

Teoretyczne podstawy informatyki



Wykład 9:
Wzorce, automaty, wyrażenia regularne.

Wzorce i automaty

- Problematyka wzorców stanowi bardzo rozwiniętą dziedzinę wiedzy. Nosi ona nazwę teorii automatów lub teorii języków, a jej podstawowe definicje i techniki stanowią istotną część informatyki.
- Poznamy trzy równoważne opisy wzorców:
 1. oparty na teorii grafów, polegać będzie na wykorzystaniu ścieżek w grafie szczególnego rodzaju który nazwiemy automatem.
 2. o charakterze algebraicznym, wykorzystujący notacje wyrażeń regularnych.
 3. oparty o wykorzystanie definicji rekurencyjnych, nazwany gramatyką bezkontekstową.

Wzorzec

- ❑ **Wzorzec** to zbiór obiektów o pewnej rozpoznawalnej właściwości.
- ❑ Jednym z typów wzorców jest zbiór ciągów znaków, taki jak zbiór poprawnych identyfikatorów języka C, z których każdy stanowi ciąg znakowy, składający się z liter, cyfr i znaków podkreślenia oraz rozpoczyna się od litery lub znaku podkreślenia.
- ❑ Inny przykład to zbiór tablic zer i jedynek o danym rozmiarze, które czytnik znaków może interpretować jako reprezentację tego samego symbolu. Poniżej trzy tablice które można interpretować jako literę **A**.
- ❑ Zbiór wszystkich takich tablic stanowiłby wzorzec o nazwie „**A**”.

```
0 0 0 1 0 0 0
0 0 1 1 1 0 0
0 0 1 0 1 0 0
0 1 1 0 1 1 0
0 1 1 1 1 1 0
1 1 0 0 0 1 1
1 0 0 0 0 0 1
```

```
0 0 0 0 0 0 0
0 0 1 0 0 0 0
0 0 1 1 0 0 0
0 1 0 1 0 0 0
0 1 1 1 0 0 0
1 0 0 1 1 0 0
1 0 0 0 1 0 0
```

```
0 0 0 1 0 0 0
0 0 1 0 1 0 0
0 1 1 0 1 0 0
0 1 1 1 1 1 0
1 1 0 0 0 1 1
1 0 0 0 0 0 1
0 0 0 0 0 0 0
```

Maszyny stanów i automaty

- **Programy służące do wyszukiwania wzorców** często charakteryzują się szczególną strukturą. W kodzie można identyfikować określone pozycje, w których można wyróżnić pewne charakterystyczne elementy związane z postępowaniem programu w zakresie osiągnięcia stawianego mu celu, jakim jest znalezienie wystąpienia wzorca.
- Takie pozycje określa się mianem **stanów** (ang. *states*). Ogólne zachowanie programu można postrzegać jako **przechodzenie od jednego stanu do drugiego** w miarę odczytywania danych wejściowych.

Przykład

□ Przykład:

- Prosty program w języku **C**, sprawdzający ciąg znaków w celu określenia czy zawiera on wszystkie pięć samogłosek w kolejności alfabetycznej. Rozpoczynając analizę od początku ciągu znaków, program najpierw wyszukuje znak **a**.
- Można powiedzieć że pozostaje w stanie **0** do czasu, aż znajdzie wystąpienie **a**, a wówczas przechodzi do stanu **1**. W stanie **1**, szuka wystąpienia znaku **e**, a kiedy je znajdzie, przechodzi do stanu **2**... Stan **i** można interpretować jako sytuację, w której program napotkał już po kolei **i** pierwszych samogłosek, gdzie **i=0,1, ..., 5**.
- Owe sześć stanów stanowi całość wymaganych przez program informacji podczas przeglądania ciągu wejściowego od lewej do prawej strony.

Przykład

```
#include <stdio.h>

#define TRUE 1
#define FALSE 0
typedef int BOOLEAN;
```

```
BOOLEAN testWord(char *p) {
    /* stan 0 */
    if (findChar(&p, 'a'))
        /* stan 1 */
        if (findChar(&p, 'e'))
            /* stan 2 */
            if (findChar(&p, 'i'))
                /* stan 3 */
                if (findChar(&p, 'o'))
                    /* stan 4 */
                    if (findChar(&p, 'u'))
                        /* stan 5 */
                        return TRUE;
    return FALSE; }
```

```
BOOLEAN findChar( char **pp, char c)
{
    while (**pp != c && **pp != '\0')
        (*pp)++;
    if( **pp == '\0')
        return FALSE;
    else {
        (*pp) ++;
        return TRUE;
    }
}
```

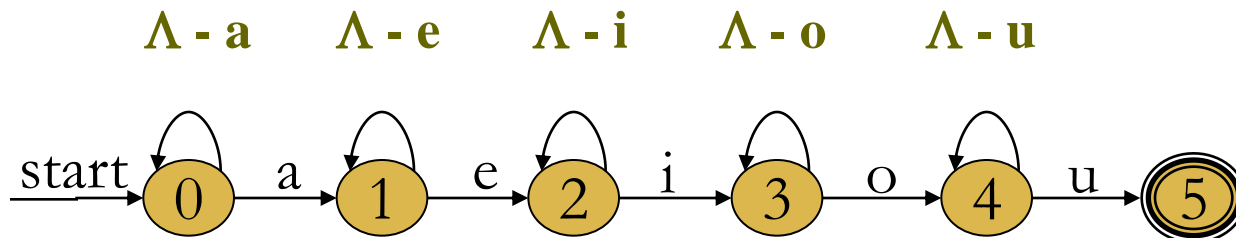
```
main()
{
    printf(“%d\n”, testWord(“abstemious”))
}
```

Grafy reprezentujące maszyny stanów

- Działanie omówionego programu można reprezentować za pomocą **grafu**, którego **wierzchołki określają stany programu**.
- I odwrotnie: program można zaprojektować przez zaprojektowanie w pierwszej kolejności grafu, a następnie mechaniczne przetłumaczenie takiego grafu na program (ręcznie lub przy pomocy istniejących narzędzi programistycznych).
- Koncepcja działania automatu jest prosta. Zakłada się że **automat pobiera listę znaków zwaną ciągiem wejściowym** (ang. input sequence).
- Jego działanie rozpoczyna się w stanie początkowym, tuż przed odczytaniem pierwszego znaku wejściowego. W zależności od tego jaki jest to znak, automat wykonuje przejście – do tego samego lub innego stanu.
- **Przejścia są zdefiniowane przez graf automatu**. Następnie automat odczytuje drugi znak i wykonuje kolejne przejście i tak dalej.

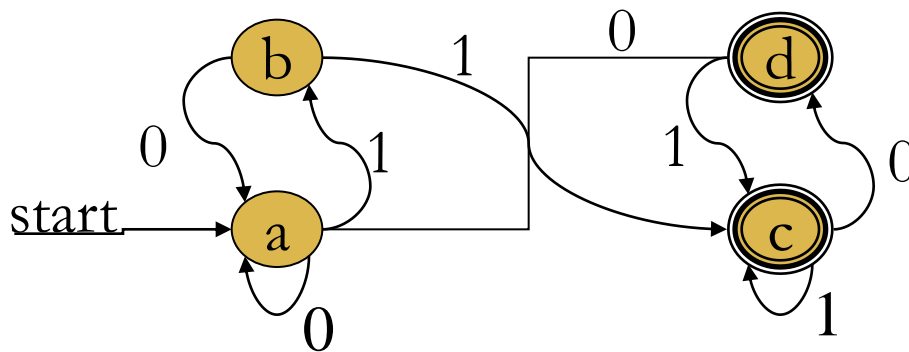
Automat rozpoznający ciągi liter

- **Stany programu są reprezentowane za pomocą grafu skierowanego**, którego krawędzie etykietuje się zbiorami znaków. Krawędzie takie nazywa się **przejściami** (ang. *transition*).
- Niektóre wierzchołki są zaznaczone jako **stany końcowe** (ang. *accepting states*). Osiągnięcie takiego stanu oznacza znalezienie poszukiwanego wzorca i jego akceptację.
- Jeden z wierzchołków jest zawsze określany jako **stan początkowy** (ang. *start state*) – stan w którym następuje rozpoczęcie procesu rozpoznawania wzorca. Wierzchołek taki oznacza się przez umieszczenie prowadzącej do niego strzałki która nie pochodzi od innego wierzchołka. Graf posiadający opisywaną postać nosi nazwę **automatu skończonego** (ang. *finite automaton*) lub po prostu automatu.



Filtr odbijający

- Automat pobiera ciąg zer i jedynek.
- Celem jego jest „wygładzenie” ciągu przez traktowanie pojedynczego symbolu **0**, który jest otoczony dwoma symbolami **1** jako „szumu” i zastąpienie go symbolem **1**.
- W podobny sposób jest traktowany symbol **1** gdy jest otoczony dwoma symbolami **0** – zastępujemy go symbolem **0**.
- Stanem początkowym jest a. Stanami końcowymi (akceptującymi) są c i d.



Wejście	Stan	Wyjście
0	a	0
1	a	0
0	b	0
1	b	0
0	a	0
1	a	0
0	b	0
1	b	0
0	c	1
1	c	1
0	d	1
1	d	1
0	c	1
1	c	1

Różnica między automatami a ich programami

- ❑ **Automaty stanowią abstrakcje.** Podejmują decyzje o akceptacji lub odrzuceniu danego ciągu znaków wejściowych przez sprawdzenie, czy istnieje ścieżka od stanu początkowego do pewnego stanu końcowego, której krawędzie będą zaetykietowane wartościami ciągu.
- ❑ Działanie filtru można interpretować jako fakt że automat odrzuca podciągi: **0,01,010,0101**, natomiast akceptuje podciągi **01011, 010110, 0101101**.
- ❑ Działanie automatu rozpoznającego litery jest takie że akceptuje ciągi znaków, takich jak „abstemiou”, ale odrzuca ciąg „abstemious”, ponieważ nie da się przejść nigdzie ze stanu 5 w przypadku końcowego znaku s.

