sigma$ taki, że para $(\delta(p, a), \delta(q, a))$ jest oznaczona.
   a) Jeżeli nie istnieje taki symbol, oznacza to, że stany p i q są równoważne pod warunkiem, że żadna z par $(\delta(p, a), \delta(q, a))$ nie zostanie w przyszłości oznaczona. Jeśli bowiem para $(\delta(p, b), \delta(q, b))$ zostanie oznaczona dla pewnego $b \in \Sigma$, to istnieje

słowo w takie, że po przeczytaniu słowa w w stanach $\delta(p,b)$ i $\delta(q,b)$ znajdziemy się w dwóch różnych stanach, z których jeden jest akceptowalny, a drugi nie. Po przeczytaniu słowa bw w stanach p i q znaleźlibyśmy się w tych samych stanach, co po przeczytaniu słowa w w stanach $\delta(p,b)$ i $\delta(q,b)$, zatem stany p i q nie są równoważne. W celu oznaczenia pary (p,q) w momencie, w którym oznaczamy pewną parę $(\delta(p,b),\delta(q,b))$, a para (p,q) była odwiedzona wcześniej, dla każdej pary stanów (q_1,q_2) będziemy tworzyć listę par (p_1,p_2) takich, że istnieje symbol $a\in\Sigma$ taki, że $\delta(p_1,a)=q_1$ oraz $\delta(p_2,a)=q_2$, a para (p_1,p_2) była nieoznaczona w momencie dodawania jej do listy związanej z parą (q_1,q_2). W tym momencie dla każdego symbolu $a\in\Sigma$ do listy $(\delta(p,a),\delta(q,a))$ dodajemy parę (p,q), o ile $\delta(p,a)\neq\delta(q,a)$.

b) W przeciwnym razie istnieje symbol $a\in\Sigma$, dla którego para $(\delta(p,a),\delta(q,a))$ jest oznaczona i musimy oznaczyć parę (p,q). W tym momencie musimy również oznaczyć wszystkie pary (p',q') na liście (p,q), następnie wszystkie pary na listach (p',q') i tak dalej, zatem oznaczamy wszystkie te pary z kolejnych list w sposób rekurencyjny.

Po wykonaniu powyższych kroków dla wszystkich par $(p,q)\in(F\times F)\cup(Q\setminus F\times Q\setminus F)$, wszystkie pary nierównoważnych stanów są oznaczone, a pozostałe pary stanów są równoważne. Niech $P_1,\ldots,P_n$ będą zbiorami równoważnych stanów. Dla każdego z tych zbiorów wybierzmy po jednym reprezentancie $p_i\in P_i$. Nowym zbiorem stanów będzie zbiór $Q'=\{p_i : i\in[n]\}$. Niech $j(q)$ dla $q\in Q$ oznacza taką liczbę, że $q\in P_{j(q)}$. Niech $M'=(Q',\Sigma,q_0',\delta',F')$ będzie minimalnym DFA otrzymanym z M. M' otrzymujemy w ten sposób, że $q_0'=p_{j(q_0)}$, $F'=F\cap Q'$ oraz dla każdego $i\in[n]$ i każdego symbolu $a\in\Sigma$ mamy $\delta'(p_i,a)=p_{j(\delta(p_i,a))}$.

3. PROBLEM ZLICZANIA SŁÓW

W rozdziale tym będziemy się zajmowali następującym zagadnieniem: mamy ustalony język regularny R nad skończonym alfabetem Σ; dla $n \in \mathbb{N}$ definiujemy

$$r_n = card(\{x \in R : |x| = n\}) ;$$

chcemy wyznaczyć asymptotykę ciągu $(r_n)_{n\geq 0}$. W wielu sytuacjach możliwe będzie wyznaczenie zwartych formuł na elementy r_n, z których asymptotyka będzie łatwa do wyznaczenia.

Naszym głównym narzędziem będą funkcje tworzące związane z rozważanym językiem R. Są to szeregi potęgowe postaci

$$R(x) = \sum_{n=0}^{\infty} r_n x^n .$$

Funkcję tę będziemy nazywali funkcją tworzącą języka R.

Lemat 3.1. *Jeśli R jest językiem regularnym nad skończonym alfabetem Σ, to promień zbieżności r funkcji tworzącej $R(x)$ spełnia nierówność*

$$r \geq \frac{1}{card(\Sigma)} .$$

Dowód. Z kryterium Cauchy'ego zbieżności szeregów wiemy, że promień zbieżności szeregu potęgowego $R(x)$ jest równy

$$r = \frac{1}{\limsup_{n\to\infty} \sqrt[n]{|r_n|}} .$$

Ponadto, korzystając z definicji liczb r_n, otrzymujemy

$$r_n = card(\{x \in R : |x| = n\}) \leq card(\Sigma^{[n]}) = card(\Sigma)^n .$$

Stąd

$$r = \frac{1}{\limsup_{n\to\infty} \sqrt[n]{|r_n|}} \geq \frac{1}{\limsup_{n\to\infty} \sqrt[n]{card(\Sigma)^n}} = \frac{1}{card(\Sigma)} .$$

□

Z udowodnionego Lematu wynika, że funkcja tworząca języka regularnego nad skoń-

czonym alfabetem jest analityczna w pewnym otoczeniu zera. Do badania jej własności można więc stosować metody analizy matematycznej oraz zespolonej.

3.1. JEDNOZNACZNOŚĆ I WIELOZNACZNOŚĆ WYRAŻEŃ

Rozważania w tym rozdziale rozpoczniemy od rozważenia pewnego przykładu. Niech alfabet $\Sigma = \{a, b\}$. Wyrażenie regularne

$$r = (a + b)^* aa(a + b)^* \tag{3.1}$$

generuje język $L(r)$ złożony z tych skończonych ciągów $x \in \Sigma^*$, w których występuje podciąg dwóch kolejnych wystąpień litery a. Bezpośrednie przetłumaczenie tego wyrażenia regularnego na język struktur kombinatorycznych prowadzi do struktury

$$\mathcal{C} = \mathbf{SEQ}(\{a, b\}) \times \{a\} \times \{a\} \times \mathbf{SEQ}(\{a, b\}) \,,$$

której funkcja tworząca wyraża się wzorem

$$\mathcal{C}(x) = \frac{1}{1-2x} \cdot x \cdot x \cdot \frac{1}{1-2x} = \frac{x^2}{(1-2x)^2} \,,$$

a jej współczynniki wynoszą

$$[x^n]\mathcal{C}(x) = (n-1)2^{n-2}$$

dla $n > 0$.

