Logika cyfrowa

Wykład 9: automaty skończone

Marek Materzok

29 kwietnia 2024

Języki regularne

Alfabet i słowo

Definicja (alfabet)
Zbiór Σ symboli używanych do budowy słów. Na przykład:

- $\Sigma = \{0, 1\} = \mathbb{B}$ – alfabet binarny,
- $\Sigma = \{a, b, \ldots, z\}$ – alfabet małych znaków łacińskich.

Definicja (słowo)
Ciąg znaków nad wybranym alfabetem Σ. Na przykład:

- 1, 101, 1111 – słowa nad alfabetem binarnym,
- *a*, *baba*, *qwerty* – słowa nad alfabetem małych znaków łacińskich.

Słowa – podstawowe definicje

Definicja (długość słowa)
$|w|$ oznacza liczbę liter słowa w. Przykładowo, $|abc| = 3$.

Definicja (słowo puste)
ϵ oznacza słowo puste, tzn. jedyne słowo o długości 0: $|\epsilon| = 0$.

Definicja (konkatenacja słów)
vw oznacza słowo zbudowane z wszystkich liter słowa v, a następnie wszystkich liter słowa w, w kolejności. Przykładowo $(abc)(def) = abcdef$.

Konkatenacja słów – własności

Elementem neutralnym konkatenacji jest ϵ:

$$\epsilon w = w\epsilon = w$$

Konkatenacja jest łączna:

$$(uv)w = u(vw) = uvw$$

Konkatenacja **nie jest** przemienna.

Potęgowanie słów

Definicja (potęgowanie słów)
Słowo w^n – n-krotne powtórzenie słowa w. Przykłady:

- $a^0 = \epsilon$
- $(ba)^2 = baba$
- $(rowe)^2 = rowerowe$
- $(esem)^2 = esemesem$

Domknięcie Kleene'go

Definicja (domknięcie (gwiazdka) Kleene'go)
Zbiór Σ^* – wszystkie słowa, dowolnej długości, nad alfabetem Σ:

$$\Sigma^* = \{a_1 a_2 \ldots a_n \mid n \in \mathbb{N} \land \forall_{k \in \{1,\ldots,n\}} a_k \in \Sigma\}$$

Formalnie definiowane jako najmniejszy zbiór spełniający warunki:

$$\epsilon \in \Sigma^*$$
$$a \in \Sigma \land w \in \Sigma^* \to aw \in \Sigma^*$$

Przykłady:

- $\{a\}^* = \{\epsilon, a, aa, aaa, \ldots\}$
- $\{a, b\}^* = \{\epsilon, a, b, aa, ab, ba, bb, \ldots\}$

Język

Definicja (język)
Językiem nad alfabetem Σ nazywamy dowolny zbiór $L \subseteq \Sigma^*$.

Przykłady:

- $\{w0 \mid w \in \mathbb{B}^*\}$ – zbiór parzystych liczb binarnych,
- $\{w \in \mathbb{B}^* \mid |w| = n\}$ – zbiór ciągów binarnych długości n,
- $\{0^n1^n \mid n \in \mathbb{N}\}$ – zbiór ciągów binarnych zbudowanych z konkatenacji bloku zer i bloku jedynek tej samej długości.

Języki – podstawowe definicje

Definicja (język pusty)
$\varnothing$ – język nie zawierający żadnego słowa.

Nie mylić z $\{\epsilon\}$ – językiem zawierającym jedno słowo, słowo puste.

Definicja (potęgowanie języków)
L^n – język słów zbudowanych z konkatenacji n słów z L:

$$L^n = \{w_1 w_2 \dots w_n \mid \forall_{k \in \{1,\dots,n\}} w_k \in L\}$$

Przykłady:

- $\mathbb{B}^0 = \{\epsilon\}$
- $\mathbb{B}^3 = \{000, 001, 010, 011, 100, 101, 110, 111\}$
- $\{ba, za\}^2 = \{baba, baza, zaba, zaza\}$

Języki – podstawowe definicje

Definicja (singleton)
Język zawierający tylko jedno słowo, na przykład $\{10\}$.

Definicja (konkatenacja języków)
Język $L_1 L_2$ – zbiór konkatenacji wszystkich słów z L_1 ze wszystkimi słowami z L_2:

$$L_1 L_2 = \{w_1 w_2 \mid w_1 \in L_1 \wedge w_2 \in L_2\}$$

Na przykład:

- $\varnothing L = L \varnothing = \varnothing$
- $\{\epsilon\} L = L \{\epsilon\} = L$
- $\{ko, pło\}\{ń, za\} = \{koń, koza, płoń, płoza\}$

Języki – podstawowe definicje

Definicja (suma języków)
Język $L_1 \cup L_2$ – zbiór słów, które należą do L_1 lub L_2.

Przykład: $\{0, 00\} \cup \{1, 11\} = \{0, 00, 1, 11\}$

Definicja (gwiazdka Kleene'go (dla języków))
Język L^* – zbiór konkatenacji dowolnej liczby słów z L:

$$L^* = L^0 \cup L^1 \cup L^2 \cup \dots$$

Definicja (plus Kleene'go (dla języków))
Język L^+ – zbiór konkatenacji dowolnej **dodatniej** liczby słów z L:

$$L^+ = L^1 \cup L^2 \cup \dots = LL^*$$

Definicja (języki regularne)
Języki, które można zbudować wyłącznie tymi operacjami:

- $\varnothing$ (język pusty),
- $\{\epsilon\}$ (singleton zawierający słowo puste),
- $\{a\}$ gdzie $a \in \Sigma$ (singleton zawierający symbol),
- $L_1 \cup L_2$ (suma języków regularnych),
- $L_1 L_2$ (konkatenacja języków regularnych),
- L^* (domknięcie Kleene'go języka regularnego),

nazywamy językami regularnymi.

