Uniwersytet Warszawski
Wydział Matematyki, Informatyki i Mechaniki

Jakub Piotrowicz
Nr albumu: 429170

Wstęp do Teorii Automatów

błędy (nawet literówki) / uwagi / spostrzeżenia proszę zgłaszać mailowo na adres: jp429170@students.mimuw.edu.pl

Praca licencjacka
na kierunku MATEMATYKA

Praca wykonana pod kierunkiem
prof. UW Michał Skrzypczak
Instytut Informatyki

Warszawa, 29 Marca 2023

Streszczenie

W pracy przedstawiono wybrane modele automatów oraz odpowiadające im klasy języków wraz z kluczowymi własnościami. Poza tematem przewodnim przedstawione zostały również dodatki uzupełniające mających na celu wyjaśnienie bardziej zaawansowanych technik bądź tematów pobocznych. Praca została napisana z zamysłem posługi osobom o różnym stopniu zaznajomienia z dziedziną, nie zakłada więc znajomości pojęć ani faktów uważanych powszechnie za nieoczywiste lub wykraczające znacząco poza poziom szkoły średniej oraz pierwszego roku studiów informatycznych bądź matematycznych.

Słowa kluczowe

automat skończony, wyrażenie regularne, język regularny, gramatyka bezkontekstowa, automat ze stosem, deterministyczny automat ze stosem, maszyna Turinga, funkcja obliczalna, redukcja obliczalna, funkcja częściowo obliczalna, problem nierozstrzygalny

Dziedzina pracy (kody wg programu Socrates-Erasmus)

11.0 Matematyka, Informatyka

Klasyfikacja tematyczna

68Q45 – Formal languages and automata

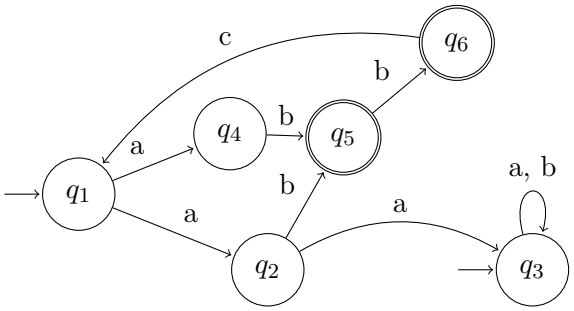
Tytuł pracy w języku angielskim

Introduction to Automata Theory

Spis treści

Wprowadzenie

Teoria automatów jest dziedziną informatyki teoretycznej zapoczątkowaną w połowie XX wieku jako część matematycznej teorii systemów. Głównym obszarem jej badań są abstrakcyjne maszyny wykorzystywane w celu modelowania obliczeń. Automat to proste urządzenie oddziałujące na bodźce zgodnie z ustalonymi regułami. Na ogół liczba różnych bodźców oraz stanów takiego urządzenia jest z góry ograniczona, w wyniku czego dostajemy skończoną reprezentację struktur z reguły nieskończonych – zwanych językami.



Rysunek powyżej przedstawia przykładowy automat. Składa się on ze skończonego zbioru stanów (q_i) oznaczonych kołami oraz skończonego zbioru tranzycji oznaczonych strzałkami. Każda strzałka ma etykietę – symbol (literę), po której następuje dana tranzycja. Gdy dozwolone jest przejście od jednego stanu do drugiego różnymi literami, używamy skrótowej notacji, zapisując je po przecinku, aby utrzymać przejrzystość rysunku. Co więcej wyróżniamy dwa zbiory stanów – początkowe oraz akceptujące. Reprezentantów pierwszego z nich oznaczamy strzałką wchodzącą, natomiast elementy drugiego charakteryzują się podwojoną obwódką.

Taką maszynę możemy postrzegać jako grę, wybieramy słowo $w = a_1 a_2 \dots a_n$ i stan początkowy, a następnie próbujemy idąc tranzycjami o kolejnych etykietach $a_1, a_2, \dots, a_n$ dojść do stanu akceptującego – jeśli nam się uda (istnieje stan początkowy i ścieżka), to słowo jest akceptowane, czyli należy do języka naszego automatu. Zwróćmy uwagę, że opisywany język może być nieskończony (posiadać nieskończenie wiele słów), pomimo tego, że opis akceptującego go narzędzia jest skończony.

Wróćmy jeszcze na chwilę do powyższego przykładu. Jaki język rozpoznaje ten automat? Szukając ścieżki ze stanu q_1 do q_6, rozpoznamy słowa postaci $(abbc)^*abb$, czyli takie, które posiadają jakąś liczbę (być może zero) powtórzeń słowa $abbc$, a potem jeszcze słowo abb. Wędrując natomiast do q_5, rozpoznamy słowa postaci $(abbc)^*ab$. Pozostaje zaobserwować, że zaczynając w q_3, nie da się osiągnąć ani q_5 ani q_6. Rozpoznawany język jest więc wyrażany jako $(abbc)^*ab(\varepsilon + b)$, w szczególności posiada słowo $(abbc)^k ab$ dla każdego $k \geqslant 0$, jest więc zbiorem nieskończonym (posiada nieskończenie wiele słów), pomimo tego, że powyższy automat jest jego skończoną reprezentacją. Zanim zagłębimy się w szczegóły, przyswoimy kilka pojęć.

Rozdział 1

Podstawowe pojęcia

W tym rozdziale wprowadzimy standardowe obiekty będące przedmiotem badań teorii automatów. Obiekty te, jakkolwiek matematycznie proste, są opisywane z użyciem specyficznego dla dziedziny żargonu. Stąd, warto poświęcić chwilę na oswojenie z terminologią.

1.1. Definicje

1. Oznaczenie $\mathbb{N}$ (zbiór liczb naturalnych) rozumiemy jako $\{0, 1, 2, \ldots\}$.

2. Oznaczenie $X \rightharpoonup Y$ rozumiemy jako funkcję częściową z X do Y

3. Przez **alfabet** rozumiemy dowolny zbiór, którego elementy nazywane są **literami** bądź **symbolami**. Zazwyczaj zakładamy, że jest *skończony* i *niepusty*. Standardowe oznaczenie to Σ. Jeśli alfabet jest jednoliterowy, to nazywamy go **alfabetem unarnym**.

4. **Słowo** nad alfabetem Σ to skończony ciąg liter z tego alfabetu, dla przykładu: 0101011 jest słowem nad alfabetem $\{0, 1\}$ oraz $ababa$ jest słowem nad alfabetem $\{a, b, c\}$.

5. Jeśli słowo w ma długość n, to piszemy $|w| = n$. Mówimy też, że słowo w ma n pozycji numerowanych od 1 do n. Jeśli $1 \leqslant i \leqslant |w|$, to przez $w[i]$ oznaczamy i-tą literę słowa w. Podsłowo zaczynające się na pozycji i, a kończące się na pozycji j oznaczamy $w[i..j]$. Jest tylko jedno słowo długości 0, nazywane słowem pustym i oznaczone symbolem ε. Przykładowo:
$$w = abcd,\ w[2] = b,\ w[1..3] = abc.$$

6. Dla słowa w długości n oraz słowa v mówimy, że:

   - v jest **prefiksem** w, jeśli albo $v = w[1..k]$ dla pewnego $k \leqslant n$, albo $v = \varepsilon$,
   - v jest **infiksem** w, jeśli $v = w[i..j]$ dla pewnych $1 \leqslant i \leqslant j \leqslant n$, albo $v = \varepsilon$,
   - v jest **sufiksem** w, jeśli $v = w[k..n]$ dla pewnego $k \leqslant n$, albo $v = \varepsilon$.

7. Przez Σ^* oznaczamy zbiór wszystkich słów nad alfabetem Σ.

8. **Językiem** nad alfabetem Σ nazywamy dowolny podzbiór $L \subseteq \Sigma^*$. Język zna dodatkowo swój alfabet, to znaczy, że język $\{aa\}$ nad alfabetem $\{a\}$ to inny język niż $\{aa\}$ nad alfabetem $\{a, b\}$.

9. **Odwróceniem słowa** w nazywamy słowo w^R spełniające warunki:

$$\left|w^R\right| = |w|,\ \forall 1 \leqslant i \leqslant n. w[i] = w^R[n-i+1],\ \text{gdzie } n = |w|.$$

Przykładowe odwrócenia słów:

$$(aba)^R = aba,\ (0011)^R = 1100,\ (abc)^R = cba.$$

10. **Odwróceniem języka** L nazywamy język L^R zdefiniowany jako:

$$L^R = \{w^R \mid w \in L\}.$$

Przykładowe odwrócenie języka:

$$L = \{abb, ba, a\},\ L^R = \{bba, ab, a\}$$

11. **Konkatenacja słów** $u = a_1 \dots a_k$ oraz $v = b_1 \dots b_l$ to słowo $a_1 \dots a_k b_1 \dots b_l$, oznaczane $u \cdot v$ lub uv. Zwróćmy uwagę, że operacja ta nie jest na ogół przemienna, czyli $uv \neq vu$. Dla słów $u = abaa,\ v = cb$ przykładami konkatenacji są: $u \cdot v = abaacb, v \cdot u = cbabaa$.

12. Przez **k-tą potęgę słowa** w, oznaczaną jako w^k, rozumiemy słowo powstałe z k-krotnej konkatenacji tego słowa, na przykład $(abc)^3 = abcabcabc,\ (add)^0 = \varepsilon,\ a^4 = aaaa$.

13. Dla każdej litery $a \in \Sigma$ definiujemy funkcję $\#_a \colon \Sigma^* \to \mathbb{N}$ zliczającą liczbę liter „a" w danym słowie, na przykład:

$$\#_a(abb) = 1,\ \#_b(abb) = 2, \#_c(abb) = 0.$$

14. Dla języków L i K definiujemy następujące operacje:

(a) $L + K \coloneqq L \cup K$ (**suma języków**)

(b) $LK = L \cdot K \coloneqq \{v \cdot w \mid v \in L, w \in K\}$ (**iloczyn języków**)

(c) $L^* \coloneqq \bigcup_{n \geqslant 0} L^n$, gdzie $L^n = L \cdot L^{n-1}$ dla $n \geqslant 1$, $L^0 = \{\varepsilon\}$ (**domknięcie Kleenego**)

Przykładami tych operacji są:

$$\begin{aligned}
\varnothing^* &= \{\varepsilon\}, \\
(\{a\} + \{b\})^* &= \{a, b\}^*, \\
\{a, b\} \cdot \{a, b\}^* &= \{w \mid w \neq \varepsilon\}, \\
\{ab\} \cdot \{w \mid k \in \mathbb{N},\ |w| = 2^k\} &= \{w \mid k \in \mathbb{N},\ |w| = 2^k + 2, w[1] = a, w[2] = b\}.
\end{aligned}$$

15. **Wyrażeniami regularnymi** *(ang. regular expression)* nad alfabetem Σ są: $\varnothing$, ε, $a \in \Sigma$. Ponadto jeśli $\mathcal{E}$, $\mathcal{F}$ są wyrażeniami regularnymi to również $\mathcal{E}^*$, $\mathcal{E}\mathcal{F}$, $\mathcal{E} + \mathcal{F}$ są wyrażeniami regularnymi. Wszystkie wyrażenia regularne są tej postaci. Warto zaznaczyć, że wyrażenia regularne są formalnymi obiektami, powstającymi z wymienionych symboli za pomocą opisanych reguł.

W naturalny sposób wyrażeniu $\mathcal{E}$ przypisujemy język oznaczany $L(\mathcal{E})$, mówimy wtedy, że $\mathcal{E}$ generuje (lub rozpoznaje) język $L(\mathcal{E})$. Dla wygody używamy następujących skrótów:

- pomijamy operację L utożsamiając wyrażenie regularne z jego językiem,

- używamy nawiasów, by określić kolejność i zakres stosowania odpowiednich reguł, na przykład $(a+b)(ab+\varepsilon)$ to inne wyrażenie regularne niż $a+bab+\varepsilon$.

Przykładami wyrażeń regularnych oraz przypisywanych im języków nad $\Sigma = \{a, b\}$ są:

$$(a+b)^* \mapsto \Sigma^*,$$
$$a(aa)^* \mapsto \{a^{2k+1} \mid k \in \mathbb{N}\},$$
$$(b^*abb^*)^*b^* \mapsto \{w \mid \text{w słowie } w \text{ bezpośrednio po każdej literze } a \text{ występuje litera } b\}.$$

Często używanym skrótem notacyjnym jest + w indeksie górnym, oznacza on „co najmniej jedno powtórzenie słowa", a definiowany jest poprzez: $w^+ = ww^*$.

Przykładowe użycie plusa:

$$(ab)^+ = ab(ab)^*.$$

Do zrozumienia niektórych sformułowań przydatna może okazać się wiedza z dodatków umieszczonych na końcu skryptu. Zachęcam do przeczytania ich dopiero w momencie, w którym materiał objaśniony tam, a nieznany czytelnikowi, pojawi się w aktualnie czytanym fragmencie pracy.

Rozdział 2

Języki regularne

W tym rozdziale skupimy swoją uwagę na wybranej klasie języków, którą nazywamy językami regularnymi. Na początku poznamy definicje, następnie podstawowe własności, aby na końcu przejść do algorytmów oraz najróżniejszych reprezentacji – korzystających między innymi z logiki oraz algebry.

Łatwo się domyślić, że niektóre języki są bardziej skomplikowane, a inne mniej. Na przykład język słów parzystej długości $\{w \mid |w| \equiv_2 0\}$ intuicyjnie wydaje się dużo prostszy od języka $\{a^p \mid p \text{ jest liczbą pierwszą}\}$. Będziemy próbowali sformalizować tę intuicję, aby nauczyć się rozpoznawać języki stosunkowo „proste".

Najpierw przyjrzyjmy się innym przykładom języków:

- skończony zbiór słów, np. $\{a, ab, aac\}$,
- język pusty, czyli $\varnothing$,
- język składający się tylko ze słowa pustego $\{\varepsilon\}$, jest to język istotnie różny od języka pustego, ma on dokładnie jeden element, a nie zero,
- język pełny nad ustalonym alfabetem Σ, czyli Σ^*,
- język palindromów, to znaczy $\{w \mid \forall 1 \leqslant i \leqslant n.w[i] = w[n-i+1], \text{ gdzie } n = |w|\}$,
- język słów nieparzystej długości nad alfabetem unarnym $a(aa)^*$,
- język zawierający wszystkie niepuste słowa postaci $ab \dots ab$, czyli $(ab)^+ = ab(ab)^*$,
- język słów zawierających literę b, czyli $\Sigma^* b \Sigma^*$,
- język słów które mają tyle samo liter a i b, czyli $\big\{w \mid \#_a(w) = \#_b(w)\big\}$.

2.1. Definicje

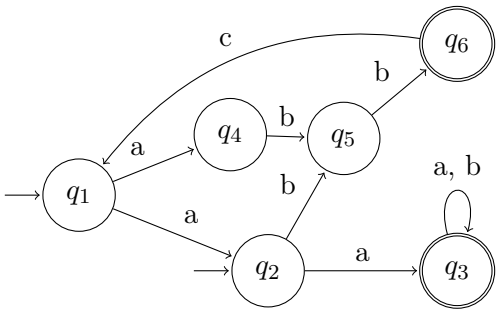
W całym rozdziale, jeżeli alfabet nie został podany w sposób jawny, to przyjmujemy, że jest ustalony, oznaczany przez Σ, niepusty oraz skończony.

Język L nazywamy **językiem regularnym** jeśli jest on generowany przez pewne wyrażenie regularne, to znaczy, że istnieje takie wyrażenie regularne $\mathcal{E}$, że zachodzi $L = L(\mathcal{E})$.

Klasa języków regularnych zdefiniowana poprzez wyrażenia regularne wydaje się dosyć przyjazna w obyciu. Łatwo jest na przykład zauważyć, że jest ona zamknięta na sumy, ponieważ jeśli mamy języki $L_1 = L(\mathcal{E}_1), L_2 = L(\mathcal{E}_2)$, to wyrażenie $\mathcal{E}_1 + \mathcal{E}_2$ generuje język $L_1 \cup L_2$.

Niedużym wysiłkiem można zauważyć, że wszystkie języki skończone należą do tej klasy, wystarczy bowiem skonstruować wyrażenie będące sumą skończonej liczby języków jednoelementowych. Okazuje się, że klasa ta jest zamknięta również na inne operacje – na przykład na przecięcie i dopełnienie. Aby to pokazać będziemy potrzebować kilku nowych pojęć, a także równoważnych definicji omawianej klasy języków.

Automat niedeterministyczny *(ang. NFA – nondeterministic finite automaton)* to model prostego urządzenia mającego skończoną liczbę stanów, reagującego na bodźce w postaci liter ustalonego alfabetu Σ. Będąc w danym stanie, otrzymując daną literę, automat wykonuje tranzycję do jakiegoś stanu (być może tego samego). Wyróżniamy ponadto dwa zbiory stanów (niekoniecznie rozłączne): stany początkowe oraz stany akceptujące. Automaty możemy przedstawiać za pomocą diagramów, na których koła reprezentują stany, a skierowane krawędzie tranzycje:



Formalnie jest to piątka $\langle \Sigma, Q, I, F, \delta \rangle$, gdzie Σ to skończony alfabet, Q to skończony zbiór stanów, $I \subseteq Q$ to zbiór stanów początkowych *(ang. initial)*, $F \subseteq Q$ to zbiór stanów końcowych *(ang. final / accepting)*, a $\delta \subseteq Q \times \Sigma \times Q$ to relacja tranzycji. Należenie $(p, a, q) \in \delta$ oznacza, że istnieje tranzycja:

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

- **Biegiem** po słowie w w automacie $\mathcal{A}$ nazywamy ścieżkę w diagramie automatu $\mathcal{A}$ zaczynającą się w jednym ze stanów początkowych, która składa się z tranzycji po kolejnych literach słowa w. Co więcej, mówimy, że jest on **biegiem akceptującym**, jeśli kończy się w stanie akceptującym. Czasami biegiem nazywamy też ciąg stanów, przez które bieg przechodzi. Formalizując, bieg po słowie w długości n to ciąg $n+1$ stanów $q_0, q_1, \ldots, q_n$, taki, że $q_0 \in I$ oraz dla $1 \leqslant i \leqslant n$ zachodzi $(q_{i-1}, w[i], q_i) \in \delta$. Bieg nazywamy akceptującym jeśli $q_n \in F$.

- **Językiem automatu** $\mathcal{A}$, oznaczanym przez $L(\mathcal{A})$, nazywamy zbiór wszystkich słów skończonych $\omega \in \Sigma^*$, po których $\mathcal{A}$ ma co najmniej jeden bieg akceptujący.

Automat deterministyczny *(ang. DFA – deterministic finite automaton)* to szczególny przypadek NFA o dwóch dodatkowych warunkach:

(1) $|I| = 1$, wymóg ten sprawia, że zacząć bieg możemy tylko w jeden sposób, ponieważ istnieje tylko jeden stan początkowy. Jest to naturalne wymaganie determinizmu.

(2) Relacja możliwych tranzycji δ jest **funkcją częściową** $\delta : (Q \times \Sigma) \rightharpoonup Q$. Oznacza to, że dla każdego stanu i każdej litery istnieje co najwyżej jedna tranzycja z tego stanu po tej literze.

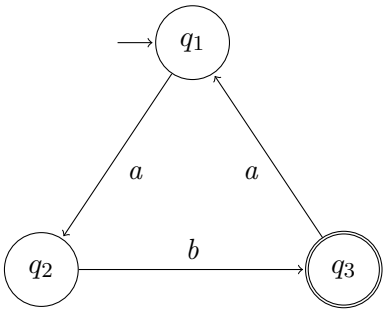
Jeżeli funkcja częściowa δ automatu deterministycznego $\mathcal{A}$ jest zwykłą funkcją (tzn. z każdego stanu wychodzi dokładnie Σ tranzycji po jednej dla każdej litery), to mówimy, że automat $\mathcal{A}$ jest **zupełny**.

Dla zupełnych DFA możemy zdefiniować funkcję δ nie tylko na elementach Σ, ale też dla słów $w \in \Sigma^*$, dostając $\delta(q, w) = p$, gdzie $w \in \Sigma^*$ oraz p to stan który otrzymamy idąc z q tranzycjami po kolejnych literach słowa w (przejście to zawsze istnieje i jest jednoznaczne). Warto odnotować, że DFA może mieć liczbę stanów akceptujących różną od 1, lecz stan początkowy jest unikalny.

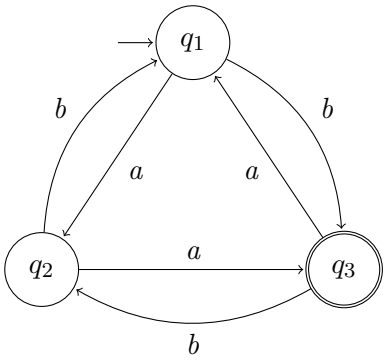
Aby lepiej zrozumieć naturę automatów deterministycznych, przyjrzyjmy się dwóm przykładom DFA – zupełnego oraz niezupełnego nad alfabetem $\Sigma = \{a, b\}$.

Przykład 2.1. *Dobrym ćwiczeniem może być próba wywnioskowania jakie języki rozpoznają te automaty. Czy te języki są regularne? Jeśli tak, to jakie są ich wyrażenia regularne?*

DFA:



zupełny DFA:



2.2. Równoważność NFA, DFA oraz wyrażeń regularnych

W tej sekcji udowodnimy, że klasy języków rozpoznawanych przez automaty niedeterministyczne (NFA), automaty deterministyczne (DFA) oraz klasa języków generowanych przez wyrażenia regularne są równe. W konsekwencji da nam to aż trzy równoważne definicje klasy języków regularnych, co znacząco ułatwi badanie jej własności.

Zanim przejdziemy do głównego wyniku tego podrozdziału, wprowadzimy nowy rodzaj automatów, zwany automatami z ε-przejściami.

Automat z ε-przejściami od NFA różni się tym, że na tranzycjach poza literami alfabetu Σ może posiadać również słowo puste ε. Słowo danego biegu takiego automatu, tak jak w zwykłym automacie, jest wynikiem konkatenacji kolejnych etykiet tranzycji tego biegu.

Uwaga 2.1. *Warto zwrócić uwagę na fakt, że w NFA bieg o $n+1$ stanach zawsze idzie po słowie długości n, dla automatów z ε-przejściami nie jest to prawda, ponieważ przejście po ε oznacza konkatenacje ze słowem pustym.*

Udowodnimy teraz twierdzenie, które pozwoli nam stosować automaty z ε-przejściami w miejscach, w których docelowo będziemy chcieli posiadać NFA. Zabieg ten często jest wygodny przy dowodzeniu twierdzeń, ze względu na fakt, że ε-przejścia pomagają w łatwy i przejrzysty sposób modyfikować automaty w celu uzyskania konkretnych właściwości.

Twierdzenie 2.1. *Niech $L \subseteq \Sigma^*$ będzie językiem. Następujące warunki są równoważne:*

1. *istnieje NFA $\mathcal{A}$ taki, że $L(\mathcal{A}) = L$,*
2. *istnieje automat z ε-przejściami $\mathcal{B}$ taki, że $L(\mathcal{B}) = L$.*

Dowód. Pokażemy równoważność dowodząc dwóch implikacji. Oczywiście jeśli istnieje automat niedeterministyczny $\mathcal{A}$ rozpoznający język L, to automatem z ε-przejściami rozpoznającym L jest również $\mathcal{A}$, ponieważ automaty z ε-przejściami są ogólniejszym modelem niż automaty niedeterministyczne. Przejdźmy do dowodu implikacji (2) $\implies$ (1).

Wykonujemy następujące kroki:

1. dodajemy tranzycję (p, a, q) dla każdej trójki $(p, a, q) \in Q \times \Sigma \times Q$ dla której w oryginalnym automacie istnieje ścieżka po $\varepsilon^* a \varepsilon^*$ z p do q,

2. dodajemy do zbioru stanów początkowych wszystkie stany do których istnieje ε-ścieżka z jakiegoś początkowego,

3. dodajemy do stanów akceptujących wszystkie stany z których istnieje ε-ścieżka do jakiegoś stanu końcowego,

4. kasujemy wszystkie ε przejścia.

■

Twierdzenie 2.2. *Dla każdego języka $L \subseteq \Sigma^*$ następujące warunki są równoważne:*

(1) istnieje NFA $\mathcal{A}$ taki, że $L(\mathcal{A}) = L$,

(2) istnieje zupełny DFA $\mathcal{B}$ taki, że $L(\mathcal{B}) = L$,

(3) istnieje wyrażenie regularne $\mathcal{E}$ takie, że $L(\mathcal{E}) = L$.

Dowód. Pokażemy dwie równoważności: (1) $\iff$ (2) oraz (1) $\iff$ (3), poprzez wykazanie czterech implikacji.

- Implikacja (2) $\implies$ (1) jest oczywista, ponieważ DFA to szczególny przypadek NFA.

- Przejdźmy do (1) $\implies$ (2). Niech $\mathcal{A} = \langle \Sigma, Q, I, F, \delta \rangle$. Skonstruujemy tak zwany automat potęgowy $\mathcal{B}$, w którym stanami będą podzbiory stanów automatu $\mathcal{A}$. Zdefiniujmy:

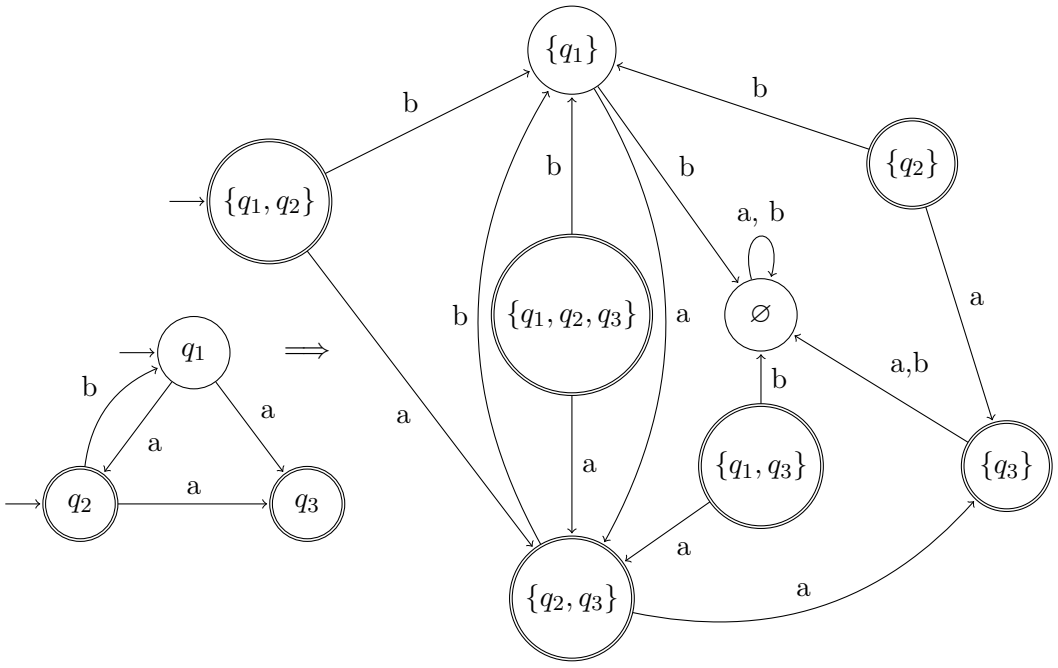
$$\mathcal{B} = \langle \Sigma, Q_{\mathcal{B}}, I_{\mathcal{B}}, F_{\mathcal{B}}, \delta_{\mathcal{B}} \rangle,$$

gdzie:

$$Q_{\mathcal{B}} = P(Q), \quad I_{\mathcal{B}} = \{I\}, \quad F_{\mathcal{B}} = \{X \mid X \subseteq Q, X \cap F \neq \varnothing\},$$

$$\delta_{\mathcal{B}}(X, a) = \{y \mid \exists x \in X. \exists t \in \delta. t = (x, a, y)\}.$$

Poniższy rysunek przedstawia przykładową konstrukcję automatu potęgowego.