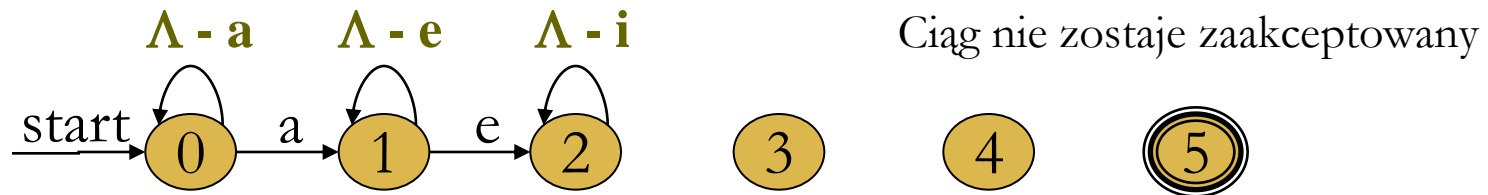
Różnica między automatami a ich programami

- ❑ **Programy tworzone dla automatów mogą podejmować decyzje o akceptacji lub odrzuceniu na różne sposoby.**
- ❑ Przykładowo program akceptujący ciągi znaków i wykorzystujący automat ze slajdu 6. powinien zaakceptować zarówno słowo „abstemious” jak i „abstemiou”.
- ❑ W momencie przetworzenia litery **u** program wypisywałby całe słowo bez dalszego badania go.
- ❑ Program wykorzystujący automat ze slajdu 8. powinien interpretować każdy stan akceptacji jako akcję polegającą na wypisaniu znaku **1**, zaś każdy stan odrzucenia jako akcję polegającą na wypisaniu znaku **0**.

Automaty deterministyczne i niedeterministyczne

- Jedną z podstawowych operacji wykonywanych za pomocą automatu jest pobranie ciągu symboli a_1, a_2, \dots, a_k i przejście od stanu początkowego ścieżką, której krawędzie posiadają etykiety zawierające te symbole po kolei.
- Tzn. dla $i = 1, 2, \dots, k$ symbol a_i należy do zbioru S_i , który etykietuje i -tą krawędź ścieżki.
- Konstruowanie takiej ścieżki oraz sekwencji jej kolejnych stanów określa się mianem **symulacji** (ang. *simulation*) automatu dla ciągu wejściowego a_1, a_2, \dots, a_k .
- O takiej ścieżce mówi się że posiada etykietę a_1, a_2, \dots, a_k .

Symulacja działania automatu ze str. 6 dla słowa "adept".



Wejście:	a	d	e	p	t	
Stan:	0	1	1	2	2	2

Automaty deterministyczne

- Automaty które dotychczas omówiliśmy posiadają pewną istotną własność.
- Dla każdego stanu s oraz dowolnego znaku wejściowego x istnieje co najwyżej jedno przejście ze stanu s , którego etykieta zawiera znak x .
- O tego rodzaju automatach mówimy że są **deterministyczne** (ang. *deterministic*).
- Symulacja automatu deterministycznego dla danej sekwencji wejściowej jest bardzo prosta. W każdym stanie s dla kolejnego znaku wejściowego x należy rozpatrzyć każdą etykietę przejść ze stanu s . Jeżeli uda się znaleźć przejście, którego etykieta zawiera znak x , to przejście to określa właściwy stan następny. Jeżeli żadna nie zawiera znaku x , to automat „zamiera” i nie może przetwarzać dalszych znaków wejściowych.

Automaty deterministyczne

- Można z łatwością przerobić automat deterministyczny na program.
- Dla każdego stanu tworzy się fragment kodu. Kod stanu **s** bada znak wejściowy i określa, które (o ile jakiegokolwiek) przejście ze stanu **s** powinno zostać wybrane.
- Jeżeli zostanie wybrane przejście ze stanu **s** do stanu **t**, to kod napisany dla stanu **s** musi uwzględniać przekazanie sterowania do kodu stanu **t**, np. przy użyciu instrukcji **go to**.

Funkcja implementująca automat deterministyczny

```
void bounce()
{
    char x;

    /* stan a */
a: putchar('0');
   x = getchar();
   if ( x == '0') goto a;
   if ( x == '1') goto b;
   goto finis;

    /* stan b */
b: putchar('0');
   x = getchar();
   if ( x == '0') goto a;
   if ( x == '1') goto c;
   goto finis;
```

```
/* stan c */
c: putchar('1');
   x = getchar();
   if ( x == '0') goto d;
   if ( x == '1') goto c;
   goto finis;

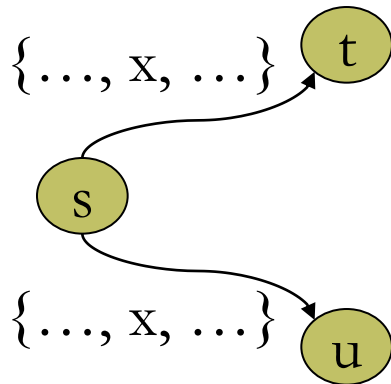
    /* stan d */
d: putchar('1');
   x = getchar();
   if ( x == '0') goto a;
   if ( x == '1') goto c;
   goto finis;

finis;

}
```

Automaty niedeterministyczne

- ❑ **Automaty niedeterministyczne** (ang. *nondeterministic*) mogą (ale nie muszą) posiadać dwa (lub więcej) przejścia z danego stanu zawierające ten sam symbol.
- ❑ Warto zauważyć, że automat deterministyczny jest równocześnie automatem niedeterministycznym, który nie posiada wielu przejść dla jednego symbolu.
- ❑ Automaty niedeterministyczne nie mogą być bezpośrednio implementowane za pomocą programów, ale stanowią przydatne pojęcie abstrakcyjne.



W momencie podjęcia próby symulacji automatu niedeterministycznego w przypadku ciągu wejściowego składającego się ze znaków a_1, a_2, \dots, a_k może okazać się, że ten sam ciąg etykietuje wiele ścieżek. Wygodnie jest przyjąć, że automat niedeterministyczny akceptuje taki ciąg wejściowy, jeżeli co najmniej jedna z etykietowanych ścieżek prowadzi do stanu akceptującego.

niedeterminizm = „zgadywanie”

Jak zastąpić automat niedeterministyczny deterministycznym?

- Automat niedeterministyczny zastępujemy deterministycznym przez skonstruowanie **równoważnego** automatu deterministycznego. Technika ta nosi nazwę **konstrukcji podzbiorów** (ang. *subset construction*).
- **Równoważność automatów:**
 - Mówimy, że automat **A** jest **równoważny** (ang. *equivalent*) automатовi **B**, jeżeli akceptują ten sam zbiór ciągów wejściowych. Innymi słowy, jeżeli $a_1a_2\dots a_k$ jest dowolnym ciągiem symboli, to spełnione są dwa następujące warunki:
 1. Jeżeli istnieje ścieżka zaetykietowana jako $a_1a_2\dots a_k$ wiodąca od stanu początkowego automatu **A** do pewnego stanu akceptującego automatu **A**, to istnieje również ścieżka zaetykietowana $a_1a_2\dots a_k$ wiodąca od stanu początkowego automatu **B** do stanu końcowego tego automatu.
 2. Jeżeli istnieje ścieżka zaetykietowana jako $a_1a_2\dots a_k$ wiodąca od stanu początkowego automatu **B** do pewnego stanu akceptującego automatu **B**, to istnieje również ścieżka zaetykietowana $a_1a_2\dots a_k$ wiodąca od stanu początkowego automatu **A** do stanu końcowego tego automatu.

Indukcyjna konstrukcja automatu niedeterministycznego

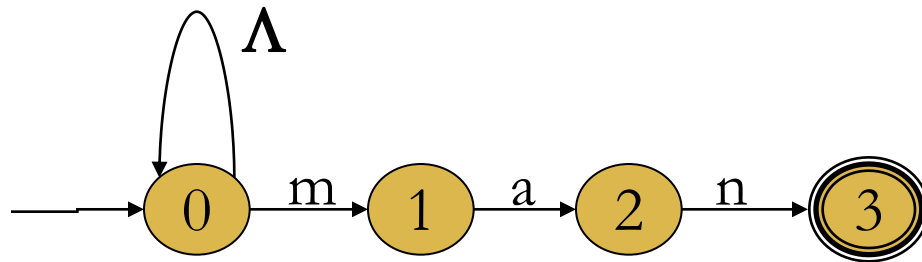
□ Podstawa:

- Jeżeli stanem początkowym automatu niedeterministycznego \mathbf{N} jest \mathbf{s}_0 , to stanem początkowym automatu deterministycznego \mathbf{D} jest $\{\mathbf{s}_0\}$, czyli zbiór zawierający tylko element \mathbf{s}_0 .

□ Indukcja:

- Załóżmy, że ustalono, iż \mathbf{S} , zbiór stanów automatu \mathbf{N} , jest stanem automatu \mathbf{D} .
- Rozpatrujemy po kolei każdy możliwy znak wejściowy \mathbf{x} . Dla danego \mathbf{x} określamy, że \mathbf{T} jest zbiorem stanów \mathbf{t} automatu \mathbf{N} , takich, że dla pewnego stanu \mathbf{s} należącego do zbioru \mathbf{S} istnieje przejście z \mathbf{s} do \mathbf{t} etykietą zawierającą znak \mathbf{x} .
- Wówczas zbiór \mathbf{T} jest stanem automatu \mathbf{D} i istnieje przejście z \mathbf{S} do \mathbf{T} względem znaku wejściowego \mathbf{x} .

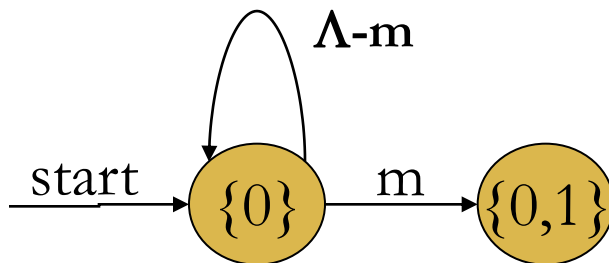
Konstrukcja automatu deterministycznego D



Niedeterministyczny automat rozpoznający ciąg znaków kończący się sekwencją "man".

Rozpoczynamy od zbioru $\{0\}$, który jest stanem początkowym automatu D .

