Teoretyczne podstawy informatyki

Wykład 10:

Wzorce, automaty.

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:
  - oparty na teorii grafów, polegać będzie na wykorzystaniu ścieżek w grafie szczególnego rodzaju który nazwiemy automatem.
  - o charakterze algebraicznym, wykorzystujący notacje wyrażeń regularnych.
  - 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 postępem działania 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 sytuacje, 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;
      }
}
```

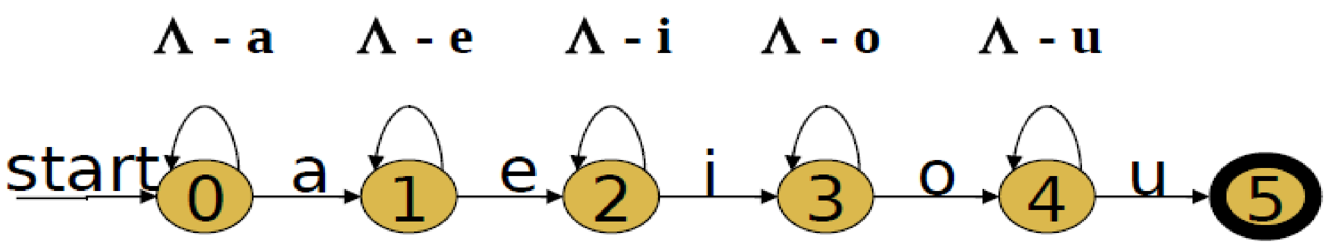
```
main()
{
    printf(‘’%d\n”, testWord(‘’abstemious”)
}
```

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.

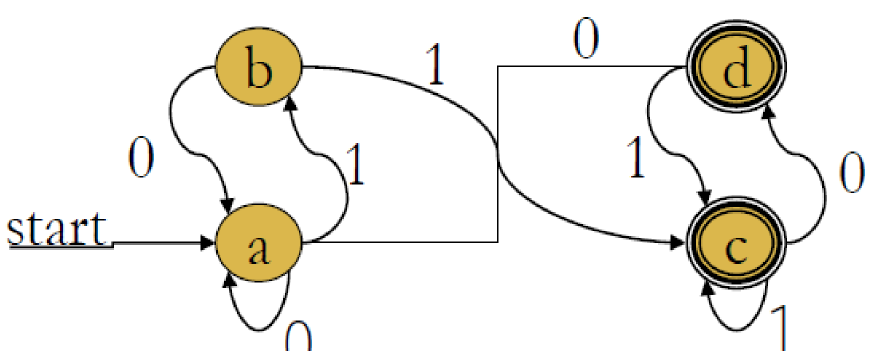
Automat rozpoznający ciągi liter

- **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
	a	0
0		
	a	0
1		
	b	0
0		
	a	0
1		
	b	0
1		
	c	1
0		
	d	1
1		
	c	1

Automaty a ich programamy

- **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.

