Chapter 4 Algorithmic State Machines and Finite State Machines

In this Chapter, we will introduce Algorithmic state machines and consider their use for description of the behavior of control units. Next, we will use algorithmic state machines to design Finite State Machines (FSM) with hardly any constraints on the number of inputs, outputs and states.

4.1 Flowcharts and Algorithmic state machines

4.1.1 Example of ASM. An Algorithmic state machine (ASM) is the directed connected graph containing an initial vertex (Begin), a final vertex (End) and a finite set of operator and conditional vertices (Fig. 1). The final, operator and conditional vertices have only one input, the initial vertex has no input. Initial and operator vertices have only one output, a conditional vertex has two outputs marked by "1" and "0". A final vertex has no outputs.

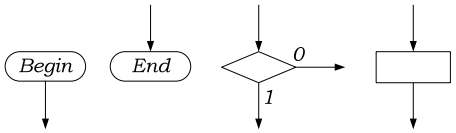

Figure 1. Vertices of Algorithmic state machine

As the first example, let us consider a very simple Traffic Light Controller (TLC) presented in the flowchart in Fig. 2. This controller is at the intersection of a main road and a secondary road. Immediately after vertex *Begin* we have a waiting vertex (one of the outputs of this vertex is connected to its input) with a logical condition *Start*. It means that the controller begins to work only when signal *Start = 1*. At this time, cars can move along the main road for two minutes. For that, the traffic light at the main road is green, the traffic light at the secondary road is red and the special timer that counts seconds is set to zero *(main_grn := 1; sec_red := 1; t := 0).*

Although our TLC is very simple it is also a little smart – it can recognize an ambulance on the road. When an ambulance is on the road the signal *amb* is equal to one (*amb = 1*), when there is no ambulance on the road this signal is equal to zero (*amb = 0*). First we will discuss the case when there are no ambulances on the road.

Thus, when *amb = 0* and *t = 120 sec* TLC transits into some intermediate state to allow cars to finish driving along the main road: *main_yel := 1; sec_red := 1; t := 0.* TLC is in this state only for three seconds (*t = 3 sec*), after which cars can move along the secondary road for 30 seconds: *main_red := 1; sec_grn := 1; t := 0.*

Thirty seconds later, if there are no ambulances on the road (*amb = 0; t = 30 sec*), there is one more intermediate state. Now cars must finish driving along the secondary road: *main_red := 1; sec_yel := 1; t := 0.* After three seconds, if, once again, there are no ambulances on the road, the process reaches vertex End, or, that is the same, it returns to the beginning vertex *Begin*.

When there is ambulance on the road (*amb = 1*) outputs of conditional vertices with logical condition *amb*, marked by “1” bring us to the intermediate state to let cars to finish their driving: *main_yel := 1; sec_yel := 1; t := 0.* One more logical condition *dmain* tells us where the ambulance is – whether it is on the main road or on the secondary one. If it is on the main road (*dmain = 1*), after three seconds the traffic

light will be green on the main road, otherwise (*dmain = 0*) the traffic light will be green on the secondary road.

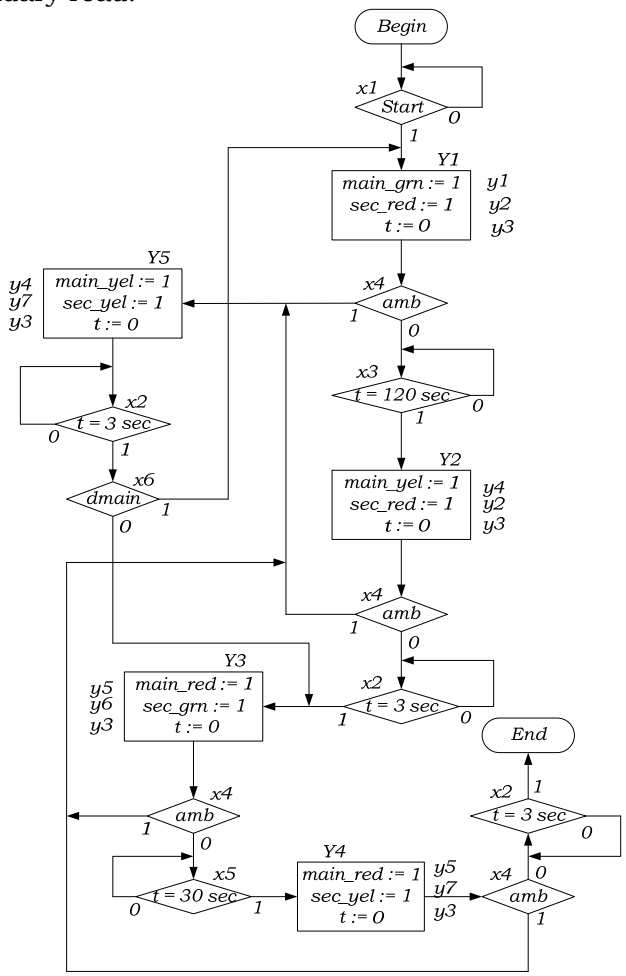

Figure 2. A simple Traffic Light Controller

In the flowchart, a logical condition is written in each conditional vertex. It is possible to write the same logical condition in different conditional vertices. A microinstruction (an operator), containing one, two, three or more microoperations, is written in each operator vertex of the flowchart. It is possible to write the same operator in different operator vertices.

If we replace logical conditions by $x_1, x_2, \ldots, x_L$, microoperations by $y_1, y_2, \ldots, y_N$ and operators by $Y_1, Y_2, \ldots, Y_T$ we will get Algorithmic State Machine (ASM). ASM for the flowchart in Fig. 2 is shown in Fig. 3.

ASM vertices are connected in such a way that:

1. Inputs and outputs of the vertices are connected by arcs directed from an output to an input, each output is connected with only one input;
2. Each input is connected with at least one output;
3. Each vertex is located on at least one of the paths from vertex “Begin” to vertex “End”. Hereinafter we will not consider ASMs with subgraphs, containing an infinite cycle. An example of such a subgraph with an infinite

loop between vertices with Y_1 and Y_3 is shown in Fig. 4. The dots in this ASM between vertex *"Begin"* and the conditional vertex with x_1 and between this vertex and vertex *"End"* mean that ASM has other vertices on the path from vertex *"Begin"* to vertex *"End"*. The vertices in the loop are not on the path from *"Begin"* to *"End"*.

4. One of the outputs of a conditional vertex can be connected with its input. We will call such conditional vertices the "*waiting vertices*", since they simulate the waiting process in the system behavior description.

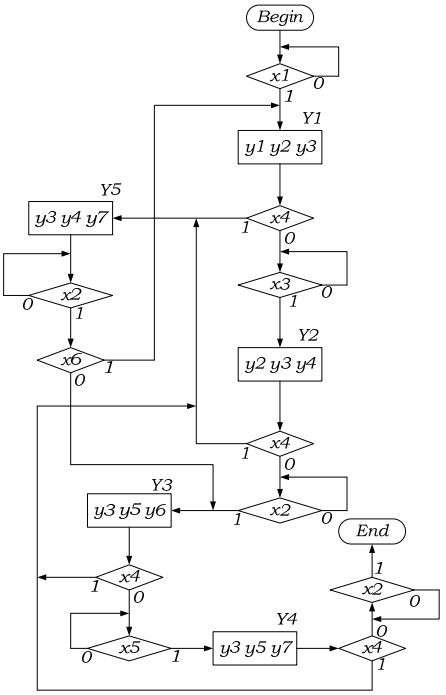

Figure 3. ASM for the flowchart in Fig. 2

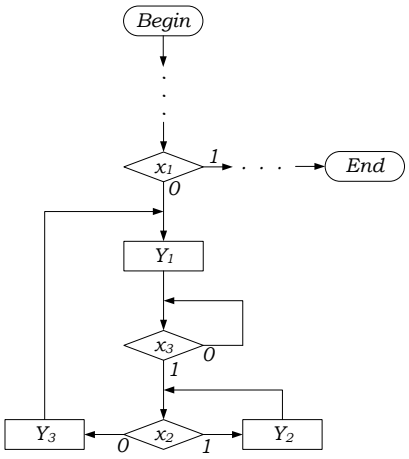

Figure 4. Subgraph with an infinite loop

One more example of ASM G_1 with logical conditions $X = \{x_1, \ldots, x_7\}$ and microoperations $Y = \{y_1, \ldots, y_{10}\}$ is shown in Fig. 5. This ASM has eight operators $Y_1, \ldots, Y_8$, they are written near operator vertices.

4.1.2 Transition functions. Let us discuss the paths between the vertex "Begin", the vertex "End" and operator vertices passing only through conditional vertices. We will write such paths as follows:

$$Y_i \tilde{x}_{i1} \ldots \tilde{x}_{iR} Y_j \qquad (1)$$

In such a path, $\tilde{x}_{ir}$ is equal to x_{ir} if the path proceeds from the conditional vertex with x_{ir} via output *'1'*, and $\tilde{x}_{ir}$ is equal to x'_{ir} if the path proceeds from the conditional vertex with x_{ir} via output *'0'*. For example, we have the following paths from Y_b (vertex *Begin*) in ASM G_1:

$Y_b\ x'_1\ Y_2;$
$Y_b\ x_1 x_2 x'_3\ Y_6;$
$Y_b\ x_1 x'_2\ Y_1;$
$Y_b\ x_1 x_2 x_3\ Y_5.$

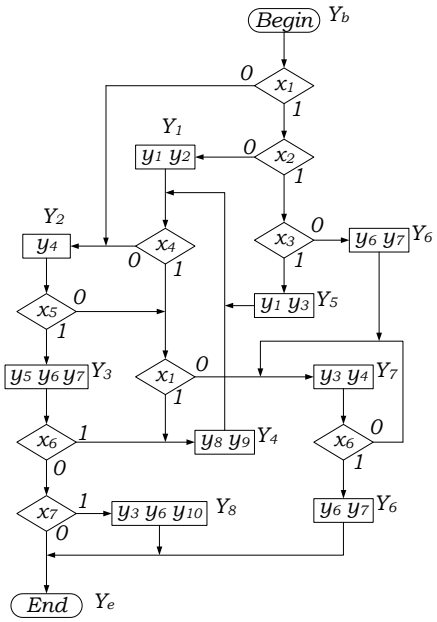

Figure 5. ASM G_1

Let us match a product of variables in the path (1) from operator vertex Y_i to operator vertex Y_j

$$\alpha_{ij} = \tilde{x}_{i1} \ldots \tilde{x}_{iR}$$

with this path from Y_i to Y_j. For example, for ASM G_1 in Fig. 5

$$a_{17} = x_4\, x'_1; \quad a_{12} = x'_4; \quad a_{14} = x_4\, x_1.$$

If there exist H paths between Y_i and Y_j through the conditional vertices, then

$$a_{ij} = a^1_{ij} + a^2_{ij} + \ldots + a^H_{ij}$$

where a^h_{ij} ($h = 1, \ldots, H$) is the product for the h-th path. Let us call a_{ij} a transition function from operator (microinstruction) Y_i to operator (microinstruction) Y_j.

Note that for the path Y_6Y_7 (operator Y_7 follows operator Y_6 immediately without conditional vertices) $a_{67} = 1$, as the product of an empty set of variables is equal to one.

4.1.3 Value of ASM at the sequence of vectors. Denote all possible L-component vectors of the logical conditions $x_1, \ldots, x_L$ by $\Delta_1, \ldots, \Delta_2{}^L$ and define the execution of an ASM on any given sequence of vectors $\Delta_1, \ldots, \Delta_{mq}$ beginning from the initial operator Y_b. We will demonstrate this procedure by means of ASM G_1 in Fig. 5 and the sequence (2) containing eight vectors $\Delta_1, \ldots, \Delta_8$:

		x_1	x_2	x_3	x_4	x_5	x_6	x_7
Δ_1	=	1	0	1	0	1	1	1
Δ_2	=	0	1	1	0	1	0	0
Δ_3	=	1	0	1	0	0	1	0
Δ_4	=	0	1	0	0	0	0	1
Δ_5	=	1	1	0	1	1	1	0
Δ_6	=	1	1	0	0	1	0	1
Δ_7	=	0	1	1	1	0	0	0
Δ_8	=	0	1	0	1	0	0	1

(2)

ASM G_1 in Fig. 5 contains logical variables $x_1, \ldots, x_7$ and operators $Y_b, Y_1, \ldots, Y_8, Ye$. Now let us find the sequence of operators which would be implemented, if we consecutively, beginning from Y_b, give variables the values from these vectors. We suppose that the values of logical conditions can be changed only during an execution of operators.

Step 1. Write the initial operator

$$Y_b.$$

Step 2. Let logical variables $x_1, \ldots, x_7$ take their values from vector Δ_1. From the set of the transition functions $a_{b1}, \ldots, a_{b8}, a_{be}$ we choose such a function a_{bt} that $a_{bt}(\Delta_1) = 1$. In our example for the operator Y_b, the following transition functions are not identically equal to zero:

$$a_{b5} = x_1\, x_2\, x_3; \qquad a_{b6} = x_1\, x_2\, x'_3; \qquad a_{b1} = x_1\, x'_2; \qquad a_{b2} = x'_1.$$

We will call such functions *non-trivial transition functions* to distinguish them from the trivial functions, which are identically equal to zero. Function a_{ij} is *trivial* if there is no path from operator Y_i to operator Y_j. In the example at this step, we choose the function a_{b1}, since only a_{b1} is equal to one on the first vector Δ_1:

$$a_{b1}\,(\Delta_1) = 1.$$

Write Y_1 to the right of Y_b:

$$Y_bY_1.$$

Step 3. Let $x_1,\ldots,x_7$ take their values from vector Δ_2. From the set of the transition functions $a_{11},\ldots, a_{18}, a_{1e}$ we choose non-trivial functions

$$a_{14} = x_4 x_1; \qquad a_{17} = x_4 x'_1; \qquad a_{12} = x'_4$$

and among them – the only function $a_{12}(\Delta_2) = 1$. Write Y_2 to the right of $Y_b Y_1$:

$$Y_b Y_1 Y_2.$$

The computational process for the given sequence of vectors may reach its end in two cases:

1. The final vertex "End" is reached. In this case, the last operator is Y_e. The number of operators in the operator row (without Y_b and Y_e) is less or equal (if we reached the final vertex with the last vector) to the number of vectors;
2. The vectors are exhausted but we have not yet reached the final vertex. In this case, the number of operators in the operator row is equal to the number of vectors.

In our example, we reached the final vertex "End" at the seventh vector

$$\Delta_7 = 0\ 1\ 1\ 1\ 0\ 0\ 0$$

and we get the row

$$Y_b\ Y_1\ Y_2\ Y_4\ Y_2\ Y_3\ Y_8\ Y_e. \tag{3}$$

The operator row thus obtained is *the value of the ASM G_1 for the given sequence of vectors (2).*

4.2 Synthesis of Mealy FSM

We will use Algorithmic state machines to describe the behavior of digital systems, mainly of their control units. But if we must construct a logic circuit of the control unit we should use a Finite state machine (FSM). We will consider methods of synthesis of FSM Mealy, Moore and their combined model implementing a given ASM, with hardly any constrains on the number of inputs, outputs and states.

4.2.1 Construction of a marked ASM. As an example we will use ASM G_1 in Fig. 6. A Mealy FSM implementing given ASM may be constructed in two stages:

Stage1. Construction of a marked ASM;

Stage 2. Construction of a state diagram (state graph).

At the first stage, the inputs of vertices following operator vertices are marked by symbols $a_1, a_2, \ldots, a_M$ as follows:

1. Symbol a_1 marks the input of the vertex following the initial vertex *"Begin"* and the input of the final vertex *"End"*;
2. Symbols $a_2, \ldots, a_M$ mark the inputs of all vertices following operator vertices;
3. Vertex inputs are marked only once;
4. Inputs of different vertices, except the final one, are marked by different symbols.

Marked ASM G_1 in Fig. 6 is a result of the first step. Symbols $a_1, \ldots, a_6$ are used to mark this ASM. Note, that we mark the inputs not only of conditional vertices but of operator vertices as well (see mark a_3 at the input of the vertex with operator Y_7). It is important that each marked vertex follows an operator vertex.

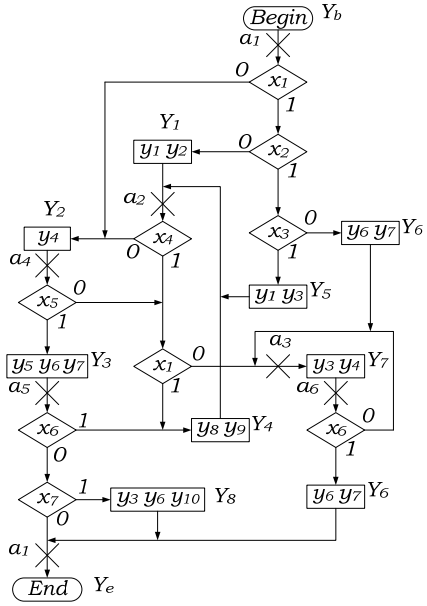

Figure 6. ASM G_1 marked for the Mealy FSM synthesis

4.2.2 Transition Paths. At the second stage, we will consider the following paths in the marked ASM:

$$a_m \tilde{x}_{m1} \ldots \tilde{x}_{mR_m} Y_g a_s \qquad (P1)$$

$$a_m \tilde{x}_{m1} \ldots \tilde{x}_{mR_m} a_1 \qquad (P2)$$

We call these paths *transition paths*. Thus, the path *P1* proceeds from a_m to a_s ($a_m = a_s$ is also allowed) and contains only one operator vertex at the end of this path. The path *P2* proceeds from a_m only to a_1 without operator vertex. Here, $\tilde{x}_{mr} = x_{mr}$, if on the transition path we leave the conditional vertex with x_{mr} via output '1' and $\tilde{x}_{mr} = x'_{mr}$ if we leave it via output '0'. If $R_m = 0$ on the path *P1*, two operator vertices follow one after another and this path turns into

$$a_m Y_g a_s .$$

There are sixteen transition paths in the marked ASM G_2 in Fig. 6:

$a_1\ x_1x_2x_3\ Y_5\ a_2$	$a_2\ x_4x_1\ Y_4\ a_2$	$a_4\ x_5\ Y_3\ a_5$	$a_5\ x'_6x_7\ Y_8\ a_1$
$a_1\ x_1x_2x'_3\ Y_6\ a_3$	$a_2\ x_4x'_1\ Y_7\ a_6$	$a_4\ x'_5x_1\ Y_4\ a_2$	$a_5\ x'_6x'_7\ a_1$
$a_1\ x_1x'_2\ Y_1\ a_2$	$a_2\ x'_4\ Y_2\ a_4$	$a_4\ x'_5x'_1\ Y_7\ a_6$	$a_6\ x_6\ Y_6\ a_1$
$a_1\ x'_1\ Y_2\ a_4$	$a_3\ Y_7\ a_6$	$a_5\ x_6\ Y_4\ a_2$	$a_6\ x'_6\ Y_7\ a_6$

Note, that the path $a_2\ x_4x'_1\ a_3$ doesn't correspond to the transition path *P1* (the operator vertex is absent on the path) and to transition path *P2* (it isn't a path to a_1). Thus, it isn't a transition path and we should go on to get the path $a_2\ x_4x'_1\ Y_7\ a_6$. For the same reason, paths $a_4\ x'_5x'_1\ a_3$ and $a_6\ x'_6\ a_3$ are not the transition paths either.

