

Lingwistyka formalna i automaty
Studenckie opracowanie teoretyczne

## Spis treści

---

<b>1 Ćwiczenia nr 1 - Gramatyki</b>	<b>4</b>
1.1 Pojęcia podstawowe . . . . .	4
1.1.1 Język - rodzaje i składowe . . . . .	4
1.1.2 Auto-test . . . . .	4
1.1.3 Problem formalizacji języków . . . . .	5
1.1.4 Auto-test . . . . .	5
1.1.5 Metajęzyk . . . . .	5
1.1.6 Auto-test . . . . .	6
1.2 Metody opisywania składni języka . . . . .	7
1.2.1 Drzewa syntaktyczne . . . . .	7
1.2.2 Notacja BNF . . . . .	7
1.2.3 Auto-test . . . . .	9
1.3 Semisystem Thuego i domknięcie Kleene'ego . . . . .	10
1.3.1 Pojęcia podstawowe . . . . .	10
1.3.2 Alfabet $V$ . . . . .	10
1.3.3 Słowo nad alfabetem $V$ . . . . .	10
1.3.4 Zbiór słów nad alfabetem $V^*$ . . . . .	10
1.3.5 Operacja składania (konkatenacji) . . . . .	10
1.3.6 Słowo puste $\varepsilon$ . . . . .	11
1.3.7 $n$ -ta potęga słowa . . . . .	11
1.3.8 Język nad alfabetem $V$ . . . . .	11
1.3.9 Auto-test . . . . .	12
1.3.10 Semisystem Thuego - definicja . . . . .	12
1.3.11 Auto-test . . . . .	13
1.3.12 Domknięcie Kleene'ego . . . . .	13
1.3.13 Auto-test . . . . .	13
1.4 Gramatyka formalna . . . . .	15
1.4.1 Auto-test . . . . .	16
1.5 Hierarchia Chomsky'ego . . . . .	16
1.5.1 0 - Gramatyki struktur frazowych (kombinatoryczne, nieograniczone) . . . . .	17
1.5.2 1 - Gramatyki kontekstowe . . . . .	17
1.5.3 2 - Gramatyki bezkontekstowe . . . . .	18
1.6 3 - Gramatyki regularne . . . . .	19
1.6.1 Auto-test . . . . .	20
1.7 Język generowany przez gramatykę . . . . .	20
1.7.1 Auto-test . . . . .	21
<b>2 Ćwiczenia nr 2 - Automaty</b>	<b>22</b>
2.1 Automat deterministyczny DAS . . . . .	22
2.1.1 Słowo akceptowane przez DAS . . . . .	22
2.1.2 Język akceptowany przez DAS . . . . .	23
2.1.3 Rozszerzona funkcja przejścia . . . . .	23
2.2 Reprezentacje automatów . . . . .	23

2.2.1	Reprezentacja graficzna automatu . . . . .	23
2.2.2	Reprezentacja tabelaryczna automatu . . . . .	25
2.3	Automat niedeterministyczny NAS . . . . .	27
2.3.1	Słowo akceptowane przez NAS . . . . .	28
2.4	Automat niedeterministyczny $\varepsilon$ NAS . . . . .	29
2.4.1	Słowo akceptowane przez $\varepsilon$ NAS . . . . .	30
2.4.2	Sprawdzanie czy słowo jest akceptowane przez automat DAS, NAS lub $\varepsilon$ NAS przy pomocy rysunku . . . . .	30
<b>3</b>	<b>Ćwiczenia nr 3 - Tworzenie automatów dla języków oraz konwersje NAS → DAS i <math>\varepsilon</math> NAS → DAS</b>	<b>33</b>
3.1	Konwersja NAS do DAS . . . . .	34
3.1.1	Metoda ogólna konwersji NAS do DAS . . . . .	34
3.1.2	Metoda przypośredzona konwersji NAS do DAS . . . . .	37
3.2	Konwersja $\varepsilon$ NAS do DAS . . . . .	38
3.2.1	Epsilon domknięcie $\varepsilon^*$ . . . . .	38
3.2.2	Metoda przypośredzona konwersji $\varepsilon$ NAS do DAS . . . . .	38
<b>4</b>	<b>Ćwiczenia nr 4 - Wyrażenia regularne</b>	<b>41</b>
4.1	Pojęcia wstępne . . . . .	41
4.1.1	Suma dwóch języków . . . . .	41
4.1.2	Złożenie (konkatenacja) . . . . .	41
4.1.3	Domknięcie Kleene'ego (domknięcie) . . . . .	41
4.2	Wyrażenia regularne . . . . .	42
4.2.1	Konwencja notacyjna . . . . .	42
4.2.2	Definicja . . . . .	43
4.3	Priorytety operatorów i symbol $\Sigma$ . . . . .	43
4.4	Równoważność automatów i wyrażeń regularnych . . . . .	44
4.4.1	Jak przekształcać automat aby spełniał warunki ? . . . . .	44
4.4.2	Dowód . . . . .	45
4.5	Konwersja reg-exp $\rightarrow \varepsilon$ NAS . . . . .	48
<b>5</b>	<b>Ćwiczenia nr 5 - UNAS, minimalizacja</b>	<b>52</b>
5.1	Uogólniony niedeterministyczny automat stanowy (UNAS) . . . . .	52
5.2	Konwersja DAS do UNAS . . . . .	52
5.3	Algorytm zamiany UNAS na wyrażenie regularne . . . . .	55
5.3.1	Ogólna idea . . . . .	55
5.3.2	Nieformalny przykład . . . . .	55
5.3.3	Przypomnienie własności wyrażeń regularnych $\emptyset$ oraz $\varepsilon$ . . . . .	56
5.3.4	Przykład 1 . . . . .	57
5.3.5	Przykład 2 . . . . .	58
5.4	Równoważność i minimalizacja automatów . . . . .	60
5.4.1	Równoważność stanów . . . . .	60
5.4.2	Algorytm wyznaczania stanów rozróżnialnych . . . . .	62
5.4.3	Opis algorytmu . . . . .	62
5.4.4	Przykład . . . . .	62
5.4.5	Zastosowania algorytmu tabelkowego . . . . .	65
5.4.6	Sprawdzanie równoważności opisów języka . . . . .	65
5.4.7	Minimalizacja automatu . . . . .	66
5.4.8	Przykład cd. . . . .	66

<b>6 Ćwiczenia numer 6 - CYK</b>	<b>68</b>
6.1 Postać normalna Chomsky'ego . . . . .	68
6.2 Konwersja gramatyki do CNF . . . . .	68
6.2.1 Etap pierwszy . . . . .	69
6.2.2 Drugi etap . . . . .	71
6.3 Stosowanie algorytmu CYK . . . . .	72
<b>7 Uzupełnienie teoretyczne do egzaminu</b>	<b>73</b>
7.1 Gramatyka jednoznaczna i niejednoznaczna . . . . .	73
7.1.1 Drzewo syntaktyczne (parse tree, drzewo parsowania) . . . . .	73
7.1.2 Wyprowadzenie lewostronne i prawostronne . . . . .	74
7.1.3 Definicje . . . . .	75
7.1.4 Eliminowanie niejednoznaczności . . . . .	76
7.1.5 Sprawdzanie jednoznaczności . . . . .	76
7.2 Rozbiór gramatyczny + pojęcia i przykłady . . . . .	76
<b>8 Egzamin - typy zadań + przykłady z lat poprzednich</b>	<b>79</b>
8.1 Zadania teoretyczne . . . . .	79
8.1.1 Metody rozbioru gramatycznego ☐ . . . . .	79
8.1.2 Pojęcia ☐ . . . . .	79
8.1.3 Definicja jednoznaczności ☐ . . . . .	79
8.1.4 Definicja stanów nieroróżnialnych + przykład ☐ . . . . .	79
8.1.5 Klasifikacja Chomsky'ego i jakie ma odniesienie do automatów ☐ . . . . .	80
8.2 Zadania praktyczne . . . . .	81
8.2.1 Skonstruować gramatykę dla... ☐ . . . . .	81
8.2.2 Jaki język generuje gramatyka... ☐ . . . . .	88
8.2.3 Minimalizacja automatu ☐ . . . . .	92
8.2.4 Sprawdzić stosując CYK, czy zdanie ... należy do podanej gramatyki ☐ . . . . .	94
8.2.5 Typ gramatyki generującej $a^k b^l c^m$ w zależności od k,l,m ☐ . . . . .	99
8.2.6 Jaki język generuje G ? Czy jest to gramatyka jednoznaczna ? Jeśli nie, to podać przykład słowa dla którego istnieją dwa wywody. . . . .	101

## 1 Ćwiczenia nr 1 - Gramatyki

---

Jako, że są to pierwsze zajęcia, to opracowanie do tych ćwiczeń rozpoczyna się od informacji ogólnych wprowadzających do tematu lingwistyki. Tematy, które koniecznie należy przyswoić zaczynają się od podrozdziału *Semisystem Thuego i domknięcie Kleen'ego*.

Na tych ćwiczeniach poznamy sposób na formalne zdefiniowanie języka. Formalny opis języka takiego jak C++ czy Java jest bardzo ważny ze względu na jego przetwarzanie, sprawdzanie poprawności wyrażeń napisanych w tym języku. Metodą, którą poznamy będzie opis języka przy pomocy gramatyk, ale zanim to zrobimy musimy poznać parę podstawowych pojęć.

### 1.1. Pojęcia podstawowe

W tym rozdziale wprowadzimy sobie podstawowe definicje związane z lingwistyką formalną i automatami, ograniczymy się do niezbędnego minimum.

#### 1.1.1. Język - rodzaje i składowe

##### Definicja - Język

Pojęcie języka przyjmujemy jako podstawowe i nie definiujemy go formalnie (chocż później powdamy formalną definicję, ale w kontekście gramatyk). Przez język rozumiemy pewien system zorganizowanych znaków służący do utrwalania rezultatów działalności myślowej, a także wzajemnej komunikacji.

Języki dzielimy na naturalne (powstałe w procesie ewolucji biologicznej i społecznej, np. j.polski) i sztuczne (ściśle określony system znaków wraz z regułami postępowania z tymi znakami, a także regułami ich interpretowania np. Java).

Każdy język składa się z trzech elementów:

- składnia - opis reguł konstrukcji języka pozwalających na zbudowanie wszystkich zdań uznanych za poprawne dla danego języka (abstrahuje całkowicie od sensowności tych zdań)
- semantyka - opis znaczenia przypisywany zdaniom poprawnym
- pragmatyka - to zespół reguł i zaleceń mówiących o tym jak stosować składnię i semantykę języka

#### 1.1.2. Auto-test

- Czym jest język ?
- Jak możemy podzielić języki ?
- Czym są języki naturalne, a czym sztuczne ?
- Z czego składa się język ?
- Czym jest składnia, semantyka, pragmatyka języka ?

### 1.1.3. Problem formalizacji języków

To problem formalnego określenia reguł języka.

- Najłatwiej formalizacji poddaje się składnia
- Formalizacja semantyki jest trudniejsza, ale możliwa
- Dla pragmatyki nie ma zadowalających metod formalizacji

Pamiętaj...

Metody definiowania składni języków to

- gramatyki formalne (generacyjne) - tymi będziemy się zajmować na tych ćwiczeniach
- gramatyki dwupoziomowe
- składnia abstrakcyjna
- inne, które będziemy stopniowa poznawać na kolejnych ćwiczeniach...

Pamiętaj...

Dla zainteresowanych, metody definiowania semantyki języków to:

- metoda operacyjna
- metoda denotacyjna
- metoda aksjomatyczna

Tym nie będziemy się w ogóle zajmować na tym przedmiocie.

### 1.1.4. Auto-test

- Co najłatwiej, co trudniej, a co najtrudniej poddaje się formalizacji w problemie formalizacji języków ?
- Jakie znasz metody definiowania składni języków ?

### 1.1.5. Metajęzyk

Języka nie można zdefiniować przy pomocy niego samego ! Konieczny jest do tego metajęzyk.

Definicja - Metajęzyk

Metajęzyk to dowolny język służący do opisu innego języka. Np. dla uczących się języka angielskiego, metajęzykiem może być język polski.

Przy definiowaniu języka wprowadzamy rozróżnienie na symbole terminalne i nieterminalne.

Definicja - Symbol nieterminalny (metajęzyka)

Należy do metajęzyka, jest stosowany do definiowania języka właściwego.

Definicja - Symbol terminalny (języka właściwego)

Należy do języka właściwego (do tego który definiujemy).

#### 1.1.6. Auto-test

- Czym jest metajęzyk ?
- Czym jest symbol nieterminalny ?
- Czym jest symbol terminalny ?

## 1.2. Metody opisywania składni języka

Na tym przedmiocie skupimy się na formalnym definiowaniu składni języków formalnych. Jak już wiemy jedną z metod do tego służących są gramatyki generacyjne. Zanim jednak przejdziemy do gramatyk poznamy dwa mechanizmy, które później będą pośrednio stosowane w gramatykach jako reguły produkcji:

- drzewa syntaktyczne
- notacja BNF

Pamiętaj...

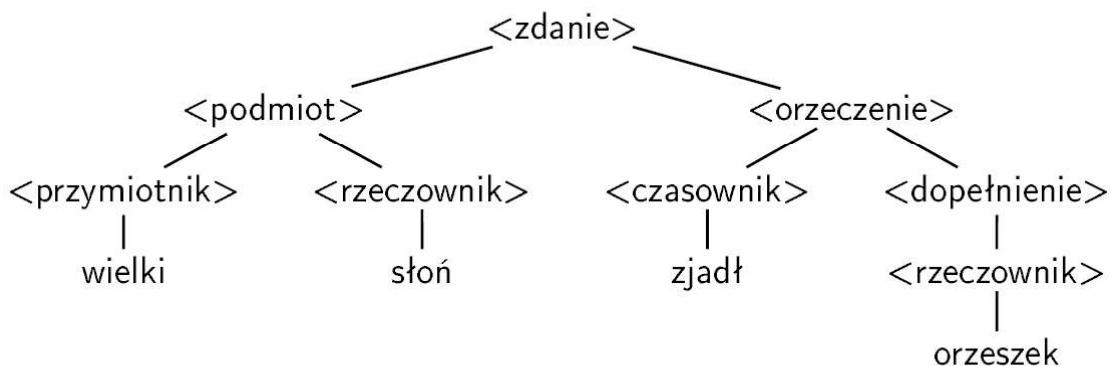
- W obu notacjach symbole metajęzyka (nieterminalne) ujmują się w nawiasy kątowe < oraz >
- Symboli terminalnych nie ujmują się w nawiasy kątowe

### 1.2.1. Drzewa syntaktyczne

Definicja - Drzewo syntaktyczne

Drzewo syntaktyczne, lub drzewo AST (ang. abstract syntax tree) to drzewo etykietowane, gdzie każdy węzeł wewnętrzny reprezentuje pewną konstrukcję języka, a jego synowie znaczące składowe tej konstrukcji.

Drzewo syntaktyczne dla pewnego zdania



Rysunek 1: Przykład drzewa syntaktycznego

### 1.2.2. Notacja BNF

- w notacji BNF wprowadza się symbol produkcji ::=
- po lewej stronie występuje pojedynczy symbol nieterminalny

- po prawej stronie występuje ciąg symboli nieterminalnych i terminalnych

Otrzymana notacja BNF dla rozważanego drzewa

```
<zdanie> ::= <podmiot><orzeczenie>
<podmiot> ::= <przymiotnik><rzeczownik>
<przymiotnik> ::= wielki
<orzeczenie> ::= <czasownik><dopełnienie>
<czasownik> ::= zjadł
<dopełnienie> ::= <rzeczownik>
<rzeczownik> ::= słoń
<rzeczownik> ::= orzeszek
```

Rysunek 2: Przykład BNF

Zaczynając od symbolu  $\langle \text{zdanie} \rangle$ , zastępując kolejne symbole nieterminalne z prawej strony, dochodzimy do samych symboli terminalnych, co możemy odnotować:

$$\langle \text{zdanie} \rangle \xrightarrow{\cdot} \text{wielki słoń zjadł orzeszek}$$

UWAGA: Za każdym razem w ciągu przekształceń możemy stosować tylko jedno przekształcenie.