Automaty a ich programy

- **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**.

Deterministyczny automat skończony

$$M = \{Q, \Sigma, \delta, q_0, F\}$$

Q – skończony zbiór stanów (sterowanie skończone);

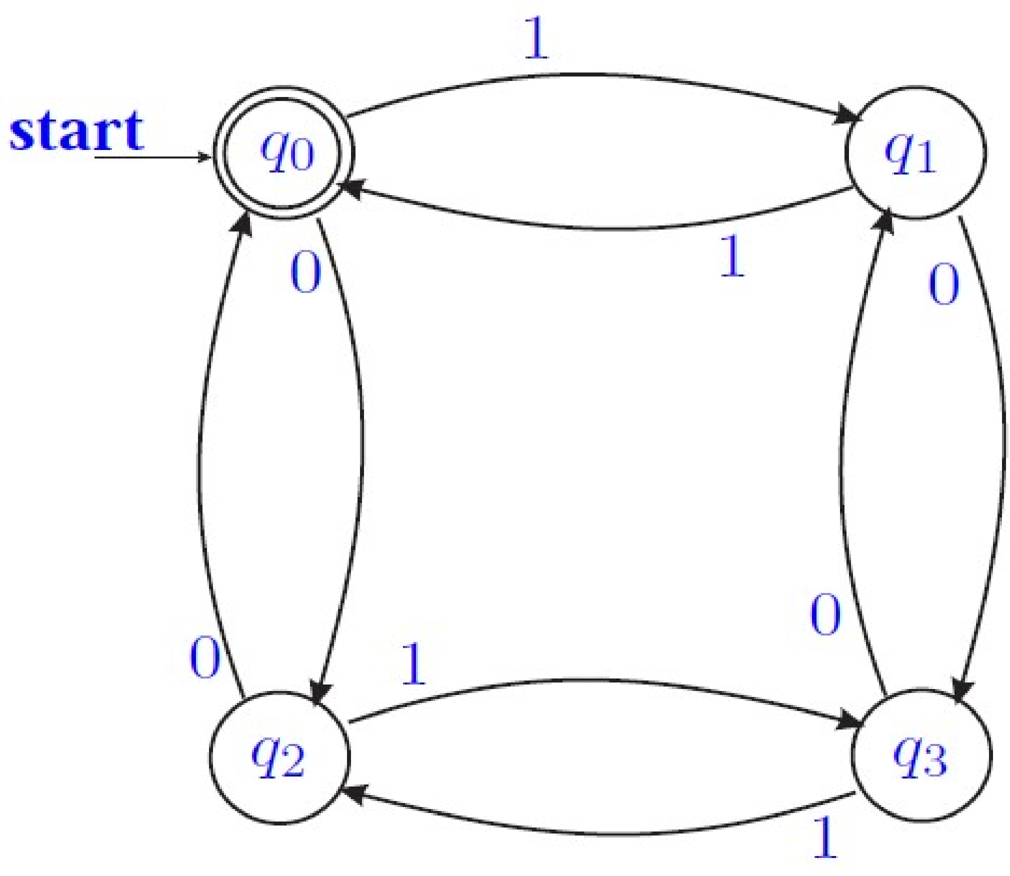
Σ – skończony alfabet wejściowy;

$\delta : Q \times \Sigma \rightarrow Q$ – funkcja przejścia;

$q_0 \in Q$ – stan początkowy;

$F \subseteq Q$ – zbiór stanów końcowych.

Deterministyczny automat skończony



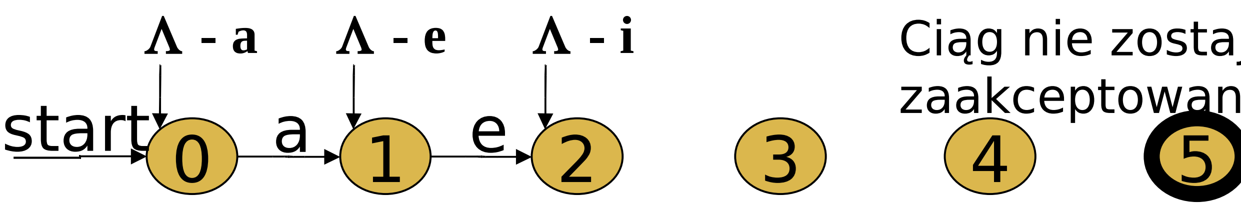
	0	1
q_0	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

Automaty deterministyczne i niedeterministyczne

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

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

Ciąg nie zostaje zaakceptowany



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 jakiekolwiek) 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**.

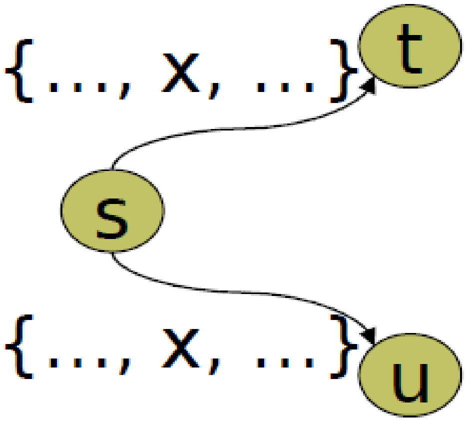
Funkcja implementująca automat deterministyczny