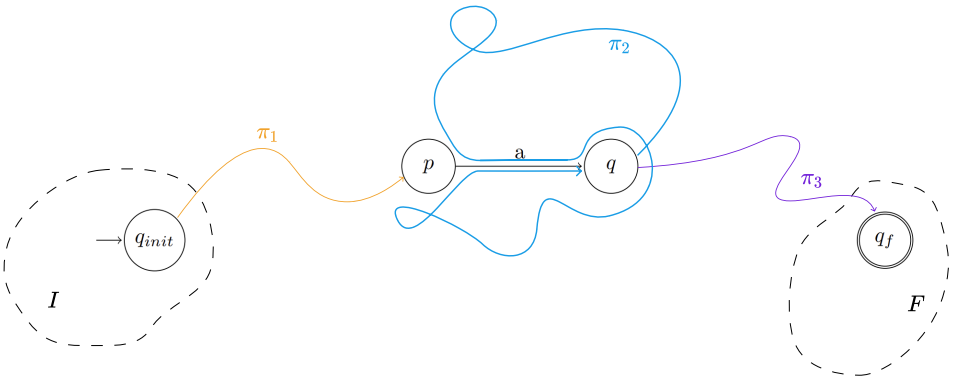
Wprost z definicji jest to zupełny DFA, ponieważ $|I_{\mathcal{B}}| = 1$ oraz $\delta_{\mathcal{B}}$ jest funkcją. Wystarczy więc pokazać, że akceptuje te i tylko te słowa, dla których istnieje bieg akceptujący w $\mathcal{A}$. Zauważmy jednak, że jeśli po słowie w istnieje bieg akceptujący w $\mathcal{A}$, to w $\mathcal{B}$ możemy w kolejnych zbiorach będących stanami na ścieżce $\delta_{\mathcal{B}}(I, w)$ wskazać stany z biegu akceptującego w w $\mathcal{A}$, więc stan $\delta_B(I, w)$ na pewno spełnia $\delta_B(I, w) \cap F \neq \varnothing$, czyli $\mathcal{B}$ akceptuje słowo w. Implikacja odwrotna również zachodzi. Jeśli $\delta_{\mathcal{B}}(I, w) \in F_B$, to wystarczy wybrać stan akceptujący automatu $\mathcal{A}$ zawarty w $\delta_{\mathcal{B}}(I, w)$, a następnie odtwarzać „idąc w tył" dowolną ścieżkę do tego stanu. Wprost z definicji automatu jest to możliwe, a gdy dojdziemy do początku znajdziemy się w jakimś elemencie I, więc $\mathcal{A}$ akceptuje słowo w. Uzyskaliśmy więc $\mathrm{L}(\mathcal{B}) = \mathrm{L}(\mathcal{A})$.

Otrzymujemy więc równoważność (1) $\iff$ (2).

- Udowodnijmy teraz implikację (1) $\implies$ (3).

  Ustalmy alfabet Σ. Przeprowadzimy dowód indukcyjny ze względu na $|\delta|$ – ilości tranzycji. Baza indukcyjna: $\delta = \varnothing$, język automatu jest więc równy $\varnothing$ lub $\{\varepsilon\}$, wyrażenia dla tych przypadków to odpowiednio: $\varnothing, \varepsilon$. Wystarczy więc wykonać krok indukcyjny. Załóżmy, że dla każdego automatu niedeterministycznego $\mathcal{A}$ o co najwyżej n tranzycjach potrafimy skonstruować wyrażenie regularne rozpoznające $L(\mathcal{A})$. Należy więc pokazać, że dla dowolnego automatu posiadającego dokładnie $n+1$ tranzycji również jesteśmy w stanie wskazać takie wyrażenie.

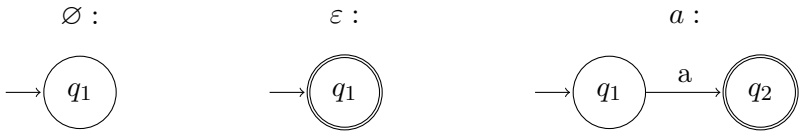
  Ustalmy NFA $\mathcal{A}$ o $n+1$ tranzycjach oraz dowolną tranzycję $t = (p, a, q) \in \delta_{\mathcal{A}}$. Zauważmy, że każdy bieg akceptujący w tym automacie albo (przypadek 1) nie używa tranzycji t, albo (przypadek 2) używa jej $m \geqslant 1$ razy i jest postaci $\pi_1\pi_2\pi_3$, gdzie π_1 jest fragmentem biegu do pierwszego użycia t, natomiast π_3 jest fragmentem po ostatnim użyciu t.



Ponadto π_1 idzie z jakiegoś stanu $q_{init} \in I$ do p, a π_3 idzie z q do jakiegoś stanu akceptującego $q_f \in F$. Dla biegów, które nie używają t możemy z założenia indukcyjnego dostać wyrażenia generujące równoważne języki – niech więc $\mathcal{E}$ będzie wyrażeniem rozpoznającym język automatu $\mathcal{A}$ z usuniętą tranzycją t (załatwia przypadek 1). Niech $\mathcal{E}_1, \mathcal{E}_3$ będą wyrażeniami rozpoznającymi języki $\mathcal{A}$ bez t z odpowiednio zmodyfikowanymi zbiorami I, F – dla $\mathcal{E}_1$ zostawiamy I, a zmieniamy $F = \{p\}$, natomiast dla $\mathcal{E}_3$ zmieniamy $I = \{q\}$, a F pozostawiamy bez zmian. Zdefiniujmy $\mathcal{E}_2$ jako wyrażenie dla $\mathcal{A}$ bez tranzycji t, z $I = \{q\}, F = \{p\}$. Otrzymujemy, że biegi z przypadku 2 akceptują dokładnie słowa postaci $\mathcal{E}_1 a(\mathcal{E}_2 a)^{m-1}\mathcal{E}_3$, więc w ogólności po zsumowaniu obu przypadków $\mathcal{E} + \mathcal{E}_1 a(\mathcal{E}_2 a)^* \mathcal{E}_3$ jest szukanym wyrażeniem, co kończy dowód.

- Pozostała implikacja (3) $\implies$ (1).

  W tym wypadku również przeprowadzimy indukcję, jednak po strukturze wyrażenia. Ustalmy alfabet Σ. Zauważmy, że dla wyrażeń $\varnothing, \varepsilon, a \in \Sigma$ wystarczy rozważyć odpowiednie automaty:



  Bazę indukcyjną udowodniliśmy, pozostaje więc dla ustalonych wyrażeń regularnych $\mathcal{E}, \mathcal{F}$ oraz automatów $\mathcal{A}_\mathcal{E}, \mathcal{A}_\mathcal{F}$ spełniających $L(\mathcal{E}) = L(\mathcal{A}_\mathcal{E}), L(\mathcal{F}) = L(\mathcal{A}_\mathcal{F})$ pokazać konstrukcję automatów rozpoznających języki wyrażeń: $\mathcal{E}^*, \mathcal{E}\mathcal{F}$ oraz $\mathcal{E} + \mathcal{F}$.

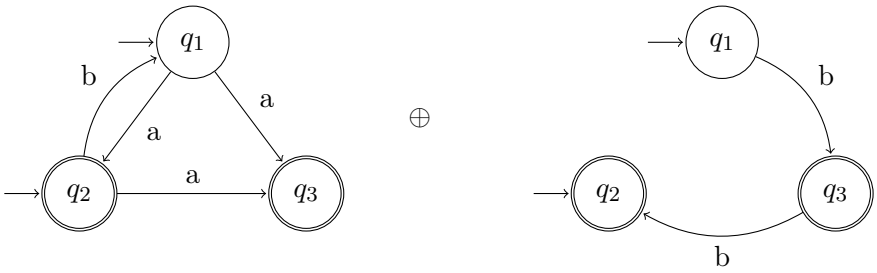
  Wyrażenie $\mathcal{E} + \mathcal{F}$ – należy rozważyć automat składający się z dwóch automatów $\mathcal{A}_\mathcal{E}, \mathcal{A}_\mathcal{F}$ narysowanych „jeden obok drugiego” (suma rozłączna), będzie on niedeterministycznym automatem rozpoznającym sumę języków. Przedstawmy konstrukcję:

$$\mathcal{A}_\mathcal{E} = \langle Q_\mathcal{E}, \Sigma, I_\mathcal{E}, F_\mathcal{E}, \delta_\mathcal{E} \rangle, \qquad \mathcal{A}_\mathcal{F} = \langle Q_\mathcal{F}, \Sigma, I_\mathcal{F}, F_\mathcal{F}, \delta_\mathcal{F} \rangle$$

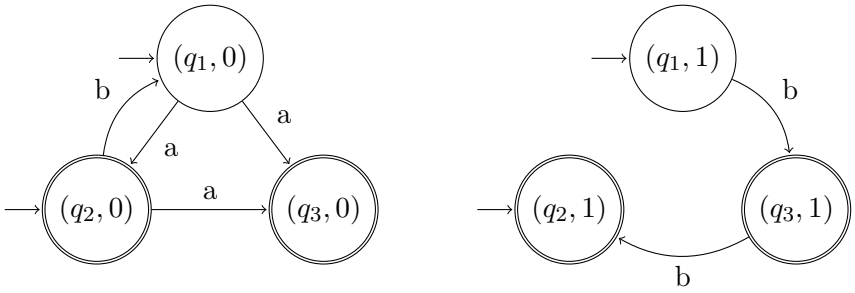
  Sumę rozłączną automatów $\mathcal{A}_\mathcal{E}, \mathcal{A}_\mathcal{F}$ definiujemy następująco:

$$\langle Q_\mathcal{E} \times \{0\} \cup Q_\mathcal{F} \times \{1\}, \Sigma, I_\mathcal{E} \times \{0\} \cup I_\mathcal{F} \times \{1\}, F_\mathcal{E} \times \{0\} \cup F_\mathcal{F} \times \{1\}, \delta \rangle,$$

  gdzie $\delta = \Big\{ \big((q,x), a, (p,x)\big) \mid \big((q,a,p) \in \delta_\mathcal{E} \wedge x = 0\big) \vee \big((q,a,p) \in \delta_\mathcal{F} \wedge x = 1\big) \Big\}$. Suma rozłączna odpowiada „narysowaniu dwóch automatów jeden obok drugiego”, więc rozpoznaje sumę języków, czyli właśnie $L(\mathcal{E} + \mathcal{F})$. Przykładowo sumą rozłączną automatów:

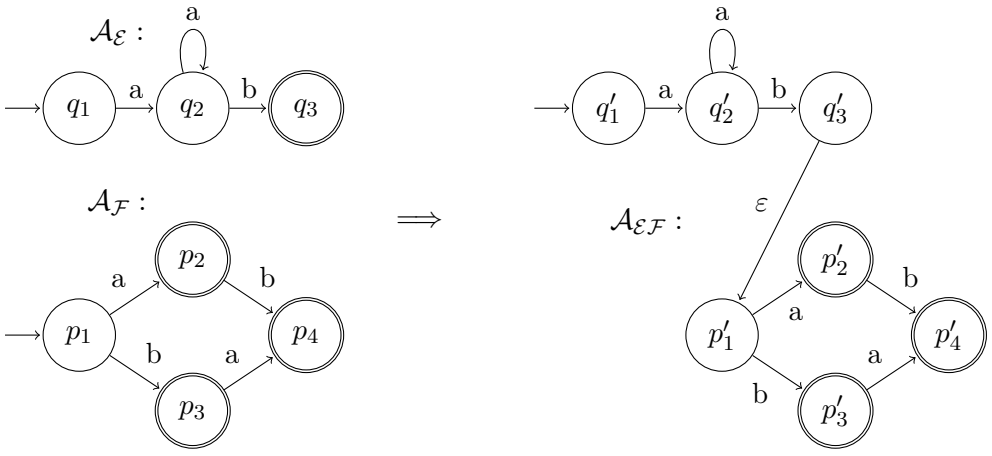


Jest automat:



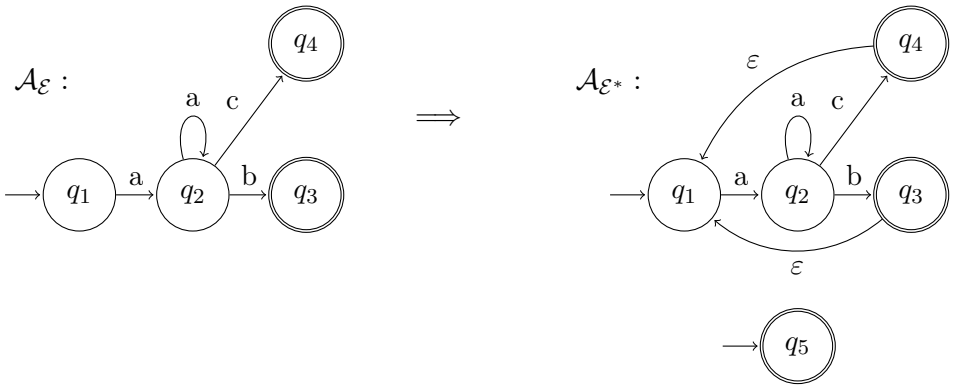
Pozostały dwa przypadki: $\mathcal{EF}$ oraz $\mathcal{E}^*$ – w tych przypadkach konstrukcja będzie dosyć podobna, użyjemy automatów z ε-przejściami.

Wyrażenie $\mathcal{EF}$ – narysujmy automaty $\mathcal{A}_\mathcal{E}$ oraz $\mathcal{A}_\mathcal{F}$ obok siebie a następnie dodajmy tranzycje (f, ε, i) dla wszystkich $f \in F_\mathcal{E}, i \in I_\mathcal{F}$. W końcu usuńmy ze zbioru stanów końcowych wszystkie stany $F_\mathcal{E}$ (czyli zostawmy tylko $F_\mathcal{F}$) oraz ze zbioru stanów początkowych wszystkie stany $I_\mathcal{F}$ (czyli zostawmy tylko stany $I_\mathcal{E}$).



Działanie tak otrzymanego automatu możemy opisać poprzez „przejdź bieg akceptujący automatu $\mathcal{A}_\mathcal{E}$, a następnie teleportuj się do dowolnego stanu początkowego automatu $\mathcal{A}_\mathcal{F}$ i przejdź w nim bieg akceptujący”, co jest pożądanym efektem.

Wyrażenie $\mathcal{E}^*$ – dodajmy do $\mathcal{A}_\mathcal{E}$ tranzycje po ε ze stanów akceptujących do początkowych. Należy również zadbać o akceptację słowa ε – wystarczy dodać nowy stan początkowy będący jednocześnie stanem akceptującym.



Korzystając z twierdzenia 2.1 otrzymujemy prawdziwość dowodzonej implikacji.

Otrzymujemy więc równoważność (1) $\iff$ (3). Wobec uzyskanej wcześniej równoważności (1) $\iff$ (2) otrzymujemy równoważność wszystkich trzech warunków, czyli tezę twierdzenia 2.2.

■

2.3. Podstawowe własności języków regularnych

Wiemy już, że automaty niedeterministyczne, automaty deterministyczne oraz wyrażenia regularne definiują tę samą klasę języków zwaną językami regularnymi. Fakt ten pomoże nam badać własności tej klasy. Warto zauważyć dodatkowo, że dowód twierdzenia 2.2 jawnie konstruował odpowiednie automaty i wyrażenia, z czego później wyciągniemy dodatkowe wnioski.

Mówimy, że klasa języków $\mathcal{F}$ jest zamknięta na daną operację, jeśli po zastosowaniu tej operacji na dowolnych językach z klasy $\mathcal{F}$ wynik również jest językiem należącym do klasy $\mathcal{F}$. Na początku przyjrzyjmy się własnościom mówiącym o byciu zamkniętym ze względu na niektóre operacje.

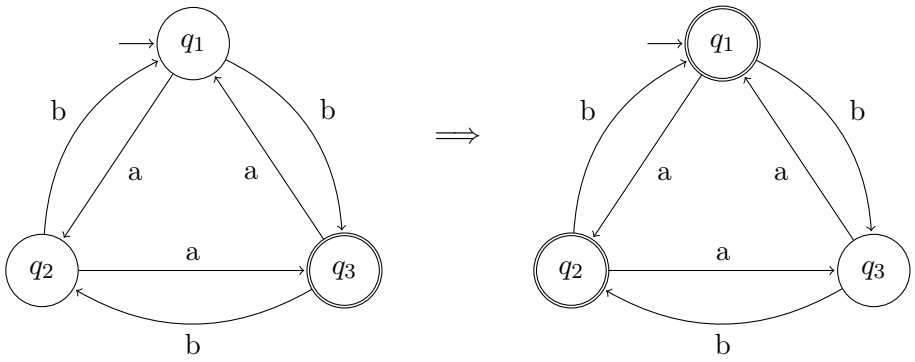
Twierdzenie 2.3. *Jeśli $L, K \subseteq \Sigma^*$ są językami regularnymi, to regularne są też języki:*

- $L \cup K$ ***(suma)***
- $L \cap K$ ***(przecięcie)***
- $L^c = \Sigma^* \setminus L$ ***(dopełnienie)***
- $L^R = \{a_n a_{n-1} \dots a_1 \mid a_1 a_2 \dots a_n \in L\}$ ***(odwrócenie)***

Dowód zamknięcia na sumę. Niech $\mathcal{E}_L, \mathcal{E}_K$ będą wyrażeniami regularnymi które rozpoznają języki L, K odpowiednio. Wyrażenie $\mathcal{E}_L + \mathcal{E}_K$ rozpoznaje język $L \cup K$.

Prawdziwość zamknięcia na operację sumy możemy również uzyskać za pomocą automatów, wystarczy bowiem użyć sumy rozłącznej automatów, tak jak przy konstrukcji automatu dla wyrażenia $\mathcal{E} + \mathcal{F}$ w dowodzie twierdzenia 2.2. ■

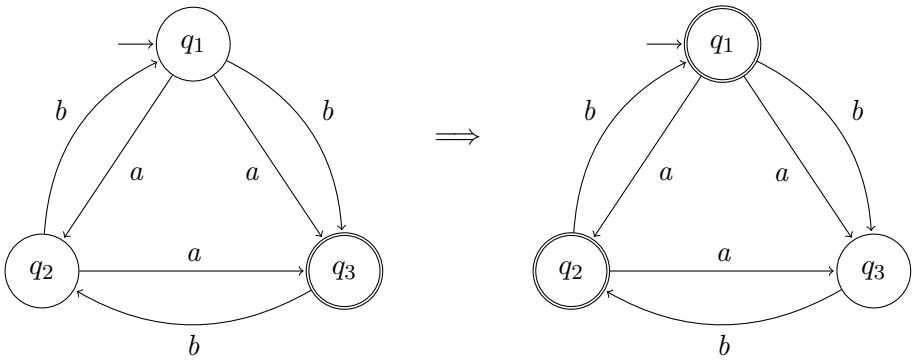
Dowód zamknięcia na dopełnienie. Niech $\mathcal{A}_L$ będzie zupełnym, deterministycznym automatem rozpoznającym język L. Rozważmy automat $\mathcal{C} = \langle Q_L, \Sigma, \{q_i\}, Q_L \setminus F_L, \delta_L \rangle$, który od automatu $\mathcal{A}_L$, różni się tylko zbiorem stanów akceptujących, w taki sposób, że jego zbiór stanów akceptujących to zbiór stanów nieakceptujących automatu $\mathcal{A}_L$. Automat $\mathcal{C}$ jest więc deterministyczny i rozpoznaje dopełnienie języka L (oznaczane $L^c = \Sigma^* \setminus L$), ponieważ po każdym słowie ma on dokładnie jeden bieg, co wynika z faktu, że jest zupełny. Przykładowa zamiana wygląda następująco:



■

Uwaga 2.2. *Założenie o determinizmie jest w istocie konieczne, aby przy takiej zamianie uzyskać dopełnienie. Problem może pojawić się w momencie, w którym po jakimś słowie w możemy dojść zarówno do stanu akceptującego jak i nieakceptującego. Automat akceptuje wtedy słowo w, jednak przedstawiona konstrukcja dostarczy automat który również zaakceptuje słowo w, więc nie będzie rozpoznawał dopełnienia.*

Przykład 2.2. *Spójrzmy na przykład dla jednoliterowego słowa $w = a$:*



Uwaga 2.3. *W celu skonstruowania wyrażenia regularnego odpowiadającego dopełnieniu danego wyrażenia $\mathcal{E}$, można zastosować następujący algorytm:*

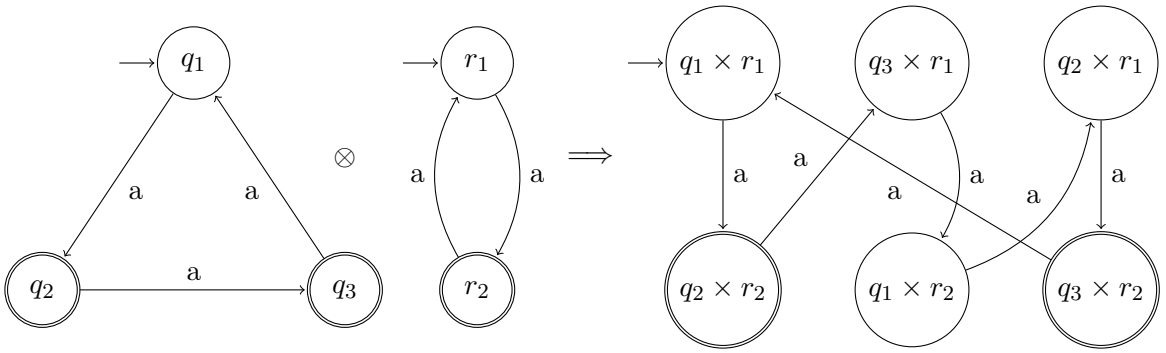
1. *skonstruować automat $\mathcal{A}$ rozpoznający język $L(\mathcal{E})$,*
2. *skonstruować automat $\mathcal{B}$ rozpoznający język $L(\mathcal{E})^c$ zgodnie z dowodem twierdzenia 2.3,*
3. *skonstruować wyrażenie regularne rozpoznające język automatu $\mathcal{B}$ zgodnie z dowodem twierdzenia 2.2.*

Dowód zamknięcia na przecięcie. Wykażemy teraz, że $L \cap K$ jest językiem regularnym. Zauważmy, że język ten wyraża się za pomocą sum i dopełnień, które zachowują regularność:

$$L \cap K = (L^c \cup K^c)^c = \Sigma^* \setminus ((\Sigma^* \setminus L) \cup (\Sigma^* \setminus K))$$

Innym sposobem dowiedzenia tego faktu jest rozważenie automatu (zwanego produktowym). Niech $\mathcal{A}_L$, $\mathcal{A}_K$ będą automatami rozpoznającymi języki L i K odpowiednio. Automat produktowy to automat którego stany to pary stanów – jeden stan z automatu $\mathcal{A}_L$, a drugi z automatu $\mathcal{A}_K$. Stany początkowe/akceptujące to tylko te, które zawierają dwa odpowiednio początkowe/akceptujące stany. Tranzycje w tej konstrukcji dziedziczymy na każdej współrzędnej z odpowiadającego automatu.

Przykładowa konstrukcja automatu iloczynowego znajduje się poniżej.

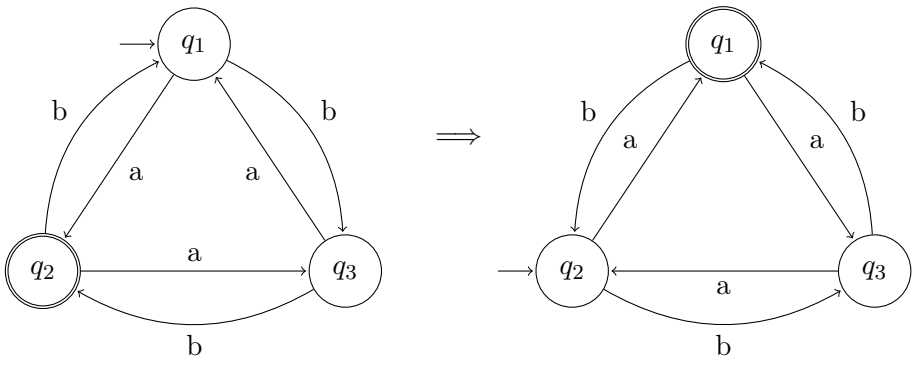


■

Dowód zamknięcia na odwrócenie. Pozostało wykazać zamkniętość na odwrócenie. Rozważmy automat $\mathcal{A} = \langle Q_L, \Sigma, I_L, F_L, \delta_L \rangle$, rozpoznający język L, oraz automat $\mathcal{D}$ zdefiniowany poprzez:

$$\mathcal{D} = \langle Q_L, \Sigma, F_L, I_L, \{(q, a, p) \mid (p, a, q) \in \delta_L\} \rangle.$$

Automat $\mathcal{D}$ ma zamienione stany akceptujące z początkowymi oraz odwrócone tranzycje, co sprawia, że rozpoznaje język L^R. Jest tak ponieważ implementuje on przejście po żądanym słowie „od tyłu".



■

Uwaga 2.4. *Warto zaznaczyć, że powyższa konstrukcja nie wymaga, aby początkowy automat $\mathcal{A}$ był deterministyczny, jednak sprawia, że wynikowy automat $\mathcal{D}$ może nie być deterministyczny, nawet jeśli początkowo $\mathcal{A}$ był deterministyczny.*

Twierdzenie 2.4. *Niech L będzie językiem regularnym nad alfabetem Σ, a $f\colon \Sigma \to P(\Gamma^*)$ funkcją przypisującą literom alfabetu Σ języki regularne nad alfabetem Γ. Następujący język jest regularny:*

$$f(L) = \bigcup_{a_1 \ldots a_n \in L} f(a_1) \cdot f(a_2) \cdot \ldots \cdot f(a_n)$$

Dowód. Aby pokazać, że $f(L)$ jest językiem regularnym skonstruujemy wyrażenie regularne rozpoznające ten język. Niech $\mathcal{E}$ będzie wyrażeniem regularnym języka L. Niech $(\mathcal{E}_a)_{a \in \Sigma}$ będzie taką rodziną wyrażeń regularnych, że dla każdego $a \in \Sigma$ zachodzi $\mathrm{L}(\mathcal{E}_a) = f(a)$. Niech $\mathcal{E}_f$ będzie wyrażeniem powstającym w wyniku podstawienia w miejsce liter wyrażenia $\mathcal{E}$ odpowiadających im wyrażeń z rodziny $(\mathcal{E}_a)_{a \in \Sigma}$. Prosty dowód indukcyjny po strukturze wyrażenia $\mathcal{E}_f$ pokazuje, że wyrażenie to generuje język $f(L)$. ■

Aby poznać dalsze własności języków regularnych potrzebować będziemy pojęcia homomorfizmu. Funkcję $h\colon \Sigma^* \to \Gamma^*$ nazwiemy **homomorfizmem** jeśli spełnia warunki:

- $f(\varepsilon) = \varepsilon$,
- $\forall w, v \in \Sigma^*. f(w \cdot v) = f(w) \cdot f(v)$.

Twierdzenie 2.5. *Niech $h\colon \Sigma^* \to \Gamma^*$ będzie homomorfizmem. Jeżeli $L \subseteq \Gamma^*$ jest językiem regularnym, to również $h^{-1}(L)$ jest językiem regularnym.*

Dowód. Niech $\mathcal{A}$ będzie DFA rozpoznającym L. Skonstruujmy niedeterministyczny automat $\mathcal{B}$ biorąc Q, Σ, I, F z automatu $\mathcal{A}$. Tranzycje automatu $\mathcal{B}$ tworzymy w następujący sposób. Jeżeli $\delta_{\mathcal{A}}(p, h(a)) = q$, dla pewnych $p, q \in Q$ oraz $a \in \Sigma$, to w $\mathcal{B}$ dodajemy tranzycję $p \xrightarrow{a} q$. Automat $\mathcal{B}$ rozpoznaje $h^{-1}(L)$, należy pokazać dwa zawierania. Jeśli $\mathcal{B}$ akceptuje słowo $w = w_1 \dots w_n$, to po przyłożeniu h do tego słowa otrzymujemy słowo $h(w) = h(w_1) \dots h(w_n)$ akceptowane przez $\mathcal{A}$ (wprost z konstrukcji, kolejne w_i odpowiadają tranzycjom w $\mathcal{A}$). Jeżeli słowo w należy do $h^{-1}(L)$, to dla każdego jego podziału $w = h(v_1) \dots h(v_n)$, któremu wprost odpowiada słowo $v_1 \dots v_n$, istnieje w $\mathcal{B}$ bieg akceptujący po literach $v_1, v_2, \dots v_n$. ■

Uwaga 2.5. *Zauważmy, że jeśli $h\colon \Sigma^* \to \Gamma^*$ jest homomorfizmem, to z własności:*

$$\forall w, v \in \Sigma^*. h(w)h(v) = h(wv)$$

wynika własność:

$$\forall w \in \Sigma^*. h(w) = h(w[1]) \cdot \ldots \cdot h(w[|w|]),$$

ponieważ możemy indukcyjnie dokładać nową literę na koniec słowa, korzystając z szczególnej postaci dla słowa v będącego pojedynczą literą:

$$\forall w \in \Sigma^*, a \in \Sigma. h(wa) = h(w)h(a).$$

Wniosek 2.1. *Wobec powyższej uwagi języki regularne są zamknięte na obrazy przy funkcjach homomorficznych h. Fakt ten wynika z twierdzenia 2.4 gdy podstawimy za funkcję f z treści twierdzenia homomorfizm h.*

Zdefiniujemy teraz operacje ilorazu lewostronnego oraz prawostronnego, aby poznać kolejne własności omawianej klasy języków.

Dla słów w, v, gdzie $w = c_1 \dots c_k$ jest prefiksem $v = c_1 \dots c_n$ definiujemy **iloraz lewostronny** słowa v przez słowo w jako:

$$w^{-1}v = c_{k+1} \dots c_n.$$

Analogicznie, jeśli $w = c_{k+1} \dots c_n$ jest sufiksem słowa $v = c_1 c_2 \dots c_n$, to **prawostronnym ilorazem** v przez w nazywamy:

$$vw^{-1} = c_1 \dots c_k.$$

Dla języków L, K **ilorazem lewostronnym** języka L przez język K nazywamy język:

$$K^{-1}L = \{w \mid \exists v. v \in K \wedge vw \in L\}.$$

Odpowiednio **ilorazem prawostronnym** języka L przez język K jest:

$$LK^{-1} = \{w \mid \exists v. v \in K \wedge wv \in L\}.$$

Przejdźmy teraz do sformułowania twierdzenia.