W każdym słowie $x \in L(r)$ przed pierwszym wystąpieniem podciągu aa znajduje się pewien ciąg symboli a i b (możliwe, że pusty), w którym nie występuje podciąg aa, natomiast po pierwszym wystąpieniu podciągu aa w słowie x znajduje się dowolny ciąg znaków a i b. Język $L(r)$ możemy więc również opisać za pomocą wyrażenia regularnego

$$r' = b^*(abb^*)^* aa(a + b)^* \,. \tag{3.2}$$

Ponadto, bezpośrednie przetłumaczenie wyrażenia r' na język struktur kombinatorycznych generuje klasę

$$\mathcal{C}' = \mathbf{SEQ}(\{b\}) \times \mathbf{SEQ}(\{a\} \times \{b\} \times \mathbf{SEQ}(\{b\})) \times \{a\} \times \{a\} \times \mathbf{SEQ}(\{a, b\}) \,,$$

której funkcja tworząca wyraża się równaniem

$$\mathcal{C}'(x) = \frac{1}{1-x} \cdot \frac{1}{1-\frac{x^2}{1-x}} \cdot x \cdot x \cdot \frac{1}{1-2x} = \frac{x^2}{(1-x-x^2)(1-2x)} = \frac{1}{1-2x} - \frac{1+x}{1-x-x^2} \,,$$

z czego po przekształceniach wnioskujemy, że

$$[x^n]\mathcal{C}'(x) = 2^n - F_{n+2}\ , \tag{3.3}$$

gdzie F_k oznacza k-tą liczbę Fibonacciego. Zauważmy, że tłumaczenie wyrażeń 3.1 i 3.2 dało dwa różne wyniki, mimo, że $L(r) = L(r')$, z czego wynika, że **naiwne tłumaczenie wyrażeń regularnych na język struktur kombinatorycznych nie zawsze daje prawidłową funkcję tworzącą języka**. Warto w tym miejscu zaznaczyć, że to wzór (3.3) poprawnie zlicza rozważany język, co pokażemy w części 5.

3.1.1. Wyrażenia jednoznaczne

W celu wyjaśnienia różnicy między rozważanymi wyrażeniami wprowadźmy pojęcie jednoznaczności wyrażenia regularnego.

Wprowadźmy oznaczenie $r^n = \underbrace{r \cdots r}_{n}$ ($r^0 = \varepsilon$), gdzie r jest wyrażeniem regularnym. Mówimy, że wyrażenie regularne jest jednoznaczne, jeśli każde słowo należące do języka generowanego przez to wyrażenie możemy dopasować do tego wyrażenia tylko na jeden sposób ([1] strona 734). Równoważnie, wyrażenie regularne r nad alfabetem Σ jest jednoznaczne, gdy spełnia jeden z następujących warunków:

1. $r = \emptyset \vee r = \varepsilon \vee r = a$, gdzie $a \in \Sigma$,
2. $r = r_1 + r_2$, gdzie wyrażenia r_1 i r_2 są jednoznaczne i języki $L(r_1)$ i $L(r_2)$ są rozłączne
3. $r = r_1 r_2$, gdzie wyrażenia r_1 i r_2 są jednoznaczne oraz dla każdego słowa $x \in L(r)$ istnieje dokładnie jedna para słów y i z takich, że $y \in L(r_1)$ i $z \in L(r_2)$ oraz $yz = x$
4. $r = r_1^*$, gdzie dla każdego n wyrażenie

   $$\varepsilon + r_1 + r_1^2 + \ldots + r_1^n$$

   jest jednoznaczne (równoważnie dla każdych różnych m i n wyrażenie r_1^n jest jednoznaczne, a języki $L(r_1)^n = L(r_1^n)$ i $L(r_1)^m = L(r_1^m)$ są rozłączne).

Należy przy tym pamiętać, że gwiazdka wiąże najsilniej, konkatenacja słabiej, a suma najsłabiej.

Wróćmy do przykładu z wyrażeniami 3.1 i 3.2. Wyrażenie $r = (a+b)^*aa(a+b)^*$ nie jest jednoznaczne, gdyż słowo $x = aabaa$ możemy wyprowadzić na dwa sposoby: $x = y_1y_2y_3$, gdzie $y_1 = aab \in L((a+b)^*)$, $y_2 = aa \in L(aa)$, $y_3 = \varepsilon \in L((a+b)^*)$ lub $x = z_1z_2z_3$, gdzie $z_1 = \varepsilon \in L((a+b)^*)$, $z_2 = aa \in L(aa)$, $z_3 = baa \in L((a+b)^*)$.
Przyjrzyjmy się wyrażeniu $r' = b^*(abb^*)^*aa(a+b)^*$. Jasne jest, że wyrażenia ab i aa są jednoznaczne. Wyrażenia b^n są jednoznaczne oraz języki $L(b^n)$ i $L(b^m)$ są rozłączne dla różnych n i m, więc b^* jest wyrażeniem jednoznacznym. Podobnie wyrażenia $(a+b)^n$ są jednoznaczne (można to udowodnić indukcyjnie), a języki $L((a+b)^n)$ i $L((a+b)^m)$

są rozłączne dla różnych n i m, gdyż wszystkie słowa z tych języków mają długości odpowiednio n i m, zatem wyrażenie $(a+b)^*$ jest jednoznaczne. Dla każdego n wyrażenie $(abb^*)^n$ jest jednoznaczne, gdyż wyrażenie abb^* jest jednoznaczne, a każde słowo x należące do języka $L((abb^*)^{n+1})$ możemy jednoznacznie przedstawić jako $x = yz$, gdzie $y \in L((abb^*)^n)$, a $z \in L(abb^*)$ jest sufiksem słowa x zaczynającym się od ostatniego symbolu a w słowie x (dowód indukcyjny po n). Ponadto każde słowo z języka $L((abb^*)^n)$ zawiera dokładnie n symboli a, więc języki $L((abb^*)^n)$ i $L((abb^*)^m)$ są rozłączne dla $n \neq m$, zatem wyrażenie $(abb^*)^*$ jest jednoznaczne. Idąc dalej, nie jest trudnym udowodnić, że wyrażenia $b^*(abb^*)^*$ i $aa(a+b)^*$ są jednoznaczne. Wobec tego każde słowo $x \in L(r')$ możemy jednoznacznie przedstawić w formie $x = yz$, gdzie $y \in L(b^*(abb^*)^*)$, a $z \in L(aa(a+b)^*)$ jest sufiksem słowa x, który zaczyna się od pierwszego wystąpienia podciągu aa w słowie x, zatem finalnie wyrażenie r' jest jednoznaczne.

Twierdzenie 3.1. *Niech r będzie jednoznacznym wyrażeniem regularnym. Wówczas funkcję tworzącą języka generowanego przez wyrażenie r możemy obliczyć według następujących reguł:*

1. $L(\emptyset)(x) = 0$, $L(\varepsilon)(x) = 1$, $L(a)(x) = x$ *dla* $a \in \Sigma$,
2. *jeśli* $r = r_1 + r_2$, *to* $L(r)(x) = L(r_1)(x) + L(r_2)(x)$,
3. *jeśli* $r = r_1 r_2$, *to* $L(r)(x) = L(r_1)(x) \cdot L(r_2)(x)$,
4. *jeśli* $r = r_1^*$, *to* $L(r)(x) = \frac{1}{1-L(r_1)(x)}$.