4.2.3 Graph of FSM. Next we construct a graph (state diagram) of FSM Mealy with states (marks) $a_1, \ldots, a_M$, obtained at the first stage. We have six such states $a_1, \ldots, a_6$ in our example. Thus, the FSM graph contains as many states as the number of marks we get at the previous stage. Now we should define transitions between these states.

FSM has a transition from state a_m to state a_s with input $X(a_m, a_s)$ and output Y_g (see the upper subgraph in Fig. 7) if, in ASM, there is transition path *P1*

$$a_m \tilde{x}_{m1} \ldots \tilde{x}_{mR_m} Y_g a_s .$$

Here $X(a_m, a_s)$ is the product of logical conditions written in this path:

$$X(a_m, a_s) = \tilde{x}_{m1} \ldots \tilde{x}_{mR_m} .$$

In exactly the same way, for the path $a_m Y_g a_s$ we have a transition from state a_m to state a_s with input $X(a_m, a_s) = 1$ and output Y_g, as the product of an empty set of variables is equal to zero. If, for a certain r $(r = 1, \ldots, R_m)$, symbol x_{mr} (or x'_{mr}) occurs several times on the transition path, all symbols x_{mr} (x'_{mr}) but one are deleted; if for a certain r $(r = 1, \ldots, R_m)$, both symbols x_{mr} and x'_{mr} occur on the transition path, this path is removed. In such a case $X(a_m, a_s) = 0$.

For the second transition path *P2*, FSM transits from state a_m to the initial state a_1 with input $X(a_m, a_1)$ and output Y_0 (see the lower subgraph in Fig. 7). Y_0 is the operator containing an empty set of microoperations.

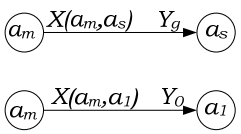

Figure 7. Subgraphs for transition paths *P1* and *P2*

As a result, we obtain a Mealy FSM with as many states as the number of marks we used to mark the ASM in Fig. 6. The state diagram of the Mealy FSM is shown in Fig. 8.

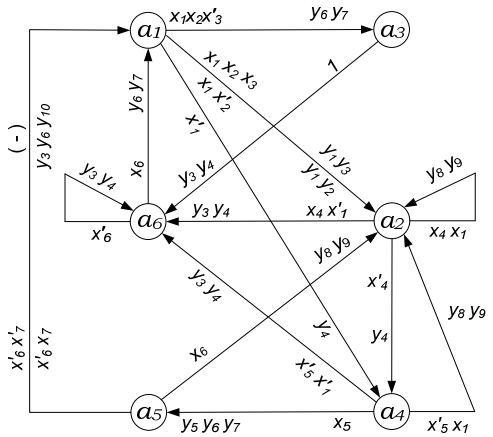

Figure 8. The state diagram of the Mealy FSM

4.2.4 How not to loose transition paths. Sometimes, if ASM contains many conditional vertices, it is difficult not to loose one or several transition paths. Here

we give a very simple algorithm to resolve this problem. This algorithm has only two steps.

1. Find the first transition path leaving each conditional vertex through output '*1*'. For subgraph of ASM in Fig. 9 we will get the following first path from state a_2:

$$a_2\ x_1\ x_2\ x_5\ Y_6\ a_3.$$

2. Invert the *last non-inverted* variable in the previous path, return to ASM and continue the path (if it is possible) leaving each conditional vertex through output '*1*'. To construct the second path, we should invert variable x_5. We cannot continue because we reached an operator vertex:

$$a_2\ x_1\ x_2\ x'_5\ Y_2\ a_3.$$

We should construct paths in the same manner until all variables in a transition path will be inverted. For our example, we will get the following paths:

$a_2\ x_1\ x'_2\ x_5\ x_6\ Y_3\ a_4;$
$a_2\ x_1\ x'_2\ x_5\ x'_6\ x_7\ x_4\ Y_5\ a_5;$
$a_2\ x_1\ x'_2\ x_5\ x'_6\ x_7\ x'_4\ Y_7\ a_4;$
$a_2\ x_1\ x'_2\ x_5\ x'_6\ x'_7\ Y_5\ a_5;$

$a_2\ x_1\ x'_2\ x'_5\ Y_2\ a_3;$
$a_2\ x'_1\ x_3\ x_7\ x_4\ Y_5\ a_5;$
$a_2\ x'_1\ x_3\ x_7\ x'_4\ Y_7\ a_4;$
$a_2\ x'_1\ x_3\ x'_7\ Y_5\ a_5;$

$a_2\ x'_1\ x'_3\ x_6\ Y_3\ a_4;$
$a_2\ x'_1\ x'_3\ x'_6\ x_7\ x_4\ Y_5\ a_5;$
$a_2\ x'_1\ x'_3\ x'_6\ x_7\ x'_4\ Y_7\ a_4;$
$a_2\ x'_1\ x'_3\ x'_6\ x'_7\ Y_5\ a_5.$

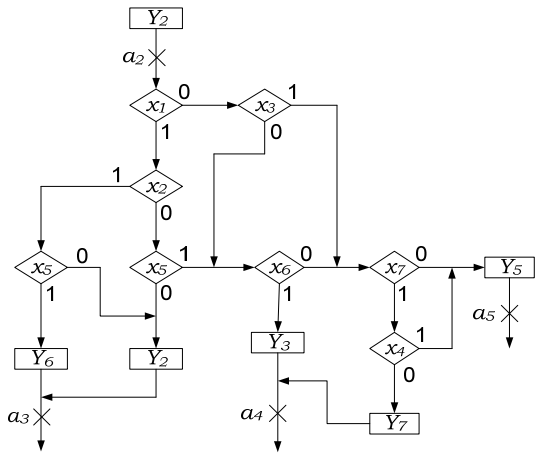

Figure 9. Subgraph of ASM

4.2.5 Transition tables of Mealy FSM. The graph of Mealy FSM in Fig. 8 has only 6 states and 16 arcs. Practically, however, we must construct FSMs with tens of states and more than one-two hundreds of transitions. In such a case, it is difficult to use a graph, so we will present it as a table. Table 1 for the same Mealy FSM has five columns:

- a_m – a current state;
- a_s – a next state;
- $X(a_m,a_s)$ – an input signal;
- $Y(a_m,a_s)$ – an output signal;
- H – a number of line.

Actually, immediately from ASM, we should write transition paths, one after another, into the transition table. In Table 1, $\sim x_t$ is used instead of x'_t for the inversion of x_t.

Now we will discuss what kind of FSM we have received. Our ASM G_1 in Fig. 6 which we used to construct FSM S_1 in Table 1, has seven logical conditions and ten microoperations. FSM S_1 has seven binary inputs in the column $X(a_m,a_s)$ and ten binary outputs in the column $Y(a_m,a_s)$. The input signal of this FSM (Fig. 10) is the 7-component vector, the output signal of this FSM is the 10-component vector.

Table 1. Direct transition table of Mealy FSM S_1

a_m	a_s	$X(a_m,a_s)$	$Y(a_m,a_s)$	H
a_1	a_2	$x_1x_2x_3$	y_1y_3	1
	a_3	$x_1x_2\sim x_3$	y_6y_7	2
	a_2	$x_1\sim x_2$	y_1y_2	3
	a_4	$\sim x_1$	y_4	4
a_2	a_2	x_4x_1	y_8y_9	5
	a_6	$x_4\sim x_1$	y_3y_4	6
	a_4	$\sim x_4$	y_4	7
a_3	a_6	1	y_3y_4	8
a_4	a_5	x_5	$y_5y_6y_7$	9
	a_2	$\sim x_5x_1$	y_8y_9	10
	a_6	$\sim x_5\sim x_1$	y_3y_4	11
a_5	a_2	x_6	y_8y_9	12
	a_1	$\sim x_6x_7$	$y_3y_6y_{10}$	13
	a_1	$\sim x_6\sim x_7$	-	14
a_6	a_1	x_6	y_6y_7	15
	a_6	$\sim x_6$	y_3y_4	16

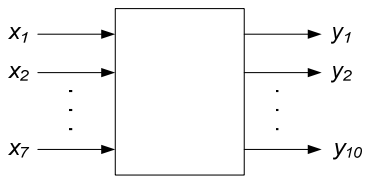

Figure 10. FSM as a black box

Let us take one of the rows from Table 1, for example row 3, and look at the behavior of FSM presented in this row. Our FSM transits from state a_1 into state a_2 when the product $x_1 x'_2 = 1$. It is clear that such a transition takes place for any input vector in which the first component is equal to 1, the second component is equal to 0. The values of other components are not important. Thus, we can say that the third row of Table 1 presents transitions from a_1 with any vector which is covered by cube $10xxxxx$. In other words, this row presents not one but $2^5 = 32$ transitions. In exactly the same way, the first and the second row present 16 transitions, the fourth row – 64 transitions and the eighth row – 128 transitions.

Two microoperations y_1, y_2, written in the third row of the output column, mean that two components y_1 and y_2 are equal to 1 and others are equal to 0 ($y_1 = y_2 = 1$; $y_3 = y_4 = \ldots = y_{10} = 0$) in the output vector. I remind you that if the operator, written in the operator vertex of some ASM, contains microoperations y_m, y_n, only these microoperations are equal to 1 and other microoperations are equal to 0 during implementation of this operator.

Let us compare Table 1 with a classical FSM representation in Table 3.7 from Chapter 3. If we would like to present our FSM with six states a_1, …, a_6 and seven inputs x_1, …, x_7 in the classical table, this table will have about $6x2^7$ rows, because each row of

this table describes only one FSM transition. In our Table 1 from this Chapter, we have only 16 rows because each row of such table presents lot of transitions.

The specific feature of such FSM is the multiplicity of inputs in the column $X(a_m,a_s)$, maybe several tens or even hundreds, but each product in one row contains only few variables from the whole set of input variables – as a rule, not more than 8 – 10 variables. It means that each time the values of the output variables depend only on the values of a small number of the input variables. Really, if, for example, FSM has 30 input variables, the total number of input vectors is equal to 2^{30}, and if each time the values of the output variables depended on the values of all the input variables, no designer could either describe or construct such an FSM.

Let us briefly discuss the correspondence between FSM S_1 (Table 1) and ASM G_1 (Fig. 6) which we used to construct FSM S_1. In Section 4.1.3 we got the value of ASM G_1

$$Y_b\, Y_1\, Y_2\, Y_4\, Y_2\, Y_3\, Y_8\, Y_e$$

for some random sequence of vectors (2) of logical conditions:

		x_1	x_2	x_3	x_4	x_5	x_6	x_7
Δ_1	=	1	0	1	0	1	1	1
Δ_2	=	0	1	1	0	1	0	0
Δ_3	=	1	0	1	0	0	1	0
Δ_4	=	0	1	0	0	0	0	1
Δ_5	=	1	1	0	1	1	1	0
Δ_6	=	1	1	0	0	1	0	1
Δ_7	=	0	1	1	1	0	0	0
Δ_8	=	0	1	0	1	0	0	1

Now we will find the response of FSM S_1 in the initial state a_1 to the same sequence of input vectors:

State sequence	a_1	a_2	a_4	a_2	a_4	a_5	a_1
Input sequence	Δ_1	Δ_2	Δ_3	Δ_4	Δ_5	Δ_6	
Response	y_1y_2	y_4	y_8y_9	y_4	$y_5y_6y_7$	$y_3y_6y_{10}$	(4)
Microinstructions	Y_1	Y_2	Y_4	Y_2	Y_3	Y_8	

Let FSM be in the initial state a_1 with the first vector $\Delta_1 = 1010111$ at its input. To determine the next state and the output we should find such a row in the array of transitions from a_1 (Table 1) that the product $X(a_m,a_s)$, written in this row, be equal to one at input vector Δ_1. Since $x_1x'_2(\Delta_1) = 1$ (the third row), FSM S_1 produces output signal $y_1y_2 = Y_1$ and transits into state a_2. Similarly, we find that $x'_4(\Delta_2)$ is equal to one at one of transitions from state a_2 and FSM transits to the state a_4 with the output signal $y_4 = Y_2$ (see row 7 in Table 1) etc. As a result, we get the response of FSM S_1 in the initial state a_1 to the input sequence Δ_1, …, Δ_6 in the fourth row of sequence (4).

As seen from this row, the FSM response is equal to the value of ASM G_1 for the same input sequence. Note, that we consider here only the FSM response until its return to the initial state a_1 and this response $Y_1\, Y_2\, Y_4\, Y_2\, Y_3\, Y_8$ corresponds to the value of ASM G_1 between the operator Y_b (vertex "Begin") and the operator Y_e (vertex "End").

Let us define *FSM S as implementing ASM G* if the response of this FSM in the state a_1 to any input sequence (until its return to the state a_1) is equal to the value of ASM G

for the same input sequence. From the considered method of synthesis of Mealy FSM S_1 from ASM G_1 it follows that this FSM S_1 implements ASM G_1.

4.2.6 Synthesis of Mealy FSM logic circuit. As in Chapter 3, we will construct a Mealy FSM logic circuit with the structure presented in Fig. 11. To design this circuit we will use an FSM structure table (Table 2). This table was constructed from the direct transition table (Table 1) by adding three additional columns:

- $K(a_m)$ – a code of the current state;
- $K(a_s)$ – a code of the next state;
- $D(a_m,a_s)$ – an input memory function.

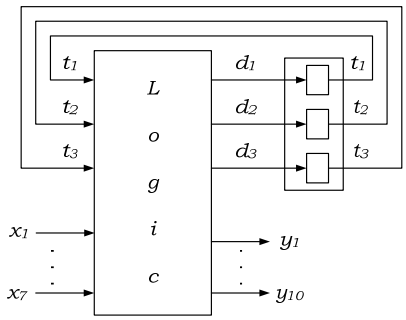

Figure 11. The structure for the Mealy FSM logic circuit

Table 2. Structure table of FSM S_1

a_m	$K(a_m)$	a_s	$K(a_s)$	$X(a_m,a_s)$	$Y(a_m,a_s)$	$D(a_m,a_s)$	H
a_1	001	a_2	000	$x_1x_2x_3$	y_1y_3	–	1
		a_3	101	$x_1x_2\sim x_3$	y_6y_7	d_1d_3	2
		a_2	000	$x_1\sim x_2$	y_1y_2	–	3
		a_4	010	$\sim x_1$	y_4	d_2	4
a_2	000	a_2	000	x_4x_1	y_8y_9	–	5
		a_6	100	$x_4\sim x_1$	y_3y_4	d_1	6
		a_4	010	$\sim x_4$	y_4	d_2	7
a_3	101	a_6	100	1	y_3y_4	d_1	8
a_4	010	a_5	110	x_5	$y_5y_6y_7$	d_1d_2	9
		a_2	000	$\sim x_5x_1$	y_8y_9	–	10
		a_6	100	$\sim x_5\sim x_1$	y_3y_4	d_1	11
a_5	110	a_2	000	x_6	y_8y_9	–	12
		a_1	001	$\sim x_6x_7$	$y_3y_6y_{10}$	d_3	13
		a_1	001	$\sim x_6\sim x_7$	-	d_3	14
a_6	100	a_1	001	x_6	y_6y_7	d_3	15
		a_6	100	$\sim x_6$	y_3y_4	d_1	16

To encode FSM states we constructed Table 3 where $p(a_s)$ is the number of appearances of each state in the next state column a_s in Table 2. The algorithm for state assignment is absolutely the same as in Chapter 3. First, we use the zero code for state a_2 with max $p(a_2) = 5$. Then codes with one '1' are used for states a_6, a_1, a_4 with the next max appearances and, finally, two codes with two 'ones' are used for the left states a_3 and a_5.

To fill column $D(a_m,a_s)$ it is sufficient to write there column $K(a_s)$ because the input of D flip-flop is equal to its next state. However, here we use the same notation as in column $Y(a_m,a_s)$ and write d_r in the column $D(a_m,a_s)$ if d_r is equal to 1 at the

corresponding transition (a_m,a_s) – equal to 1 in column $K(a_s)$. After that, the shaded part of Table 2 is something like a truth table with input variables $t_1, t_2, t_3, x_1, \ldots, x_7$ in the columns $K(a_m)$ and $X(a_m,a_s)$ and output variables (functions) $y_1, \ldots, y_{10}, d_1, d_2, d_3$ in the columns $Y(a_m,a_s)$ and $D(a_m,a_s)$.