#### Definicja liczby całkowitej zapisanej w notacji BNF

```
<liczba całkowita> ::= <znak><ciąg cyfr> | <ciąg cyfr>
<ciąg cyfr> ::= <ciąg cyfr><cyfra> | <cyfra>
<znak> ::= + | -
<cyfra> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Rysunek 3: Przykład BNF

### 1.2.3. Auto-test

- Jak zapisujemy symbol nieterminalny, a jak symbol terminalny w AST i BNF ?
- Czym są drzewa syntaktyczne ?
- Czym jest notacja BNF ?

### 1.3. Semisystem Thuego i domknięcie Kleene'ego

Wiemy już, że do opisu składni języków formalnych służą nam gramatyki, ale gramatyki powstały na bazie dwóch wcześniej istniejących pojęć i systemów:

- Gramatyka struktur frazowych
- Semisystem Thuego

Nie będziemy zajmować się gramatyką struktur fazowych (ani tutaj ani na ćwiczeniach tego nie będzie). Jednak ćwiczenia rozpoczynają się od semisystemu Thuego i dlatego najpierw go sobie omówimy, a później zgodnie z ćwiczeniami przejdziemy do gramatyk.

#### 1.3.1. Pojęcia podstawowe

Tutaj wprowadzimy sobie kilka podstawowych pojęć aby zdefiniować system Thuego, pojęcia te będą też konieczne do zdefiniowania gramatyk.

#### 1.3.2. Alfabet $V$

##### Definicja - Alfabet

Alfabetem  $V$  pewnego języka nazywamy dowolny skończony zbiór symboli  $V = \{a_1, a_2, a_3, \dots, a_n\}$ .

#### 1.3.3. Słowo nad alfabetem $V$

##### Definicja - Słowo

Słowem nad alfabetem  $V$  nazywamy dowolny, skończony ciąg z powtórzeniami elementów zbioru  $V$ . Np.  $a_1a_2a_1a_3, a_1, a_2a_3$ . Słowa zazwyczaj oznaczymy małymi literami  $x, y, z$ .

#### 1.3.4. Zbiór słów nad alfabetem $V^*$

##### Definicja - Zbiór słów

Zbiór wszystkich słów nad alfabetem  $V$  oznaczamy  $V^*$ . Oczywiście  $V \subseteq V^*$ . W ogólnym przypadku zbiór  $V^*$  jest zbiorem nieskończonym, chyba, że składa się tylko z symbolu pustego, wówczas jest skończony.

#### 1.3.5. Operacja składania (konkatenacji) $\cdot$

W zbiorze  $V^*$  określana jest operacja składania lub inaczej konkatenacji oznaczana symbolem  $\cdot$  (kropka, tak jak w mnożeniu).

Definicja - Składanie słów

Operację składania słów definiujemy w ten sposób, że jeżeli mamy dane dwa słowa  $x, y \in V^*$  oraz  $x = x_1 \dots x_m$  i  $y = y_1 \dots y_n$ , to złożeniem  $x$  i  $y$  nazywamy takie słowo  $z$ , że  $z = x \cdot y = x_1 \dots x_m y_1 \dots y_n$ .

Dla wygody zapisu symbol konkatenacji można opuścić i wtedy zapisujemy  $z = xy$

**1.3.6. Słowo puste  $\varepsilon$**

Definicja - Słowo puste

Zakładamy, że do zbioru  $V^*$  należy słowo puste  $\varepsilon$ .

Dla każdego słowa  $x \in V^*$  zachodzi, że:

$$x\varepsilon = \varepsilon x = x$$

**1.3.7.  $n$ -ta potęga słowa**

Definicja -  $n$ -ta potęga słowa

$n$ -tą potęgę słowa określamy w oparciu o operację złożenia. Dla dowolnego słowa  $x \in V^*$ :

$$x^0 = \varepsilon$$

$$x^n = x \cdot x^{n-1}$$

Przykład

$$x^3 = x \cdot x^2 = x \cdot x \cdot x^1 = x \cdot x \cdot x \cdot x^0 = x \cdot x \cdot x \cdot \varepsilon = x \cdot x \cdot x$$

**1.3.8. Język nad alfabetem  $V$**

Definicja - Język

Jeżeli  $V$  jest dowolnym alfabetem, to każdy podzbiór  $L$  zbioru  $V^*$  nazywamy językiem nad alfabetem  $V$ , tzn. dowolny podzbiór słów tego alfabetu jest językiem.

Przykład

$$V = \{a, b, c, d\}$$

$$V^* = \{\varepsilon, a, b, c, d, ab, ac, ad, bc, bd, cd, \dots\}$$

to językiem nad alfabetem  $V$  jest przykładowo:

$$L_1 = \{a, b, c\}$$

$$L_2 = \{ac\}$$

$$L_3 = \{ad, bd\}$$

...

### 1.3.9. Auto-test

- Czym jest alfabet ? Jak go oznaczamy ?
- Czym jest słowo nad alfabetem ? Jak go oznaczamy ?
- Czym jest zbiór wszystkich słów nad alfabetem ? Jak go oznaczamy ?
- Czym jest operacji składania ? Jak ją oznaczamy ?
- Czym jest słowo puste ? Jak je oznaczamy i jakie ma własności ?
- Jak definiujemy n-tą potęgę słowa ?
- Czym jest język ?

### 1.3.10. Semisystem Thuego - definicja

Definicja - Semisystem Thuego

Semisystemem Thuego nazywamy system:

$$T = \langle V, P, x \rangle$$

gdzie

$V$  – alfabet

$P \subseteq (V^* - \{\varepsilon\}) \times (V^* - \{\varepsilon\})$  – lista produkcji

$x \in V^*$  – wyznaczone niepuste słowo zwane aksjomatem (slowem startowym)

Lista produkcji, to po prostu zbiór par równoważnych słów, czyli słów, które można stosować zamiennie.

Lista produkcji jest w definicji zapisana jako iloczyn kartezjański słów, czyli są to po prostu pary słów, przykładową produkcją jest więc np.  $(aa, ba)$ , a także  $(abc, cc)$ , ale nie jest już  $(\varepsilon, abc)$ , bo wykluczamy słowo puste w produkcji w systemie Thuego.

Wygodniej i bardziej intuicyjnie w systemie Thuego jest nam jednak zapisać listę produkcji korzystając ze strzałek zamiast par (patrz przykład poniżej).

Przykład

$$T = \langle V, P, x \rangle$$

$$V = \{a, b, c, d\}$$

$$P = \{a \rightarrow b, ab \rightarrow c, c \rightarrow a\}$$

$$x = abc$$

Pamiętajmy, że system Thuego nie zezwala na słowo puste w produkcji.

System Thuego nie był jednak wystarczający, ponieważ mógł zawierać tylko słowa składające się z symboli pochodzących z alfabetu symboli terminalnych, dlatego wprowadzono gramatyki, które są wzbogacone o alfabet metasymboli (symboli nieterminalnych).

### 1.3.11. Auto-test

- Podaj definicję oraz przykład semisystemu Thuego.

### 1.3.12. Domknięcie Kleene'ego

Poznaliśmy już pojęcie zbioru słów nad alfabetem  $V$ , które oznaczamy  $V^*$ . Formalnie zbiór ten definiujemy jako tzw. domknięcie Kleene'ego.

Definicja - Domknięcie Kleene'ego

Domknięciem Kleene'ego zbioru nazywamy sumę mnogościową:

$$V^* = \bigcup_{i=0}^{+\infty} V^i = V^0 \cup V^1 \cup V^2 \cup V^3 \cup \dots$$

*gdzie*

$$V^0 = \varepsilon$$

$$V^{i+1} = \{wv : w \in V^i \wedge v \in V\}$$

Domknięciem Kleene'ego dowolnego alfabetu jest język złożony ze wszystkich słów nad tym alfabetem. Przykładowo jeśli  $A = \{0, 1\}$ , to  $A^*$  jest zbiorem wszystkich ciągów zerojedynkowych o skończonej długości.

Z perspektywy naszych zajęć ważne będzie też wyznaczanie  $V^n$ , gdzie  $n$  będzie dowolną liczbą naturalną.

Przykład

Wyznacz  $V^2$  oraz  $V^3$  dla  $V = \{0, 1\}$ .

$$V^2 = V^2 \cup V^1 \cup V^0$$

$$V^3 = V^3 \cup V^2 \cup V^1 \cup V^0$$

Następnie korzystamy z zależności  $V^{i+1} = \{wv : w \in V^i \wedge v \in V\}$ :

$$\begin{aligned} V^0 &= \{\varepsilon\} \\ V^1 &= \{wv : w \in V^0 \wedge v \in V\} = \{\varepsilon 0, \varepsilon 1\} = \{0, 1\} \\ V^2 &= \{wv : w \in V^1 \wedge v \in V\} = \{00, 01, 10, 11\} \\ V^3 &= \{wv : w \in V^2 \wedge v \in V\} = \{000, 010, 100, 110, 001, 011, 101, 111\} \end{aligned}$$

stąd

$$V^2 = \{\varepsilon, 0, 1, 00, 01, 10, 11\} \quad V^3 = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 010, 100, 110, 001, 011, 101, 111\}$$

### 1.3.13. Auto-test

- Czym jest domknięcie Kleene'ego ? Czym jest dla danego alfabetu ?
- Jak wyznaczyć  $V^n$  ?

Na koniec dodajmy jeszcze dwie definicje, nie trzeba ich znać na pamięć, ważne jednak aby wiedzieć o co chodzi gdy zobaczymy zapis  $z \implies_T y$  lub  $z \implies^*_T y$ .

Niech  $z, y \in V^*$ , słowo  $y$  jest bezpośrednio wyprowadzalne ze słowa  $z$  w systemie  $T$ , co zapisujemy  $z \implies_T y$  gdy istnieją słowa  $z_1, z_2 \in V^*$  oraz produkcja  $(w_1, w_2) \in P$  taka, że  $z = z_1 w_1 z_2$  oraz  $y = z_1 w_2 z_2$ .

Mówimy, że słowo  $y$  jest wyprowadzalne (pośrednio) ze słowa  $z$  w systemie  $T$ , co zapisujemy  $z \implies^*_T y$  gdy istnieje taki ciąg  $z_1, z_2, \dots, z_n \in V^*$ , że

- $z_1 = z, z_n = y$
- $z_1 \implies_T z_{i+1}$  dla  $i = 1, \dots, n-1$

Ciąg  $z_1, z_2, \dots, z_n$  nazywamy wyprowadzeniem słowa  $y$  w  $T$ .

Zbiór wszystkich słów wyprowadzalnych w systemie  $T$  z aksjomatu  $x$  nazywamy *językiem generowanym przez system  $T$* .

## 1.4. Gramatyka formalna

Teraz omówimy sobie gramatyki, jak już wiemy powstały one na bazie systemu Thuego, ponieważ system ten nie był wystarczający. Definicja jest w pewnym sensie rozszerzeniem systemu Thuego. Do definicji systemu Thuego dodajemy bowiem tylko alfabet nieterminalny oraz zakładamy, że po prawej stronie produkcji może występować symbol pusty.

Gramatyka jest zasadniczo jednym ze sposobów definiowania języków formalnych.

### Gramatyka formalna

Gramatyką formalną  $G$  nazywamy czwórkę  $G = \langle N, V, P, S \rangle$ , gdzie:

- 1  $N$  – alfabet nieterminalny (zbiór symboli metajęzyka, zbiór symboli pomocniczych);
- 2  $V$  – alfabet terminalny (zbiór końcowy, symbole języka, symbole z których tworzone są słowa języka określonego przez gramatykę);
- 3  $P$  – zbiór skończony reguł produkcji (lista produkcji, zbiór reguł zastępowania) tj. relacja  $P \subset (N \cup V)^+ \times (N \cup V)^*$ ;
- 4  $S$  – symbol początkowy (aksjomat języka, wyróżniony symbol nieterminalny).

Zwróćmy uwagę na znak + w powyższej definicji produkcji. Oznacza on, że po lewej stronie produkcji nie może być symbolu pustego (tak samo jak u Thuego !), czyli, że nasza suma mnogościowa w domknięciu Kleene'ego zaczyna się od 1, a nie od 0. Natomiast po prawej stronie produkcji już tak nie jest, tam może być symbol pusty (inaczej niż u Thuego !).

Oprócz tego w gramatykach aksjomat może zawierać także symbole nieterminalne, a nie tylko terminalne jak było to u Thuego.

Przykład prostej gramatyki formalnej generującej liczby binarne  
(ciąg zero-jedynkowy)

$$\begin{aligned}
 G &= \langle N, V, P, S \rangle \\
 N &= \{\langle \text{liczba binarna} \rangle, \langle \text{cyfra} \rangle\} \\
 V &= \{0, 1\} \\
 P &= \left\{ \begin{array}{l} \langle \text{liczba binarna} \rangle, \langle \text{cyfra} \rangle \\ \langle \text{liczba binarna} \rangle, \langle \text{liczba binarna} \rangle \langle \text{cyfra} \rangle \\ \langle \text{cyfra} \rangle, 0 \\ \langle \text{cyfra} \rangle, 1 \end{array} \right\} \\
 S &= \langle \text{liczba binarna} \rangle
 \end{aligned}$$

### Użyteczna konwencja

- Często symbole nieterminalne należące do alfabetu  $N$  oznaczać będziemy dużymi literami alfabetu łacińskiego:  $A, B, C, \dots$
- Symbole terminalne należące do  $V$  oznaczać będziemy małymi literami alfabetu łacińskiego:  $a, b, c, \dots$
- Zmienne przebiegające przez alfabet  $N \cup V$  oznaczać będziemy małymi literami greckimi:  $\alpha, \beta, \gamma, \dots$
- Słowa nad alfabetem  $N \cup V$  oznaczać będziemy końcowymi małymi literami łacińskimi:  $x, y, z, \dots$

#### 1.4.1. Auto-test

- Czym jest gramatyka formalna ?

### 1.5. Hierarchia Chomsky'ego

Zdefiniowaliśmy jak narazie gramatykę w sensie ogólnym, teraz wprowadzimy sobie różne rodzaje gramatyk, rodzaje te są ułożone w tzw. hierarchię Chomsky'ego. Numerowanie tej hierarchii rozpoczynamy od 0.

Gramatyki na kolejnych poziomach różnią się pomiędzy sobą tylko ograniczeniami nakładanymi na postać produkcji, które mogą być w nich stosowane. W kolejnych podrozdziałach omówimy sobie ogólnie każdą z klas.

Najbardziej ogólne klasy gramatyk znajdują się na początku tej hierarchii, im wyższy numer tym bardziej szczegółowa gramatyka. Z punktu widzenia języków formalnych interesujące są tylko gramatyki o numerach od 2 włącznie w górę.

