Logika cyfrowa

Wykład 11: więcej o automatach, diagramy ASM

Marek Materzok

12 maja 2021

Minimalizacja automatów

Pojęcie rozróżnialności stanów

Definicja (stany równoważne)
Stany $q_i, q_j \in Q$ są równoważne wtw automat uruchomiony od stanu q_i lub q_j dla takich samych wejść produkuje takie same wyjścia, to znaczy:

$$\forall w \in \Sigma^* \; O(\mathcal{M})(q_i, w) = O(\mathcal{M})(q_j, w)$$

Pojęcie rozróżnialności stanów

Definicja (stany równoważne)
Stany $q_i, q_j \in Q$ są równoważne wtw automat uruchomiony od stanu q_i lub q_j dla takich samych wejść produkuje takie same wyjścia, to znaczy:

$$\forall w \in \Sigma^* \; O(\mathcal{M})(q_i, w) = O(\mathcal{M})(q_j, w)$$

Równoważność stanów jest trudna do ustalenia, ale łatwo stwierdzić, kiedy stany *nie są* równoważne.

Pojęcie następnika

Definicja (a-następnik)
Stan $\delta(q, a)$ nazywamy a-następnikiem stanu q.

Idea algorytmu

Definicja (podział zbioru stanów)
Rodzina zbiorów niepustych i rozdzielnych P taka, że $\bigcup P = Q$, spełniająca warunek, że dla dowolnych równoważnych stanów $q_i, q_j \in Q$, należą one do tego samego zbioru z P.

Będziemy pisać $P = (q_1 q_2)(q_3 q_4 q_5)$
dla podziału $P = \{\{q_1, q_2\}, \{q_3, q_4, q_5\}\}$.
Zbiory stanów należące do P będziemy nazywać blokami.

Przez $P(q)$ oznaczymy blok P zawierający q. Dla przykładu powyżej $P(q_1) = \{q_1, q_2\}$.

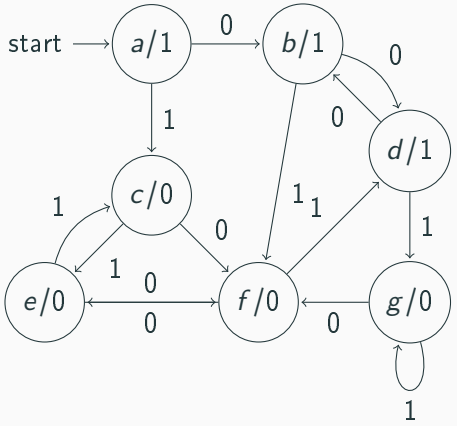
Algorytm minimalizacji

Idea algorytmu: iteracja stałopunktowa. Zaczniemy od podziału $P = \{Q\}$. Będziemy rozbijać bloki P na mniejsze, do momentu, gdy nie będzie to już możliwe.

1. Rozdziel do osobnych bloków stany, które produkują różne wyjścia.
   - Moore: rozdziel, gdy $\chi(q_1) \neq \chi(q_2)$.
   - Mealy: rozdziel, gdy istnieje $a \in \Sigma$ t. że $\chi(q_1, a) \neq \chi(q_2, a)$.
2. Rozdziel do osobnych bloków stany, których a-następnicy są w różnych blokach. Tzn. gdy $P(\delta(q_1, a)) \neq P(\delta(q_2, a))$.

Przykład

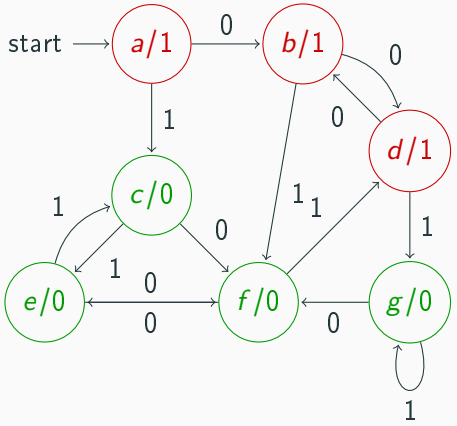
q	q_o $\bar{w}$	q_o w	o
a	b	c	1
b	d	f	1
c	f	e	0
d	b	g	1
e	f	c	0
f	e	d	0
g	f	g	0



$P_1 = (abcdefg)$

Przykład

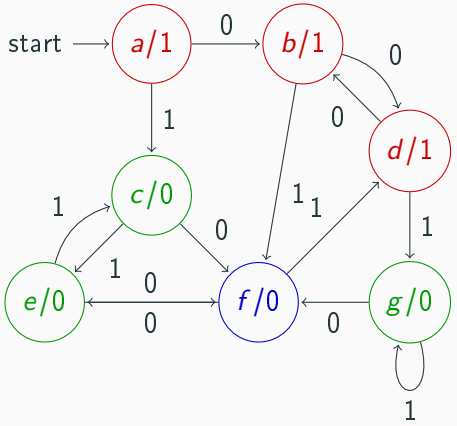
q	q_o $\bar{w}$	q_o w	o
a	b	c	1
b	d	f	1
c	f	e	0
d	b	g	1
e	f	c	0
f	e	d	0
g	f	g	0