Table 3. State assignment

a_s	$p(a_s)$	$t_1\ t_2\ t_3$
a_1	3	0 0 1
a_2	5	0 0 0
a_3	1	1 0 1
a_4	2	0 1 0
a_5	1	1 1 0
a_6	4	1 0 0

Let A_m be a product, corresponding to the state code $K(a_m)$, and X_h be the product of input variables, written in the column $X(a_m,a_s)$ in the h row. For example, from the column $K(a_m)$: $K(a_1) = 001$, then $A_1 = t'_1t'_2t_3$; $K(a_2) = 000$, then $A_2 = t'_1t'_2t'_3$; $K(a_3) = 101$, then $A_3 = t_1t'_2t_3$ etc. Immediately from the column $X(a_m,a_s)$ we get:

$$X_1 = x_1x_2x_3;\ X_2 = x_1x_2x'_3;\ X_6 = x_4x'_1;\ X_8 = 1;\ X_{16} = x'_6.$$

We call the term

$$e_h = A_m X_h$$

the product corresponding to the h row of the FSM structure table if a_m is the current state in this row. For example, from Table 2, we get:

$$\begin{aligned} e_1 &= t'_1t'_2t_3\ x_1x_2x_3; \\ e_2 &= t'_1t'_2t_3\ x_1x_2x'_3; \\ e_6 &= t'_1t'_2t'_3\ x_4x'_1; \\ e_8 &= t_1t'_2t_3; \\ e_{16} &= t_1t'_2t'_3\ x'_6. \end{aligned}$$

Let $H(y_n)$ is the set of rows with y_n in the column $Y(a_m,a_s)$. Then, as in the truth table:

$$y_n = \sum_{h \in H(y_n)} e_h.$$

For example, y_6 is written in rows 2, 9, 13, 15 in the column $Y(a_m,a_s)$. Then

$$y_6 = e_2 + e_9 + e_{13} + e_{15} = t'_1t'_2t_3x_1x_2x'_3 + t'_1t_2t'_3\ x_5 + t_1t_2t'_3\ x'_6x_7 + t_1t'_2t'_3\ x_6.$$

In exactly the same way, if $H(d_r)$ is the set of rows with d_r in the column $D(a_m,a_s)$, then

$$d_r = \sum_{h \in H(d_r)} e_h.$$

For example, d_2 is written in rows 4, 7, 9 in the column $D(a_m,a_s)$. Then

$$d_2 = e_4 + e_7 + e_9 = t'_1t'_2t_3\ x'_1 + t'_1t'_2t'_3\ x'_4 + t'_1t_2t'_3\ x_5.$$

Thus, immediately from Table 1 we can get expressions for outputs of circuit *"Logic"* in Fig. 11:

$$y_1 = e_1 + e_3 = t'_1t'_2t_3\ x_1x_2x_3 + t'_1t'_2t_3\ x_1x'_2;$$

$y_2 = e_3 = t'_1t'_2t_3\, x_1x'_2;$

$y_3 = e_1 + e_6 + e_8 + e_{11} + e_{13} + e_{16} = t'_1t'_2t_3\, x_1x_2x_3 + t'_1t'_2t'_3\, x_4x'_1 + t_1t'_2t_3 +$
$+ t'_1t_2t'_3\, x'_5x'_1 + t_1t_2t'_3\, x'_6x_7 + t_1t'_2t'_3\, x'_6;$
. . .
$y_{10} = e_{13} = t_1t_2t'_3\, x'_6x_7;$

$d_1 = e_2 + e_6 + e_8 + e_9 + e_{11} + e_{16} = t'_1t'_2t_3\, x_1x_2x'_3 + t'_1t'_2t'_3\, x_4x'_1 + t_1t'_2t_3 +$
$+ t'_1t_2t'_3\, x_5 + t'_1t_2t'_3\, x'_5x'_1 + t_1t'_2t'_3\, x'_6;$
$d_2 = e_4 + e_7 + e_9 = t'_1t'_2t_3\, x'_1 + t'_1t'_2t'_3\, x'_4 + t'_1t_2t'_3\, x_5;$
$d_3 = e_2 + e_{13} + e_{14} + e_{15} = t'_1t'_2t_3\, x_1x_2x'_3 + t_1t_2t'_3\, x'_6x_7 + t_1t_2t'_3\, x'_6x'_7 + t_1t'_2t'_3\, x_6.$

How many *different products* are there in these expressions? The answer is very simple – only sixteen, because we have 16 rows in Table 2 and only one product corresponds to one row. Thus, we should not write any expressions but can design the logic circuit immediately from the structure table. For that, it is sufficient to construct H AND-gates, one for each row, and $N+R$ OR-gates, one for each output variable y_n ($n = 1, \ldots, 10$ in our example) and one for each input memory function d_r ($r = 1, 2, 3$ in our example). The logic circuit of Mealy FSM is shown in Fig. 12. We have constructed 16 AND-gates, as there are 16 rows in its structure table. The number of OR-gates in this circuit is less than the number of input memory functions and output functions. Really, if y_n or d_r (y_2 and y_{10} in our example) are written only in one row of the structure table, it is not necessary to construct OR-gate for such y_n or d_r, we can get these signals from the corresponding AND-gates. Moreover, we have constructed one OR-gate for y_8 and y_9 since these outputs are always together in the structure table of Mealy FSM S_1.

4.2.7 ASM with waiting vertices. In this section, we will show that the algorithm for FSM synthesis does not change if ASM contains waiting vertices. In a waiting vertex, one of its outputs is connected with its input (see the ASM subgraph in Fig. 13). Let us find all transition paths from the state a_8. The first two are trivial – see the first two rows in Table 4.

To find the next path we should invert the variable x_7. The output *'0'* for x_7 brings us to the input of this conditional vertex. So, the next paths will be:

$$a_8 \sim x_7\, x_7\, x_{12}\, (y_{11})\, a_{13};$$
$$a_8 \sim x_7\, x_7\, \sim x_{12}\, (y_{23}, y_{29})\, a_{17}.$$

The products of input variables for both of these paths are equal to zero ($x'_7\, x_7 = 0$), so FSM cannot transit from the state a_8 to any other state when $x_7 = 0$. If FSM cannot transit into any other state, it remains in the same state a_8 or, we can say, it transits from a_8 to a_8 with $X(a_8, a_8) = x'_7$. No output variables are equal to *'1'* at this transition, so we have '–' in the column $Y(a_m, a_s)$ in the third row.

The next example (Fig.14) presents a general case. The only difference from the previous example – the waiting vertex is in the middle of the path. After the third path in Table 5 we should invert variable x_{11} and again return to the input of the conditional vertex with x_{11}. We can construct the following transitions paths:

$$a_{10} \sim x_4\, \sim x_{11}\, x_{11}\, x_{27}\, (y_{33})\, a_{22};$$
$$a_{10} \sim x_4\, \sim x_{11}\, x_{11}\, \sim x_{27}\, (y_7, y_{31})\, a_{17}.$$

The products for both of these paths are equal to zero. So, when $x_4 = 0$, we reached a waiting vertex with condition x_{11}. If $x_{11} = 0$ (return to the input), FSM transits from state a_{10} to state a_{10} (remains in this state) with input $x'_4 x'_{11}$ and each output variable is equal to zero (the forth row in Table 5).

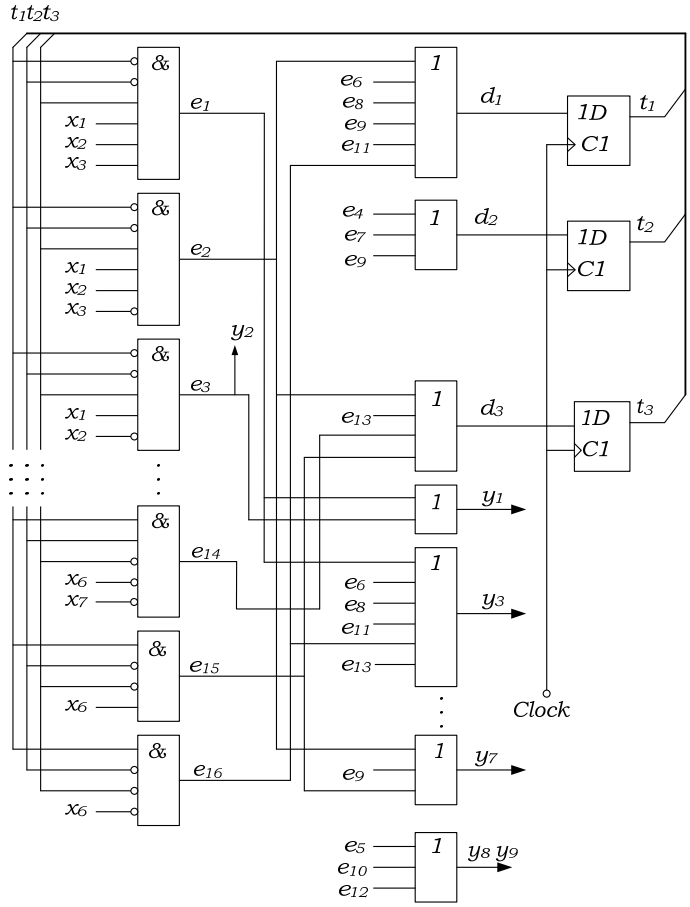

Figure 12. The logic circuit for Mealy FSM S_1

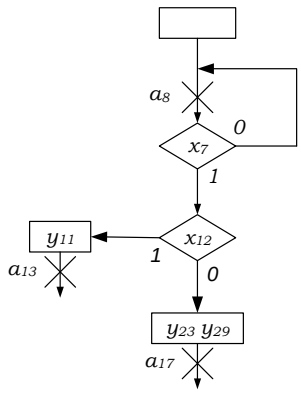

Table 4. Transitions for subgraph G_1

a_m	a_s	$X(a_m,a_s)$	$Y(a_m,a_s)$	H
		. . .		
a_8	a_{13} a_{17} a_8	x_7x_{12} $x_7 \sim x_{12}$ $\sim x_7$	y_{11} $y_{23}y_{29}$ –	

Figure 13. Subgraph G_1 with waiting vertex

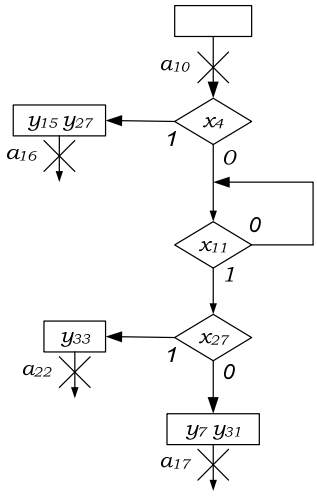

Figure 14. Subgraph G_2 with a waiting vertex

Table 5. Transitions for subgraph G_2

a_m	a_s	$X(a_m,a_s)$	$Y(a_m,a_s)$	H
		. . .		
a_{10}	a_{16}	x_4	$y_{15}y_{27}$	
	a_{22}	$\sim x_4 x_{11} x_{27}$	y_{33}	
	a_{17}	$\sim x_4 x_{11} \sim x_{27}$	$y_7 y_{31}$	
	a_{10}	$\sim x_4 \sim x_{11}$	–	

4.3 Synthesis of Moore FSM

As an example, we will use ASM G_1 in Fig. 15. A Moore FSM, implementing given ASM, can be constructed in two stages:

Stage 1. Construction of a marked ASM;
Stage 2. Construction of an FSM transition table.

At the first stage, the vertices "Begin", "End" and operator vertices are marked by symbols $a_1, a_2, \ldots, a_M$ as follows:

1. Vertices "Begin" and "End" are marked by the same symbol a_1;
2. Operator vertices are marked by different symbols $a_2, \ldots, a_M$;
3. All operator vertices should be marked.

Thus, while synthesizing a Moore FSM, symbols of states do not mark inputs of vertices following the operator vertices (as in the Mealy FSM) but operator vertices themselves. The number of marks is $T+1$, where T is the number of operator vertices in the marked ASM. In our example (Fig. 15), we need marks $a_1, \ldots, a_{10}$ for ASM G_1.

We will find the following transition paths in the marked ASM:

$$a_m \tilde{x}_{m1} \ldots \tilde{x}_{mR_m} a_s .$$

Thus, the transition path is the path between two operator vertices, containing R_m conditional vertices. Here, as above in the case of Mealy FSM, $\tilde{x}_{mr} = x_{mr}$, if in the transition path, we leave the conditional vertex with x_{mr} via output '1' and $\tilde{x}_{mr} = x'_{mr}$ if we leave the vertex with x_{mr} via output '0'. If $R_m = 0$ in such a path, there are no conditional vertices between two operator vertices, and this path turns into $a_m a_s$.

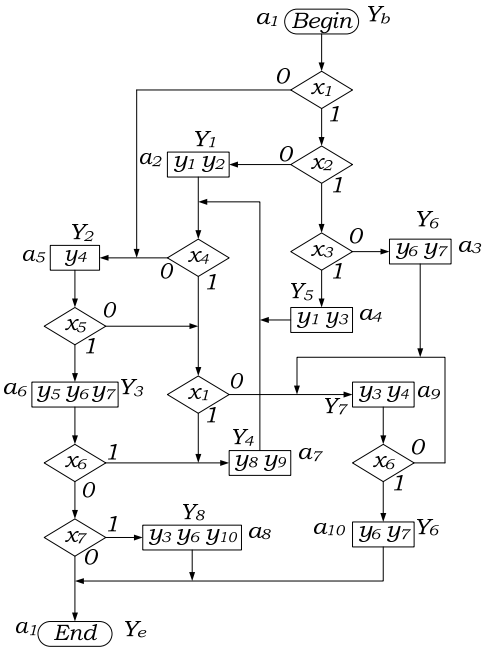

Figure 15. ASM G_1 marked for the Moore FSM synthesis

At the second stage we construct a transition table (or the state diagram) of the Moore FSM with states (marks) $a_1, \ldots, a_M$, obtained at the first stage. We have ten such states $a_1, \ldots, a_{10}$ in our example. Thus, the FSM contains as many states as the number of marks we get at the previous stage. Now we should define transitions between these states.

Thus, a Moore FSM has a transition from state a_m to state a_s with input $X(a_m, a_s)$ (see the upper subgraph in Fig. 16) if, in ASM, there is a transition path $a_m \tilde{x}_{m1} \ldots \tilde{x}_{mR_m} a_s$. Here $X(a_m, a_s)$ is a product of logical conditions written in this path: $X(a_m, a_s) = \tilde{x}_{m1} \ldots \tilde{x}_{mR_m}$. In exactly the same way, for the path $a_m a_s$ (see the lower subgraph in Fig. 16) we have a transition from state a_m to state a_s with input $X(a_m, a_s) = 1$, because the product of an empty set of variables is equal to zero. If a_m marks the operator vertex with operator Y_t, then $\lambda(a_m) = Y_t$, i.e. we identify the operator Y_t written in the operator vertex with this state a_m.

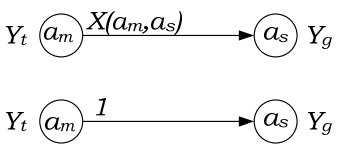

Figure 16. Subgraphs to illustrate transitions in the Moore FSM

The transition table for Moore FSM S_2, thus constructed, is presented in Table 6. The outputs are written in column $Y(a_m)$ immediately after the column with the current states. To design the logic circuit for this FSM we will use the structure presented in Fig. 17. It consists of two logic blocks (*Logic1* and *Logic2*) and memory block with four *D* flip-flops. *Logic1* implements input memory functions, depending on flip-flop

outputs $t_1, \ldots, t_4$ (feedback) and input variables $x_1, \ldots, x_7$. *Logic2* implements output functions, depending only on flip-flop outputs $t_1, \ldots, t_4$.

Table 6. The transition table of Moore FSM S_2

a_m	$Y(a_m)$	a_s	$X(a_m, a_s)$	h
a_1	–	a_4	$x_1x_2x_3$	1
		a_3	$x_1x_2{\sim}x_3$	2
		a_2	$x_1{\sim}x_2$	3
		a_5	$\sim x_1$	4
a_2	y_1y_2	a_7	x_4x_1	5
		a_9	$x_4{\sim}x_1$	6
		a_5	$\sim x_4$	7
a_3	y_6y_7	a_9	1	8
a_4	y_1y_3	a_7	x_4x_1	9
		a_9	$x_4{\sim}x_1$	10
		a_5	$\sim x_4$	11
a_5	y_4	a_6	x_5	12
		a_7	$\sim x_5x_1$	13
		a_9	$\sim x_5{\sim}x_1$	14
a_6	$y_5y_6y_7$	a_7	x_6	15
		a_8	$\sim x_6x_7$	16
		a_1	$\sim x_6{\sim}x_7$	17
a_7	y_8y_9	a_7	x_4x_1	18
		a_9	$x_4{\sim}x_1$	19
		a_5	$\sim x_4$	20
a_8	$y_3y_6y_{10}$	a_1	1	21
a_9	y_3y_4	a_{10}	x_6	22
		a_9	$\sim x_6$	23
a_{10}	y_6y_7	a_1	1	24

To encode FSM states we constructed Table 7 where $p(a_s)$, as before, is the number of appearances of each state in the next state column a_s in Table 6. The algorithm for state assignment is absolutely the same as in the case of Mealy FSM. First, we use the zero code for state a_9 with max $p(a_9) = 6$. Then codes with one '1' are used for states a_7, a_5, a_1 and a_2 with the next max appearences and, finally, five codes with two 'ones' are used for the left five states a_3, a_4, a_6, a_8 and a_{10}.