Dowód. Przypadek 1. Równości $L(\emptyset)(x) = 0$, $L(\varepsilon)(x) = 1$ i $L(a)(x) = x$ dla $a \in \Sigma$ są oczywiste.

Przypadek 2. Jeśli $r = r_1 + r_2$ i r jest wyrażeniem jednoznacznym, to języki $L(r_1)$ i $L(r_2)$ są rozłączne, zatem

$$\begin{aligned} L(r)(x) &= L(r_1 + r_2)(x) = (L(r_1) \cup L(r_2))(x) \\ &= (L(r_1) + L(r_2))(x) = L(r_1)(x) + L(r_2)(x)\,. \end{aligned}$$

Przypadek 3. Jeśli $r = r_1 r_2$ i r jest wyrażeniem jednoznacznym, to dla każdego słowa $t \in L(r)$ istnieje dokładnie jedna para słów $(y, z) \in L(r_1) \times L(r_2)$ taka, że $t = yz$, zatem

$$\begin{aligned} L(r)(x) = L(r_1 r_2)(x) &= \{yz : y \in L(r_1) \wedge z \in L(r_2)\}(x) \\ &= \sum_{y \in L(r_1)} \sum_{z \in L(r_2)} \frac{x^{|yz|}}{|\{(y', z') \in L(r_1) \times L(r_2) : y'z' = yz\}|} \\ &= \sum_{y \in L(r_1)} \sum_{z \in L(r_2)} x^{|y|+|z|} = \left(\sum_{y \in L(r_1)} x^{|y|} \right) \left(\sum_{z \in L(r_2)} x^{|z|} \right) \\ &= L(r_1)(x) \cdot L(r_2)(x). \end{aligned}$$

Przypadek 4. Niech $r = r_1^*$. Korzystając z jednoznaczności wyrażenia r otrzymujemy

$$\begin{aligned} L(r)(x) = L(r_1^*)(x) = L(r_1)^*(x) &= \left(\bigcup_{n\in\mathbb{N}} L(r_1)^n\right)(x) = \left(\sum_{n\in\mathbb{N}} L(r_1)^n\right)(x) \\ &= (\varepsilon + L(r_1) + L(r_1)\times L(r_1) + L(r_1)\times L(r_1)\times L(r_1) + \ldots)(x) \\ &= \mathbf{SEQ}(L(r_1))(x) = \frac{1}{1 - L(r_1)(x)} \end{aligned}$$

(czwarte przejście wynika z tego, że języki $L(r_1)^n$ i $L(r_1)^m$ są rozłączne dla $n \neq m$, a kolejne przejście z tego, że wyrażenia r_1^n są jednoznaczne dla każdego $n \in \mathbb{N}$, czyli każde słowo $x \in L(r_1)^n$ ma dokładnie jeden rozkład $x = x_1 \cdots x_n$ taki, że $(x_1, \ldots, x_n) \in \underbrace{L(r_1) \times \cdots \times L(r_1)}_{n}$, a ponadto

$$|x_1 \cdots x_n| = |x_1| + \ldots + |x_n| = |(x_1, \ldots, x_n)|$$

dla $x_1, \ldots, x_n \in L(r_1)$, więc funkcja tworząca języka $L(r_1)^n$ jest równa funkcji tworzącej języka $\underbrace{L(r_1) \times \cdots \times L(r_1)}_{n}$). □

3.1.2. Podstawowe twierdzenie o zliczaniu słów

Przedstawimy teraz kluczowe twierdzenie w kontekście zliczania liczby słów ustalonej długości należących do języka regularnego. Twierdzenie to pochodzi z [1].

Twierdzenie 3.2. *Niech $M = (Q, \Sigma, q_0, \delta, F)$ będzie skończonym automatem deterministycznym, gdzie $Q = \{q_0, \ldots, q_{n-1}\}$. Funkcja tworząca języka akceptowanego przez M wyraża się wzorem*

$$L(M)(x) = \mathbf{u}(I - xT)^{-1}\mathbf{v} \,,$$

gdzie $\mathbf{u}$ jest wektorem $(1, 0, 0, \ldots, 0)$, I macierzą jednostkową, T jest macierzą o rozmiarze $n \times n$, której elementy są dane wzorem

$$T_{i,j} = card\{a \in \Sigma : q_j = \delta(q_i, a)\} \,,$$

a $\mathbf{v} = (v_0, \ldots, v_{n-1})^T$ jest wektorem kolumnowym takim, że $v_i = [\![q_i \in F]\!]$.

Dowód. Niech L_q oznacza język słów, które byłyby akceptowalne przez M, gdyby stanem początkowym był stan q. Dla każdego stanu $q \in Q$ zachodzi

$$L_q = \Delta_q \cup \bigcup_{a\in\Sigma} \left\{aw : w \in L_{\delta(q,a)}\right\} \,,$$

gdzie $\Delta_q = \{\varepsilon\}$, jeśli $q \in F$, a w przeciwnym razie $\Delta_q = \emptyset$. Zauważmy, że dla różnych $a_1, a_2 \in \Sigma$ zbiory $\left\{a_1 w : w \in L_{\delta(q,a_1)}\right\}$, $\left\{a_2 w : w \in L_{\delta(q,a_2)}\right\}$ i Δ_q są parami rozłączne. Ponadto klasy $(\left\{aw : w \in L_{\delta(q,a)}\right\}, |\cdot|)$ i $(\{a\} \times L_{\delta(q,a)}, |\cdot|)$ są izomorficzne. Wobec tego

$$L_q \cong \Delta_q + \sum_{a \in \Sigma} \{a\} \times L_{\delta(q,a)} \ .$$

Przekładając to na funkcje tworzące otrzymujemy

$$L_q(x) = [\![q \in F]\!] + x \sum_{a \in \Sigma} L_{\delta(q,a)}(x) = [\![q \in F]\!] + x \sum_{p \in Q} card\{a \in \Sigma : p = \delta(q,a)\} L_p(x) \ .$$

Po zapisaniu tego równania w postaci

$$L_{q_i}(x) = [\![q_i \in F]\!] + x \sum_{j=0}^{n-1} card\{a \in \Sigma : q_j = \delta(q_i, a)\} L_{q_j}(x) = v_i + x \sum_{j=0}^{n-1} T_{i,j} L_{q_j}(x)$$

dla $i = 1, \ldots, n-1$ i podstawieniu $\mathbf{L}(x) = (L_{q_1}(x), \ldots, L_{q_{n-1}}(x))^T$ otrzymujemy

$$\mathbf{L}(x) = \mathbf{v} + xT\,\mathbf{L}(x),$$

co po przekształceniach daje $\mathbf{L}(x) = (I - xT)^{-1}\mathbf{v}$. Interesuje nas funkcja tworząca języka $L(M) = L_{q_0}$, zatem finalnie

$$L(M)(x) = L_{q_0}(x) = \mathbf{u} \cdot \mathbf{L}(x) = \mathbf{u}(I - xT)^{-1}\mathbf{v} \ .$$

□

Zauważmy, że przy obliczaniu funkcji tworzącej języka akceptowanego przez DFA wykonujemy tylko podstawowe operacje elementarne. Konsekwencją tej obserwacji jest poniższe twierdzenie.