Twierdzenie 2.6. *Niech $L, K \subseteq \Sigma^*$ będą językami regularnymi, wtedy języki:*

1. $K^{-1}L$,
2. LK^{-1}

również są regularne.

Dowód. Niech $\mathcal{A} = \langle \Sigma, Q, q_i, F, \delta \rangle$ będzie zupełnym automatem deterministycznym rozpoznającym L. Zarówno punkt pierwszy jak i drugi udowodnimy modyfikując automat $\mathcal{A}$. Zacznijmy od dowodu punktu pierwszego.

Zdefiniujmy automat $\mathcal{B}$ jako modyfikację automatu $\mathcal{A}$ uzyskaną poprzez zmianę zbioru stanów początkowych według następującej formuły:

$$I_{\mathcal{B}} = \{q \in Q \mid \exists w.w \in K \wedge \delta(q_i, w) = q\}$$

Jeśli $v \in \mathrm{L}(\mathcal{B})$, to istnieje $w \in K$ (z definicji wyżej) takie, że $wv \in L$. W drugą stronę, jeśli istnieje $s \in K^{-1}L$, to rozkłada się w taki sposób, że $s = w^{-1}v$ dla pewnych $w \in K, v \in L$. Aby uzyskać bieg akceptujący po słowie s w automacie $\mathcal{B}$ należy przejść drugą część biegu akceptującego słowa v w automacie $\mathcal{A}$ (zaczynającą się po przejściu słowa w). Język $K^{-1}L$ jest więc regularny.

Przejdźmy do dowodu drugiego punktu. Zdefiniujmy automat $\mathcal{C}$ jako modyfikację automatu $\mathcal{A}$ uzyskaną poprzez zamianę zbioru stanów akceptujących w następującej sposób:

$$F_{\mathcal{C}} = \{q \in Q \mid \exists w.\exists p.p \in F, w \in K \wedge \delta(q, w) = p\}$$

Jeśli $v \in \mathrm{L}(\mathcal{C})$, to istnieje $w \in K$ (z definicji wyżej) takie, że $vw \in L$. W drugą stronę, jeśli istnieje $s \in LK^{-1}$, to rozkłada się w taki sposób, że $s = vw^{-1}$ dla pewnych $w \in K, v \in L$. Aby uzyskać bieg akceptujący po słowie s w automacie $\mathcal{B}$ należy przejść pierwszą część biegu akceptującego słowa v w automacie $\mathcal{A}$ (kończącą się na początku sufiksu w). Język LK^{-1} jest więc regularny.

Innym sposobem udowodnienia drugiego punktu jest zaobserwowanie, że

$$\left((K^R)^{-1}L^R\right)^R = LK^{-1}.$$

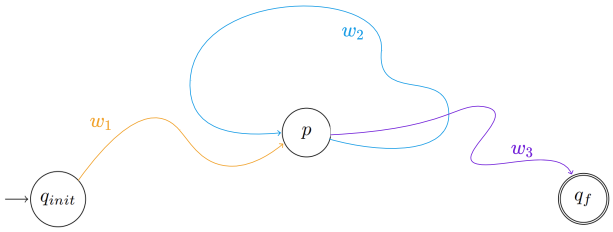
Wobec tego wystarczy skorzystać z tego, że języki regularne są zamknięte na operacje odwracania i ilorazu lewostronnego, co już wiemy. ■

2.4. Lemat o pompowaniu

W tej sekcji udowodnimy ważny lemat, który w wygodny sposób pozwoli nam w niektórych przypadkach stwierdzać, że dany język **nie** jest językiem regularnym. Warto zauważyć, że dotychczasowe informacje o klasie języków regularnych raczej dają nam narzędzia dowodzące, że dany język **jest** regularny.

Lemat 2.1 (Lemat o pompowaniu)**.** *(Patrz na przykład [HMU06, rozdział Properties of Regular Languages, strona 126]) Dla każdego języka regularnego L istnieje stała N taka, że dla każdego słowa $w \in L$ spełniającego $|w| > N$ istnieje podział $w = w_1w_2w_3$ taki, że $w_2 \neq \varepsilon$, $|w_1w_2| \leqslant N$ oraz $w_1w_2^kw_3 \in L$ dla każdego $k \geqslant 0$.*

Dowód. Niech L będzie ustalonym językiem regularnym, $\mathcal{A}$ automatem rozpoznającym język L. Oznaczmy $N = |Q|$, gdzie Q to zbiór stanów automatu $\mathcal{A}$. Rozważmy dowolne słowo $w \in L$ takie, że $|w| > N$ oraz bieg akceptujący π idący po w. Automat $\mathcal{A}$ ma N stanów, więc po co najwyżej N tranzycjach w π wystąpi pierwsze powtórzenie stanu.



Dokonajmy podziału w za pomocą pierwszego powtórzenia stanu w π, czyli jeśli pierwsze powtórzenie wystąpiło na pozycjach (i, j) to

$$w = w_1 w_2 w_3,$$

gdzie

$$w_1 = w[1..i], \quad w_2 = w[(i+1)..j], \quad w_3 = w[(j+1)..|w|].$$

Mając taki podział widzimy, że $|w_1 w_2| \leqslant N, w_2 \neq \varepsilon$ oraz możemy „pompować" pętelkę w_2 (można też ją usunąć, dlatego lemat dopuszcza $k = 0$), co dowodzi tezy. ■

Uwaga 2.6. *Jeśli wiemy, że $0 < |L| < \infty$, to wystarczy wziąć za stałą N wartość nie mniejszą od długości wszystkich słów należących do języka L, na przykład $N = \max_{w \in L} |w|$.*

Uwaga 2.7. *Prawdziwe jest następujące wzmocnienie lematu o pompowaniu:*

Dla każdego języka regularnego L istnieje stała N taka, że dla każdego słowa $w \in L$ i każdego podziału $w = utv$ spełniającego $|t| > N$ istnieje podział $w = uxyzv$ taki, że:

$$y \neq \varepsilon, \ |xy| \leqslant N, \ \forall k \geqslant 0. uxy^k zv \in L.$$

Dowód. Rozumowanie dowodzące powyższej uwagi nie różni się znacząco od dowodu lematu o pompowaniu – stała N jest równa $|Q|$. Powtórzenie stanu musi wystąpić na podsłowie y ze względu na jego długość, tak jak to miało miejsce dla całego słowa w w dowodzie lematu o pompowaniu. Aby otrzymać standardową wersję lematu ze wzmocnionej wersji wystarczy przyjąć:

$$u = v = \varepsilon, \ t = w.$$

■

2.5. Minimalizacja

W tym rozdziale zajmiemy się zupełnymi automatami deterministycznymi. Dla każdego języka regularnego L możemy znaleźć wiele zupełnych DFA rozpoznających ten język, jednak pokażemy, że istnieje pewien szczególny – automat syntaktyczny języka L.

Automat syntaktyczny (lub minimalny) jest wyjątkowy nie tylko ze względu na swój rozmiar – posiada najmniejszą ilość stanów, ale także w pewien sposób zawiera się w każdym innym zupełnym DFA rozpoznającym język L – dowolny zupełny DFA można „posklejać" (jego stany) tak, aby w wyniku dostać automat syntaktyczny $\mathcal{A}_L$. W celu uzyskania powyższych wniosków zdefiniujmy relację Myhilla-Nerodego [RS59].

Relacja Myhilla-Nerodego

Niech $L \subseteq \Sigma^*$ będzie dowolnym językiem (niekoniecznie regularnym). Zdefiniujmy relację równoważności $\sim_L \subseteq \Sigma^* \times \Sigma^*$ w następujący sposób: słowa w, v są w relacji $\sim_L$ wtedy i tylko wtedy, gdy nie nie istnieje takie słowo $u \in \Sigma^*$, że po jego doklejeniu na końce słów w oraz v, jedno z powstałych słów należy do L, a drugie nie należy. Ujmując precyzyjniej:

$$w \sim_L v \iff (\forall u \in \Sigma^*.\ wu \in L \iff vu \in L)$$

Dlaczego $\sim_L$ to relacja równoważności? Z definicji, relacja równoważności to relacja która jest: zwrotna, symetryczna oraz przechodnia.

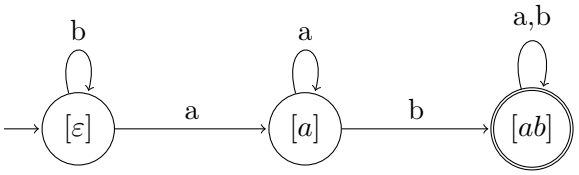
- zwrotna – $\forall w \in \Sigma^*. w \sim_L w$ – wprost z definicji, ponieważ rozważane słowa wu, vu są równe dla $w = v$.
- symetryczna – kolejność słów nie ma znaczenia dla naszej relacji, jeśli wu, vu (nie)należą do L, to również vu, wu (nie)należą do L.
- przechodnia – jeśli wiemy, że par słów: w_1, w_2 oraz w_2, w_3 nie da się rozróżnić (względem przynależności do L) doklejając słowo u na ich koniec, to również w_1, w_3 nie da się rozróżnić w ten sposób.

Automat syntaktyczny *(minimalny)* języka $L \subseteq \Sigma^*$ to automat składający się z:

- alfabetu Σ,
- zbioru stanów $Q = \Sigma^*/_{\sim_L}$,
- zbioru stanów początkowych $I = \{[\varepsilon]_{\sim_L}\}$,
- zbioru stanów akceptujących $F = \{[w]_{\sim_L} \mid w \in L\}$,
- funkcji przejścia $\delta = \{[w]_{\sim_L} \xrightarrow{a} [w \cdot a]_{\sim_L} \mid a \in \Sigma, w \in \Sigma^*\}$.

Jak rozumieć powyższe napisy? Aby skonstruować automat syntaktyczny należy wziąć wszystkie klasy abstrakcji relacji $\sim_L$ jako stany automatu, stan reprezentowany słowem pustym uznać za początkowy, a stany reprezentowane przez słowa należące do języka L uznać za akceptujące. Tranzycja wychodząca ze stanu reprezentowanego przez słowo w idzie po literze $a \in \Sigma$ do klasy reprezentowanej przez słowo wa. Zauważmy, że jeśli $\sim_L$ ma nieskończenie wiele klas abstrakcji, to otrzymany automat ma nieskończony zbiór stanów (nie jest więc automatem skończonym). Standardowo automat syntaktyczny języka L oznaczamy przez $\mathcal{A}_L$.

Przykład dla języka $L = L((a+b)^*ab(a+b)^*)$.



Każdemu stanowi można przypisać klasę abstrakcji relacji $\sim_L$. Reprezentantów tych klas przedstawiamy w kwadratowych nawiasach wewnątrz stanów.

Lemat 2.2. *Ustalmy język $L \subseteq \Sigma^*$. Jeśli relacja $\sim_L$ ma skończenie wiele klas abstrakcji, to automat syntaktyczny jest zupełnym deterministycznym automatem rozpoznającym język L.*

Dowód. Jeśli relacja $\sim_L$ ma skończenie wiele klas abstrakcji, to automat syntaktyczny ma skończony zbiór stanów, dokładnie jeden stan początkowy. Zauważmy, że jeśli $v \sim_L w$, to $(v \cdot a) \sim_L (w \cdot a)$, więc dla każdego stanu $[w]_{\sim_L}$ i każdej litery a mamy dokładnie jedną tranzycję:

$$[w]_{\sim_L} \xrightarrow{a} [w \cdot a]_{\sim_L}.$$

Relacja tranzycji jest więc funkcją, czyli automat jest deterministyczny oraz zupełny. Dlaczego $\mathcal{A}_L$ rozpoznaje L? Mamy dokładnie jeden bieg po każdym słowie. Wiemy, że stany akceptujące to te, których reprezentanci należą do L. Funkcja przejścia wprost mówi, że po słowie v przejdziemy do stanu $[v]_{\sim_L}$, więc $\mathcal{A}_L$ rozpoznaje L, ponieważ:

$$v \in L \iff [v]_{\sim_L} \in F.$$

Powyższa równoważność wynika z tego, że jeśli $v \sim_L w$ to $v \in L \Leftrightarrow w \in L$. ■

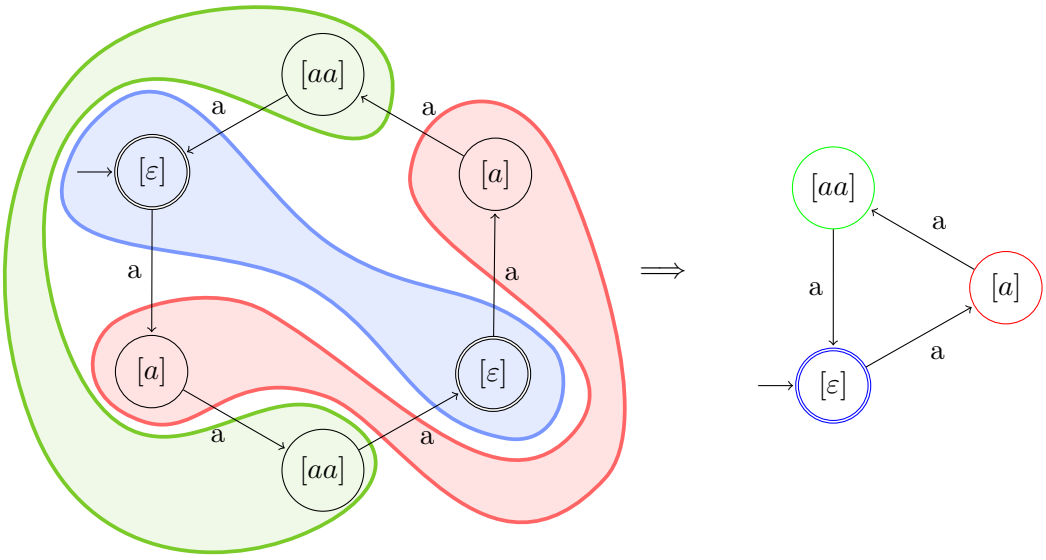
Wprowadźmy oznaczenie $\text{reach}(Q_\mathcal{A})$, które oznaczać będzie zbiór osiągalnych stanów automatu $\mathcal{A}$, to znaczy takich stanów, do których da się dojść, podążając zgodnie z tranzycjami automatu $\mathcal{A}$, zaczynając w jakimś stanie początkowym tego automatu. Sprecyzujmy co rozumiemy poprzez wspomniane **„sklejenie" (iloraz)** automatu $\mathcal{A}$ do zupełnego DFA $\mathcal{B}$.

Powiemy, że **zupełny DFA** $\mathcal{B}$ jest ilorazem **zupełnego DFA** $\mathcal{A}$ jeśli istnieje suriekcja $f : \text{reach}(Q_\mathcal{A}) \xrightarrow{na} Q_\mathcal{B}$ taka, że:

- $p \in I_\mathcal{A} \implies f(p) \in I_\mathcal{B}$,
- $q \in F_\mathcal{A} \iff f(q) \in F_\mathcal{B}$,
- $\forall q \in \text{reach}(Q_\mathcal{A}). \forall a \in \Sigma. f(\delta_\mathcal{A}(q, a)) = \delta_\mathcal{B}(f(q), a)$.

Uwaga 2.8. *Warto odnotować, że po sklejeniu automat rozpoznaje ten sam język. Wynika to z tego, że dla każdego słowa $w \in \Sigma^*$ jedyny bieg Π w automacie $\mathcal{A}$ po słowie w odpowiada wprost jedynemu biegowi po tym słowie w automacie $\mathcal{B}$ powstałemu przez zaaplikowanie funkcji f do stanów biegu Π.*

Przykładowe sklejenie wygląda następująco (sklejamy w obrębie kolorowych pętli)



Lemat 2.3. *Niech $\mathcal{A} = \langle \Sigma, Q_{\mathcal{A}}, \{q_{init}\}, F_{\mathcal{A}}, \delta_{\mathcal{A}} \rangle$ będzie ustalonym zupełnym automatem deterministycznym rozpoznającym język L. Wówczas relacja $\sim_L$ ma skończenie wiele klas abstrakcji, a co więcej automat syntaktyczny $\mathcal{A}_L$ jest ilorazem automatu $\mathcal{A}$ (można go otrzymać poprzez sklejanie stanów automatu $\mathcal{A}$).*

Dowód. Przypiszmy każdemu osiągalnemu stanowi $q \in \text{reach}(Q_{\mathcal{A}})$ słowo w_q w taki sposób, że $\delta_{\mathcal{A}}(q_{init}, w_q) = q$, dla każdego stanu osiągalnego takie słowo musi istnieć (wybieramy dowolne). Rozważmy funkcję $f\colon \text{reach}(Q_{\mathcal{A}}) \to \Sigma^*/_{\sim_L}$ zdefiniowaną wzorem $f(q) = [w_q]_{\sim_L}$. Klasa $[w_q]_{\sim_L}$ nie zależy od wyboru słowa w, po którym idziemy do stanu q, ponieważ określana jest przez słowa po których dochodzimy w automacie $\mathcal{A}$ do stanów akceptujących ze stanu q, co nie jest zależne od drogi przebytej z q_{init} do q. Funkcja f jest więc dobrze określona, a co więcej jest suriekcją (po każdym słowie gdzieś możemy dojść, ponieważ rozważane DFA są zupełne) spełniającą definicję sklejenia. Istotnie dla $I = \{[\varepsilon]_{\sim_L}\}$, $F = \{[w]_{\sim_L} \mid w \in L\}$ pierwsze dwa warunki definicji sklejenia (patrz definicja na stronie 25) są spełnione, wystarczy więc zauważyć, że trzeci warunek również zachodzi. Niech $\delta_{\mathcal{A}}$ będzie funkcją przejścia automatu syntaktycznego, jeśli $(q, a, p) \in \delta_{\mathcal{A}}$, to zachodzą równości

$$f(\delta_{\mathcal{A}}(q, a)) = f(p) = [w_p]_{\sim_L} = [w_q \cdot a]_{\sim_L} = \delta_{\mathcal{A}}([w_q]_{\sim_L}, a),$$

więc trzeci warunek również jest spełniony.

Relacja $\sim_L$ ma skończenie wiele klas abstrakcji, ponieważ zadaliśmy suriekcję ze skończonego zbioru na zbiór klas abstrakcji $\sim_L$. ■

Twierdzenie 2.7 (Myhill-Nerode)**.** *[RS59] Niech $L \subseteq \Sigma^*$ będzie dowolnym językiem. Język L jest regularny wtedy i tylko wtedy, gdy relacja syntaktyczna $\sim_L$ ma skończenie wiele klas abstrakcji.*

Dowód. Pokażemy implikacje w dwie strony.

Załóżmy, że $L \subseteq \Sigma^*$ jest językiem regularnym. Istnieje automat deterministyczny $\mathcal{A}$ rozpoznający L (twierdzenie 2.2), więc na mocny lematu 2.3 otrzymujemy, że $\sim_L$ ma skończenie wiele klas abstrakcji, a co więcej $\mathcal{A}$ możemy skleić do $\mathcal{A}_L$, co kończy dowód implikacji.

Załóżmy z kolei, że $\sim_L$ ma skończenie wiele klas abstrakcji. Możemy skorzystać z lematu 2.2, więc istnieje automat rozpoznający L – czyli język ten jest regularny. ■

Wniosek 2.2. *Dla każdego języka regularnego L istnieje unikalny, z dokładnością do nazw stanów, zupełny automat deterministyczny (zwany syntaktycznym) $\mathcal{A}_L$ rozpoznający L – ma on minimalną liczbę stanów w zbiorze wszystkich zupełnych DFA rozpoznających L, a co więcej każdy zupełny DFA rozpoznający L można do niego skleić poprzez usunięcie stanów nieosiągalnych, a następnie zlepienie niektórych stanów.*

2.5.1. Algorytmy minimalizujące

Zajmiemy się teraz algorytmami minimalizującymi, czyli takimi które otrzymując automat zwracają automat syntaktyczny rozpoznający ten sam język.

Algorytm 2.1 (Rafinacja podziałów *(ang. partition refinement algorithm)*)**.** Pierwszym algorytmem minimalizującym, który omówimy, będzie algorytm rafinacji podziałów. Algorytm ten otrzymuje na wejściu zupełny DFA $\mathcal{A}$, rozpoznający język L, a następnie oblicza automat syntaktyczny $\mathcal{A}_L$ w czasie $O(n^2 \cdot s)$, gdzie $|Q_{\mathcal{A}}| = n$ oraz $|\Sigma| = s$.

Ustalmy zupełny DFA $\mathcal{A}$ rozpoznający język L. Na początku usuńmy wszystkie stany które nie należą do zbioru stanów osiągalnych $\text{reach}(Q_{\mathcal{A}})$ (patrz definicja na stronie 25). Można to

zrobić w czasie $O(n)$ (liniowym od liczby stanów) za pomocą przeszukania automatu od stanów początkowych wszerz używając tablicy do zapamiętywania informacji czy dany stan został już odwiedzony. Przykładowy pseudokod oznaczenia stanów osiągalnych wygląda następująco.

```
bool odwiedzone[|Q_A|]; // Tablica wypełniona wartościami false.
bool tranzycje [|Q_A|][|Q_A|];
queue <int> kolejka; // Kolejka typu first-in-first-out (FIFO).

for (q ∈ I_A) do
    kolejka.dodaj(q);

while (!kolejka.pusta()) do
    q = kolejka.zdejmij();
    for (int i = 1; i ⩽ |Q_A|; i++) do
        if (tranzycje[q][i] && !odwiedzone[i])
            kolejka.dodaj(i);
            odwiedzone[i] = true;

// Wynik znajduje się w tablicy odwiedzone.
```

O automacie powiemy, że jest **automatem osiągalnym** jeśli wszystkie jego stany są osiągalne (patrz definicja na stronie 25). W dalszej części przyjmijmy, że każdy stan automatu $\mathcal{A}$ jest osiągalny.

Dowodząc lematu 2.3, wskazaliśmy funkcję $f\colon Q_\mathcal{A} \to \Sigma^*/_{\sim_L}$ mapującą stany automatu $\mathcal{A}$ na stany automatu syntaktycznego $\mathcal{A}_L$. Niech $\sim$ będzie relacją równoważności zadaną przez przeciwobraz przy f (w klasie abstrakcji są wszystkie stany, które dają tą samą wartość).

Pokażemy, że relację $\sim$ da się obliczyć w czasie $O(n^2 \cdot s)$. Będzie to główna trudność całego algorytmu, ponieważ, znając relację $\sim$, wystarczy zlepić stany w obrębie klas abstrakcji, aby otrzymać automat $\mathcal{A}_L$.

Skonstruujmy automat $\mathcal{B}$ o zbiorze stanów $Q_\mathcal{A} \times Q_\mathcal{A}$, gdzie tranzycja

$$(p, q) \xrightarrow{a} (\delta(p, a), \delta(q, a))$$

występuje dla każdych $p, q \in Q, a \in \Sigma$. Zauważmy, że następuje równoważność: stany p i q nie są w tej samej klasie abstrakcji relacji $\sim$ wtedy i tylko wtedy, gdy ze stanu (p, q) automatu $\mathcal{B}$ można dojść do takiego stanu, który składa się z pary stanów automatu $\mathcal{A}$, w której jeden stan jest akceptujący, a drugi nie. Możemy więc przeszukać wszerz (używając tablicy do spamiętywania stanów odwiedzonych algorytmem którego użyliśmy do znalezienia stanów nieosiągalnych), z tą różnicą, że tranzycjami chodzimy teraz „w tył", a zaczynamy od stanów złożonych z par w których dokładnie jeden stan jest akceptujący, co pozwoli nam obliczyć relację $\sim$.

Powyższa konstrukcja oraz algorytm zajmuje oczywiście czas $O(n^2 \cdot s)$ – takiej wielkości jest konstruowany automat, następnie dokonujemy liniowego przeszukania wszerz oraz złączenia stanów na podstawie uzyskanej informacji.

Algorytm 2.2 (Hopcroft)**.** [Hop71] Następnym algorytmem, który omówimy będzie algorytm Hopcrofta. Tak jak algorytm rafinacji podziałów przyjmuje on na wejście zupełny DFA $\mathcal{A}$ rozpoznający język L, a następnie oblicza automat $\mathcal{A}_L$. Przewagą algorytmu Hopcrofta nad algorytmem rafinacji podziałów jest czas działania, działa w lepszej pesymistycznej złożoności

$$O(n \log(n) \cdot s), \text{ gdzie } |Q_\mathcal{A}| = n, |\Sigma| = s.$$

Algorytm ten utrzymuje podział zbioru stanów P, początkowo będący rozróżnieniem na stany akceptujące i nieakceptujące, oraz listę zbiorów W, którymi będzie w przyszłości dzielił aktualnie trzymany podział P.

Algorytm wykorzystuje listę zbiorów (zmienna W), którymi rozdrabnia podział P. Pobiera z listy W kolejne zbiory (zmienna A). Dla każdego rozważanego zbioru A iteruje się po literach $a \in \Sigma$ oraz zbiorach $Y \in P$ próbując dokonywać dalszej rafinacji. W tym celu próbuje znaleźć takie Y, dla których nie wszystkie tranzycje po a z Y prowadzą do dzielącego w danym momencie zbioru A. Jeśli znajdzie taki zbiór Y dzieli go na mniejsze jednocześnie zastępując go w podziale P. Poniżej znajduje się pseudokod algorytmu Hopcrofta.

```
P = {F, Q \ F}
W = {F, Q \ F}

while W.niepusta() do
    A = W.zdejmij(); // Pobieramy element z W do zmiennej A.
    for a ∈ Σ do
        // Niech X to zbiór stanów z których
        // tranzycja po „a'' idzie do A.
        X = {x | δ(x, a) ∈ A};
        for Y ∈ P : X ∩ Y ≠ ∅ and Y \ X ≠ ∅ do
            // Rozbija w P zbiór Y na zbiory X ∩ Y, Y \ X.
            P.rozbij(Y, {X ∩ Y, Y \ X});
            if Y ∈ W
                // Rozbija w W zbiór Y na zbiory X ∩ Y, Y \ X.
                W.rozbij(Y, {X ∩ Y, Y \ X});
            else
                if |X ∩ Y| ⩽ |Y \ X|
                    W.dodaj(X ∩ Y);
                else
                    W.dodaj(Y \ X);
```

Na początku zauważmy, że algorytm ten posiada własność stopu, ponieważ podział P w każdym obrocie pętli while albo się rozdrabnia, albo się nie zmienia. Podział P może rozdrobnić się jedynie skończenie wiele razy, więc od pewnego momentu zmienna P się nie zmienia, co oznacza, że wewnętrzna pętla for nie wykonuje żadnego obrotu, czyli do zbioru W nic nie jest dodawane a każdy następny obrót pętli while usuwa elementy z W aż do osiągnięcia $W = \varnothing$.

Niech $\mathcal{A} = \langle \Sigma, Q, q_i, F, \delta \rangle$ oraz niech $P_1, P_2, \ldots, P_m$ będą kolejnymi podziałami zbioru stanów Q uzyskanymi w trakcie działania algorytmu uruchomionego na automacie $\mathcal{A}$. Zdefiniujmy relacje równoważności $\sim_1, \sim_2, \ldots, \sim_m$ przez kolejne podziały P_i, to znaczy

$$q_a \sim_i q_b \iff \exists Y \in P_i . \{q_a, q_b\} \subseteq P_i.$$

Zdefiniujmy $L(q) = \{w \mid \delta(q, w) \in F\}$.

Zaczniemy od udowodnienia poprawności algorytmu, czyli od wykazania, że

$$q \sim_m p \iff L(q) = L(p).$$

Pokażemy dwa lematy, które udowodnią powyższą równoważność.

Lemat 2.4. *Niech $p, q \in Q$. Jeśli $q \nsim_i p$, to $L(q) \neq L(p)$.*

Dowód. Załóżmy nie wprost, że istnieje i takie, że $q \nsim_i p$ oraz $L(q) = L(p)$. Załóżmy dodatkowo, że i jest najmniejszym indeksem o tej własności. Jeżeli $i = 1$, to z racji, że $P_1 = \{Q \setminus F, F\}$ wiemy, że dokładnie jeden ze stanów p, q akceptuje słowo ε, co sprawia, że $L(p) \neq L(q)$ i prowadzi do sprzeczności. W przeciwnym przypadku wiemy, że $q \nsim_i p$ oraz $q \sim_{i-1} p$. Oznacza to, że istnieje zbiór $Y_{i-1} \in P_{i-1}$ taki, że $p \in Y_{i-1}$ oraz $q \in Y_{i-1}$ oraz został dokonany podział zbiorem $A \in W$ oraz literą $a \in \Sigma$ w taki sposób, że stany p i q zostały rozdzielone. Bez straty ogólności możemy założyć, że $\delta(p, a) \in A, \delta(q, a) \notin A$. Zauważmy, że jeśli zbiór A był zbiorem wyjętym z W, to istniał on w podziale P (początkowo $P = W$, następnie przy rozbijaniu P rozbijamy również W). Mamy więc sytuację w której $\delta(q, a) \nsim_k \delta(p, a)$, z założenia indukcyjnego dostajemy, że $L(\delta(q, a)) \neq L(\delta(p, a))$, co implikuje $L(p) \neq L(q)$. ■

Lemat 2.5. *Niech $p, q \in Q$. Jeśli $p \sim_m q$, to $L(p) = L(q)$.*

Dowód. $L(p) = L(q)$ oznacza, że dla każdego $w \in \Sigma^*$ zachodzi

$$\forall w.\delta(p, w) \in F \iff \delta(q, w) \in F.$$

Pokażę indukcyjnie po długości słowa w, że jeśli $p \sim_m q$ to powyższa równoważność zachodzi. Dla $w = \varepsilon$ mamy $p \sim_m q \implies p \sim_1 q,\ p \sim_1 q \iff (p, q \in F) \vee (p, q \in Q \setminus F)$, więc

$$\delta(p, \varepsilon) = p \in F \iff \delta(q, \varepsilon) = q \in F.$$

Baza indukcyjna jest więc udowodniona, przejdźmy do dowodu kroku indukcyjnego.