$P_2 = (abd)(cefg)$

Przykład

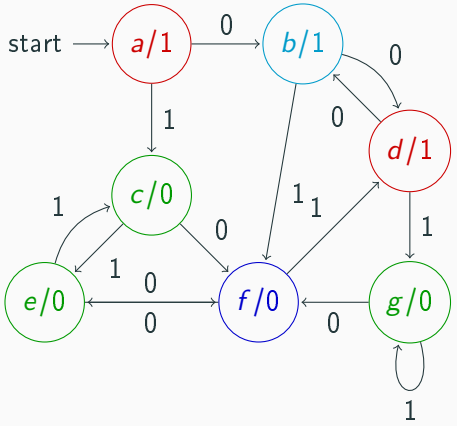
q	q_o $\bar{w}$	q_o w	o
a	b	c	1
b	d	f	1
c	f	e	0
d	b	g	1
e	f	c	0
f	e	d	0
g	f	g	0



$P_3 = (abd)(ceg)(f)$

Przykład

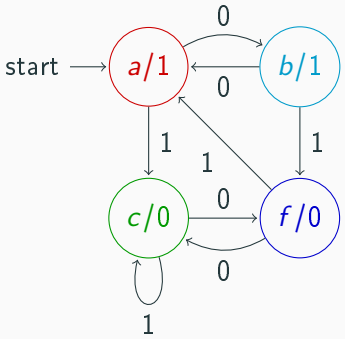
q	q_o $\bar{w}$	q_o w	o
a	b	c	1
b	d	f	1
c	f	e	0
d	b	g	1
e	f	c	0
f	e	d	0
g	f	g	0



$P_4 = (ad)(b)(ceg)(f)$

Przykład – po minimalizacji

q	q_o $\bar{w}$	q_o w	o
a	b	c	1
b	a	f	1
c	f	c	0
f	c	a	0



Przykład – automat wydający batoniki

Automat, który:

- Przyjmuje złotówki i dwuzłotówki
- Wydaje batoniki kosztujące 3 złote
- Zachowuje resztę do kolejnego zakupu

Przykład – automat wydający batoniki

Automat, który:

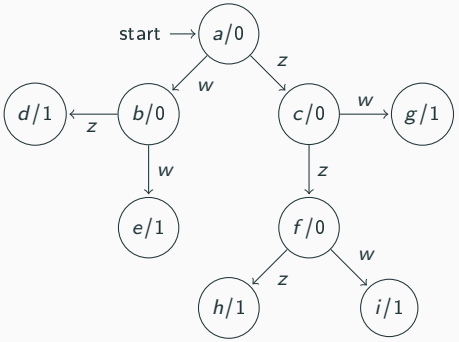
- Przyjmuje złotówki i dwuzłotówki
- Wydaje batoniki kosztujące 3 złote
- Zachowuje resztę do kolejnego zakupu

Szczegóły:

- Dwa wejścia z (**z**łoty) i w (d**w**a złote)
- Wrzut monety aktywuje z lub w na jeden cykl, nigdy oba naraz
- Gdy wydawany batonik, z i w niskie

Przykład – automat wydający batoniki

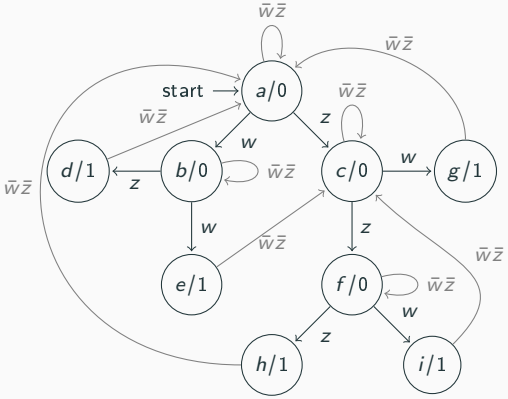
q	q_o $\bar{w}\bar{z}$	q_o $\bar{w}z$	q_o $w\bar{z}$	o
a	a	c	b	0
b	b	d	e	0
c	c	f	g	0
d	a	-	-	1
e	c	-	-	1
f	f	h	i	0
g	a	-	-	1
h	a	-	-	1
i	c	-	-	1



$P_1 = (abcdefghi)$

Przykład – automat wydający batoniki

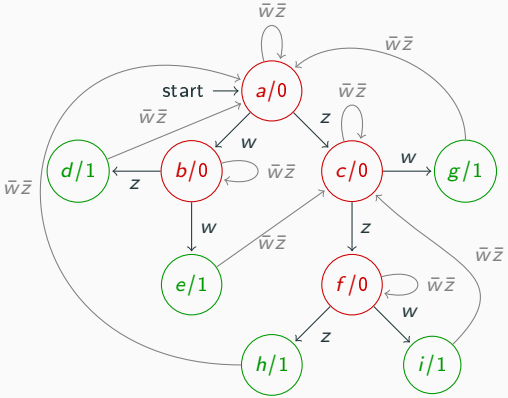
q	q_o $\bar{w}\bar{z}$	q_o $\bar{w}z$	q_o $w\bar{z}$	o
a	a	c	b	0
b	b	d	e	0
c	c	f	g	0
d	a	-	-	1
e	c	-	-	1
f	f	h	i	0
g	a	-	-	1
h	a	-	-	1
i	c	-	-	1