Jeżeli więc klasa należy do klasy gramatyk 3, to z pewnością należy też do klasy 0 oraz 1, każda kolejna klasa gramatyk zawiera się bowiem w poprzedniej.

**UWAGA:** Czasami stosowane są inne oznaczenia do opisu danego typu gramatyki (dlaczego? nie wiem, tak już jest), przykładowo gramatykę kombinatoryczną często opisuje się symbolami:  $G = \langle V, \Sigma, P, \sigma \rangle$ , oczywiście symbole te odpowiadają znany nam już pojęciom, które wcześniej zapisywaliśmy jako  $G = \langle N, V, P, S \rangle$ , są jednak zapisane w innej kolejności,  $V$  w obu przypadkach oznacza alfabet terminalny, ale w pierwszym przypadku zapisane jest na pierwszym, a w drugim na drugim miejscu.  $\Sigma$  oraz  $N$  to alfabety nieterminalne.

#### 1.5.1. 0 - Gramatyki struktur frazowych (kombinatoryczne, nieograniczone)

##### 0 - gramatyki struktur frazowych

Zawiera produkcje postaci

$$\alpha \rightarrow \beta$$

gdzie  $\alpha \in (N \cup V)^+$ , a  $\beta \in (N \cup V)^*$ , przy czym nie nakłada się żadnych ograniczeń odnośnie  $\alpha$  i  $\beta$ .

Czyli ogólnie rzecz biorąc nie mamy żadnych ograniczeń na postać produkcji, może być jakkolwiek chcemy, poza oczywistym założeniem, że po lewej stronie nie może być symbolu pustego i wszystkie symbole muszą pochodzić z alfabetów terminalnego lub nieterminalnego.

#### 1.5.2. 1 - Gramatyki kontekstowe

Gramatyka kontekstowa to gramatyka formalna, której reguły są postaci:

$$\alpha A \beta \rightarrow \alpha \gamma \beta,$$

gdzie:

- $A$  – symbol nieterminalny,
- $\alpha, \beta$  – dowolne ciągi symboli terminalnych i nieterminalnych (mogą być puste),
- $\gamma$  – dowolny niepusty ciąg symboli terminalnych i nieterminalnych.

Każda gramatyka kontekstowa definiuje pewien język kontekstowy. Należy zwrócić uwagę, że właściwa reguła to

$$A \rightarrow \gamma,$$

Ciągi  $\alpha$  oraz  $\beta$  stanowią kontekst, w którym dopuszczalne jest zastosowanie tej reguły, stąd właśnie pochodzi nazwa tej klasy gramatyk.

Funkcjonuje również równoważna (z dokładnością do słowa pustego) definicja gramatyki kontekstowej: gramatyką kontekstową nazywa się gramatykę, której reguły są postaci:

$$\alpha \rightarrow \beta,$$

gdzie  $\alpha$  i  $\beta$  są dowolnymi ciągami symboli terminalnych i nieterminalnych spełniającymi warunek:

$$|\alpha| \leq |\beta|$$

gdzie  $|\alpha|$  oznacza liczbę symboli w ciągu  $\alpha$ . Takie gramatyki nazywa się też gramatykami monotonicznymi z uwagi na to, że liczba symboli podczas wyprowadzania słowa nigdy nie maleje.

#### 1.5.3. 2 - Gramatyki bezkontekstowe

### 2 - gramatyki bezkontekstowe

Zawiera produkcje ograniczone do postaci

$$A \rightarrow \pi$$

gdzie  $A \in N$ ,  $\pi \in (N \cup V)^*$ , czyli  $P \subset N \times (N \cup V)^*$ , a więc lewa strona produkcji jest pojedynczym symbolem nieterminalnym.

Gramatyka bezkontekstowa, to dowolna taka czwórka

$G = < N, \Sigma, P, S >$ , gdzie:

- $N$  to (skończony) alfabet symboli nieterminalnych,
- $\Sigma$  to (skończony) alfabet symboli terminalnych,  $\Sigma \cap N = \emptyset$ ,
- $P$  to (skończony) zbiór produkcji, reguł postaci  $A \rightarrow \alpha$  dla  $A \in N, \alpha \in (N \cup \Sigma)^*$ ,
- $S \in N$  to aksjomat wyróżniony symbol nieterminalny

### 1.6. 3 - Gramatyki regularne

#### 3 - gramatyki regularne

Możemy wyróżnić gramatyki prawostronne liniowe i lewostronne liniowe.

Gramatyki prawostronne liniowe zawierają produkcje ograniczone do postaci

$$\begin{aligned} A &\rightarrow a \\ A &\rightarrow aB \end{aligned}$$

gdzie  $A, B \in N$ ,  $a \in V^*$ .

Prawa strona produkcji zawiera albo tylko symbol terminalny, albo symbol terminalny po którym występuje symbol nieterminalny.

Natomiast gramatyki lewostronne liniowe zawierają produkcje ograniczone do postaci

$$\begin{aligned} A &\rightarrow a \\ A &\rightarrow Ba \end{aligned}$$

gdzie  $A, B \in N$ ,  $a \in V^*$ .

Prawa strona produkcji zawiera albo tylko symbol terminalny, albo symbol terminalny przed którym występuje symbol nieterminalny.

Podsumowując:

- Po lewej stronie występuje zawsze dokładnie jeden symbol nieterminalny (tak samo jak w gramatykach bezkontekstowych)
- Po prawej stronie występuje nie więcej niż jeden symbol nieterminalny i dowolny łańcuch symboli terminalnych. W gramatykach prawostronne regularnych symbol terminalny występuje przed nieterminalnym (jeśli ten się tam znajduje), a w lewostronne regularnych na odwrót.

**Przykłady** [[edytuj](#) | [edytuj kod](#)]**poprawne reguły** [[edytuj](#) | [edytuj kod](#)]

$A \rightarrow A$   
 $A \rightarrow B$   
 $A \rightarrow aA$   
 $A \rightarrow aB$   
 $A \rightarrow \epsilon$   
 $A \rightarrow a$   
 $A \rightarrow Ca$  (reguła z gramatyki lewostronnej regularnej)

**reguły niepoprawne** [[edytuj](#) | [edytuj kod](#)]

$AB \rightarrow cD$  (dwa symbole nieterminalne z lewej strony)  
 $A \rightarrow BC$  (dwa symbole nieterminalne z prawej strony)

**1.6.1. Auto-test**

- Jakie znasz klasy gramatyk ?
- Która z tych klas jest najbardziej, a która najmniej ogólna ?
- Czym charakteryzują się gramatyki kombinatoryczne ?
- Czym charakteryzują się gramatyki kontekstowe ?
- Czym charakteryzują się gramatyki bezkontekstowe ?
- Czym charakteryzują się gramatyki regularne ?

**1.7. Język generowany przez gramatykę****Definicja**

Językiem *definowanym*, lub *generowanym*, przez gramatykę  $G$ , co oznaczamy  $L(G)$ , przy czym  $S$  jest symbolem początkowym gramatyki, nazywamy zbiór

$$L(G) = \{x : x \in V^* \wedge S \xrightarrow{*} x\}$$

Z definicji widać, że symbole alfabetu pomocniczego  $N$  nie należą do słów języka  $L(G)$ , ale pojawiają się w wyprowadzeniach tych słów.

**Definicja**

Dwie gramatyki  $G_1$  i  $G_2$  nazywamy *równoważnymi* jeżeli zachodzi, że  $L(G_1) = L(G_2)$ .

Czyli ogólnie rzecz ujmując język generowany przez gramatykę, to język składający się ze wszystkich słów, które mogą powstać z alfabetu nieterminalnego (w tym z symboli alfabetu nieterminalnego samego w sobie), a także ze wszystkich słów, które można wyprowadzić z tego alfabetu przy pomocy produkcji.

#### 1.7.1. Auto-test

- Czym jest język generowany przez gramatykę ?

## 2 Ćwiczenia nr 2 - Automaty

Na tych ćwiczeniach koncentrujemy się na rozwiązyaniu modeli zwanych automatami. Automaty to modele, które będą nam służyć do odpowiedzi na pytanie: *Czy dane słowo należy do języka?* lub równoważnie *Czy dany język akceptuje dane słowo?* Wiemy już jak opisać język przy pomocy gramatyki, teraz stawiamy się w sytuacji gdy chcemy sprawdzić czy jakieś wyrażenie pasuje do naszego języka, dlatego właśnie będziemy rozważać automaty.

Pamiętajmy, że automaty są również modelem, który służy zwyczajnie do opisywania języka (tak jak gramatyki), jest jednak jedna ważna uwaga - automaty mogą opisywać tylko języki regularne, czyli te oznaczone numerem 2 w klasyfikacji Chomskyego. Dodatkowo w dalszej części poznamy różne rodzaje automatów, wszystkie one służą jednak do opisu tylko języków regularnych.

### 2.1. Automat deterministyczny DAS

#### Definicja - DAS

DAS (Deterministyczny Automat Skończeniestanowy) to piątka  $A = (Q, \Sigma, \delta, q_0, F)$ , gdzie:

$Q$  - skończony zbiór stanów

$\Sigma$  - skończony zbiór symboli (alfabet wejściowy)

$\delta : Q \times \Sigma \rightarrow Q$  - funkcja przejścia

$q_0 \in Q$  - stan początkowy

$F \subseteq Q$  - zbiór stanów końcowych (akceptujących)

Patrząc na tą definicję możemy od razu intuicyjnie powiedzieć jak będzie działać nasz model.

1. Początkowo automat jest w  $q_0$
2. Automat pobiera symbol wejściowy i przechodzi do kolejnego stanu zgodnie z funkcją  $\delta$

#### 2.1.1. Słowo akceptowane przez DAS

Na początku tego rozdziału powiedzieliśmy sobie, że automat będzie nam służył do sprawdzania czy dane słowo należy do języka. Automat DAS będzie więc nam służył do opisu postaci słów języka, na wejście automatu podawać będziemy napis i sprawdzać będziemy czy doprowadzi nas do stanu akceptującego (końcowego), jeśli tak, to oznacza, że język akceptuje to słowo.

#### Definicja - Słowo akceptowane przez DAS

Niech  $A = (Q, \Sigma, \delta, q_0, F)$  będzie DAS i niech  $w = w_1 w_2 \dots w_n$  będzie napisem nad  $\Sigma$ . Definiujemy sekwencję stanów  $r_0, r_1, r_2, \dots, r_n$ , taką, że  $r_0 = q_0$ .

$$r_{i+1} = \delta(r_i, w_{i+1}), \text{ gdzie } i = 0, 1, 2, \dots, n-1$$

- Jeżeli ostatni z tych stanów  $r_n \in F$ , wtedy mówimy, że  $A$  akceptuje  $w$
- Jeżeli  $r_n \notin F$ , wtedy mówimy, że  $A$  nie akceptuje  $w$

### 2.1.2. Język akceptowany przez DAS

Możemy również sprawdzać nie tylko czy pojedyncze słowo jest akceptowane przez DAS, ale także jaki jest język akceptowany przez DAS, tzn. jaki jest język generowany przez wszystkie słowa akceptowane przez DAS. Czyli zapisując formalnie:

Definicja - Język akceptowany przez DAS

Niech  $A = (Q, \Sigma, \delta, q_0, F)$  będzie DAS. Język  $L(A)$  akceptowany przez  $A$  jest zdefiniowany jako zbiór wszystkich napisów, które są akceptowane przez  $A$ .

$$L(A) = \{w | w \text{ jest napisem nad } \Sigma \text{ oraz } A \text{ akceptuje } w\}$$

### 2.1.3. Rozszerzona funkcja przejścia

Definicję języka akceptowanego przez DAS możemy również zapisać przy użyciu tzw. rozszerzonej funkcji przejścia. Definicja ta określa w jakim stanie znajdzie się automat po przyjęciu na wejście danego ciągu znaków  $w$ . Nie jest ona specjalnie nam potrzebna, ale była podana na ćwiczeniach.

Ta definicja jest po prostu rozszerzeniem funkcji  $\delta$ , która jest obecna w definicji automatu. Ta z definicji automatu przyjmowała pojedynczy symbol i zwracała stan kolejny, a funkcja rozszerzona przyjmuje ciąg symboli i zwraca stan w którym znajduje się automat.

Definicja - Rozszerzona funkcja przejścia

Rozszerzona funkcja przejścia  $\tilde{\delta}$  - bierze stan  $q$  oraz łańcuch  $w$  i zwraca stan  $p$  - stan jaki osiągnie automat zaczynając działanie w  $q$  i przetwarzając łańcuch  $w$ , oczywiście  $\tilde{\delta}(q, \epsilon) = q$ .

Zależność pomiędzy zwykłą funkcją, a funkcją rozszerzoną jest taka, że jeśli:

$w = \chi \cdot a$  - czyli wszystkie znaki oprócz ostatniego oznaczone są symbolem  $\chi$ ,

symbol  $a$  jest niepusty, a chi może być puste

to wtedy

$$\tilde{\delta}(q, w) = \delta(\tilde{\delta}(q, \chi), a)$$

Natomiast język generowany przez automat możemy teraz zapisać jako:

$$L(A) = \{w | \tilde{\delta}(q_0, w) \in F\}$$

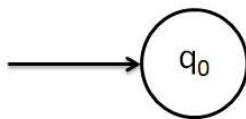
## 2.2. Reprezentacje automatów

Automaty łatwiej jest przedstawiać w innych postaciach niż piątka  $A = (Q, \Sigma, \delta, q_0, F)$ . Najbardziej oczywiste sposoby reprezentacji automatów to graf oraz tabela. Na ćwiczeniach mamy stosować określoną konwencję, którą sobie tutaj przedstawimy.

### 2.2.1. Reprezentacja graficzna automatu

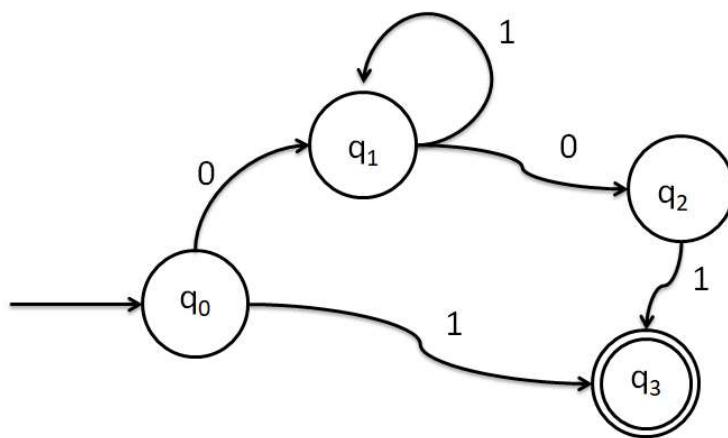
Automat przedstawimy w postaci grafu w którym wierzchołki to stany, a krawędzie to symbole wejściowe.