Załóżmy, że dla wszystkich słów o długości co najwyżej d równoważność zachodzi, to znaczy, że

$$\forall |w| \leqslant d.\delta(p, w) \in F \iff \delta(q, w) \in F.$$

Pokażemy, że dla każdego słowa długości $d + 1$ równoważność zachodzi. Załóżmy nie wprost, że istnieje słowo w długości $d + 1$ które rozróżnia stany p i q. Bez straty ogólności załóżmy, że $\delta(q, w) \in F, \delta(p, w) \notin F$. Niech $w = av$, dla pewnych $a \in \Sigma, v \in \Sigma^d$. Mamy więc, że $L(\delta(q, a)) \neq L(\delta(p, a))$, więc z założenia indukcyjnego mamy $\delta(q, a) \nsim_m \delta(p, a)$. Zauważmy, że jeśli $\delta(q, a) \nsim_m \delta(p, a)$, to w pewnym momencie wstawiliśmy na listę W zbiór który zawiera dokładnie jeden ze stanów $\delta(q, a), \delta(p, a)$, albo jest to jeden z początkowych zbiorów $F, Q \setminus F$, albo jest to zbiór powstały w wyniku podziału który rozdzielił te stany. Zbiór ten mógł być rozbijany w dalszych fazach algorytmu (15 linijka pseudokodu), lecz takie rozbicie zachowuje własność, że w W istnieje zbiór do którego należy dokładnie jeden ze stanów $\delta(p, a), \delta(q, a)$. W pewnym momencie musieliśmy zdjąć z listy W ten zbiór i dokonać podziału za jego pomocą, wtedy dostajemy rozdzielenie stanów p, q po literze a, więc $p \nsim_m q$. ■

Powyższy lemat zawiera kluczową do otrzymania złożoności algorytmu obserwację – do uzyskania poprawności algorytmu wystarcza dodawanie jednego ze zbiorów $X \cap Y, Y \setminus X$ (linie $17 - 20$ pseudokodu) do listy W, więc rozmiar wkładanego zbioru jest co najwyżej połową wartości $|Y|$ (bierzemy mniej liczny zbiór).

Zauważmy, że łączna liczba tranzycji w automacie to $n \cdot s$. Oszacujemy sumaryczną moc zbiorów X którymi dokonujemy podziałów w P. Przyjrzyjmy się tranzycji $t = (p, a, q) \in \delta$, jeśli świadczy ona o przynależności $p \in X$ przy pewnym podziale, to po rozbiciu jakiegoś zbioru Y przez X do listy W trafi zbiór rozmiaru co najwyżej $\frac{1}{2}|X|$, stan q może przy takich podziałach należeć do dodawanego do W zbioru co najwyżej $\log_2(n)$ razy. Sumując po wszystkich tranzycjach dostajemy, że sumaryczna moc zbiorów X to $ns \cdot \log_2(n)$.

Do otrzymania złożoności $O(n \log(n) \cdot s)$ brakuje nam już tylko struktury danych, która potrafi rozbić podział zbioru względem zbioru X w czasie proporcjonalnym do $|X|$. Taką strukturę *(ang. partition refinement)* (Przykładowe opracowanie [VL08]) można uzyskać utrzymując:

- uporządkowaną listę dwukierunkową zbiorów S_i,
- dla każdego zbioru S_i dwukierunkową listę jego elementów,
- dla każdego elementu informację w którym zbiorze S_i się znajduje oraz gdzie (wskaźnik na element listy stanów należących do S_i).

Aby dokonać rozbicia iterujemy się po $x \in X$, odwołujemy się do zbioru S_j zawierającego x i dzielimy go (tworząc nowy zbiór będący docelowo $S_j \cap X$) oraz dodajemy ten zbiór na listę L (jeśli go tam jeszcze nie ma) zbiorów podzielonych przez tę operację. Efektywnie usuwamy więc x z S_j oraz dodajemy do powstałego w tej iteracji zbioru $S_j \cap X$. Gdy iteracja po $x \in X$ się zakończy, dokonujemy podziału zgodnie z listą L.

Algorytm 2.3 (Brzozowski)**.** [Brz62]

Algorytm Brzozowskiego, chociaż znacznie wolniejszy – działa pesymistycznie w czasie wykładniczym względem ilości stanów, pozwala minimalizować również automaty niedeterministyczne oraz jest wyjątkowo prosty w zapisie. Wykorzystamy następujące operacje:

- `P` – konstruuje automat potęgowy, a następnie usuwa z niego stany nieosiągalne (tworzy zupełny DFA z NFA),
- `R` – zamienia zbiory F oraz I, a następnie odwraca kierunki tranzycji, na końcu usuwa stany nieosiągalne (odwraca rozpoznawany język).

Algorytm Brzozowskiego przedstawia się następująco: `return P(R(P(R(`$\mathcal{A}$`))))`.

Dlaczego ten algorytm działa? Odpowiedź na to pytanie można uzyskać analizując złożenie `P(R(`$\mathcal{A}$`))`. Zauważmy najpierw poniższą równość:

$$\texttt{P}(\texttt{R}(\texttt{P}(\texttt{R}(\mathcal{A})))) = (\texttt{P} \circ \texttt{R})^2(\mathcal{A}).$$

Cały algorytm dwukrotnie wykonuje operację P $\circ$ R – poprawność otrzymamy dowodząc następującą obserwację.

Obserwacja 2.1. *Dla dowolnego zupełnego DFA $\mathcal{B}$ który jest osiągalny (patrz sekcja o algorytmie 2.1) automat* P(R($\mathcal{B}$)) *to automat syntaktyczny rozpoznający $L(\mathcal{B})^R$, czyli odwrócenie języka L (patrz punkt 10 na stronie 8).*

Dowód. Mając zupełny osiągalny automat deterministyczny $\mathcal{B}$ o zbiorze stanów początkowych $\{q_i\}$ możemy przyporządkować każdemu stanowi q_x takie słowo w_x, że $\delta(q_i, w_x) = q_x$ (ponieważ automat jest deterministyczny zupełny i osiągalny). Niech przyporządkowanie to będzie zadane funkcją $f\colon Q_{\mathcal{B}} \to \Sigma^*$. Po wykonaniu operacji R tranzycje się odwrócą, a jedynym stanem akceptującym będzie q_i. Wynika z tego, że dla każdego stanu q_x będziemy mieć słowo $f(q_x)^R = w_x^R$, które jest akceptowane z tego stanu, a z dowolnego innego nie. W automacie potęgowym z usuniętymi stanami nieosiągalnymi – `P(R(`$\mathcal{B}$`))`, stany odpowiadają podzbiorom stanów automatu $\mathcal{B}$. Stany te będą rozróżnialne ze względu na relację Myhilla-Nerodego, ponieważ będą akceptowały różne podzbiory $f(Q)$ – dla stanu odpowiadającego $Q_x \subseteq Q$ będzie to $f(Q_x)$. ■

Operacja R odwraca język automatu, na którym jest wykonywana, a operacja P zwraca zupełny automat deterministyczny, nie zmieniając języka, więc po wykonaniu pierwszej operacji $P \circ R$ otrzymamy zupełny deterministyczny automat rozpoznający $L(\mathcal{A})^R$, a po drugim wykonaniu automat syntaktyczny języka $L(\mathcal{A})$, co dowodzi poprawności algorytmu Brzozowskiego.

2.6. Zbiory semi-liniowe a języki regularne

W tym podrozdziale zajmiemy się opisem języków regularnych nad alfabetem unarnym (patrz punkt 3 na stronie 7) oraz możliwymi zbiorami długości słów języka regularnego. Aby badać te struktury wprowadzimy najpierw pojęcia liniowości oraz semi-liniowości.

O zbiorze $X \subseteq \mathbb{N}$ powiemy, że jest liniowy, jeśli istnieją takie $a, b \in \mathbb{N}$, że:

$$X = \{a + bx \mid x \in \mathbb{N}\}.$$

O zbiorze $X \subseteq \mathbb{N}$ powiemy, że jest semi-liniowy, jeśli jest skończoną sumą teoriomnogościową zbiorów liniowych, to znaczy, że istnieje $n \in \mathbb{N}$ takie, że istnieją $a_1, a_2, \ldots a_n, b_1, b_2, \ldots, b_n \in \mathbb{N}$ dla których spełniona jest równość:

$$X = \bigcup_{i=1}^{n} \{a_i + b_i x \mid x \in \mathbb{N}\}.$$

Twierdzenie 2.8. *Niech $X \subseteq \mathbb{N}$, równoważne są warunki:*

1. *X jest zbiorem semi-liniowym,*
2. *istnieje język regularny $L \subseteq \{1\}^*$ taki, że $X = \{|w| \mid w \in L\}$,*
3. *istnieje język regularny K taki, że $X = \{|w| \mid w \in K\}$.*

Dowód. Pokażemy cztery implikacje, dowodząc dwóch równoważności: (1) $\iff$ (2) $\iff$ (3).

Implikacja (2) $\implies$ (3) jest oczywista, gdyż możemy wziąć język z punktu (2).

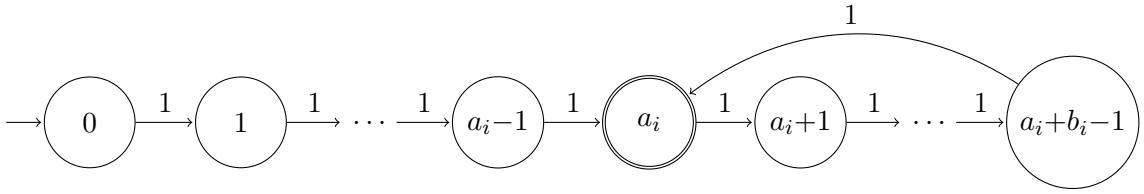
Przejdźmy do (3) $\implies$ (2). Rozważmy automat $\mathcal{A}_K$ rozpoznający język K z punktu (3). Zamieńmy wszystkie etykiety tranzycji na 1 – tak otrzymany automat jest automatem języka L spełniającym warunek (2), ponieważ zbiór długości słów akceptowanych się nie zmienił.

Przejdźmy teraz do (1) $\implies$ (2). Skonstruujmy NFA $\mathcal{A}$ rozpoznający język $\{1^n \mid n \in X\}$. Niech $X = \bigcup_{i=1}^{n} \{a_i + b_i x \mid x \in \mathbb{N}\}$. Skonstruujemy automaty $\mathcal{A}_i$ rozpoznające języki:

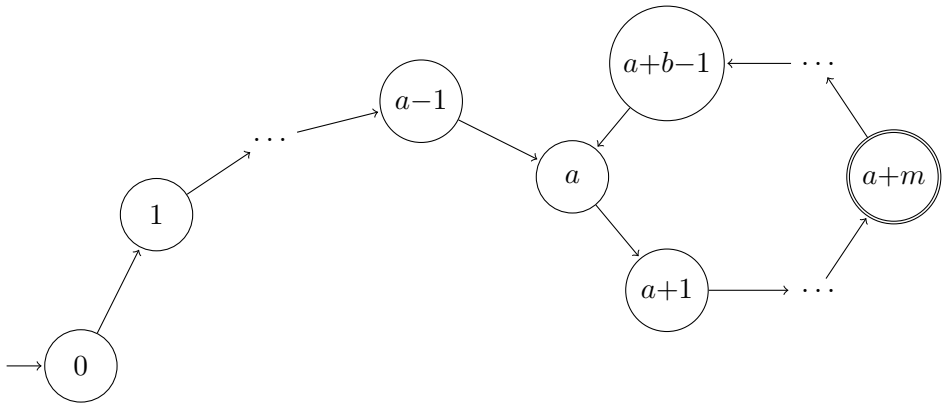
$$L_i = \{1^m \mid m = a_i + b_i x,\ x \in \mathbb{N}\}.$$

Oczywiście $L = \bigcup_{i=1}^{n} L_i$, więc będzie to wystarczające do wykazania rozważanej implikacji.

Automaty $\mathcal{A}_i$ konstruuje się dosyć prosto, każdy z nich składa się z stanu początkowego w którym zaczyna się linia długości a_i, na której końcu jest pętla długości b_i, jedynym stanem akceptującym jest złączenie linii oraz pętli.



Pokażmy więc ostatnią implikację, (2) $\implies$ (1). Rozważmy dowolny język regularny nad $\{1\}^*$ oraz jego automat syntaktyczny $\mathcal{A}$. Z faktu, że jest to zupełny DFA nad językiem jednoliterowym, wiemy, że ma jeden stan początkowy oraz dokładnie jedną tranzycję z każdego stanu. Jest minimalny, więc każdy jego stan jest osiągalny. Idąc ze stanu początkowego, idziemy (jedynym biegiem) cały czas do nowych stanów, aż dotrzemy do pierwszego powtórzenia, a wtedy kręcimy się w kółko – $\mathcal{A}$ wygląda więc jak lasso:



Ma on jeden stan początkowy, linię długości a i cykl długości b. Jaki język generuje? Stany akceptujące nieobjęte pętlą generują po jednym słowie, które możemy dodać w postaci zbiorów liniowych jednoelementowych $\{k + 0 \cdot x \mid x \in \mathbb{N}\}$, ponieważ jest ich skończenie wiele. Jeśli stan akceptujący jest m–tym na pętli licząc od złączenia, to biegi kończące się w nim świadczą o akceptacji zbioru słów $\{1^t \mid t = a + m + bx, x \in \mathbb{N}\}$. Cały zbiór długości słów języka jest skończoną sumą zbiorów liniowych – czyli jest semi-liniowy. ■

Obserwacja 2.2. *Automaty deterministyczne nad alfabetem unarnym, w których nie ma stanów nieosiągalnych oraz takich, z których nie da się zaakceptować żadnego słowa, są postaci „lassa" przedstawionego na rysunku wyżej – składają się ze stanu początkowego, linii oraz pętli.*

Powyższe równoważności nasuwają wnioski o własnościach semi-liniowych podzbiorów $\mathbb{N}$.

Wniosek 2.3. *Niech $X, Y \subseteq \mathbb{N}$ będą semi-liniowe. Wówczas, na mocy wykazanych własności języków regularnych, zbiorami semi-liniowymi są również:*

- $X \cup Y = \{z \mid z \in X \vee z \in Y\}$ *- teoriomnogościowa suma,*
- $XY = \{x + y \mid x \in X,\ y \in Y\}$ *- suma Minkowskiego,*
- $X \cap Y$ *- przecięcie,*
- $\mathbb{N} \setminus X$ *- dopełnienie,*
- $X^* = \left\{ \sum_{i=1}^{n} x_i \mid n \in \mathbb{N},\ x_i \in X \right\}$ *- wniosek z domknięcia Kleenego.*

2.7. Nauka automatu

W tym podrozdziale poruszymy kwestię uczenia się automatu deterministycznego rozpoznającego język regularny. Opiszmy wpierw proces „nauczania". Mamy dwie osoby – ucznia oraz nauczyciela. Nauczyciel zna język regularny $L \subseteq \Sigma^*$, a uczeń próbuje podać automat deterministyczny rozpoznający ten język, znając na początku tylko alfabet Σ. W tym celu uczeń może zadawać dwa rodzaje pytań, na które odpowiada nauczyciel:

- przynależność – uczeń podaje słowo w, a nauczyciel odpowiada, czy w należy do L,
- zgadnięcie – uczeń podaje DFA $\mathcal{A}$, a nauczyciel odpowiada, czy $L(\mathcal{A}) = L$. Jeśli odpowiedź jest negatywna, to nauczyciel podaje uczniowi kontrprzykład – słowo w rozróżniające te języki.

Gdy uczeń zapyta opcją zgadującą i otrzyma odpowiedź pozytywną, uznajemy, że nauczył się języka L, czyli osiągnął swój cel.

W oczywisty sposób same zapytania przynależnościowe nie wystarczą do nauczenia się języka (z jednym końcowym zgadnięciem), ponieważ dla dowolnego skończonego zbioru słów i informacji o ich przynależności do języka istnieje więcej niż jeden język regularny spełniający zadaną przynależność (takich języków regularnych jest nieskończenie wiele).

Z drugiej strony, same pytania zgadujące wystarczają do nauczenia się języka – możemy zadać bijekcję z liczb naturalnych w języki regularne (gdyż jest ich przeliczalnie wiele) i pytać o ich automaty syntaktyczne, w ten sposób otrzymujemy algorytm używający samych zgadnięć.

Wypada wyjaśnić, jak określamy złożoność w tym przypadku – jest to maksymalna używana ilość zapytań w zależności od wielkości automatu syntaktycznego języka L oraz rozmiaru kontrprzykładów podawanych przez nauczyciela, przy czym drugą składową opisu złożoności można usunąć, jeśli założymy dodatkowo, że nauczyciel podaje zawsze minimalne co do długości kontrprzykłady. Powyższy algorytm używający samych zgadnięć działa w czasie wykładniczym, ponieważ ilość automatów dających się zakodować jako napis o długości co najwyżej n jest wykładnicza względem n.

Powyższe obserwacje nasuwają na myśl to, czym teraz się zajmiemy – pokażemy wielomianowy algorytm pozwalający nauczyć się języka regularnego.

Uczeń pamięta dwa skończone zbiory: $Q, T \subseteq \Sigma^*$. Intuicja jest taka, że słowa z Q reprezentują stany automatu, a T jest zbiorem słów rozróżniających klasy abstrakcji relacji, która przybliża relację Myhilla-Nerodego. Powiemy, że słowa w, v są T-równoważne ($w \sim_T v$) jeśli:

$$\forall u \in T. \quad wu \in L \iff vu \in L.$$

Jeżeli para słów nie jest T-równoważna, to mówimy, że jest T-nierównoważna.

Obserwacja 2.3. *Jeśli $\sim_T$ rozróżnia dwa słowa, to tym bardziej relacja Myhilla-Nerodego rozróżnia te słowa, więc słowa z Q są parami w różnych klasach równoważności $\sim_L$.*

Dla pary zbiorów (Q, T) zdefiniujmy dwie własności:

- poprawność – wszystkie słowa zbioru Q są parami $\boldsymbol{T}$**-nierównoważne**,
- zupełność – dla każdych $q \in Q, a \in \Sigma$ istnieje $p \in Q$ takie, że $qa \sim_T p$.

Jeśli (Q, T) jest parą zarówno poprawną jak i zupełną oraz $\varepsilon \in Q$, to w naturalny sposób definiuje automat deterministyczny. Jego zbiór stanów to Q, stanem początkowym jest słowo puste, a zbiorem stanów akceptujących jest $F = \{w \mid w \in L \cap Q\}$. Funkcja przejścia zadana jest poprzez $\delta(q, a) = p$, gdzie $qa \sim_T p$.

Lemat 2.6. *Niech Q, T będą skończonymi podzbiorami Σ^*. Jeśli para (Q, T) jest poprawna dla ustalonego języka regularnego L, to przy użyciu wielomianowej liczby (względem $\mathcal{A}$, Q oraz T) zapytań przynależnościowych uczeń może znaleźć $P \supseteq Q$ takie, że para (P, T) jest poprawna i zupełna.*

Dowód. Jeśli mamy $q \in Q$, które nie ma swojego następnika (T-równoważnego) po jakiejś literze $a \in \Sigma$, to wystarczy dodać słowo qa. Aby sprawdzić, że tak faktycznie jest, wystarczy zapytać o przynależność do języka L słów qau, pu dla wszystkich $p \in Q, u \in T$. Jeżeli dla każdego p znajdziemy u rozróżniające słowa qa i p dodajemy nowe słowo qa do zbioru Q. Zauważmy, że nie możemy dodać więcej słów niż $|\Sigma^*/_{\sim_L}|$, ponieważ relacja $\sim_T$ rozróżnia słabiej, niż $\sim_L$. Dodanie nowego słowa zajmuje czas wielomianowy, ograniczenie na $|Q|$ jest liniowe, więc cała procedura jest wielomianowa na skutek nieusuwania elementów z Q. ■

Algorytm 2.4 (Angluin)**.** [Ang87]

1. Zaczynamy z $Q = T = \{\varepsilon\}$.
2. Niezmiennik: para (Q, T) jest poprawna, niekoniecznie zupełna.
3. Postępujemy zgodnie z lematem 2.6 zamieniając Q na P tak, aby para (P, T) była zarówno poprawna jak i zupełna. Redefiniujemy $Q \coloneqq P$.
4. Liczymy automat syntaktyczny $\mathcal{A}$ pary (Q, T) i zgadujemy czy $L(\mathcal{A}) = L$?
5. Jeśli się udało, to kończymy.
6. Dodajemy wszystkie sufiksy kontrprzykładu do T tworząc T'.
7. Wracamy do punktu 2 podstawiając $T \coloneqq T'$.

Lemat 2.7. *Po szóstym kroku para (Q, T') nie jest zupełna.*

Dowód. Załóżmy nie wprost, że para (Q, T') jest zupełna, a otrzymany kontrprzykład dla automatu który powstał z (Q, T) to w. Oczywiście para (Q, T) jest zupełna więc definiuje ten sam automat co para (Q, T'), ponieważ do rozróżniania wystarczy używać tylko słów z T. Zauważmy jednak, że idąc po słowie w w obu automatach, dostajemy różne wyniki – słowo jednocześnie jest akceptowane jak i nie jest akceptowane przez izomorficzne automaty deterministyczne, co jest sprzecznością, ponieważ w automacie deterministycznym istnieje tylko jedna droga po danym słowie. ■

Wniosek 2.4. *$|Q| \leqslant |\Sigma^*/_{\sim_L}|$, więc algorytm Angluiny terminuje po co najwyżej $|\Sigma^*/_{\sim_L}|$ obrotach „pętli" złożonej z punktów $2 - 6$, ponieważ punkt 3 wykonany po punkcie 6 zawsze zwiększa rozmiar Q. Gdy $|Q| = |\Sigma^*/_{\sim_L}|$ otrzymany w punkcie 4 automat rozpozna L, ponieważ Q będzie odpowiadało zbiorowi klas abstrakcji relacji Myhill'a-Nerode'go.*

Wniosek 2.5. *Złożoność algorytmu Angluiny jest wielomianowa, ponieważ wielomianowo wiele razy wykonuje pętlę która jest możliwa do wykonania w wielomianowej liczbie pytań.*

Powyższe wnioski kończą dowód poprawności i złożoności przedstawionego algorytmu.

2.8. Automaty a logika

W tym podrozdziale pokażemy kolejne spojrzenie na języki regularne. Tym razem wykorzystamy do tego celu *MSO (ang. monadic second-order logic)*, czyli monadyczną logikę drugiego rzędu. Jednak zanim dowiemy się czym jest *MSO* i jaki ma związek z językami regularnymi, zdefiniujmy *FO (ang. first-order logic)* - logikę pierwszego rzędu. W ogólności logiki rozważa się nad tak zwanymi strukturami relacyjnymi, my ograniczymy się do rozważań na zbiorze słów skończonych (nad ustalonym, skończonym alfabetem).

Logika pierwszego rzędu na zbiorze Σ^*, gdzie Σ jest pewnym alfabetem, jest to logika w której do budowania zdań możemy używać standardowych operacji logicznych:

$$\vee, \wedge, \neg, \Rightarrow, \Leftarrow, \iff$$

oraz kwantyfikować po pozycjach słowa za pomocą kwantyfikatora ogólnego $\forall$ i szczególnego $\exists$. Co więcej dostępne są relacje o różnych arnościach (liczbach argumentów). Możemy korzystać z dwóch relacji o arności dwa – $x \leqslant y$ służące do porównywania kolejności pozycji w słowie

oraz $\text{succ}(x, y)$ wyrażającej, że $x + 1 = y$. Do dyspozycji mamy również $|\Sigma|$ relacji o arności równej jeden, sprawdzających czy na danej pozycji słowa stoi konkretna litera, to znaczy dla każdego $a \in \Sigma$ dostępna jest relacja unarna $a(x)$ sprawdzająca, czy na pozycji x stoi litera a. Dodatkowo używamy nawiasów. Jeżeli po kwantyfikowanej zmiennej stoi kropka, oznacza to, że kwantyfikator sięga możliwie daleko (zazwyczaj do końca formuły, bądź końca nawiasu w którym kwantyfikator się znajduje), w przeciwnym przypadku kwantyfikator obejmuje tylko podformułę stojącą bezpośrednio po nim.

Przyjrzyjmy się następującej formule nad $\Sigma^* = \{a, b\}^*$:

$$\exists x.\forall y.\neg\text{succ}(y, x) \wedge a(x)$$

Formuła ta opisuje słowa zaczynające się od litery a. Jeśli formuła ϕ jest spełniona przez słowo w, to piszemy $w \models \phi$. Zbiór wszystkich słów spełniających formułę ϕ oznaczamy $L(\phi)$.

Czym różni się *MSO* od *FO*? **Logika drugiego rzędu** pozwala nam kwantyfikować po relacjach, a nie tylko po pojedynczych pozycjach – słowo **monadyczna** oznacza, że kwantyfikujemy po relacjach unarnych, czyli tylko po zbiorach pozycji słowa.

Dodatkowo możemy pisać $x \in X$, jeśli x jest zmienną pierwszego rzędu, a X drugiego (zbiorem). Aby rozróżnić, którego rzędu jest kwantyfikowana zmienna, używamy małych liter do zmiennych przebiegających po pozycjach, a wielkich liter do zmiennych przebiegających po zbiorach pozycji słowa.

Przykładowo nad $\Sigma^* = \{a, b\}^*$:

$$\exists X.\exists x.x \in X \wedge \forall y.\neg\text{succ}(y, x) \wedge \\ \wedge(\forall z.(z \in X) \Rightarrow (a(z) \wedge \forall p.\forall q.(\text{succ}(z, p) \wedge \text{succ}(p, q)) \Rightarrow q \in X))$$

Powyższa formuła mówi, że istnieje zbiór X (pozycji nieparzystych), w którym istnieje element x, który nie ma poprzednika (jest pozycją 1) oraz wszystkie pozycje zbioru X mają tę własność, że pozycja o dwa dalsza, jeśli istnieje, to też jest w X. Wszystkie elementy $z \in X$ spełniają $a(z)$, czyli na pozycjach z X stoją litery a. Formuła ta rozpoznaje więc słowa posiadające na pozycjach nieparzystych litery a, czyli język $\text{L}((a(a + b))^*(a + \varepsilon))$.

Często zamiast pisać $\exists q.\text{succ}(p, q) \wedge \phi(q)$, będziemy pisać $\phi(\text{succ}(p))$, a zamiast formuły $\exists x.\forall y.\neg\text{succ}(y, x) \wedge \psi(x)$ będziemy po prostu pisać 1 w miejsce x, czyli $\psi(1)$. W powyższym przypadku skraca to formułę do:

$$\exists X.1 \in X \wedge \forall z.(z \in X) \Rightarrow (a(z) \wedge \text{succ}(\text{succ}(z)) \in X)$$

Będziemy również używać $\text{last}(x)$ w celu oznaczenia $\neg\exists y.\text{succ}(x, y)$.

Uwaga 2.9. *Za pomocą* $\exists, \leqslant, \neg, \vee$ *możemy zdefiniować* $\wedge, >, <, \geqslant, =, \forall, \Rightarrow, \Leftarrow \iff$.

Dowód.

$$\begin{aligned}
(\phi \wedge \psi) &\ wtw.\ (\neg((\neg\phi) \vee (\neg\psi))) \\
(x > y) &\ wtw.\ (\neg(x \leqslant y)) \\
(x < y) &\ wtw.\ (y > x) \\
(x \geqslant y) &\ wtw.\ (y \leqslant x) \\
(x = y) &\ wtw.\ (x \leqslant y) \wedge (x \geqslant y) \\
(\forall x.\phi(x)) &\ wtw.\ (\neg(\exists x.\neg\phi(x))) \\
(\phi \Rightarrow \psi) &\ wtw.\ (\neg\phi \vee \psi) \\
(\phi \Leftarrow \psi) &\ wtw.\ (\neg\psi \vee \phi) \\
(\phi \iff \psi) &\ wtw.\ ((\phi \wedge \psi) \vee (\neg\phi \wedge \neg\psi))
\end{aligned}$$

■

Twierdzenie 2.9 (Büchi–Elgot–Trakhtenbrot)**.** *[Elg61] Ustalmy język $L \subseteq \Sigma^*$. Następujące warunki są równoważne:*

1. *istnieje NFA $\mathcal{A}$, taki, że $L(\mathcal{A}) = L$,*
2. *istnieje formuła MSO ϕ taka, że $L(\phi) = L$.*

Dowód. Pokażemy implikacje w obie strony. Zacznijmy od (1) $\implies$ (2). Możemy założyć, że $\mathcal{A}$ jest automatem deterministycznym o n stanach, ze stanem początkowym 1 na mocy twierdzenia 2.2.