Table 8 is the structure table of the Moore FSM S_2. Its logic circuit is constructed in Fig. 18. In this circuit, A_m is a product of state variables for the state a_m ($m = 1, \ldots, 10$). As above we construct one AND-gate for one row of the structure table, but we need not construct the gates for rows 6, 8, 10, 14, 19 and 23, as all input memory functions are equal to zero in these rows (see the column $D(a_m,a_s)$ in Table 8). Neither

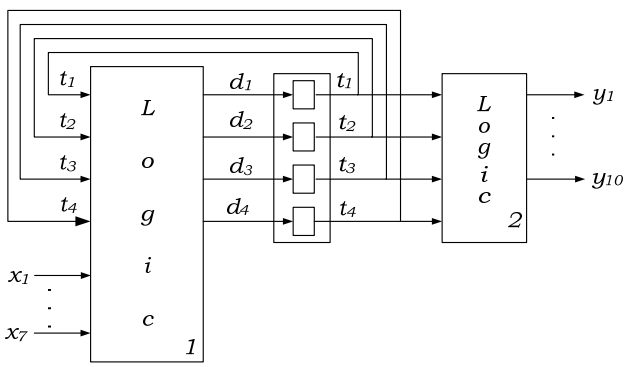

Figure 17. Moore FSM structure

Table 7. State assignment

a_s	$p(a_s)$	$t_1t_2t_3t_4$
a_1	3	0100
a_2	1	0010
a_3	1	1001
a_4	1	0110
a_5	4	0001
a_6	1	1100
a_7	5	1000
a_8	1	0011
a_9	6	0000
a_{10}	1	0101

Table 8. The structure table of the Moore FSM S_2

a_m	$Y(a_m)$	$K(a_m)$	a_s	$K(a_s)$	$X(a_m,a_s)$	$D(a_m,a_s)$	h
a_1	–	0100	a_4	0110	$x_1x_2x_3$	d_2d_3	1
			a_3	1001	$x_1x_2\sim x_3$	d_1d_4	2
			a_2	0010	$x_1\sim x_2$	d_3	3
			a_5	0001	$\sim x_1$	d_4	4
a_2	y_1y_2	0010	a_7	1000	x_4x_1	d_1	5
			a_9	0000	$x_4\sim x_1$	–	6
			a_5	0001	$\sim x_4$	d_4	7
a_3	y_6y_7	1001	a_9	0000	1	–	8
a_4	y_1y_3	0110	a_7	1000	x_4x_1	d_1	9
			a_9	0000	$x_4\sim x_1$	–	10
			a_5	0001	$\sim x_4$	d_4	11
a_5	y_4	0001	a_6	1100	x_5	d_1d_2	12
			a_7	1000	$\sim x_5x_1$	d_1	13
			a_9	0000	$\sim x_5\sim x_1$	–	14
a_6	$y_5y_6y_7$	1100	a_7	1000	x_6	d_1	15
			a_8	0100	$\sim x_6x_7$	d_2	16
			a_1	0011	$\sim x_6\sim x_7$	d_3d_4	17
a_7	y_8y_9	1000	a_7	1000	x_4x_1	d_1	18
			a_9	0000	$x_4\sim x_1$	–	19
			a_5	0001	$\sim x_4$	d_4	20
a_8	$y_3y_6y_{10}$	0011	a_1	0100	1	d_2	21
a_9	y_3y_4	0000	a_{10}	0101	x_6	d_2d_4	22
			a_9	0000	$\sim x_6$	–	23
a_{10}	y_6y_7	0101	a_1	0100	1	d_2	24

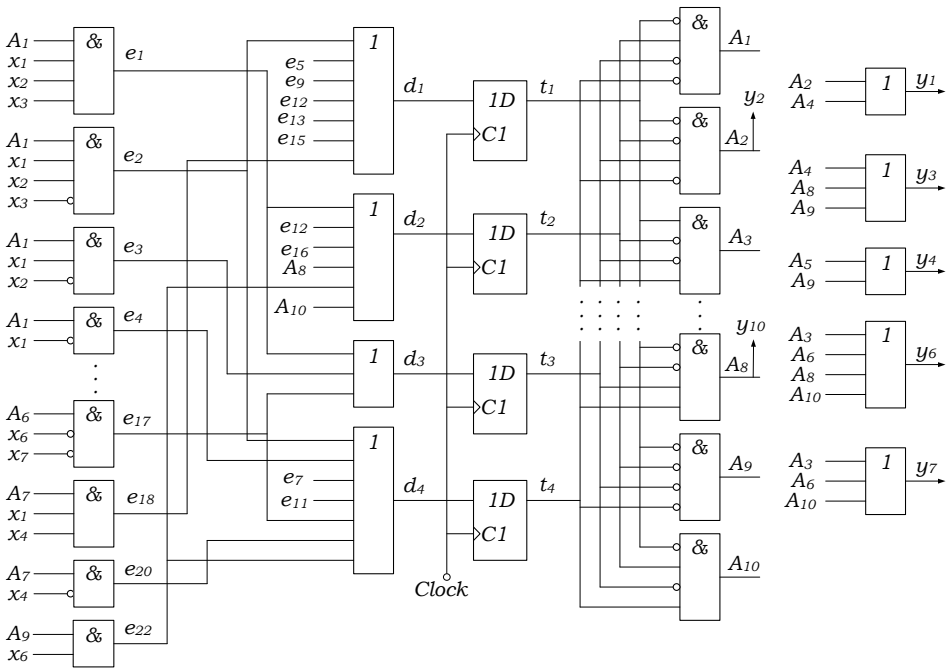

Figure 18. The logical circuit for the Moore FSM S_2

do we construct the gates for rows 21 and 24, since there are no input variables in the corresponding terms e_{21} and e_{24}: $e_{21} = A_8$ and $e_{24} = A_{10}$ and we use A_8 and A_{10} directly as inputs in OR-gate for d_2.

4.4. Synthesis of Combined FSM model

In this book we will use two kinds of transition tables – direct and reverse. In a *direct table* (Table 9), transitions are ordered according to the current state (the first column in this table) – first we write all transitions *from* the state a_1, then *from* the state a_2 , etc. In a *reverse table* (Table 10), transitions are ordered according to the next state (the second column in this table) – first we write all transitions *to* the state a_1, then *to* the state a_2 , etc.

Table 9. Direct transition table of Mealy FSM S_3

a_m	a_s	$X(a_m,a_s)$	$Y(a_m,a_s)$	h
a1	a2	x6	y8y9	1
a1	a5	~x6*x7	y6	2
a1	a5	~x6*~x7	y3y6y10	3
a2	a2	x4*x1	y1y2	4
a2	a6	x4*~x1	y3y4	5
a2	a4	~x4	y4	6
a3	a6	1	y3y5	7
a4	a1	x5	--	8
a4	a2	~x5*x1	y8y9	9
a4	a6	~x5*~x1	y3y4	10
a5	a2	x1*x2*x3	y1y3	11
a5	a3	x1*x2*~x3	y1y4	12
a5	a2	x1*~x2	y1y2	13
a5	a4	~x1	y4	14
a6	a5	x6	y6y7	15
a6	a6	~x6	y3y5	16

Now we will discuss the transformation of Mealy FSM into Combined FSM and synthesis of its logic circuit. I remind here that Combined FSM has two kinds of output signals:

1. Signals depending on the current state and the current input (as in the Mealy model);
2. Signals depending only on the current state (as in the Moore model);

As an example, we use the transition table of Mealy FSM in Table 9. Our first step is to construct a reverse table for this FSM (Table 10).

Fig. 19,a illustrates all transitions into state a_5 of Mealy FSM from Table 10. Here we have three transitions with different outputs but all of them contain the same output variable y_6. So, we can identify this output variable y_6 with the state a_5 as a Moore signal (see Fig. 19,b).

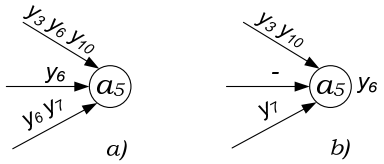

Figure 19. Transformation from Mealy FSM to Combined FSM

Table 10. Reverse transition table of Mealy FSM S_3

a_m	a_s	$X(a_m,a_s)$	$Y(a_m,a_s)$	H
a4	a1	x5	--	1
a2	a2	x4*x1	y1y2	2
a1	a2	x6	y8y9	3
a4	a2	~x5*x1	y8y9	4
a5	a2	x1*x2*x3	y1y3	5
a5	a2	x1*~x2	y1y2	6
a5	a3	x1*x2*~x3	y1y4	7
a2	a4	~x4	y4	8
a5	a4	~x1	y4	9
a1	a5	~x6*x7	y6	10
a1	a5	~x6*~x7	y3y6y10	11
a6	a5	x6	y6y7	12
a4	a6	~x5*~x1	y3y4	13
a2	a6	x4*~x1	y3y4	14
a3	a6	1	y3y5	15
a6	a6	~x6	y3y5	16

After this, the transformation of Mealy FSM into Combined model is trivial. Let us return to the reverse Table 10 and begin to construct the reverse transition table of Combined FSM S_4 (Table 11 In Table10, we look at the transitions to each state, beginning from transitions to state a_1. Let Y_s be the set of output variables at the transitions into state a_s ($Y_5 = \{y_3, y_6, y_7, y_{10}\}$ in Fig19,a or in Table 10) and Y_s^{Moore} be the subset of common output variables at all transitions into a_s ($Y_5^{Moore} = \{y_6\}$ in Fig19,a or in Table 10). Then, in Table 11, we delete Y_s^{Moore} from the column $Y(a_m,a_s)$ at each row with transition to a_s and write Y_s^{Moore} next to a_s in the column $Y(a_s)$. In our example:

$Y_1^{Moore} = Y_2^{Moore} = \varnothing$; $Y_3^{Moore} = \{y_1, y_4\}$; $Y_4^{Moore} = \{y_4\}$;
$Y_5^{Moore} = \{y_6\}$; $Y_6^{Moore} = \{y_3\}$.

Table 11. Reverse transition table of Combined FSM S_4

a_m	a_s	$Y(a_s)$	$X(a_m,a_s)$	$Y(a_m,a_s)$	H
a4	a1	--	x5	--	1
a1	a2	--	x6	y8y9	2
a2	a2	--	x4*x1	y1y2	3
a4	a2	--	~x5*x1	y8y9	4
a5	a2	--	x1*x2*x3	y1y3	5
a5	a2	--	x1*~x2	y1y2	6
a5	a3	y1y4	x1*x2*~x3	--	7
a2	a4	y4	~x4	--	8
a5	a4	y4	~x1	--	9
a1	a5	y6	~x6*~x7	y3y10	10
a1	a5	y6	~x6*x7	--	11
a6	a5	y6	x6	y7	12
a2	a6	y3	x4*~x1	y4	13
a3	a6	y3	1	y5	14
a4	a6	y3	~x5*~x1	y4	15
a6	a6	y3	~x6	y5	16

Now we consider the design of the logic circuit of Combined FSM. For this, let us return to the Mealy FSM S_1 with direct transition Table 1. Its reverse transition table is presented in Table 12. Immediately from this table we construct the direct transition table of Combined FSM S_1 (Table 13). To construct the logic circuit for this FSM we should encode the states and construct FSM structure table. But before state assignment we will make one more step.

Table 12. Reverse transition table of Mealy FSM S_1

a_m	a_s	$X(a_m,a_s)$	$Y(a_m,a_s)$	H
a_5	a_1	$\sim x_6x_7$	$y_3y_6y_{10}$	1
a_5		$\sim x_6\sim x_7$	-	2
a_6		x_6	y_6y_7	3
a_1	a_2	$x_1x_2x_3$	y_1y_3	4
a_1		$x_1\sim x_2$	y_1y_2	5
a_2		x_4x_1	y_8y_9	6
a_4		$\sim x_5x_1$	y_8y_9	7
a_5		x_6	y_8y_9	8
a_1	a_3	$x_1x_2\sim x_3$	y_6y_7	9
a_1	a_4	$\sim x_1$	y_4	10
a_2		$\sim x_4$	y_4	11
a_4	a_5	x_5	$y_5y_6y_7$	12
a_2	a_6	$x_4\sim x_1$	y_3y_4	13
a_3		1	y_3y_4	14
a_4		$\sim x_5\sim x_1$	y_3y_4	15
a_6		$\sim x_6$	y_3y_4	16

Unlike the transition table of the Mealy FSM, Table 13 contains many empty entries in the column $Y(a_m,a_s)$. It means that all output variables are equal to zero in these rows. If, after state assignment, we get an empty entry in column $D(a_m,a_s)$ for such a row, we shouldn't construct a product for this row, because all output variables and input memory functions are equal to zero in this row. Now we will try to maximize the number of such rows in the structure table of S_5.

Table 13. Direct transition table of Combined FSM S_5

a_m	$Y(a_m)$	a_s	$X(a_m,a_s)$	$Y(a_m,a_s)$	H
a_1	--	a_2	$x_1x_2x_3$	y_1y_3	1
	--	a_3	$x_1x_2\sim x_3$	-	2
	--	a_2	$x_1\sim x_2$	y_1y_2	3
	--	a_4	$\sim x_1$	-	4
a_2	--	a_2	x_4x_1	y_8y_9	5
	--	a_6	$x_4\sim x_1$	-	6
	--	a_4	$\sim x_4$	-	7
a_3	y_6y_7	a_6	1	-	8
a_4	y_4	a_5	x_5	-	9
		a_2	$\sim x_5x_1$	y_8y_9	10
		a_6	$\sim x_5\sim x_1$	--	11
a_5	$y_5y_6y_7$	a_2	x_6	y_8y_9	12
		a_1	$\sim x_6x_7$	$y_3y_6y_{10}$	13
		a_1	$\sim x_6\sim x_7$	-	14
a_6	y_3y_4	a_1	x_6	y_6y_7	15
		a_6	$\sim x_6$	-	16

Table 13 contains one row with empty entry in the column $Y(a_m,a_s)$ for the next states a_1 (row 14) and a_3 (row 2), two rows for a_4 (rows 4 and 7), one row for a_5 (row 9) and

four rows for a_6 (rows 6, 8, 11 and 16). This information is presented in the first two columns of Table 14, $z(a_s)$ is the number of empty entries in column $Y(a_m,a_s)$ for the next state a_s in Table 13. So, if we use zero code for states a_1 or a_3 or a_5, we shouldn't construct a product for one row ($z(a_1) = z(a_3) = z(a_5) = 1$), if we use zero code for state a_4 – for two rows ($z(a_4) = 2$); but if we use zero code for state a_6, we will construct four product less ($z(a_6) = 4$). Thus, we use code *000* for state a_6 with max $z(a_s)$. State assignment for other states is presented in Table 15. We have used here the same algorithm as we have used previously for Mealy and Moore models.

Table 14. Next states with zero outputs

a_s	$z(a_s)$	$t_1\ t_2\ t_3$
a_1	1	
a_3	1	
a_4	2	
a_5	1	
a_6	4	0 0 0

Table 15. State assignment

a_s	$p(a_s)$	$t_1\ t_2\ t_3$
a_1	3	0 1 0
a_2	5	0 0 1
a_3	1	1 0 1
a_4	2	1 0 0
a_5	1	1 1 0
a_6	4	0 0 0

Table 16. Structure table of Combined FSM S_5

a_m	$Y(a_m)$	$K(a_m)$	a_s	$K(a_s)$	$X(a_m,a_s)$	$Y(a_m,a_s)$	$D(a_m,a_s)$	H
a_1	--	010	a_2	001	$x_1x_2x_3$	y_1y_3	d_3	1
	--		a_3	101	$x_1x_2\sim x_3$	-	d_1d_3	2
	--		a_2	001	$x_1\sim x_2$	y_1y_2	d_3	3
	--		a_4	100	$\sim x_1$	-	d_1	4
a_2	--	001	a_2	001	x_4x_1	y_8y_9	d_3	5
	--		a_6	000	$x_4\sim x_1$	-	-	6
	--		a_4	100	$\sim x_4$	-	d_1	7
a_3	y_6y_7	101	a_6	000	1	-	-	8
a_4	y_4	100	a_5	110	x_5	-	d_1d_2	9
			a_2	001	$\sim x_5x_1$	y_8y_9	d_3	10
			a_6	000	$\sim x_5\sim x_1$	-	-	11
a_5	$y_5y_6y_7$	110	a_2	001	x_6	y_8y_9	d_3	12
			a_1	010	$\sim x_6x_7$	$y_3y_6y_{10}$	d_2	13
			a_1	010	$\sim x_6\sim x_7$	-	d_2	14
a_6	y_3y_4	000	a_1	010	x_6	y_6y_7	d_2	15
			a_6	000	$\sim x_6$	-	-	16

Table 16 is the structure table of Combined FSM S_5. We have three kinds of output variables here:

1. *Only Mealy signals:* y_1, y_2, y_8, y_9, y_{10}. They are written in column $Y(a_m,a_s)$ and are not written in column $Y(a_m)$ in Table 16;
2. *Only Moore signals:* y_4, y_5. They are written in the column $Y(a_m)$ and are not written in column $Y(a_m,a_s)$ in Table 16;
3. *Combined signals:* (both Mealy and Moore type) y_3, y_6, y_7. They are written in both columns $Y(a_m,a_s)$ and $Y(a_m)$ in Table 16.

The logic circuit of FSM S_5 is constructed in Fig. 20. In this circuit, A_m is a product of state variables for the state a_m ($m = 1, \ldots, 6$). The left part of this circuit, exactly as in the synthesis of the Mealy FSM logic circuit, implements input memory functions d_1, d_2, d_3 and *Mealy signals* y_1, y_2, y_8, y_9, y_{10}. As above, we construct one AND-gate for one row of the structure table, but we need not construct the gates for rows 6, 8, 11,

16 because all output variables and input memory functions are equal to zero in these rows in the columns $Y(a_m,a_s)$ and $D(a_m,a_s)$ in Table 16. As in the Mealy case, we do not construct OR gates for y_2 and y_{10} since they appear only once in the column $Y(a_m,a_s)$.

Moore signals y_4, y_5 are constructed as in the synthesis of Moore FSM logic circuit. Signal y_4 appears twice near the states a_4 and a_6 in the column $Y(a_m)$, so $y_4 = A_4 + A_6$ and we construct OR gate for this signal. Output signal y_5 appears only once in the column $Y(a_m)$ for the state a_5, so we get it straight from A_5: $y_5 = A_5$.