```
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 $\mathbf{a_1, a_2, \ldots, a_k}$ 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''

Niedeterministyczny automat

$$M = \{Q, \Sigma, \delta, q_0, F\}$$

Q – skończony zbiór stanów (sterowanie skończone);

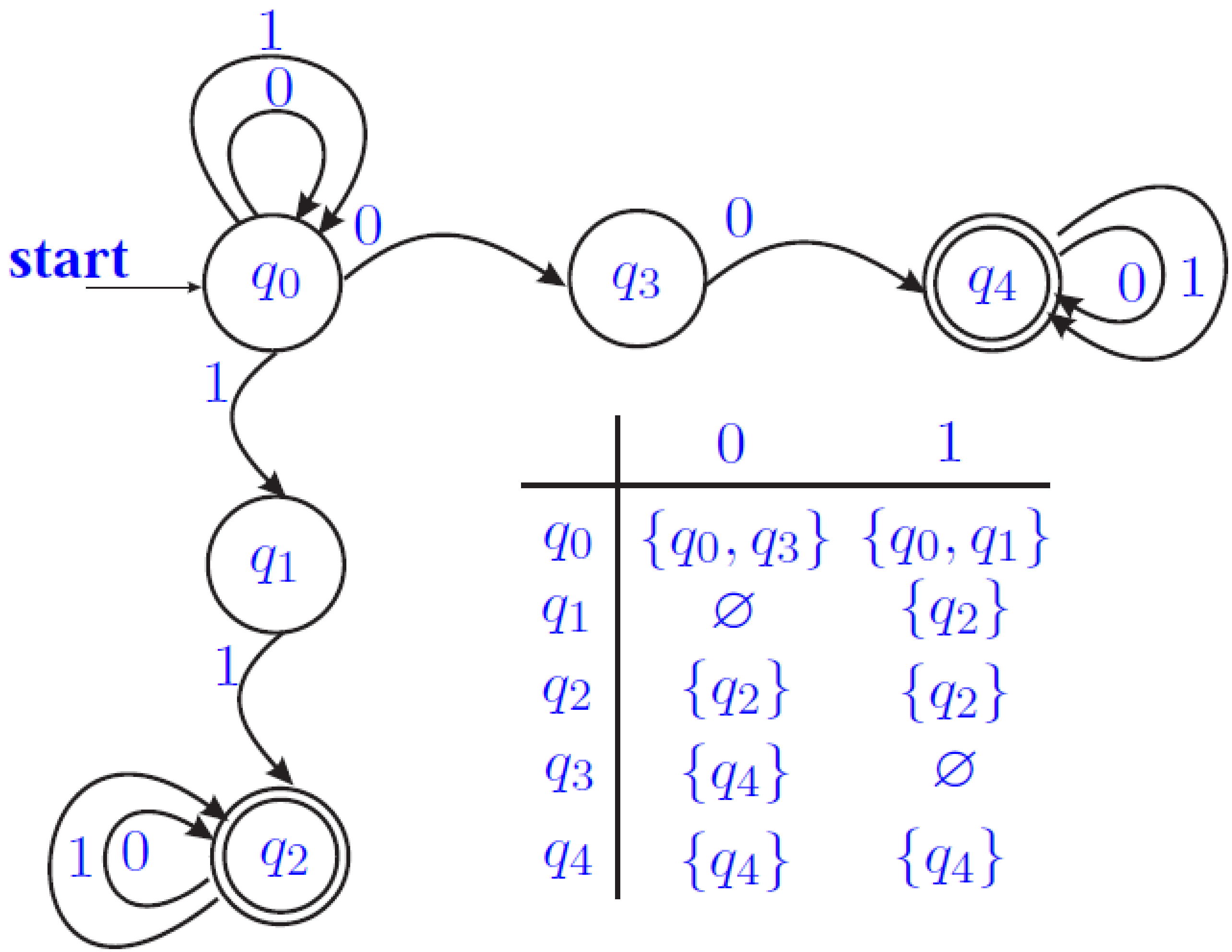
Σ – skończony alfabet wejściowy;

$\delta : Q \times \Sigma \to 2^Q$ – funkcja przejścia;

$q_0 \in Q$ – stan początkowy;

$F \subseteq Q$ – zbiór stanów końcowych.