Szukana formuła ϕ:

- Na początku kwantyfikuje egzystencjalnie (tzn. za pomocą $\exists$) po zbiorach $X_1, X_2, \ldots, X_n$, zbiór X_i będzie reprezentować pozycje w słowie na których $\mathcal{A}$ był w stanie o numerze i.
- Następnie wyraża, że $X_i \cap X_j$ jest puste dla $i \neq j$, $1 \in X_1$ oraz dla każdej pozycji istnieje zbiór, w którym ona się znajduje.
- Gdy wiemy już, że $X_1, \ldots, X_n$ są podziałem wszystkich pozycji wystarczy napisać

  $$\forall x.(\psi_1 \vee \psi_2 \vee \ldots \vee \psi_m) \vee \mathrm{last}(x),$$

  gdzie $m = n \cdot |\Sigma|$, a ψ_j odpowiadają kolejnym tranzycjom w taki sposób, że jeśli j–ta tranzycja to (p, a, q), to ψ_j jest formułą

  $$(x \in X_p \wedge a(x) \wedge \mathrm{succ}(x) \in X_q).$$

- Na końcu wystarczy sprawdzić, czy $\exists x.\mathrm{last}(x) \wedge x \in X_f$ dla jakiegoś f będącego numerem stanu akceptującego.

Uwaga 2.10. *Warto zaznaczyć, iż powyższa konstrukcja prowadzi do otrzymania formuły szczególnej postaci (tzw. normalnej), w której kwantyfikatory drugiego rzędu znajdują się tylko na początku formuły i są wyłącznie egzystencjonalne (formuła nie używa $\forall$ kwantyfikujacego po zmiennej drugiego rzędu).*

Przejdźmy do dowodu drugiej implikacji, (2) $\implies$ (1). Dowód przeprowadzimy indukcyjnie po budowie formuły ϕ. Korzystając z uwagi 2.9 możemy ograniczyć się do formuł zbudowanych z $\wedge, \neg, \exists, \leqslant$. Aby móc stosować indukcję po strukturze formuły ϕ, musimy najpierw zdefiniować kilka pojęć.

Ustalmy alfabet Σ, słowo $w \in \Sigma^*$ długości n oraz zbiór zmiennych $\mathcal{X}$ (pierwszego i drugiego rzędu). **Wartościowaniem** zbioru $\mathcal{X}$ na słowie w jest funkcja $F\colon \mathcal{X} \to \{0,1\}^n$ taka, że obrazem każdej zmiennej pierwszego rzędu jest wektor o dokładnie jednej niezerowej współrzędnej (obrazy zmiennych drugiego rzędu mogą przyjmować dowolne wartości przeciwdziedziny). Przykładowo dla słowa $w = aabca$ i zbioru $\mathcal{X} = \{x, y, Z\}$ wartościowaniem jest:

$$F(x) = 00100, \quad F(y) = 01000, \quad F(Z) = 11101,$$

gdzie ciąg $e_1e_2e_3e_4e_5$ jest skrótem notacyjnym oznaczającym wektor $(e_1, e_2, e_3, e_4, e_5) \in \{0,1\}^5$. Dla uproszczenia będziemy pisać:

$$F(x) = 3,\ F(y) = 2,\ F(Z) = \{1, 2, 3, 5\}.$$

Wartościowanie wraz ze słowem nad którym jest rozpatrywane będziemy reprezentować tabelą i traktować jako słowem nad alfabetem $\Sigma_{\mathcal{X}} = \Sigma \times \{0,1\}^{|\mathcal{X}|}$ (literami są kolumny).

x	0	0	1	0	0
y	0	1	0	0	0
Z	1	1	1	0	1
w	a	a	b	c	a

Będziemy używać operatora $\otimes$ do operowania na słowach nad alfabetem $\Sigma_{\mathcal{X}}$. W powyższym przykładzie trzecią literą słowa $w \otimes F$ jest para $(b, [x = 1, y = 0, Z = 1])$. Konkretniej i–ta litera słowa $w \otimes F$ to para $(a, f) \in \Sigma_{\mathcal{X}}$ gdzie $a \in \Sigma$ oraz $f \colon \mathcal{X} \to \{0, 1\}$ jest taka, że $f(x) = 1 \iff F(x) = i$ oraz $f(X) = 1 \iff i \in F(X)$.

Rozważmy formułę *MSO* ϕ składającą się z $\vee, \neg, \exists, \leqslant$ o zmiennych wolnych (czyli takich, do których nie odnosi się żaden kwantyfikator, tzn. nie są kwantyfikowane) $\mathcal{X}$. Zdefiniujmy język $L_{\text{val}}(\phi)$ nad alfabetem $\Sigma_{\mathcal{X}}$ w sposób rekurencyjny. Dla $x, y \in \mathcal{X}$ pierwszego rzędu oraz X drugiego rzędu i wartościowania F zbioru $\mathcal{X}$ zawierającego x, y, X:

- Mając $\phi = x \leqslant y$, definiujemy $L_{\text{val}}(\phi)$ jako zbiór słów postaci $w \otimes F$ w których:

$$F(x) \leqslant F(y).$$

- Jeśli ϕ jest równe $a(x)$, to $L_{\text{val}}(\phi)$ jest zbiorem słów postaci $w \otimes F$, w których:

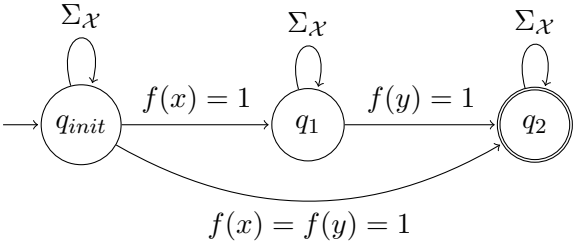
$$w[F(x)] = a.$$

- Dla $\phi = x \in X$ definiujemy $L_{\text{val}}(\phi)$ jako podzbiór $w \otimes F$ w którym $F(x) \in F(X)$.
- $L_{\text{val}}(\phi \vee \psi) = L_{\text{val}}(\phi) \cup L_{\text{val}}(\psi)$.
- $L_{\text{val}}(\neg\phi) = (\Sigma_{\mathcal{X}})^* \setminus L_{\text{val}}(\phi)$.
- Jeśli ϕ jest formułą postaci $\exists z.\psi$, to $L_{\text{val}}(\phi)$ jest zbiorem słów $w \otimes F$, dla których istnieje wartościowanie G rozszerzające F na zbiór $\mathcal{X} \cup \{z\}$ w taki sposób, że $w \otimes G \in L_{\text{val}}(\psi)$.
- Jeśli ϕ jest formułą postaci $\exists Z.\psi$, to $L_{\text{val}}(\phi)$ jest zbiorem słów $w \otimes F$, dla których istnieje wartościowanie G rozszerzające F na zbiór $\mathcal{X} \cup \{Z\}$ w taki sposób, że $w \otimes G \in L_{\text{val}}(\psi)$.

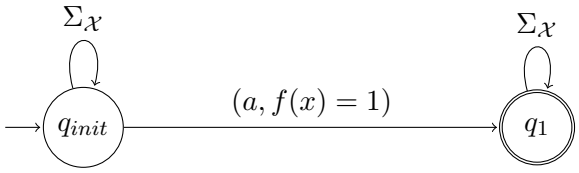
Dla formuł ψ, w których nie występują zmienne wolne, zachodzi $L_{\text{val}}(\psi) = L(\psi)$.

Aby zakończyć dowód, wystarczy indukcyjnie zbudować automat $\mathcal{A}$. Konstrukcję prowadzimy zgodnie z budową formuły. Kolejne tworzone automaty (na ogół niedeterministyczne) opierają się na rozważaniu podformuł rozważanej formuły ϕ oraz ich automatów nad alfabetem $\Sigma_{\mathcal{X}}$. Konkretniej dla formuły:

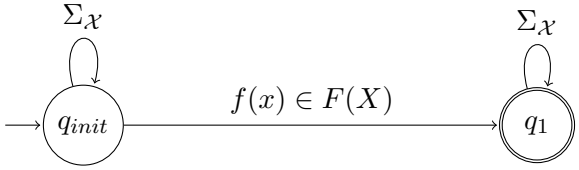
- $x \leqslant y$ tworzymy automat o stanach q_{init}, q_1, q_2, gdzie stanem początkowym jest q_{init} oraz akceptującym q_2. Każdy stan posiada pętlę po dowolnej literze oraz dla $a \in \Sigma$ tranzycje $(q_{init}, (a, f), q_1)$ dla f takich, że $f(x) = 1$, $(q_1, (a, f), q_2)$ dla f spełniających $f(y) = 1$ oraz $(q_{init}, (a, f), q_2)$ dla f spełniających $f(x) = f(y) = 1$.



- $a(x)$ tworzymy automat o stanach q_{init}, q_1 gdzie q_{init} jest początkowy a q_1 akceptujący, posiadający pętle na każdym stanie po dowolnej literze oraz tranzycję $(q_{init}, (a, f), q_1)$ dla f takich, że $f(x) = 1$.



- $x \in X$ tworzymy automat o stanach q_{init}, q_1 gdzie q_{init} jest początkowy a q_1 akceptujący, posiadający pętle na każdym stanie po dowolnej literze oraz tranzycję $(q_{init}, (a, f), q_1)$ dla f takich, że $f(x) \in F(X)$.



- $\phi \vee \psi$ tworzymy automat będący sumą rozłączną (definicja na stronie 16) automatów dla ψ oraz ϕ.

- $\neg\phi$ tworzymy automat rozpoznający dopełnienie $L(\phi)$ (konstrukcja na stronie 18).

- $\exists x.\phi$ tworzymy automat powstały z DFA rozpoznającego $L(\phi)$ poprzez zamianę na tranzycjach (a, f) na (a, g), gdzie g jest obcięciem f z aktualnego $\mathcal{X}$ do $\mathcal{X} \setminus \{x\}$.

- $\exists X.\phi$ tworzymy automat powstały z DFA rozpoznającego $L(\phi)$ poprzez zamianę na tranzycjach (a, f) na (a, g), gdzie g jest obcięciem f z aktualnego $\mathcal{X}$ do $\mathcal{X} \setminus \{X\}$.

■

Zakończyliśmy dowód twierdzenia 2.9, wiemy więc, że logika MSO jest równoważna językom regularnym, to znaczy, że każdą formułę monadycznej logiki drugiego rzędu umiemy przekształcić w automat, oraz dla każdego automatu możemy znaleźć równoważną mu formułę.

Innym ciekawym wynikiem wiążącym logikę (tym razem pierwszego rzędu) z językami regularnymi jest twierdzenie Schützenbergera (2.12), jednak zanim przejdziemy do przedstawienia tego wyniku potrzebować będziemy odrobiny algebry.

2.9. Automaty a półgrupy

W tym podrozdziale zajmiemy się zupełnie odmienną reprezentacją języków regularnych, zanim jednak przejdziemy do twierdzeń, zdefiniujmy kilka pojęć.

1. **Półgrupa** to zbiór S z łącznym działaniem „$\cdot$” nazywanym mnożeniem, to znaczy:

$$\forall a, b, c \in S.(a \cdot b) \cdot c = a \cdot (b \cdot c).$$

2. **Monoid** to półgrupa z elementem neutralnym (oznaczanym przez 1 lub e), czyli takim, który spełnia warunek:
$$\forall s \in S. 1 \cdot s = s \cdot 1 = s.$$

   Uwaga 2.11. *Czasami element neutralny nazywa się jedynką monoidu.*

3. Monoid nazywamy **skończonym**, gdy moc zbioru jego elementów jest skończona.

4. Przykładami monoidów (więc też półgrup) są:

   - $\mathbb{N}$ z dodawaniem i jedynką,
   - $\mathbb{N}$ z działaniem $a \cdot b = \max(a, b)$ oraz zerem,
   - Dla dowolnego zbioru X zbiór X^X wszystkich funkcji $X \to X$ z operacją składania oraz jedynką będącą funkcją identycznościową (oznaczaną $\mathbf{1}_X$),
   - Σ^* z operacją konkatenacji słów i słowem pustym,
   - Dowolna grupa $(G, \cdot, e)$.

   Wszystkie powyższe półgrupy są monoidami, przykładem półgrupy która nie jest monoidem jest $\Sigma^* \setminus \{\varepsilon\}$ z operacją konkatenacji.

5. Dla danych monoidów $(S, \cdot, e_S)$ i $(M, \bullet, e_M)$, **homomorfizmem** z S do M nazywamy każdą funkcję $h\colon S \to M$ spełniającą warunki:
$$\forall s_1, s_2 \in S. h(s_1) \bullet h(s_2) = h(s_1 \cdot s_2), \quad h(e_S) = e_M.$$

   Pierwszą z tych własności nazywamy kompozycjonalnością. Odnotujmy, że w ogólności homomorfizm nie musi być ani injekcją, ani surjekcją.

Lemat 2.8. *Niech M będzie monoidem, a X półgrupą. Rozważmy $h\colon M \to X$ będące kompozycjonalną surjekcją. Istnieje jednoznaczna struktura monoidu na X, dla której h jest homomorfizmem monoidów.*

Dowód. Funkcja h jest kompozycjonalna, więc istnieje:
$$f\colon X \times X \to X$$
taka, że dla każdych $m, n \in M$ spełnione jest:
$$h(m \cdot n) = f(h(m), h(n)).$$
Wystarczy użyć f jako mnożenia w X oraz $h(e_M)$ jako identyczności. ■

Twierdzenie 2.10.

Niech $L \subseteq \Sigma^*$, następujące warunki są równoważne:

1. *L jest językiem regularnym.*
2. *Istnieje monoid **skończony** S, homomorfizm $h\colon \Sigma^* \to S$ oraz podzbiór $K \subseteq S$ taki, że:*
$$L = h^{-1}(K).$$

Dowód. Udowodnimy implikacje w obie strony.

Zacznijmy od implikacji (1) $\rightarrow$ (2). L jest regularny, więc istnieje zupełny DFA rozpoznający L, nazwijmy go $\mathcal{A}$. Nazwijmy jego zbiór stanów Q. Rozważmy monoid S funkcji Q^Q, z działaniem lewostronnego składania funkcji $(f \cdot g)(a) = g(f(a))$. Niech $h(w) = f$, dla takiej funkcji f, że $\forall q \in Q.f(q) = \delta(q, w)$. Funkcja h jest homomorfizmem, ponieważ $h(\varepsilon) = \mathbf{1}_Q$ oraz

$$h(w_1)h(w_2) = h(w_1w_2)$$

jest równoważne

$$\forall q \in Q.\delta(q, w_1w_2) = \delta(\delta(q, w_1), w_2).$$

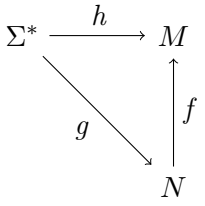
Niech K będzie zbiorem funkcji $f\colon Q \rightarrow Q$ o własności $f(q_{init}) \in F$, gdzie q_{init} to stan początkowy, a F to zbiór stanów akceptujących automatu $\mathcal{A}$. Wiemy, że $h(w) \in K \iff \delta(q_{init}, w) \in F$ z definicji K. Zachodzi również równoważność $\delta(q_{init}, w) \in F \iff w \in L$, więc $h^{-1}(K) = L$.

Przejdźmy teraz do implikacji (2) $\rightarrow$ (1). Niech S będzie monoidem skończonym. Ustalmy podzbiór $K \subseteq S$ oraz homomorfizm $h\colon \Sigma^* \rightarrow S$. Niech $L = h^{-1}(K)$, pokażemy, że L jest regularny poprzez konstrukcję DFA rozpoznającego L. Zdefiniujmy automat $\mathcal{B}$ w następujący sposób: zbiór stanów to zbiór elementów S, funkcja przejścia zadana jest poprzez $\delta(s, a) = s \cdot h(a)$, stan początkowy $q_{init} = h(\varepsilon)$, zbiór stanów akceptujących $F = K$. Na mocy indukcji otrzymujemy, że $h(w) = \delta(q_{init}, w)$. Zachodzą następujące równości:

$$L(\mathcal{B}) = \{w \in \Sigma^* \mid \delta(q_{init}, w) \in K\} = \{w \in \Sigma^* \mid h(w) \in K\} = h^{-1}(K) = L,$$

więc L jest regularny. ■

Twierdzenie 2.11. *Niech $L \subseteq \Sigma^*$. Istnieje monoid M oraz homomorfizm $h : \Sigma^* \rightarrow M$ rozpoznający L, o tej własności, że dla każdego surjektywnego homomorfizmu $g\colon \Sigma^* \rightarrow N$ rozpoznającego L istnieje dokładnie jeden homomorfizm spełniający przemienność poniższego diagramu ($g \circ f = h$).*



Dowód. Rozważmy relację równoważności $\sim_L$ zdefiniowaną jako $w \sim_L w'$ wtw. gdy:

$$\forall v, u \in \Sigma^*.vwu \in L \iff vw'u \in L$$

Uwaga 2.12. *Zdefiniowana właśnie relacja nie jest relacją Mihilla-Nerodego (definicja na stronie 23), chociaż otrzymujemy ją podstawiając $v = \varepsilon$.*

Niech h będzie mapowaniem $w \in \Sigma^*$ na klasę równoważności $\sim_L$. Przyjmijmy mnożenie na klasach równoważności zadane wzorem $[w]_{\sim_L} \cdot [v]_{\sim_L} = [wv]_{\sim_L}$. Funkcja h jest kompozycjonalną surjekcją ($h(w_1) \cdot h(w_2) = h(w_1w_2)$ oraz jest „na"), a więc z lematu 2.8 jest homomorfizmem. Jednoznaczność h wynika wprost z rozróżniania klas $\sim_L$. ■

Powyżej skonstruowany monoid nazywamy **monoidem syntaktycznym** języka L.

Lemat 2.9 (o potędze idempotentnej)**.** *Niech S będzie monoidem skończonym. Dla każdego $s \in S$ istnieje $m \in \mathbb{N}$, takie, że s^m jest idempotentem, czyli $s^m s^m = s^m$, co więcej, jeśli $k \in \mathbb{N}$ i $s^k s^k = s^k$, to $s^k = s^m$ (element jest unikalny).*

Dowód. Ustalmy skończony monoid S oraz jego element $s \in S$. Monoid S jest skończony, więc patrząc na elementy $s, s^2, s^3, \ldots$, dostaniemy powtórzenie. Niech $s^n = s^{n+m}$ będzie pierwszym powtórzeniem, to znaczy, niech (n, m) będzie najmniejszą (leksykograficznie) parą spełniającą tę równość, gdzie $m \neq 0$. Struktura potęg s wygląda jak lasso (najpierw od s do s^{n-1} elementy są różne, od $s^n = s^{n+m}$ do s^{n+m-1} jest pętelka, potem krążymy już tylko po pętelce).

Elementy $s, s^2, \ldots s^{n-1}$ nie mogą być szukaną potęgą s, pozostało więc rozważyć elementy $s^n, s^{n+1}, \ldots s^{n+m-1}$. Równość elementów na pętelce jest równoważna temu, że indeks elementu $q \in \{n, n+1, \ldots, n+m-1\}$ spełnia:

$$q \equiv 2q \pmod{m},$$

co jest równoważne:

$$0 \equiv q \pmod{m}.$$

To równanie ma dokładnie jedno rozwiązanie (ponieważ $|\{n, n+1, \ldots, n+m-1\}| = m$), które daje szukany element $\left(\left\lceil \frac{n}{m} \right\rceil \cdot m\right)$ oraz jego unikalność. ■

Wniosek 2.6. *Niech S będzie monoidem skończonym, $|S| = n$. Dla każdego $s \in S$ element $s^{n!}$ jest unikalną potęgą elementu s (potęgą, a nie wykładnikiem) będącą idempotentem, którą oznaczamy $s^{\#}$.*

Wprowadźmy teraz pojęcie **monoidu aperiodycznego** – jest to taki monoid M, w którym $\forall m \in M. m^{\#} = m^{\#} \cdot m$, co jest równoważne temu, że ciąg $a_n = m^n$ dla każdego $m \in M$ jest stały od pewnego momentu.

Ostatnim potrzebnym nam terminem będzie **wyrażenie regularne typu star-free**, jest to takie wyrażenie regularne, które nie może używać domknięcia Kleenego, za to może używać dopełnień. Niektóre wyrażenia pomimo posiadania gwiazdki w „standardowej formie" dają się definiować jako wyrażenia typu star-free, np. $(ab)^* = (\Sigma^*(aa + bb)\Sigma^* + b\Sigma^* + \Sigma^* a)^c$, przy czym $\Sigma^* = (\varnothing)^c$.

Twierdzenie 2.12 (Schützenberger)**.** *[Sch65] Ustalmy język $L \subseteq \Sigma^*$. Następujące warunki są równoważne:*

1. *L jest definiowalny przez wyrażenie typu star-free,*
2. *L jest definiowalny w logice pierwszego rzędu (FO),*
3. *monoid syntaktyczny języka L jest aperiodyczny.*

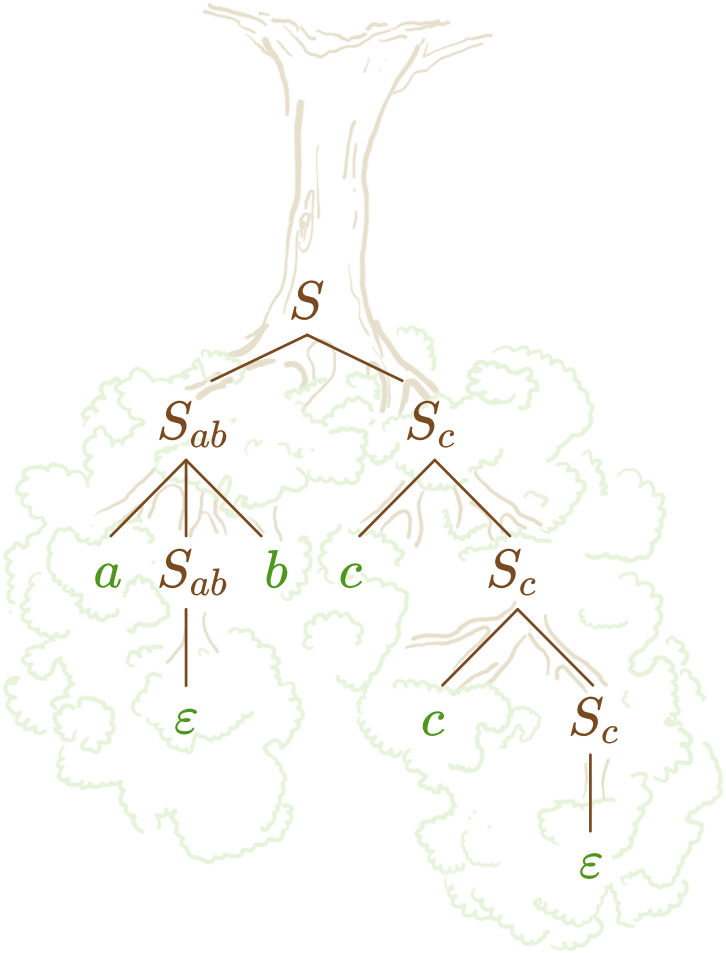
Standardowe dowody tego twierdzenia są dosyć złożone, dlatego pominiemy uzasadnienie prawdziwości przytoczonego twierdzenia.

Rozdział 3

Języki bezkontekstowe

Języki regularne są wygodnym narzędziem do opisywania wielu obiektów, na przykład: numerów telefonów, kodów pocztowych, adresów e-mail czy nazw plików – nie są jednak wystarczające do opisania bardziej złożonych struktur takich jak na przykład składnie większości języków programowania czy poprawnie nawiasujące się wyrażenia arytmetyczne. Do opisu części bardziej skompilowanych struktur posłużą nam *języki bezkontekstowe*.

Jest kilka sposobów patrzenia na języki bezkontekstowe, poprzez drzewo parsowania:



Słowo danego drzewa parsowania uzyskujemy czytając etykiety liści od lewej do prawej.

Możemy też rozważać gramatyki bezkontekstowe, które generują słowa danego języka. Zanim przejdziemy do definicji obu sposobów patrzenia przyjrzyjmy się przykładowi gramatyki bezkontekstowej:

$$\begin{aligned} \mathbf{S} &\to \mathrm{S}_{ab}\mathrm{S}_c \\ \mathrm{S}_{ab} &\to a\mathrm{S}_{ab}b \mid \varepsilon \\ \mathrm{S}_c &\to c\mathrm{S}_c \mid \varepsilon \end{aligned}$$

Za pomocą takiej gramatyki opisującej język $\{a^n b^n c^m \mid n, m \in \mathbb{N}\}$ możemy wyprowadzić słowo *abcc* w następujący sposób:

$$\mathrm{S} \to \mathrm{S}_{ab}\mathrm{S}_c \to a\mathrm{S}_{ab}b\mathrm{S}_c \to a\mathrm{S}_{ab}bc\mathrm{S}_c \to a\mathrm{S}_{ab}bcc\mathrm{S}_c \to abcc\mathrm{S}_c \to abcc$$

3.1. Gramatyki bezkontekstowe

Formalnie **gramatyka bezkontekstowa** $\mathcal{G}$ składa się z:

- Skończonego zbioru terminali Σ (litery języka, jest to odpowiednik alfabetu),
- Skończonego zbioru nieterminali N,
- Nieterminala startowego $S \in N$,
- Skończonego zbioru produkcji P, które graficznie zapisujemy w postaci $X \to w$, gdzie $X \in N$, a $w \in (\Sigma \cup N)^*$ (formalnie taka produkcja to element zbioru $N \times (\Sigma \cup N)^*$).

Zapis $X \to \alpha_1 \mid \ldots \mid \alpha_m$ jest skrótem notacyjnym oznaczającym zbiór produkcji $\{X \to \alpha_1, \ldots, X \to \alpha_m\}$.

Uwaga 3.1. *Warto odnotować, że $X \to \varepsilon$ (słowo puste) jest poprawną produkcją.*

Drzewo parsowania (wyprowadzenia), jest to etykietowane drzewo posiadające w korzeniu nieterminal S, w liściach elementy $\Sigma \cup \{\varepsilon\}$, a w węzłach wewnętrznych etykiety ze zbioru nieterminali N. Dla ustalonego drzewa T oraz jego wierzchołka v etykietę wierzchołka v oznaczamy przez $T(v)$. Dzieci wierzchołka numerujemy od lewej do prawej (zaczynając od 1). Słowem wyprowadzanym przez dane drzewo T nazywamy wynik konkatenacji etykiet (stosowanej od lewej do prawej) liści drzewa T. Jeśli przejścia między wierzchołkami drzewa $(T(v) \to T(v_1) \ldots T(v_n))$, gdzie v_i to kolejne dzieci wierzchołka v, zadane są przez zbiór produkcji P gramatyki $\mathcal{G}$, to mówimy, że słowo w wyprowadzane przez T należy do języka tej gramatyki. Językiem gramatyki jest zbiór wszystkich słów które posiadają swoje drzewa wyprowadzenia w tej gramatyce (lub odpowiednie wyprowadzenia za pomocą gramatyki). Język gramatyki $\mathcal{G}$ oznaczamy $L(\mathcal{G})$.

Język bezkontekstowy *(ang. CFL – context-free language)* jest to język rozpoznawany przez pewną gramatykę bezkontekstową.

Powiemy, że gramatyka jest w **postaci normalnej Chomskyego** gdy każda jej produkcja jest jednej z postaci:

$$X \to a,\ X \to X_1X_2,\ S \to \varepsilon,$$

gdzie $a \in \Sigma$ oraz $X_1, X_2 \in N,\ X_1 \neq S \neq X_2$.

Lemat 3.1 (Postać normalna Chomskyego)**.** *[MRJ83] Niech $\mathcal{G}$ będzie gramatyką bezkontekstową. Istnieje równoważna do $\mathcal{G}$ gramatyka $\mathcal{G}'$ będąca w postaci normalnej Chomskyego.*

Dowód. Konstrukcję gramatyki $\mathcal{G}'$ przeprowadzimy w czterech krokach zachowujących $L(\mathcal{G})$.

1. Jeśli nieterminal S występuje po prawej stronie którejkolwiek produkcji zamieńmy wszystkie wystąpienia S w gramatyce (po lewej i prawej stronie produkcji) na nowy nieterminal S', a następnie dodajmy produkcję $S \to S'$.

2. Następnym krokiem konstrukcji $\mathcal{G}'$ będzie usunięcie produkcji typu $X \to \varepsilon$, dla $X \neq S$. Aby tego dokonać najpierw obliczamy zbiór Z nieterminali które mogą wyprowadzić słowo ε (najpierw do zbioru dodajemy wszystkie nieterminale posiadające produkcję o wyniku ε, następnie aż do ustabilizowania dodajemy do Z wszystkie nieterminale posiadające produkcje o wyniku należącym do Z^*). Sprawdzamy, czy $\varepsilon \in L(\mathcal{G})$, jeśli wynik jest pozytywny, to dodajemy produkcję $S \to \varepsilon$. Iterujemy się po wszystkich tranzycjach, jeśli posiadają jakieś nieterminale ze zbioru Z po prawej stronie, to dodajemy wszystkie produkcje powstające z usunięcia kombinacji wystąpień zmiennych z Z z wyniku tej produkcji (mogą powstać nowe ε– produkcje). Usuwamy wszystkie ε–produkcje nieposiadające S po lewej stronie.