Przykładowe języki regularne

- Wszystkie języki skończone (zawierające skończoną liczbę słów) są regularne.
- Język ciągów zer parzystej długości:

$$(\{0\}\{0\})^* = \{\epsilon, 00, 0000, \dots\}$$

- Język ciągów zer i jedynek zawierający parzystą liczbę jedynek i dowolną liczbę zer:

$$\{0\}^*(\{1\}\{0\}^*\{1\}\{0\}^*)^*$$

Wyrażenia regularne

Definicja (wyrażenia regularne)
Wyrażenia opisujące języki regularne:

- $L(\varnothing) = \varnothing$,
- $L(\epsilon) = \{\epsilon\}$,
- $L(a) = \{a\}$,
- $L(e_1 + e_2) = L(e_1) \cup L(e_2)$,
- $L(e_1 e_2) = L(e_1) L(e_2)$,
- $L(e^*) = L(e)^*$.

Przykłady wyrażeń regularnych

- Wyrażenie opisujące język ciągów zer parzystej długości:

$$(00)^*$$

- Wyrażenie opisujące język zawierający parzystą liczbę jedynek i dowolną liczbę zer:

$$0^*(10^*10^*)^*$$

Deterministyczne automaty skończone

DFA – deterministyczne automaty skończone

Definicja (deterministyczny automat skończony)
Krotka $\mathcal{M} = \langle Q, \Sigma, \delta, q_0, F \rangle$, gdzie:

- Q – skończony niepusty zbiór *stanów*,
- Σ – skończony niepusty zbiór *symboli* (*alfabet*),
- $\delta : Q \times \Sigma \to Q$ – *funkcja przejścia*,
- $q_0 \in Q$ – *stan początkowy*,
- $F \subseteq Q$ – *stany akceptujące*.

DFA – przykład

Automat rozpoznający ciągi zer parzystej długości:

$$\mathcal{M} = \langle Q, \Sigma, \delta, q_0, F \rangle = \langle \{q_p, q_n\}, \{0\}, \delta, q_p, \{q_p\} \rangle$$
$$\delta = \{(q_p, 0, q_n), (q_n, 0, q_p)\}$$

DFA – przykład

Automat rozpoznający ciągi zer parzystej długości:

$$\mathcal{M} = \langle Q, \Sigma, \delta, q_0, F \rangle = \langle \{q_p, q_n\}, \{0\}, \delta, q_p, \{q_p\} \rangle$$
$$\delta = \{(q_p, 0, q_n), (q_n, 0, q_p)\}$$

Tabelka:

q	a	q_o
q_p	0	q_n
q_n	0	q_p

DFA – przykład

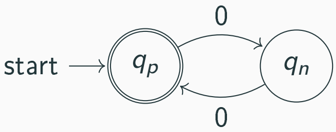
Automat rozpoznający ciągi zer parzystej długości:

$$\mathcal{M} = \langle Q, \Sigma, \delta, q_0, F \rangle = \langle \{q_p, q_n\}, \{0\}, \delta, q_p, \{q_p\} \rangle$$
$$\delta = \{(q_p, 0, q_n), (q_n, 0, q_p)\}$$

Tabelka:

q	a	q_o
q_p	0	q_n
q_n	0	q_p

Diagram:



DFA – definicje

Definicja (domknięcie funkcji przejścia)
Funkcja $\delta^* : Q \times \Sigma^* \to Q$.

Jeśli $q \in Q$ i $w \in \Sigma^*$, to $\delta^*(q, w)$ jest stanem osiąganym ze stanu q po przeczytaniu (dowolnie długiego) w.

$$\begin{aligned} \delta^*(q, \epsilon) &= q \\ \delta^*(q, aw) &= \delta^*(\delta(q, a), w) \end{aligned}$$

Definicja (język automatu)
$L(\mathcal{M}) \subseteq \Sigma^*$ to język słów rozpoznawanych przez automat.

$$L(\langle Q, \Sigma, \delta, q_0, F\rangle) = \{w \mid \delta^*(q_0, w) \in F\}$$

Języki rozpoznawane przez DFA

Twierdzenie
DFA rozpoznają tylko i wyłącznie języki regularne.

Języki rozpoznawane przez DFA

Twierdzenie
DFA rozpoznają tylko i wyłącznie języki regularne.

Dokładniej: dla każdego języka $L \subseteq \Sigma^*$ *istnieje DFA* $\mathcal{M}$ *taki, że* $L(\mathcal{M}) = L$ *wtedy i tylko wtedy, gdy istnieje wyrażenie regularne* e *takie, że* $L(e) = L$.

Języki rozpoznawane przez DFA

Twierdzenie
DFA rozpoznają tylko i wyłącznie języki regularne.

Dokładniej: dla każdego języka $L \subseteq \Sigma^*$ *istnieje DFA* $\mathcal{M}$ *taki, że* $L(\mathcal{M}) = L$ *wtedy i tylko wtedy, gdy istnieje wyrażenie regularne* e *takie, że* $L(e) = L$.