Stan $\{0\}$ i jego przejścia



Dla dowolnej litery oprócz m ze stanu 0 następuje przejście do stanu 0, w przypadku m następuje przejście do stanu 0 lub 1. Automat D potrzebuje stanu $\{0\}$, który już posiada oraz stanu $\{0,1\}$ który należy dodać.

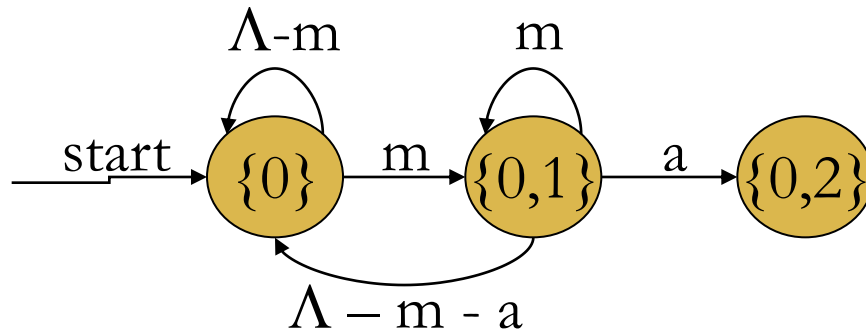
Konstrukcja automatu deterministycznego D

Następnie, należy rozważyć przejścia ze stanu $\{0,1\}$.

Dla niedeterministycznego automatu, wszystkie znaki wejściowe oprócz m i a powodują przejście ze stanu 0 tylko do stanu 0 , a ze stanu 1 nie można nigdzie przejść. Stąd istnieje przejście ze stanu $\{0,1\}$ do stanu $\{0\}$,

zaetykietowane ($\Lambda - m - a$). W przypadku wejściowego znaku m , ze stanu 1 nie można przejść nigdzie, a ze stanu 0 można przejść do stanu 0 lub 1 .

Dlatego istnieje przejście $\{0,1\}$ do niego samego zaetykietowane litera m . W przypadku znaku wejściowego a , ze stanu 0 przechodzi się do niego samego, ze stanu 1 przechodzi się do stanu 2 . Stąd przejście zaetykietowane przez a ze stanu $\{0,1\}$ do stanu $\{0,2\}$.

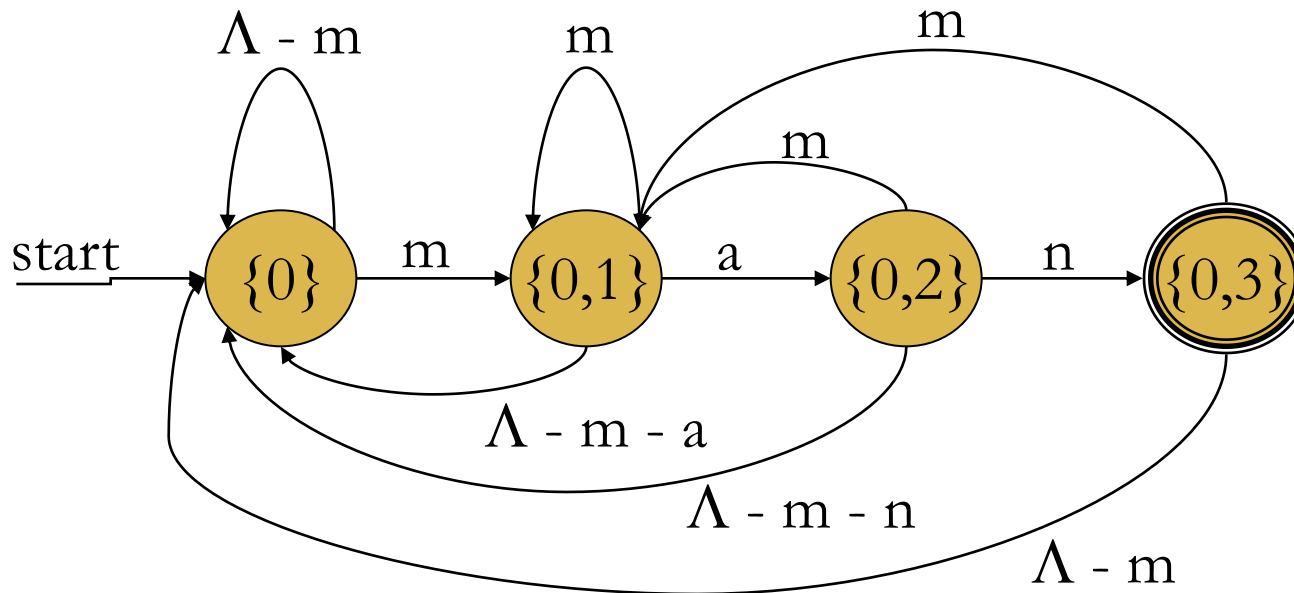


Stan $\{0\}$ i $\{0,1\}$ oraz ich przejścia

Konstrukcja automatu deterministycznego **D**

- Teraz należy skonstruować przejścia ze stanu $\{0,2\}$. W przypadku wszystkich znaków wejściowych oprócz **m** i **n**, ze stanu **0** można przejść jedynie do stanu **0**, zaś ze stanu **2** nie można nigdzie przejść. Zatem istnieje przejście ze stanu $\{0,2\}$ do stanu $\{0\}$ zaetykietowane wszystkimi literami oprócz **m** i **n**.
- W przypadku znaku wejściowego **m**, ze stanu **2** nie można nigdzie przejść, a ze stanu **0** można przejść zarówno do stanu **0** jak i stanu **1**. Zatem istnieje przejście ze stanu $\{0,2\}$ do stanu $\{0,1\}$ zaetykietowane przez **m**.
- W przypadku znaku wejściowego **n**, ze stanu **0** można przejść tylko do tego samego stanu, zaś ze stanu **2** przechodzi się do stanu **3**. zatem istnieje przejście ze stanu $\{0,2\}$ do stanu $\{0,3\}$ zaetykietowane przez **n**.
- Ten stan automatu **D** jest stanem akceptującym gdyż zawiera stan akceptujący automatu niedeterministycznego, czyli stan **3**.
- W końcu należy określić przejścia ze stanu $\{0,3\}$. Z uwagi na to że ze stanu **3** nie można nigdzie przejść w przypadku dowolnego znaku wejściowego, przejścia ze stanu $\{0,3\}$ odpowiadają przejściom tylko ze stanu **0**, stąd ograniczają się jedynie do przejść do stanu $\{0\}$. Ponieważ przejścia ze stanu $\{0,3\}$ nie prowadzą do żadnego stanu automatu **D**, którego jeszcze nie sprawdziliśmy, konstrukcję automatu **D** można uznać za skończoną.

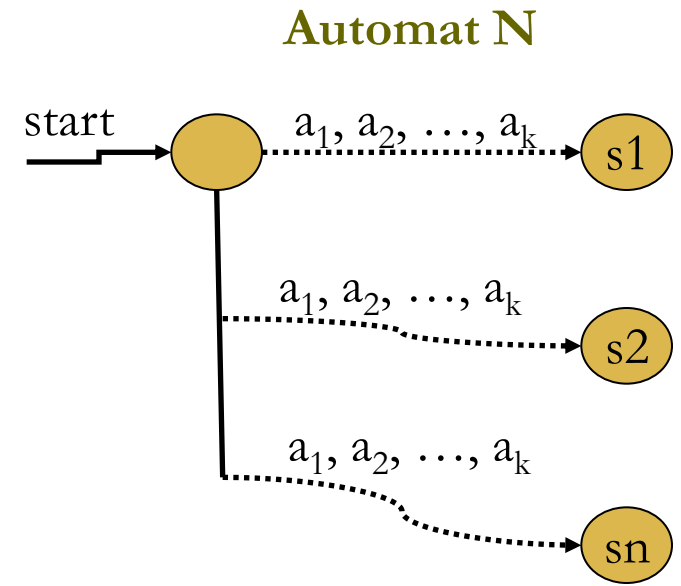
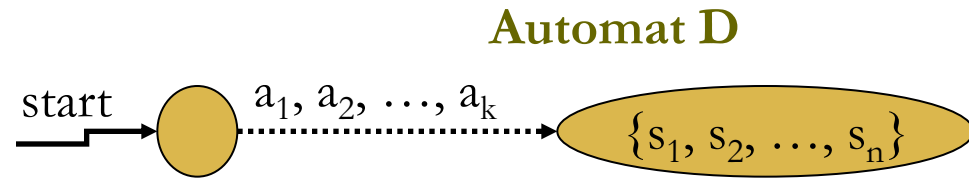
Automat deterministyczny D



Konstrukcje automatu można uznać za skończoną. Przejścia ze stanu $\{0,3\}$ nie prowadzi do żadnego stanu automatu **D** którego jeszcze nie sprawdziliśmy. Stan $\{0\}$ oznacza że odczytany ciąg nie kończy się żadnym przedrostkiem wyrazu **man**, stan $\{0,1\}$ że kończy się sekwencją **m**, stan $\{0,2\}$ że kończy się sekwencją **ma**, stan $\{0,3\}$ że kończy się sekwencją **man**.

Uzasadnienie poprawności konstrukcji podzbiorów

- Jeżeli automat **D** jest konstruowany na podstawie automatu niedeterministycznego **N** przy użyciu konstrukcji podzbiorów, to **D** jest automatem deterministycznym.
- Dla każdego stanu wejściowego **x** oraz dla każdego stanu **S** automatu **D** definiuje się określony stan **T** automatu **D**, taki, że etykieta przejścia z **S** do **T** zawiera symbol **x**.
- Należy jeszcze wykazać że automaty **S** i **T** są równoważne. Przeprowadza się dowód indukcyjny.