- Stan startowy oznaczamy wierzchołkiem do którego wchodzi strzałka, która nie ma węzła poprzednika, węzeł ten oznaczamy  $q_0$



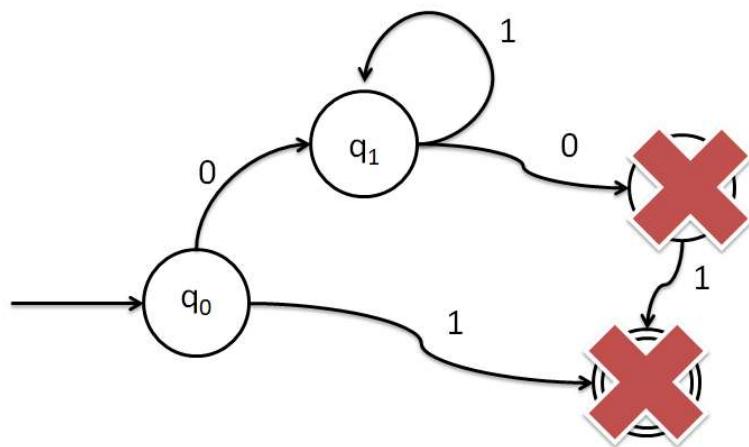
Rysunek 4: Stan startowy

- Symbol wejściowe zaznaczamy jako krawędzie z etykietą oznaczającą nazwę symbolu, natomiast stany akceptujące (końcowe) oznaczamy podwójnym kółkiem, czy wobec tego poniższy automat jest więc poprawny ?



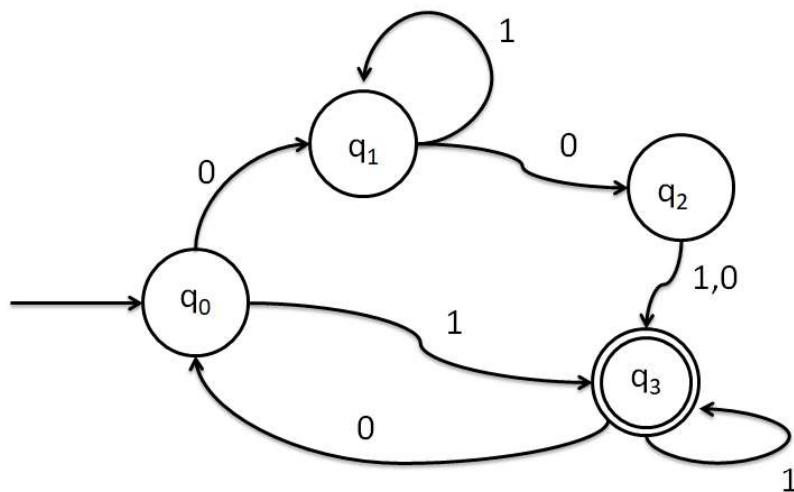
Rysunek 5: Czy ten DAS jest poprawny ?

Nie, automat ten nie jest poprawny, ponieważ brak w nim determinizmu. W stanie numer 2 nie określiliśmy co stanie się pod wpływem 0, a w stanie numer 3 nie określiliśmy nic ! W stanach końcowych również musimy określić reakcję na symbole wejściowe !



Rysunek 6: Niepoprawny DAS

Poprawmy więc automat tak aby był poprawny:



Rysunek 7: Poprawny DAS

### 2.2.2. Reprezentacja tabelaryczna automatu

Kolejną często stosowaną reprezentacją jest tabela. Wiersze tabeli odpowiadają stanom, a kolumny etykietom symboli wejściowych. W tabeli zapisujemy kolejny stan do którego przechodzimy z danego stanu pod wpływem pojawięcia się danego symbolu wejściowego. W danej komórce tabeli możemy wpisać tylko jeden stan, ponieważ nasz automat jest deterministyczny. Gwiazdką \* w wierszach oznaczamy stan akceptujący, a symbolem > oznaczamy stan startowy.

Naszą reprezentację grafową automatu zapiszemy przy użyciu tabeli:

	0	1
$> q_0$	$q_1$	$q_3$
$q_1$	$q_2$	$q_1$
$q_2$	$q_3$	$q_3$
$* q_3$	$q_0$	$q_3$

Rysunek 8: Reprezentacja tabelaryczna

### 2.3. Automat niedeterministyczny NAS

Automaty NAS (Niedeterministyczne automaty stanowe) różnią się tym od DAS, że mogą zawierać zero, jedną lub wiele krawędzi o tej samej etykiecie wychodzące z jednego węzła (stanu). Oznacza to, że będąc w danym stanie, przy danym symbolu wejściowym istnieje możliwość przejścia do zera, jednego lub wielu stanów, a nie tylko do jednego jak było to w przypadku DAS. Nie ma określonej zasady, które z przejściami wybierzemy, dzieje się to losowo (niedeterminizm).

Zwróćmy uwagę, że liczba przejść z daną etykietą jest tutaj dowolna, w szczególności może być równa zera, czyli przejścia z daną etykietą może nie być wcale (w DAS konieczne musielibyśmy mieć dokładnie jedno przejście z daną etykietą dla danego stanu).

Podajmy formalną definicję:

#### Definicja - NAS

NAS (Niedeterministyczny Automat Skończeniestanowy) to piątka  $A = (Q, \Sigma, \delta, q_0, F)$ , gdzie:

$Q$  - skończony zbiór stanów

$\Sigma$  - skończony zbiór symboli (alfabet wejściowy)

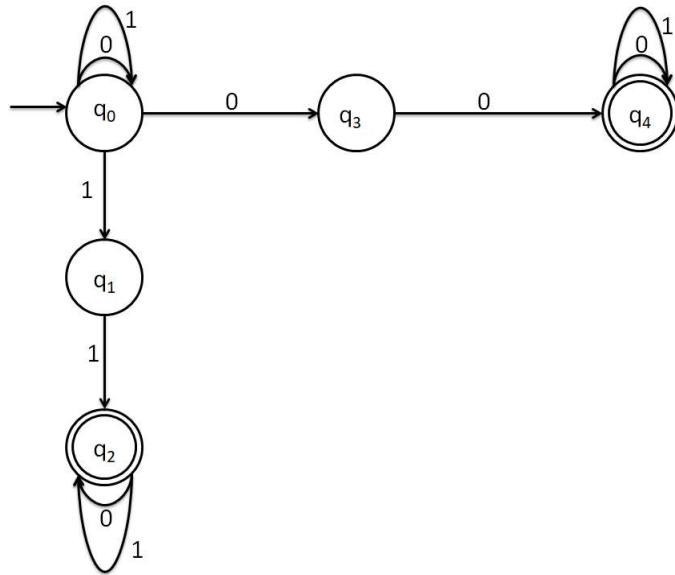
$\delta : Q \times \Sigma \rightarrow 2^Q$  - funkcja przejścia

$q_0 \in Q$  - stan początkowy

$F \subseteq Q$  - zbiór stanów końcowych (akceptujących)

Zwróćmy uwagę, że definicja ta jest bardzo podobna do definicji automatu DAS, z jedną różnicą, funkcja  $\delta$  może teraz przyjmować wartości, które są podzbiorami stanów, a nie tylko pojedynczy stanem, w szczególności może to być zbiór pusty !!!.

Jak zmienia się reprezentacja grafu? W reprezentacji grafowej możemy dodawać kolejne krawędzie z tą samą etykietą wychodzące od danego węzła, w szczególności możemy nie zatrzymać żadnej z krawędzi z daną etykietą. Z kolei w reprezentacji przy pomocy tabeli w komórki wpisujemy zbiory stanów, w szczególności może to być zbiór pusty. Spójrzmy na przykład:



Rysunek 9: Reprezentacja grafowa NAS

Widzimy, że niektóre węzły zawierają dwie krawędzie wychodzące z tymi samymi etykietami (mogło by być ich więcej: trzy, cztery, ile chcemy), a niektóre nie zawierają wcale krawędzi z daną etykietą, wtedy oznacza to, że danego przejścia nie można z danego stanu wykonać, przykładowo nie można wykonać przejścia do innego stanu ze stanu 1 pod wpływem 0 na wejściu.

	0	1
$>q_0$	$\{q_0, q_3\}$	$\{q_0, q_1\}$
$q_1$	$\emptyset$	$\{q_2\}$
$*q_2$	$\{q_2\}$	$\{q_2\}$
$q_3$	$\{q_4\}$	$\emptyset$
$*q_4$	$\{q_4\}$	$\{q_4\}$

Rysunek 10: Reprezentacja tabelaryczna NAS

### 2.3.1. Słowo akceptowane przez NAS

Słowo akceptowane przez NAS definiujemy analogicznie jak dla DAS, z tą różnicą, że w przypadku NAS słowo jest akceptowane jeśli istnieje przynajmniej jedna (dowolna z kilku możliwych) ścieżek, która prowadzi do akceptacji słowa. Dla przykładu słowa *01001* zostało zaakceptowane przez nasz niedeterministyczny automat, ponieważ istnieje jedna taka ścieżka, która doprowadziłaby do stanu końcowego.

## 2.4. Automat niedeterministyczny $\varepsilon$ NAS

Jak już wiemy  $\varepsilon$  oznacza słowo puste.  $\varepsilon$ NAS (Niedeterministyczny automat z  $\varepsilon$  ruchami) to automat który definiujemy tak samo jak NAS z tym dodatkiem, że mogą w nim występować krawędzie z symbolem pustym. Krawędź taka oznacza, że do danego stanu możemy przejść bez zużycia symbolu wejściowego. Tzn. podczas sprawdzania czy dane słowo jest akceptowane przez automat, możemy w przypadku dostępnosci krawędzi z symbolem pustym wykonać tą krawędź i przejść do innego stanu bez zużycia żadnego symbolu wejściowego, następnie analizę kontynuujemy dla kolejnych symboli wejściowych, ale już od nowego stanu, do którego przeszliśmy korzystając z krawędzi z symbolem pustym.

Zapiszmy formalną definicję:

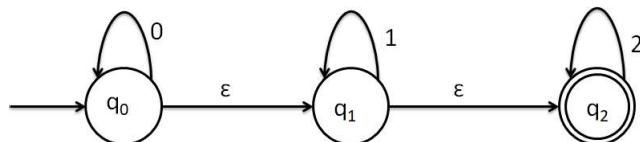
Definicja -  $\varepsilon$ NAS

$\varepsilon$ NAS (Niedeterministyczny Automat Skończeniestanowy z  $\varepsilon$  ruchami) to piątka  $A = (Q, \Sigma, \delta, q_0, F)$ , gdzie:

- $Q$  - skończony zbiór stanów
- $\Sigma$  - skończony zbiór symboli (alfabet wejściowy)
- $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow 2^Q$  - funkcja przejścia
- $q_0 \in Q$  - stan początkowy
- $F \subseteq Q$  - zbiór stanów końcowych (akceptujących)

Zauważmy, że definicja różni się od NAS tylko tym, że dodajemy  $\varepsilon$  jako możliwy argument funkcji  $\delta$ .

Przykład



Rysunek 11: Reprezentacja grafowa  $\varepsilon$ NAS

	0	1	2	$\varepsilon$
$q_0$	{ $q_0$ }	$\emptyset$	$\emptyset$	{ $q_1$ }
$q_1$	$\emptyset$	{ $q_1$ }	$\emptyset$	{ $q_2$ }
$q_2$	$\emptyset$	$\emptyset$	{ $q_2$ }	$\emptyset$

Rysunek 12: Reprezentacja tabelaryczna  $\varepsilon$ NAS

#### **2.4.1. Słowo akceptowane przez $\varepsilon$ NAS**

Słowo akceptowane przez automat  $\varepsilon$ NAS definiujemy podobnie jak dla automatu NAS, z tą różnicą, że teraz dodajemy jeszcze możliwość, że zostanie w jakimś momencie wykonane przejście z symbolem pustym. Ponownie wystarczy nam, że istnieje jedna ścieżka kończąca się w stanie akceptującym aby słowo zostało zaakceptowane.

#### **2.4.2. Sprawdzanie czy słowo jest akceptowane przez automat DAS, NAS lub $\varepsilon$ NAS przy pomocy rysunku**

Na ćwiczeniach do sprawdzania czy dane słowo jest akceptowane przez automat DAS, NAS lub  $\varepsilon$ NAS posługujemy się rysunkiem.

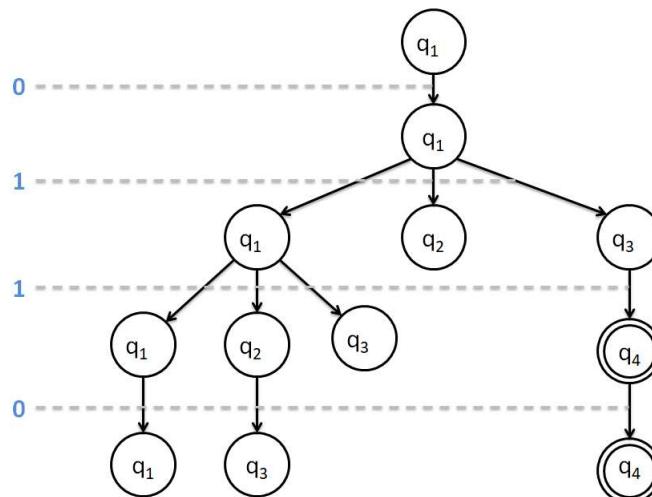
Dany mamy automat oraz słowo wejściowe. Na rysunku umieszczamy stany automatu, pierwszy na samej górze i schodzimy wierszami w dół, każdy kolejny wiersz oznacza stany możliwe do osiągnięcia po odczytaniu kolejnego symbolu wejściowego. Oczywiście ze względu na niedeterminizm, w wierszu może znajdować się wiele stanów, może też się zdarzyć, że dany stan będzie ostatnim, z którego dalej nie możemy już przejść, np. z powodu tego, że w danym stanie nie możemy się nigdzie przemieścić pod wpływem symbolu wejściowego.

Przykład

Mamy dane słowo wejściowe  $w = 0110$  oraz  $\varepsilon$ NAS:

	<b>0</b>	<b>1</b>	<b><math>\varepsilon</math></b>
$> q_1$	$\{q_1\}$	$\{q_1, q_2\}$	$\emptyset$
$q_2$	$\{q_3\}$	$\emptyset$	$\{q_3\}$
$q_3$	$\emptyset$	$\{q_4\}$	$\emptyset$
$* q_4$	$\{q_4\}$	$\{q_4\}$	$\emptyset$

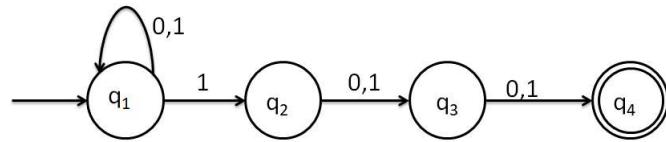
Sprawdzamy czy słowo jest akceptowane przez nasz automat:



Bardzo ważne jest zrozumienie skąd wziął się stan  $q_3$  w wierszu numer 3, przecież nie mamy tam przejścia pod wpływem 1 ze stanu  $q_1$ ! To oczywiście prawda, ale mamy tam przejście ze stanu  $q_2$  pod wpływem stanu symbolu pustego i dlatego zawieramy to od razu w tym wierszu, tak musimy postępować przy budowaniu drzewa, od razu sprawdzamy wszystkie możliwe  $\varepsilon$ -przejścia.

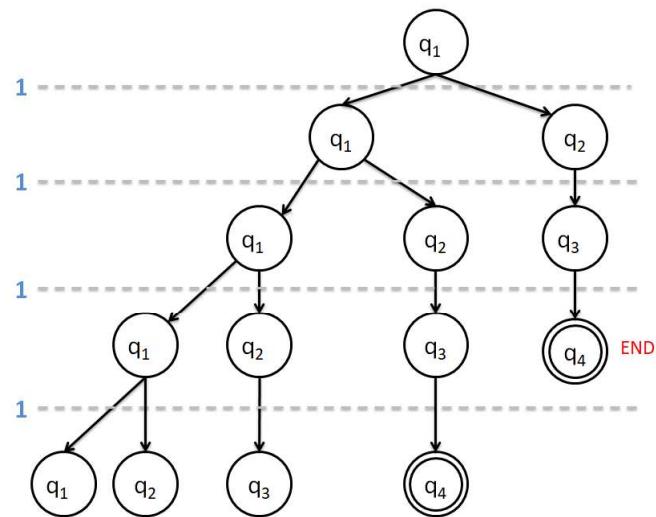
Kolejną ważną do przyswojenia rzeczą przy sprawdzaniu jest sytuacja w której dana ścieżka kończy się (umiera) na stanie akceptującym, słowo wejściowe jeszcze się nie skończyło. Czy wtedy taka ścieżka akceptuje dane słowo (a w konsekwencji czy taki automat akceptuje dane słowo)? Otóż nie, ponieważ jeśli ścieżka umrze, a całe słowo nie zostało 'zużyte', to wtedy automat nie akceptuje tego słowa. Zobaczmy to na przykładzie.

Mamy dany automat:



Jest to automat niedeterministyczny, który akceptuje słowa, które mają na trzecim miejscu od końca 1.

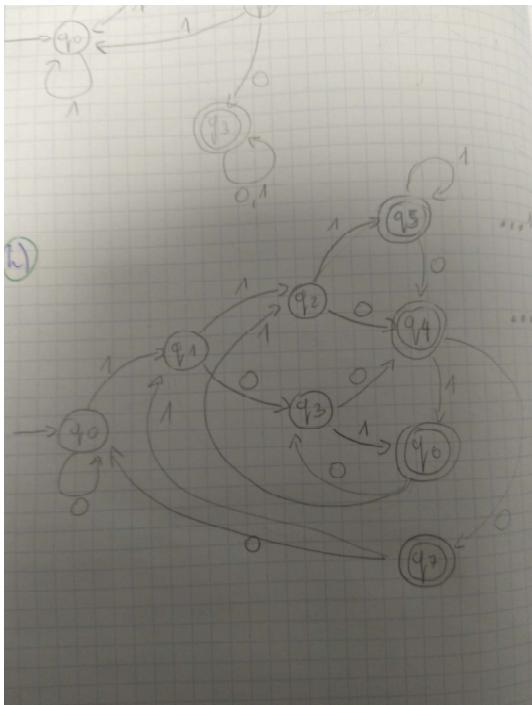
Badamy czy słowo *1111* jest akceptowane przez ten automat przy pomocy rysunku. Robimy więc rysunek:



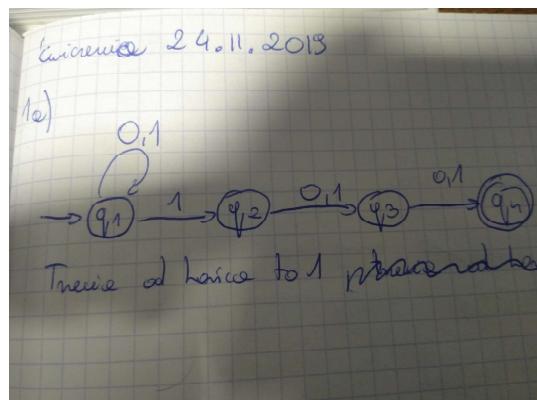
Widzimy, że automat ten akceptuje słowo *1111*, ale jest tak dlatego, że jest ono akceptowane przez gałąź, która po czterech symbolach wejściowych jest w stanie akceptującym. Natomiast gałąź, która jest najbardziej na prawo na rysunku nie akceptuje słowa *1111*, ponieważ po trzech symbolach się kończy.

### 3 Ćwiczenia nr 3 - Tworzenie automatów dla języków oraz konwersje $NAS \rightarrow DAS$ i $\varepsilon NAS \rightarrow DAS$

Na tych ćwiczeniach głównie skupiamy się na tworzeniu automatów dla zadanych języków oraz konwersji pomiędzy różnymi rodzajami automatów. Czy dwa automaty różnego typu mogą reprezentować taki sam język? Okazuje się, że tak, spójrzmy na przykład rozwiązania zadań o numerach 1h z ćwiczeń numer 2 oraz 1a z ćwiczeń numer 3.



(a) DAS



(b) NAS

Rysunek 13: Dwa różne typy automatów opisujące ten sam język (słowa mające na trzecim miejscu od końca 1).

Nasuwa się więc od razu pytanie czy dla każdego automatu niedeterministycznego możemy wykonać konwersję do DAS? Odpowiedź jest pozytywna.

Pamiętaj...

Dla każdego automatu niedeterministycznego możliwe jest utworzenie równoważnego automatu deterministycznego. Automat ten nie zawsze jest minimalny, może zawierać stany równoważne lub nieosiągalne.

Ktoś może sobie jednak zadawać pytanie, 'a po co mi to wszystko?', skoro jeden typ automatu mnie zadowala. Problem jest tylko taki, że w automacie niedeterministycznym nigdy nie wiemy, którą z dróg wybrać w momencie gdy mamy ich wiele do wyboru. Gdybyśmy więc chcieli zalgorytmizować taki automat i zapisać go np. w jakimś języku programowania, byłoby nam bardzo trudno. Dlatego właśnie konieczne jest dla nas poznanie metod pozwalających na konwersję z automatów niedeterministycznych na deterministyczne.

### 3.1. Konwersja NAS do DAS

Spróbujmy więc przekonwertować nasz automat NAS z rysunku powyżej do DAS.

Na tym przykładzie zaprezentujemy ogólny sposób postępowania. Od razu powiedzmy sobie jednak, że konwersję taką możemy wykonać metodą ogólną lub przyśpieszoną. Metoda ogólna bazuje na analizie wszystkich możliwości, nawet tych, które nie mogą zajść, a metoda przyśpieszona na analizie tylko tych możliwości, które rzeczywiście mogą wystąpić.