Twierdzenie 3.3. *Funkcja tworząca dowolnego języka regularnego jest funkcją wymierną, to jest funkcją dającą się przedstawić w postaci $\frac{P(x)}{Q(x)}$, gdzie $P(x)$ i $Q(x)$ są wielomianami zmiennej x. Ponadto, jeśli 0 nie jest pierwiastkiem wielomianu $P(x)$, to nie jest też pierwiastkiem wielomianu $Q(x)$.*

Dowód. Z każdym językiem regularnym związany jest pewien deterministyczny automat skończony M. Funkcję tworzącą tego języka możemy obliczyć ze wzoru

$$L(M)(x) = \mathbf{u}(I - xT)\mathbf{v}$$

przy oznaczeniach jak w twierdzeniu 3.2. Wszystkie elementy wektorów $\mathbf{u}$, $\mathbf{v}$ oraz macierzy I i T możemy traktować jako funkcje wymierne. Podczas wyznaczania macierzy odwrotnej

oraz obliczania iloczynu dwóch macierzy wykonujemy wyłącznie operacje dodawania, odejmowania, mnożenia i dzielenia na elementach macierzy. Klasa funkcji wymiernych jest zamknięta na te operacje, zatem wynikiem tych działań jest funkcja wymierna.
Przyjrzyjmy się teraz drugiej części tego twierdzenia. Niech L będzie językiem regularnym i $L(x) = \frac{P(x)}{Q(x)}$. Gdyby 0 było pierwiastkiem wielomianu $Q(x)$, ale nie byłoby pierwiastkiem wielomianu $P(x)$, to wartość $L(0)$ nie byłaby określona, jednak $L(0) = \sum_{k=0}^{\infty} l_n 0^n = 0$. Wobec tego, jeśli wielomiany $P(x)$ i $Q(x)$ są względnie pierwsze (nie istnieje wielomian stopnia większego od 0 dzielący oba te wielomiany), to 0 nie jest pierwiastkiem wielomianu $Q(x)$. □

Twierdzenie 3.4. *Każda funkcja wymierna $\frac{P(x)}{Q(x)}$ posiada rozkład na ułamki proste*

$$\frac{P(x)}{Q(x)} = R(x) + \sum_{i=1}^{m} \sum_{j=1}^{a_i} \frac{A_{i,j}}{(x_i - x)^j} ,$$

gdzie $Q(x)$ ma m różnych pierwiastków zespolonych x_i o krotności a_i, $A_{i,j}$ są stałymi, a $R(x)$ jest wielomianem stopnia $\deg P(x) - \deg Q(x)$, jeśli $\deg P(x) \geq \deg Q(x)$, a zerem w przeciwnym razie.

Skorzystajmy z powyższych twierdzeń do wyznaczenia zwartego wzoru na funkcję tworzącą języka regularnego. Niech L będzie językiem regularnym oraz

$$L(x) = \frac{P(x)}{Q(x)} = R(x) + \sum_{i=1}^{m} \sum_{j=1}^{a_i} \frac{B_{i,j}}{(1 - b_{i,j}x)^j}$$

(taką postać $L(x)$ otrzymujemy po podstawieniu $b_{i,j} = x_i^{-j}$ i $B_{i,j} = A_{i,j} b_{i,j}$ we wzorze z powyższego twierdzenia, możemy założyć, że 0 nie jest pierwiastkiem wielomianu $Q(x)$). Wówczas, korzystając ze wzoru (1.1), otrzymujemy, że liczba słów długości n należących do tego języka wynosi

$$\begin{aligned} l_n = [x^n]L(x) &= [x^n]R(x) + \sum_{i=1}^{m} \sum_{j=1}^{a_i} [x^n] \frac{B_{i,j}}{(1 - b_{i,j}x)^j} \\ &= [x^n]R(x) + \sum_{i=1}^{m} \sum_{j=1}^{a_i} \binom{n+j-1}{n} B_{i,j} b_{i,j}^n . \end{aligned} \tag{3.4}$$

W dalszych rozdziałach wykorzystamy powyższe obserwacje do zliczania słów ustalonej długości należących do języków regularnych generowanych przez wybrane wyrażenia regularne.

We wzorze (3.4) może się zdarzyć, że liczba m pierwiastków wielomianu $Q(x)$ jest równa zeru. Wtedy funkcja L jest wielomianem, więc ciąg liczb $[x^n]L(x)$ ma wielomianowe tempo wzrostu. W przeciwnym przypadku, gdy $m > 0$ we wzorze (3.4) pojawiają się

czynniki postaci

$$\frac{A}{(c-x)^k}\ .$$

Ze uogólnionego wzoru dwumianowego Newtona wynika, że odpowiadają one za wykładniczy wzrost ciągu liczb $[x^n]L(x)$. Z powyższych rozważań wynika następująca ciekawa obserwacja:

Twierdzenie 3.5. *Niech R będzie językiem regularnym nad skończonym alfabetem Σ. Niech*

$$r_n = |R \cap \Sigma^n|\ .$$

Wtedy możliwe są tylko dwa przypadki:

1. *istnieje wielomian $w(x)$ taki, że $r_n \leq w(n)$ dla każdego n*
2. *istnieje stała $c > 0$ taka, że $r_n \geq c^n$ dla każdego n*

4. ALGORYTMY

W ramach pracy dyplomowej zbudowano zestaw narzędzi, które można wykorzystać do wyznaczenia dla danego wyrażenia regularnego r funkcji wymiernej, której współczynniki rozwinięcia w szereg potęgowy o środku w punkcie 0 wyznaczają liczbę słów rozważanego języka o danym rozmiarze.