Uzasadnienie poprawności konstrukcji podzbiorów

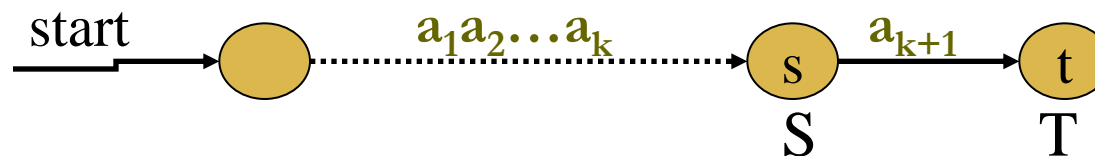
- Formalne twierdzenie, które należy udowodnić przez indukcję względem k , mówi że stan $\{s_1, s_2, \dots, s_n\}$ osiągnąony przez automat D w wyniku przejścia ścieżką zaetykietowaną symbolami $a_1 a_2 \dots a_k$ od stanu początkowego automatu D jest dokładnie zbiorem stanów automatu N , które są przez ten automat osiągnane w wyniku przejścia od stanu początkowego pewną ścieżką zaetykietowaną symbolami $a_1 a_2 \dots a_k$.
- **Podstawa:**
 - Niech $k=0$. Ścieżka o długości 0 nie powoduje przejścia do żadnego innego stanu, co oznacza, że powoduje pozostanie w stanie początkowym obu automatów D i N . Należy przypomnieć, że jeżeli s_0 jest stanem początkowym automatu N , to stanem początkowym automatu D jest $\{s_0\}$. Stąd twierdzenie indukcyjne jest spełnione dla $k=0$.
- **Indukcja:**
 - Załóżmy, że twierdzenie jest spełnione dla k i rozważmy ciąg wejściowy $a_1, a_2, \dots, a_k, a_{k+1}$. Wówczas ścieżka wiodąca od stanu początkowego automatu D do stanu T , zaetykietowana stanami $a_1 a_2 \dots a_k a_{k+1}$, ma postać:



Uzasadnienie poprawności konstrukcji podzbiorów

- Można założyć przez hipotezę indukcyjną, że S jest dokładnie zbiorem stanów, które osiąga automat N od swojego stanu początkowego wzdłuż ścieżki zaetykietowanej symbolami $a_1 a_2 \dots a_k$ i należy udowodnić, że T jest dokładnie zbiorem stanów, które automat N osiąga od swojego stanu początkowego wzdłuż ścieżki zaetykietowanej symbolami $a_1, a_2, \dots, a_k, a_{k+1}$. Dowód kroku indukcyjnego składa się z dwóch części.

 1. Należy wykazać, że zbiór T nie zawiera zbyt wielu elementów. To znaczy, jeżeli t jest stanem automatu N należącym do T , to stan t należy do ścieżki zaetykietowanej symbolami $a_1, a_2, \dots, a_k, a_{k+1}$ wiodącej ze stanu początkowego automatu N .
 2. Należy wykazać że zbiór T zawiera wystarczającą ilość elementów. To znaczy, jeżeli t jest stanem automatu N osiąganym ze stanu początkowego ścieżką zaetykietowaną symbolami $a_1, a_2, \dots, a_k, a_{k+1}$, to t należy do T .



Minimalizacja automatów

- Jednym z zagadnień dotyczących automatów jest kwestia określenia **minimalnej liczby stanów** wymaganych do wykonania danego zadania.
- Posiadając pewien automat możemy zadać pytanie czy istnieje równoważny automat posiadający mniejszą liczbę stanów, a jeśli tak, jaka jest najmniejsza liczba stanów dowolnego automatu równoważnego.
- **Dla automatów deterministycznych zawsze istnieje pewien unikatowy automat deterministyczny o minimalnej liczbie stanów, równoważny z danym automatem.**
- Dowodzimy przez pokazanie że nie ma stanów nierównoważnych.
- Nie istnieje podobna teoria w sytuacji automatów niedeterministycznych.
- **Nierównoważność dwóch stanów:**
 - **Podstawa:** Jeżeli stan **s** jest stanem akceptującym, a stan **t** nie jest stanem akceptującym (lub odwrotnie), to **s** i **t** nie są równoważne.
 - **Indukcja:** Jeżeli istnieje pewien symbol wejściowy **x**, taki, że ze stanów **s** i **t** następują przejścia względem tego symbolu do dwóch różnych stanów, o których wiadomo że nie są równoważne, to stany **s** i **t** nie są równoważne.

Posumowanie

- ❖ Automat skończony jest to oparty na modelu grafowym sposób określania wzorców. Występują jego dwie odmiany: **automat deterministyczny** oraz **automat niedeterministyczny**.
- ❖ Istnieje możliwość konwersji automatu deterministycznego na program rozpoznający wzorce.
- ❖ Istnieje możliwość konwersji automatu niedeterministycznego na automat deterministyczny rozpoznający te same wzorce dzięki konstrukcji podzbiorów.
- ❑ Istnieje bliski związek między automatami deterministycznymi a programami, które wykorzystują pojęcie stanu w celu odróżnienia ról, jakie odgrywają różne części programu. Zaprojektowanie automatu deterministycznego stanowi często dobry sposób opracowania takiego programu. Może to być trudne zadanie, często łatwiej zaprojektować automat niedeterministyczny, a następnie za pomocą konstrukcji podzbiorów zamienić go na jego odpowiednik deterministyczny.

Wyrażenia regularne

- ❑ **Wyrażenia regularne** (ang. *regular expressions*) stanowią algebraiczny sposób definiowania wzorców.
- ❑ Wyrażenia regularne stanowią analogię do algebry wyrażeń arytmetycznych oraz do algebry relacyjnej.
- ❑ Zbiór wzorców które można wyrazić w ramach algebry wyrażeń regularnych odpowiada dokładnie zbiorowi wzorców, które można opisać za pomocą automatów.

Operandy wyrażeń regularnych

- Wyrażenia regularne posiadają pewne rodzaje operandów niepodzielnych (ang. *atomic operands*). Poniżej lista:
 1. Znak
 2. Symbol ϵ
 3. Symbol \emptyset
 4. Zmienna która może być dowolnym wzorcem zdefiniowanym za pomocą wyrażenia regularnego.
- **Wartość wyrażenia regularnego jest wzorcem** składającym się ze zbioru ciągów znaków, który często **określa się mianem języka** (ang. *language*).
- Język określony przez wyrażenia regularne **E** oznaczony będzie jako **L(E)** lub określany jako **język wyrażenia E**.

Języki operandów niepodzielnych

- Języki operandów niepodzielnych definiuje się w następujący sposób.
 1. Jeżeli x jest dowolnym znakiem, to wyrażenie regularne x oznacza język $\{x\}$, to znaczy $L(x) = \{x\}$.
Należy zauważyć, że taki język jest zbiorem zawierającym jeden ciąg znakowy. Ciąg ten ma długość 1 i jedyna pozycja tego ciągu określa znak x .
 2. $L(\epsilon) = \{\epsilon\}$. Specjalny symbol ϵ jako wyrażenie regularne oznacza zbiór, którego jedynym ciągiem znakowym jest ciąg pusty, czyli ciąg o długości 0 .
 3. $L(\emptyset) = \emptyset$. Specjalny symbol \emptyset jako wyrażenie regularne oznacza zbiór pustych ciągów znakowych.
- Istnieją trzy operatory w odniesieniu do wyrażeń regularnych.
- Można je grupować przy użyciu nawiasów, podobnie jak ma to miejsce w przypadku innych znanych algebr.
- Definiuje się prawa kolejności działań oraz prawa łączności, które pozwalają na pomijanie niektórych par nawiasów – tak jak w przypadku wyrażeń arytmetycznych.

Operatory wyrażeń regularnych

□ Suma:

- Symbol sumy (ang. *union*) oznacza się za pomocą symbolu $|$. Jeżeli \mathbf{R} i \mathbf{S} są dwoma wyrażeniami regularnymi, to $\mathbf{R} | \mathbf{S}$ oznacza sumę języków określanych przez \mathbf{R} i \mathbf{S} . To znaczy $\mathbf{L}(\mathbf{R} | \mathbf{S}) = \mathbf{L}(\mathbf{R}) \cup \mathbf{L}(\mathbf{S})$.
- $\mathbf{L}(\mathbf{R})$ i $\mathbf{L}(\mathbf{S})$ są zbiorami ciągów znakowych, notacja sumowania jest uzasadniona.

□ Złożenie:

- Operator złożenia (ang. *concatenation*) nie jest reprezentowany przez żaden odrębny symbol.
- Jeżeli \mathbf{R} i \mathbf{S} są wyrażeniami regularnymi to \mathbf{RS} oznacza ich złożenie. $\mathbf{L}(\mathbf{RS})$, czyli język określony przez \mathbf{RS} , jest tworzony z języków $\mathbf{L}(\mathbf{R})$ i $\mathbf{L}(\mathbf{S})$ w sposób następujący:
 - Dla każdego ciągu znakowego \mathbf{r} należącego do $\mathbf{L}(\mathbf{R})$ oraz każdego ciągu znakowego \mathbf{s} należącego do $\mathbf{L}(\mathbf{S})$, ciąg \mathbf{rs} , czyli złożenie ciągów \mathbf{r} i \mathbf{s} , należy do $\mathbf{L}(\mathbf{RS})$.
- **Złożenie dwóch list takich jak ciągi znaków, jest wykonywane przez pobranie po kolei elementów pierwszej z nich i uzupełnienie ich po kolei elementami drugiej listy.**

Operatory wyrażeń regularnych

□ Domknięcie:

- Operator domknięcia (ang. *closure*), jest to operator jednoargumentowy przyrostkowy. Domknięcie oznacza się za pomocą symbolu $*$, tzn. R^* oznacza domknięcie wyrażenia regularnego R . Operator domknięcia ma najwyższy priorytet.
- Efekt działania operatora domknięcia można zdefiniować jako „określenie występowania zera lub większej liczby wystąpień ciągów znaków w R ”.
- Oznacza to że $L(R^*)$ składa się z:
 1. Ciagu pustego ϵ , który można interpretować jako brak wystąpień ciągów znaków w $L(R)$.
 2. Wszystkich ciągów znaków języka $L(R)$. Reprezentują one jedno wystąpienie ciągów znaków w $L(R)$.
 3. Wszystkich ciągów znaków języka $L(RR)$, czyli złożenia języka $L(R)$ z samym sobą. Reprezentują one dwa wystąpienia ciągów znaków z $L(R)$.
 4. Wszystkich ciągów znaków języka $L(RRR)$, $L(RRRR)$ i tak dalej, które reprezentują trzy, cztery i więcej wystąpień ciągów znaków z $L(R)$.
- Nieformalnie można napisać: $R^* = \epsilon \mid R \mid RR \mid RRR \mid \dots$
- **Wyrażenie po prawej stronie to nie jest wyrażeniem regularnym ponieważ zawiera nieskończoną liczbę wystąpień operatora sumy. Wszystkie wyrażenia regularne są tworzone ze skończonej liczby wystąpień operatorów.**

Kolejność operatorów wyrażeń regularnych

- Istnieje określona kolejność wykonywania trzech działań wyrażeń regularnych: sumy, złożenia oraz domknięcia. Kolejność ta jest następująca:
 1. Domknięcie (najwyższy priorytet)
 2. Złożenie
 3. Suma (najniższy priorytet)
- **Przykład:**
$$a | bc^*d = (a | (b (c^*)) d)$$

Prawa algebraiczne wyrażeń regularnych

- Możliwe jest aby dwa wyrażenia regularne określały ten sam język.
- Dwa wyrażenia regularne $\mathbf{R} \mid \mathbf{S}$ oraz $\mathbf{S} \mid \mathbf{R}$ określają ten sam język bez względu na postać wyrażeń regularnych jakie się podstawią za \mathbf{R} i \mathbf{S} . Wynika to z faktu że sumowanie jest przemienne.
- Dwa wyrażenia regularne są **równoważne** (ang. *equivalent*) jeżeli $\mathbf{L(R)} = \mathbf{L(S)}$.