Combined signal y_6 is written in rows 13 and 15 in the column $Y(a_m,a_s)$ and near the states a_3 and a_5 in the column $Y(a_m)$, so

$$y_6 = e_{13} + e_{15} + A_3 + A_5.$$

Exactly in the same way

$$y_3 = e_1 + e_{13} + A_6; \quad y_7 = e_{15} + A_3 + A_5.$$

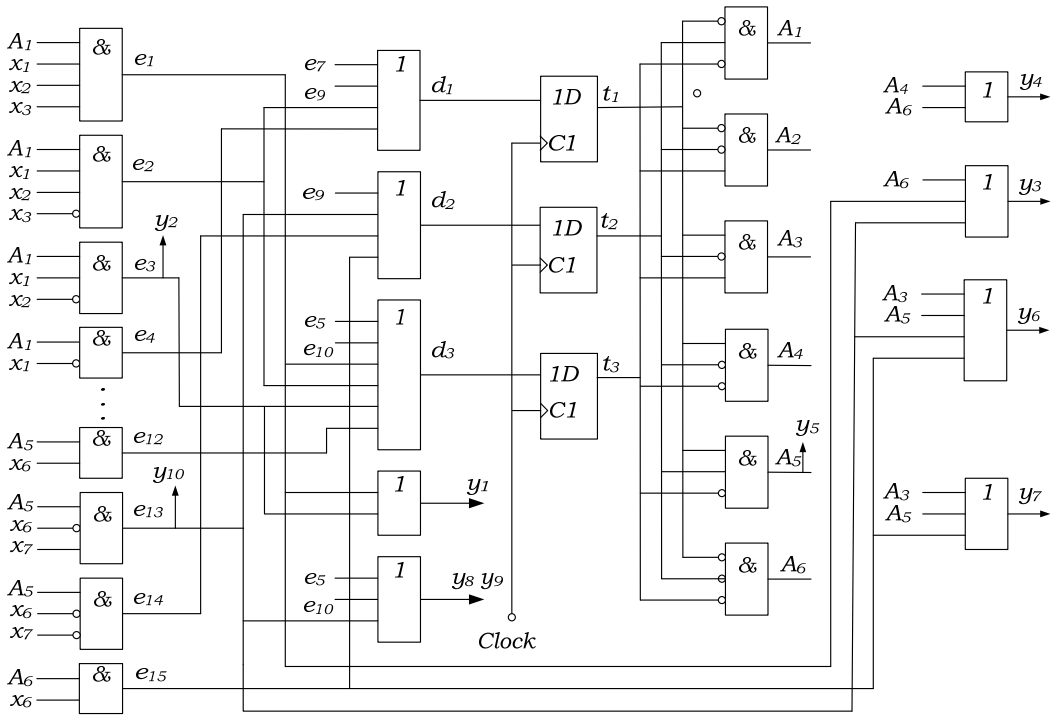

Figure 20. Logic circuit of Combined FSM S_5

4.5. FSM decomposition

In this section, we will discuss a very simple model for FSM decomposition. As an example, we use Mealy FSM S_6 (Table 17) and a partition π on the set of its states:

$$\pi = \{A_1, A_2, A_3\};$$
$$A_1 = \{a_2, a_3, a_9\}; A_2 = \{a_4, a_7, a_8\}; A_3 = \{a_1, a_5, a_6\}.$$

The number of component FSMs in the FSM network is equal to the number of blocks in partition π. Thus, in our example, we have three component FSMs S^1, S^2, S^3.

Let B^m is the set of states in the component FSM S^m. B^m contains the corresponding block of the partition π plus one additional state b_m. So, in our example:

S^1 has the set of states $B^1 = \{a_2, a_3, a_9, b_1\}$;
S^2 has the set of states $B^2 = \{a_4, a_7, a_8, b_2\}$;
S^3 has the set of states $B^3 = \{a_1, a_5, a_6, b_3\}$.

Table 17. Mealy FSM S_6

a_m	a_s	$X(a_m,a_s)$	$Y(a_m,a_s)$	H
a1	a3	x1*x2*x3	y1y2	1
a1	a6	x1*x2*~x3	y2y12	2
a1	a1	x1*~x2	y1y2	3
a1	a5	~x1	y1y2y12	4
a2	a2	x6	--	5
a2	a3	~x6	y3y5	6
a3	a3	x10	y3y5	7
a3	a9	~x10*x4	y10y15	8
a3	a8	~x10*~x4	y5y8y9	9
a4	a6	x7	y13	10
a4	a4	~x7*x9	y13y18	11
a4	a8	~x7*~x9	y13y14	12
a5	a6	x1	y16y17	13
a5	a5	~x1	y7y11	14
a6	a1	x5	y1y2	15
a6	a1	~x5	y16y17	16
a7	a2	x8	y14y18	17
a7	a4	~x8	y13y18	18
a8	a7	x9	y4y6	19
a8	a4	~x9	y6	20
a9	a9	x11*x6	y10y15	21
a9	a2	x11*~x6	y5y8y9	22
a9	a3	~x11	y3y8y9	23

To construct a transition table for each component FSM we should define the transitions between the states of these FSMs. For this, each transition between two states a_i and a_j of Mealy FSM S_6 from Table 17 should be implemented one after another as one or two transitions in component FSMs. There are two possible cases:

1. In Mealy FSM S_6, there is a transition between a_i and a_j (Fig. 21, left) and both of these states are in the same component FSM S^m. In such a case, we will have the same transition in this component FSM S^m (Fig. 21, right). It means that we must rewrite the corresponding row from the table of FSM S_6 into the table of component FSM S^m.

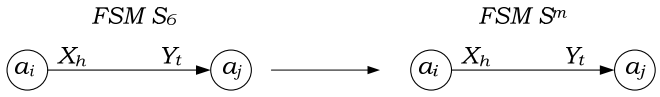

Figure 21. Two states a_i and a_j are in the same component FSM

2. Two states a_i and a_j are in different component FSMs (Fig. 22). Let a_i be in the component FSM S^m ($a_i \in B^m$) and a_j be in the component FSM S^p ($a_j \in B^p$). In such a case, one transition of FSM S_6 should be presented as two transitions – one in the component FSM S^m and one in the component FSM S^p:
   - FSM S^m transits from a_i into its additional state b_m with the same input X_h. At its output, we have the same output variables from set Y_t plus one additional output variable z_j, where index j is the index of state a_j in the component FSM S^p.
   - FSM S^p is in its additional state b_p. It transits from this state into state a_j with input signal z_j, that is an additional output variable in the component FSM S^m. The output at this transition is Y_0 – the signal with all output variables being equal to zero.

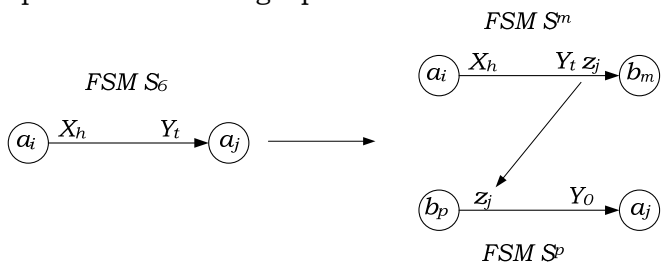

Figure 22. Two states a_i and a_j are in the different component FSMs

Thus, the procedure for FSM decomposition is reduced to:

a) Copying the row

$$a_i \quad a_j \quad X(a_i, a_j) \quad Y(a_i, a_j)$$

from the table of the decomposed FSM S to the table of the component FSM S^m if both states a_i and a_j are the states of S^m;

b) Replacing the row

$$a_i \quad a_j \quad X(a_i, a_j) \quad Y(a_i, a_j)$$

in the table of the decomposed FSM S by the row

$$a_i \quad b_m \quad X(a_i, a_j) \quad Y(a_i, a_j)\ z_j$$

in the table of the component FSM S^m, and by the row

$$b_p \quad a_j \quad z_j \quad --$$

in the table of the component FSM S^p, if a_i is the state of S^m and a_j is the state of S^p.

As a result of decomposition of FSM S_6, we obtain the network with three component FSMs in Fig. 23. Their transition tables are presented in Tables 18 – 20.

Now we will illustrate some examples of transitions for cases (a) and (b):

- In FSM S_6, there is a transition from state *a2* to state *a3* with input *~x6* and output *y3y5* (row 6 in Table 17). As both these states *a2* and *a3* are in the same component FSM S_1, in this FSM there is a transition from *a2* to *a3* with the same input *~x6* and the same output *y3y5* (row 2 in Table 18). Exactly in the same way, we rewrite row 12 of Table 17 into row 3 of Table 19 and row 2

of Table 17 into row 2 of Table 20 because the current states and the next states are in the same component FSMs.

- In FSM S_6, there is a transition from state *a3* to state *a8* with input *~x10*~x4* and the output *y5y8y9* (row 9 in Table 17). Since *a3* is the state of component FSM S^1 and *a8* is the state of another component FSM S^2, in FSM S^1 there is a transition from *a3* to *b1* with the same input *~x10*~x4* and output *y5y8y9z8* (row 5 in Table 18). The last output *z8* is the input of FSM S^2 that wakes this FSM up and transits it from state *b2* to state *a8* (row 8 in Table 19). Similarly, we convert row 1 of Table 17 into two rows – the first in Table 20 and the tenth in Table 18 etc. Note that we add the last row in each FSM table to remain component FSMs in the state b_m when each z_j is equal to zero.

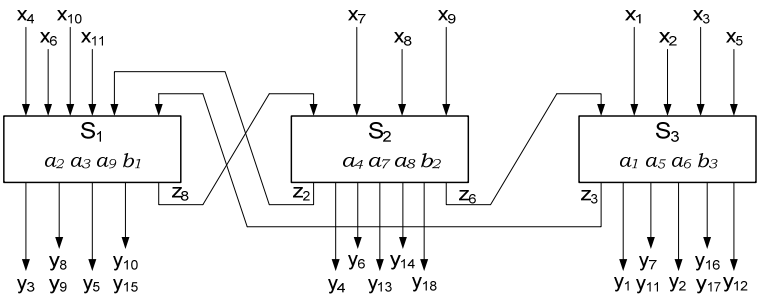

Figure 23. Network with three component FSMs

Table 18. Component FSM S^1

a_m	a_s	$X(a_m,a_s)$	$Y(a_m,a_s)$	H
a2	a2	x6	--	1
a2	a3	~x6	y3y5	2
a3	a3	x10	y3y5	3
a3	a9	~x10*x4	y10y15	4
a3	b1	~x10*~x4	y5y8y9z8	5
a9	a9	x11*x6	y10y15	6
a9	a2	x11*~x6	y5y8y9	7
a9	a3	~x11	y3y8y9	8
b1	a2	z2	--	9
b1	a3	z3	--	10
b1	b1	~z2*~z3	--	11

Table 19. Component FSM S^2

a_m	a_s	$X(a_m,a_s)$	$Y(a_m,a_s)$	H
a4	b2	x7	y13z6	1
a4	a4	~x7*x9	y13y18	2
a4	a8	~x7*~x9	y13y14	3
a7	b2	x8	y14y18z2	4
a7	a4	~x8	y13y18	5
a8	a7	x9	y4y6	6
a8	a4	~x9	y6	7
b2	a8	z8	--	8
b2	b2	~z8	--	9

Let us discuss how this network works. Let *a1* be an initial state in FSM S_6. After decomposition, state a_1 is in FSM S^3, so, at the beginning, just FSM S^3 is in state *a1*. Other FSMs are in states *b1* and *b2* correspondingly. It is possible to say that they "are sleeping" in these states. FSM S^3 transits from the state to the state until *x1*x2*x3 = 1* in state *a1* (see row 1 in Table 20). Only at this transition FSM S^3 produces output signal *z3* and transits into state *b3* (sleeping state). This signal *z3* is the input signal of FSM S^1. It wakes FSM S^1 up and transits it from the sleeping state *b1* to state *a3* (see row 10 in Table 18). Now FSM S^1 transits from the state to the state until, in state *a3,* it transits into state *b1* with input signal *~x10*~x4 = 1* and wakes FSM S^2 up by signal *z8* (see row 5 in Table 18 and row 8 in Table 19).

Table 20. Component FSM S^3

a_m	a_s	$X(a_m,a_s)$	$Y(a_m,a_s)$	H
a1	b3	x1*x2*x3	y1y2z3	1
a1	a6	x1*x2*~x3	y2y12	2
a1	a1	x1*~x2	y1y2	3
a1	a5	~x1	y1y2y12	4
a5	a6	x1	y16y17	5
a5	a5	~x1	y7y11	6
a6	a1	x5	y1y2	7
a6	a1	~x5	y16y17	8
b3	a6	z6	--	9
b3	b3	~z6	--	10

Unlike FSMs S^1 and S^3, the component FSM S^2 has two possibilities to wake other component FSMs up – in state *a4* with input signal *x7 = 1* (row 1 of Table 19) and in state *a7* with input signal *x8 = 1* (row 4 in the same Table), etc. Thus, each time all component FSMs, except one, are in the states of type b_m and only one of them is in the state of type a_i.

Chapter 5 Multilevel and Multioutput Synthesis

In this Chapter, we will concentrate on the multilevel minimization of logic circuits. Several simple and straightforward methods for obtaining circuit structure with more than two levels will be considered. In these methods, we will present four procedures – factoring, term decomposition, full inclusion and equal gates removal. At the end of the Chapter we will show how to construct optimized multilevel and multioutput circuits of Finite State Machines using only these four procedures.

5.1 Factoring

5.1.1 Two factoring structures. The first example of *factoring* is presented in Fig. 1. The left part of this figure implements the function

$$f_1 = x_1x_2x'_3x_4 + x_1x_2x'_5 + x_1x_2x_3x'_4. \qquad (1)$$

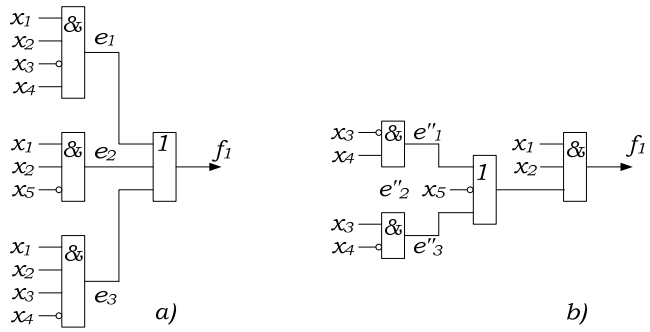

Figure 1. Factoring from all terms

All AND-gates of this circuit have the common input x_1x_2, so we can factor this common term (we call it a *factor*) in function (1):

$$f_1 = x_1x_2 (x'_3x_4 + x'_5 + x_3x'_4) \qquad (2)$$

The corresponding logic circuit is constructed in Fig. 1,b. In this circuit, e''_1, e''_2 and e''_3 contain inputs remained after deleting factor x_1x_2 from e_1, e_2 and e_3, and if there remains only one letter (x'_5 in our example), it will be an exact input into OR-gate.

Let us suppose again that the cost of a gate is equal to the number of its inputs, and that the cost of logic circuit is the sum of the costs of gates – the total number of inputs into all gates. If C_1 and C_2 are the costs of circuits before and after factoring then $C_1 - C_2$ is a minimization or a *gain of factoring*. We can evaluate the gain of factoring for the common term z by the formula

$$w(z) = m(n - 1) - 1 + r. \qquad (3)$$

Here m is the number of letters in factor z, n is the number of gates in factoring and r is the number of gates in which only one letter is left after factoring. In our example $w(z) = 2(3 - 1) - 1 + 1 = 4$. Really, if we count C_1 and C_2 in Fig.1, $C_1 - C_2 = 4$.

One more example of factoring is presented in Fig. 2. Unlike the previous example, here we can factor the common term $x_1x_3x'_4$ only from two AND-gates, not from all of them:

$$f_2 = x_1x_2x_3x'_4x_5 + x_4x_6x'_7 + x_1x'_2x_3x'_4x'_5 + x'_1x'_2;$$
$$f_2 = x_1x_3x'_4 (x_2x_5 + x'_2x'_5) + x_4x_6x'_7 + x'_1x'_2.$$

The result of factoring is shown in Fig. 2,b. On the right, we have OR-gate with three inputs – two of them from all AND-gates that do not take part in factoring (e_2, e_4) and the third one – from the output of the factoring structure for e_1, e_3 similar to Fig. 1,b.

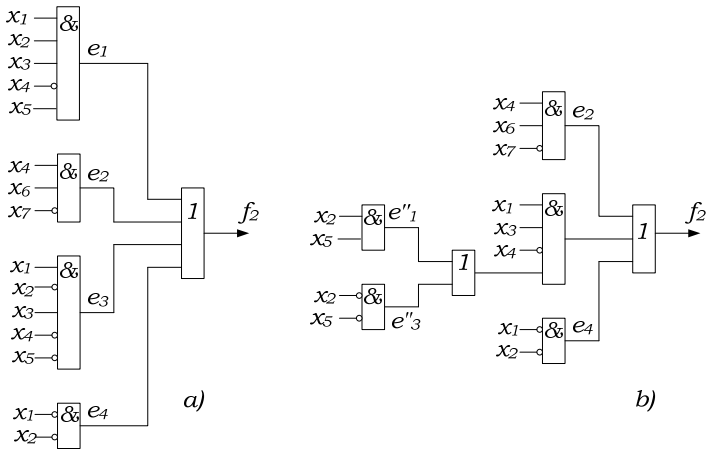

Figure 2. Factoring not from all terms

Again, we can evaluate the gain of factoring for the common term z by the formula

$$w(z) = m(n - 1) - 2 + r. \qquad (4)$$

Here m, n, and r are the same as in expression (3). See if you can understand why "-2" is used in this formula instead of "*-1*".

We discussed here two structures for factoring – structure one in Fig. 1,b (factoring from all AND-gates) and structure two in Fig. 2,b (factoring from some of AND-gates). The duality of Boolean functions permits us to use factoring not only for the sum-of-products, but for the product-of-sum as well (see Fig.3 and Fig. 4).