$P_1 = (abcdefghi)$

Przykład – automat wydający batoniki

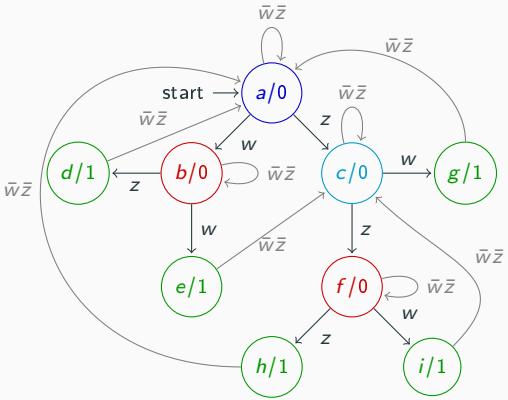
q	q_o $\bar{w}\bar{z}$	q_o $\bar{w}z$	q_o $w\bar{z}$	o
a	a	c	b	0
b	b	d	e	0
c	c	f	g	0
d	a	-	-	1
e	c	-	-	1
f	f	h	i	0
g	a	-	-	1
h	a	-	-	1
i	c	-	-	1



$$P_2 = (abcf)(deghi)$$

Przykład – automat wydający batoniki

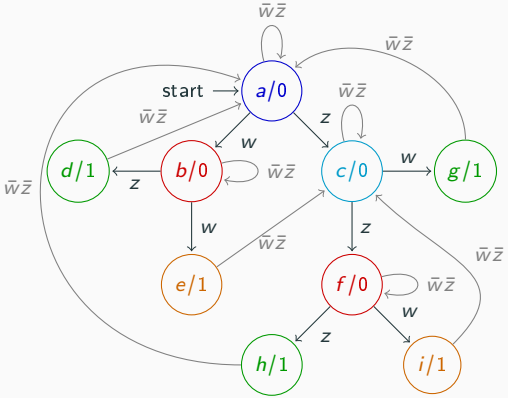
q	q_o $\bar{w}\bar{z}$	q_o $\bar{w}z$	q_o $w\bar{z}$	o
a	a	c	b	0
b	b	d	e	0
c	c	f	g	0
d	a	-	-	1
e	c	-	-	1
f	f	h	i	0
g	a	-	-	1
h	a	-	-	1
i	c	-	-	1



$$P_3 = (a)(c)(bf)(deghi)$$

Przykład – automat wydający batoniki

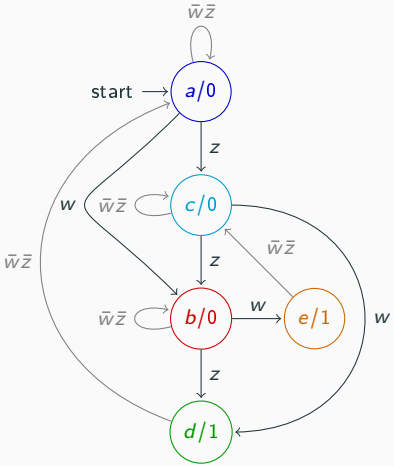
q	q_o $\bar{w}\bar{z}$	q_o $\bar{w}z$	q_o $w\bar{z}$	o
a	a	c	b	0
b	b	d	e	0
c	c	f	g	0
d	a	-	-	1
e	c	-	-	1
f	f	h	i	0
g	a	-	-	1
h	a	-	-	1
i	c	-	-	1



$$P_4 = (a)(c)(bf)(dgh)(ei)$$

Przykład – automat wydający batoniki, po minimalizacji

q	q_o $\bar{w}\bar{z}$	q_o $\bar{w}z$	q_o $w\bar{z}$	o
a	a	c	b	0
b	b	d	e	0
c	c	b	d	0
d	a	-	-	1
e	c	-	-	1



Przerzutniki JK i T w automatach skończonych

Przerzutniki JK i T – tabele stanów

Przerzutnik JK

q	j	k	q_o
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Przerzutnik T

q	t	q_o
0	0	0
0	1	1
1	0	1
1	1	0

Przerzutnik D

q	d	q_o
0	0	0
0	1	1
1	0	0
1	1	1

Odwzorowują stan bieżący i wejścia na kolejny stan.

Przerzutniki JK i T – tabele wzbudzeń

Przerzutnik JK

q	q_o	j	k
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

Przerzutnik T

q	q_o	t
0	0	0
0	1	1
1	0	1
1	1	0

Przerzutnik D

q	q_o	d
0	0	0
0	1	1
1	0	0
1	1	1

Odwzorowują bieżący i kolejny stan na wartości wejść potrzebne do uzyskania takiej zmiany stanu.