Prawa algebraiczne wyrażeń regularnych

- Tożsamość sumowania:

$$(\emptyset \mid R) \equiv (R \mid \emptyset) \equiv R$$

- Tożsamość złożenia:

$$\varepsilon R \equiv R \varepsilon \equiv R$$

- Anihilator złożenia:

$$\emptyset R \equiv R \emptyset \equiv \emptyset$$

- Przemienność sumowania: $(R \mid S) \equiv (S \mid R)$

- Łączność sumowania:

$$((R \mid S) \mid T) \equiv (R \mid (S \mid T))$$

- Łączność złożenia:

$$((RS)T) \equiv (R(ST))$$

Prawa algebraiczne wyrażeń regularnych

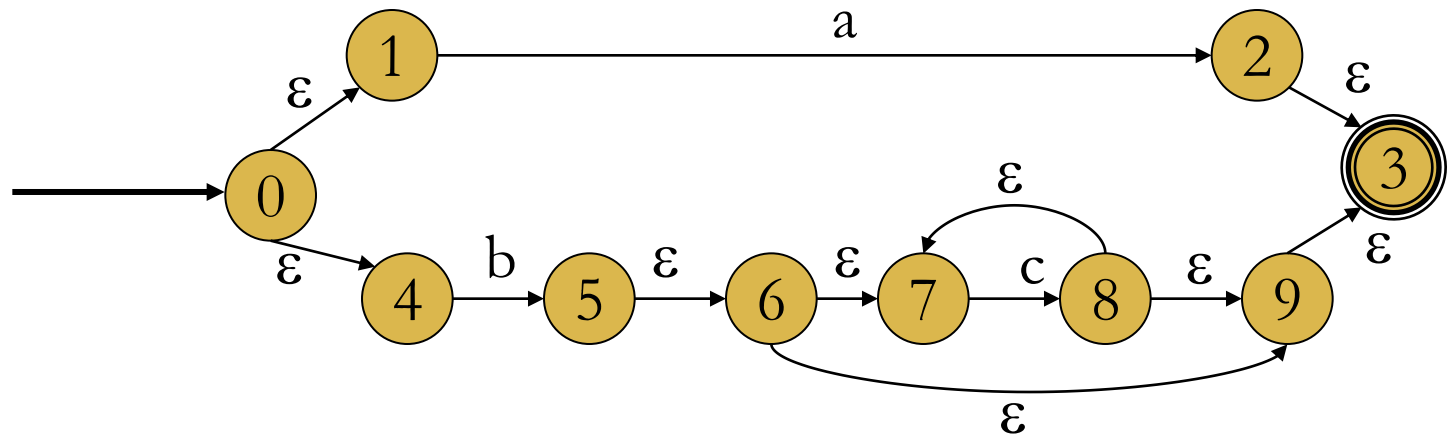
- Lewostronna rozdzielność złożenia względem sumowania:
$$(R(S \mid T)) \equiv (RS \mid RT)$$
- Prawostronna rozdzielność złożenia względem sumowania:
$$((S \mid T)R) \equiv (SR \mid TR)$$
- Idempotencja sumowania:
$$(R \mid R) \equiv R$$
- Równoważności operatora domknięcia:
$$\emptyset^* \equiv \varepsilon$$
$$RR^* \equiv R^*R$$
$$(RR^* \mid \varepsilon) \equiv R^*$$

Od wyrażeń regularnych do automatów

- Istnieje sposób na zamianę dowolnego wyrażenia regularnego na automat niedeterministyczny, a następnie przez użycie konstrukcji podzbiorów – zamiany takiego automatu na automat deterministyczny.
- Istnieje także możliwość zamiany dowolnego automatu na wyrażenie regularne, którego język dokładnie odpowiada zbiorowi ciągów znaków akceptowanych przez automat. Stąd automaty i wyrażenia regularne dają te same możliwości opisywania języków.

Automaty z epsilon przejściami

- Należy rozszerzyć notację używaną w przypadku automatów w celu umożliwienia opisu krawędzi **posiadających etykietę ϵ** . Takie automaty wciąż akceptują ciąg znaków **s** wtedy i tylko wtedy, gdy ścieżka zaetykietowana ciągiem **s** wiedzie od stanu początkowego do stanu akceptującego. **Symbol ϵ** , ciąg pusty, **jest „niewidoczny”** w ciągach znaków, stąd w czasie konstruowania etykiety danej ścieżki w efekcie usuwa się wszystkie symbole **ϵ** i używa tylko rzeczywistych znaków.



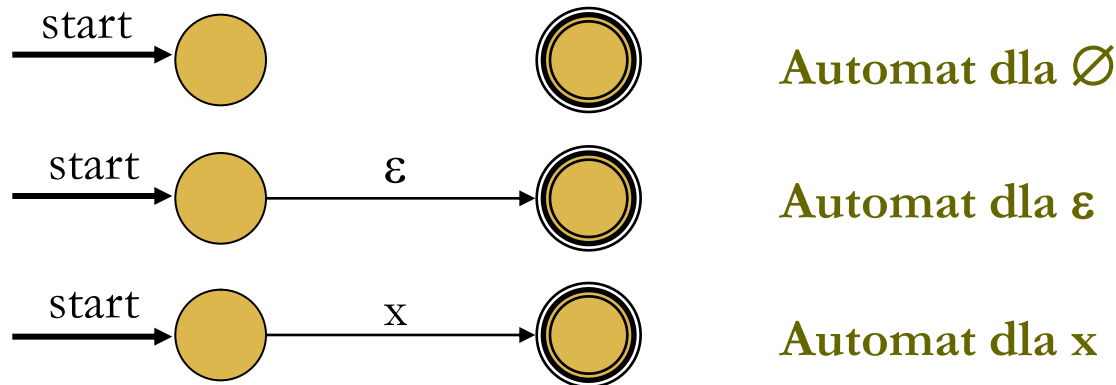
Automat z ϵ -przejściami dla wyrażenia $a \mid bc^*$

Od wyrażeń regularnych do automatów z epsilon przejściami

- Wyrażenie regularne zamienia się na automat przy użyciu algorytmu opracowanego na podstawie indukcji zupełnej względem liczby wystąpień operatorów w wyrażeniu regularnym.
- **Twierdzenie S(n):**
 - Jeżeli **R** jest wyrażeniem regularnym o **n** wystąpieniach operatorów i braku zmiennych jako operatorów niepodzielnych, to istnieje automat **A** z ϵ -przejściami, który akceptuje ciągi znaków należące do języka **L(R)** i żadne inne.
 - Ponadto automat **A**:
 1. posiada tylko jeden stan akceptujący,
 2. nie posiada krawędzi wiodących do jego stanu początkowego,
 3. nie posiada krawędzi wychodzących z jego stanu akceptującego.

Podstawa

- Jeżeli $n=0$, to R musi być operandem niepodzielnym, którym jest \emptyset , ε lub x dla pewnego symbolu x .
- Dla owych trzech przypadków można zaprojektować 2-stanowy automat, spełniający wymagania twierdzenia $S(0)$.



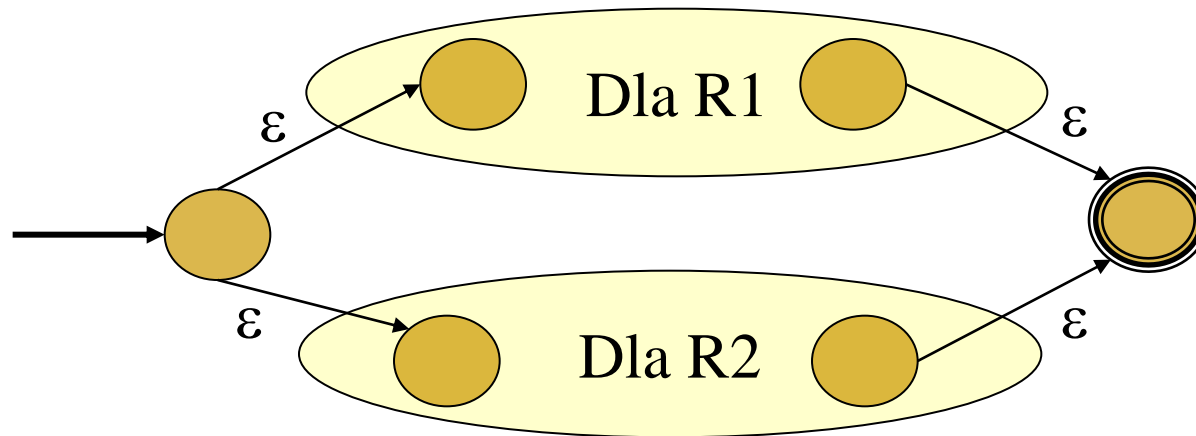
- **Automaty dla przypadków bazowych.**
Każdy spełnia warunki 1, 2, 3 (patrz poprzednia strona).