Proces ten składa się z następujących etapów:

1. przekształcenie zadanego wyrażania regularnego w skończony niedeterministyczny automat skończony z epsilon przejściami
2. przekształcenie automatu niedeterministycznego z epsilon przejściami w automat niedeterministyczny bez epsilon przejść
3. przekształcenie automatu niedeterministycznego bez epsilon przejść w deterministyczny automat skończony
4. minimalizacja deterministycznego automatu skończonego
5. wyznaczanie funkcji tworzącej z automatu skończonego

W pierwszej kolejności wywoływana jest metoda statyczna klasy NFA

```
NFA* NFA::regexToAutomaton(std::string regex);
```

która jako argument przyjmuje ciąg znaków podany przez użytkownika opisujący wyrażenie regularne, gdzie każdy znak różny od znaków (,), + i * jest interpretowany jako symbol alfabetu, a zwraca wskaźnik na obiekt klasy `NFA` reprezentujący stan początkowy automatu równoważnego językowi regularnemu generowanemu przez podane wyrażenie regularne (wcześniej sprawdzana jest poprawność podanego wyrażenia regularnego, a w przypadku podania błędnego wyrażenia metoda zwraca `nullptr`). Następnie na obiekcie wskazywanym przez zwracany wskaźnik wywoływana jest metoda

```
NFA* NFA::removeEpsilonTransitions();
```

która usuwa z automatu wszystkie ε-przejścia i zwraca wskaźnik na obiekt reprezentujący stan początkowy otrzymanego automatu. Kolejną wywoływaną funkcją jest metoda obiektu, którego wskaźnik jest zwracany przez poprzednią metodę, mianowicie jest to metoda

```
DFA* NFA::toDFA();
```

która wyznacza deterministyczny automat skończony równoważny z otrzymanym wcześniej niedeterministycznym automatem skończonym i zwraca wskaźnik na obiekt klasy `DFA` reprezentujący stan początkowy tego automatu. Zanim obliczana jest funkcja tworząca języka akceptowanego przez otrzymany automat, automat ten podlega minimalizacji za pomocą metody

```
DFA* DFA::minimize()
```

W celu uzyskania funkcji tworzącej języka akceptowanego przez ten automat, na obiekcie wskazywanym przez zwrócony wskaźnik wywoływana jest metoda

```
RationalFunction<Rational<integer> > DFA::getGeneratingFunction();
```

która zwraca obiekt klasy `RationalFunction<Rational<integer> >` reprezentujący oczekiwaną funkcję tworzącą w postaci funkcji wymiernej $\frac{P(x)}{Q(x)}$, gdzie $P(x)$ i $Q(x)$ są względnie pierwszymi wielomianami o współczynnikach wymiernych (integer jest tu zdefiniowany jako typ `mpz_class` z zewnętrznej biblioteki GMP). Metoda ta wywołuje metodę statyczną klasy `MatrixInversion<T>`

```
std::vector<std::vector<T> >
↪  MatrixInversion<T>::gaussianElimination(std::vector<std::vector<T> >
↪  a);
```

dla typu T będącego klasą `RationalFunction<Rational<integer> >`. Jest to funkcja wyznaczająca macierz odwrotną do macierzy podanej jako argument. Obliczenia wykonywane podczas wyznaczania tej macierzy obejmują dodawanie, odejmowanie, mnożenie i dzielenie funkcji wymiernych zapisanych jako obiekty klasy `RationalFunction<Rational<integer> >`, które z kolei wymagają obliczeń na wielomianach reprezentowanych przez obiekty klasy `Polynomial<Rational<integer> >`. Współczynnikami tych wielomianów są liczby wymierne, z którymi związana jest klasa `Rational<integer>`. Oprócz prostych operacji elementarnych wykonywanych na wielomianach i liczbach wymiernych, program wykorzystuje metody statyczne

```
T Operators<T>::gcd(T a, T b);
Polynomial<T> Polynomial<T>::gcd(Polynomial<T> a, Polynomial<T> b);
```

do obliczania największego wspólnego dzielnika liczb całkowitych i wielomianów w celu znalezienia postaci nieskracalnej funkcji wymiernej. Otrzymana funkcja tworząca jest na koniec przekształcana do postaci $R(x) + \frac{\hat{P}(x)}{\hat{Q}(x)}$, gdzie $R(x)$ jest wielomianem, a $\hat{P}(x)$ i $\hat{Q}(x)$ są wielomianami takimi, że stopień wielomianu $\hat{P}(x)$ jest mniejszy od stopnia wielomianu $\hat{Q}(x)$, a wielomian $\hat{Q}(x)$ jest przedstawiony w postaci iloczynu pewnych potęg wielomianów, które nie posiadają pierwiastków wymiernych lub mają dokładnie jeden pierwiastek, który jest wymierny. W tym celu tworzony jest obiekt klasy `ExtendedRationalFunction<integer>`, którego konstruktor przyjmuje jako argument obiekt klasy `RationalFunction<Rational<integer> >` i używa metody wyznaczającej wymierne pierwiastki wielomianu $Q(x)$, wykorzystującej twierdzenie o wymiernych pierwiastkach wielomianu o współczynnikach całkowitych. Ostatecznie, funkcja tworząca wyznaczona dla wyrażenia regularnego r jest wynikiem instrukcji

```
ExtendedRationalFunction(
    NFA::regexToAutomaton(r)
    ->removeEpsilonTransitions()
    ->toDFA()
    ->minimize()
    ->getGeneratingFunction()
);
```

Szczegółowe działanie powyższych algorytmów zostało opisane w poprzednich rozdziałach. Wszystkie zrealizowane kody znajdują się w skompresowanym pliku praca_dyplomowa.zip. Opisy modułów znajdujących się w tym pliku omówione są w Dodatku A.

5. BADANIA WYBRANYCH WYRAŻEŃ REGULARNYCH

W rozdziale tym wykorzystamy zaimplementowane narzędzia do wyznaczenia funkcji tworzących języków regularnych zadanych przez wybrane wyrażenia regularne.

5.1. KLASYCZNE PRZYKŁADY

5.1.1. Słowa zawierające "aa"

Wróćmy na chwilę do przykładu z rozdziału 3. Po wprowadzeniu przez użytkownika wyrażenia regularnego $r = (a+b)^*aa(a+b)^*$ program zwraca funkcję tworzącą $\frac{x^2}{(1-2x)(1-x-x^2)}$. Zauważmy, że

$$\frac{x^2}{(1-2x)(1-x-x^2)} = \frac{1}{1-2x} - \frac{x+1}{1-x-x^2}$$

Przypomnijmy, że $\frac{1}{1-2x} = \sum_{n=0}^{\infty} 2^n x^n$. Mamy również (patrz wzór (1.2))

$$\frac{x}{1-x-x^2} = \sum_{n=0}^{\infty} F_n x^n \ ,$$

gdzie F_n oznacza n-tą liczbę Fibonacciego. Zatem

$$\frac{x+1}{1-x-x^2} = \frac{x}{1-x-x^2} + \frac{1}{1-x-x^2} =$$
$$\sum_{n=0}^{\infty} F_n x^n + \sum_{n=0}^{\infty} F_{n+1} x^n =$$
$$\sum_{n=0}^{\infty} (F_n + F_{n+1}) x^n = \sum_{n=0}^{\infty} F_{n+2} x^n$$

Zatem, liczba słów długości n języka nad $\{a, b\}$ zawierających podciąg aa wyraża się wzorem

$$r_n = 2^n - F_{n+2} \ .$$

Otrzymaliśmy więc wzór (3.3) o którym wspomnieliśmy w Rozdziale 3.