Tabele wzbudzeń dla automatów skończonych

- Tabelę wzbudzeń można zapisać jako funkcję częściową $e_j, e_k, e_t, e_d : Q \times Q \nrightarrow \mathbb{B}$.
- Potrzebną wartość wejścia przerzutnika x dla funkcji przejścia δ wyraża funkcja:

$$f(q, a) = e_x(q, \delta(q, a))$$

- Zwróć uwagę, że $e_d(q, \delta(q, a)) = \delta(q, a)$.

Przykład dla przerzutnika typu D

q_1	q_0	w	q_{o1}	q_{o0}	d_1	d_0
0	0	0	0	0	0	0
0	0	1	0	1	0	1
0	1	0	0	0	0	0
0	1	1	1	1	1	1
1	0	0	x	x	x	x
1	0	1	x	x	x	x
1	1	0	0	0	0	0
1	1	1	1	1	1	1

$$d_0 = w$$

$$d_1 = wq_0$$

Przykład dla przerzutnika typu T

q_1	q_0	w	q_{o1}	q_{o0}	t_1	t_0
0	0	0	0	0	0	0
0	0	1	0	1	0	1
0	1	0	0	0	0	1
0	1	1	1	1	1	0
1	0	0	x	x	x	x
1	0	1	x	x	x	x
1	1	0	0	0	1	1
1	1	1	1	1	0	0

$$t_0 = w \oplus q_0$$

$$t_1 = q_0(w \oplus q_1)$$

Przykład dla przerzutnika typu JK

q_1	q_0	w	q_{o1}	q_{o0}	j_1	j_0	k_1	k_0
0	0	0	0	0	0	0	x	x
0	0	1	0	1	0	1	x	x
0	1	0	0	0	0	x	x	1
0	1	1	1	1	1	x	x	0
1	0	0	x	x	x	x	x	x
1	0	1	x	x	x	x	x	x
1	1	0	0	0	x	x	1	1
1	1	1	1	1	x	x	0	0

$$j_0 = w$$

$$j_1 = wq_0$$

$$k_0 = \bar{w}$$

$$k_1 = \bar{w}$$

Ścieżka danych i sterowania

Dane i informacje sterujące

Sygnały można podzielić na *dane* i *informacje sterujące*:

Dane i informacje sterujące

Sygnały można podzielić na *dane* i *informacje sterujące*:

- **Dane** to porcje informacji przetwarzane arytmetyką, funkcjami logicznymi, operacjami bitowymi itp.
  Implementowane sumatorami, dekoderami, multiplekserami, licznikami, rejestrami itp.

Dane i informacje sterujące

Sygnały można podzielić na *dane* i *informacje sterujące*:

- **Dane** to porcje informacji przetwarzane arytmetyką, funkcjami logicznymi, operacjami bitowymi itp.
  Implementowane sumatorami, dekoderami, multiplekserami, licznikami, rejestrami itp.
- **Informacje sterujące** zarządzają elementami przetwarzającymi dane.

Dane i informacje sterujące

Sygnały można podzielić na *dane* i *informacje sterujące*:

- **Dane** to porcje informacji przetwarzane arytmetyką, funkcjami logicznymi, operacjami bitowymi itp.
  Implementowane sumatorami, dekoderami, multiplekserami, licznikami, rejestrami itp.
- **Informacje sterujące** zarządzają elementami przetwarzającymi dane.

Projektowanie układu cyfrowego można podzielić na projektowanie częsci przetwarzającej dane i części kontrolującej przetwarzanie.

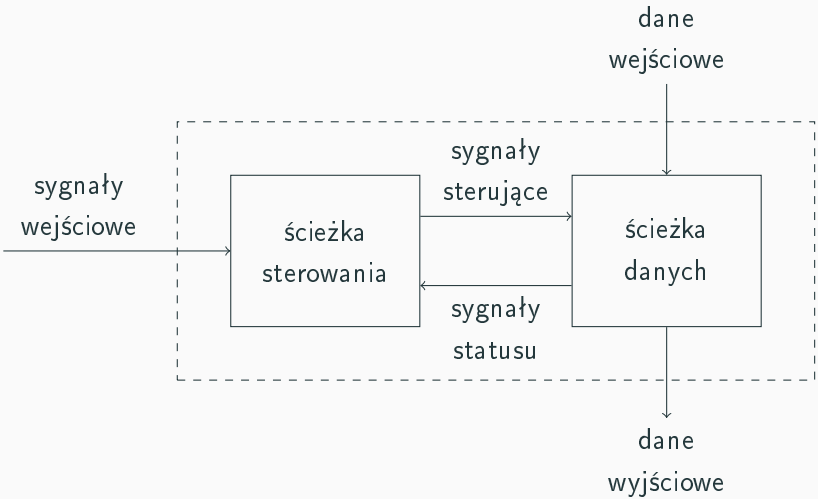
Ścieżka danych i sterowania

- *Ścieżka danych* manipuluje zawartością rejestrów. Przetwarza *dane wejściowe* w *dane wyjściowe*.