Indukcja

- Zakładamy teraz, że $S(i)$ jest prawdziwe dla wszystkich $i \leq n$.
- To znaczy, że dla każdego wyrażenia regularnego R o maksymalnie n wystąpieniach istnieje automat spełniający warunek hipotezy indukcyjnej i akceptujący wszystkie ciągi znaków języka $L(R)$ i żadnych innych.
- Zajmiemy się tylko najbardziej zewnętrznym operatorem w R , co oznacza, że wyrażenie R może mieć tylko formę
$$R1 \mid R2, \quad R1 R2, \quad R1^*$$
w zależności od tego czy ostatni użyty operator był operatorem sumy, złożenia lub domknięcia.
- Wyrażenie $R1, R2$ nie mogą posiadać więcej niż n operatorów.

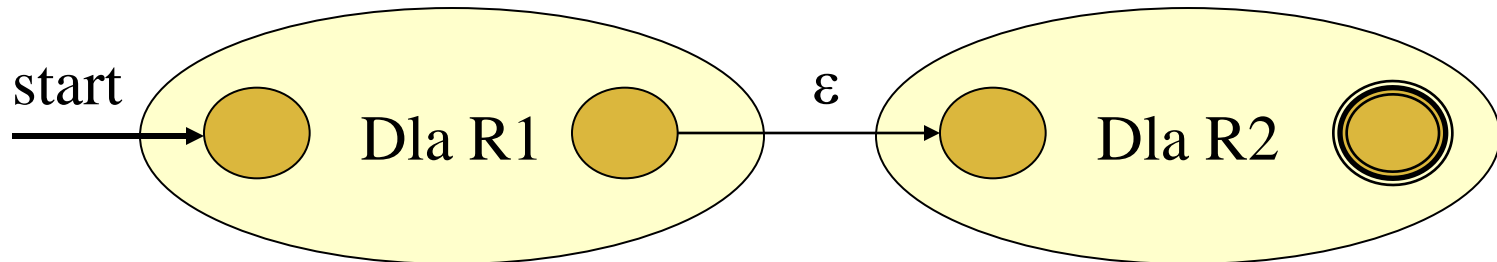
Przypadek 1: $R = R1 \mid R2$

- ❑ Przechodzimy krawędzią zaetykietowaną symbolem ϵ do stanu początkowego automatu dla **R1** lub automatu dla **R2**.
- ❑ Następnie przechodzimy do stanu akceptującego tego automatu, a później przejściem ϵ do stanu akceptującego automatu **R**.



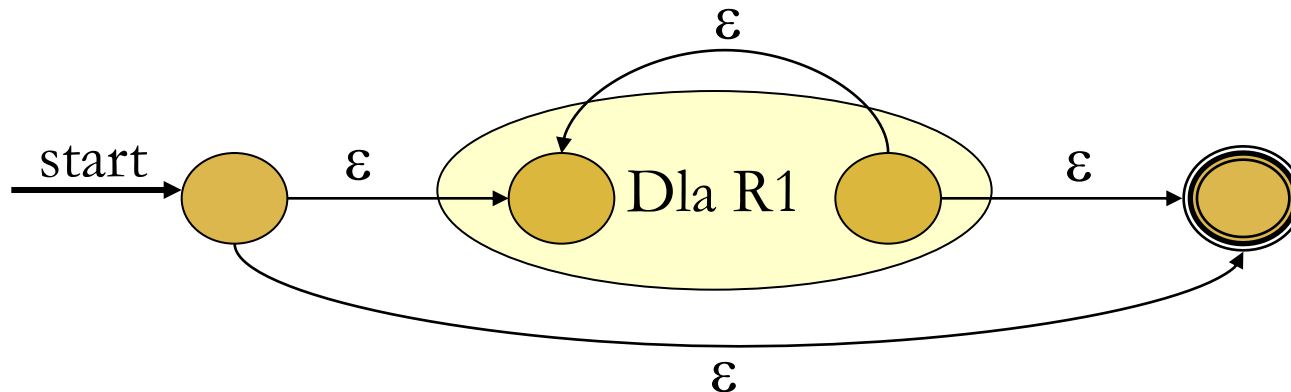
Przypadek 2: $R = R1 R2$

- Automat posiada jako swój stan początkowy stan początkowy automatu dla wyrażenia **R1**, a jako swój stan akceptujący – stan akceptujący dla wyrażenia **R2**.
- Dodajemy także ϵ - przejście ze stanu akceptującego automatu dla wyrażenia **R1** do stanu początkowego automatu dla wyrażenia **R2**.
- Stan akceptujący pierwszego automatu przestaje być stanem akceptującym, a stan początkowy drugiego automatu przestaje być stanem początkowym w skonstruowanym automacie.



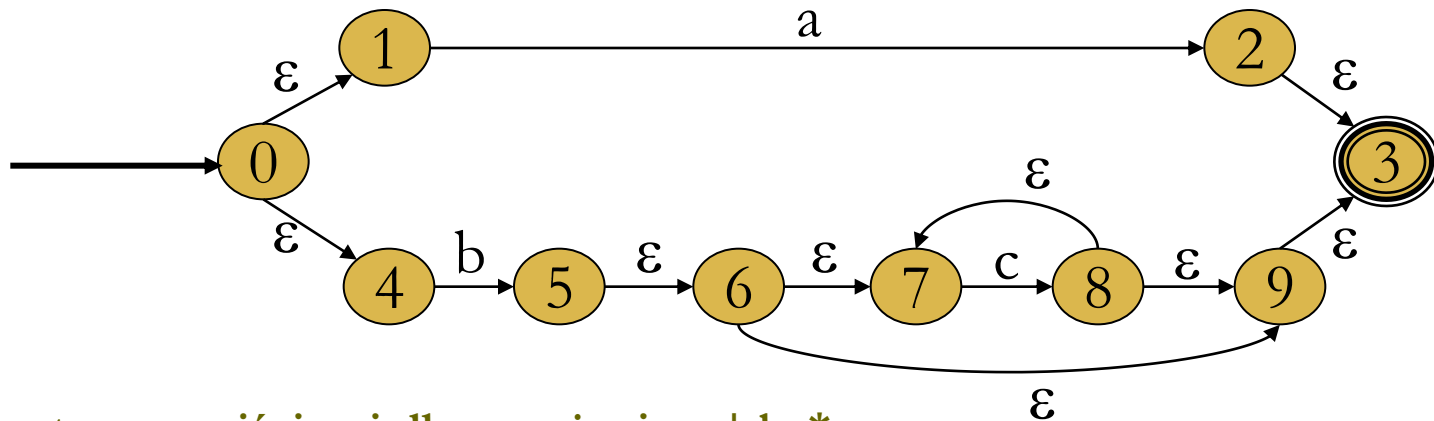
Przypadek 3: $R = R1^*$

- Do automatu dla wyrażenia **R1** dodajemy nowy stan początkowy i akceptujący.
- Stan początkowy posiada ϵ przejście do stanu akceptującego (a więc akceptowany jest ciąg ϵ) oraz do stanu początkowego automatu dla wyrażenia **R1**.
- Stan akceptujący automatu dla wyrażenia **R1** otrzymuje ϵ -przejście z powrotem do swojego stanu początkowego oraz do stanu akceptującego automatu dla wyrażenia **R**.
- Stan początkowy i akceptujący automatu dla wyrażenia **R1** nie są stanami początkowym i akceptującym konstruowanego automatu. Etykiety ścieżek odpowiadają ciągom należącym do języka $L(R1^*)$ czyli $L(R)$.



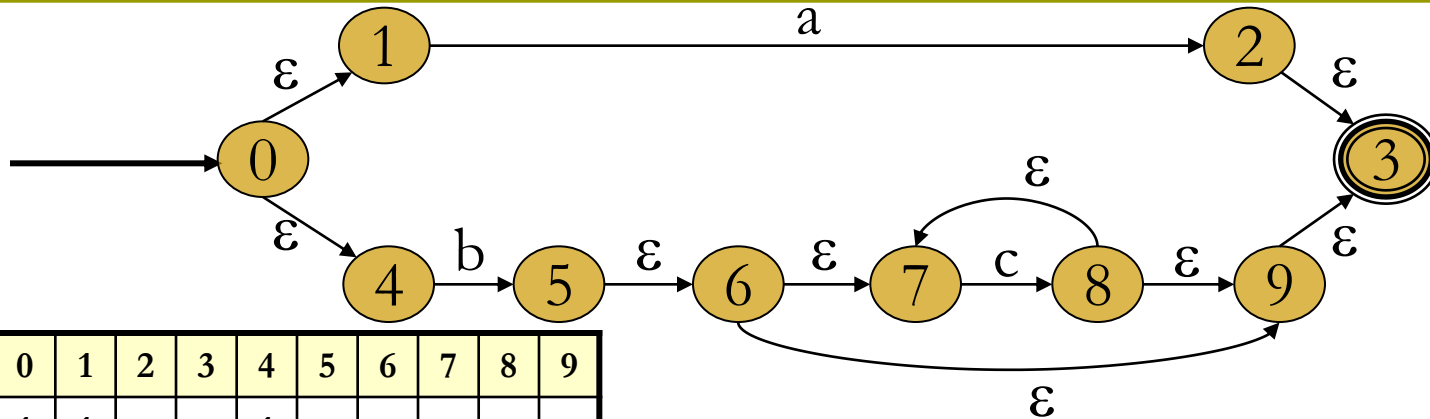
Eliminacja epsilon-przejęć

- Jeżeli stanem bieżącym jest dowolny stan s automatu z ϵ -przejęciami, oznacza to że jednocześnie stanem bieżącym jest dowolny stan, do którego można się dostać z s w wyniku przejścia ścieżki zawierającej krawędzie zaetykietowane symbolem ϵ .
- Wynika to z faktu, że bez względu na to, jaki ciąg etykietuje wybraną ścieżkę prowadzącą do s , ten sam ciąg będzie także stanowił etykietę ścieżki rozszerzonej o ϵ -przejęcia.