3. Jeśli gramatyka posiada produkcje postaci $X \to Y$, to usuwamy takie produkcje oraz dla każdej produkcji posiadającej co najmniej jedno wystąpienie X dodajemy do gramatyki produkcje podmieniające wystąpienia X na Y (dla każdego podzbioru wystąpień X w takiej produkcji).

4. Dla każdej litery $a \in \Sigma$ stwórzmy nieterminal X_a posiadający tylko jedną produkcję $X_a \to a$.

5. Przekształcamy gramatykę $\mathcal{G}$ do postaci posiadającej tylko produkcje typów:

$$X \to a,\ X \to X_1 X_2 \dots X_k,\ S \to \varepsilon.$$

   Tworząc produkcje $Y_{i,j} \to X_i X_j$ możemy zmniejszać długości formuł typu:

$$X \to X_1 X_2 \dots X_k,$$

   poprzez dodawanie:

$$X \to Y_{1,2} X_3 \dots X_k,$$

   aż osiągną długości dokładnie 2 elementów po prawej stronie. Dostajemy więc gramatykę której każda produkcja jest jednej z postaci:

$$X \to a,\ S \to \varepsilon,\ X \to ZY,$$

   gdzie symbol startowy S nie występuje po prawej stronie.

■

Będziemy dążyć do pokazania własności języków bezkontekstowych, najpierw jednak wskażemy model automatu rozpoznający tę klasę języków.

3.2. Automaty ze stosem

Wprowadzimy nowy model automatu – automat ze stosem *(ang. PDA – push-down automata)*, różni się on od zwykłego automatu tym, że decyzję o wybieranej tranzycji może podjąć na podstawie nie tylko stanu i litery, lecz również informacji z szczytu stosu.

Formalniej ujmując **automat ze stosem** składa się z:

- skończonego alfabetu Σ,
- skończonego zbioru stanów Q,
- zbioru stanów początkowych $I \subseteq Q$,
- zbioru stanów akceptujących $F \subseteq Q$,
- skończonego alfabetu stosowego Γ (zakładamy $\# \in \Gamma$),
- skończonej relacji tranzycji $\delta \subseteq Q \times \Gamma \times (\Sigma \cup \{\varepsilon\}) \times \Gamma^* \times Q$.

Tranzycje są postaci:

$$p \xrightarrow{\text{pop}(Z),a,\text{push}(\gamma)} q$$

Co oznacza, że będąc w stanie $p \in Q$ mając na szczycie stosu symbol $Z \in \Gamma$ po literze $a \in \Sigma$ możemy przejść do stanu $q \in Q$ usuwając element Z ze szczytu stosu a następnie wstawiając na stos kolejne elementy $\gamma \in \Gamma^*$. Jeśli $\gamma[1] = Z$ to zazwyczaj nie piszemy instrukcji pop ani pierwszej litery słowa γ. Będziemy przyjmować, że jeśli γ zawiera symbol $\#$, to tylko jako pierwszą literę. W tranzycji która posiada $\#$ na początku γ zachodzi również $Z = \#$, aby symbol ten zawsze oznaczał dno stosu.

Poprzez **konfigurację** automatu ze stosem rozumiemy aktualny stan oraz cały jego stos, czyli element zbioru $Q \times \Gamma^*$. **Konfiguracją początkową** jest konfiguracja zawierająca stan początkowy oraz stos jednoelementowy – specjalny symbol dna stosu $\# \in \Gamma$. **Konfiguracją akceptującą** jest konfiguracja składająca się ze stanu akceptującego $q_f \in F$ i dowolnego stosu $\gamma \in \Gamma^*$.

Uwaga 3.2. *Należy zwrócić uwagę na fakt, że osiągalnych konfiguracji automatu ze stosem może być nieskończenie wiele, pomimo tego, że zbiory Q i Γ są skończone, ponieważ zawartość stosu jest elementem Γ^*, a ten zbiór jest nieskończony.*

Automat ze stosem jest **deterministycznym automatem ze stosem** (ang. DPDA – deterministic pushdown automaton) jeśli dla każdej konfiguracji i elementu $\Sigma \cup \{\varepsilon\}$ ma co najwyżej jedną tranzycję – czyli jego relacja tranzycji jest funkcją częściową $\delta : Q \times \Gamma \times (\Sigma \cup \{\varepsilon\}) \to \Gamma^* \times Q$ oraz jeśli z danej konfiguracji posiada tranzycję po ε, to nie posiada żadnej innej tranzycji z tej konfiguracji. O DPDA mówimy, że jest **zupełny** jeśli z każdej konfiguracji posiada albo dokładnie jedną tranzycję po ε i żadnej innej, albo dokładnie jedną tranzycję dla każdej litery z Σ.

Podajmy przykład automatu ze stosem rozpoznającego poprawne wyrażenia nawiasowe aby rozjaśnić opisane pojęcia. Automat $\mathcal{A} = \langle \Sigma, Q, I, F, \Gamma, \delta \rangle$, gdzie:

$$\Sigma = \{\mathbf{\{}, \mathbf{\}}\} \quad Q = \{q_{init}, q_1\} \quad I = \{q_{init}\} \quad F = \{q_1\} \quad \Gamma = \{\#, 1\}$$
$$\delta = \{(q_{init}, \#, \varepsilon, \varepsilon, q_1), (q_{init}, \#, \mathbf{\{}, \#1, q_{init}), (q_{init}, 1, \mathbf{\{}, 11, q_{init}), (q_{init}, 1, \mathbf{\}}, \varepsilon, q_{init})\}$$

Powyższy automat rozpoznaje poprawne wyrażenia nawiasowe trzymając na stosie tyle znaków 1 ile nawiasów musi jeszcze zamknąć.

Ważną modyfikacją automatu ze stosem jest zmiana warunku akceptacji na akceptację przez pusty stos. Zazwyczaj PDA akceptują poprzez stany akceptujące, można jednak zmodyfikować definicję akceptacji w taki sposób, że pozbywamy się stanów akceptujących na rzecz akceptowania gdy stos jest pusty (nie ma na nim nic, w szczególności # też został usunięty).

Formalnie **automat ze stosem akceptujący przez stos** jest to piątka

$$\mathcal{A} = \langle \Sigma, Q, I, \Gamma, \delta \rangle,$$

więc od zwykłego automatu ze stosem akceptującego za pomocą stanów akceptujących różni się tylko brakiem zbioru stanów akceptujących. Zmieniamy jednak definicję języka automatu w taki sposób, że

$$\mathrm{L}(\mathcal{A}) = \{w \mid \text{istnieje bieg w automacie } \mathcal{A} \text{ rozpoczynający się w konfiguracji początkowej, a kończący się w konfiguracji z pustym stosem}\}.$$

Jest to równoważna definicja, ponieważ aby przejść z automatu akceptującego przez stan do automatu akceptującego przez pusty stos wystarczy na początku dodać na dół stosu (pod #) nowy specjalny symbol, np. \$, następnie dodać ze stanów akceptujących tranzycję po ε do nowego stanu z pętlą pop'ującą dowolny znak, w drugą stronę mając akceptację przez pusty stos dodajemy nowy stan q_f, który będzie jedynym stanem akceptującym, a wszystkie tranzycje wykonujące pop(#) usuwamy na rzecz identycznych prowadzących do q_f.

Zmodyfikowany warunek pozwala nam rozważać automaty ze stosem posiadające tylko jeden stan, o czym dokładniej mówi poniższy lemat.

Lemat 3.2 (o PDA z 1 stanem)**.** *Niech $\mathcal{A}$ będzie PDA akceptującym przez stany. Istnieje równoważny mu językowo jednostanowy PDA $\mathcal{B}$ akceptujący przez pusty stos.*

Dowód. Jeśli $\mathcal{A}$ akceptuje przez stany, to zamieniamy go na równoważny PDA akceptujący przez pusty stos zgodnie z opisem powyżej. Aby skonstruować PDA $\mathcal{B}$ musimy zapamiętać informację o stanach na stosie, w tym celu zmienimy alfabet stosowy automatu. Literami nowego alfabetu stosowego będą trójki $(p, X, q) \in Q \times \Gamma \times Q$, gdzie Q, Γ to zbiór stanów oraz alfabet stosowy automatu $\mathcal{A}$. Automat $\mathcal{B}$ posiada jeden stan, więc pominiemy go w tranzycjach. Dla każdej tranzycji (p, Z, a, γ, q) automatu $\mathcal{A}$ dodajemy do automatu $\mathcal{B}$ tranzycje wedle przypadków:

- Jeśli $\gamma = \varepsilon$ dodajemy tranzycję $(a, [p, Z, q], \varepsilon)$
- Gdy $\gamma = X$ dla $X \in \Gamma$, dodajemy dla każdego $b \in Q$ tranzycję $(a, [p, Z, b], [q, X, b])$
- Dla $\gamma = X_1 X_2 \dots X_m$ dodajemy dla każdych $b_1, b_2, \dots, b_m \in Q$ tranzycję

$$(a, [p, Z, b_m], [qX_1b_1][b_1X_2b_2] \dots [b_{m-1}X_mb_m])$$

Tranzycje te poza aktualnym stanem i literą stosową dodatkowo zgadują stan w którym automat będzie gdy wróci do tego miejsca na stosie popując literę – jeśli nie uda się zgadnąć, to automat nigdy nie zaakceptuje, ponieważ nie będzie w stanie usunąć tej litery ze stosu, w przypadku umieszczania na stosie więcej niż jednej litery musimy zgadnąć cały ciąg stanów, lecz argumentacja poprawności konstrukcji pozostaje taka sama, wykonujemy jedno zgadnięcie na każdy umieszczony symbol na stosie. ■

Twierdzenie 3.1. *[HMU06, rozdział Pushdown Automata, strony 237–245] Niech $L \subseteq \Sigma^*$. Następujące warunki są równoważne:*

1. *istnieje gramatyka bezkontekstowa $\mathcal{G}$ taka, że $L(\mathcal{G}) = L$,*

2. *istnieje PDA $\mathcal{A}$ taki, że $L(\mathcal{A}) = L$.*

Dowód. Pokażemy implikacje w dwie strony, zacznijmy od (1) $\implies$ (2).

Na początku przekształćmy gramatykę $\mathcal{G}$ do postaci normalnej Chomsky'ego z ewentualnymi dodanymi produkcjami postaci $X \to \varepsilon$ (przekształcenie opisane w lemacie 3.1)

Stworzymy teraz automat $\mathcal{A}$ równoważny gramatyce $\mathcal{G}$ akceptujący przez pusty stos posiadający tylko jeden stan, co będzie wystarczające do zakończenia dowodu pierwszej implikacji. Jako # wybieramy symbol startowy S gramatyki $\mathcal{G}$. Dla każdej produkcji postaci $X \to \varepsilon$ tworzymy tranzycję (pop(X), ε), dla produkcji $X \to a$ tworzymy tranzycję (pop(X),a) natomiast dla produkcji $X \to X_1X_2$ tworzymy tranzycję (pop(X), ε, X_1X_2) (w powyższych tranzycjach pominięte zostały stany i puste operacje pop, push).

Powstały automat odtwarza dokładnie wyprowadzenie za pomocą drzewa parsowania, ponieważ o stosie możemy myśleć w ten sposób, że jeśli posiadamy symbole $X_1, X_2, \dots X_k$ to najpierw musimy skończyć przetwarzanie X_1 dostając jakieś wygenerowane słowo w_1, następnie przetwarzamy X_2 dostając w_2, przetwarzamy po kolei wszystkie symbole ze stosu dostając na końcu słowo $w_1w_2 \dots w_k$.

Aby uzyskać implikację (2) $\implies$ (1) najpierw przekształćmy PDA $\mathcal{A}$ do automatu akceptującego przez pusty stos o dokładnie jednym stanie (lemat 3.2). Zauważmy teraz, że każdą tranzycję postaci (pop(X), a, push(γ)), gdzie $X \in \Gamma,\ a \in (\Sigma), \gamma \in \Gamma^*$ możemy przekształcić na dwie tranzycje (pop(X), ε, push(γP_a)), (pop(P_a), a) gdzie P_a to nowa litera alfabetu stosowego Γ. Po takiej zamianie nasz automat będzie miał tranzycje postaci:

$$(\text{pop}(X), \varepsilon, \text{push}(\gamma)),\ (\text{pop}(X), a)$$

Będziemy mogli więc symulować jego działanie gramatyką bezkontekstową o produkcjach postaci $X \to a,\ X \to \varepsilon,\ X \to X_1X_2 \dots X_k$, co kończy dowód twierdzenia. ■

3.3. Lemat o pompowaniu dla języków bezkontekstowych

Języki bezkontekstowe, tak jak języki regularne, posiadają swoją wersję lematu o pompowaniu. Jest to przydatne narzędzie do dowodzenia, że dany język nie jest bezkontekstowy.

Lemat 3.3 (o pompowaniu dla języków bezkontekstowych)**.** *[HMU06, rozdział Properties of Context-Free Languages, strony 275–276] Niech L będzie językiem bezkontekstowym. Istnieje $N \in \mathbb{N}$ takie, że dla każdego słowa $w \in L$ długości co najmniej N, istnieje podział $w = w_1w_2w_3w_4w_5$ spełniający:*

- $w_2w_4 \neq \varepsilon$,
- $|w_2w_3w_4| \leqslant N$,
- $w_1w_2^kw_3w_4^kw_5 \in L$ *dla każdego* $k \geqslant 0$.

Dowód. Ustalmy gramatykę $\mathcal{G}$ w postaci normalnej Chomskiego (lemat 3.1) taką, że $L(\mathcal{G}) = L$. Jeśli $L = \varnothing$ bądź $L = \{\varepsilon\}$, to teza jest oczywista – załóżmy przeciwnie. Niech p będzie maksymalną długością wyniku (prawej strony) produkcji w $\mathcal{G}$, a m liczbą nieterminali w tej gramatyce, czyli $m = |N_\mathcal{G}|$. Rozważmy $N = p^{m+1} + 1$. Niech w będzie słowem długości co najmniej N należącym do L. Rozważmy dowolne drzewo wyprowadzenia słowa w za pomocą gramatyki $\mathcal{G}$.

Rozważane drzewo ma co najmniej $|w| \geqslant N = p^{m+1} + 1$ liści, więc jego wysokość jest nie mniejsza niż $m + 2$. Istnieje więc ścieżka od liścia do korzenia tego drzewa na której znajduje się powtórzenie jakiegoś nieterminala γ, ponieważ nieterminali na ścieżce długości $m + 2$ lub większej jest przynajmniej $m+1$, a wszystkich nieterminali w gramatyce jest m. Taką sytuację możemy pompować, ponieważ z nieterminala γ możemy wyprodukować $u\gamma v$, dla pewnych słów u, v spełniających $uv \neq \varepsilon$ (korzystamy tu z postaci normalnej Chomskyego, jedyny nieterminal który produkuje słowo zerowej długości to S – występuje on tylko w korzeniu, więc się nie powtarza).

$$\gamma \to u\gamma v \to u \cdot u\gamma v \cdot v \to \ldots \to u^k \gamma v^k$$

Ponadto pompując ostatnie powtórzenie najdłuższej ścieżki możemy uzyskać $|w_2 w_3 w_4| \leqslant N$, ponieważ pompowane wystąpienia znajdują się co najwyżej m produkcji od najdalszych liści w ich poddrzewie. ■

Przykład 3.1. *Język $\{a^n b^n c^n \mid n \in \mathbb{N}\}$ nie jest językiem bezkontekstowym.*

Dowód. Załóżmy przeciwnie, że język $L = \{a^n b^n c^n \mid n \in \mathbb{N}\}$ jest bezkontekstowy. Niech N będzie stałą z lematu o pompowaniu dla języka L. Rozważmy słowo $w = a^N b^N c^N$ oraz jego podział $w = w_1 w_2 w_3 w_4 w_5$ wynikający z tego lematu. Wiemy, że $|w_2 w_3 w_4| \leqslant N$, więc słowo $w_2 w_3 w_4$ może zawierać co najwyżej dwa rodzaje liter, korzystając z faktu $w_2 w_4 \neq \varnothing$ dostajemy, że słowo $v = w_1 w_3 w_5$ spełnia:

$$\max(\#_a(v), \#_b(v), \#_c(v)) = n, \ \min(\#_a(v), \#_b(v), \#_c(v)) < n.$$

Mamy więc sprzeczność, ponieważ z lematu o pompowaniu dla $k = 0$ dostajemy $v \in L$, jednak z powyższego rozumowania dostajemy $v \notin L$. Język L nie jest więc językiem bezkontekstowym. ■

Tak jak dla języków regularnych, istnieją silniejsze wersja lematu o pompowaniu. Wzmacniać możemy na różne sposoby, na przykład poprzez rozważanie dłuższych słów, co wykorzystuje poniższy lemat.

Lemat 3.4 (wzmocniony lemat o pompowaniu dla języków bezkontekstowych)**.** *Niech L będzie językiem bezkontekstowym. Istnieje stała M, taka, że dla każdego $k > 0$ oraz słowa $w \in L$ długości co najmniej M^k istnieje podział $w = ux_1x_2 \ldots x_k z y_k \ldots y_2 y_1 v$ spełniający:*

- $x_i y_i \neq \varepsilon$,
- $|x_1 x_2 \ldots x_k z y_k \ldots y_2 y_1| \leqslant N^k$,
- *Istnieją nieterminal $Z \in N$ oraz wyprowadzenia:*

$$S \to uZv, \ Z \to z, \ Z \to x_i Z y_i \text{ (dla każdego } i\text{).}$$

Dowód. Ustalmy język bezkontekstowy L, niech N będzie stałą z lematu o pompowaniu dla języka L. Biorąc $M = N^k$ możemy uzyskać k krotne powtórzenie tego samego nieterminala na ścieżce drzewa parsowania, z czego wynika powyższy lemat. ■

Uwaga 3.3. *Dla słowa w długości większej niż M^k w dowolnym drzewie wyprowadzenia znajdziemy nieterminal $Z \in N$ posiadający wyprowadzenia zgodne z powyższym lematem.*

Innym kierunkiem wzmocnienia lematu o pompowaniu jest uwzględnianie położenia pompowanego fragmentu w słowie. Takie podejście prezentuje lemat Ogdena.

Lemat 3.5 (Ogden). *[Ogd68] Niech L będzie CFL. Istnieje $N \in \mathbb{N}$ takie, że dla każdego słowa $w \in L$ spełniającego $|w| \geqslant N$ oraz każdego zaznaczenia co najmniej N pozycji ze słowa w istnieje podział $w = uxyzv$ spełniający następujące warunki:*

- *Przynajmniej jeden z fragmentów x, z słowa w posiada co najmniej jedną zaznaczoną pozycję,*
- *xyz posiada co najwyżej N zaznaczonych pozycji,*
- *$ux^kyz^kv \in L$ dla każdego $k \geqslant 0$.*

Dowód. Niech $\mathcal{G}$ będzie gramatyką w postaci normalnej Chomskyego rozpoznającą język L. Niech $N = 2^{|N_\mathcal{G}|+1}$, rozważmy słowo $w \in L$, długości co najmniej N, z zaznaczonymi przynajmniej N pozycjami. Ustalmy drzewo parsowania w w gramatyce $\mathcal{G}$, wybierzmy pewną ścieżkę w tym drzewie od korzenia do liścia wedle następującej reguły: schodzimy zawsze do takiego dziecka (w postaci normalnej Chomskyego zawsze są co najwyżej dwa produkty), które ma przynajmniej połowę zaznaczonych pozycji (liści) w swoim poddrzewie (połowę zaznaczonych liści poddrzewa swojego ojca). Na początku zaznaczonych jest co najmniej $N = 2^{|N_\mathcal{G}|+1}$ pozycji, więc zanim dojdziemy do liścia (tracąc za każdym razem co najwyżej połowę zaznaczonych pozycji) zobaczymy $|N_\mathcal{G}|+1$ nieterminali, uzyskamy więc powtórzenie dające pompowanie. ■

Obserwacja 3.1. *Lemat o pompowaniu jest szczególnym przypadkiem lematu Ogdena, który można otrzymać zaznaczając wszystkie litery rozważanego słowa.*

3.4. Własności języków bezkontekstowych

Języki regularne posiadały różne przydatne własności, można było je sumować, przecinać, dopełniać i nie tylko. Zbadajmy więc języki bezkontekstowe pod tym kątem.

Obserwacja 3.2. *Każdy język regularny jest też bezkontekstowy, ponieważ automat NFA jest specjalnym przypadkiem automatu ze stosem. Co więcej, każdy język regularny jest też rozpoznawany przez deterministyczny automat ze stosem, co wynika z równoważności językowej NFA oraz DFA (twierdzenie 2.2).*

Uwaga 3.4. *Język Σ^* jest regularny, więc też bezkontekstowy.*

Lemat 3.6. *Niech L, K będą językami bezkontekstowymi. Język $L \cup K$ jest bezkontekstowy.*

Dowód. Klasa języków bezkontekstowych jest zamknięta na sumy, ponieważ mając dwie gramatyki $\mathcal{G}_1, \mathcal{G}_2$ możemy stworzyć gramatykę $\mathcal{G}$ której nieterminal startowy S będzie miał tylko dwie produkcje $S \to S_{\mathcal{G}_1}, S_{\mathcal{G}_2}$ wyprowadzające nieterminale startowe tych gramatyk. ■

Lemat 3.7. *Niech K będzie językiem bezkontekstowym, a L regularnym. Język $K \cap L$ jest bezkontekstowy.*

Dowód. Klasa języków bezkontekstowych jest zamknięta na przecięcia z językami regularnymi, ponieważ mając automaty bezkontekstowego języka K oraz regularnego języka L możemy stworzyć stosowy automat produktowy który będzie modyfikował stos zgodnie z automatem języka K, a jako stan trzymał parę stanów – po jednym z każdgo automatu. ■

Uwaga 3.5. *Powyższe rozumowanie pozostaje prawdziwe dla K rozpoznawanego przez DPDA. Wystarczy użyć przy powyższej konstrukcji DFA rozpoznającego L (przecięcie jest wtedy językiem rozpoznawanym przez deterministyczny automat ze stosem).*

Lemat 3.8. *Klasa języków bezkontekstowych nie jest zamknięta ani ze względu na operację przecięcia, ani dopełnienia.*

Dowód. Aby pokazać brak zamkniętości na przecięcia wskażemy przykład – dwa języki CFL których przecięcie nie jest CFL. Języki $\{a^n b^n c^m \mid n, m \in \mathbb{N}\}, \{a^n b^m c^n \mid n, m \in \mathbb{N}\}$ są bezkontekstowe, przykładowa gramatyka pierwszego z nich to:

$$S \to XC, \quad C \to c \mid \varepsilon, \quad X \to aXb \mid \varepsilon,$$

drugi natomiast jest rozpoznawany przez:

$$S \to aSc \mid B, \ B \to bB \mid \varepsilon.$$

Przecięciem tych języków jest $\{a^n b^n c^n \mid n \in \mathbb{N}\}$ – nie jest on jednak językiem bezkontekstowym, co udowodniliśmy w przykładzie 3.1.

Klasa języków bezkontekstowych nie jest zamknięta na dopełnienia, ponieważ jeśli moglibyśmy sumować i dopełniać to uzyskalibyśmy przecięcie:

$$L \cap K = \Sigma \setminus ((\Sigma \setminus L) \cup (\Sigma \setminus K)).$$

Aby uzyskać przykład świadczący o braku zamkniętości na operację dopełnienia można rozważyć język $\{a^n b^n c^n \mid n \in \mathbb{N}\}$ – nie jest on bezkontekstowy, co udowodniliśmy w przykładzie 3.1, jednak jest dopełnieniem języka bezkontekstowego. ■

W rozdziale o językach regularnych dowiedzieliśmy się o związkach liniowych i semiliniowych podzbiorów $\mathbb{N}$ z językami regularnymi. Przyjrzymy się teraz podobnym związkom pewnych podzbiorów $\mathbb{N}^d$ z językami bezkontekstowymi. Zacznijmy od kilku definicji.

Ustalmy alfabet $\Sigma = \{a_1, a_2, \ldots, a_d\}$ oraz język $L \subseteq \Sigma^*$.

- **Obrazem Parikha słowa** w nazywamy wektor $P(w) = (\#_{a_1}(w), \ldots, \#_{a_d}(w)) \in \mathbb{N}^d$.
- **Obrazem Parikha języka** L nazywamy zbiór $P(L) = \{P(w) \mid w \in L\} \subseteq \mathbb{N}^d$.

O zbiorze $X \subseteq \mathbb{N}^d$ powiemy, że jest **liniowy** jeśli istnieją takie $c_0, c_1, \ldots c_d \in \mathbb{N}^d$, że:

$$X = \{c_0 + c_1 t_1 + \ldots + c_d t_d \mid t_1, \ldots, t_d \in \mathbb{N}\}.$$

O zbiorze $X \subseteq \mathbb{N}$ powiemy, że jest **semi-liniowy** jeśli jest skończoną sumą teoriomnogościową zbiorów liniowych, to znaczy istnieją zbiory liniowe $X_1, \ldots, X_n \subseteq \mathbb{N}^d$ takie, że:

$$X = \bigcup_{i=1}^{n} X_i.$$

Uwaga 3.6. *Powyższe definicje podzbiorów liniowych i semi-liniowych zbioru $\mathbb{N}^d$ są uogólnieniem wprowadzonych wcześniej (patrz strona 31) dla podzbiorów $\mathbb{N}$.*

Twierdzenie 3.2 (Parikha)**.** *[Par61] Niech L będzie językiem bezkontekstowym. Obraz Parikha języka L jest zbiorem semi-liniowym. Jeśli $X \subseteq \mathbb{N}^d$ jest zbiorem semi-liniowym to istnieje język regularny K nad alfabetem d literowym taki, że $P(K) = X$.*

Dowód. Druga część twierdzenia jest prostsza do udowodnienia, dlatego od niej zaczniemy. Aby skonstruować język K posiadający obraz X wystarczy ograniczyć się do skonstruowania poszczególnych języków liniowych składających się na X. Ustalmy więc zbiór liniowy

$$M = \{c_0 + c_1 t_1 + \ldots + c_d t_d \mid t_1, \ldots, t_d \in \mathbb{N}\}.$$

Niech $w_0, w_1, \ldots w_d$ będą słowami o obrazach $P(w_0) = c_0, \ldots, P(w_d) = c_d$. Aby skonstruować odpowiedni język należy rozważyć wyrażenie:

$$\mathcal{E} = w_0 w_1^* w_2^* \ldots w_d^*.$$

Oczywiście język $L(\mathcal{E})$ jest regularny oraz $P(L(\mathcal{E})) = M$.

Przejdźmy do pierwszej – trudniejszej części twierdzenia. Jako pierwszy krok skonstruujmy gramatykę $\mathcal{G}$ w postaci normalnej Chomsky'ego spełniającą $L(\mathcal{G}) = L$.

Niech $\mathcal{G} = \langle \Sigma, N, S, P \rangle$. Dla każdego $U \subseteq N$ zdefiniujmy język L_U słów posiadających wyprowadzenie używające zbioru nieterminali równego dokładnie U (nie używają żadnych spoza U oraz używają każdego z U co najmniej raz).

Zauważmy, że zachodzi równość:

$$P(L) = \bigcup_{U \subseteq N} P(L_U),$$

wystarczy więc pokazać, że wszystkie L_U są semi-liniowe. Ustalmy $U \subseteq N, k = |U|$.

Niech M będzie stałą z wzmocnionego lematu o pompowaniu (patrz lemat 3.4) dla języka L_U.

Zdefiniujmy dwa zbiory:

- $F = \{w \in L_U \mid |w| \leqslant M^k\}$,
- $G = \{xy \mid 1 \leqslant |xy| \leqslant M^k,\ \exists Z \in N. Z \Rightarrow_U^* xZy\}$,

gdzie $\Rightarrow_U^*$ oznacza wyprowadzenie używające nieterminali z U (być może nie wszystkich).

Udowodnimy $P(L_U) = P(FG^*)$ pokazując dwa zawierania.

Zacznijmy od $P(L_U) \subseteq P(FG^*)$. Dowód przeprowadzimy indukcyjnie po długości słowa $w \in P(L_U)$. Jeśli $|w| \leqslant M^k$, to z definicji F wiemy, że $w \in F$, więc $P(w) \in P(F) \subseteq P(FG^*)$. W przeciwnym przypadku $|w| > M^k$, więc z wzmocnionego lematu o pompowaniu dla języków bezkontekstowych oraz uwagi 3.3 uzyskujemy, że istnieje nieterminal $X \in U$ oraz ciąg przejść:

$$S \Rightarrow_U^* uXv \Rightarrow_U^* ux_1 X y_1 v \Rightarrow_U^* \ldots \Rightarrow_U^* ux_1 \ldots x_k z y_k \ldots y_1,$$

gdzie $x_i y_i \neq \varepsilon$ dla każdego i.

Zauważmy jednak, że co najmniej jedno z k środkowych przejść możemy usunąć, dalej pozostając w L_U. Jest tak ponieważ $|U \setminus Z| = k - 1$, więc istnieją dwa kolejne przejścia nie posiadające między sobą żadnego unikalnego (nie występującego nigdzie indziej) nieterminala z U. Skracając słowo w ($x_i y_i \neq \varepsilon$, więc faktycznie skracamy) dostajemy możliwość użycia tezy indukcyjnej dla krótszego słowa w'. Skracane $x_i y_i \in G$, więc

$$P(w) = P(x_i y_i) + P(w') \in P(FG^*),$$

co kończy dowód pierwszej inkluzji.