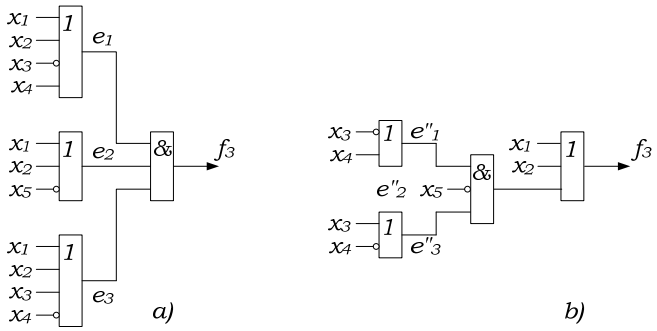

Figure 3. The first factoring structure for the product-of-sums

5.1.2 More than one factor. In the previous examples we have only one possible factor for factoring. Now we will discuss a case with several probable factors. As an

example let us use a two-level logic circuit corresponding to Boolean function $f = e_1 + e_2 + e_3 + e_4 + e_5$ with the products:

$e_1 = x_1x_2x_3x_4x_5x_6x_7x_{11}$; $e_4 = x_5x_6x_9$;
$e_2 = x_1x_2x_3x_8$; $e_5 = x_1x_2x_5x_6x_{10}x_{12}x_{13}$.
$e_3 = x_1x_2x_5x_6x_{10}x_{11}x_{12}$;

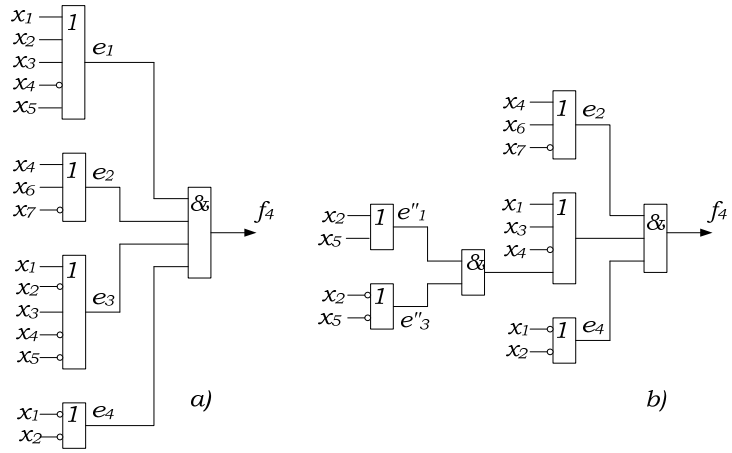

Figure 4. The second factoring structure for the product-of-sums

Let $e_i \cap e_j$ be the intersection between the products e_i and e_j (the common letters in these products). Our first step is to form all possible intersections between each pair of products in f. To do this, we construct Table 1. The first column of this table contains products $e_1, \ldots, e_5$. Intersections between all pairs of products are in the next columns, for example, $e_1 \cap e_2$ is in the column e_1 in the second row, $e_1 \cap e_3$ – in the column e_1 in the third row etc.

Table 1. Possible factors at the first step

$e_1 = x_1x_2x_3x_4x_5x_6x_7x_{11}$	e_1			
$e_2 = x_1x_2x_3x_8$	$x_1x_2x_3$	e_2		
$e_3 = x_1x_2x_5x_6x_{10}x_{11}x_{12}$	$x_1x_2x_5x_6x_{11}$	x_1x_2	e_3	
$e_4 = x_5x_6x_9$	x_5x_6	-	x_5x_6	e_4
$e_5 = x_1x_2x_5x_6x_{10}x_{12}x_{13}$	$x_1x_2x_5x_6$	x_1x_2	$x_1x_2x_5x_6x_{10}x_{12}$	x_5x_6

To find all possible factors, thus constructed, we should extract all different intersections from Table 1. There are six such factors $z_1, \ldots, z_6$ in this table. In this step, do not pay attention at the information in the parenthesis after each factor in expression (5):

$z_1 = x_1x_2x_3$ (e_1, e_2^*); $w(z_1) = 2$;
$z_2 = x_1x_2x_5x_6x_{11}$ (e_1, e_3); $w(z_2) = 3$;
$z_3 = x_5x_6$ (e_1, e_3, e_4^*, e_5); $w(z_3) = 5$; (5)
$z_4 = x_1x_2x_5x_6$ (e_1, e_3, e_5); $w(z_4) = 6$;
$z_5 = x_1x_2$ (e_1, e_2, e_3, e_5); $w(z_5) = 4$;
$z_6 = x_1x_2x_5x_6x_{10}x_{12}$ (e_3^*, e_5^*); $w(z_6) = 6$.

We will use formulas (3) and (4) to evaluate the gain of each factor. To do this we should find m, n and r for each factor. Here m is the number of letters in the factor, n is the number of gates in factoring and r is the number of gates in which only one letter is left after factoring of this factor. m is trivial – for z_1, m is equal to 3; for z_2, m is equal to 5 etc. To find n, we should intersect each z_t ($t = 1, \ldots, 6$) with each e_i ($i = 1,$

…, 5). If z_t is contained in e_i, then z_t is the factor of e_i and we write e_i in the parenthesis after z_t. Thus, for z_1, z_2 and z_6, n is equal to 2, for z_3 and z_5, n is equal to 4 etc.

While performing such intersections, it is possible to find r as well. For example, when we intersect z_1 with e_2 we see that $z_1 \in e_2$ and only one letter is left after factoring z_1 from e_2, because z_1 has three letters but e_2 has four. The symbol * next to e_2 in the line for z_1 means that only one letter is left. We have the same for z_3 (e_4*) and z_6 (e_3*, e_5*). When we have m, n and r for each factor, the evaluation is trivial. $w(z_t)$ for each z_t is presented in the second column of (5).

In the first step of factoring, we use a factor with a maximal gain. If we have several such factors (two in our example – z_4 and z_6) it is possible to implement one of the following strategies:

1. Take the first of such factors (the simplest strategy);
2. Take the factor with maximal length from these factors;
3. Take the factor contained in the maximum number of gates;
4. Move one step forward for each such factor and select factor after the second evaluation step etc.

We will use the first trivial strategy and select z_4 with

$$w(z_4) = 6 = max.$$

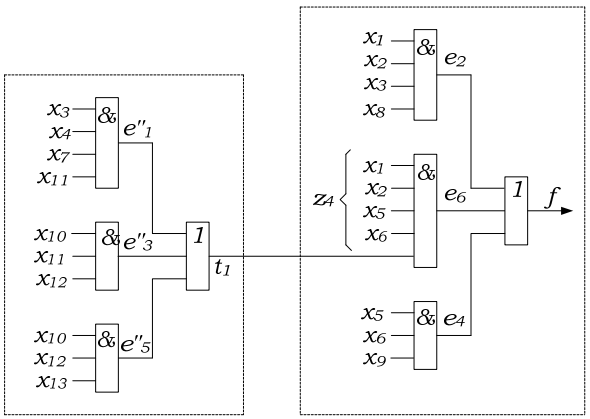

Figure 5. The circuit after the first step of factoring

The circuit after factoring of z_4 is shown in Fig. 5. It implements two functions presented as sum-of-product:

1. Function f is the sum-of-products with three AND-gates, one of them contains the factor z_4, and two others – the products that do not take part in factoring;
2. Function t_1 is the sum-of-products with three AND-gates, each of them corresponds to one of the products that took part in factoring. These ANDs have inputs remaining after factoring of z_4.

Fig. 6 presents the factoring process. The first box in this figure contains the set of products e_1, …, e_5, the second one – the partitioning of this set into two subsets after the first step. Thus, we must continue the factoring separately for two functions presented as sum-of-products – function f containing products e_2, e_4, e_6 and function t_1 containing products e''_1, e''_3, e''_5. A similar partition will be at each next step so the process of factoring converges very fast.

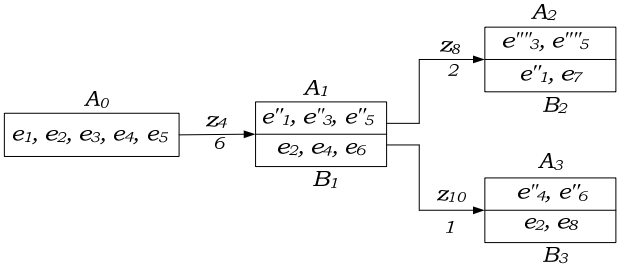

Figure 6. Steps of factoring

The subsequent steps of factoring for functions t_1 and f are presented in Tables 2 and 3. The factoring process comes to the end when there are no factors with the gain greater than zero. The final circuit after factoring is presented in Fig. 7. The total cost reduction is equal to

$$w(z_4) + w(z_8) + w(z_{10}) = 9.$$

Table 2. Factoring of function t_1

$e''_1 = x_3x_4x_7x_{11}$	e''_1	
$e''_3 = x_{10}x_{11}x_{12}$	x_{11}	e''_3
$e''_5 = x_{10}x_{12}x_{13}$	-	$x_{10}x_{12}$

$z_7 = x_{11}\,(e''_1, e''_3);$ $\quad w(z_7) = -1;$
$z_8 = x_{10}x_{12}\,(e''_3{}^*, e''_5{}^*);$ $\quad w(z_8) = 2.$

$$w(z_8) = 2 = max.$$

Table 3. Factoring of function f

$e_2 = x_1x_2x_3x_8$	e_2	
$e_4 = x_5x_6x_9$	-	e_4
$e_6 = x_1x_2x_5x_6t_1$	x_1x_2	x_5x_6

$z_9 = x_1x_2\,(e_2, e_6);$ $\quad w(z_9) = 0;$
$z_{10} = x_5x_6\,(e_4{}^*, e_6);$ $\quad w(z_{10}) = 1.$

$$w(z_{10}) = 1 = max.$$

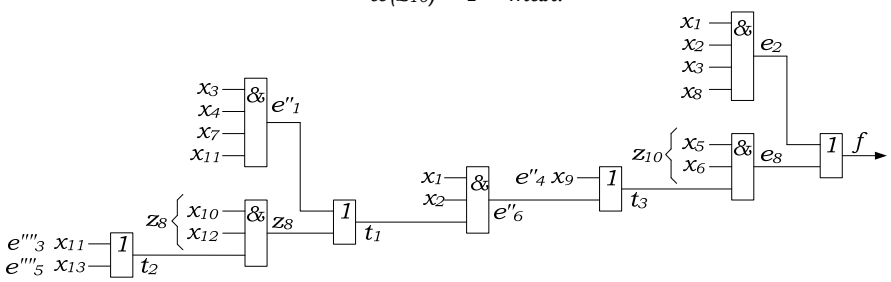

Figure 7. The circuit after factoring

5.2 Term Decomposition

5.2.1 Simple example. The first example of *term decomposition* is presented in Fig. 8. Left part of this figure contains three separate AND-gates implementing three functions g_1, g_2 and g_3. All gates of this circuit have the common inputs x_4, x'_5, x'_6, so we can construct additional AND-gate z with these inputs and replace inputs x_4, x'_5, x'_6 of initial gates with the output of gate z (Fig. 8,b).

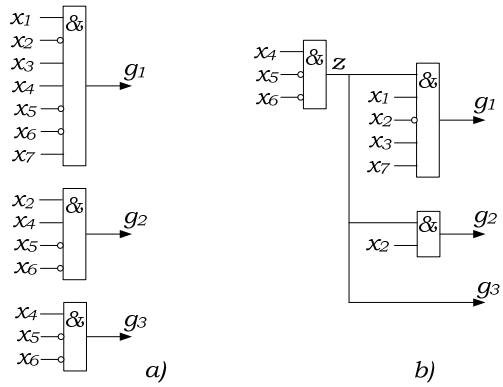

Figure 8. Simple term decomposition

If C_1 and C_2 are the costs of circuits before and after term decomposition then $C_1 - C_2$ is a minimization or a *gain of term decomposition*. We can evaluate the gain of term decomposition for the common term z by the formula

$$w(z) = m(n - 1) - n + r. \qquad (6)$$

Here m is the number of letters in the common term z, n is the number of gates in term decomposition and r is the number of functions (initial AND-gates) equal to the common term. In our example $w(z) = 3(3 - 1) - 3 + 1 = 4$. Really, if we count C_1 and C_2 in Fig. 8, $C_1 - C_2 = 4$.

5.2.2 More than one common term. In the previous example, we had only one possible term for term decomposition. Now we will discuss the case with several probable common terms. As an example let us use a circuit in Fig. 9 that corresponds to the following products:

$g_1 = x_1x_2x_3x_4x_5x_6x_7x_{11};$
$g_2 = x_1x_2x_3x_8;$
$g_3 = x_1x_2x_5x_6x_{10}x_{11}x_{12};$
$g_4 = x_5x_6x_9;$
$g_5 = x_1x_2x_5x_6x_{10}x_{12}.$

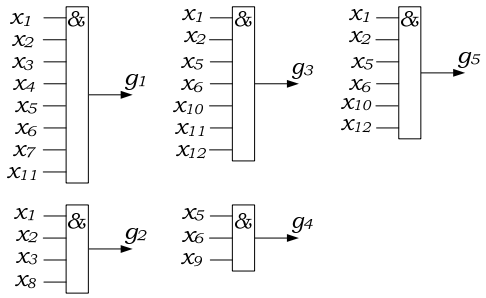

Figure 9. Logic circuit before term decomposition

As in factoring, the algorithm of term decomposition consists of several steps. The first step is to form intersections between each pair of products to find all possible common terms containing two or more variables (see Table 4). We will use formula (6) to evaluate the gain of each common term. To do this we should find m, n and r for each term: m is trivial – it is the number of letters in the common term. For z_1, m is

equal to 3; for z_2, m is equal to 5 etc. It is clear that the common term with one variable makes no sense in term decomposition.

Table 4. Possible common terms at the first step

$g_1 = x_1x_2x_3x_4x_5x_6x_7x_{11}$	g_1			
$g_2 = x_1x_2x_3x_8$	$x_1x_2x_3$	g_2		
$g_3 = x_1x_2x_5x_6x_{10}x_{11}x_{12}$	$x_1x_2x_5x_6x_{11}$	x_1x_2	g_3	
$g_4 = x_5x_6x_9$	x_5x_6	-	x_5x_6	g_4
$g_5 = x_1x_2x_5x_6x_{10}x_{12}$	$x_1x_2x_5x_6$	x_1x_2	$x_1x_2x_5x_6x_{10}x_{12}$	x_5x_6

$$
\begin{array}{ll}
z_1 = x_1x_2x_3\ (g_1, g_2); & w(z_1) = 1; \\
z_2 = x_1x_2x_5x_6x_{11}\ (g_1, g_3); & w(z_2) = 3; \\
z_3 = x_5x_6\ (g_1\ g_2, g_4, g_5); & w(z_3) = 2; \\
z_4 = x_1x_2x_5x_6\ (g_1, g_3, g_5); & w(z_4) = 5; \\
z_5 = x_1x_2\ (g_1, g_2, g_3, g_5); & w(z_5) = 2; \\
z_6 = x_1x_2x_5x_6x_{10}x_{12}\ (g_3, g_5^*); & w(z_6) = 5.
\end{array}
\qquad (7)
$$

To find n, we should intersect each z_t ($t = 1, \ldots, 6$) with each g_i ($i = 1, \ldots, 5$). If $z_t \in g_i$, then z_t is the common term for g_i and we write g_i in the parenthesis after z_t. While performing such an intersection it is possible to find r as well. For example, when we intersect z_6 with g_5 we see that $z_6 = g_5$. The symbol * near g_5 in the line for z_6 means that the product g_5 is equal to the common term z_6. When we have m, n and r for each common term, the evaluation is trivial – $w(z_t)$ for each z_t is presented in the second column of (7).

In the first step of term decomposition, we use a common term with the maximal gain. If we have several such terms (two in our example – z_4 and z_6), as in factoring, it is possible to implement the following several strategies:

1. Take the first of such common terms (the simplest strategy);
2. Take the common term with maximal length from these common terms;
3. Take the common term contained in the maximum number of gates;
4. Move one step forward for each such common term and select common term after the second step evaluation etc.

We will use the first strategy and select z_4 with

$$w(z_4) = 5 = max.$$

The circuit after decomposition of z_4 is shown in Fig. 10.

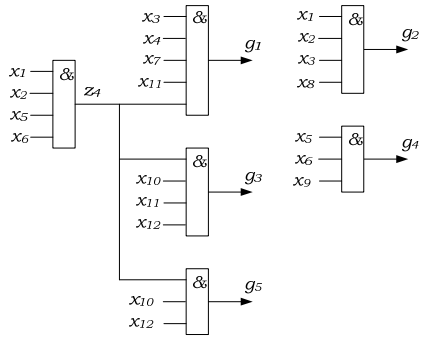

Figure 10. Logic circuit after the first step of term decomposition

Unlike factoring, where we had a partition of products into two subsets after each step, there is no partition of initial products is here. Moreover, the common term taking part in term decomposition should be added to the set of products and will be used at the next step together with other products. Only the product equal to the common term should be excluded from the list of products in the next step of term decomposition.

The next step of term decomposition is presented in Table 5. The process comes to the end when there are no factors with the gain greater than zero. The final circuit after term decomposition is shown in Fig. 11, the whole process is illustrated by Fig. 12.