Automat z ϵ -przejęciami dla wyrażenia $a \mid bc^*$

Eliminacja ϵ przejść

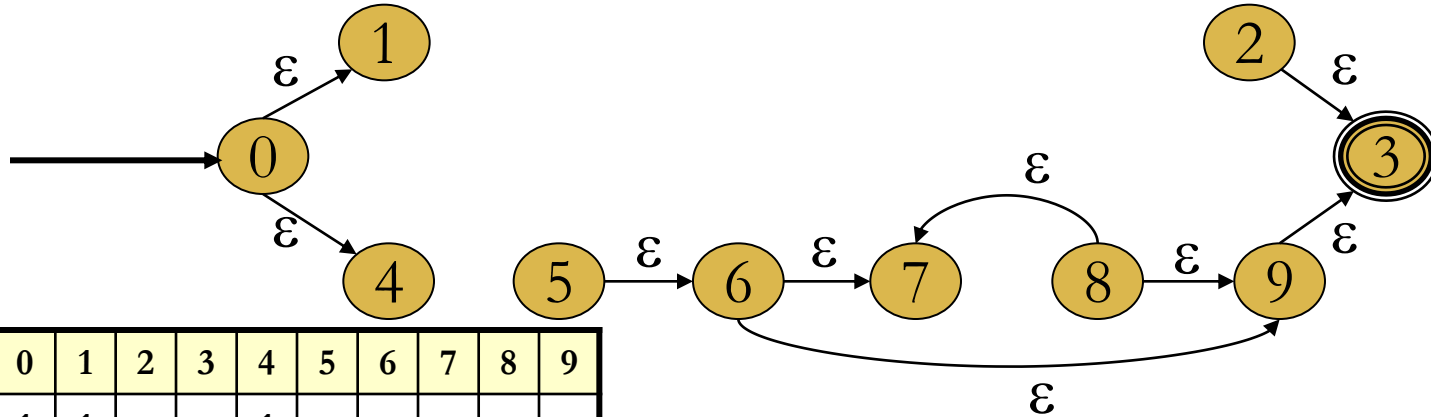


	0	1	2	3	4	5	6	7	8	9
0	1	1			1					
1		1								
2			1	1						
3				1						
4					1					
5				1		1	1	1		1
6				1			1	1		1
7				1				1		
8				1				1	1	1
9				1						1

Automat z ϵ -przejściami dla wyrażenia $a \mid bc^*$

- Dla grafu automatu usuwamy wszystkie ścieżki oznaczone rzeczywistymi etykietami. Przeprowadzamy przeszukiwanie w głąb pozostałego grafu.

Tabela osiągalności



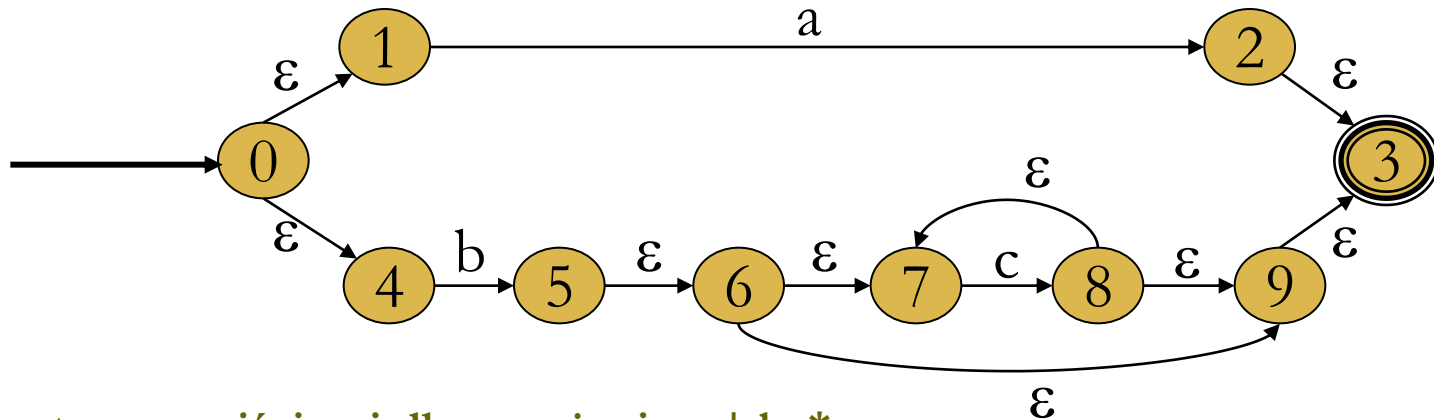
	0	1	2	3	4	5	6	7	8	9
0	1	1			1					
1		1								
2			1	1						
3				1						
4					1					
5				1		1	1	1		1
6				1			1	1		1
7				1				1		
8				1				1	1	1
9				1						1

Automat z ϵ -przejściami dla wyrażenia $a | bc^*$

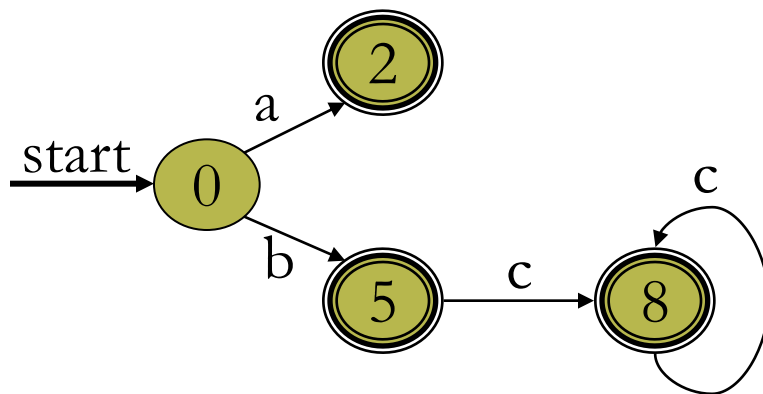
- Dla grafu automatu usuwamy wszystkie ścieżki oznaczone rzeczywistymi etykietami. Przeprowadzamy przeszukiwanie w głąb pozostałego grafu.

Tabela osiągalności

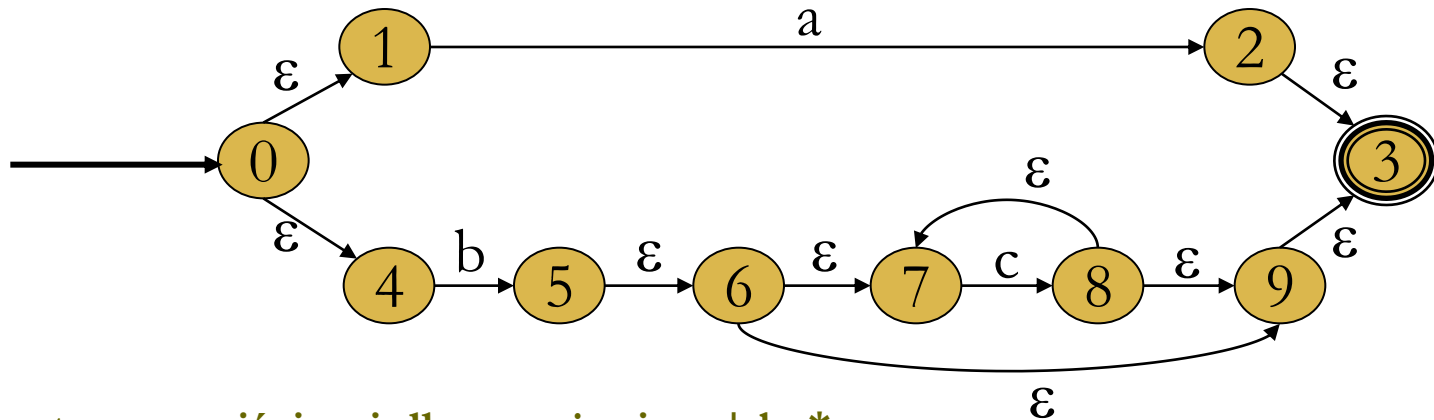
- Posiadając informacje o osiągalności, możemy skonstruować równoważny automat nie posiadający ϵ -przejęć. Stany do których przechodzi się krawędziami zaetykietowanymi symbolami rzeczywistymi nazywamy **stanami istotnymi**.
- W nowym automacie chcemy zawrzeć tylko te stany oraz stany początkowe dla zbioru jego własnego zbioru stanów. Należy też zdecydować które stany będą stanami akceptującymi.



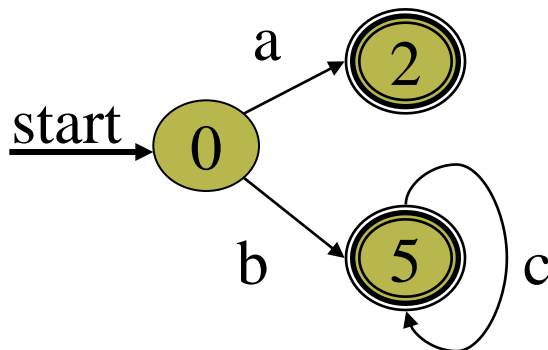
Automat z ϵ -przejściami dla wyrażenia $a \mid bc^*$



Automat skonstruowany na podstawie eliminacji ϵ -przejść. Automat akceptuje wszystkie ciągi języka $L(a \mid bc^*)$.



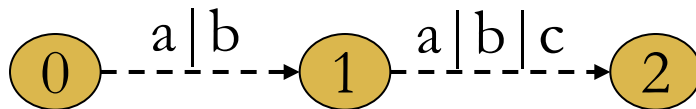
Automat z ϵ -przejściami dla wyrażenia $a \mid bc^*$



- Prostsza wersja automatu dla języka $L(a \mid bc^*)$.
- Wykorzystano obserwację ze stany (5) i (8) są równoważne i mogą zostać połączone.

Od automatów do wyrażeń regularnych.