Przejdźmy do drugiej inkluzji – $P(FG^*) \subseteq P(L_U)$. Dowód również przeprowadzimy indukcyjnie, tym razem po długości słowa $w \in FG^*$. Jeśli $w \in F$, to również $w \in L_U$. Ograniczmy się więc do przypadku $w \notin F$. Wiemy, że $w \in FG^*$ oraz $w \notin F$, więc $w = w'xy$ dla pewnych $w' \in FG^*, xy \in G$. Na mocy tezy indukcyjnej wiemy, że $P(w') = P(w'')$ dla pewnego $w'' \in L_U$. Wiemy też, że w'' posiada wyprowadzenie używające wszystkich nieterminali z U, wystarczy więc dodać produkcję $Z \to xZy$ aby otrzymać $P(w) \in P(L_U)$. ■

Wniosek 3.1. *Otrzymujemy następujący, na pierwszy rzut oka nieoczywisty, rezultat. Języki bezkontekstowe nad alfabetem unarnym (jednoliterowym) są regularne, ponieważ odpowiadają im te same zbiory długości słów – semi-liniowe podzbiory* $\mathbb{N}$.

3.5. Własności deterministycznych automatów ze stosem

W odróżnieniu od NFA niedeterministyczne automaty ze stosem nie zawsze posiadają równoważne deterministyczne odpowiedniki – zanim wykażemy ten fakt udowodnimy, że klasa języków rozpoznawanych przez deterministyczne automaty ze stosem (DCFL) posiada następującą własność.

Lemat 3.9. *Klasa DCFL jest zamknięta na dopełnienia.*

Dowód. Niech $\mathcal{A}$ będzie zupełnym DPDA. Niech $\mathcal{B}$ będzie automatem powstałym z $\mathcal{A}$ przez zamianę zbioru stanów akceptujących na zbiór $Q_{\mathcal{A}} \setminus F_{\mathcal{A}}$. Z faktu, że w automacie $\mathcal{A}$ istnieje dokładnie jeden bieg po każdym słowie wynika, że $L(\mathcal{B}) = \Sigma^* \setminus L(\mathcal{A})$, co dowodzi tezy. ■

Lemat 3.10. *Klasa języków rozpoznawanych przez deterministyczne automaty ze stosem jest mniejsza niż klasa języków bezkontekstowych.*

Dowód. Deterministyczne automaty ze stosem są podzbiorem PDA, więc wystarczy wykazać, że klasy te są różne. Korzystając z lematów: 3.8 oraz 3.9 wnioskujemy, że jedna klasa jest zamknięta na dopełnienia, a druga nie – muszą więc być różne. ■

Uwaga 3.7. *Przykładem języka będącego CFL, lecz nie będącego DCFL, jest:*

$$L = L_1 \cup L_2, \text{ gdzie: } L_1 = \{a^n b^n c^m \mid n, m \in \mathbb{N}\},\ L_2 = \{a^n b^m c^n \mid n, m \in \mathbb{N}\}$$

Dowód. Język L jest CFL ponieważ jest sumą języków bezkontekstowych – to, że L_1, L_2 są CFL pokazaliśmy dowodząc lemat 3.8.

Pozostaje wykazać, że nie jest on DCFL – w tym celu pokażemy, że jego dopełnienie nie jest CFL. Będzie to wystarczające, ponieważ jeśli L byłby DCFL, to jego dopełnienie też – ta klasa jest zamknięta na dopełnienia zgodnie z lematem 3.9, z lematu 3.10 wynika jednak, że byłby też CFL – wykazanie, że nie jest to prawdą będzie więc dla nas wystarczające.

Załóżmy nie wprost, że $\Sigma^* \setminus L$ jest CFL. Wtedy również $K = (\Sigma^* \setminus L) \cap \{a^*b^*c^*\}$ jest CFL, ponieważ klasa języków bezkontekstowych jest zamknięta na przecięcia z językami regularnymi zgodnie z lematem 3.7. Język ten wyraża się poprzez:

$$K = \{a^n b^m c^k \mid m \neq n \neq k\},$$

otrzymamy sprzeczność korzystając z lematu Ogdena (3.5).

Niech N będzie stałą z lematu Ogdena. Rozważmy słowo $w = a^N b^{N!} c^{N!+N}$, w którym zaznaczone są wszystkie litery a. Na mocy lematu istnieje podział $w = uxyzv$ spełniający następujące warunki:

- xz posiada co najmniej jedną zaznaczoną pozycję,
- xyz posiada co najwyżej N zaznaczonych pozycji,
- $ux^k yz^k v \in L$ dla każdego $k \geqslant 0$.

Jeśli któreś ze słów x, z posiadałoby dwie różne litery, to pompując dla $k = 2$ otrzymalibyśmy słowo spoza języka $a^*b^*c^*$. Bez starty ogólności załóżmy więc, że x oraz z używają co najwyżej jednej litery każde. Słowo xz posiada jakąś literę a (zaznaczoną), więc co najmniej jedno ze słów x, z jest postaci a^m dla pewnego m spełniającego $0 < m \leqslant N$ (górne ograniczenie wynika z długości bloku liter a w słowie w). Bez straty ogólności niech będzie to x. Jednak teraz możemy rozważyć trzy przypadki:

- z jest postaci a^n (wtedy $n+m \leqslant N$), to pompując dla $k = \frac{N!}{n+m}$ dostajemy sprzeczność,
- z jest postaci b^n, to pompując dla $k = \frac{N!}{m}$ dostajemy sprzeczność,
- jeśli nie zachodzi żaden z powyższych przypadków, wiemy, że z jest postaci c^n – pompując dla $k = \frac{N!-N}{m}$ dostajemy sprzeczność.

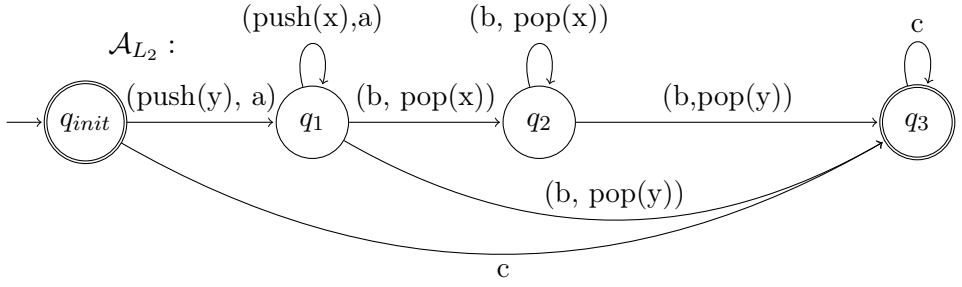
W każdym przypadku otrzymujemy sprzeczność z lematem Ogdena, więc język L nie jest językiem rozpoznawanym przez deterministyczny automat ze stosem. ■

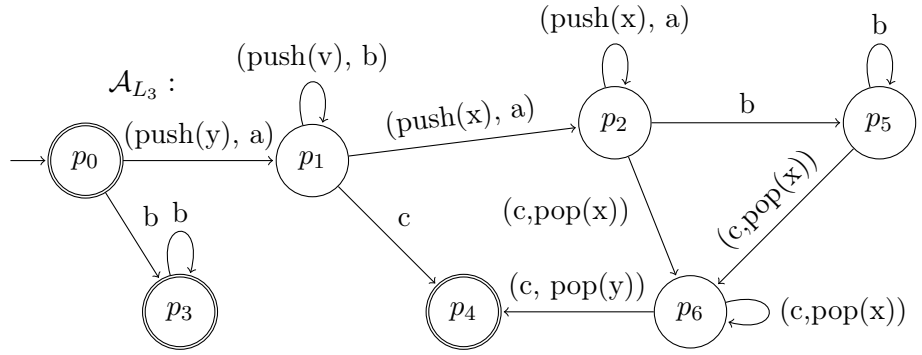
Lemat 3.11. *Klasa DCFL nie jest zamknięta ze względu na operacje: sumy, przecięcia.*

Dowód. Jak pokazaliśmy w przykładzie 3.1 język $L = \{a^n b^n c^n \mid n \in \mathbb{N}\}$ nie jest bezkontekstowy. Pokażmy, że języki:

- $L_1 = \{a^n b^m c^k \mid n, m, k \in \mathbb{N}\}$,
- $L_2 = \{a^n b^n c^k \mid n, k \in \mathbb{N}\}$,
- $L_3 = \{a^n b^m c^n \mid n, m \in \mathbb{N}\}$,

są rozpoznawane przez deterministyczne automaty ze stosem. Język L_1 jest regularny, więc na mocy obserwacji 3.2 posiada pożądaną własność. Przejdźmy do języków L_2 i L_3. Wskażemy automaty rozpoznające te języki:





Skorzystamy z następującej równości:

$$\Sigma^* \setminus L = (\Sigma^* \setminus L_1) \cup (\Sigma^* \setminus L_2) \cup (\Sigma^* \setminus L_3),$$

wynika ona z faktu, że jeśli słowo w nie jest postaci $a^n b^n c^n$, to spełniony jest przynajmniej jeden z poniższych warunków:

- w nie należy do języka generowanego przez wyrażenie $a^* b^* c^*$,
- $\#_a(w) \neq \#_b(w)$,
- $\#_a(w) \neq \#_c(w)$.

Język L nie jest CFL, więc tym bardziej nie jest DCFL. Klasa DCFL jest zamknięta na dopełnienia, więc dopełnienie L również nie jest DCFL. Języki L_1, L_2, L_3 są DCFL, więc ich dopełnienia też. Jeśli klasa DCFL byłaby zamknięta na sumy, to prawa strona równości byłaby DCFL, a lewa nie – otrzymalibyśmy sprzeczność, więc klasa języków rozpoznawanych przez DPDA nie jest zamknięta na sumy. Klasa ta: zawiera $\varnothing, \Sigma^*$, jest zamknięta na dopełnienia oraz nie jest zamknięta na sumy – wynika z tego, że nie jest zamknięta na przecięcia, na mocy prawa De Morgana:

$$L \cap K = \Sigma^* \setminus ((\Sigma^* \setminus L) \cup (\Sigma^* \setminus K))$$

■

3.6. Problemy decyzyjne powiązane z klasami CFL oraz DCFL

W tym podrozdziale będziemy zajmować się szukaniem algorytmów sprawdzających różne własności, bądź też dowodów na to, że posiadania niektórych własności nie da się sprawdzać. Osoby niezaznajomione z pojęciem rozstrzygalności zachęcam do przeczytania dodatków B i C które wprowadzają pojęcia potrzebne do zrozumienia dalszej części tego podrozdziału.

Pierwszym problemem który rozważymy będzie problem polegający na stwierdzeniu, czy język gramatyki bezkontekstowej jest pusty.

Problem 3.1 (**Pustości CFL**). *[HMU06, rozdział Properties of Context-Free Languages, strony 296–298]*

Problem: CFL–EMPTINESS

Dane: Gramatyka bezkontekstowa $\mathcal{G}$

Rozstrzygnąć: Czy $L(\mathcal{G}) = \varnothing$?

Opiszemy procedurę który rozwiązuje powyższy problem w czasie $O(d \cdot |N_\mathcal{G}| \cdot |P_\mathcal{G}|)$, gdzie d jest maksymalną długością wyniku (prawej strony) produkcji w gramatyce $\mathcal{G}$.

Algorytm 3.1 (Pustość CFL)**.**

Dla wygody opisu wprowadźmy następujące pojęcie.

Nieterminalem produktywnym nazywamy nieterminal z którego można wyprodukować jakieś słowo.

Algorytm konstruuje zbiór nieterminali produktywnych *(oznaczony zmienną X)*, a następnie sprawdza czy $S_\mathcal{G} \in X$. Aby skonstruować zbiór X wykorzystywana jest poniższa obserwacja.

Obserwacja 3.3. *Zbiór nieterminali produktywnych gramatyki $\mathcal{G}$ jest punktem stałym przekształcenia $f : P(N_\mathcal{G}) \to P(N_\mathcal{G})$ zadanego formułą:*

$$f(X) = X \cup \{Y \in N_\mathcal{G} \mid \exists (Y \to (\Sigma \cup X)^*) \in P_\mathcal{G}\}$$

iterowanego na argumencie początkowym $\varnothing$, czyli najmniejszym punktem stałym tego przekształcenia.

```
X = ∅;
bool zmiana = true;

while zmiana do
    zmiana = false;
    for (Y → w) ∈ P_G do
        if Y ∉ X and w ∈ (X ∪ Σ)*
            X.dodaj(Y);
            zmiana = true;
// Wynikiem (zbiorem nieterminali produktywnych) jest X.
```

Dowód. Oczywiście jeśli jakiś nieterminal Y znajdzie się w zbierze X to jest produktywny, ponieważ aby wyprodukować z niego słowo możemy najpierw użyć tranzycji która sprawiła, że znalazł się w zbiorze X, a następnie postępować tak z wszystkimi powstałymi nieterminalami, aż otrzymamy słowo składające się wyłącznie z terminali. W drugą stronę, jeśli nieterminal Y jest produktywny, to posiada słowo w które da się z niego wyprodukować. Oczywiście takie wyprowadzenie używa skończonej liczby tranzycji. Załóżmy nie wprost, że Y nie znajduje się w zbiorze X. Rozważmy ostatni moment wyprowadzenia w w którym występuje nieterminal produktywny nie występujący w X, niech takim nieterminalem będzie Z. Zauważmy, że wykonanie dodatkowego obrotu pętli for doda nieterminal Z do X, co daje sprzeczność, ponieważ wiemy, że $Z \notin X$. ■

Następnym problemem będzie problem przynależnościowy. Mając dane słowo w chcemy stwierdzić, czy należy ono do języka podanej gramatyki.

Problem 3.2 (**Przynależności do CFL**). *[HMU06, rozdział Properties of Context-Free Languages, strony 298–302]*

Problem: CFL–MEMBERSHIP

Dane: Słowo w, gramatyka bezkontekstowa $\mathcal{G}$ w postaci normalnej Chomsky'ego.

Rozstrzygnąć: Czy $w \in L(\mathcal{G})$?

Odpowiedź na powyższe pytanie dla słowa w długości n możemy uzyskać w czasie $O(n^3|\mathcal{G}|)$ za pomocą algorytmu Cooke'a, Youngera i Kasamiego.

Algorytm 3.2 (CYK).

Niech $w \in \Sigma^*$. Algorytm konstruuje tablicę wartości logicznych $\mathrm{T}[first][size][X]$ która mówi, czy z nieterminala X da się wyprowadzić słowo $w[first..first + size - 1]$. Aby uzyskać odpowiedź na pytanie czy $w \in L(\mathcal{G})$ należy sprawdzić, czy $\mathrm{T}[1][n][S]$ jest prawdą.

Początkowe wartości $\mathrm{T}[i][1][X]$ algorytm ustala za pomocą sprawdzenia, czy $X \to w[i]$ jest produkcją gramatyki $\mathcal{G}$. W celu obliczenia następnych wartości dokonywana jest iteracja po długości podsłowa (zmienna $size$). Wykorzystywana jest następująca obserwacja.

Obserwacja 3.4. *Jeśli z nieterminala X można wyprowadzić słowo $w[i..j]$, gdzie $j > i$, to wyprowadzenie to zaczyna się od $X \to YZ$, gdzie Y i Z to takie nieterminale, że istnieje k spełniające $i \leqslant k < j$ takie, że Y wyprowadza $w[i..k]$ oraz Z wyprowadza $w[k+1..j]$.*

Dowód. Gramatyka $\mathcal{G}$ jest w postaci normalnej Chomsky'ego, więc wszystkie produkcje posiadające nieterminal po prawej stronie mają postać $X \to YZ$. Nieterminal $S_\mathcal{G}$ nie występuje po prawej stronie żadnej produkcji oraz żaden nieterminal poza $S_\mathcal{G}$ nie może wyprowadzić słowa pustego. Wobec przytoczonych faktów wyprowadzenie musi zacząć się od $X \to YZ$, a każdy z nieterminali Y, Z musi wyprowadzić słowo niezerowej długości, co dowodzi obserwacji. ■

Zgodnie z powyższą obserwacją algorytm iteruje się po wszystkich podziałach i wypełnia tablicę T, następnie sprawdza wartość T[1][n][S]. Przykładowy pseudokod:

```
1  bool check(fst, spl, size, X, Y)
2      return T[fst][spl][X] and T[fst+spl][size-spl][Y];
3
4  for (X → w[j]) ∈ P_G do
5      T[j][1][X] = true;
6
7  for (size = 2; size ⩽ n; size++) do
8      for (first = 1, first ⩽ n - size, first++) do
9          for (split = 1, split ⩽ size - 1, split++) do
10             if (Z → XY) ∈ P_G and check(first, split, size, X, Y)
11                 T[first][size][Z] = true;
12
13 if T[1][n][S]
14     print("OK");
```

Przejdziemy teraz do bardzo ważnego problemu – stwierdzenia czy dana gramatyka bezkontekstowa posiada wyprowadzenie każdego słowa. Zagadnienie to jest o tyle ciekawe, że nie jest rozstrzygalne, a co więcej redukuje się do wielu innych problemów w prosty sposób, co pozwala wykazywać ich nierozstrzygalność.

Problem 3.3 (Pełność CFL). *[HMU06, rozdział Undecidability, strony 407–408]*

Problem: : CFL–UNIVERSALITY

Dane: Gramatyka bezkontekstowa $\mathcal{G}$.

Rozstrzygnąć: Czy $L(\mathcal{G}) = \Sigma^*$?

Pokażemy, że ten problem jest nierozstrzygalny (patrz definicja na stronie 69) korzystając z wniosku C.2, który mówi, że nie istnieje algorytm rozstrzygający czy dana maszyna Turinga $\mathcal{M}$ spełnia $L(\mathcal{M}) = \varnothing$.

Dla danej maszyny Turinga $\mathcal{M}$ skonstruujemy gramatykę bezkontekstową $\mathcal{G}_\mathcal{M}$ rozpoznającą dokładnie te słowa, które nie są poprawnymi kodowaniami biegów akceptujących $\mathcal{M}$. Jeśli $\mathcal{M}$ posiada jakiś bieg akceptujący, to słowo reprezentujące ten bieg nie należy do języka $L(\mathcal{G}_\mathcal{M})$, więc zachodzi $L(\mathcal{G}_\mathcal{M}) = \Sigma^* \iff L(\mathcal{M}) = \varnothing$. Wspomniana konstrukcja dowodzi tego, że problem uniwersalności dla gramatyk bezkontekstowych jest nierozstrzygalny.

Konstrukcję poprzedzimy pewną intuicją. Język:

$$\{w\#w^R \mid w \in \Sigma^*\},$$

jest bezkontekstowy (jest przecięciem języka palindromów z regularnym $\Sigma^*\#\Sigma^*$), lecz język:

$$\{w\#w \mid w \in \Sigma^*\},$$

bezkontekstowy już nie jest (np. z lematu o pompowaniu dla słowa $a^N b^N \# a^N b^N$).

Będziemy więc chcieli zamiast zwykłych biegów maszyny Turinga:

$$\#C_0\#C_1\#\ldots\#C_m\#,$$

kodować odwracane biegi:

$$\#C_0\#C_1^R\#C_2\#C_3^R\#\ldots\#.$$

Chcemy rozpoznawać słowa niebędące kodowaniami biegów akceptujących $\mathcal{M}$, więc szukamy świadectw na to, że dane słowo w nie jest poprawnym odwróconym biegiem. Taka sytuacja może mieć miejsce z czterech powodów:

- ♠ Słowo w nie jest postaci $\#C_0\#C_1^R\#C_2\#C_3^R\#\ldots\#$,
- ♠ Konfiguracja C_0 nie jest konfiguracją początkową $\mathcal{M}$ (na żadnym słowie),
- ♠ Ostatnia konfiguracja – C_m nie jest konfiguracją akceptującą,
- ★ Istnieje i dla którego przejście $C_i \to C_{i+1}$ nie jest poprawnym przejściem maszyny $\mathcal{M}$.

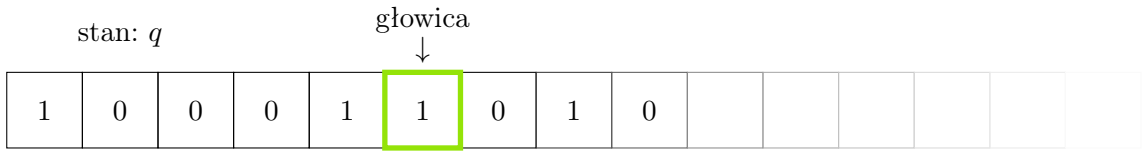
Zauważmy jednak, że warunki oznaczone ♠ są regularne, więc możemy je przeciąć (lemat 3.7) z jedynym warunkiem bezkontekstowym ★ – sprawdzeniem przejść $C_i \to C_{i+1}$, otrzymując język bezkontekstowy (a więc też szukaną gramatyką $\mathcal{G}_\mathcal{M}$).

Dodatek A

Maszyny Turinga

Ludzkość od zarania dziejów konstruuje i projektuje przepisy, receptury i algorytmy. Choć intuicyjnie każdy wie, czy dany tekst jest przepisem na ciasto, czy nie, zazwyczaj podanie definicji pojęcia algorytmu sprawia nam dużo więcej problemów. Prawdziwy przełom w tej tematyce nastąpił za sprawą Alana Turinga, który w roku 1936 podał matematyczną definicję algorytmu. Maszyna Turinga w zamyśle inspirowana jest rachmistrzem, który posiada ołówek z gumką oraz nieskończoną jednowymiarową kartkę papieru (taśmę). Na początku rachmistrz dostaje napisane słowo w, następnie rozpoczyna obliczenie. Oczywiście rachmistrz postępuje zgodnie ze z góry określonym zbiorem reguł (relacją tranzycji) oraz posiada swój wewnętrzny stan obliczenia (stan maszyny). Dodatkowo rachmistrz patrzy na konkretne miejsce swojej kartki (wskazywane przez głowicę), po zaaplikowaniu reguły obliczenia może przesunąć wzrok o jedną kratkę w lewą lub prawą stronę kartki.

Maszyny Turinga (skrót: MT) dostają na wejściu słowa, w wyniku czego wykonują pewne obliczenia, które mogą się skończyć poprzez osiągnięcie stanu końcowego – wynikiem obliczenia jest wtedy słowo zapisane na taśmie, bądź trwać w nieskończoność.



Maszyny Turinga można definiować na wiele sposobów, w wielu wariantach. Za podstawowy wariant maszyny uznamy deterministyczny, jednotaśmowy – w którym taśma jest (jednostronnie) nieskończona w prawo. Składać się będzie z następujących komponentów:

- alfabet wejściowy Σ,
- alfabet roboczy Γ zawierający w sobie alfabet Σ oraz znak $\lrcorner$ (blank) oznaczający puste pole taśmy,
- zbiór stanów Q,
- stan początkowy $q_{init} \in Q$,
- stan końcowy $q_f \in Q$,
- funkcja przejścia $\delta\colon (Q \setminus \{q_f\}) \times \Gamma \to \Gamma \times \{\leftarrow, \bullet, \rightarrow\} \times Q$.

Maszyna wykonuje obliczenie na taśmie będącej (jednostronnie) nieskończonym ciągiem komórek, w każdej komórce znajduje się jedna litera alfabetu Γ. Zbiór słów nieskończonych nad alfabetem Γ oznaczamy przez Γ^ω. Aktualnie rozważane przez maszynę miejsce zaznaczone jest głowicą, jeśli maszyna jest w stanie q, a głowica wskazuje na komórkę z literą a, to możliwe jest użycie tranzycji $t = (p, a, b, d, q)$. Po wykonaniu tranzycji t taśma pod głowicą zawiera literę b (zamazując a), stan maszyny to q, a głowica przesuwa się w lewo, prawo bądź zostaje w miejscu w zależności od wartości $d \in \{\leftarrow, \bullet, \rightarrow\}$. Będąc na początku taśmy próba przemieszczenie się w lewą stronę skutkuje pozostaniem na tej samej pozycji.

Formalnie **konfiguracja maszyny** składa się z:

- stanu $q \in Q$,
- pozycji głowicy $n \in \mathbb{N}$,
- nieskończonego słowa $w \in \Gamma^\omega$ napisanego na taśmie, którego tylko skończona ilość liter jest różna od $\lrcorner$ (tę część można utożsamiać z najkrótszym słowem skończonym będącym prefiksem w zawierającym wszystkie pozycje słowa w różne od $\lrcorner$).

Konfigurację oznaczamy przez $[q, n, w]$. O konfiguracji mówimy, że jest **początkowa** jeśli $q = q_{init}$, natomiast jeśli $q = q_f$ konfigurację nazywamy **końcową**. Tranzycje wykonywane są zgodnie z funkcją przejścia, jeżeli δ pozwala przejść z konfiguracji $[q, n, w]$ do $[p, n', w']$ piszemy:

$$[q, n, w] \to [p, n', w'].$$

Biegiem maszyny po słowie w nazywamy ciąg konfiguracji (być może nieskończony) postaci:

$$c_1 \to c_2 \to c_3 \to \dots,$$

gdzie c_1 jest konfiguracją początkową.

Gdy maszyna $\mathcal{M}$ zaczynając ze słowem w osiągnie konfigurację końcową **wynikiem** obliczenia jest najkrótszy prefiks taśmy zawierający wszystkie litery różne od $\lrcorner$ znajdujące się aktualnie na taśmie. W takiej sytuacji mówimy, że $\mathcal{M}$ **terminuje** na słowie w (odpowiedni bieg określamy terminującym). Maszyna Turinga **oblicza** funkcję częściową $f_{\mathcal{M}} \colon \Sigma^* \rightharpoonup \Gamma^*$, takie funkcje nazywamy **funkcjami obliczalnymi**. Przez $L(\mathcal{M})$ rozumiemy język maszyny $\mathcal{M}$, czyli zbiór słów na których $\mathcal{M}$ terminuje (obliczenie kończy się), takie języki nazywamy **częściowo obliczalnymi**. Często będziemy przyjmować, że $\Sigma = \{0, 1\}$ oraz $\Gamma = \{0, 1, \lrcorner\}$. Wtedy, traktując $\lrcorner$ jako 0, oraz stosując standardowe kodowanie binarne liczb oraz krotek uzyskujemy maszyny obliczające tzw. **obliczalne funkcje liczbowe**, czyli funkcje obliczalne postaci $f \colon \mathbb{N}^k \to \mathbb{N}^n$. Typowo $k = n = 1$.

Okazuje się, że bardzo wiele funkcji można obliczyć za pomocą maszyn Turinga (Teza Churcha–Turinga mówi, że MT obliczają dokładnie to, co języki takie jak C, Python itp.), przykładami takich funkcji są:

- $(a, b) \mapsto a + b$,
- $(a, b) \mapsto a \cdot b$,
- $(a, b) \mapsto a^b$,
- zamiana kodowania binarnego i unarnego (w obie strony),
- $\Sigma^* \mapsto \{0, 1\}$, gdzie $f(w) = 1$ wtw. gdy w to kod maszyny Turinga,
- $\Sigma^* \mapsto \{0, 1\}$, gdzie $f(w) = 1$ wtw. gdy w to konfiguracja maszyny Turinga.

Co ciekawe, maszyny Turinga mają tak dużą moc wyrazu, że potrafią symulować działanie innych maszyn. Maszynę która otrzymawszy na wejściu słowo w oraz kod maszyny $\mathcal{M}$ oblicza wynik uruchomienia $\mathcal{M}$ na słowie w nazywamy **Uniwersalną maszyną Turinga**. Maszyna ta przechowuje na taśmie reprezentację konfiguracji $\mathcal{M}$ i aktualizuje ją „krok po kroku".

Omówimy teraz rozszerzenie maszyn Turinga o dodanie większej ilości taśm. Aby posiadać $k \in \mathbb{N}$ taśm, **wielotaśmowa maszyna Turinga** potrzebować będzie również k głowic. Formalnie funkcja przejścia takiej maszyny jest postaci:

$$\delta \colon (Q \setminus \{q_f\}) \times \Gamma^k \to (\Gamma \times \{\leftarrow, \bullet, \rightarrow\})^k \times Q.$$

Pierwszą taśmę nazywamy **wejściową** – na niej początkowo jest zapisane słowo, ostatnią nazywamy **wyjściową** – z niej odczytywany jest wynik obliczenia. Co ciekawe dodanie dodatkowych taśm, chociaż pomaga zapisywać algorytmy, nie zwiększa siły wyrazu, o czym mówi następujące twierdzenie.

Twierdzenie A.1. *Ustalmy $k \in \mathbb{N}$. Niech $f \colon \Sigma^* \rightharpoonup \Gamma^*$. Następujące warunki są równoważne:*

1. *funkcja f jest obliczana przez jednotaśmową deterministyczną maszynę Turinga,*
2. *funkcja f jest obliczana przez k-taśmową deterministyczną maszynę Turinga.*

Dowód. Przed rozpoczęciem dowodu odnotujmy, że w ogólności rozważana funkcja jest funkcją częściową, więc dla niektórych argumentów może nie przyjmować żadnej wartości.

Aby pokazać równoważność pokażemy dwie implikacje.