Dowód.
Na przedmiocie „Języki Formalne i Złożoność Obliczeniowa”. $\square$

DFA – drugi przykład: parzysta liczba jedynek

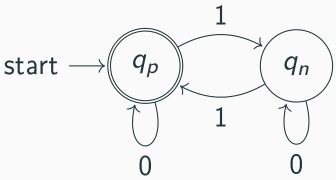
$\mathcal{M} = \langle \{q_p, q_n\}, \{0, 1\}, \delta, q_p, \{q_p\} \rangle$

$\delta = \{(q_p, 0, q_p), (q_p, 1, q_n), (q_n, 0, q_n), (q_n, 1, q_p)\}$

Tabelka:

q	a	q_o
q_p	0	q_p
q_p	1	q_n
q_n	0	q_n
q_n	1	q_p

Diagram:



Parzysta liczba jedynek – przydział stanów

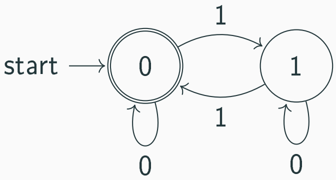
$\mathcal{M} = \langle \{0, 1\}, \{0, 1\}, \delta, 0, \{0\} \rangle$

$\delta = \{(0, 0, 0), (0, 1, 1), (1, 0, 1), (1, 1, 0)\}$

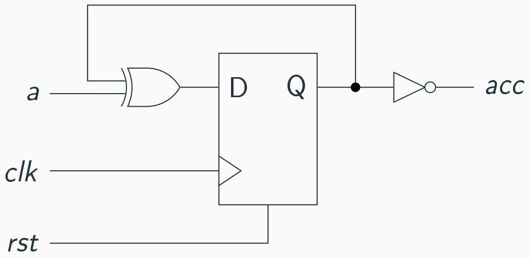
Tabelka:

q	a	q_o
0	0	0
0	1	1
1	0	1
1	1	0

Diagram:



Parzysta liczba jedynek – schemat



Parzysta liczba jedynek – SystemVerilog

```
module parity(input clk, rst, a, output acc);
    logic q;                // stan automatu
    always_ff @(posedge clk or posedge rst)
        if (rst) q <= 0; // stan początkowy
        else q <= a ^ q; // funkcja przejścia
    assign acc = !q;        // stan akceptujący
endmodule
```

Automaty Moore'a i Mealy'ego

Transducery

Deterministyczne automaty skończone produkują tylko jednobitowe wyjście: wejście zostało zaakceptowane lub nie.

Chcemy zamodelować w formie automatu układy z bardziej złożonym wyjściem.

Służą do tego automaty nazywane *transducerami skończonymi*. A konkretnie – automaty *Moore'a* i *Mealy'ego*.

Automaty Moore'a

Definicja (automat Moore'a)
Krotka $\mathcal{M} = \langle Q, \Sigma, \Omega, \delta, \chi, q_0 \rangle$, gdzie:

- Q – skończony niepusty zbiór *stanów*,
- Σ – skończony niepusty *alfabet* wejściowy,
- Ω – skończony niepusty *alfabet* wyjściowy,
- $\delta : Q \times \Sigma \to Q$ – *funkcja przejścia*,
- $\chi : Q \to \Omega$ – *funkcja wyjścia*,
- $q_0 \in Q$ – *stan początkowy*.

Automaty Moore'a

Definicja (automat Moore'a)
Krotka $\mathcal{M} = \langle Q, \Sigma, \Omega, \delta, \chi, q_0 \rangle$, gdzie:

- Q – skończony niepusty zbiór *stanów*,
- Σ – skończony niepusty *alfabet* wejściowy,
- Ω – skończony niepusty *alfabet* wyjściowy,
- $\delta : Q \times \Sigma \to Q$ – *funkcja przejścia*,
- $\chi : Q \to \Omega$ – *funkcja wyjścia*,
- $q_0 \in Q$ – *stan początkowy*.

Deterministyczne automaty skończone są też automatami Moore'a; $\Omega = \mathbb{B}$, $\chi(q) = q \in F$.

Automaty Moore'a – definicje

Niech $\mathcal{M} = \langle Q, \Sigma, \Omega, \delta, \chi, q_0 \rangle$.

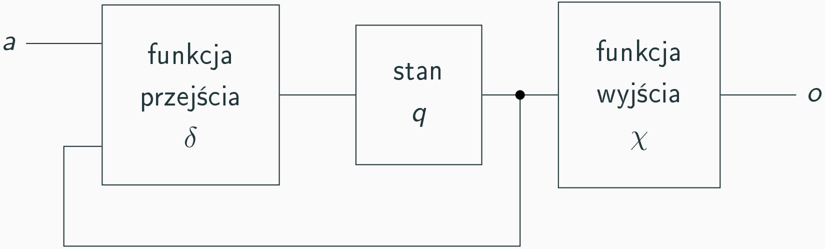
- $O(\mathcal{M}) : Q \times \Sigma^* \to \Omega^*$ – wyjście dla zadanego wejścia i stanu

$$O(\mathcal{M})(q, \epsilon) = \epsilon$$
$$O(\mathcal{M})(q, aw) = \chi(\delta(q, a))O(\mathcal{M})(\delta(q, a), w)$$

- $O(\mathcal{M}) : \Sigma^* \to \Omega^*$ – wyjście dla zadanego wejścia, zaczynając od stanu początkowego

$$O(\mathcal{M})(w) = O(\mathcal{M})(q_0, w)$$

Automaty Moore'a – idea



Automat Moore'a – przykład: automat liczący od 0 do 3

$$\mathcal{M} = \langle Q, \Sigma, \Omega, \delta, \chi, q_0 \rangle = \langle \{0, 1, 2, 3\}, \mathbb{B}, \{0, 1, 2, 3\}, \delta, \mathrm{id}, 0 \rangle$$

$$\delta(q, a) = \begin{cases} q & \text{gdy } a = 0 \\ (q + 1) \bmod 4 & \text{gdy } a = 1 \end{cases}$$