Ścieżka danych i sterowania

- *Ścieżka danych* manipuluje zawartością rejestrów.
  Przetwarza *dane wejściowe* w *dane wyjściowe*.
- *Ścieżka sterowania* wydaje polecenia ścieżce danych.
  Podejmuje decyzje na podstawie *sygnałów wejściowych* (pochodzących z zewnątrz) i *sygnałów statusu* (generowanych przez ścieżkę danych).
  Może być stanowa (automat skończony) lub bezstanowa (układ kombinacyjny).

Ścieżka danych i sterowania



Przykład – liczenie na przemian w górę i w dół

Kolejność liczenia: 0 1 2 3 2 1 0 1 2 ...

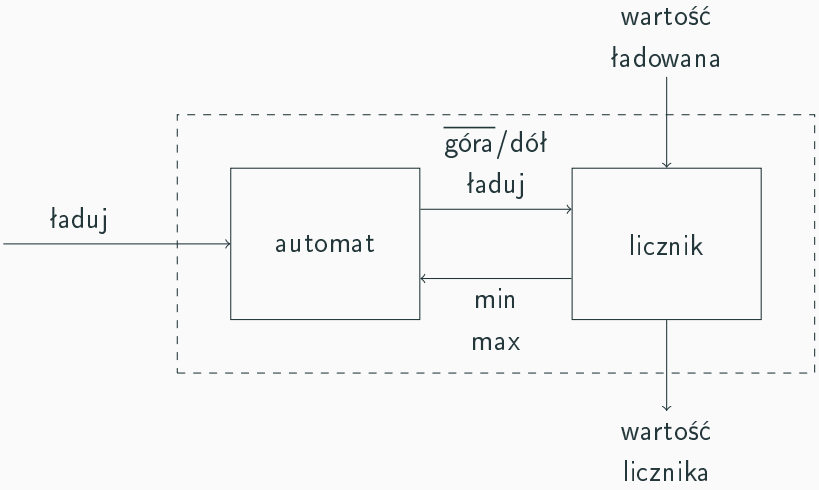
Ścieżka danych:

- licznik liczący w górę/dół z ładowaniem równoległym
- wykrywanie samych zer (min) i samych jedynek (max)

Ścieżka sterowania:

- automat sterujący kierunkiem licznika i ładowaniem

Przykład – ogólny schemat



Przykład – automat

Automat Mealy'ego

- l – ładowanie
- b – licznik w minimum
- t – licznik w maksimum
- d – kierunek liczenia
  (1 – dół, 0 – góra)

l	b	t	d	d_o	l_o
0	0	0	d	d	0
0	1	0	x	0	0
0	0	1	x	1	0
0	1	1	x	x	x
1	x	x	x	0	1

Przykład – automat

Automat Mealy'ego

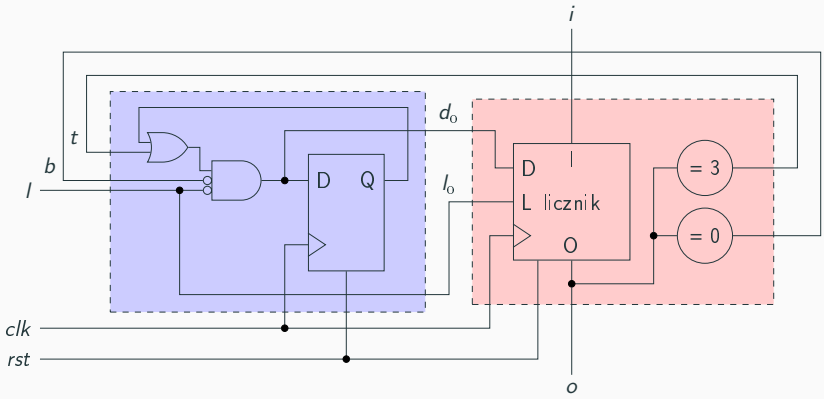
- l – ładowanie
- b – licznik w minimum
- t – licznik w maksimum
- d – kierunek liczenia (1 – dół, 0 – góra)

l	b	t	d	d_o	l_o
0	0	0	d	d	0
0	1	0	x	0	0
0	0	1	x	1	0
0	1	1	x	x	x
1	x	x	x	0	1

$$d_o = \overline{l}\overline{b}(t + d)$$

$$l_o = l$$

Przykład – schemat



ścieżka sterowania / ścieżka danych

Ścieżka sterowania – SystemVerilog

```
module ctlpath(
    input clk, rst, t, b, l,
    output d_o, l_o
);
    logic d;
    always_ff @(posedge clk or posedge rst)
        if (rst) d <= 0'b0;
        else d <= d_o;
    assign d_o = (d || t) && !b && !l;
    assign l_o = l;
endmodule
```

Ścieżka danych – SystemVerilog

```
module datapath(
    input clk, rst, d, l,
    input [1:0] i,
    output t, b,
    output logic [1:0] o
);
    always_ff @(posedge clk or posedge rst)
        if (rst) o <= 2'd0;
        else if (l) o <= i;
        else o <= d ? o - 1 : o + 1;
    assign t = o == 2'd3;
    assign b = o == 2'd0;
endmodule
```