5.1.2. Słowa zawierające "aa" i "bb"

Rozważmy język słów złożonych z liter a i b, które zawierają co najmniej jedno wystąpienie podciągu aa i co najmniej jedno wystąpienie słowa bb. Wyrażeniem regularnym opisującym ten język jest wyrażenie

$$r_1 = (a+b)^*(aa(a+b)^*bb + bb(a+b)^*aa)(a+b)^* ,$$

jednak nie jest to wyrażenie jednoznaczne. W celu znalezienia jednoznacznego wyrażenia regularnego opisującego ten język rozpatrzmy osobno słowa, w których pierwsze wystąpienie podciągu aa znajduje się przed pierwszym wystąpieniem podciągu bb oraz słowa, w których pierwsze wystąpienie podciągu aa znajduje się za pierwszym wystąpieniem podciągu bb. Otrzymujemy w ten sposób wyrażenie

$$r_2 = (\varepsilon + b)(ab)^*aaa^*(baa^*)^*bb(a+b)^* + (\varepsilon + a)(ba)^*bbb^*(abb^*)^*aa(a+b)^* .$$

Można pokazać, że wyrażenie to jest jednoznaczne. Stąd funkcja tworząca tego języka wynosi

$$\begin{aligned} L(r_1)(x) = L(r_2)(x) &= \frac{2(1+x)x^4}{(1-x^2)(1-x)(1-2x)(1-\frac{x^2}{1-x})} \\ &= \frac{2x^4}{(1-x)(1-2x)(1-x-x^2)} = \frac{1}{1-2x} - \frac{2(x+1)}{1-x-x^2} + \frac{2}{1-x} - 1 , \end{aligned}$$

zatem liczba ciągów liter a i b długości n zawierająca podciągi aa i bb wynosi

$$l_n = 2^n - 2F_{n+2} + 2 - [\![n = 0]\!] .$$

Funkcja tworząca wyznaczona przez zaimplementowany program jest identyczna.

Zauważmy, że im więcej jest podciągów, które musi zawierać rozważany język, tym dłuższe jest wyrażenie regularne opisujące ten język, musimy bowiem rozważyć wszystkie kolejności, w jakich występują pierwsze wystąpienia takich podciągów. Liczba różnych kolejności jest równa liczbie permutacji zbioru złożonego z tych podciągów, zatem wyrażenie regularne rośnie bardzo szybko wraz ze zwiększaniem liczby podciągów, co powoduje duży wzrost liczby stanów skończonego automatu deterministycznego związanego z tym wyrażeniem regularnym oraz znaczne zwiększenie czasu oczekiwania wynik zwrócony przez zaimplementowany program. Wówczas lepiej jest zastanowić się nad wyrażeniem jednoznacznym opisującym taki język i przetłumaczyć to wyrażenie bezpośrednio na funkcję tworzącą.

5.1.3. Słowa o ustalonym znaku na danej pozycji

Rozważmy wyrażenie regularne

$$r_k = (a+b)^* a (a+b)^{k-1} ,$$

które opisuje wszystkie ciągi liter a i b, w których k-ty symbol od końca jest literą a. Jest to znany przykład wyrażeń, dla których klasyczna konstrukcja automatu skończonego zawiera tzw. eksplozję wykładniczą. Jednakże problem zliczania dla tego języka jest bardzo prosty, gdyż wystarczy zauważyć, że liczba słów tego języka długości n jest równa liczbie słów długości n języka generowanego przez wyrażenie

$$s_k = (a+b)^{k-1} a (a+b)^* ,$$

która jest równa 0 dla $n < k$ oraz 2^{n-1} dla $n \geq k$. Dla $k = 11$ NFA bez ε-przejść, wygenerowany za pomocą zaimplementowanego programu, dla pierwszego wyrażenia zawiera 36 stanów, a dla drugiego 32 stany, natomiast DFA otrzymany przez przekształcenie NFA w DFA dla pierwszego wyrażenia zawiera aż 2049 stanów (po minimalizacji 2048), natomiast dla drugiego wyrażenia ma ich zaledwie 12. Wniosek, który możemy wysnuć z tych obserwacji, jest taki, że w przypadku, gdy chcemy wyznaczyć liczbę słów z języka danego przez wyrażenie regularne, a DFA wygenerowany na podstawie tego wyrażenia ma dużą liczbę stanów, warto zastanowić się nad innym izomorficznym językiem, w przypadku którego DFA akceptujący ten język może zawierać znacznie mniejszą liczbę stanów.

Warto również odnotować, że funkcja tworząca języka opisanego przez to wyrażenie regularne jest równa $f(x) = \frac{2^{k-1}x^k}{1-2x}$, a zapisana w postaci kanonicznej $f(x) = R(x) + \frac{P(x)}{Q(x)}$, gdzie $\deg P(x) < \deg Q(x)$, jest dużo mniej czytelna. Stąd warto czasem pozostawić funkcję tworzącą w postaci $\frac{P(x)}{Q(x)}$ w celu jaśniejszej interpretacji.

5.2. SIŁA HASEŁ

Hasła są standardowym rozwiązaniem służącym do ograniczania dostępu do systemów informatycznych do uprawnionych użytkowników. Z powodu powszechności ich występowania i obecności wielu systemów użytkownicy serwisów mają tendencję do stosowania prostych do przewidzenia haseł, co ułatwia stosowanie ataków na takie zabezpieczenia za pomocą metod słownikowych lub technik typu "brute force" (systematycznego przeszukiwania wszystkich możliwych haseł). Jedną z metod stosowanych przez projektantów systemów informatycznych do uniemożliwiania stosowania stosunkowo prostych do złamania prostych haseł polega na stawianiu różnego rodzaju ograniczeń na możliwe do stosowania hasła.

W rozdziale tym zajmiemy się zbadaniem jednego, często stosowanego, typu ograniczeń. Załóżmy, że nasze hasła mogą się składać z dowolnych małych i wielkich liter alfabetu

łacińskiego (razem z 52 liter), 10 cyfr oraz 10 powszechnie stosowanych znaków specjalnych. Łącznie nasze hasła mogą zawierać 72 różne znaki. Nałóżmy na możliwe do zastosowania hasła następujące ograniczenie:

> W haśle występować ma przynajmniej jedna duża litera, po której występuje przynajmniej jedna mała litera, po której występuje przynajmniej jedna cyfra, po której wystąpić ma przynajmniej jeden znak specjalny.

Jest do dosyć silne ograniczenie, gdyż oprócz wymuszenia stosowania dużych liter, małych liter, cyfr i znaków specjalnych narzucona jest w nim kolejność występowania.