Oczywiście dla zadanego przypadku zawsze będziemy stosować metodę przyśpieszoną z uwagi na czas, metoda ta bazuje jednak na metodzie ogólniej, dlatego najpierw będziemy chcieli poznać tą ogólną, później tylko delikatnie ją zmodyfikujemy i dostaniemy metodą przyśpieszoną.

#### 3.1.1. Metoda ogólna konwersji NAS do DAS

Automat DAS tworzymy w postaci tabelarycznej na podstawie NAS.

- Patrząc na automat NAS, widzimy ile ma stanów, w pierwszym kroku wypisujemy w pierwszej kolumnie naszej tabeli wszystkie możliwe podzbiory zbioru wszystkich stanów, które ma automat NAS. W naszym przypadku automat NAS ma stany  $\{q_1, q_2, q_3, q_4\}$ , musimy wypisać wszystkie podzbiory tych stanów razem ze zbiorem pustym. W ogólnym przypadku podzbiorów takich jest  $2^n$ , gdzie n to liczba stanów.

	0	1
$\emptyset$		
$\{q_1\}$		
$\{q_2\}$		
$\{q_3\}$		
$\{q_4\}$		
$\{q_1, q_2\}$		
$\{q_1, q_3\}$		
$\{q_1, q_4\}$		
$\{q_2, q_3\}$		
$\{q_2, q_4\}$		
$\{q_3, q_4\}$		
$\{q_1, q_2, q_3\}$		
$\{q_1, q_2, q_4\}$		
$\{q_1, q_3, q_4\}$		
$\{q_2, q_3, q_4\}$		
$\{q_1, q_2, q_3, q_4\}$		

- Zaznaczamy w tabelce stan początkowy oraz stany akceptujące, stan początkowy jest taki jak dla automatu NAS, a stany akceptujące to te dla których przynajmniej jeden ze stanów jest akceptującym

	0	1
$\emptyset$		
$>\{q_1\}$		
$\{q_2\}$		
$\{q_3\}$		
$*\{q_4\}$		
$\{q_1, q_2\}$		
$\{q_1, q_3\}$		
$*\{q_1, q_4\}$		
$\{q_2, q_3\}$		
$*\{q_2, q_4\}$		
$*\{q_3, q_4\}$		
$\{q_1, q_2, q_3\}$		
$*\{q_1, q_2, q_4\}$		
$*\{q_1, q_3, q_4\}$		
$*\{q_2, q_3, q_4\}$		
$*\{q_1, q_2, q_3, q_4\}$		

3. Wypełniamy tabelkę, tj. patrzymy dla każdego elementu ze zbioru z pierwszej kolumny, gdzie możliwe są przejścia pod wpływem 0 i 1, zapisujemy zbiór stanów, które możliwe są do osiągnięcia pod wpływem 0 lub 1, ze stanów z pierwszej kolumny. Np. jeśli w pierwszej kolumnie mamy  $\{q_1, q_2, q_3\}$  to w drugiej kolumnie będziemy mieć wszystkie stany, które można osiągnąć z  $q_1$  pod wpływem zero,  $q_2$  pod wpływem 0 i  $q_3$  pod wpływem 0.

	0	1
$\emptyset$	$\emptyset$	$\emptyset$
$>\{q_1\}$	$\{q_1\}$	$\{q_1, q_2\}$
$\{q_2\}$	$\{q_3\}$	$\{q_3\}$
$\{q_3\}$	$\{q_4\}$	$\{q_4\}$
$*\{q_4\}$	$\emptyset$	$\emptyset$
$\{q_1, q_2\}$	$\{q_1, q_3\}$	$\{q_1, q_2, q_3\}$
$\{q_1, q_3\}$	$\{q_1, q_4\}$	$\{q_1, q_2, q_4\}$
$*\{q_1, q_4\}$	$\{q_1\}$	$\{q_1, q_2\}$
$\{q_2, q_3\}$	$\{q_3, q_4\}$	$\{q_3, q_4\}$
$*\{q_2, q_4\}$	$\{q_3\}$	$\{q_3\}$
$*\{q_3, q_4\}$	$\{q_4\}$	$\{q_4\}$
$\{q_1, q_2, q_3\}$	$\{q_1, q_3, q_4\}$	$\{q_1, q_2, q_3, q_4\}$
$*\{q_1, q_2, q_4\}$	$\{q_1, q_3\}$	$\{q_1, q_2, q_3\}$
$*\{q_1, q_3, q_4\}$	$\{q_1, q_4\}$	$\{q_1, q_2, q_4\}$
$*\{q_2, q_3, q_4\}$	$\{q_3, q_4\}$	$\{q_3, q_4\}$
$*\{q_1, q_2, q_3, q_4\}$	$\{q_1, q_3, q_4\}$	$\{q_1, q_2, q_3, q_4\}$

4. Teraz badamy, które z tych stanów z pierwszej kolumny są rzeczywiście osiągalne, tj. które faktycznie można osiągnąć wychodząc od stanu startowego. Stany te zaznaczamy sobie w jakiś sposób, my zaznaczać będziemy kolorem. Wychodzimy od stanu startowego, ten wiemy, że jest osiągalny. Pod wpływem 0 możemy przejść do stanu startowego, który już mamy zaznaczony, a pod wpływem 1 do stanu  $\{q_1, q_2\}$ . UWAGA: Teraz już nie rozważamy pojedynczych stanów ze zbioru, ale cały zbiór traktujemy jako jeden stan. Jeśli więc możemy przejść z  $\{q_1\}$  do  $\{q_1, q_2\}$ , to jako stan osiągalny zaznaczamy tylko  $\{q_1, q_2\}$ , a nie stany  $\{q_1\}$  oraz  $\{q_2\}$  osobno.

	0	1
$>\{q_1\}$	$\{q_1\}$	$\{q_1, q_2\}$
$\{q_2\}$	$\{q_3\}$	$\{q_3\}$
$\{q_3\}$	$\{q_4\}$	$\{q_4\}$
$*\{q_4\}$		
$\{q_1, q_2\}$	$\{q_1, q_3\}$	$\{q_1, q_2, q_3\}$
$\{q_1, q_3\}$	$\{q_1, q_4\}$	$\{q_1, q_2, q_4\}$
$*\{q_1, q_4\}$	$\{q_1\}$	$\{q_1, q_2\}$
$\{q_2, q_3\}$	$\{q_3, q_4\}$	$\{q_3, q_4\}$
$*\{q_2, q_4\}$	$\{q_3\}$	$\{q_3\}$
$*\{q_3, q_4\}$	$\{q_4\}$	$\{q_4\}$
$\{q_1, q_2, q_3\}$	$\{q_1, q_3, q_4\}$	$\{q_1, q_2, q_3, q_4\}$
$*\{q_1, q_2, q_4\}$	$\{q_1, q_3\}$	$\{q_1, q_2, q_3\}$
$*\{q_1, q_3, q_4\}$	$\{q_1, q_4\}$	$\{q_1, q_2, q_4\}$
$*\{q_2, q_3, q_4\}$	$\{q_3, q_4\}$	$\{q_3, q_4\}$
$*\{q_1, q_2, q_3, q_4\}$	$\{q_1, q_3, q_4\}$	$\{q_1, q_2, q_3, q_4\}$

Wiersze z pokolorowanymi w pierwszej kolumnie stanami stanowią nasz DAS !

5. Na koniec stany z pierwszej kolumny zamieniamy na pojedyncze, tzn. przemianowujemy zbiór z pierwszej kolumny na pojedynczy stan, wygodnie jest przy tym zachować konwencję, że przykładowo zbiór  $\{q_1, q_2, q_3\}$  będziemy mapować na stan  $q_{123}$ . Otrzymujemy więc:

	0	1
∅	∅	∅
>q <sub>1</sub>	q <sub>1</sub>	q <sub>12</sub>
q <sub>2</sub>	q <sub>3</sub>	q <sub>3</sub>
q <sub>3</sub>	q <sub>4</sub>	q <sub>4</sub>
*q <sub>4</sub>	∅	∅
q <sub>12</sub>	q <sub>13</sub>	q <sub>123</sub>
q <sub>13</sub>	q <sub>14</sub>	q <sub>124</sub>
*q <sub>14</sub>	q <sub>1</sub>	q <sub>12</sub>
q <sub>23</sub>	q <sub>34</sub>	q <sub>34</sub>
*q <sub>24</sub>	q <sub>3</sub>	q <sub>3</sub>
*q <sub>34</sub>	q <sub>4</sub>	q <sub>4</sub>
q <sub>123</sub>	q <sub>134</sub>	q <sub>1234</sub>
*q <sub>124</sub>	q <sub>13</sub>	q <sub>123</sub>
*q <sub>134</sub>	q <sub>14</sub>	q <sub>124</sub>
*q <sub>234</sub>	q <sub>34</sub>	q <sub>34</sub>
*q <sub>1234</sub>	q <sub>134</sub>	q <sub>1234</sub>

6. Ostatecznie nasz DAS to:

	0	1
>q <sub>1</sub>	q <sub>1</sub>	q <sub>12</sub>
q <sub>12</sub>	q <sub>13</sub>	q <sub>123</sub>
q <sub>13</sub>	q <sub>14</sub>	q <sub>124</sub>
*q <sub>14</sub>	q <sub>1</sub>	q <sub>12</sub>
q <sub>123</sub>	q <sub>134</sub>	q <sub>1234</sub>
*q <sub>124</sub>	q <sub>13</sub>	q <sub>123</sub>
*q <sub>134</sub>	q <sub>14</sub>	q <sub>124</sub>
*q <sub>1234</sub>	q <sub>134</sub>	q <sub>1234</sub>

Zauważmy, że ma on dokładnie tyle stanów ile nasz DAS, który utworzyliśmy w zadaniu 1h z ćwiczeń numer 2 na zajęciach. W tym przypadku przez konwersję otrzymaliśmy automat minimalny, ale nie zawsze tak musi być.

### 3.1.2. Metoda przyśpieszona konwersji NAS do DAS

Metoda przyśpieszona jest w zasadzie bardzo podobna. Różnica polega na tym, że na początku nie wypisujemy wszystkich możliwych podzbiorów stanów automatu NAS, a jedynie stan początkowy i od razu dla niego wypełniamy wszystkie kolumny. Następnie do pierwszej kolumny dopisujemy tylko te stany, które osiągnęliśmy ze stanu startowego. W metodzie tej pomijamy zbędne rozważanie stanów, które i tak nie mają szans być osiągnięte.

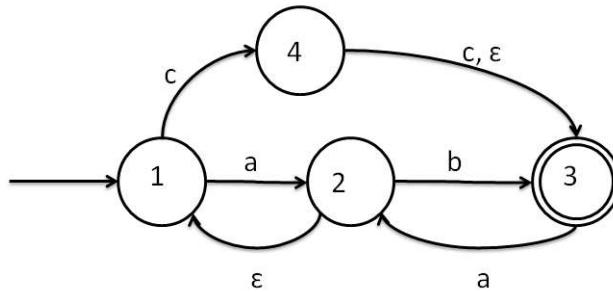
	0	1
$>\{q_1\}$	$\xrightarrow{\quad}$	$\xrightarrow{\quad} \{q_1, q_2\}$
$\{q_1, q_2\}$	$\xleftarrow{\quad} \{q_1, q_3\}$	$\{q_1, q_2, q_3\}$
$\{q_1, q_3\}$	$\{q_1, q_4\}$	$\{q_1, q_2, q_4\}$
$\{q_1, q_2, q_3\}$	$\{q_1, q_3, q_4\}$	$\{q_1, q_2, q_3, q_4\}$
$*\{q_1, q_2, q_4\}$	$\{q_1, q_3\}$	$\{q_1, q_2, q_3\}$
$*\{q_1, q_3, q_4\}$	$\{q_1, q_4\}$	$\{q_1, q_2, q_4\}$
$*\{q_2, q_3, q_4\}$	$\{q_3, q_4\}$	$\{q_3, q_4\}$
$*\{q_1, q_2, q_3, q_4\}$	$\{q_1, q_3, q_4\}$	$\{q_1, q_2, q_3, q_4\}$

Teraz tylko zamieniamy zbiory na stany i otrzymujemy to co metodą ogólną, tyle, że szybciej.

### 3.2. Konwersja $\varepsilon$ NAS do DAS

Teraz zajmiemy się konwersją automatów  $\varepsilon$ NAS do DAS. Znowu w tym przypadku możemy postępować zgodnie z ogólną metodą lub przyśpieszoną. Metoda ogólna jest jednak trochę bardziej skomplikowana niż przyśpieszona, dla nas wystarczająca będzie znajomość metody przyśpieszonej i tylko tą sobie tutaj omówimy.

Omówienie tej metody przedstawimy na przykładzie. Rozważamy automat  $\varepsilon$ NAS jak na rysunku poniżej:



#### 3.2.1. Epsilon domknięcie $\varepsilon^*$

Przed rozpoczęciem opisu algorytmu wprowadźmy sobie jedno konieczne pojęcie. *Epsilon domknięcie*, oznaczane  $\varepsilon^*$  to zbiór wszystkich stanów do których można przejść od wybranego stanu korzystając z przejścia  $\varepsilon$ . Dla stanu  $q_n$  w zbiorze  $\varepsilon^*$  zawsze zawieramy sam stan  $q_n$ . Uwaga: czasami może zdarzyć się, że można wykonać sekwencje  $\varepsilon$  przejść, tj. ze stanu  $q_i$  przejść pod wpływem  $\varepsilon$  do stanu  $q_{i+1}$ , a z niego pod wpływem  $\varepsilon$  jeszcze do kolejnego itd... wtedy do naszego zbioru  $\varepsilon^*$  wstawiamy wszystkie te stany.

#### 3.2.2. Metoda przyśpieszona konwersji $\varepsilon$ NAS do DAS

1. Robimy tabelkę dla naszego automatu  $\varepsilon$ NAS

	a	b	c	$\epsilon$
>1	{2}	$\emptyset$	{4}	$\emptyset$
2	$\emptyset$	{3}	$\emptyset$	$\emptyset$
*3	{2}	$\emptyset$	$\emptyset$	$\emptyset$
4	$\emptyset$	$\emptyset$	{3}	{3}

2. Do tabelki dodajemy z prawej strony kolumnę z *Epsilon domknięciem*  $\epsilon^*$  (tutaj pominąłem kolumnę  $\epsilon$ , ale nie jest ona nam już potrzebna)

	a	b	c	$\epsilon^*$
>1	{2}	$\emptyset$	{4}	{1}
2	$\emptyset$	{3}	$\emptyset$	{1,2}
*3	{2}	$\emptyset$	$\emptyset$	{3}
4	$\emptyset$	$\emptyset$	{3}	{3,4}

3. Tworzymy nową tabelkę, w której będziemy wypełniać po kolejnych wierszach na podstawie poprzednich. W pierwszej kolumnie będziemy wypisywać nasze stany. W pierwszym wierszu w pierwszej kolumnie wpisujemy zawsze to co otrzymaliśmy jako  $\epsilon^*$  dla naszego stanu początkowego, w naszym przykładzie jest to {1}. Kolejne kolumny tej tabeli nazywamy zgodnie z regułą: symbol\_wejściowy  $\epsilon$ , czyli u nas mamy:

	$a\epsilon^*$	$b\epsilon^*$	$c\epsilon^*$
>{1}			

W pierwszej kolumnie mamy tutaj akurat zbiór jednoelementowy, ale w ogólnym przypadku może być wieloelementowy. Co dalej?

4. Bierzemy po kolejny stan z naszego zbioru z pierwszej kolumny, u nas mamy tylko jeden więc bierzemy stan numer 1. Dla tego stanu patrzymy co się dzieje w  $\epsilon$ NAS jeśli w tym stanie wykonamy przejście a (kolumna  $a\epsilon^*$ ), b (kolumna  $b\epsilon^*$ ) oraz c (kolumna  $c\epsilon^*$ ). Będąc w stanie 1, pod wpływem a przechodzimy do stanu 2. Dla stanu w którym się znaleźliśmy po przejściu bierzemy jego  $\epsilon^*$ , czyli my po przejściu ze stanu 1 do 2 pod wpływem a, bierzemy  $\epsilon^*$  dla 2, a ono wynosi {1,2}, tą wartość wpisujemy w tabelę. Gdybyśmy mieli w pierwszej kolumnie wiele stanów, wówczas procedurę musimy powtórzyć dla każdego z nich i w tabelę wpisujemy sumę teoriomnogościową.

	$a\epsilon^*$	$b\epsilon^*$	$c\epsilon^*$
>{1}	{1,2}	$\emptyset$	{3,4}
{1,2} ←			
{3,4} ←			

Jeśli w wyniku tego dostaniemy jakiś zbiór stanów którego nie mamy w pierwszej kolumnie, to dodajemy go jako nowy wiersz i dla niego powtarzamy procedurę. Końcowa postać naszej tabeli:

	$a\epsilon^*$	$b\epsilon^*$	$c\epsilon^*$
$>\{1\}$	{1,2}	$\emptyset$	{3,4}
{1,2}	{1,2}	{3}	{3,4}
$*\{3,4\}$	{1,2}	$\emptyset$	{3}
$*\{3\}$	{1,2}	$\emptyset$	$\emptyset$

Jako stany akceptacyjne zaznaczmy te, które mają w swoim zbiorze stan akceptacyjny z naszego  $\epsilon$ NAS, czyli w tym przypadku 3.

5. Kolejną rzeczą, którą należy zrobić jest zamiana zbiorów na pojedyncze stany, tak jak już to robiliśmy przy konwersji NAS do DAS.

6. W tym momencie możemy mieć dwie sytuacje

- W utworzonej przez nas ostatnio tabeli nie ma zbiorów pustych (wtedy mamy już gotowy DAS i kończymy algorytm)