Niedeterministyczny automat



	0	1
q_0	$\{q_0, q_3\}$	$\{q_0, q_1\}$
q_1	$\varnothing$	$\{q_2\}$
q_2	$\{q_2\}$	$\{q_2\}$
q_3	$\{q_4\}$	$\varnothing$
q_4	$\{q_4\}$	$\{q_4\}$

Zastępowanie automatów

Automat niedeterministyczny zastępujemy deterministycznym przez skonstruowanie **równoważnego** automatu deterministycznego. Technika ta nosi nazwę **konstrukcji podzbiorów** (ang. *subset construction*).

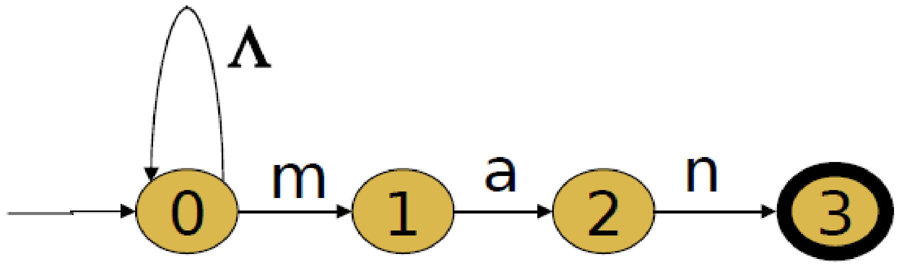
ównoważność automatów

- Mówimy, że automat **A** jest **równoważny** (ang. *equivalent*) automatowi **B**, jeżeli akceptują ten sam zbiór ciągów wejściowych.
Innymi słowy, jeżeli $\mathbf{a_1a_2...a_k}$ jest dowolnym ciągiem symboli, to spełnione są dwa następujące warunki:
- Jeżeli istnieje ścieżka zaetykietowana jako $\mathbf{a_1a_2...a_k}$ wiodąca od stanu początkowego automatu **A** do pewnego stanu akceptującego automatu **A**, to istnieje również ścieżka zaetykietowana $\mathbf{a_1a_2...a_k}$ wiodąca od stanu początkowego automatu **B** do stanu końcowego tego automatu.
- Jeżeli istnieje ścieżka zaetykietowana jako $\mathbf{a_1a_2...a_k}$ wiodąca od stanu początkowego automatu **B** do pewnego stanu akceptującego automatu **B**, to istnieje również ścieżka zaetykietowana $\mathbf{a_1a_2...a_k}$ wiodąca od stanu początkowego automatu **A** do stanu końcowego tego automatu.

Konstrukcja automatu niedeterministycznego

- **Podstawa:**
  - Jeżeli stanem początkowym automatu niedeterministycznego **N** jest $\mathbf{s_0}$, to stanem początkowym automatu deterministycznego **D** jest $\mathbf{\{s_0\}}$, czyli zbiór zawierający tylko element $\mathbf{s_0}$.
- **Indukcja:**
  - Załóżmy, że ustalono, iż **S**, zbiór stanów automatu **N**, jest stanem automatu **D**.
  - Rozpatrujemy po kolei każdy możliwy znak wejściowy **x**. Dla danego **x** określamy, że **T** jest zbiorem stanów **t** automatu **N**, takich, że dla pewnego stanu **s** należącego do zbioru **S** istnieje przejście z **s** do **t** etykietą zawierającą znak **x**.
  - Wówczas zbiór **T** jest stanem automatu **D** i istnieje przejście z **S** do **T** względem znaku wejściowego **x**.

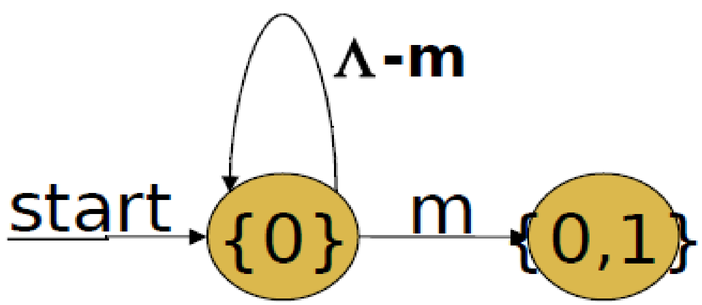
Konstrukcja automatu deterministycznego D



Niedeterministyczny automat rozpoznający ciąg znaków kończący się sekwencja ‘’ man ‘’.