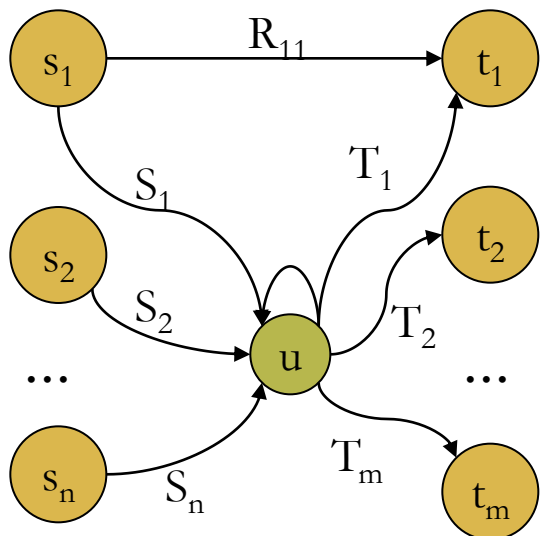
- Dla każdego automatu istnieje **A** wyrażenie regularne, którego język dokładnie odpowiada zbiorowi ciągu znaków akceptowanych przez automat **A**.
- Konstrukcja polega na eliminacji stanów automatów. Etykiety krawędzi, które są zbiorami znaków, zastępuje się bardziej skomplikowanymi wyrażeniami regularnymi.
- Jeżeli dla pewnej krawędzi istnieje etykieta $\{x_1, x_2, \dots, x_n\}$, zastępuje się ją wyrażeniem regularnym $x_1 \mid x_2 \mid \dots \mid x_n$, które reprezentuje ten sam zbiór symboli.
- Etykiety ścieżki można postrzegać jako złożenie wyrażeń regularnych opisujących krawędzie tej ścieżki, lub jako język zdefiniowany przez złożenie tych wyrażeń.
- **Przykład:**
 - Wyrażenia regularne etykietujące krawędzie to $a \mid b$ i $a \mid b \mid c$. Zbiór znaków etykietujących tę ścieżkę składa się z tych, które występują w języku zdefiniowanym przez wyrażenia regularne: $(a \mid b)(a \mid b \mid c)$ czyli $\{aa, ab, ac, ba, bb, bc\}$.



Ścieżka z wyrażeniami regularnymi jako etykietami. Etykieta ścieżki należy do wyrażeń regularnych utworzonych w wyniku złożenia.

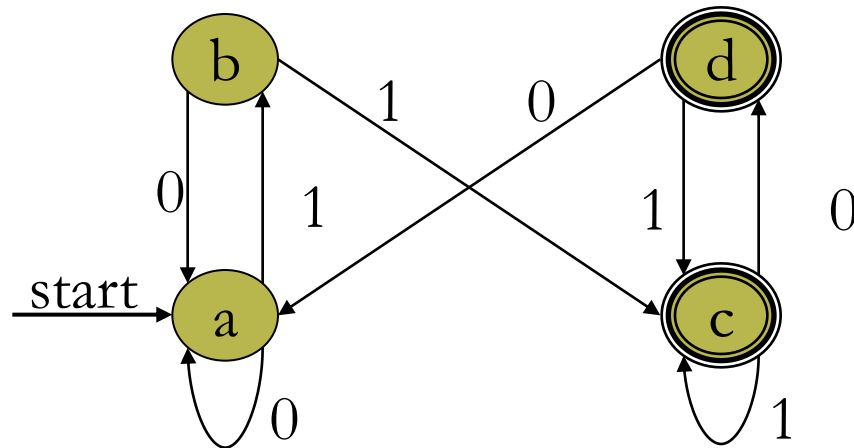
Konstrukcja eliminacji stanów.

- Kluczowym etapem konwersji z postaci automatu na wyrażenie regularne jest eliminacja stanów. Chcemy wyeliminować stan u , ale chcemy zachować etykiety krawędzi występujące w postaci wyrażen regularnych, tak aby zbiór etykiet ścieżek między dowolnymi pozostałymi stanami nie uległ zmianie.
- Poprzedniki stanu u to s_1, s_2, \dots, s_n zaś następniki stanu u to t_1, t_2, \dots, t_m (mogą też istnieć stany wspólne).



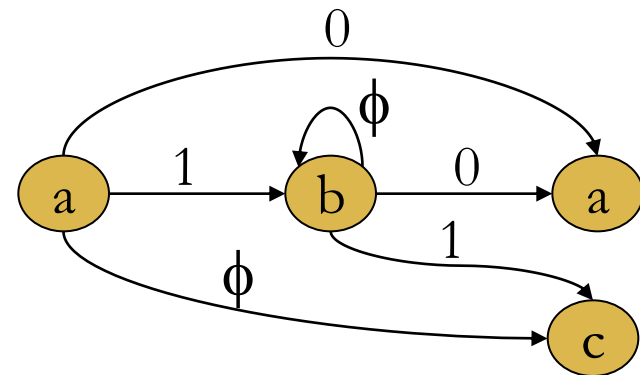
Zbiór ciągów znaków etykietujących ścieżki wiodące z wierzchołków s_i do wierzchołka u , włącznie z ścieżkami biegnącymi kilkakrotnie wokół pętli $u \rightarrow u$, oraz z wierzchołka u do wierzchołka t_j , jest opisany za pomocą wyrażenia regularnego $S_i U^* T_j$.

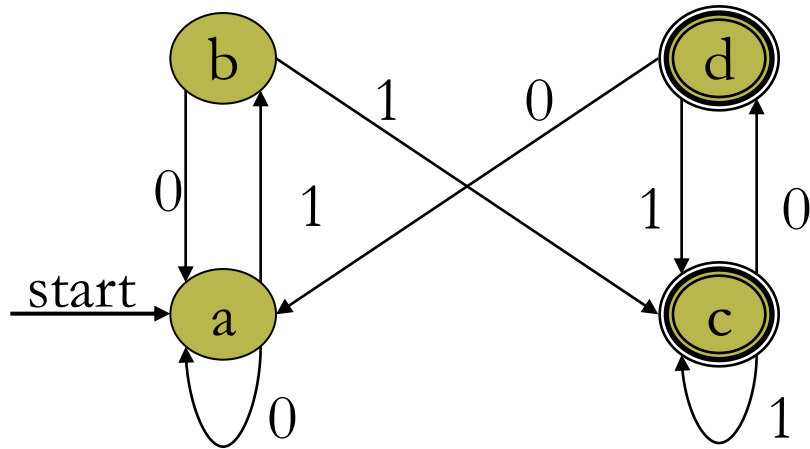
Po eliminacji wierzchołka u należy zastąpić etykietę R_{ij} , czyli etykietę krawędzi $s_i \rightarrow t_j$ przez etykietę $R_{ij} \mid S_i U^* T_j$.



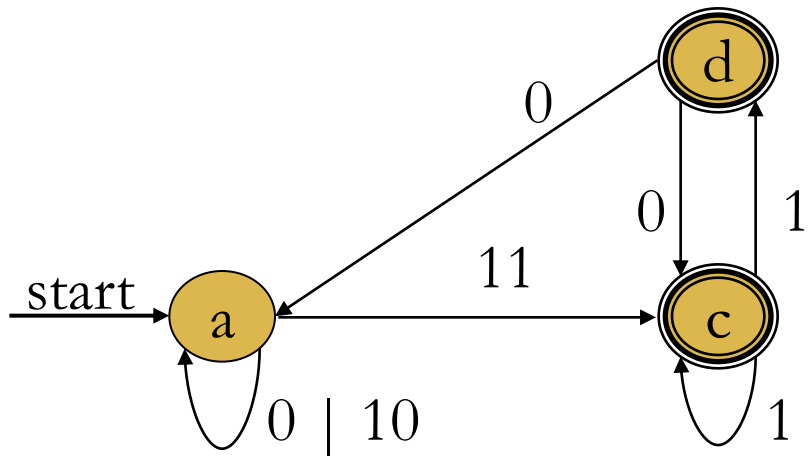
**Automat skończony
filtra odbijającego.**

**Stan b, jego poprzedniki oraz
następniki.**





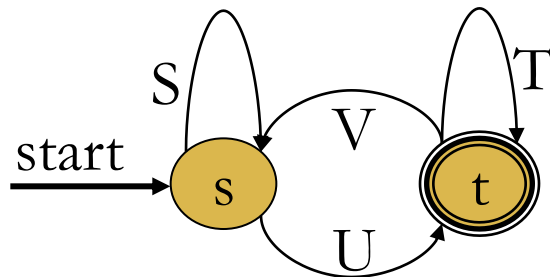
**Automat skończony
filtra odbijającego.**



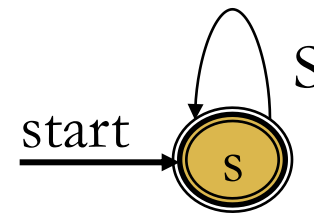
**Automat skończony
filtra odbijającego
po wyeliminowaniu
stanu b.**

Redukcja zupełna automatu

- W celu otrzymania wyrażenia regularnego określającego wszystkie ciągi znaków akceptowane przez automat **A** i żadne inne, należy rozpatrzyć po kolei każdy stan akceptujący **t** automatu **A**.
- Każdy ciąg znaków akceptowany przez automat **A** jest akceptowany dlatego, że etykietuje on ścieżkę wiodącą ze stanu początkowego **s** do pewnego stanu akceptującego **t**.



Automat zredukowany do dwóch stanów.
Wyrażenie regularne: $S^*U(T \mid VS^*U)^*$



Automat posiadający jedynie stan początkowy.
Wyrażenie regularne: S^*

Posumowanie

- Trzy sposoby określania języków dają te same możliwości wyrażania:
 1. Istnieje pewien automat deterministyczny, akceptujący wszystkie ciągi znaków języka **L** i żadne inne.
 2. Istnieje pewien, być może niedeterministyczny automat, akceptujący wszystkie ciągi znaków języka **L** i żadne inne.
 3. Język **L** jest językiem **L(R)** pewnego wyrażenia regularnego.
- Konstrukcja podzbiorów pokazuje, że **2.** implikuje **1.**
- Stwierdzenie **1.** implikuje **2.** gdyż automat deterministyczny jest szczególnym rodzajem automatu niedeterministycznego.
- Przechodzenie od wyrażeń regularnych do automatów oznacza że **3.** implikuje **2.**
- Przechodzenie od automatów do wyrażeń regularnych oznacza że **2.** implikuje **3.**
- Automaty deterministyczne mogą być używane jako podstawa programów, które rozpoznają wiele różnych rodzajów wzorców w ciągach znaków.
- Wyrażenia regularne są często przydatną konwencją tonacyjną opisywania wzorców. Istnieje formalizm praw algebraicznych dla wyrażeń regularnych.