Automat Moore'a – przykład: automat liczący od 0 do 3

$$\mathcal{M} = \langle Q, \Sigma, \Omega, \delta, \chi, q_0 \rangle = \langle \{0,1,2,3\}, \mathbb{B}, \{0,1,2,3\}, \delta, \mathrm{id}, 0 \rangle$$

$$\delta(q,a) = \begin{cases} q & \text{gdy } a = 0 \\ (q+1) \bmod 4 & \text{gdy } a = 1 \end{cases}$$

Tabelka δ:

q	a	q_o
q	0	q
0	1	1
1	1	2
2	1	3
3	1	0

Automat Moore'a – przykład: automat liczący od 0 do 3

$$\mathcal{M} = \langle Q, \Sigma, \Omega, \delta, \chi, q_0 \rangle = \langle \{0, 1, 2, 3\}, \mathbb{B}, \{0, 1, 2, 3\}, \delta, \mathrm{id}, 0 \rangle$$

$$\delta(q, a) = \begin{cases} q & \text{gdy } a = 0 \\ (q+1) \bmod 4 & \text{gdy } a = 1 \end{cases}$$

Tabelka δ:

q	a	q_o
q	0	q
0	1	1
1	1	2
2	1	3
3	1	0

Tabelka χ:

q	o
0	0
1	1
2	2
3	3

Automat Moore'a – przykład: automat liczący od 0 do 3

$$\mathcal{M} = \langle Q, \Sigma, \Omega, \delta, \chi, q_0 \rangle = \langle \{0, 1, 2, 3\}, \mathbb{B}, \{0, 1, 2, 3\}, \delta, \mathrm{id}, 0 \rangle$$

$$\delta(q, a) = \begin{cases} q & \text{gdy } a = 0 \\ (q + 1) \bmod 4 & \text{gdy } a = 1 \end{cases}$$

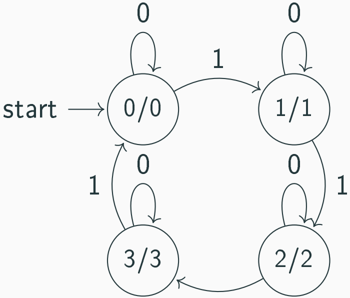
Tabelka δ:

q	a	q_o
q	0	q
0	1	1
1	1	2
2	1	3
3	1	0

Tabelka χ:

q	o
0	0
1	1
2	2
3	3

Diagram:



Licznik – przydział stanów

$$\mathcal{M} = \langle Q, \Sigma, \Omega, \delta, \chi, q_0 \rangle = \langle \mathbb{B}^2, \mathbb{B}, \mathbb{B}^2, \delta, \mathrm{id}, 00 \rangle$$

$$\delta(q, a) = \begin{cases} q & \text{gdy } a = 0 \\ (q+1) \bmod 4 & \text{gdy } a = 1 \end{cases}$$

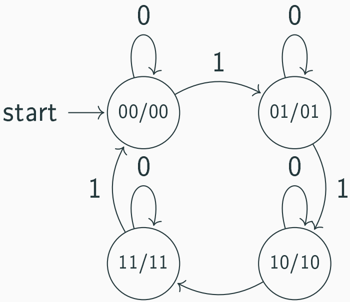
Tabelka δ:

q	a	q_o
q	0	q
00	1	01
01	1	10
10	1	11
11	1	00

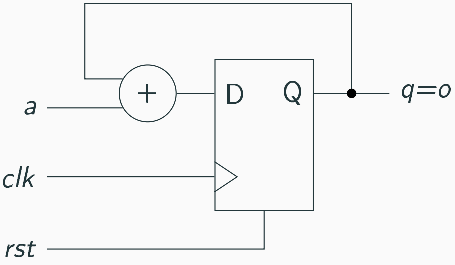
Tabelka χ:

q	o
00	00
01	01
10	10
11	11

Diagram:



Licznik – schemat



Licznik – SystemVerilog

```
module counter(input clk, rst, a, output [1:0] o);
    logic [1:0] q;            // stan automatu
    always_ff @(posedge clk or posedge rst)
        if (rst) q <= 2'b0; // stan początkowy
        else q <= a + q;    // funkcja przejścia
    assign o = q;            // funkcja wyjścia
endmodule
```

Licznik, podejście 2 – mapy Karnaugh

Tabelka δ:

q	a	q_o
q	0	q
00	1	01
01	1	10
10	1	11
11	1	00

$q_{o0} =$

$q_{o1} =$

$q_1 q_0$

a \ q_1q_0	00	01	11	10
0	0	1	1	0
1	1	0	0	1

$q_1 q_0$

a \ q_1q_0	00	01	11	10
0	0	0	1	1
1	0	1	0	1

Licznik, podejście 2 – mapy Karnaugh

Tabelka δ:

q	a	q_o
q	0	q
00	1	01
01	1	10
10	1	11
11	1	00

$q_{o0} =$

$q_{o1} =$

a \ $q_1 q_0$	00	01	11	10
0	0	1	1	0
1	1	0	0	1

a \ $q_1 q_0$	00	01	11	10
0	0	0	1	1
1	0	1	0	1

Licznik, podejście 2 – mapy Karnaugh

Tabelka δ:

q	a	q_o
q	0	q
00	1	01
01	1	10
10	1	11
11	1	00

$$q_{o0} = a\bar{q}_0 + \bar{a}q_0$$

$$q_{o1} = a\bar{q}_1 q_0 + \bar{a}q_1 + q_1\bar{q}_0$$

a \ $q_1 q_0$	00	01	11	10
0	0	1	1	0
1	1	0	0	1

a \ $q_1 q_0$	00	01	11	10
0	0	0	1	1
1	0	1	0	1

Licznik, podejście 2 – mapy Karnaugh

Tabelka δ:

q	a	q_o
q	0	q
00	1	01
01	1	10
10	1	11
11	1	00

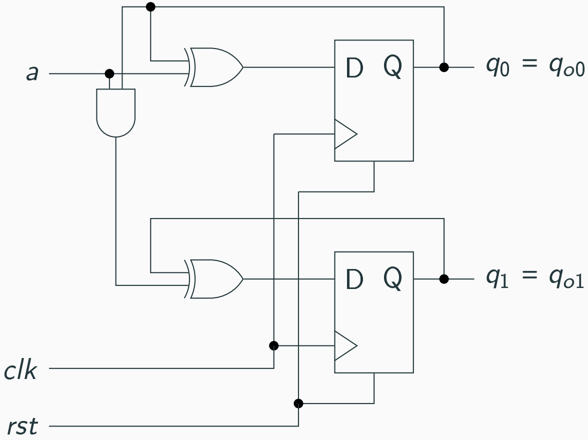
$$q_{o0} = a\bar{q}_0 + \bar{a}q_0 = a \oplus q_0$$

$$\begin{aligned} q_{o1} &= a\bar{q}_1 q_0 + \bar{a}q_1 + q_1\bar{q}_0 \\ &= \bar{q}_1(aq_0) + q_1\overline{(aq_0)} \\ &= aq_0 \oplus q_1 \end{aligned}$$

a \ $q_1 q_0$	00	01	11	10
0	0	1	1	0
1	1	0	0	1

a \ $q_1 q_0$	00	01	11	10
0	0	0	1	1
1	0	1	0	1

Licznik, podejście 2 – schemat



Wygląda znajomo?

Licznik, podejście 2 – SystemVerilog

```
module counter(input clk, rst, a, output [1:0] o);
    logic q1, q0;               // stan automatu
    always_ff @(posedge clk or posedge rst)
        if (rst)                // stan początkowy
            {q1, q0} <= 2'b0;
        else                    // funkcja przejścia
            {q1, q0} <= {a & q0 ^ q1, a ^ q0};
    assign o = {q1, q0};        // funkcja wyjścia
endmodule
```

Licznik – inny przydział stanów

$$\mathcal{M} = \langle Q, \Sigma, \Omega, \delta, \chi, q_0 \rangle = \langle \mathbb{B}^4, \mathbb{B}, \mathbb{B}^2, \delta, \chi, 0001 \rangle$$

$$\delta(q, a) = \begin{cases} q & \text{gdy } a = 0 \\ q \text{ rol } 1 & \text{gdy } a = 1 \end{cases}$$

$$\chi(q) = \log_2 q \text{ gdy } q \text{ potęgą dwójki}$$

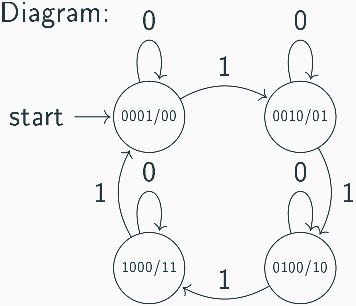
Tabelka δ:

q	a	q_o
q	0	q
0001	1	0010
0010	1	0100
0100	1	1000
1000	1	0001
...	1	...

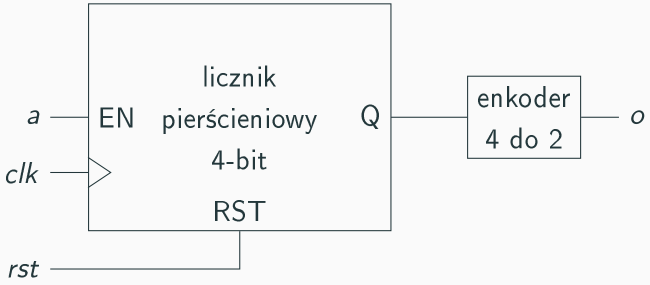
Tabelka χ:

q	o
0001	00
0010	01
0100	10
1000	11
xxxx	xx

Diagram:



Licznik, podejście 3 – schemat



Licznik, podejście 3 – SystemVerilog

```
module counter(input clk, rst, a, output logic [1:0] o);
    logic [3:0] q;                    // stan automatu
    always_ff @(posedge clk or posedge rst)
        if (rst)                      // stan początkowy
            q <= 4'b0001;
        else                          // funkcja przejścia
            q <= a ? {q[2:0], q[3]} : q;
    always_comb unique casez(q) // funkcja wyjścia
        4'b???1: o = 2'd0;
        4'b??1?: o = 2'd1;
        4'b?1??: o = 2'd2;
        4'b1???: o = 2'd3;
        default: o = 2'dx;
    endcase
```

Automat Moore'a – przykład: wykrywanie zmian

$$\mathcal{M} = \langle \{s_{00}, s_{01}, s_{10}, s_{11}\}, \{0,1\}, \{o_-, o_\uparrow, o_\downarrow\}, \delta, \chi, s_{00} \rangle$$

$$\delta = \{(s_{ca}, b, s_{ab}) \mid a \in \{0,1\}, b \in \{0,1\}, c \in \{0,1\}\}$$

$$\chi = \{(s_{00}, o_-), (s_{11}, o_-), (s_{01}, o_\uparrow), (s_{10}, o_\downarrow)\}$$