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

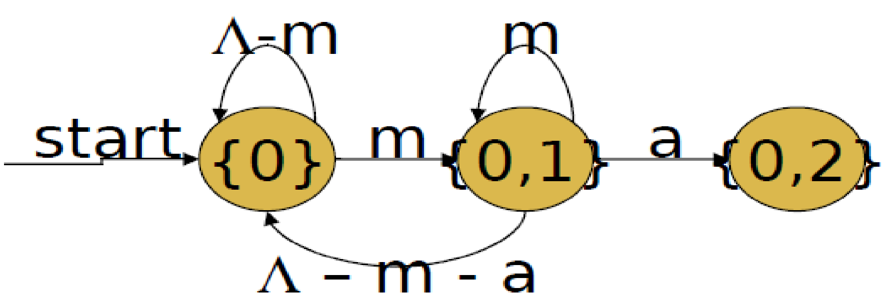
Stan {0} i jego przejścia



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 **(L –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}**.

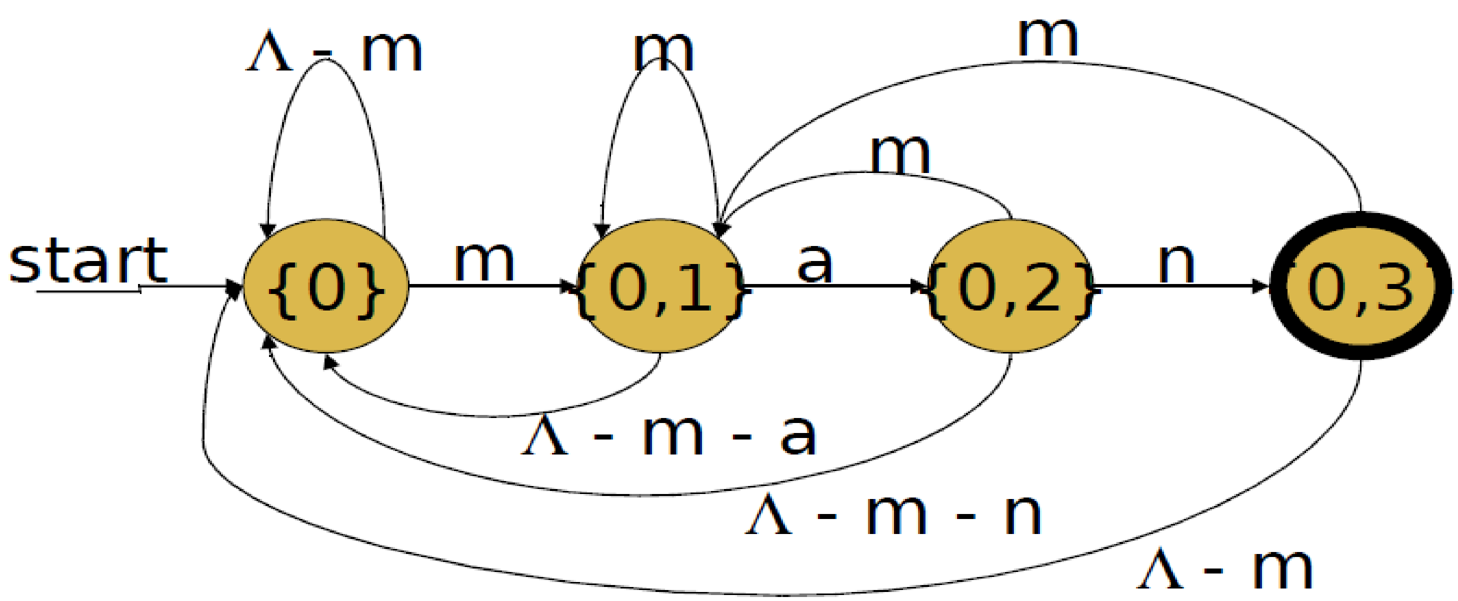


Stan {0} i {0,1} oraz ich przejścia

Konstrukcja automatu deterministycznego D

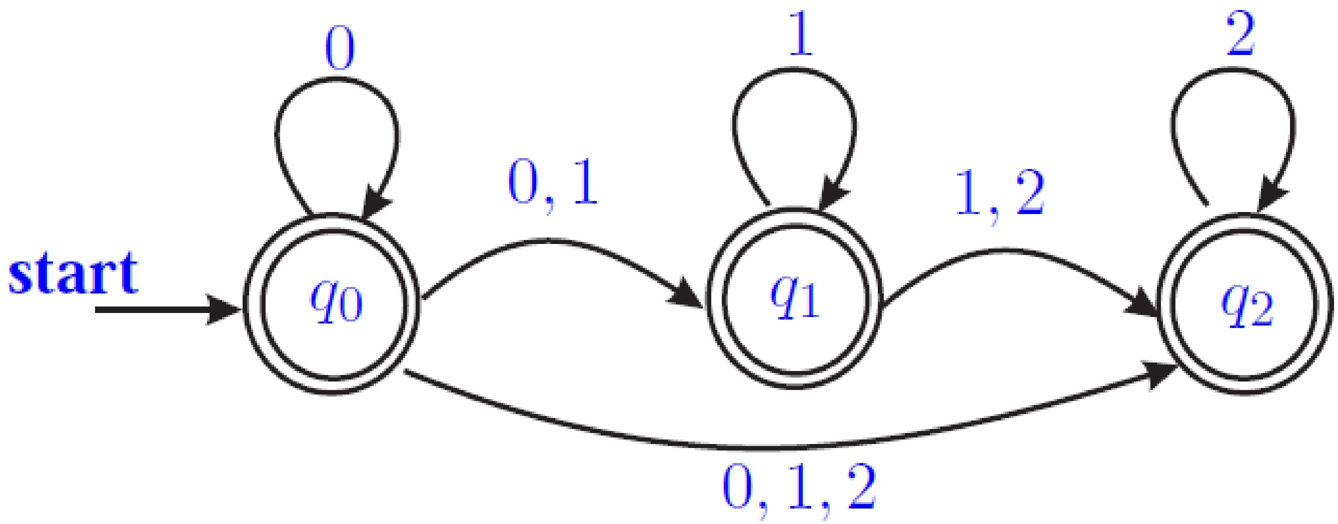
- Teraz należy skonstruować przejścia ze stanu **{0,2}**. W przypadku wszystkich znaków wejściowych oprócz **m** i **n**, ze stanu **0** można przejść jedynie do stanu **0**, zaś ze stanu **2** nie można nigdzie przejść. Zatem istnieje przejście ze stanu **{0,2}** do stanu **{0}** zaetykietowane wszystkimi 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, zś 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**, stad ograniczają się jedynie do przejść stanu **{0}**. Ponieważ przejścia ze stanu **{0,3}** nie prowadza do żadnego stanu automatu **D**, którego jeszcze nie sprawdziliśmy, konstrukcje automatu **D** można uznać za skończoną.

Automat deterministyczny D



Konstrukcje automatu można uznać za skończoną. Przejścia ze stanu **{0,3}** nie prowadza 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 sekwencja **man**.