- Lub tak jak w naszym przypadku mamy przynajmniej jeden zbiór pusty, wtedy aby przekonwertować taką tabelę do DAS, wprowadzamy dodatkowy stan, który reprezentuje stan pusty, czyli w naszym przypadku robimy tak:

	$a\epsilon^*$	$b\epsilon^*$	$c\epsilon^*$
$>1$	12	5	34
12	12	3	34
$*34$	12	5	3
$*3$	12	5	5
5	5	5	5

To już nasz gotowy DAS.

## 4 Ćwiczenia nr 4 - Wyrażenia regularne

Na tych ćwiczeniach zajmujemy się wyrażeniami regularnymi. Wyrażenia regularne to proste wyrażenia, które są kolejnym narzędziem do opisywania języka. Przy pomocy wyrażeń regularnych możemy opisać każdy język, który jest akceptowany przez automat skończony.

### 4.1. Pojęcia wstępne

Wyrażenia regularne będziemy definiować przy pomocy operacji na językach dlatego najpierw omówimy sobie niezbędne dla nas pojęcia, a potem przejdziemy do definicji wyrażeń regularnych. Interesujące nas operacje to: suma teoriomnogościowa języków, złożenie języków oraz domknięcie.

#### 4.1.1. Suma dwóch języków

Suma teoriomnogościowa dwóch języków  $L$  oraz  $M$  oznaczana jako  $L \cup M$  to zbiórłańcuchów, które należą do  $L$ , do  $M$  lub do obu naraz.

Przykład

Niech  $L = \{0, 10, 111, 001\}$  oraz  $M = \{\varepsilon, 1, 01, 10\}$  wtedy  
 $L \cup M = \{\varepsilon, 0, 1, 10, 01, 111, 001\}$

#### 4.1.2. Złożenie (konkatenacja)

Złożenie dwóch języków  $L$  oraz  $M$  oznaczamy  $LM$ , to zbiórłańcuchów powstałycych przez złożenie dowolnegołańcucha z  $L$  z dowolnymłańcuchem z  $M$ . UWAGA: złożenie nie jest operacją przemienną, tzn.  $LM \neq ML$ . Złożenie języka z samym sobą oznaczamy  $L^i$  i obliczamy jako  $L^i = LL^{i-1}$ , gdzie  $i$  to liczba złożień natomiast  $L^0 = \{\varepsilon\}$ .

Przykład

Zadanie 2a)

$$L = \{0, 10, 111, 001\}$$

$$L^0 = \{\varepsilon\}$$

$$L^1 = \{0, 10, 111, 001\}$$

$$L^2 = \{00, 010, 0111, 0001, 100, 110, 1111, 10001, 1010, 100, 10111, 10001, 110, 1111, 1110, 11001, 001001, 00110, 0010, 0011\} \quad 16 \text{ wyników}$$

#### 4.1.3. Domknięcie Kleene'ego (domknięcie)

Domknięcie języka  $L$  oznaczamy symbolem  $L^*$ , to nieskończona suma teoriomnogościowa złożień języka z samym sobą:  $L^* = \bigcup_{i=0}^{\infty} L^i = L^0 \cup L^1 \cup L^2 \cup \dots$

A zatem  $L^*$  to zbiór wszystkich słów otrzymanych w wyniku złożenie dowolnej liczby słów z  $L$ .

Przykład z Hopcrofta:

**Przykład 2.10.** Niech  $L_1 = \{10, 1\}$  i  $L_2 = \{011, 11\}$ . Wtedy  $L_1L_2 = \{10011, 1011, 111\}$ .  
 $\{10, 11\}^* = \{\varepsilon, 10, 11, 1010, 1011, 1110, 1111, \dots\}$ .  
it alfabetem, to  $\Sigma^*$  jest takie: ...

## 4.2. Wyrażenia regularne

Mamy już pojęcia wstępne potrzebne nam do zdefiniowania wyrażeń regularnych, przystępujemy więc do właściwych definicji. Wyrażenie regularne oznaczamy dużą literą  $E$ . Wyrażenie regularne  $E$  określa język (regularny)  $L(E)$  podobnie jak automat skończony  $A$  określa język  $L(A)$ . Pojedyncze wyrażenie regularne reprezentuje po prostu zbiór słów. Omówimy sobie w poniższej definicji jakie zbiory odpowiadają jakim wyrażeniom regularnym.

### 4.2.1. Konwencja notacyjna

Zanim jednak to zrobimy, musimy wprowadzić pewną konwencję notacyjną. Konieczność taka zachodzi ponieważ w wyrażeniach regularnych czasami zdarza się, że symbol wyrażenia jest taki sam jak słowa, które reprezentuje (brzmi zagmatwanie, ale spójrzmy na przykłady).

Przykładowo wyrażenie regularne  $\varepsilon$  reprezentuje słowo puste. Możemy więc zapisać, że  $L(\varepsilon) = \{\varepsilon\}$ . Problem z tą notacją jest taki, że symbol  $\varepsilon$  po stronie lewej reprezentuje coś innego niż ten po stronie prawej. Po stronie lewej w nawiasach mamy wyrażenie regularne, po stronie prawej, symbol ten oznacza słowo puste. Aby móc zatem odróżnić jedno od drugiego decydujemy się (tak jest na ćwiczeniach i w Hopcroftie), że wyrażenia regularne będziemy oznaczać czcionką pogrubioną.

A zatem zapisując poprawnie mamy, że:  $L(\varepsilon) = \{\varepsilon\}$

#### 4.2.2. Definicja

##### Definicja - Wyrażenia regularne

Wyrażenia regularne i zbiory przez nie reprezentowane definiujemy rekurencyjnie w następujący sposób:

- $\emptyset$  - określa zbiór pusty, zbiór, który nie zawiera żadnego słowa  $L(\emptyset) = \emptyset$
- $\varepsilon$  - reprezentuje zbiór zawierający jedno słowo puste  $L(\varepsilon) = \varepsilon$
- $a$  - małymi literami oznaczamy wyrażenia regularne reprezentujące pojedyncze słowa  $L(a) = \{a\}$
- $E$  - wyrażenie regularne reprezentujące język
- $E + F$  - wyrażenie regularne reprezentujące sumę języków  $L(E + F) = L(E) \cup L(F)$
- $EF$  - wyrażenie regularne reprezentujące złożenie języków  $L(EF) = L(E)L(F)$
- $E^*$  - wyrażenie regularne reprezentujące domknięcie języka opisywanego przez  $E$   $L(E^*) = (L(E))^*$

Przykłady wyrażeń regularnych:

- **00** - wyrażenie regularne reprezentujące  $\{00\}$
- **$(0 + 1)^*$**  - wyrażenie regularne opisujące zbiór wszystkichłańcuchów złożonych z zer i jedynek
- **$(0 + 1)^*00(0 + 1)^*$**  - wyrażenie regularne opisujące zbiór wszystkichłańcuchów zer i jedynek zawierających przynajmniej dwa kolejne zera
- **$(0 + 1)^*011$**  - wyrażenie regularne opisujące wszystkiełańcuchy zer i jedynek kończące się na 011

#### 4.3. Priorytety operatorów i symbol $\Sigma$

W powyższych przykładach obecne są nawiasy określające priorytety operatorów. Nawiasy dodawane są w celu określenie priorytetów. Przymijmy następującą kolejność operatorów:

1.  $*$  - najważniejsze, tak jak potęga w obliczeniach
2. złożenie - kolejne, tak jak mnożenie w obliczeniach, przy czym mając kilka złożień, np.  $abc$ , to oznacza dla nas  $(ab)c$
3.  $+$  - najmniej ważny, tak jak dodawanie w obliczeniach

Przykład

Dodaj nawiasy do wyrażenia:  $ab^*a + ab$ .

Rozwiązanie:  $((a(b^*))a) + (ab)$

Często w wyrażeniach regularnych chcemy wyrazić występowanie dowolnego z symboli w danym miejscu, robimy to za pomocą  $+$  zapisując przykładowo  $0+1+2$ , co oznacza wystąpienie 0 lub 1 lub 2. Sytuacja ta jest bardzo często, a zapisywanie jej dla wielu symboli staje się niewygodne, dlatego wprowadza się dodatkowy symbol  $\Sigma$ , który reprezentuje taką właśnie alternatywę.

Przykładowo rozważając ciągi składające się tylko z 0 i 1, możemy zapisać, że:

$$0 + 1 = \Sigma$$

#### 4.4. Równoważność automatów i wyrażeń regularnych

Teraz dowiedziemy sobie tego co stwierdziliśmy na początku tych ćwiczeń, czyli, że języki opisywane przez automaty skończone są dokładnie językami opisywanymi przez wyrażenia regularne. Pokażemy, że dowolne wyrażenie regularne zamienić można na automat  $\varepsilon$ NAS, ten z kolei jak już wiemy można zamienić na NAS oraz DAS.

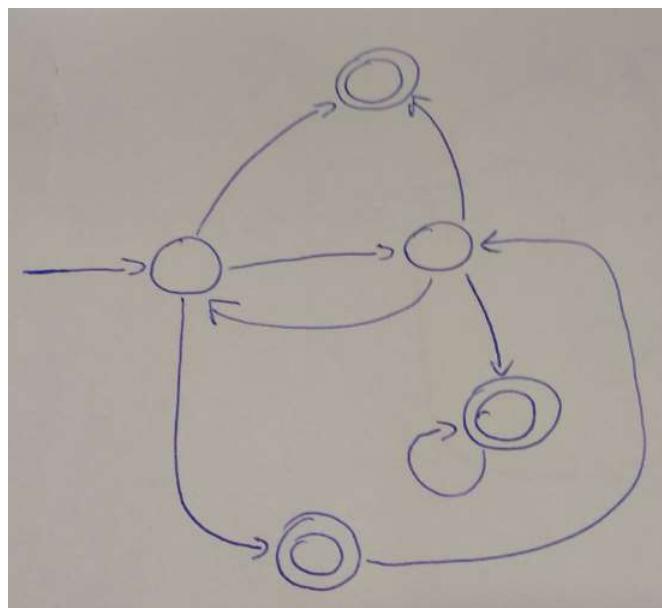
Pamiętaj...

Jeżeli  $L(R)$  jest językiem opisywanym przez wyrażenie regularne  $R$ , to istnieje  $\varepsilon$ NAS, który akceptuje  $L(R)$ . Taki  $\varepsilon$ NAS musi spełniać trzy warunki (nie do końca rozumiem dlaczego, ale tak już jest, tak było podane na ćwiczeniach...):

- Ma dokładnie jeden stan akceptujący
- Nie ma krawędzi wchodzących do stanu początkowego
- Nie ma krawędzi wychodzących ze stanu akceptującego

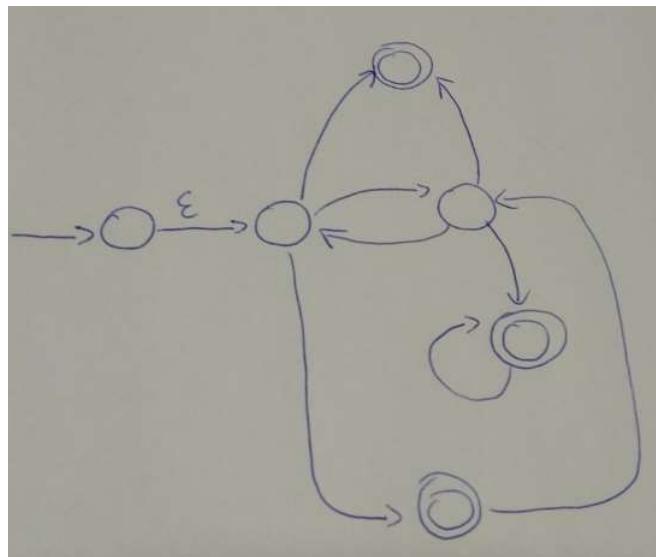
##### 4.4.1. Jak przekształcać automat aby spełniał warunki ?

Spełnienie powyższych trzech warunków można łatwo osiągnąć poprzez przekształcanie grafu automatu. Spójrzmy na przykład. Mamy automat:

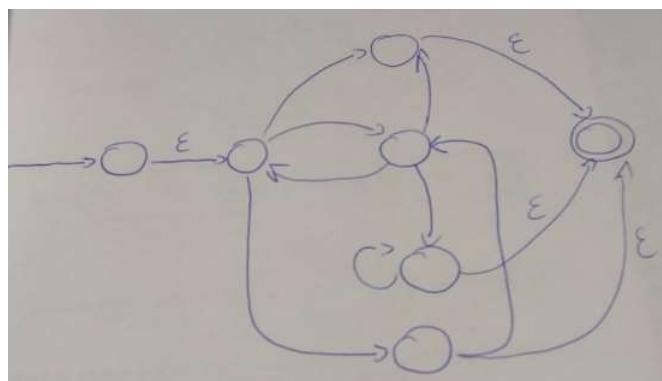


Czy taki automat może być równoważny wyrażeniu regularnemu ? Nie ! Nie spełnia trzech powyższych założeń. Jak go zatem przekształcić tak aby je spełniał ?

- Najpierw realizujemy założenie dotyczące stanu początkowego, tzn. nie może mieć krawędzi wchodzących do stanu początkowego, robimy to przez dodanie jednego stanu zaraz za początkowym i przejście do niego pod wpływem  $\varepsilon$



- Czy teraz jest ok? Prawie, widzimy bowiem, że dwa z założeń nadal nie są spełnione, mamy więcej niż jeden stan akceptujący i dodatkowo mamy krawędzie wychodzące ze stanu akceptującego. Z problemem tym radzimy sobie bardzo podobnie, dodajemy jeden stan, który jest akceptujący i ze wszystkich aktualnych stanów akceptujących prowadźmy do niego krawędź pod wpływem  $\epsilon$



Taki graf spełnia wszystkie z założeń.

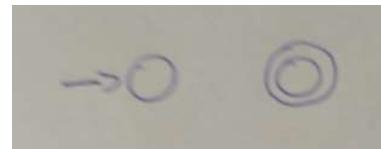
#### 4.4.2. Dowód

Teraz pokażemy dowód na to, że rzeczywiście każde wyrażenie regularne można przekształcić do automatu  $\epsilon$ NAS. Okej, ale czy to nam się do czegoś przyda? Tak, dowód ten bowiem będzie zawierał podanie równoważnych reprezentacji wyrażeń regularnych z fragmentami automatu  $\epsilon$ NAS, znajomość tych częściowych równoważności pozwoli nam następnie na konwersję z dowolnego wyrażenia regularnego na automat  $\epsilon$ NAS.

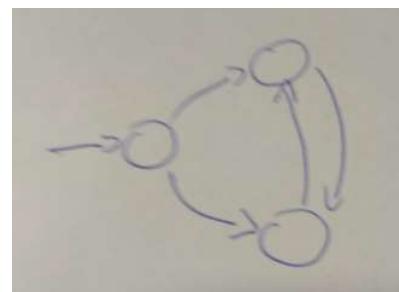
Dowód przedstawimy w taki sposób:

Dla każdego możliwego podstawowego wyrażenia regularnego podamy odpowiadający mu fragment  $\epsilon$ NAS.

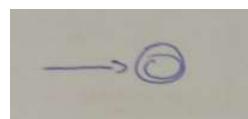
- $\emptyset$  Automat równoważny takiemu wyrażeniu regularnemu, to automat, który nie akceptuje żadnego ze słów, nawet pustego, nic nie akceptuje. Jak narysować taki automat? To po prostu automat nie mający żadnego stanu akceptującego lub taki w którym stan akceptujący jest nieosiągalny, przykładowo:



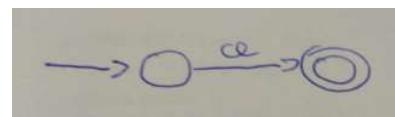
lub



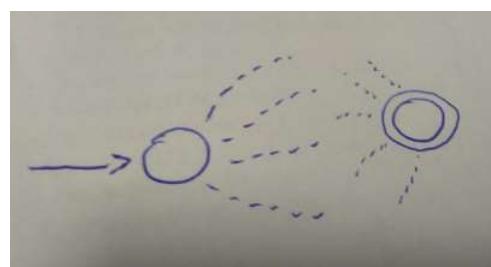
- $\epsilon$  Taki automat akceptuje tylko słowo puste. W takim automacie stan początkowy jest również stanem akceptującym. Można się zastanawiać czy taki automat nie akceptuje przypadkiem wszystkich słów? Nie, dlatego, że ze stany początkowego nie możemy już nigdzie przejść więc jakikolwiek symbol wejściowy powoduje, że gałąź wykonania automatu umiera.



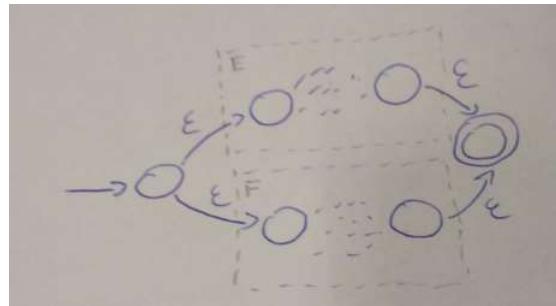
- $a$  Taki automat akceptuje tylko słowo  $a$ , a więc to automat, który ma jedną gałąź z etykietą  $a$



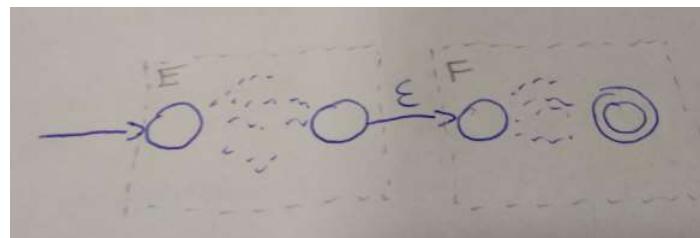
- $E$  Wyrażenie regularne to jak już wiemy automat z jednym stanem akceptującym, który nie ma krawędzi wchodzących do stanu początkowego i wychodzących ze stanu akceptującego, to co dzieje się w środku takiego automatu jest dowolny, a więc możemy go przedstawić jako:



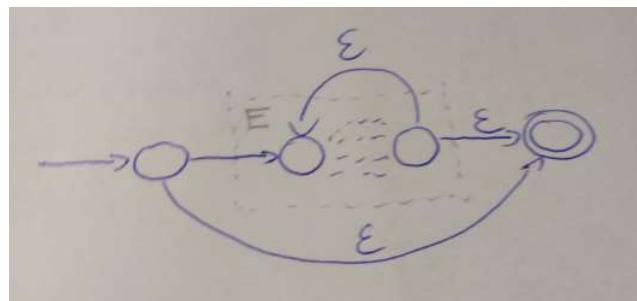
- $E + F$  To automat w którym mamy wybór pomiędzy automatem reprezentowanym przez  $E$ , a tym reprezentowanym przez  $F$ . Rysujemy więc dwie gałęzie z symbolem  $\epsilon$  od stanu początkowego, które reprezentują ten wybór, a także wprowadzamy jeden stan akceptujący (pamiętajmy o spełnieniu warunków !)



- $EF$  W takim przypadku najpierw musi się wykonać automat, który reprezentuje  $E$ , a potem ten reprezentujący  $F$ , stan akceptujący automatu  $E$  przestaje być już stanem akceptującym, bo dopiero wykonanie całego złożenia daje akceptację. Do automatu  $F$  przechodzimy pod wpływem  $\epsilon$ .



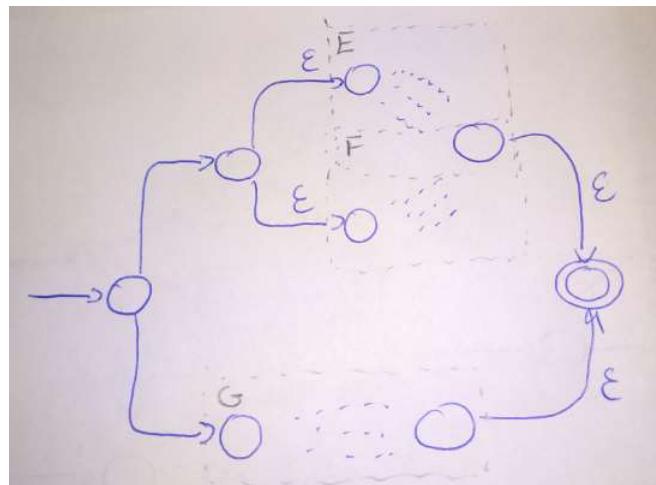
- $E^*$  W tym przypadku automat może wykonać się 0 razy, 1 raz lub dowolną inną liczbę razy, dlatego wprowadzamy przejście  $\epsilon$ , które zawsze pozwala nam przejść do kolejnego wykonania automatu  $E$ .