Niech Σ oznacza zbiór wszystkich rozważanych znaków. Mamy $|\Sigma| = 72$. Niech r_1, r_2, r_3 i r_4 będą wyrażeniami regularnymi przedstawionymi w postaci sumy odpowiednio wszystkich małych liter, wielkich liter, cyfr i rozważanych znaków specjalnych (na przykład $r_1 = a + b + \ldots + z$). Następujące wyrażenie regularne jest bezpośrednim tłumaczeniem rozważanego ograniczenia:

$$r = \Sigma^* r_1 \Sigma^* r_2 \Sigma^* r_3 \Sigma^* r_4 \Sigma^*$$

Po wykorzystaniu zbudowanego w pracy narzędzia wyrażenie to generuje następującą funkcję tworzącą

$$f(z) = \frac{67600z^4}{(1-72z)(1-62z)^2(1-46z)^2} .$$

Oto rozkład otrzymanej funkcji na ułamki proste:

$$f(z) = -\frac{325}{1472(1-46z)^2} + \frac{8619}{7936(1-62z)} - \frac{845}{1984(1-62z)^2} + \frac{1}{1-72z} - \frac{8475}{5888(1-46z)} .$$

Otrzymany rozkład umożliwia otrzymanie za pomocą wzoru (1.1) zwartych formuł na liczbę słów rozważanego języka ustalonej długości. Po serii żmudnych, acz elementarnych, przekształceń otrzymujemy następujący wzór:

$$r_n = 72^n - \frac{1}{182528}(62^n(77740n - 120497) + 46^n(40300n + 303025))$$

Liczba $r_n - 72^n$ jest liczbą wszystkich ciągów rozważanego języka o długości n które są niedopuszczalne przez wprowadzone ograniczenia. Z powyższego wzoru tego wynika, że

$$72^n - r_n \approx \frac{845}{1984} \cdot n \cdot 62^n \approx 0.5 \cdot n \cdot 62^n .$$

Poniższa tabela zawiera wartości liczb r_n dla $n = 1, \ldots, 8$ oraz przybliżone wartości liczb 72^n.

n	r_n	72^n
1	0	72
2	0	5184
3	0	373248
4	67600	2.68739×10^7
5	1.94688×10^7	1.93492×10^9
6	3.38162×10^9	1.39314×10^{11}
7	4.59172×10^{11}	1.00306×10^{13}
8	5.37093×10^{13}	7.22204×10^{14}

Z powyższej tabeli wynika, że **bez wprowadzenia dodatkowego ograniczenia na długość stosowanych haseł w rozważanym scenariuszu dojść może do wyraźnego osłabienia bezpieczeństwa systemu**. Mianowicie, jeśli pozwolimy na wprowadzanie haseł długości 4 (a można się spodziewać, że użytkownicy będą przejawiali tendencję do stosowania krótkich haseł), to liczba dopuszczalnych przez system haseł zredukowana będzie do około 70 000.

Z powyższych rozważań wynika następujące zalecenie dla projektantów systemu wprowadzających ograniczenia na dopuszczalne hasła:

> *Jeśli narzucasz silne ograniczenia na strukturę dopuszczalnych haseł, to wśród tych ograniczeń powinno być również ograniczenie na minimalną długość dopuszczalnego hasła.*

Jednym z narzędzi służących do analizy wpływu długości dopuszczalnych haseł na bezpieczeństwo implementowanego systemu jest system zbudowany w ramach tej pracy dyplomowej.

6. PODSUMOWANIE

Zaimplementowane narzędzia omówione w tej pracy pozwalają efektywnie wyznaczyć funkcję tworzącą języka generowanego przez wyrażenie regularne. Mając daną tę funkcję, przy użyciu różnych pakietów matematycznych (np. Mathematica, Wolfram Alpha, Maxima, Matlab, Scilab), można następnie wyznaczyć wzór na liczbę słów długości n należących do tego rozpatrywanego języka, który następnie można zastosować dla wielu wartości n.

Powyższe narzędzia mogą być szczególnie przydatne przy badaniu wpływu ograniczeń narzuconych na hasła przez wyrażenie regularne na podatność na złamanie systemu bezpieczeństwa. Innym potencjalnym zastosowaniem tego narzędzia może być wyznaczanie minimalnej liczby bitów, na których można zapisać słowa ograniczonej długości należące do języka opisanego przez dane wyrażenie regularne, a następnie próba kompresji tych słów. Zagadnienie to nie było rozważane w tej pracy.

Zrealizowany w ramach tej pracy program można rozszerzyć o funkcje wyznaczanie pierwiastków mianownika otrzymanej funkcji tworzącej, a następnie zamienienie tej funkcji na sumę ułamków prostych, dzięki czemu byłoby możliwe automatyczne (bez użycia innych systemów) wyznaczenie współczynników w rozwinięciu tej funkcji w szereg potęgowy, a tym samym wyznaczenie liczby słów o zadanej długości należących do rozważanego języka regularnego. Inną kwestią, którą można poruszyć w dalszych rozważaniach, jest zbadanie złożoności obliczeniowej zaimplementowanych algorytmów oraz podjęcie próby ich zoptymalizowania.

W trakcie badań przeprowadzonych w trakcie realizacji pracy natrafiliśmy na ciekawy, naszym zdaniem, problem:

> Załóżmy, że znamy funkcje tworzące $R_1(x)$ i $R_2(x)$ dwóch języków regularnych L_1 oraz L_2; czy istnieje nietrywialna zależność algebraiczna między $R_1(x)$, $R_2(x)$ oraz funkcją tworzącą przekroju $L_1 \cap L_2$?

Zauważmy, że produkt dwóch języków regularnych jest językiem regularnym. Operacja przekroju wyrażeń nie jest traktowana jako operacja elementarna na wyrażeniach regularnych. Jednak jest ona pożyteczna, jest łatwa do implementacji w algorytmach służących do rozpoznawania zgodności łańcucha ze wzorcem oraz konstrukcja automatu rozpoznającego $L_1 \cap L_2$ z automatów rozpoznających L_1 i L_2 jest bardzo prosta.

BIBLIOGRAFIA

[1] Flajolet, P., Sedgewick, R., *Analytic Combinatorics* (Cambridge University Press, 2008).

[2] Graham, R.L., Knuth, D.E., Patashnik, O., *Concrete Mathematics: A Foundation for Computer Science* (Addison-Wesley, Reading, 1989).

[3] Granlund, T., Team, G.D., *GNU MP 6.0 Multiple Precision Arithmetic Library* (Samurai Media Limited, London, GBR, 2015).

[4] Hopcroft, J.E., Ullman, J.D., *Wprowadzenie do teorii automatów, języków i obliczeń* (Wydawnictwo Naukowe PWN, 2008).

[5] Meyers, S., *Effective Modern C++: 42 Specific Ways to Improve Your Use of C++11 and C++14* (O'Reilly, 2014).