Implikacja (1) $\implies$ (2). Konstruujemy maszynę która zamiast przechodzić do stanu końcowego przepisuje wynik z pierwszej taśmy na ostatnią, a następnie terminuje.

Przejdźmy do implikacji (2) $\implies$ (1). Dla danej maszyny k-taśmowej $\mathcal{M}_k$ skonstruujemy maszynę jednotaśmową $\mathcal{M}$ obliczającą funkcję częściową $f_{\mathcal{M}_k}$ symulując działanie maszyny $\mathcal{M}_k$. Aby uzyskać zapowiedziany efekt maszyna $\mathcal{M}$ na swojej taśmie będzie pamiętać konfigurację maszyny $\mathcal{M}_k$. Oczywiście na każdą tranzycję maszyny k-taśmowej będzie przypadać wiele tranzycji maszyny jednotaśmowej. Rozszerzmy alfabet maszyny $\mathcal{M}$ o *litery z podkreśleniami*, będziemy używać ich do oznaczania miejsc na które wskazują głowice maszyny $\mathcal{M}_k$. Oznaczmy przez $u(w_i, n_i)$ słowo w_i z podkreśloną (forma zapisu pozycji głowicy) literą na pozycji n_i. Maszyna $\mathcal{M}$ na taśmie będzie trzymać:

$$q\$u(w_1, n_1)\#u(w_2, n_2)\# \ldots \#u(w_k, n_k)@,$$

gdzie:

- q to aktualny stan maszyny $\mathcal{M}_k$,
- w_i to słowo na i-tej taśmie maszyny $\mathcal{M}_k$,
- n_i to pozycja i-tej głowicy maszyny $\mathcal{M}_k$.

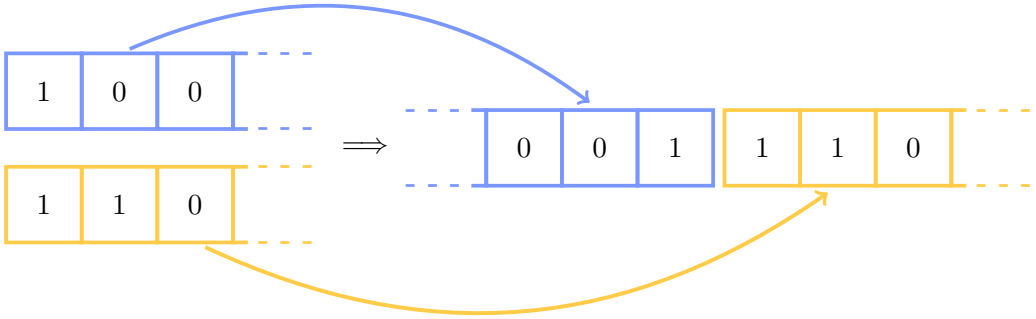
Aby symulować maszynę $\mathcal{M}_k$ maszyna $\mathcal{M}$ kolejno:

1. sprawdza którą tranzycję t maszyny $\mathcal{M}_k$ musi wykonać, w tym celu odczytuje stan q oraz litery znajdujące się na pozycjach podkreślonych zapamiętując wszystkie odczytane informacje w swoim stanie.
2. Zamienia litery podkreślone zgodnie z tranzycją t oraz przesuwa w razie potrzeby podkreślenia. Jeżeli głowica wychodzi w prawą stronę, poza zarezerwowany obszar, należy przesunąć o jedną kratkę w prawo część taśmy znajdującą się po prawej stronie od rozważanego pola.

3. Zamienia stan q na nowy stan p, chyba, że $p = q_f$. W takim wypadku zanim $\mathcal{M}$ zakończy pracę przesuwa całą swoją taśmę w prawo o $|w_k|$ pozycji, przepisuje na początek aktualną zawartość k-tej taśmy $\mathcal{M}_k$ (słowo w_k) oraz czyści taśmę tak, aby przepisane w_k pozostało wynikiem obliczenia. Dopiero po wykonaniu całego procesu „czyszczenia taśmy" terminuje.

■

Wniosek A.1. *Maszyny Turinga z taśmami dwustronnymi również mają tę samą siłę wyrazu, ponieważ możemy symulować taśmę dwustronną dwoma taśmami jednostronnymi traktując je jako „zlepione końcami skończonymi".*



Do tej pory omawialiśmy deterministyczne MT. Tak jak w innych modelach, również ten model posiada niedeterministyczną wersję. **Niedeterministyczna maszyna Turinga** różni się od zdefiniowanej przez nas deterministycznej wersji tym, że funkcja przejścia jest relacją o sygnaturze:

$$\delta \subseteq (Q \setminus \{q_f\}) \times \Gamma \times \Gamma \times \{\leftarrow, \bullet, \rightarrow\} \times Q.$$

Aby dodać niedeterminizm do k-taśmowej MT używamy następującej relacji tranzycji:

$$\delta \subseteq (Q \setminus \{q_f\}) \times \Gamma^k \times (\Gamma \times \{\leftarrow, \bullet, \rightarrow\})^k \times Q.$$

Podobnie do maszyn wielotaśmowych, niedeterministyczne MT również obliczają tę samą klasę języków co deterministyczne jednotaśmowe odpowiedniki.

Twierdzenie A.2. *Ustalmy skończony alfabet Σ oraz język $L \subseteq \Sigma^*$. Następujące warunki są równoważne:*

1. *istnieje deterministyczna MT $\mathcal{M}_D$ taka, że $\mathrm{L}(\mathcal{M}_D) = L$,*
2. *istnieje niedeterministyczna MT $\mathcal{M}_N$ taka, że $\mathrm{L}(\mathcal{M}_N) = L$.*

Dowód. Równoważność wykażemy dowodząc dwie implikacje.

Implikacja (1) $\implies$ (2) jest trywialna, ponieważ deterministyczna MT jest szczególnym przypadkiem niedeterministycznej MT. Zajmijmy się (2) $\implies$ (1).

Ustalmy niedeterministyczną MT $\mathcal{M}_N$. Skonstruujemy deterministyczną maszynę $\mathcal{M}_D$ równoważną językowo maszynie $\mathcal{M}_N$. Maszyna $\mathcal{M}_D$ na początku swojego działania zamienia wejściowe słowo w na konfigurację początkową c_0 maszyny $\mathcal{M}_N$ na słowie w. Od tego momentu $\mathcal{M}_D$ będzie utrzymywać listę konfiguracji do rozważenia na swojej taśmie. W „nieskończonej pętli" czyta pierwszą konfigurację c_i z listy, jeśli posiada ona stan końcowy, to $\mathcal{M}_D$ terminuje. W przeciwnym przypadku maszyna $\mathcal{M}_D$ rozważa wszystkie tranzycje których może

użyć maszyna $\mathcal{M}_N$ będąc w konfiguracji c_i. Dla każdej możliwej do użycia tranzycji dopisuje wynikową konfigurację na koniec przechowywanej listy.

Opisana procedura przeszukuje wszerz drzewo osiągalnych konfiguracji maszyny $\mathcal{M}_N$, wiec jeśli maszyna niedeterministyczna może zakończyć swoje działanie na słowie w, to $\mathcal{M}_D$ też terminuje. W drugą stronę, jeśli $\mathcal{M}_D$ terminuje, to musi istnieć terminujący bieg po słowie w dla maszyny $\mathcal{M}_N$. ■

Uwaga A.1. *Należy zwrócić uwagę na wykonywanie wykładniczo większej liczby kroków względem maszyny $\mathcal{M}_N$ przy przedstawionym sposobie symulowania działania tej maszyny.*

Przyjrzymy się teraz pewnej podklasie maszyn Turinga. Powiemy, że 2-taśmowa MT $\mathcal{M}$ jest **enumeratorem** języka $L \subseteq \Sigma^*$, jeśli spełnia następujące warunki uruchomiona na słowie pustym:

- głowica taśmy wyjściowej $\mathcal{M}$ nigdy się nie cofa (nie przemieszcza się w lewo),
- maszyna $\mathcal{M}$ wypisuje wszystkie słowa $w \in L$ na taśmę wyjściową (oddzielając je specjalnym symbolem $\#$). Maszyna $\mathcal{M}$ musi wypisywać na taśmę wyjściową w taki sposób, że dla każdego $w \in L$ słowo $\#w\#$ kiedyś zostanie wypisane na taśmę oraz dla każdego słowa v niezawierającego znaku $\#$ takiego, że na taśmie pojawi się $\#v\#$ zachodzi $v \in L$.

O językach posiadających enumerator mówimy, że są **rekurencyjnie przeliczalne**.

Dodatek B

Języki obliczalne i częściowo obliczalne

Powiemy, że język $L \subseteq \Sigma^*$ jest **obliczalny**, jeśli istnieje MT która oblicza funkcję charakterystyczną tego języka, to znaczy, że wynikiem jej obliczenia rozpoczętego na słowie w jest:

- 1 – gdy $w \in L$,
- 0 – gdy $w \notin L$.

Tak jak zdefiniowaliśmy w dodatku A, języki **częściowo obliczalne** to te języki, dla których istnieje MT która dla słowa wejściowego w:

- terminuje – gdy $w \in L$,
- nieterminuje, czyli prowadzi obliczenia w nieskończoność – gdy $w \notin L$.

O językach częściowo obliczalnych tak na prawdę usłyszeliśmy już wcześniej, jest to ta sama klasa co klasa języków rekurencyjnie przeliczalnych, czyli posiadających swój enumerator.

Twierdzenie B.1. *Klasa języków częściowo obliczalnych jest równa klasie języków rekurencyjnie przeliczalnych.*

Dowód. Pokażemy dwa zawierania. Ustalmy język L oraz jego enumerator $\mathcal{M}_e$. Skonstruujemy maszynę $\mathcal{M}_t$ terminującą dla słów $w \in L$ oraz pętlącą się dla słów $w \notin L$. Maszyna $\mathcal{M}_t$ symuluje działanie $\mathcal{M}_e$ na słowie pustym, za każdym razem (poza pierwszym) gdy $\mathcal{M}_e$ stawia $\#$ maszyna $\mathcal{M}_t$ sprawdza, czy właśnie wyenumerowane słowo to w. Jeśli tak – terminuje.

W drugą stronę, mając język L oraz maszynę $\mathcal{M}_t$ taką, że $\mathcal{M}_t$ terminuje na słowie w wtedy i tylko wtedy gdy $w \in L$ konstruujemy maszynę $\mathcal{M}_e$ będącą enumeratorem L w następujący sposób. Maszyna $\mathcal{M}_t$ wykonuje kolejne „obroty pętli" (obroty numerujemy kolejnymi liczbami naturalnymi: $1, 2, \ldots$), w i-tym obrocie symulując po kolei i pierwszych kroków obliczenia maszyny $\mathcal{M}_t$ na i pierwszych (w porządku sortującym po długości, a dla słów tej samej długości sortującym leksykograficznie) słowach wejściowych. Jeżeli dla jakiegoś słowa $\mathcal{M}_t$ terminuje, to $\mathcal{M}_e$ wypisuje to słowo na taśmę wyjściową (oddzielając znakiem $\#$). Maszyna $\mathcal{M}_e$ jest enumeratorem języka L ponieważ jeśli $\mathcal{M}_t$ terminuje dla słowa w po n krokach, to podczas n-tego „obrotu pętli" $\mathcal{M}_e$ wypisze słowo $\#w\#$. W drugą stronę, jeśli $\mathcal{M}_t$ nie terminuje na słowie w, to $\mathcal{M}_e$ nigdy nie wypisze na taśmę wyjściową $\#w\#$. ■

Przejdziemy teraz do wykazania lematu służący do stwierdzania czy dany język jest obliczalny. Mając wiedzę o tym czy jest on częściowo obliczalny, oraz czy jego dopełnienie jest częściowo obliczalne. Warto zwrócić uwagę na to, że dowodzenie częściowej obliczalności języka jest często łatwiejsze niż dowodzenie jego obliczalności.

Lemat B.1. *Ustalmy język $L \subseteq \Sigma^*$. Następujące warunki są równoważne:*

- *język L jest obliczalny,*
- *języki $L, L^c = \Sigma^* \setminus L$ są częściowo obliczalne.*

Dowód. Mając maszynę obliczającą L łatwo jest uzyskać częściową obliczalność L oraz L^c, wystarczy w momencie poznania odpowiedzi albo ją zwrócić, albo zamienić na przeciwną, albo zapętlić się, zgodnie z oczekiwanym rezultatem.

Mając dwie maszyny $\mathcal{M}, \mathcal{M}_c$ obliczające częściowo L oraz L^c wystarczy na dwutaśmowej maszynie współbierznie (wykonując kroki na przemian) symulować dwa biegi: $\mathcal{M}$ na pierwszej taśmie, a $\mathcal{M}_c$ na drugiej. Jeśli któraś z maszyn poda odpowiedź (a któraś kiedyś musi), wiemy jakiej odpowiedzi udzielić. ■

Okazuje się, że klasa języków obliczalnych jest zamknięta na niektóre operacje, przyjrzyjmy się tym własnościom.

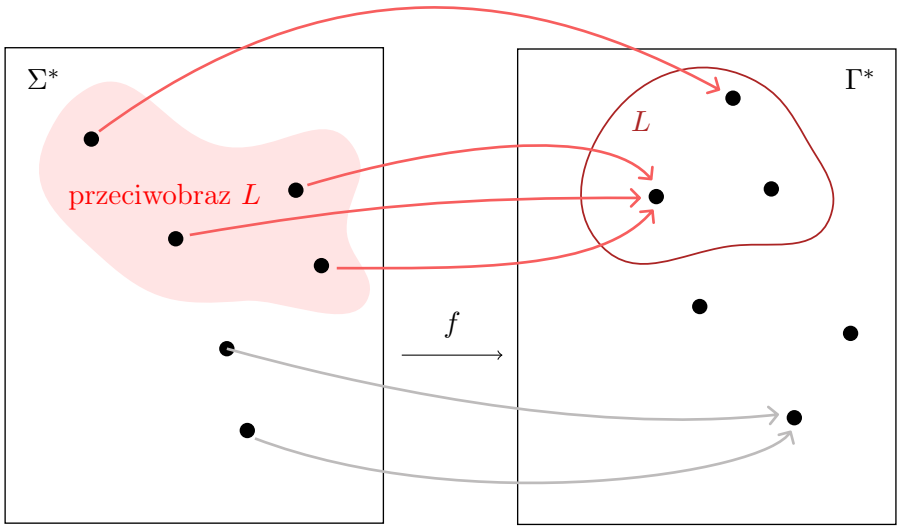
Twierdzenie B.2. *Niech $L, K \subseteq \Sigma^*$ będą językami obliczalnymi. Następujące języki:*

- *$L \cup K$,*
- *$L \cap K$,*
- *$L^c = \Sigma^* \setminus L$,*

również są obliczalne.

Dowód. Niech $\mathcal{M}_L, \mathcal{M}_K$ będą MT obliczającymi funkcje charakterystyczne języków L, K. Aby rozpoznać sumę bądź przecięcie należy otrzymując na wejściu słowo w przesymulować działanie obu maszyn $\mathcal{M}_L, \mathcal{M}_K$ na słowie w i podać odpowiedź zgodną z otrzymanymi wynikami i pożądanym efektem. Aby uzyskać dopełnienie wystarczy zasymulować maszynę $\mathcal{M}_L$, otrzymując wynik x należy na końcu wypisać $1 - x$. ■

Twierdzenie B.3. *Niech $L \subseteq \Gamma^*$ będzie językiem obliczalnym, a $f \colon \Sigma^* \to \Gamma^*$ całkowitą funkcją obliczalną. Język $f^{-1}(L)$ jest obliczalny.*



Dowód. Niech $\mathcal{M}$ oblicza dla danego w czy $w \in L$. Aby obliczać odpowiedzi na pytanie czy $w \in f^{-1}(L)$ należy symulować obliczenie $w \mapsto f(w)$, a następnie zasymulować maszynę $\mathcal{M}$ na $f(w)$, aby uzyskać odpowiedź na pytanie czy $f(w) \in L$. Zachodzi równoważność:

$$f(w) \in L \iff w \in f^{-1}(L).$$

■

Tak jak klasa języków obliczalnych, również klasa języków częściowo obliczalnych jest zamknięta na niektóre operacje, przeanalizujmy teraz tę klasę pod tym kątem.

Twierdzenie B.4. *Niech $L, K \subseteq \Sigma^*$ będą częściowo obliczalne. Następujące języki:*

- $L \cup K$,
- $L \cap K$,

również są częściowo obliczalne.

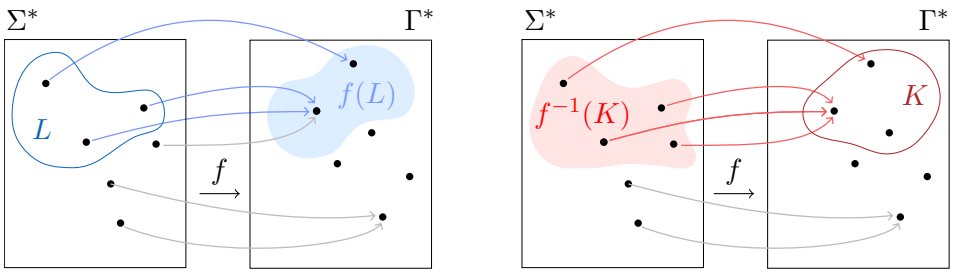
Dowód. Niech $\mathcal{M}_L, \mathcal{M}_K$ będą MT obliczającymi funkcje charakterystyczne języków L, K. Aby rozpoznać sumę bądź przecięcie należy otrzymując na wejściu słowo w symulować „w pętli" działanie obu maszyn $\mathcal{M}_L, \mathcal{M}_K$ na słowie w równolegle (analizując coraz dłuższe biegi tych maszyn). Gdy jedna z maszyn poda odpowiedź twierdzącą, wiemy że słowo należy do $L \cup K$ – możemy więc zakończyć w tym przypadku.

Jeżeli żadna z maszyn nie terminuje, to w obu przypadkach symulacja się nie zakończy, wiec osiągniemy zamierzony cel. Pozostaje przypadek, gdy terminuje jedna z maszyn, niech będzie to $\mathcal{M}_L$, a pytanie na które chcemy odpowiedzieć, to czy $w \in C \cap K$. Zauważmy jednak, że należenie $w \in L$ implikuje:

$$w \in K \iff w \in L \cap K,$$

wystarczy więc kontynuować symulowanie maszyny $\mathcal{M}_K$. ■

Twierdzenie B.5. *Niech $L \subseteq \Sigma^*, K \subseteq \Gamma^*$ będą językami częściowo obliczalnymi. Ustalmy funkcję obliczalną $f\colon \Sigma^* \to \Gamma^*$. Języki $f(L), f^{-1}(K)$ są częściowo obliczalne.*



Dowód. Najpierw zajmiemy się $f^{-1}(K)$. Tak jak dla języków obliczalnych, dostając na wejściu słowo w, najpierw wyliczamy wartość $f(w)$, a następnie symulujemy obliczenie sprawdzające, czy $f(w) \in K$. Wobec następującej równoważności:

$$f(w) \in K \iff w \in f^{-1}(K),$$

otrzymujemy odpowiedź twierdzącą, bądź pętlimy się gdy w istocie jest ona negatywna.

Przejdźmy do zamknięcia na obraz funkcji obliczalnej. Dostając słowo w, chcemy stwierdzić, czy $w \in f(L)$. Iterujemy się więc „w pętli", dla i-tego obrotu pętli rozważamy i pierwszych słów pod względem długości. Dla słowa v obliczamy wartość funkcji f, jeżeli wynosi ona w iterujemy i pierwszych kroków sprawdzenia czy $v \in L$. Jeżeli uzyskamy odpowiedź twierdzącą, to mamy $v \in L$ takie, że $f(v) = w$, więc $w \in f(L)$, terminujemy. Oczywiście jeśli istnieje $v \in L$ spełniające $f(v) = w$, to w pewnym momencie rozważymy odpowiednio długie sprawdzenie na tym słowie. ■

Skończonych napisów jest przeliczalnie wiele, maszyn Turinga jest więc co najwyżej przeliczalnie wiele. Języków – podzbiorów zbioru słów skończonych jest continnum, nie jest więc zaskakującym, że istnieją języki które nie są obliczalne.

O języku powiemy, że jest **nieobliczalny**, jeśli nie istnieje MT która go oblicza.

Dodatek C

Nierozstrzygalność, problem stopu

Teoria obliczeń tak jak każda dziedzina używa swojego „żargonu" – nazewnictwa. Gdy mówimy o algorytmach często używamy pojęć z dziedziny Algorytmiki, wprowadzimy więc teraz stosowną terminologię.

Będziemy mówić o problemach oraz algorytmach rozwiązujących problemy. Wyróżniamy dwa typy problemów: obliczeniowe oraz decyzyjne.

Problemem obliczeniowym nazywamy funkcję $f\colon \Sigma^* \to \Gamma^*$, rozumiemy przez to „zadanie" polegające na skonstruowaniu algorytmu (odpowiedniej MT) który dla wejścia w odpowiada wyjściem $f(w)$ (w szczególności zawsze terminuje).

Problemem decyzyjnym nazywamy funkcję $f\colon \Sigma^* \to \{0,1\}$, bądź równoważnie język $L = f^{-1}(1) \subseteq \Sigma^*$. Rozumiemy przez to „zadanie" polegające na skonstruowaniu algorytmu (odpowiedniej MT) który dla wejścia w odpowiada wartością 1 – gdy $w \in L$ bądź 0 w przeciwnym przypadku.

Instancją problemu nazywamy pojedyncze wejście danego problemu, aby lepiej zrozumieć to pojęcie spójrzmy na przykład dla poniższego problemu:

Problem C.1 (**Pełności CFL**)**.**

Problem: CFL–UNIVERSALITY

Dane: Gramatyka bezkontekstowa $\mathcal{G}$

Rozstrzygnąć: Czy $L(\mathcal{G}) = \Sigma^*$?

Przykładem instancji tego problemu jest (dowolna, konkretna) gramatyka $\mathcal{G}$.

O problemie powiemy, że jest **rozstrzygalny** jeśli zadająca go funkcja jest obliczalna, w przeciwnym przypadku mówimy, że dany problem jest **nierozstrzygalny**.

Mówiliśmy już, że maszyn Turinga jest przeliczalnie wiele wnioskując, że istnieją funkcje nieobliczalne. Z tego samego powodu wiemy, że istnieją problemy nierozstrzygalne, ponieważ różnych problemów jest continuum.

Wykażemy teraz, że poniższy problem jest nierozstrzygalny.

Problem C.2 (**Problem stopu**)**.** *[HMU06, rozdział Undecidability, strony 377–381]*

Problem: HALT

Dane: Maszyna Turinga $\mathcal{M}$ oraz słowo w

Rozstrzygnąć: Czy $\mathcal{M}$ terminuje uruchomiona na słowie w?

Dowód. Załóżmy nie wprost, że istnieje maszyna $\mathcal{M}_H$ która na dowolnym słowie $v \in \Sigma^*$ terminuje i zwraca:

- 1 – gdy $v = w\#kod$, gdzie kod jest kodem MT terminującej na słowie w,
- 0 – w przeciwnym przypadku.

Stwórzmy teraz maszynę $\mathcal{M}_S$ o kodzie $\mathcal{S}$ która dostając na wejściu słowo w działa następująco:

1. zamienia w na $w\#w$,
2. symuluje $\mathcal{M}_H$ na słowie $w\#w$,
3. jeśli wynikiem symulacji jest 0, to terminuje, w przeciwnym przypadku pętli się.

Co stanie się, gdy maszynie $\mathcal{M}_S$ podamy na wejście słowo $\mathcal{S}$?

Jeżeli się zapętli, oznacza to, że na słowie $\mathcal{S}\#\mathcal{S}$ maszyna $\mathcal{M}_H$ zwraca 1, czyli, że maszyna $\mathcal{M}_S$ na słowie $\mathcal{S}$ terminuje. Przeczy to naszemu założeniu.

Z drugiej strony, jeżeli terminuje, oznacza to, że na słowie $\mathcal{S}\#\mathcal{S}$ maszyna $\mathcal{M}_H$ zwraca 0, czyli, że maszyna $\mathcal{M}_S$ na słowie $\mathcal{S}$ pętli się, co znowu daje sprzeczność.

Maszyna $\mathcal{M}_S$ otrzymując jako wejście swój kod $\mathcal{S}$ nie może ani terminować, ani się pętlić, dostajemy więc sprzeczność z założeniem, że istnieje maszyna $\mathcal{M}_H$ rozwiązująca problem stopu. Problem ten jest więc nierozstrzygalny. ■

Wiemy już, że problem stopu nie jest rozstrzygalny, to znaczy, że język

$$Halt = \{w\#kod \mid kod \text{ jest kodem MT } \mathcal{M} \text{ terminującej na słowie } w\}$$

nie jest obliczalny. Jest on jednak częściowo obliczalny, ponieważ maszyna uniwersalna (patrz definicja na stronie 61) oblicza ten język częściowo.

Wniosek C.1. *Dopełnienie języka Halt nie jest częściowo obliczalne, ponieważ jeśli byłoby, to na mocy lematu B.1 język Halt byłby obliczalny.*

Zajmiemy się teraz redukcjami obliczalnymi. **Redukcja obliczalna** problemu $L \subseteq \Sigma^*$ do problemu $K \subseteq \Gamma^*$ jest to funkcja obliczalna $f\colon \Sigma^* \to \Gamma^*$ spełniająca równoważność:

$$w \in L \iff f(w) \in K$$

Technika ta jest przydatna przy badaniu rozstrzygalności ze względu na wykazane w twierdzeniach B.3, B.5 własności zamknięcia ze względu na obrazy i przeciwobrazy przy funkcjach obliczalnych.

Pokażemy teraz nierozstrzygalność innej wersji problemu stopu, przydatnej przy pokazywaniu nierozstrzygalności problemu uniwersalności dla gramatyk bezkontekstowych.

Problem C.3 (**Problem stopu na pustym wejściu**)**.**

Problem: HALT_ε

Dane: Maszyna Turinga $\mathcal{M}$

Rozstrzygnąć: Czy $\mathcal{M}$ terminuje uruchomiona na słowie ε?

Dowód. Wskażemy obliczalną redukcję f problemu Halt do problemu Halt_ε. Wobec nierozstrzygalności problemu Halt uzyskamy nierozstrzygalność problemu Halt_ε, ponieważ jeśli Halt_ε byłby rozstrzygalny, to wobec zamkniętości języków obliczalnych na przeciwobrazy przy funkcjach obliczalnych również Halt musiałby być obliczalny.

Niech $\mathcal{M}_p$ będzie maszyną Turinga która pętli się na każdym wejściu. Redukcja f otrzymując na wejściu słowo v najpierw sprawdza, czy jest ono postaci $w\#kod$ (gdzie kod to poprawny kod MT) – jeżeli tak nie jest to zwraca $\mathcal{M}_p$. Następnie jeśli $v = w\#kod$, gdzie kod to kod MT $\mathcal{M}$, oblicza KOD_w będący kodem MT która kolejno:

1. zapisuje na swojej taśmie słowo w,
2. symuluje obliczenia maszyny $\mathcal{M}$ na zapisanym aktualnie słowie w.

Następnie zwraca jako wynik wartość KOD_w. ■

Wniosek C.2. *Mając daną maszynę Turinga możemy skonstruować analogiczną, która ignoruje swoje wejście (na starcie czyści je i wraca na początek taśmy) i zawsze działa tak jak na słowie pustym. Nie istnieje więc algorytm rozstrzygający czy dana MT $\mathcal{M}$ spełnia $L(\mathcal{M}) = \varnothing$ (jest to dla zmodyfikowanej maszyny pytanie równoważne pytaniu z problemu stopu dla maszyny przed modyfikacją).*

Uwaga C.1. *Problem pustości jest rozstrzygalny dla automatów skończonych oraz gramatyk bezkontekstowych, nie jest jednak rozstrzygalny dla maszyn Turinga.*

Bibliografia

[Ang87] Dana Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, 1987.

[Brz62] Janusz A. Brzozowski. Canonical regular expressions and minimal state graphs for definite events. In *Proceedings of the Symposium on Mathematical Theory of Automata*, pages 529–561. Wiley, 1962.

[Elg61] Calvin C. Elgot. Decision problems of finite automata design and related arithmetics. *Transactions of the American Mathematical Society*, 98(1):11, 1961.

[HMU06] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Greg Tobin, 3 edition, 2006.

[Hop71] John E. Hopcroft. *Theory of machines and computations*, chapter An n log n algorithm for minimizing states in a finite automaton, pages 189–196. New York: Academic Press, 1971.

[MRJ83] Davies Martin, Segal Ron, and Weyuker Elaine J. Computability, complexity, and languages. *Elsevier eBooks*, pages 285–286, Jan 1983.

[Ogd68] William F. Ogden. A helpful result for proving inherent ambiguity. *Mathematical systems theory*, 2:191–194, 1968.

[Par61] Rohit J. Parikh. Language generating devices. *Quarterly Progress Report*, 60, 1961.

[RS59] Michael Oser Rabin and Dana Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3(2):114–125, April 1959.

[Sch65] Marcel Paul Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8(2):190–194, 1965.

[VL08] Antti Valmari and Petri Lehtinen. Efficient minimization of dfas with partial transition functions. *Dans Proceedings of the 25th Annual Symposium on the Theoretical Aspects of Computer Science - STACS, Bordeaux : France*, pages 645–656, 2008.