Moduł licznika – SystemVerilog

```
module counter(
    input clk, rst, l,
    input [1:0] i,
    output [1:0] o
);
    logic d_o, l_o, t, b;
    ctlpath ctlpath(clk, rst, t, b, l, d_o, l_o);
    datapath datapath(clk, rst, d_o, l_o, i, t, b, o);
endmodule
```

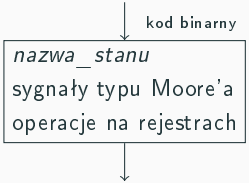
Diagramy algorytmiczne (ASM)

Diagramy algorytmiczne

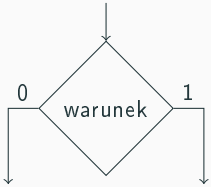
- Ang. *Algorithmic State Machine* (ASM)
- Odmiana schematów blokowych dostosowana do opisywania automatów skończonych
- Jawna informacja o czasie oraz sposobie generowania wyjść
- Trzy rodzaje bloków:
  - *blok stanu* – opisuje stany automatu i odpowiadające im wyjścia (typu Moore'a)
  - *blok decyzji* – opisuje wpływ wejść na działanie automatu
  - *blok warunkowy* – opisuje wyjścia zależne od decyzji (typu Mealy'ego)

Bloki w diagramach algorytmicznych

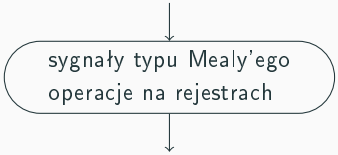
Blok stanu



Blok decyzji



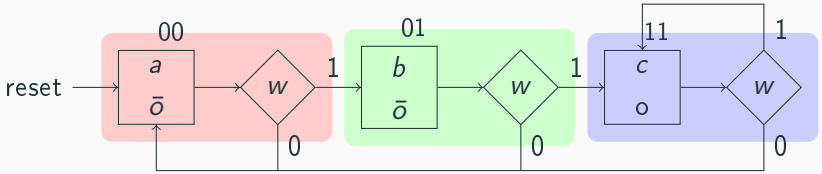
Blok warunkowy

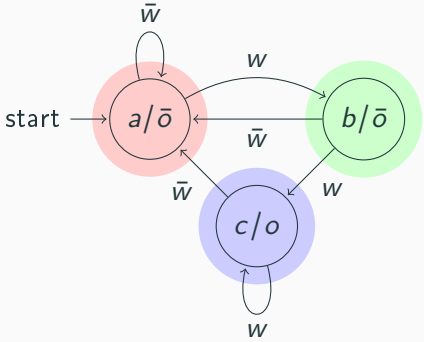


Czas w diagramach algorytmicznych

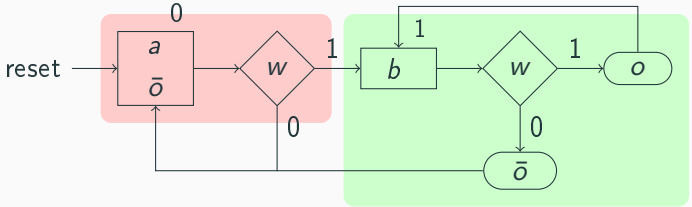
- Bieżący stan – jeden z bloków stanu
  - Pod blok stanu podlega poddiagram wszystkich następnych bloków aż do napotkania kolejnego bloku stanu
- Wartości wyjść:
  - podane w bloku stanu
  - podane w blokach warunkowych na ścieżce przez bloki decyzji odpowiadającej wartościom wejść
- Następny stan – blok stanu na końcu ścieżki przez bloki decyzji odpowiadającej wartościom wejść

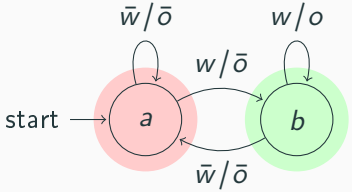
Diagram algorytmiczny – wykrywanie dwóch jedynek





Wykrywanie dwóch jedynek, wersja Mealy





Powrót do przykładu – sterownik skrzyżowania

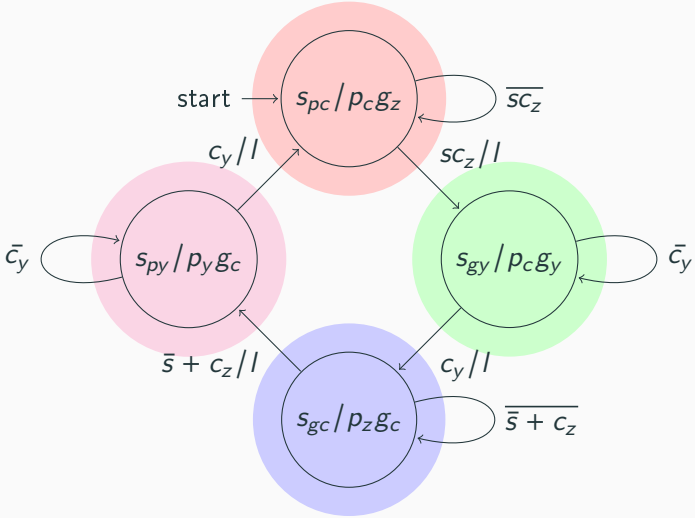
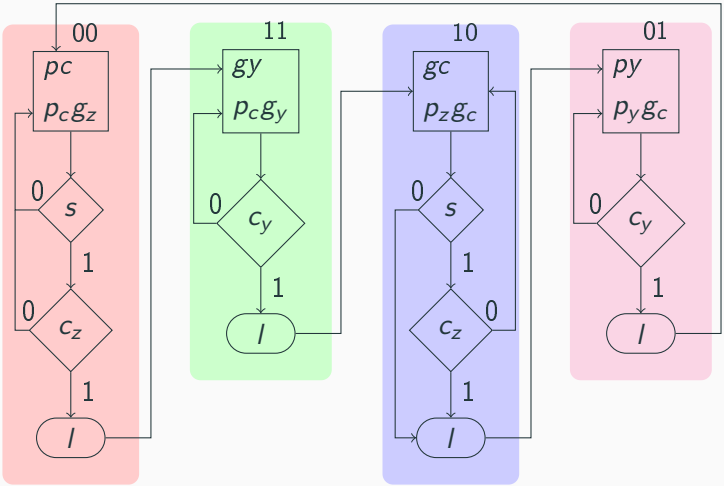