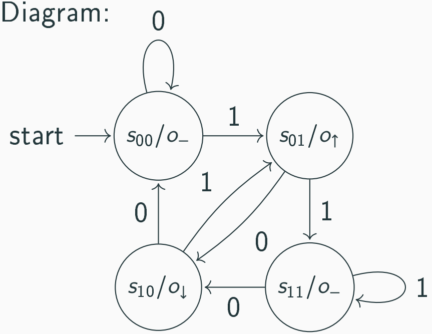
Tabelka δ:

q	a	q_o
$s_{?0}$	0	s_{00}
$s_{?1}$	0	s_{10}
$s_{?0}$	1	s_{01}
$s_{?1}$	1	s_{11}

Tab. χ:

q	o
s_{00}	o_-
s_{01}	$o_\uparrow$
s_{10}	$o_\downarrow$
s_{11}	o_-

Diagram:



Wykrywanie zmian (Moore) – przydział stanów

$$\mathcal{M} = \langle \{00, 01, 10, 11\}, \{0, 1\}, \{00, 01, 10\}, \delta, \chi, 00 \rangle$$
$$\delta = \{(ca, b, ab) \mid a \in \{0, 1\}, b \in \{0, 1\}, c \in \{0, 1\}\}$$
$$\chi = \{(00, 00), (11, 00), (01, 01), (10, 10)\}$$

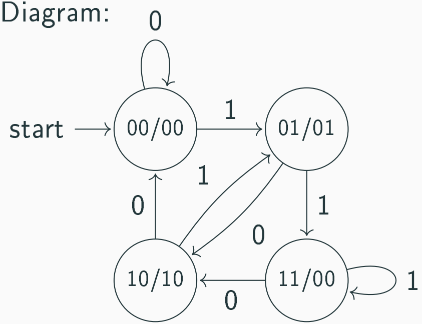
Tabelka δ:

q	a	q_o
?0	0	00
?1	0	10
?0	1	01
?1	1	11

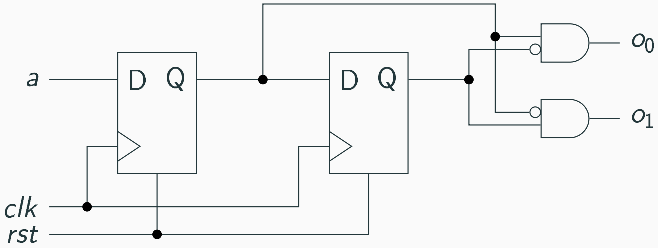
Tab. χ:

q	o
00	00
01	01
10	10
11	00

Diagram:



Wykrywanie zmian (Moore) – schemat



Wykrywanie zmian (Moore) – SystemVerilog

```
module detect_moore(input clk, rst, a, output [1:0] out);
    logic [1:0] q;              // stan automatu
    always_ff @(posedge clk or posedge rst)
        if (rst) q <= 2'b0;   // stan początkowy
        else q <= {q[0], a}; // funkcja przejścia
    assign out[0] = !q[1] && q[0]; // funkcja wyjścia
    assign out[1] = q[1] && !q[0];
endmodule
```

Automaty Mealy'ego

Definicja (automat Mealy'ego)
Krotka $\mathcal{M} = \langle Q, \Sigma, \Omega, \delta, \chi, q_0 \rangle$, gdzie:

- Q – skończony niepusty zbiór *stanów*,
- Σ – skończony niepusty *alfabet* wejściowy,
- Ω – skończony niepusty *alfabet* wyjściowy,
- $\delta : Q \times \Sigma \to Q$ – *funkcja przejścia*,
- $\chi : Q \times \Sigma \to \Omega$ – *funkcja wyjścia*,
- $q_0 \in Q$ – *stan początkowy*.

Automaty Mealy'ego

Definicja (automat Mealy'ego)
Krotka $\mathcal{M} = \langle Q, \Sigma, \Omega, \delta, \chi, q_0 \rangle$, gdzie:

- Q – skończony niepusty zbiór *stanów*,
- Σ – skończony niepusty *alfabet* wejściowy,
- Ω – skończony niepusty *alfabet* wyjściowy,
- $\delta : Q \times \Sigma \to Q$ – *funkcja przejścia*,
- $\chi : Q \times \Sigma \to \Omega$ – *funkcja wyjścia*,
- $q_0 \in Q$ – *stan początkowy*.

Automaty Moore'a są też automatami Mealy'ego.

Automaty Mealy'ego – definicje

Niech $\mathcal{M} = \langle Q, \Sigma, \Omega, \delta, \chi, q_0 \rangle$.

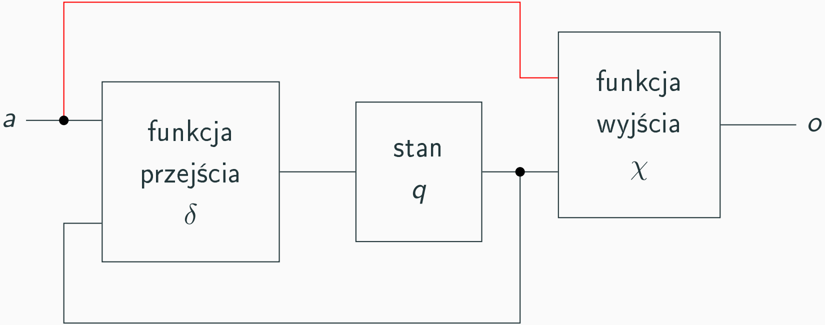
- $O(\mathcal{M}) : Q \times \Sigma^* \to \Omega^*$ – wyjście dla zadanego wejścia i stanu

$$O(\mathcal{M})(q, \epsilon) = \epsilon$$
$$O(\mathcal{M})(q, aw) = \chi(q, a) O(\mathcal{M})(\delta(q, a), w)$$