Table 5. The second (final) step of term decomposition

$g''_1 = x_3x_4x_7x_{11}z_4$	g''_1				
$g_2 = x_1x_2x_3x_8$	-	g_2			
$g''_3 = x_{10}x_{11}x_{12}z_4$	$x_{11}z_4$	-	g''_3		
$g_4 = x_5x_6x_9$	-	-	-	g_4	
$g''_5 = x_{10}x_{12}z_4$	-	-	$x_{10}x_{12}z_4$	-	g''_5
$z_4 = x_1x_2x_5x_6$		x_1x_2	-	x_5x_6	-

$z_7 = x_{11}z_4\ (g''_1, g''_3);$ $w(z_1) = 0;$
$z_8 = x_1x_2\ (g_2, z_4);$ $w(z_2) = 0;$
$z_9 = x_{10}x_{12}z_4\ (g''_3, g''_5{}^*);$ $w(z_9) = 2;$
$z_{10} = x_5x_6\ (g_4, z_4);$ $w(z_{10}) = 0.$

$$w(z_9) = 2 = max.$$

The total cost reduction is equal to

$$w(z_4) + w(z_9) = 7.$$

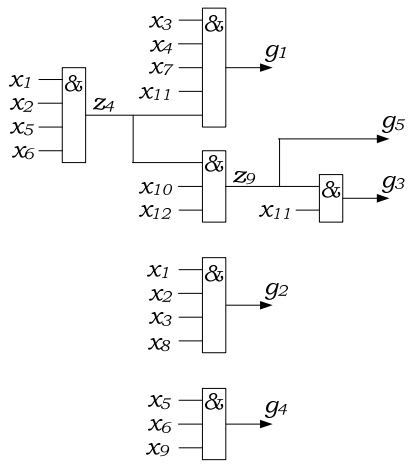

Figure 11. The circuit after t-decomposition

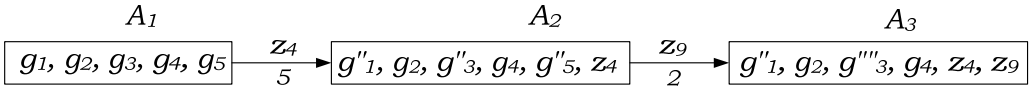

Figure 12. Steps of t-decomposition

5.2.3 Term decomposition for OR gates. Term decomposition can be applied to OR gates as well. We will give the next example without any comments and you can fulfill each step on your own (Fig. 13).

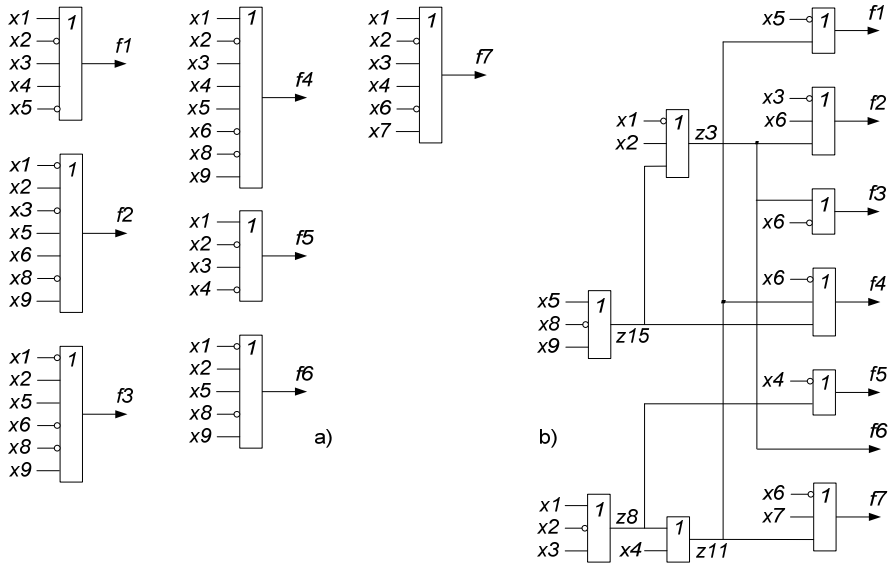

Figure 13. Term decomposition for OR gates

5.3 Gate inclusion

Let us define gate *m* as *included* in gate *n*, or gate *n* as *covering* gate *m*, if they have the same type (both AND or both OR) and the set of inputs of gate *m* is a subset of the set of inputs of gate *n*. The simplest case of gate inclusion is presented in Fig 14,a. In this case, we can replace inputs of gate *n*, equal to the inputs of gate *m* (x_1 and x_2 in our example), with the output of gate *m* (Fig. 14,b).

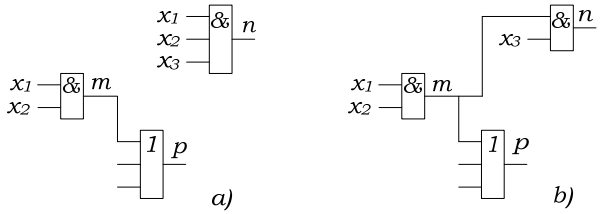

Figure 14. Gate inclusion

5.4 Removal of equal gates

Let us define as gates *m* and *n* *equal*, if they have the same type (both AND or both OR) and the set of inputs of gate *m* is equal to the set of inputs of gate *n*. The circuit in Fig. 15,a contains four equal two-input AND-gates. In this case, we should

1. Remove all equal gates, except one (gates *l*, *m* and *n* in our example);
2. Connect inputs of gates (*t*, *p* and *q*) formerly connected to the outputs of removed gates, with the output of the remained gate (gate *k* in our example).

The last two procedures – gate inclusion and removal equal gates are covered by term decomposition. Really, in the first step of term decomposition – pair intersection, we can find equal gates and gates included into other gates. However, term decomposition has two problems: (1) the large number of gates taking part in this procedure; (2) multiple comparisons demand a lot of intersections between sets of inputs. It is more simple and faster to check gate inclusion and remove equal gates before term decomposition. Moreover, after these two procedures, only gates with three and more inputs remain for term decomposition (see if you can understand why it is so).

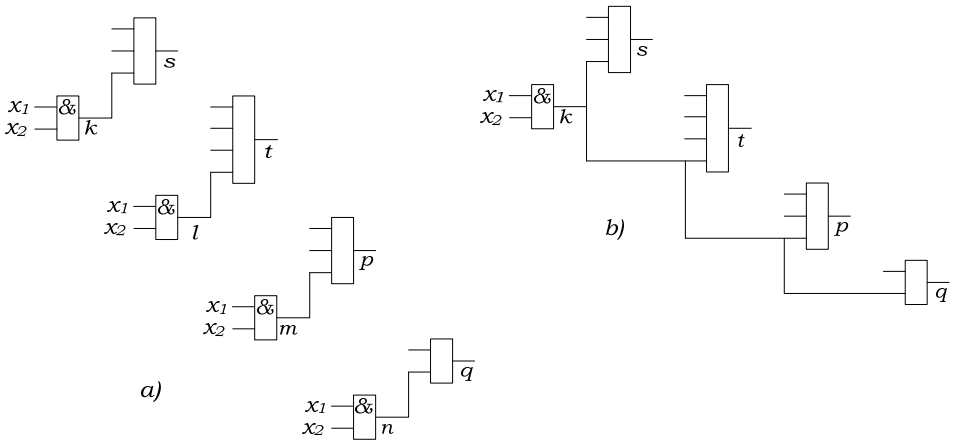

Figure 15. Removal three equal AND-gates

5.5 Multilevel and multioutput circuits for Finite State Machines

In Section 4.2.6 of Chapter 4, we considered a very simple method for synthesis of the two level FSM logic circuit from its structure table. Recall that we have used the term

$$e_h = A_m X_h$$

in accordance with the h row of such a table ($h = 1, \ldots, H$). Here A_m is a product of state variables corresponding to the current state a_m written in the h row, X_h is a product of input variables written in the same row, and H is the number of rows in the structure table. Then we constructed H AND-gates corresponding to terms $e_1, \ldots, e_H$. If the output variable y_n appears only once, for example, in row i of the structure table, we obtain the output y_n at the output of AND-gate number i. If the output variable y_n is written in several rows, for example, in rows $p_1, \ldots, p_T$ of the structure table, we construct OR-gate with T inputs and connect these inputs with the outputs of AND-gates $p_1, \ldots, p_T$. The output y_n is obtained at the output of this OR-gate. In exactly the same way, we construct OR-gate for each input memory function which occurs more than once in the column $D(a_m,a_s)$ of the structure table. The logic circuit of FSM thus constructed contains not more than H AND-gates and not more than $(N + R)$ OR-gates where N and R are the numbers of output variables and input memory functions in the FSM structure table.

In this section, we will use the reverse structure table. Recall that in such a table all transitions are ordered according the next state – first we write all transitions *to* state a_1, then *to* state a_2 , etc. As an example we will consider the logic synthesis of FSM S,

Table 6 is its reverse structure table. As in four previous sections, we assume that the circuit cost is equal to the sum of inputs of its gates.

Table 6. The reverse structure table of FSM *S*

a1	001	a1	001	x8*x7	y7y9y14y15	d3	1
a1	001	a1	001	x8*~x7*x1*x9*x5	y13	d3	2
a1	001	a1	001	~x8*x1*x9*x5	y13	d3	3
a3	011	a1	001	x9*x5	y13	d3	4
a4	000	a1	001	x4*~x9*x3	y2y10y12	d3	5
a5	010	a1	001	x4	--	d3	6
a1	001	a2	100	x8*~x7*~x1	y1y2y3	d1	7
a1	001	a2	100	~x8*~x1	y1y2y3	d1	8
a2	100	a2	100	~x2	--	d1	9
a2	100	a3	011	x2	y4	d2d3	10
a4	000	a3	011	x4*~x9*~x3	y5y6	d2d3	11
a4	000	a3	011	x4*x9	y5y6	d2d3	12
a1	001	a4	000	x8*~x7*x1*~x9*x3*~x6	y7y8y9	--	13
a1	001	a4	000	~x8*x1*x9*~x5	y7y8y9	--	14
a3	011	a4	000	x9*~x5	y7y8y9	--	15
a3	011	a4	000	~x9*x3*~x6	y7y8y9	--	16
a3	011	a4	000	~x9*~x3	y7y8y9	--	17
a1	001	a4	000	~x8*x1*~x9*x3*~x6	y7y8y9	--	18
a1	001	a4	000	~x8*x1*~x9*~x3	y7y8y9	--	19
a1	001	a4	000	x8*~x7*x1*~x9*~x3	y7y8y9	--	20
a4	000	a4	000	~x4	--	--	21
a1	001	a4	000	x8*~x7*x1*x9*~x5	y7y8y9	--	22
a1	001	a5	010	x8*~x7*x1*~x9*x3*x6	y10y11y12	d2	23
a3	011	a5	010	~x9*x3*x6	y10y11y12	d2	24
a1	001	a5	010	~x8*x1*~x9*x3*x6	y10y11y12	d2	25
a5	010	a5	010	~x4	--	d2	26

The structure table is divided into M arrays, each of which corresponds to the set of transitions into one state. For FSM in Table 6, M is equal to five. In several initial steps, we will separately design logic circuits for transitions into each state. Moreover, even then we will construct circuits separately for each subset of output signals.

A design of the logic circuit consists of the following steps:

Step 1. Divide each array of transitions to the state a_s (s = 1, …, M) into as many subarrays, as the number of different microinstructions (the subsets of output variables) in the column Y(a_m, a_s) within this array. For example, in Table 6, transitions into state a_1 have four microinstructions:

y7, y9, y14, y15;
y13; (8)
y2, y10, y12;
$\varnothing$.

We should include the empty microinstruction, corresponding to row 6, in this list because not all of input memory functions are equal to zero at this transition (*d3 = 1*) and we must construct AND-gate for this row.

Thus, in our example we have four such subarrays containing

1. Row 1 with outputs *y7, y9, y14, y15*;
2. Rows 2, 3, 4 with output *y13;*
3. Row 5 with outputs *y2, y10, y12;*
4. Row 6 without output signals.

Step 2. For each subarray corresponding to one of microinstruction in (8), construct as many AND-gates as the number of rows in this subarray of the structure table. These gates implement products $A_mX(a_m,a_s)$, corresponding to each row. In our example for the transitions into *a1* we have six such AND-gates (see Fig.16,a).

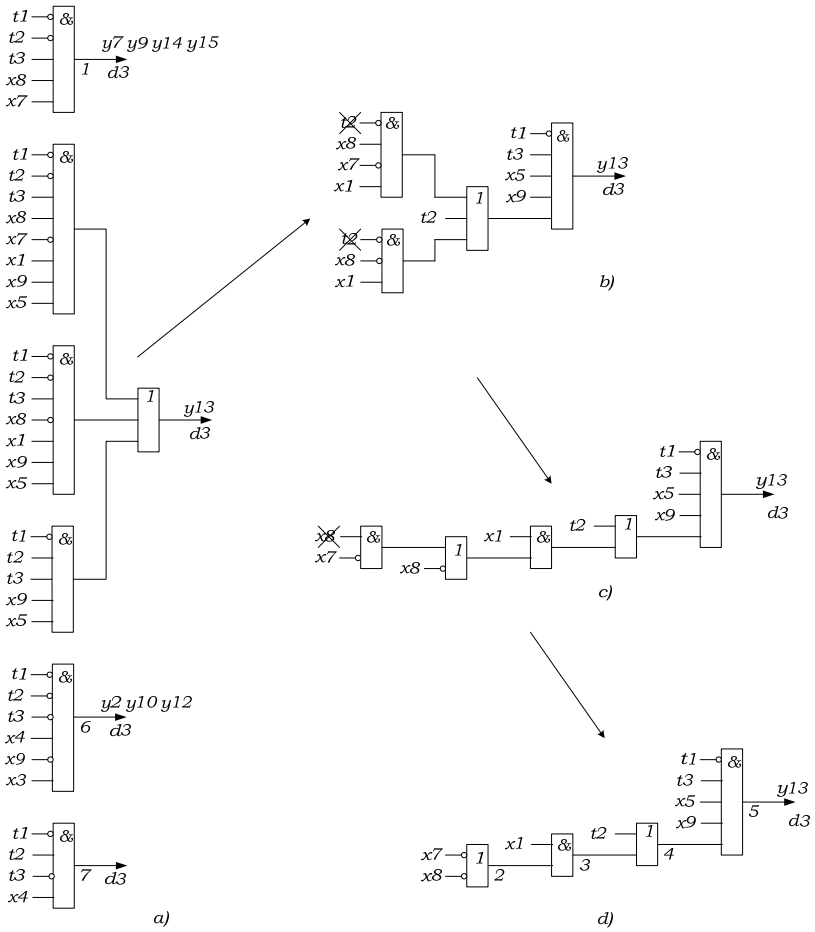

Figure 16. Logic circuit for transitions into state a_1

Step 3. If some subarray contains more than one row, connect the outputs of corresponding AND-gates, constructed at step 2 for the subarray, with OR-gate to form the signals of microoperations (output variables) and input memory functions written in the rows of this subarray (rows 2, 3, 4 for y13 – Fig. 16,a).

Step 4. Factor the logic circuits constructed in point 3 using the algorithm described in Section 5.1 'Factoring'. Let us do this for functions *y13, d3.* Table 7 contains the first step of this factoring. We made all pair intersections between products corresponding to rows with *y13, d3* and found two possible factors z_1 and z_2. We factor z_2 with max gain (see Fig. 16,b).

It is possible to make one more simplification in the circuit in Fig. 16,b. OR-gate has input t_2 and AND-gates connected with this OR-gate have inputs t'_2. According Boolean algebra $A + A'B = A + B$, so we can delete inputs t'_2 from AND-gates. To make such minimization we do not have to write any formulas. If some OR-gate has some

input p (p') we should check all AND-gates connected with this OR-gate and remove inputs p' (p) from these AND-gates.

Table 7. The first step of factoring for y_{13} and d_3

$e_1 = t'_1t'_2t_3x_8x'_7x_1x_9x_5$	e_1	
$e_2 = t'_1t'_2t_3x'_8x_1x_9x_5$	$t'_1t'_2t_3x_1x_9x_5$	e_2
$e_3 = t'_1t_2t_3x_9x_5$	$t'_1t_3x_9x_5$	$t'_1t_3x_9x_5$

$z_1 = t'_1t'_2t_3x_1x_9x_5$ (e_1, e_2^*); $w(z_1) = 6(2 - 1) - 2 + 1 = 5$;
$z_2 = t'_1t_3x_9x_5$ (e_1, e_2, e_3^*); $w(z_2) = 4(3 - 1) - 1 + 1 = 8$;

$$w(z_2) = 8 = max.$$

The last step of factoring is shown in Table 8 and Fig. 16,c. After removing input x_8 from AND-gate with two inputs we must remove this AND-gate as well, and transfer input x'_7 into the OR-gate. The final step of factoring is presented in Fig. 16,d.

Table 8. The second steps of factoring for y_{13}, d_3

$e''_1 = x_8x'_7x_1$	e''_1
$e''_2 = x'_8x_1$	x_1

$z_3 = x_1$ $(e''_1, e''_2{}^*)$; $w(z_3) = 1(2 - 1) - 1 + 1 = 1$;

$$w(z_3) = 1 = max.$$

Logic circuit after factoring for transitions into state a_1 contains seven gates – we numbered gates after the last step. Each step of circuit factoring for transitions into states a_2 and a_3 is presented in Fig.17 and Fig. 18. Logic circuits for transitions into states a_4 and a_5 without intermediate steps are shown in Fig. 19 and Fig. 20. We have left the design of these last circuits to our readers as exercise to be done on their own. At last, we bring all these circuits together in Fig. 21.