Dowiedliśmy tym samym, że każde wyrażenie regularne można przedstawić przy pomocy automatu  $\epsilon$ NAS, ponieważ każdą podstawową składową wyrażenia regularnego można przedstawić jako automat  $\epsilon$ NAS.

**UWAGA:** Zwróćmy jeszcze uwagę na to co w przypadku gdy mamy kilka wyrażeń regularnych połączonych tym samym operatorem. Przykładowo:  $E + F + G$ . Wiemy już, że kolejność jest zawsze od lewej do prawej, to oznacza, że  $E + F + G = (E + F) + G$ . Jak więc narysujemy taki automat ?

Najpierw rysujemy dla  $E + F$ , następnie traktujemy go jako jeden automat (jedno wyrażenie regularne) i do niego dodajemy G, narysujmy:

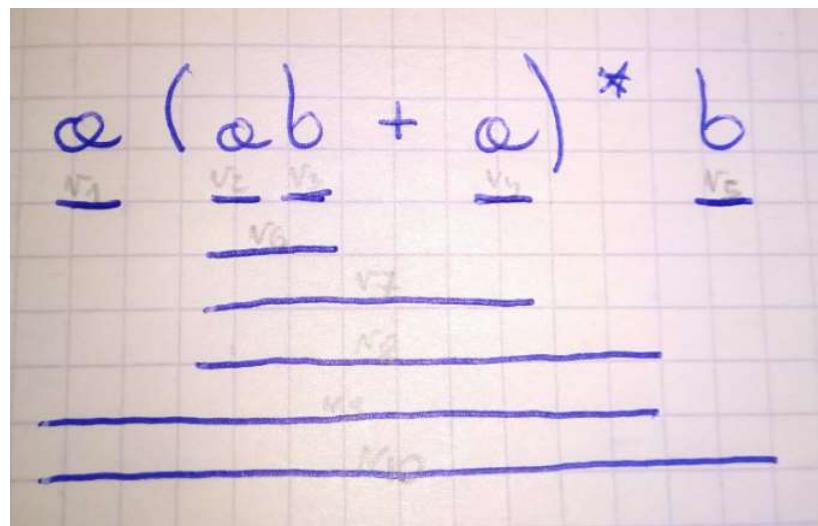


#### 4.5. Konwersja reg-exp $\rightarrow \varepsilon NAS$

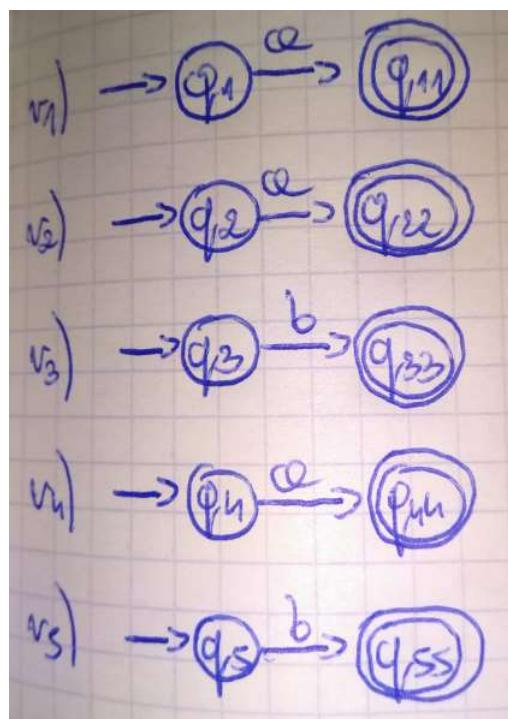
Ostatnie co sobie tutaj omówimy to coś czego w tym roku na ćwiczeniach nie było, ale co jest zazwyczaj podstawowym poleceniem na wejściówce z tematu wyrażeń regularnych, czyli konwersja  $regexp \rightarrow \varepsilon NAS$ . Procedurę przedstawię na podstawie tej opisanej w Hopcroftie, bazuje ona jednak prawie w całości na wiedzy, która została opisana powyżej, a więc jeśli znamy zamianę pojedynczych wyrażeń na automaty oraz priorytety wyrażeń regularnych, to w zasadzie teraz tylko to stosujemy w odpowiedniej kolejności :)

Algorytm omówimy sobie na przykładzie: Zamienić wyrażenie regularne  $a(ab + a)^*b$  na automat  $\varepsilon NAS$ .

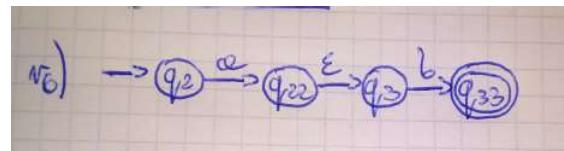
1. Algorytm rozpoczynamy od wydzielenie z wyrażenia najbardziej podstawowych wyrażeń regularnych, które je budują, następnie te wydzielone łączymy w bardziej złożone zgodnie z regułami priorytetów. Kolejne wydzielenia zaznaczamy sobie kreskami.



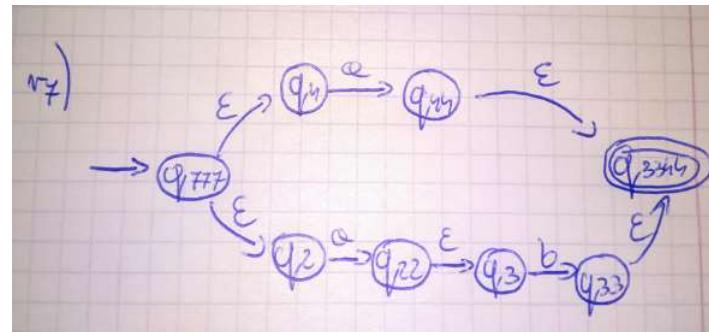
2. Teraz budujemy automaty dla wydzielonych przez nasz wyrażeń regularnych zaczynamy od góry i schodzimy coraz niżej. Wygodnie jest sobie oznaczyć kreski jakimiś symbolami aby wiedzieć, który automat odpowiada której kresce, na rysunku zaznaczyłem  $r_1, r_2, itd..$  Zaczynając od góry mamy najpierw same wyrażenia regularne typu  $a$  oraz  $b$ :



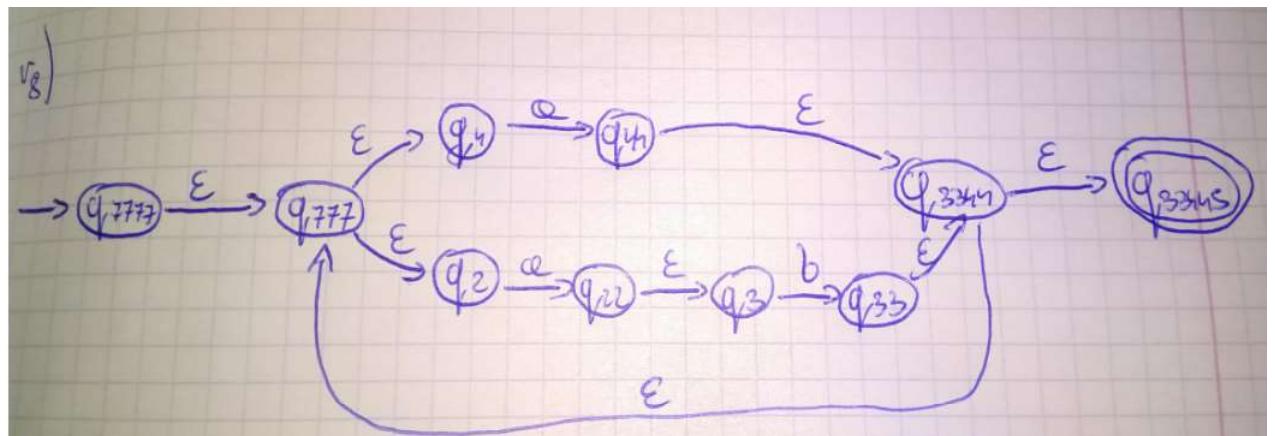
3. Teraz schodzimy poziom niżej i budujemy kolejny automat z automatów już zbudowanych zgodnie z regułą przekształcenia  $EF$ , a więc powstaje:



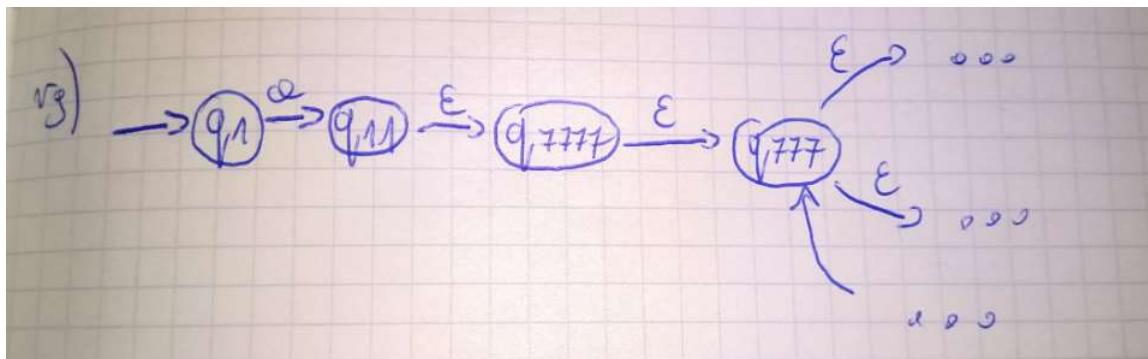
4. I dalej poziom niżej, a więc teraz reguła dla wyrażenia  $E + F$

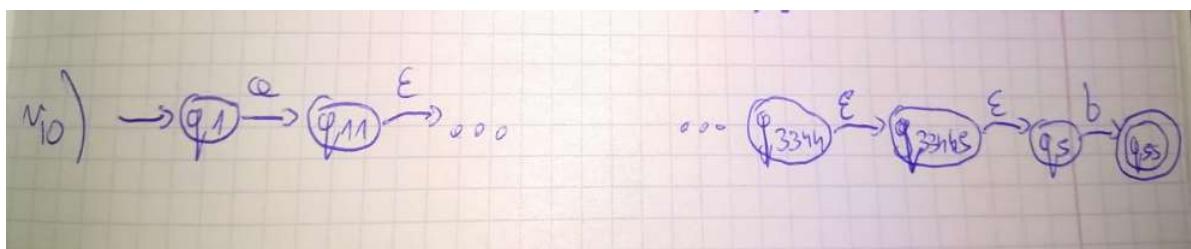


5. Na następnym poziomie reguła dla  $E^*$



6. Teraz pozostaje już tylko dodać wyrażenie  $a$  na początku i  $b$  na końcu.





Pod koniec ze względu na brak miejsca rysowałem już kropki zamiast części grafu, ale lepiej zawsze narysować cały. Zauważmy, że idąc przez kolejne etapy, coś co było stanem akceptującym może stać się zwykłym ze względu na przekształcenia.

Również ze względu na przekształcenia czasami konieczne jest wprowadzenie stanu, który nie był obecny na poprzednim poziomie (przykładowo w kroku 4 konieczny jest dla nas nowy stan początkowy oraz końcowy ponieważ wprowadzamy alternatywę).

## 5 Ćwiczenia nr 5 - UNAS, minimalizacja

---

Na ostatnich ćwiczeniach zajmowaliśmy się konwersją wyrażeń regularnych do automatów  $\varepsilon$ NAS. Pokazywaliśmy tym samym, że jeśli język jest wyrażalny za pomocą wyrażenia regularnego, to oznacza, że jest regularny.

Na tych ćwiczeniach będziemy chcieli z kolei pokazać implikację w drugą stronę, to znaczy, że jeżeli język jest regularny, to można go przedstawić przy pomocy wyrażenia regularnego. Stwierdzenie to pokażemy poprzez konwersję automatu DAS na wyrażenie regularne. Aby wykonać taką konwersję wprowadzimy sobie nowy typ automatu - Uogólniony niedeterministyczny automat stanowy (UNAS).

### 5.1. Uogólniony niedeterministyczny automat stanowy (UNAS)

UNAS to skończony niedeterministyczny automat stanowy w którym na krawędziach zamiast liter alfabetu znajdują się wyrażenia regularne. Aby uprościć sobie rozważania zawsze będziemy rozważać UNAS, które spełniać będą poniższe trzy założenia:

- Stan początkowy nie ma żadnych krawędzi wejściowych, ale ma krawędź wychodzącą do każdego stanu znajdującego się w UNAS
- Jest tylko jeden stan akceptujący, stan akceptujący nie ma krawędzi wychodzących, ale ma krawędź wchodzączą od każdego stanu znajdującego się w UNAS, stan akceptujący jest różny od stanu początkowego
- Wszystkie stany poza początkowym i akceptującym mają przejścia do wszystkich innych stanów, które nie są ani początkowym ani akceptującym

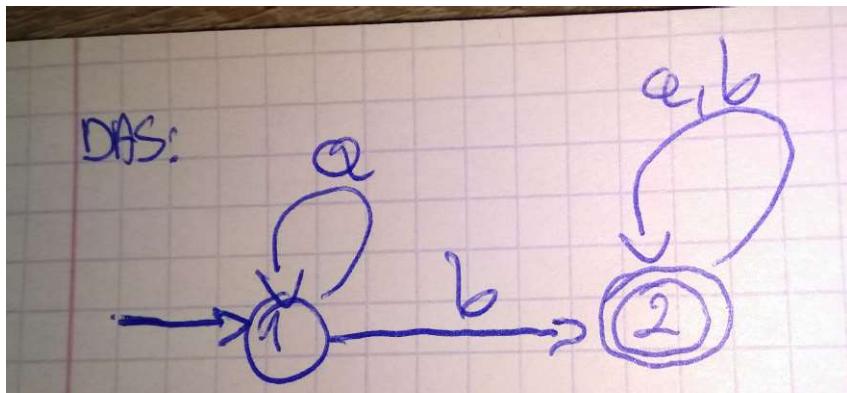
Spełnienie tych założeń ułatwia nam zamianę automatu na wyrażenia regularne.

### 5.2. Konwersja DAS do UNAS

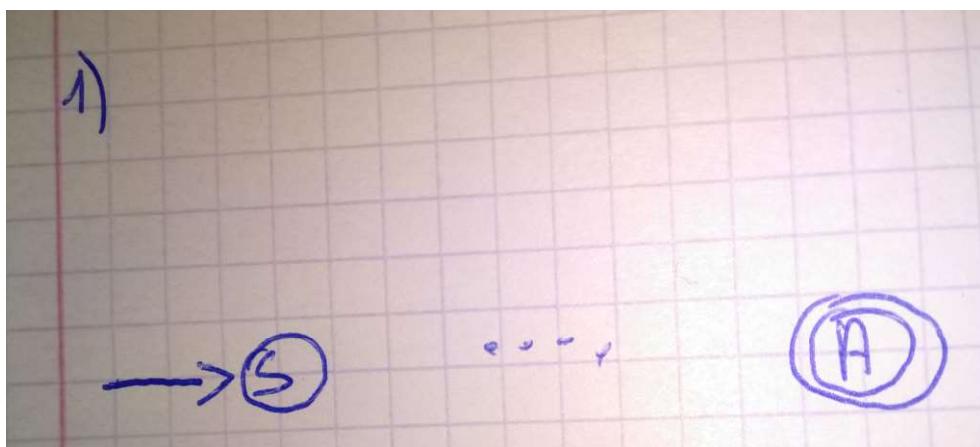
Pierwszym krokiem w celu zamiany automatu na wyrażenie regularne, jest przekształcenie go do UNAS, tak aby spełniał wymienione wyżej założenia. Aby to osiągnąć stosujemy kilka modyfikacji:

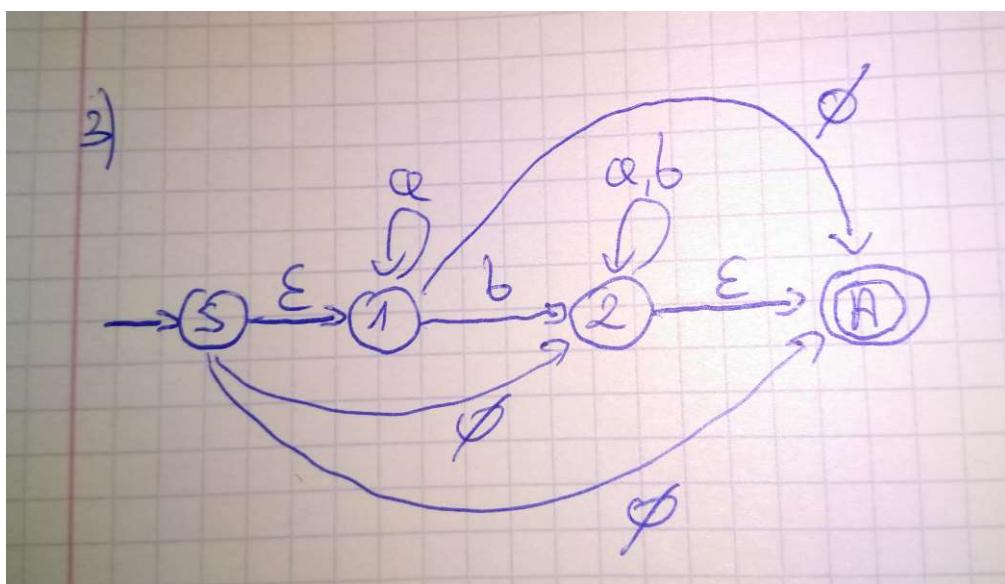
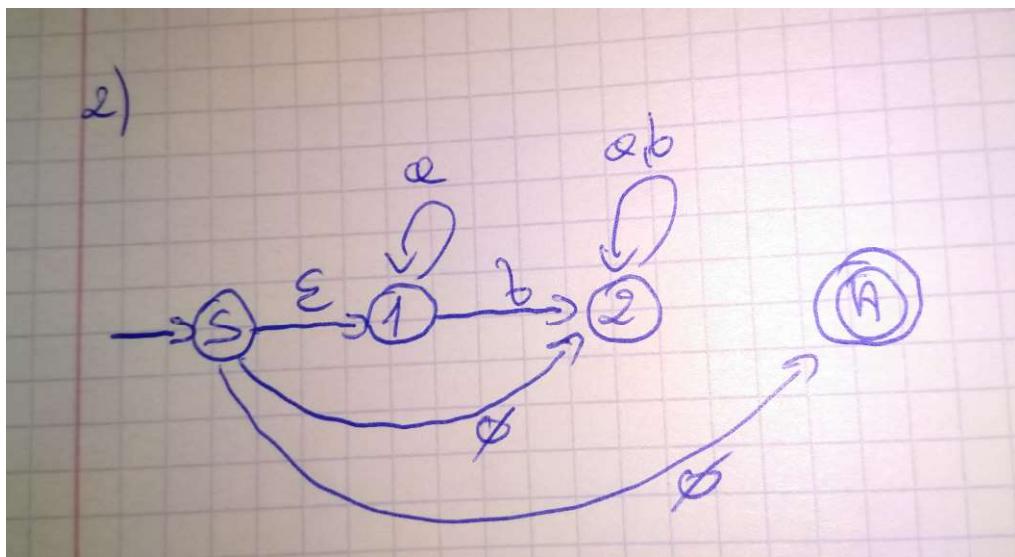
1. Dodajemy nowy stan początkowy oraz nowy stan akceptujący
2. Z nowo dodanego stanu początkowego prowadzimy krawędzie do wszystkich pozostałych stanów (także do akceptującego), krawędzie te mają etykietę zbioru pustego, poza tą, która idzie do pierwotnego stanu początkowego, ta ma etykietę  $\varepsilon$
3. Ze wszystkich stanów pierwotnego DAS prowadzimy krawędzie do stanu akceptującego i dajemy im etykietę zbioru pustego, poza tą, która idzie od pierwotnego stanu akceptującego do nowo dodanego, ta ma etykietę  $\varepsilon$
4. Ostatnia zmiana to ta dotycząca krawędzi pomiędzy stanami pierwotnego DAS, robimy dwie modyfikacje:
  - Jeśli pomiędzy dwoma stanami lub stanem a nim samym nic ma transycji, to dodajemy ją i etykietujemy zbiorem pustym
  - Jeśli transycja pomiędzy dwoma stanami ma etykietę, która zawiera więcej niż jeden symbol, to zamieniamy tą etykietę na wyrażenie regularne, które jest alternatywą tych symboli

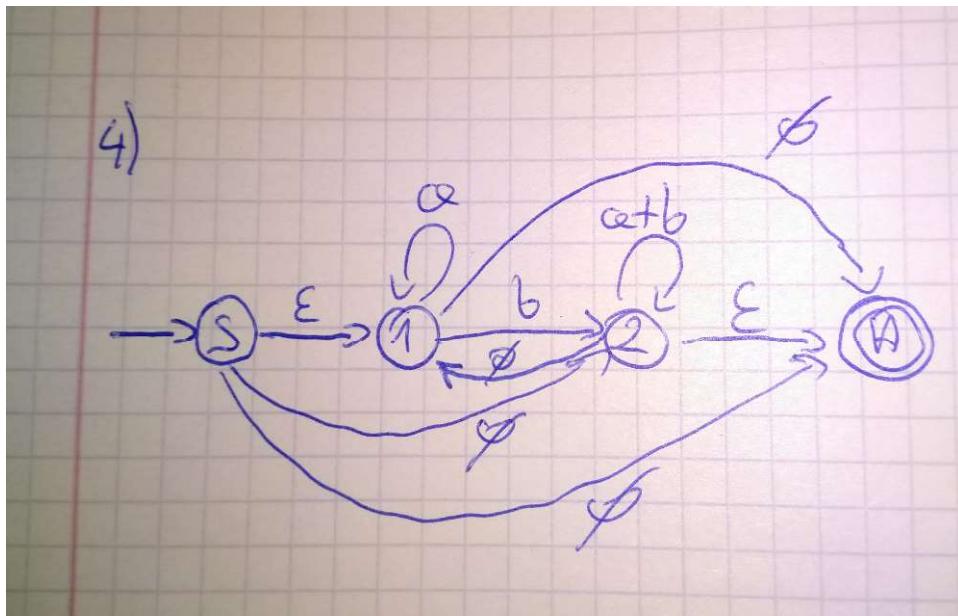
Przykład  
Dany jest DAS



Wykonajmy kolejne kroki opisane powyżej w celu przekształcenia go na UNAS:





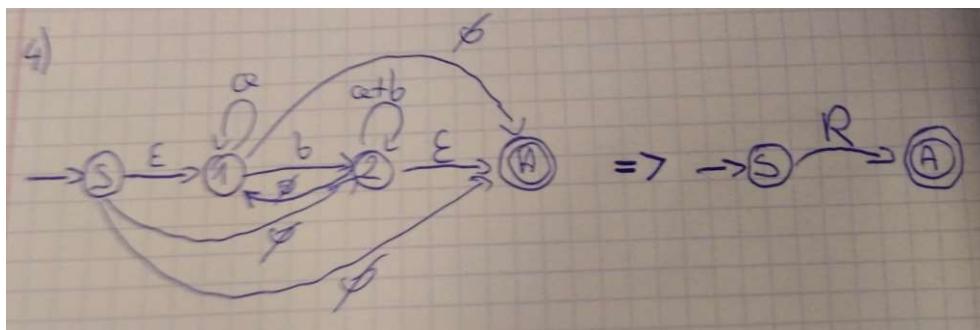


### 5.3. Algorytm zamiany UNAS na wyrażenie regularne

#### 5.3.1. Ogólna idea

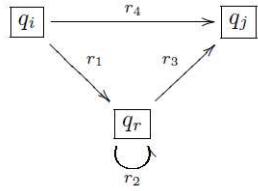
Ogólna idea tego algorytmu polega na tym, że w każdym kroku zamieniamy automat UNAS o  $k$  stanach na automat UNAS o  $k - 1$  stanach. Ostatecznie dążymy do uzyskania automatu UNAS o dwóch stanach - początkowym i akceptującym, wyrażenie regularne pomiędzy tymi stanami, to wyrażenie, które odpowiada DAS.

W naszym przykładzie będziemy więc chcieli zamienić nasz automat na taki jak po stronie prawej, gdzie  $R$  jest szukanym przez nas wyrażeniem regularnym.



#### 5.3.2. Nieformalny przykład

Mamy dany fragment UNAS taki jak poniżej, chcemy usunąć stan  $q_r$ . Usuwając ten stan pozbywamy się tym samym wszystkich krawędzi wchodzących do niego i wychodzących z niego. W pozostałej części automatu musimy uwzględnić więc to co usuwamy. Nie trudno zauważyć, że fragmenty słów przechodzące przez  $q_r$  można opisać wyrażeniem regularnym  $r_1 r_2^* r_3$ .



Takie więc wyrażenie musimy dodać w odpowiednie miejsce w pozostałą części automatu.

$$[q_i] \xrightarrow{r_1 r_2^* r_3 \cup r_4} [q_j]$$

Widać więc, że jest to procedura w pewien sposób odwrotna do tej, którą stosowaliśmy w zamianie wyrażenia regularnego na  $\varepsilon$ NAS. Tam budowaliśmy automat dodając kolejne części składowe automatu, które zastępowały fragmenty wyrażenia regularnego. Tutaj 'składamy automat', usuwając kolejne małe części składowe automatu i zastępujemy je wyrażeniem regularnym.

Ogólnie rzecz biorąc, nie ma żadnego konkretnego algorytmu, który mówi w jakiej kolejności należy usuwać stany, należy się tutaj kierować tym aby jak najbardziej uprościć sobie zadanie, a więc zacząć od eliminowania małych fragmentów automatu. Kolejne stany do usuwania wybieramy losowo, oczywiście nie możemy wybrać do usunięcia stanu początkowego i końcowego.

### 5.3.3. Przypomnienie własności wyrażeń regularnych $\emptyset$ oraz $\varepsilon$

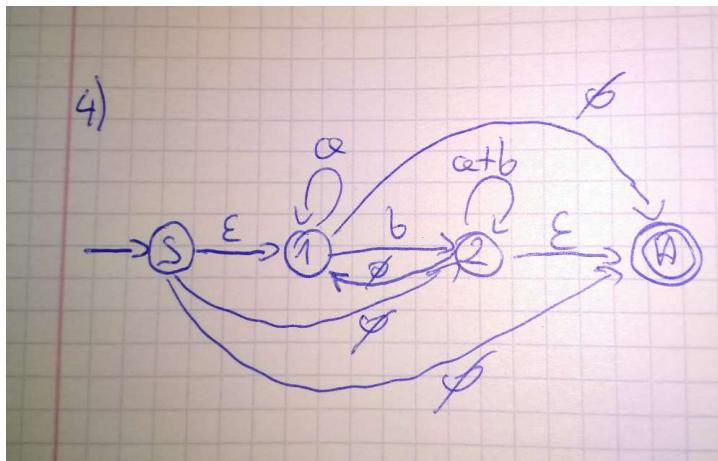
Jak zauważaliśmy w automatach UNAS często na krawędziach pojawia się wyrażenie regularne  $\emptyset$  lub też  $\varepsilon$ . Oczywiście przy usuwaniu stanów nie pomijamy tych wyrażeń, ale zawieramy je w wyrażeniach wynikowych. Powszechnie będą więc przykładowo takie wyrażenia regularne jak:  $\emptyset + \emptyset(a + b)^*\varepsilon$ , a my musimy wiedzieć jak je uprościć. Przypomnijmy sobie zatem trochę podstaw:

- $\emptyset + L = L + \emptyset = L$
- $\varepsilon L = L\varepsilon = L$
- $\emptyset L = L\emptyset = \emptyset$
- $\emptyset^* = \varepsilon$

A teraz rozwiążmy dwa przykłady, aby oswoić się w procedurą zamiany UNAS na wyrażenie regularne.

### 5.3.4. Przykład 1

Zacznijmy od przykładu, który poprzednio zamieniliśmy z DAS na UNAS:



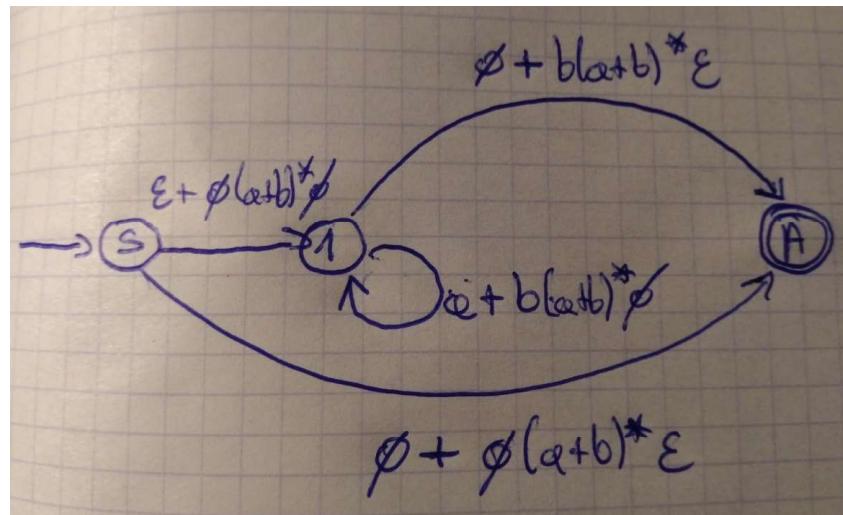
Jako pierwszy wybierzmy sobie do usunięcia stan numer 2. Przedstawię tutaj mój własny tok myślenia podczas usuwania stanu.

- Najpierw patrzymy jakie stany mają krawędzie wchodzące bezpośrednio do usuwanego stanu
- Potem jakie są osiągalne ze stanu, który usuwamy
- To tworzy nam możliwą listę przejść przez usuwany stan, dla naszego przypadku:

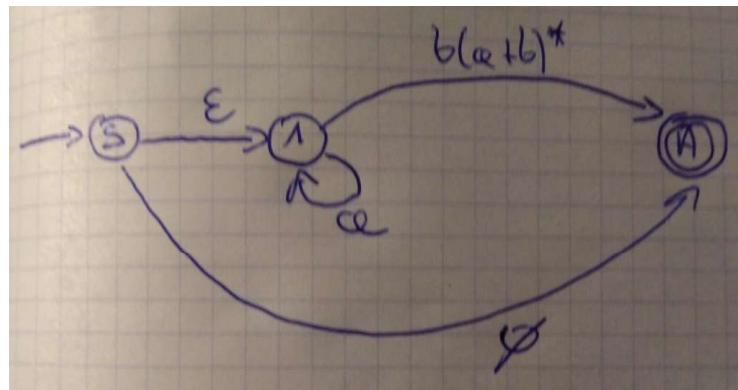
Skąd	Przez (usuwany)	Dokąd
S	2	1
S	2	A
1	2	A
1	2	1

- Wiemy, że na naszym UNAS jest zawsze krawędź pomiędzy stanami *Skąd* -> *Dokąd*, skąd to wiemy? Tutaj właśnie przydatne są nasze trzy założenia względem UNAS na których spełnienie się omówiliśmy.
- Na gałęziach *Skąd* -> *Dokąd* dodajemy alternatywę +, która po jednej stronie zawiera to co już na gałęzi było, a po drugiej to co reprezentuje usuwana droga *Skąd* -> *Przez* -> *Dokąd*

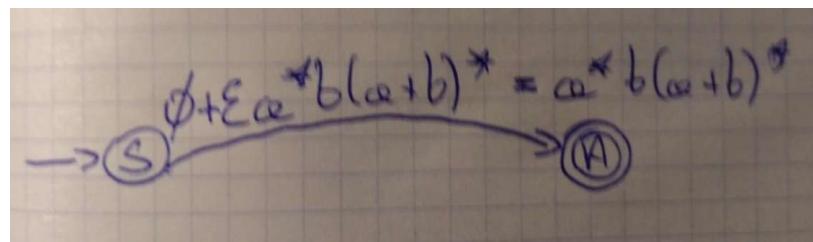
Czyli weźmy sobie np. drogę  $1- \rightarrow 2- \rightarrow A$ , patrząc na graf widzimy, że droga ta jest reprezentowana przez wyrażenie regularne  $b(a+b)^*\epsilon$ , dopiszmy więc to wyrażenia jako alternatywę na drodze  $1- \rightarrow A$ . Analogicznie postępujemy dla innych dróg prowadzących przez 2, które sobie wypisaliśmy w tabelce. Ostatecznie po wszystkim otrzymamy:



Co oczywiście możemy teraz uprościć do:

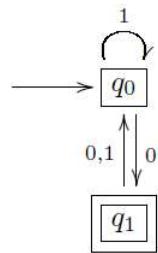


Analogicznie usuwamy stan numer 1 i otrzymujemy:

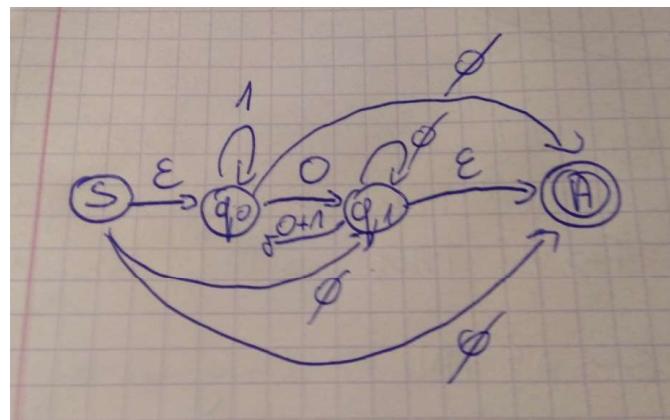


### 5.3.5. Przykład 2

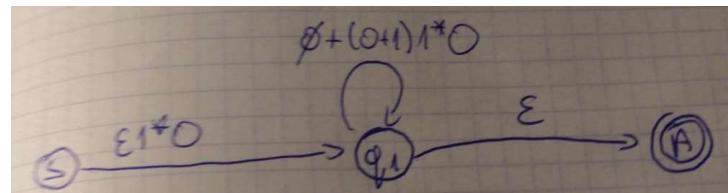
Teraz zamienimy taki automat DAS na wyrażenie regularne, tym razem już przyśpieszymy wykonywanie naszych kroków.



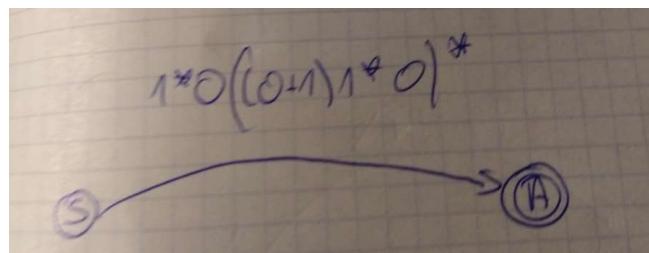
Konwersja do UNAS:



Usunięcie stanu numer 0:



Usunięcie stanu numer 1:



UWAGA: Spróbuj wykonać przekształcenia zaczynając od usunięcia stanu  $q_1$ . Czy wynik jest taki sam? Nie, wynik jest inny, ale czy jest on równoważny wyrażeniu, które uzyskaliśmy usuwając najpierw  $q_0$ ? Oczywiście powinien być, ale trudno to stwierdzić patrząc na dwa wyrażenia. Dlatego właśnie jako