- $O(\mathcal{M}) : \Sigma^* \to \Omega^*$ – wyjście dla zadanego wejścia, zaczynając od stanu początkowego

$$O(\mathcal{M})(w) = O(\mathcal{M})(q_0, w)$$

Automaty Mealy'ego – idea



Automat Mealy'ego – przykład: wykrywanie zmian

$$\mathcal{M} = \langle \{s_0, s_1\}, \{0,1\}, \{o_-, o_\uparrow, o_\downarrow\}, \delta, \chi, s_0 \rangle$$

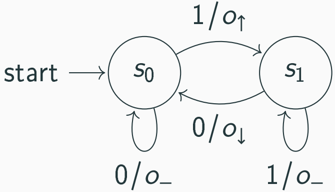
$$\delta = \{(s_b, a, s_a) \mid a \in \{0,1\}, b \in \{0,1\}\}$$

$$\chi = \{(s_0, 0, o_-), (s_1, 1, o_-), (s_0, 1, o_\uparrow), (s_1, 0, o_\downarrow)\}$$

Tabelka δ i χ:

q	a	q_{o}	o
s_0	0	s_0	o_-
s_0	1	s_1	$o_\uparrow$
s_1	0	s_0	$o_\downarrow$
s_1	1	s_1	o_-

Diagram:



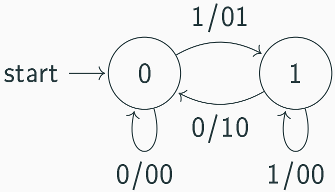
Wykrywanie zmian (Mealy) – przydział stanów

$$\mathcal{M} = \langle \{0,1\}, \{0,1\}, \{0,01,10\}, \delta, \chi, 0\rangle$$
$$\delta = \{(b,a,a) \mid a \in \{0,1\}, b \in \{0,1\}\}$$
$$\chi = \{(0,0,00),(1,1,00),(0,1,01),(1,0,10)\}$$

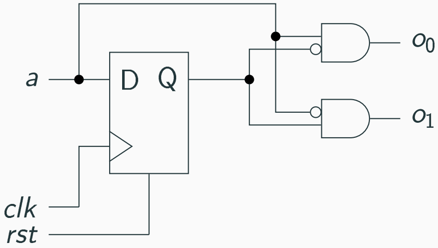
Tabelka δ i χ:

q	a	q_o	o
0	0	0	00
0	1	1	01
1	0	0	10
1	1	1	00

Diagram:



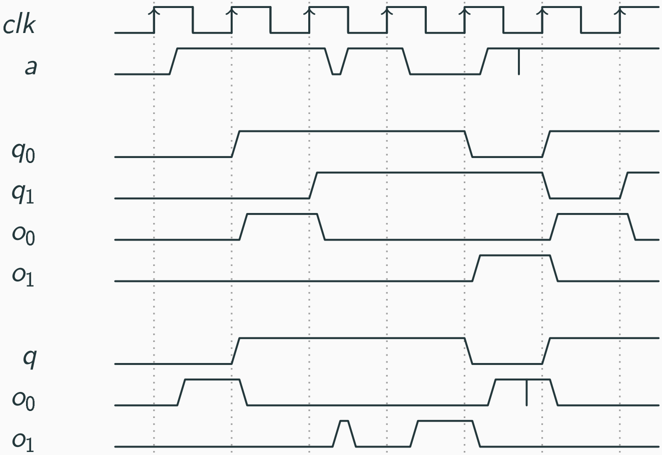
Wykrywanie zmian (Mealy) – schemat



Wykrywanie zmian (Mealy) – SystemVerilog

```
module detect_mealy(input clk, rst, a, output [1:0] out);
    logic q;                    // stan automatu
    always_ff @(posedge clk or posedge rst)
        if (rst) q <= 1'b0;     // stan początkowy
        else q <= a;            // funkcja przejścia
    assign out[0] = !q && a;    // funkcja wyjścia
    assign out[1] = q && !a;
endmodule
```

Timing – automat Moore'a i Mealy'ego



Moore czy Mealy?

Moore	Mealy
Wyjścia synchroniczne z zegarem	Wyjścia zależne od wejść
Reakcja w kolejnym cyklu zegara	Reakcja w tym samym cyklu
Ograniczone ryzyko pomyłki	Ryzyko niezamierzonego sprzężenia zwrotnego
Więcej stanów	Mniej stanów
Nie propaguje glitchy	Może propagować glitche
Sekwencjonuje przetwarzanie	Wydłuża ścieżkę krytyczną

Projektowanie przy użyciu automatów

Projektowanie przy użyciu automatów

- Wybór typu automatu – Moore lub Mealy
- Narysowanie diagramu stanów
- Minimalizacja automatu (na kolejnym wykładzie)
- Przydział stanów
- Implementacja funkcji przejścia i wyjścia
- Schemat lub kod w języku opisu sprzętu

Automaty ze stanem abstrakcyjnym

Często w praktyce pojawiają się automaty, których stany mają znaczenie opisywane słownie.

Rozważmy uproszczony sterownik skrzyżowania drogi głównej z podporządkowaną. Założenia:

- Nie ma kombinacji czerwony + żółty,
- Droga główna otrzymuje tyle samo czasu co podporządkowana,
- Liczniki odliczające czas są zewnętrzne dla automatu,
- Droga podporządkowana posiada detektor indukcyjny,
- Jeśli przy drodze podporządkowanej nie ma samochodów, droga główna ma cały czas światło zielone.