Diagram algorytmiczny – sterownik skrzyżowania



Operacje na rejestrach w diagramach algorytmicznych

W blokach stanu i blokach warunkowych mogą pojawiać się operacje na rejestrach, zapisywane:

$$\textit{rejestr} \leftarrow \textit{wyrażenie}$$

Operacje na rejestrach odpowiadają przypisaniom nieblokującym SystemVeriloga i konwencji RTL:

- wartości rejestrów nie zmieniają się w ramach stanu
- wszystkie operacje wykonują się współbieżnie przy zmianie stanu.

Operacje w bloku stanu i w blokach warunkowych na ścieżce do kolejnego bloku stanu wykonują się współbieżnie!

Diagram algorytmiczny – licznik

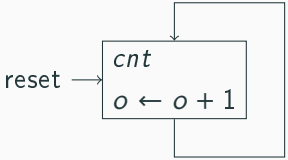


Diagram algorytmiczny – licznik z aktywacją

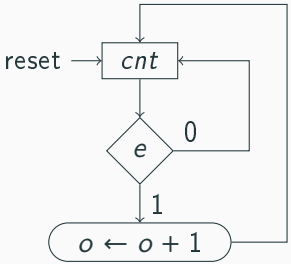


Diagram algorytmiczny – licznik z aktywacją i zerowaniem

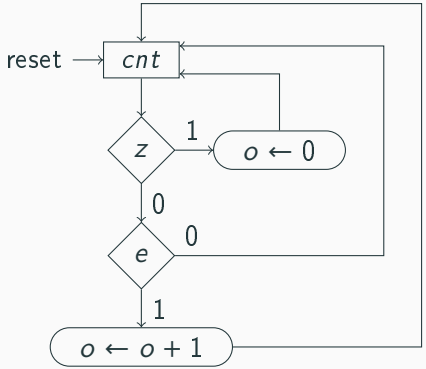
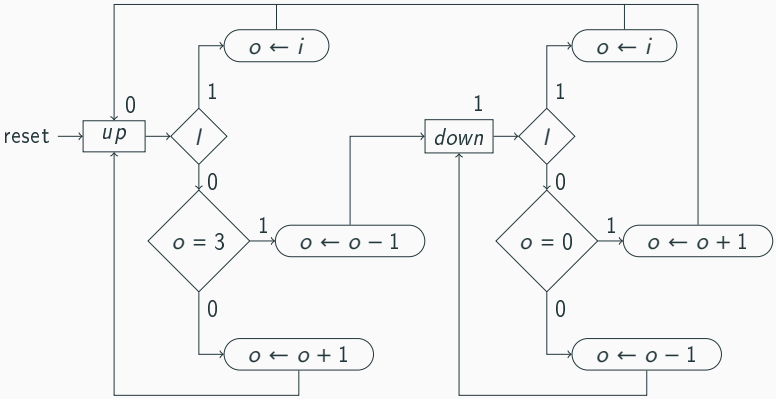


Diagram algorytmiczny – licznik zmieniający kierunek



Przykład – SystemVerilog

```
module example(
  input clk, rst, l,
  input [1:0] i,
  output logic [1:0] o
);
  const logic UP = 1'b0;
  const logic DN = 1'b1;
  logic s; // stan
  always_ff @(posedge clk
           or posedge rst)
    if (rst) begin
      s <= UP;
      o <= 2'b0;
    end else case (s)
      UP: if (l) o <= i;
        else if (o == 2'd3) begin
          o <= o - 1;
          s <= DN;
        end else o <= o + 1;
      DN: if (l) begin
          s <= UP;
          o <= i;
        end else if (o == 2'd0) begin
          s <= UP;
          o <= o + 1;
        end else o <= o - 1;
    endcase
endmodule
```