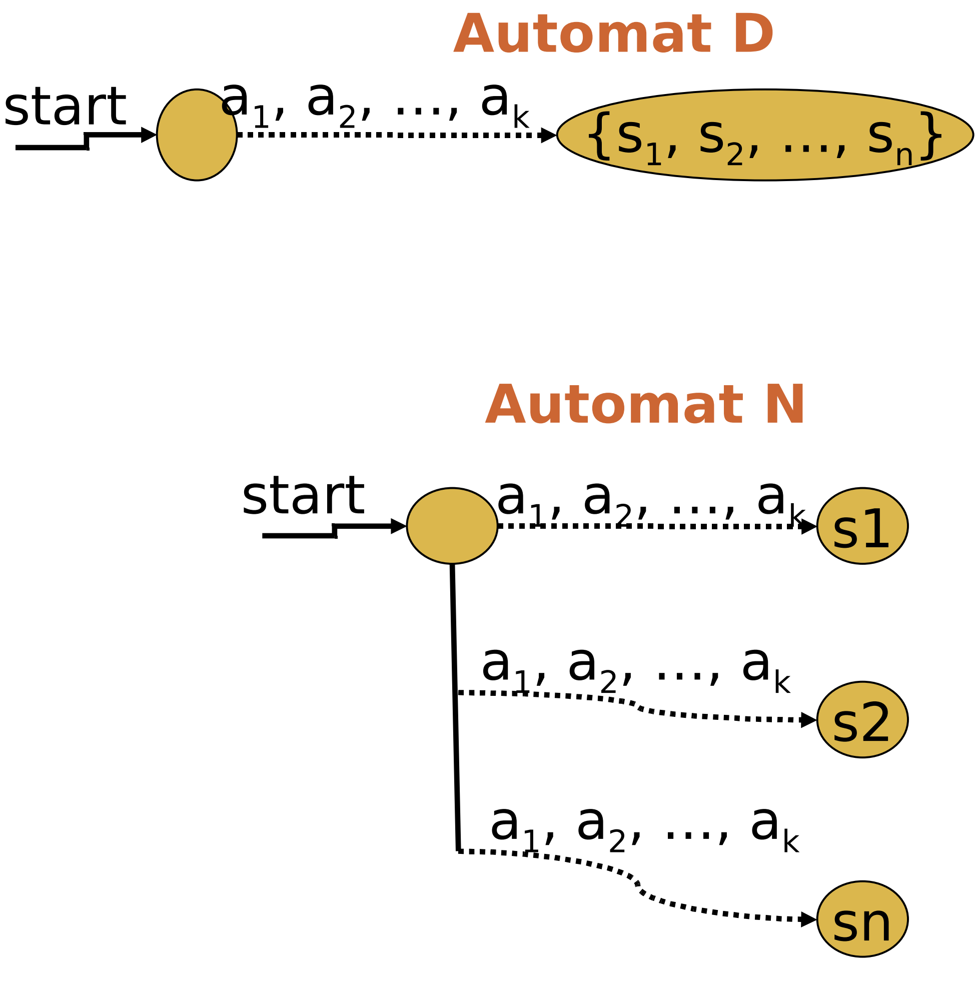
Niedeterministyczny automat



	0	1	2
q_0	$\{q_0, q_1, q_2\}$	$\{q_1, q_2\}$	$\{q_2\}$
q_1	$\varnothing$	$\{q_1, q_2\}$	$\{q_2\}$
q_2	$\varnothing$	$\varnothing$	$\{q_2\}$

Poprawnośc 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.



Poprawność konstrukcji podzbiorów

- Formalne twierdzenie, które należy udowodnić przez indukcje względem **k**, mówi że stan $\{s_1, s_2, \ldots, s_n\}$ osiągany przez automat **D** w wyniku przejścia ścieżką zaetykietowana symbolami $a_1a_2\ldots a_k$ od stanu początkowego automatu **D** jest dokładnie zbiorem stanów automatu **N**, które są przez ten automat osiągane w wyniku przejścia od stanu początkowego pewną ścieżką zaetykietowana symbolami $a_1a_2\ldots a_k$.
- **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, ze twierdzenie jest spełnione dla **k** i rozważmy ciąg wejściowy $a_1,a_2,\ldots,a_k,a_{k+1}$. Wówczas ścieżka wiodąca od stanu początkowego automatu **D** do stanu **T**, zaetykietowana stanami $a_1a_2\ldots a_ka_{k+1}$, ma postać:

Poprawność 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 $\mathbf{a_1a_2...a_k}$ i należy udowodnić, że **T** jest dokładnie zbiorem stanów, które automat **N** osiąga od swojego stanu początkowego wzdłuż ścieżki zaetykietowanej symbolami $\mathbf{a_1,a_2,...,a_k,a_{k+1}}$. Dowód kroku indukcyjnego składa się z dwóch części.
  - 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 $\mathbf{a_1,a_2,...,a_k,a_{k+1}}$ wiodącej ze stanu początkowego automatu **N**.
  - Należy wykazać że zbiór **T** zawiera wystarczająca ilość elementów. To znaczy, jeżeli **t** jest stanem automatu **N** osiąganym ze stanu początkowego ścieżką zaetykietowana symbolami $\mathbf{a_1,a_2,...,a_k,a_{k+1}}$, 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.

Minimalizacja automatów

- **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 miedzy automatami deterministycznymi a programami, które wykorzystują pojecie 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.