Dodatki

A. SPIS ZAWARTOŚCI DOŁĄCZONEGO PLIKU ZIP

A.1. KODY ŹRÓDŁOWE

W dołączonym pliku praca_dyplomowa.zip zostały zamieszczone kody źródłowe programu wyznaczającego funkcję tworzącą języka regularnego generowanego przez dane wyrażenie regularne. Program został napisany w języku C++11 z wykorzystaniem biblioteki GNU Multiple Precision Arithmetic Library do operacji na dużych liczbach. Poniżej znajduje się spis plików wraz z ich opisem.

A.1.1. NFA.h, NFA.cpp

W plikach NFA.h i NFA.cpp znajduje się deklaracja klasy NFA wraz z definicjami jej metod. Jest to klasa niedeterministycznych automatów skończonych. Każdy obiekt reprezentuje stan automatu, jego pola przechowują informacje, czy ten stan jest akceptowalny i jakie przejścia wychodzą z tego stanu. Funkcja przejścia δ jest zrealizowana za pomocą uporządkowanych map. W każdym obiekcie związanym z jakimś stanem $q \in Q$ przechowywana jest mapa, której kluczami są symbole $a \in \Sigma$, a wartością przypisaną do klucza a jest zbiór stanów $\delta(a, q)$. W tej implementacji symbolem jest wartość typu char, a znak '\0' jest traktowany jako ε-przejście. Poniżej opisane są najistotniejsze metody tej klasy.

1. `NFA* regexToAutomaton(std::string regex)` - dla danego wyrażenia regularnego zwraca niedeterministyczny automat skończony związany z tym wyrażeniem. Metoda ta wykorzystuje wcześniej omówione algorytmy przekształcania wyrażenia regularnego w ε-NFA.
2. `NFA* removeEpsilonTransitions()` - usuwa wszystkie ε-przejścia z niedeterministycznego automatu skończonego (zamienia ε-NFA na NFA) i zwraca ten automat.
3. `DFA* toDFA()` - przekształca niedeterministyczny automat skończony w deterministyczny automat skończony.

A.1.2. DFA.h, DFA.cpp

W plikach DFA.h i DFA.cpp umieszczona jest deklaracja klasy DFA oraz definicje jej metod. Jest to klasa deterministycznych automatów skończonych. Każdy obiekt tej klasy reprezentuje stan DFA, są w nim przechowywane informacje o tym, czy ten stan jest

akceptowalny oraz opisane są wszystkie przejścia, które wychodzą z tego stanu. Funkcja przejścia tego automatu jest częściowo określona, to znaczy mogą istnieć taki stan $q \in Q$ i taki symbol $a \in \Sigma$, że dla pary (q, a) nie jest określony nowy stan $\delta(q, a)$. Funkcja przejścia δ jest zrealizowana za pomocą uporządkowanych map - dla każdego stanu q obiekt związany z tym stanem zawiera mapę, której kluczami są symbole a, po których możemy przejść do jakiegoś stanu, a wartością dla klucza a jest ten stan, czyli $\delta(q, a)$. Poniżej przedstawione są najważniejsze metody tej klasy.

1. `RationalFunction<Rational<integer> > getGeneratingFunction()` - oblicza funkcję tworzącą języka regularnego związanego z deterministycznym automatem skończonym.
2. `bool regexMatch(const std::string& word)` - dla danego słowa sprawdza, czy to słowo należy do języka regularnego, który jest rozpoznawany przez deterministyczny automat skończony.
3. `DFA* minimize()` - minimalizuje deterministyczny automat skończony według lekko zmodyfikowanego algorytmu opisanego w rozdziale 2.

A.1.3. RationalFunction.h

W pliku RationalFunction.h znajduje się szablon klasy RationalFunction. Jest to klasa funkcji wymiernych w postaci $\frac{P(x)}{Q(x)}$, gdzie $P(x)$ i $Q(x)$ są względnie pierwszymi wielomianami nad zadanym typem oraz $Q(x) \neq 0$. Zaimplementowane zostały podstawowe operacje na funkcjach wymiernych, to jest dodawanie, odejmowanie, mnożenie i dzielenie oraz skracanie ułamka do postaci nieskracalnej.

A.1.4. Polynomial.h

Plik Polynomial.h zawiera szablon klasy Polynomial, która reprezentuje wielomian nad zadanym typem. W tym szablonie zaimplementowane są podstawowe operacje na wielomianach, w tym dodawanie, odejmowanie, mnożenie i dzielenie z resztą. Ponadto zaimplementowany jest algorytm wyznaczający największy wspólny dzielnik dwóch wielomianów (wielomian o największym stopniu, który dzieli oba wielomiany).

A.1.5. Rational.h

Plik Rational.h zawiera szablon klasy Rational, która zawiera implementację ułamków liczb. Implementacja zawiera podstawowe operacje na ułamkach, które zwracają ułamek w postaci nieskracalnej.

A.1.6. ExtendedRationalFunction.h

W pliku ExtendedRationalFunction.h znajduje się szablon klasy ExtendedRationalFunction zawierającej implementację funkcji wymiernych w postaci $R(x) + \frac{P(x)}{Q(x)}$, gdzie

$R(x)$, $P(x)$ i $Q(x)$ są wielomianami nad ułamkami liczb całkowitych zadanego typu, stopień wielomianu $P(x)$ jest mniejszy od stopnia wielomianu $Q(x)$, a wielomian $Q(x)$ jest iloczynem wielomianów, które posiadają nie więcej niż jeden pierwiastek wymierny. W implementacji tego szablonu znajduje się funkcja znajdująca wymierne pierwiastki wielomianu $Q(x)$.

A.1.7. MatrixInversion.h

W pliku MatrixInversion.h znajduje się szablon klas MatrixInversion, w którym zaimplementowana jest metoda obliczająca odwrotność macierzy nieosobliwej o elementach zadanego typu, wykorzystująca metodę eliminacji Gaussa.

A.1.8. Operators.h

Szablon klasy Operators zawiera implementacje algorytmu Euklidesa wyznaczającego największy wspólny dzielnik dwóch liczb całkowitych oraz implementację algorytmu wyznaczającego rozkład liczby całkowitej na czynniki pierwsze.

A.1.9. PtrMap.h

Szablon klasy PtrMap zawiera implementację pomocniczej mapy, której wartości są wskaźnikami na obiekty danej klasy. W momencie wstawiania do mapy nowej pary klucz-wartość tworzony jest nowy obiekt danej klasy, na który wskazuje wstawiany wskaźnik.

A.1.10. ZeroInversionException.h

W pliku tym znajduje się klasa wyjątku, który jest zwracany, gdy następuje dzielenie przez 0.

A.1.11. main.cpp

Jest to plik testowy, użytkownik podaje wyrażenie regularne, a program wyznacza deterministyczny automat skończony związany z tym wyrażeniem i podaje funkcję tworzącą języka generowanego przez to wyrażenie.

A.1.12. Makefile

Plik do kompilacji programu (kompilacja odbywa się po wpisaniu komendy make).