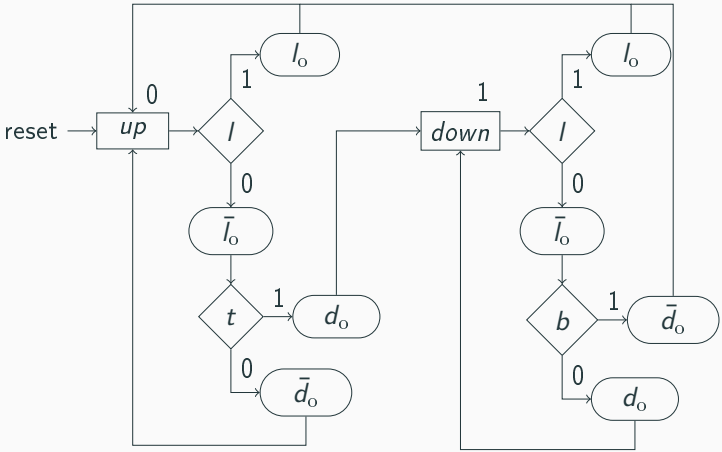
Przykład, refactor – SystemVerilog

```
module example(
  input clk, rst, l,
  input [1:0] i,
  output logic [1:0] o
);
  const logic UP = 1'b0;
  const logic DN = 1'b1;
  logic s; // stan
  always_ff @(posedge clk
           or posedge rst)
    if (rst) begin
      s <= UP;
      o <= 2'b0;
    end else if (l) begin
      s <= UP;
      o <= i;
    end else case (s)
      UP: if (o == 2'd3) begin
          s <= DN;
          o <= o - 1;
        end else o <= o + 1;
      DN: if (o == 2'd0) begin
          s <= UP;
          o <= o + 1;
        end else o <= o - 1;
    endcase
endmodule
```

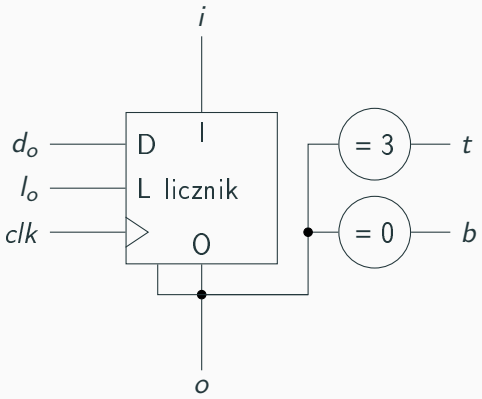
Diagramy algorytmiczne a ścieżka danych i sterowania

- Diagram algorytmiczny z operacjami na rejestrach łączy przetwarzanie danych z przepływem sterowania
- Aby je rozdzielić, należy:
  - Opracować ścieżkę danych potrafiącą wykonać każdą z operacji w diagramie ASM
  - Zastąpić operacje na rejestrach w diagramie ASM przez ustawienie odpowiednich sygnałów sterujacych
  - Zastąpić sprawdzanie stanu rejestrów w diagramie ASM przez badanie wartości odpowiednich sygnałów statusu

Diagram algorytmiczny dla ścieżki sterowania



Ścieżka danych dla licznika – przypomnienie



Przykład, ścieżka sterowania – SystemVerilog

```
module example_ctlpath(
  input clk, rst, l, t, b,
  output l_o, d_o
);
  const logic UP = 1'b0,
              DN = 1'b1;
  logic s;
  always_ff @(posedge clk
           or posedge rst)
    if (rst) s <= UP;
    else if (l) s <= UP;
    else case (s)
      UP: if (t) s <= DN;
      DN: if (b) s <= UP;
    endcase
  always_comb begin
    d_o = 1'bx;
    if (l) l_o = 1'b1;
    else begin
      l_o = 1'b0;
      case (s)
        UP: if (t) d_o = 1'b1;
            else   d_o = 1'b0;
        DN: if (b) d_o = 1'b0;
            else   d_o = 1'b1;
      endcase
    end
  end
endmodule
```

Przykład, ścieżka sterowania, refactor – SystemVerilog

```
module example_ctlpath(
  input clk, rst, l, t, b,
  output l_o, d_o
);
  const logic UP = 1'b0,
              DN = 1'b1;
  logic s;
  always_ff @(posedge clk
           or posedge rst)
    if (rst) s <= UP;
    else if (l) s <= UP;
    else case (s)
      UP: if (t) s <= DN;
      DN: if (b) s <= UP;
    endcase
  assign l_o = l;
  always_comb begin
    d_o = 1'bx;
    if (!l) case (s)
      UP: d_o = t;
      DN: d_o = !b;
    endcase
  end
endmodule
```

Pamięci w diagramach algorytmicznych

W diagramach algorytmicznych mogą pojawiać się operacje odczytu oraz zapisu pamięci, zapisywane:

$$\ldots \mathit{tablica}[\mathit{adres}] \ldots$$

$$\mathit{tablica}[\mathit{adres}] \leftarrow \mathit{wyrażenie}$$

Aby „przetłumaczyć” diagram z dostępami do pamięci na moduł w SystemVerilogu, należy pamiętać o zasadach z wykładu o pamięciach, a w szczególności:

- Dbać o liczbę portów,
- Najlepiej wydzielić pamięć do osobnego modułu.