Sterownik skrzyżowania

Stany:

- s_{gc} – główna czerwony (podporządkowana zielony)
- s_{py} – podporządkowana żółty (główna czerwony)
- s_{pc} – podporządkowana czerwony (główna zielony)
- s_{gy} – główna żółty (podporządkowana czerwony)

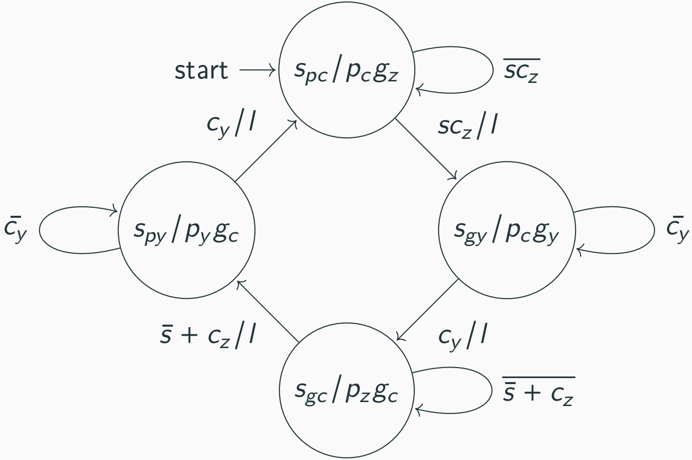
Wejścia:

- s – samochód przy drodze podporządkowanej
- c_z, c_y – minął czas zielony/żółty

Wyjścia:

- g_c, g_y, g_z, p_c, p_y, p_z – stan danego światła danej drogi
- l – rozpoczęcie odliczania czasu

Sterownik skrzyżowania – automat



Uproszczenia przy rysowaniu diagramu

- Gdy sygnał wyjściowy nie jest wymieniony, jego wartością jest 0
- Na krawędziach formuły rachunku Boole'a
- Rozwinięcie formuły do CDNF (kanonicznej postaci dysjunkcyjnej) opisuje wszystkie kombinacje wejść, dla których istnieje krawędź
- Jeśli w automacie Mealy'ego sygnał wyjściowy jest wymieniony przy stanie, to jest przy każdej krawędzi wychodzącej z tego stanu

Przydział stanów

Przyjmijmy taki przydział, używając kodów binarnych:

- s_{pc} – 00
- s_{gy} – 11
- s_{gc} – 10
- s_{py} – 01

Alternatywny przydział: kody one-hot

Funkcja przejścia i wyjścia – tabela

q	s	c_z	c_y	q_0	l	g_c	g_y	g_z	p_c	p_y	p_z
00 s_{pc}	0	?	?	00 s_{pc}	0	0	0	1	1	0	0
00 s_{pc}	1	1	?	11 s_{gy}	1	0	0	1	1	0	0
11 s_{gy}	?	?	0	11 s_{gy}	0	0	1	0	1	0	0
11 s_{gy}	?	?	1	10 s_{gc}	1	0	1	0	1	0	0
10 s_{gc}	1	0	?	10 s_{gc}	0	1	0	0	0	0	1
10 s_{gc}	?	?	?	01 s_{py}	1	1	0	0	0	0	1
01 s_{py}	?	?	0	01 s_{py}	0	1	0	0	0	1	0
01 s_{py}	?	?	1	00 s_{pc}	1	1	0	0	0	1	0

Znak „?” oznacza tu czasem „w pozostałych przypadkach”.

Funkcja przejścia i wyjścia – abstrakcyjnie

- $I \equiv (s_{gy} + s_{py})c_y + s_{pc}sc_z + s_{gc}(\bar{s} + c_z)$
- $g_c \equiv s_{gc} + s_{py}$
- $g_y \equiv s_{gy}$
- $g_z \equiv s_{pc}$
- $p_c \equiv s_{pc} + s_{gy}$
- $p_y \equiv s_{py}$
- $p_z \equiv s_{gc}$

Spostrzeżenie: stan zmienia się na kolejny, gdy I jest wysokie.

Implementacja w SystemVerilogu – część 1

```
module sygnalizator(
    input clk, nrst, s, cz, cy,
    output logic l, gc, gy, gz, pc, py, pz
);
    // kody stanów automatu
    const logic [1:0] PC = 2'b00, GY = 2'b11,
                      GC = 2'b10, PY = 2'b01;
    // stan automatu
    logic [1:0] q;
```

Implementacja w SystemVerilogu – część 2

```
// funkcja wyjścia -- światła
always_comb begin
    gc = 0; gy = 0; gz = 0; pc = 0; py = 0; pz = 0;
    unique case (q)
        PC: begin pc = 1; gz = 1; end
        GY: begin pc = 1; gy = 1; end
        GC: begin gc = 1; pz = 1; end
        PY: begin gc = 1; py = 1; end
    endcase
end
```

Powyższy styl jawnie specyfikuje, że zawsze jest zapalone jedno światło dla każdej drogi.

Implementacja w SystemVerilogu – część 3

```
// funkcja wyjścia -- start licznika
always_comb unique case (q)
    PC: l = s && cz;
    GC: l = !s || cz;
    GY, PY: l = cy;
    default: l = 1'bx;
endcase
```

Wyjście opisane wyrażeniem algebry Boole'a dla każdego stanu.

Implementacja w SystemVerilogu – część 4

```
    // funkcja przejścia
    always_ff @(posedge clk or negedge nrst)
    if (!nrst) q <= PC;
    else if (l) unique case(q)
        PC: q <= GY;
        GY: q <= GC;
        GC: q <= PY;
        PY: q <= PC;
    endcase
endmodule
```

Automaty skończone – idiom w SystemVerilogu

- Stany nazwane przez `const logic` (opcjonalnie *`define*)
- Funkcja przejścia blokiem `always_ff` z instrukcją `case`
- Funkcja wyjścia blokami `assign` lub `always_comb`

Automat zakodowany zgodnie z idiomem może być optymalizowany przez narzędzie do syntezy.

W DigitalJS: opcja „FSM transform”.