następne będziemy rozważać zagadnienie minimalizacji, które pozwoli nam odpowiedzieć na pytanie, czy dwa różne automaty są równoważne (nieroróżnialne).

#### 5.4. Równoważność i minimalizacja automatów

Teraz zajmiemy się równoważnością i minimalizacją automatów. Równoważność pozwala nam na uzyskanie informacji o tym czy dwa opisy języków regularnych definiują ten sam język.

##### 5.4.1. Równoważność stanów

Aby móc stwierdzać o równoważności automatów, najpierw musimy wiedzieć jak stwierdzić czy dwa stany automatu są ze sobą równoważne, czyli czy można je zastąpić pojedynczym stanem, który będzie się zachowywał jak tamte dwa.

Definicja - Stany równoważne (nieroróżnialne)

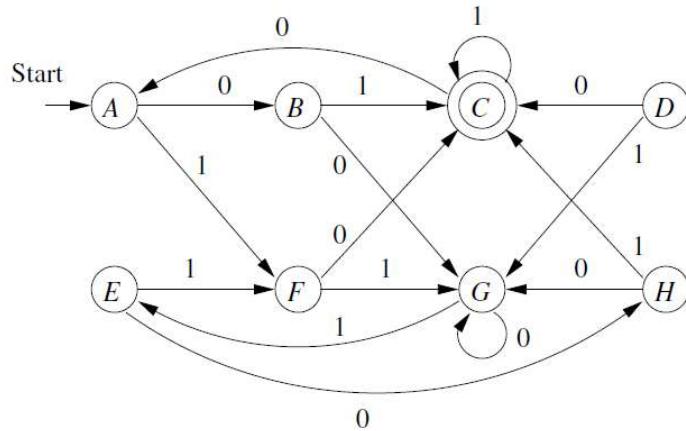
Mówimy, że dwa stany automatu  $p$  oraz  $q$  są równoważne gdy zaczynając działanie automatu od stanu  $p$  oraz od stanu  $q$  (tak jakbyśmy mieli dwie instancje tego automatu, jedna startująca od  $p$ , a druga od  $q$ ), obie te instancje zawsze osiągają równocześnie stan akceptujący lub nieakceptujący pod wpływem takich samych słów wejściowych. Nie muszą to być natomiast te same stany, ważne jest tylko czy są akceptujące czy nie.

Definicja - Stany rozróżnialne (nierównoważne)

Mówimy, że dwa stany  $p$  oraz  $q$  są rozróżnialne jeśli nie są równoważne.

Spójrzmy na przykłady

Rozważamy automat z rysunku poniżej i sprawdźmy równoważność kilku wybranych par stanów.



- Stany  $C$  i  $G$  - nie są równoważne, ponieważ jeden jest akceptującym, a drugi nie. Tym co rozróżnia te stany jest słowo puste  $\varepsilon$ , ponieważ pod jego wpływem, automat startujący w stanie  $C$  znajduje się w stanie akceptującym, a automat startujący w stanie  $G$  w stanie nieakceptującym. Istnieje więc takie słowo wejściowe, że po jego wykonaniu, dwie instancje automatu (jedna, która startuje w stanie  $C$ , a druga, która startuje w stanie  $G$ ) znajdują się w stanie akceptującym i nieakceptującym. Wystarczy, że wskażemy jedno takie słowo, i to dowodzi, że stany nie są równoważne.
- Stany  $A$  i  $G$ 
  - oba te stany są nieakceptujące więc nie są rozróżniane przez słowo puste
  - słowo wejściowe  $1$  nie rozróżnia tych stanów, ponieważ pod wpływem  $1$  w obu przypadkach znajdujemy się w stanie nieakceptującym
  - słowo wejściowe  $0$  nie rozróżnia tych stanów, ponieważ pod wpływem  $0$  idziemy w jednym przypadku do stanu  $B$ , a w drugim do  $G$  i oba te stany są nieakceptujące
  - natomiast słowo wejściowe  $01$  rozróżnia stan  $A$  od stanu  $G$ , ponieważ

$$\hat{\delta}(A, 01) = C, \hat{\delta}(G, 01) = E$$

Stan  $C$  jest akceptującym, a stan  $E$  jest nieakceptującym

- Stany  $A$  i  $E$  są nieroóżnialne, ponieważ
  - Oba są nieakceptujące więc nie rozróżnia ich  $\varepsilon$
  - Na słowo  $1$  oba te stany przechodzą do stanu  $F$ , a więc żaden kolejny symbol wejściowy nie może ich rozróżnić
  - Na słowo  $0$  stany te przechodzą do stanów odpowiednio  $B$  oraz  $H$ , oba są nieakceptujące, ale to jeszcze nie przesąduje o równoważności stanów
  - Natomiast na słowo  $0$  ze stanów  $B$  oraz  $H$  przechodzimy do tego samego stanu  $G$ , a na słowo  $1$  do tego samego stanu  $C$ , co świadczy o tym, że żadne kolejne symbole nie rozróżnią stanów, a więc ostatecznie stany  $A$  oraz  $E$  są równoważne

### 5.4.2. Algorytm wyznaczania stanów rozróżnialnych

Oczywiście losowe sprawdzanie każdej z par stanów w celu stwierdzenia o równoważności byłoby bardzo żmudną pracą. W celu wyznaczenia stanów rozróżnialnych posługujemy się algorytmem, który opiszmy sobie poniżej. Algorytm ten pozwala na znalezienie stanów rozróżnialnych w automacie, wszystkie pozostałe stany których algorytm nie uzna za rozróżnialne są nierożróżnialne (równoważne).

Algorytm ten nazywany jest algorytmem wypełniania tabelki, ponieważ opiera się na wypełnianiu części tabelki ze stanami na osiach  $X$  oraz  $Y$ . Znakiem  $x$  w tabeli oznaczamy parę stanów, która jest rozróżnialna, a puste komórki oznaczają pary stanów, które są równoważne.

### 5.4.3. Opis algorytmu

Podstawowe założenie:

Pamiętaj...

Jeżeli  $p$  jest stanem akceptującym, a  $q$  jest nieakceptującym, to para stanów  $\{p, q\}$  jest rozróżnialna.

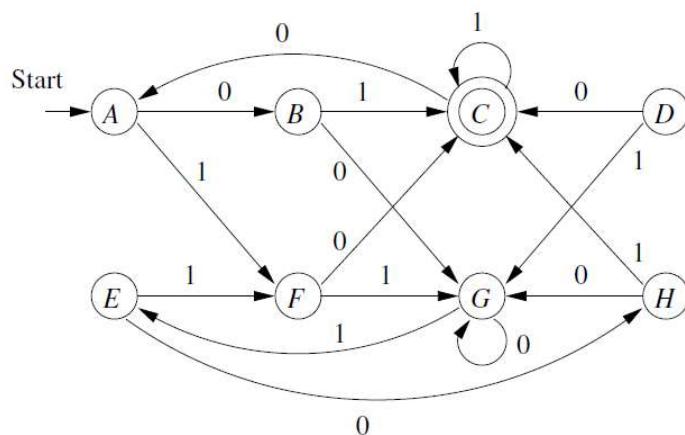
Schemat działania algorytmu opiera się o jedną prostą obserwację:

Pamiętaj...

Jeżeli wiemy, że dwa stany  $r$  oraz  $s$  są rozróżnialne przez słowo wejściowe  $w$ , to jeśli dla dwóch dowolnych stanów  $p$  oraz  $q$  istnieje takie słowo wejściowe  $a$ , że pod jego wpływem stany te osiągają dwa nierożróżnialne stany  $r$  oraz  $s$ , to stąd wynika, że stany  $p$  oraz  $q$  są rozróżnialne przez słowo wejściowe  $aw$ .

### 5.4.4. Przykład

Wykonajmy teraz algorytm dla przykładu automatu, który już rozważaliśmy:



1. W pierwszym kroku rysujemy pustą tabelkę. Na osi x zaczynamy od pierwszego stanu i zapisujemy kolejne aż do przedostatniego, na osi y zaczynamy od drugiego stanu i zapisujemy kolejne aż do ostatniego

B							
C							
D							
E							
F							
G							
H							
	A	B	C	D	E	F	G

2. W kolejnym kroku zaznaczamy w tabeli znakiem  $x$  wszystkie pary stanów nieakceptujących ze stanem akceptującym, ponieważ wiemy, że są one na pewno rozróżnialne:

B							
C	$x$	$x$					
D			$x$				
E			$x$				
F			$x$				
G			$x$				
H			$x$				
	A	B	C	D	E	F	G

3. Następnie szukamy par takich stanów, które pod wpływem tego samego symbolu wejściowego przechodzą w parę stanów, którą mamy oznaczoną w tabeli znakiem  $x$

- przykładowo  $B$  przechodzi pod wpływem  $I$  do  $C$ , a  $A$  przechodzi pod wpływem  $I$  do  $F$ , wiemy już natomiast, że stany  $\{C, F\}$  są rozróżnialne, a więc również stany  $\{B, A\}$  są rozróżnialne.

Dobrą strategią jest analizowanie na początku stanów, które mają połączenie bezpośrednio ze stanem akceptującym, łatwo dla nich znaleźć stany rozróżnialne, dlatego my zaczniemy od przeanalizowania stanów  $B, F, D, H$  w pierwszej kolejności:

B	x						
C	x	x					
D	x	x	x				
E		x	x	x			
F	x	x	x		x		
G		x	x	x		x	
H	x		x	x	x	x	x
	A	<u>B</u>	<u>C</u>	<u>D</u>	E	<u>F</u>	G

Widzimy, że z tej analizy całkiem dużo udało się nam zapełnić. Powstały też niektóre puste pola, a konkretnie pary  $(H, B)$  oraz  $(D, F)$ . Patrząc na te stany łatwo można stwierdzić, że są to stany równoważne, ponieważ zarówno dla 0 jak i 1 prowadzą do tych samych stanów. Jeśli mamy pewność, że tak jest, to możemy sobie wpisać kropkę w te miejsca:

B	x						
C	x	x					
D	x	x	x				
E		x	x	x			
F	x	x	x	.	x		
G		x	x	x		x	
H	x	.	x	x	x	x	x
	A	<u>B</u>	<u>C</u>	<u>D</u>	E	<u>F</u>	G

4. Pozostaje nam sprawdzić pary  $(A, E)$ ,  $(A, G)$  oraz  $(E, G)$

- Zaczniemy od  $(A, E)$ , łatwo zauważyc, że dla 1 przechodzą w ten sam stan, a dla 0 przechodzą w stany  $(H, B)$ , które są równoważne więc także  $(A, E)$  są równoważne
- $(A, G)$  pod wpływem 0 przechodzą w  $(B, G)$ , które są rozróżnialne, a więc  $(A, G)$  są rozróżnialne
- Podobnie  $(E, G)$  pod wpływem 0 przechodzą w  $(G, H)$ , które są rozróżnialne, a więc i  $(E, G)$  są rozróżnialne

Ostatecznie nasza tabelka ma postać:

B	x						
C	x	x					
D	x	x	x				
E	.	x	x	x			
F	x	x	x	.	x		
G	x	x	x	x	x	x	
H	x	.	x	x	x	x	x
	A	<u>B</u>	<u>C</u>	<u>D</u>	E	<u>F</u>	G

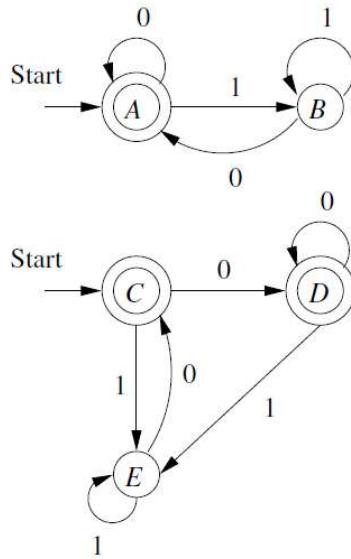
#### 5.4.5. Zastosowania algorytmu tabelkowego

Algorytm wypełniania tabelki może być stosowany do:

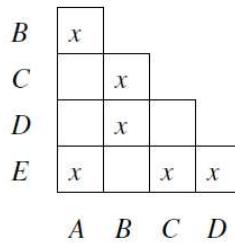
- sprawdzania czy dwa opisy języka są sobie równoważne (opisują ten sam język)
- minimalizacji automatu

#### 5.4.6. Sprawdzanie równoważności opisów języka

Aby sprawdzić przy pomocy algorytmu tabelkowego czy dwa języki są równoważne musimy przekształcić najpierw opis każdego z języków na DAS, następnie dwa otrzymane DAS, traktujemy jako jeden:



Stany początkowe nie są ważne, ponieważ nas będą obchodziły tylko przejścia pomiędzy stanami. Takie dwa DAS, traktujemy jako całość i dla niego budujemy tabelkę i sprawdzamy czy stany początkowe obu DAS są równoważne według tabelki, jeśli są, to oznacza, że języki są równoważne.



Wyszło nam, że są równoważne więc oba te automaty reprezentują ten sam język.

#### 5.4.7. Minimalizacja automatu

Częścią algorytmu minimalizacji automatu jest wyznaczanie stanów rozróżnialnych i równoważnych. Algorytm ten jest następujący:

1. Najpierw usuwamy z automatu wszystkie stany, które są nieosiagalne ze stanu początkowego (zazwyczaj takie stany nie występują, ale trzeba mieć to na uwadze) i rozważania kontynuujemy bez tych stanów
2. Następnie dzielimy stany na zbiory, takie, że:
  - stany z jednego zbioru są wszystkie równoważne ze sobą
  - żadna para stanów, które są z różnych zbiorów nie jest równoważna

Podział ten wykonujemy według algorytmu:

- Dla każdego stanu  $q$ , tworzymy zbiór składający się z  $q$  oraz wszystkich stanów z nim równoważnych, jeżeli w wyniku takiego podziału wystąpią zbiory o takich samych stanach, to zapisujemy je tylko raz w końcowym podziale
3. W zminimalizowanym automacie każdy z wyznaczonych końcowo bloków będzie stanowił osobny stan
    - Stan startowy zminimalizowanego automatu to blok, który zawiera stan startowy pierwotnego automatu
    - Stany akceptujące automatu zminimalizowanego to te zbiory stanów które zawierają stany akceptujące automatu pierwotnego, jeśli jeden ze stanów ze zbioru jest akceptujący, to wszystkie inne muszą być akceptujące (z warunku równoważności stanów w zbiorze)
  4. Ostatnim etapem jest dodanie funkcji przejścia do naszego zminimalizowanego grafu.  
Robimy to na takiej zasadzie, że biorąc blok stanów, dla każdego ze stanów w tym bloku badamy gdzie przechodzi pod wpływem danego symbolu  $a$ , musi istnieć inny blok stanów (z warunku rozróżnialności stanów z różnych bloków), który będzie zawierał wszystkie te stany, które są osiągalne ze stanów z bloku z którego wychodzimy więc do tego bloku prowadźmy krawędź z etykietą  $a$

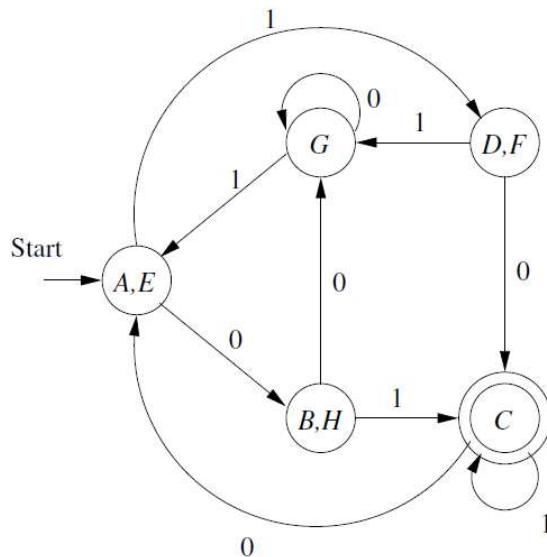
#### 5.4.8. Przykład cd.

Wszystko to może brzmieć dość skomplikowanie dlatego kontynuujmy rozwiązywanie naszego przykładu aby pokazać praktyczne działanie algorytmu. Wiemy, że nasz automat od początku nie miał stanów nieosiagalnych, następnie przeprowadziliśmy procedurę wyznaczania stanów równoważnych. W kolejnym kroku podzielimy wyznaczone stany na zbiory według krótkiego algorytmu opisanego powyżej.

- Dla stanu A równoważne są:  $\{A, E\}$
- Dla stanu B równoważne są:  $\{B, H\}$
- Dla stanu C równoważne są:  $\{C\}$
- Dla stanu D równoważne są:  $\{D, F\}$
- Dla stanu E równoważne są:  $\{A, E\}$
- Dla stanu F równoważne są:  $\{F, D\}$
- Dla stanu G równoważne są:  $\{G\}$
- Dla stanu H równoważne są:  $\{H, B\}$

Usuwając powtarzające się zbiory otrzymujemy końcowo:  $\{\{A, E\}, \{H, B\}, \{C\}, \{D, F\}, \{G\}\}$

Teraz pozostaje nam tylko dodać przejścia pomiędzy nowymi stanami:



Tak powstały automat jest automatem o najmniejszej możliwej liczbie stanów, można pokazać, że automat o mniejszej liczbie stanów nie istnieje.