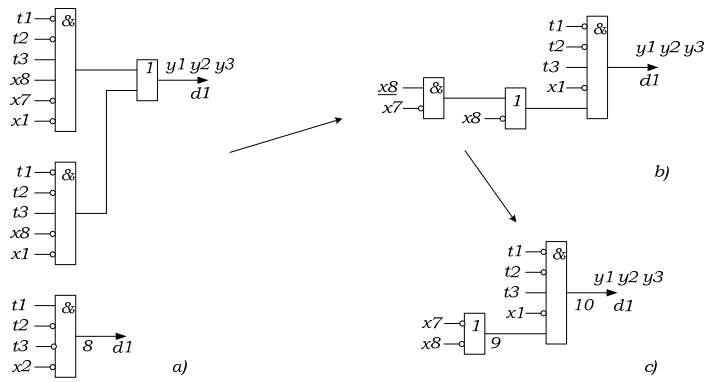

Figure 17. Logic circuit for transitions into state a_2

<u>*Step 5. Delete equal gates in the logic circuit thus constructed.*</u> If we look at the circuit after factoring in Fig. 21 we will find that it contains some equal gates. For example, six two-input OR-gates OR_2, OR_9, OR_{14}, OR_{16}, OR_{18} and OR_{27} are equal because they have the same type and the same inputs. However, AND-gates $AND_3(x_1,2)$ and $AND_{28}(x_1,27)$ are not equal because they have inputs from different gates. To find that they are also equal we must determine that OR_2 and OR_{27} are equal and change the

input 27 by input *2* in the description of AND_{28}. Therefore, to find that two gates are equal in a multilevel circuit we should find that their preceding gates are equal etc. For this reason we should rank the gates in the circuit.

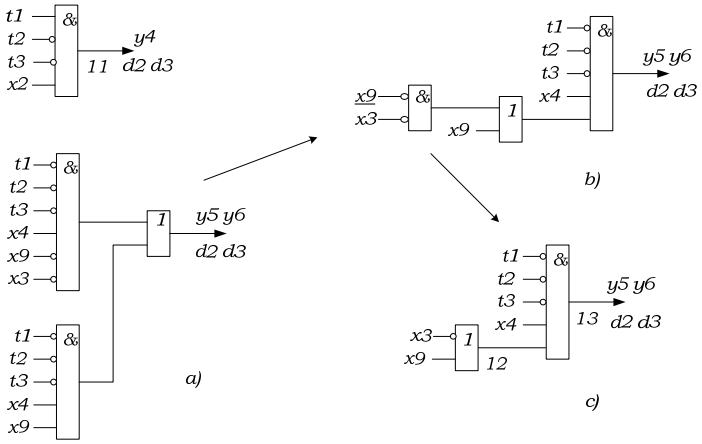

Figure 18. Logic circuit for transitions into state a_3

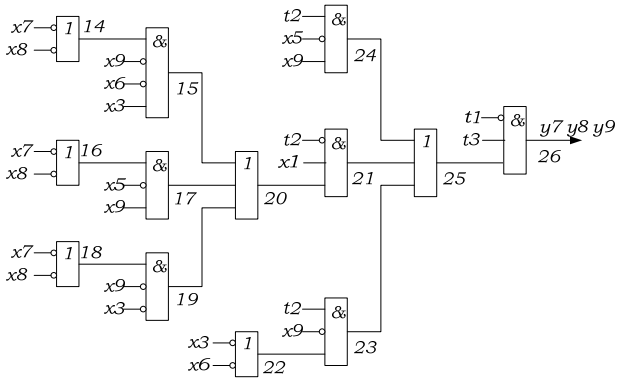

Figure 19. Logic circuit for transitions into state a_4

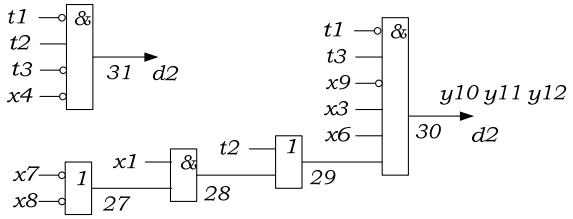

Figure 20. Logic circuit for transitions into state a_5

Gates containing only inputs $t_1, \ldots, t_R$ (the outputs of the memory elements, in our example $R = 3$) and input variables $x_1, \ldots, x_L$ (in our example $L = 9$) are referred to as gates of the first rank. The gates with inputs $t_1, \ldots, t_R$, input variables and the output of at least one gate of the first rank are referred to as gates of the second rank etc. Thus, the i-rank gate can have inputs $t_1, \ldots, t_R$, input variables and the inputs from outputs of gates with the rank less than $(i - 1)$ and at least one input from the gate

with rank (*i* – *1*). The results of ranking for the circuit in Fig. 21 are presented in Table 9 and Fig.22. In this figure, the rank of gate is written above the gate.

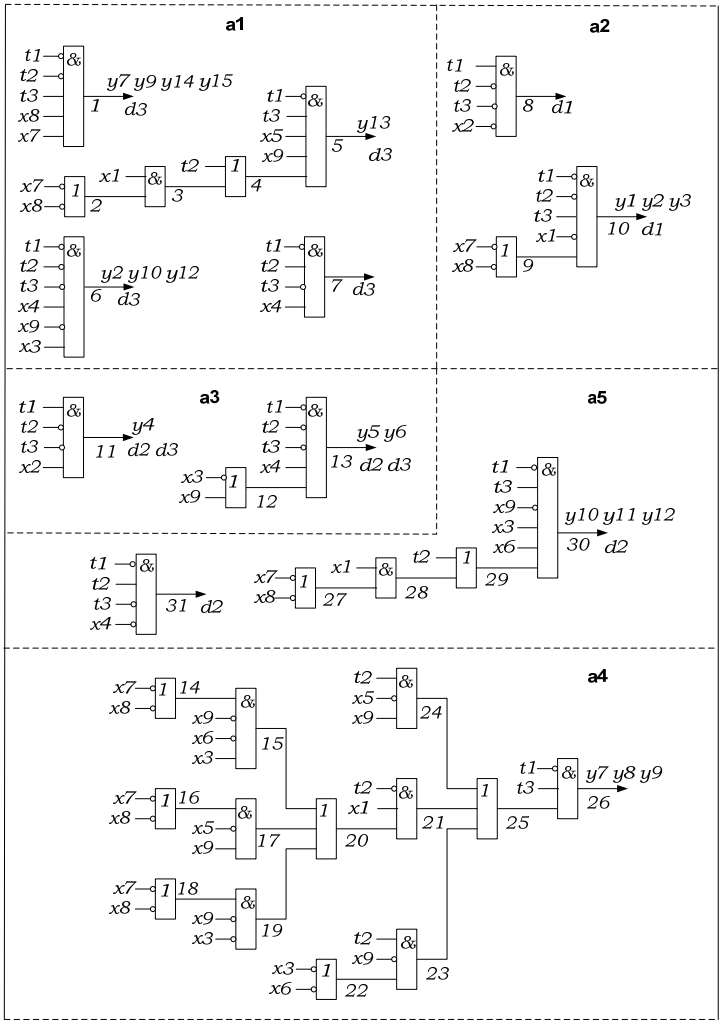

Figure 21. Logic circuit after factoring

Table 9. Ranks of gates

Rank	AND-gates	OR-gates
1	1, 6, 7, 8, 11, 24, 31	2 , 9, 12, 14, 16, 18, 22, 27
2	3, 10, 13, 15, 17, 19, 23, 28	
3		4, 20, 29
4	5, 21, 30	
5		25
6	26	

It is evident that equal gates can only be of the same rank. The following steps should be used to find and delete equal gates:

1. Find equal gates with rank *i* (*i = 1, 2, 3, ...*) beginning from rank 1, separately for AND-gates and OR-gates. In our example, we have the following set of equal first-rank gates:

$$OR_2 = OR_9 = OR_{14} = OR_{16} = OR_{18} = OR_{27}.$$

2. Remove all gates except the first one from each such set. Thus, after the first step we removed five gates OR_9, OR_{14}, OR_{16}, OR_{18}, OR_{27}. Replace the inputs from the gates thus removed with the number of the first (not removed) gate from the corresponding sets.
3. Repeat steps (1) – (3) for the elements of the $(i+1)$-*th* rank. We get equal AND-gates AND_3 and AND_{28} of the second rank and equal OR-gates OR_4 and OR_{29} of the third one.

The circuit after removal equal gates is shown in Fig. 23.

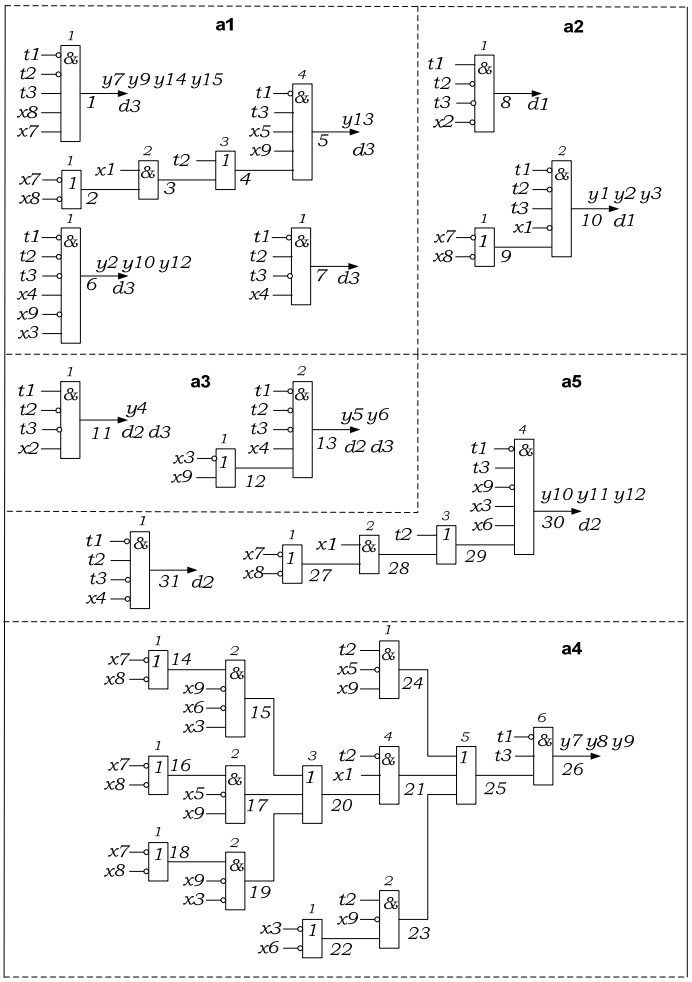

Figure 22. Ranking after factoring

Step 6. Repeat factoring and removing equal gates until the circuit cannot be change any longer. Look at the circuit implementing the transitions into a_4 in Fig. 23. We drew the part of this circuit containing gates AND_{15}, AND_{17}, AND_{19} and OR_{20} in Fig. 24,a. After the removal of the equal gates, logic elements AND_{15}, AND_{17}, and AND_{19} got the same inputs from OR_2 instead of different inputs from OR_{14}, OR_{16}, and OR_{18}. Thereby, we got new possibilities for repeated factoring – see sequential steps of factoring in Fig. 24,b,c.

The circuit after the second factoring is shown in Fig. 25. Once again, after factoring we find the equal gates: $OR_{22} = OR_{32}$ and we remove the last one (Fig. 26). Thus, we

should repeat factoring and removing equal gates as long as we get these procedures are impossible for the circuit.

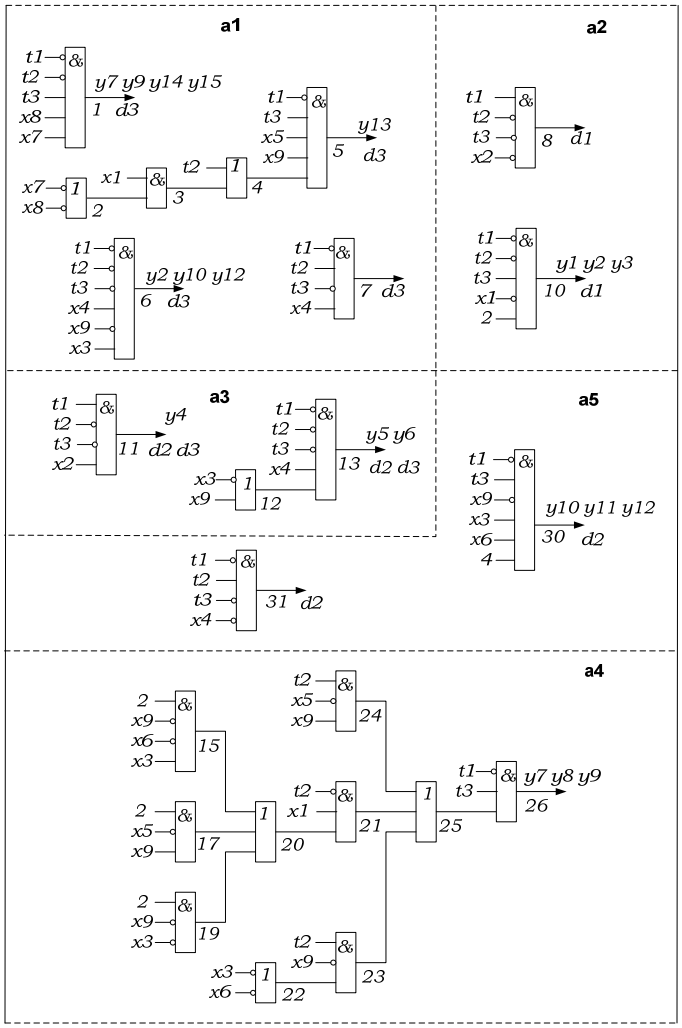

Figure 23. Logic circuit after removal equal gates

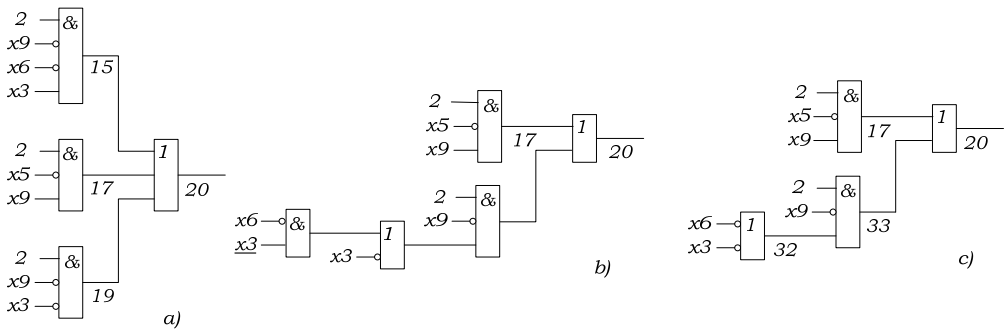

Figure 24. Repeated factoring

<u>*Step 7. Find the inclusion of gates into other gates.*</u> Unfortunately, we do not have such cases in our rather simple example.

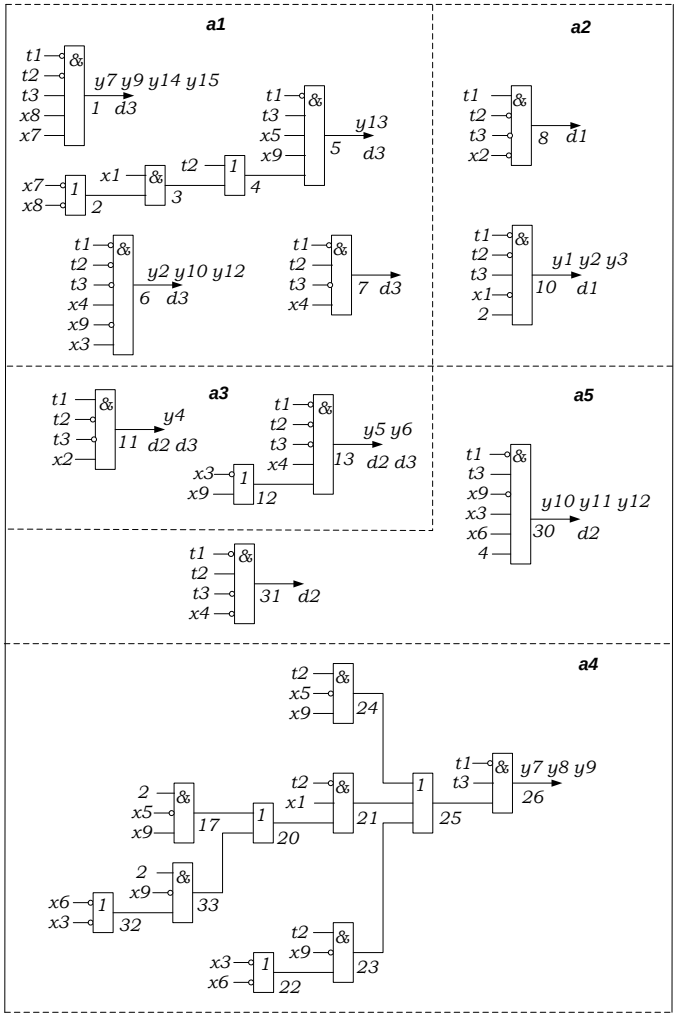

Figure 25. Circuit after the second factoring

Step 8. Make term decomposition for AND-gates. Fig. 27,a contains AND-gates with three and more inputs which we have selected from the circuit in Fig. 26 for term decomposition. It is evident that term decomposition makes it possible to find equal gates and inclusion of some gates into other ones as well. However, if a circuit contains many gates, term decomposition takes a lot of time and it is faster to implement steps 5 – 7 before term decomposition.

Let us demonstrate that the term decomposition problem may be divided into several independent subproblems. For this purpose, we define such a relation ω on the set of AND-gates that two gates AND_i and AND_j are in this relation iff they have not less than two common inputs.

Construct the graph G_ω of this relation (Fig. 28 for the circuit in Fig. 27,a). The vertices of this graph are the gates in Fig. 27. We connect two vertices by edge if the corresponding gates have two or more common inputs. From the definition of the relation ω, it is evident that there can be no common factors for the gates from various subgraphs of G_ω. Thus, the problem of term decomposition is divided into as many subproblems as the number of unconnected components in the graph G_ω. Even

in our simple example, G_ω contains five components and there are only 11 vertices (gates) in the largest component. For a complex FSM, the graph G_ω contains a large number of components, since:

1. There is a large number of input variables in a complex FSM and there are not so many input variables in each row of its structure table (in each term corresponding to each row);
2. x_i and x'_i are different inputs of gates;
3. The number of gates and the number of inputs in each gate are decreased, as a result of steps 4 – 7 (factoring, removal equal gates and inclusion of gates into other ones).

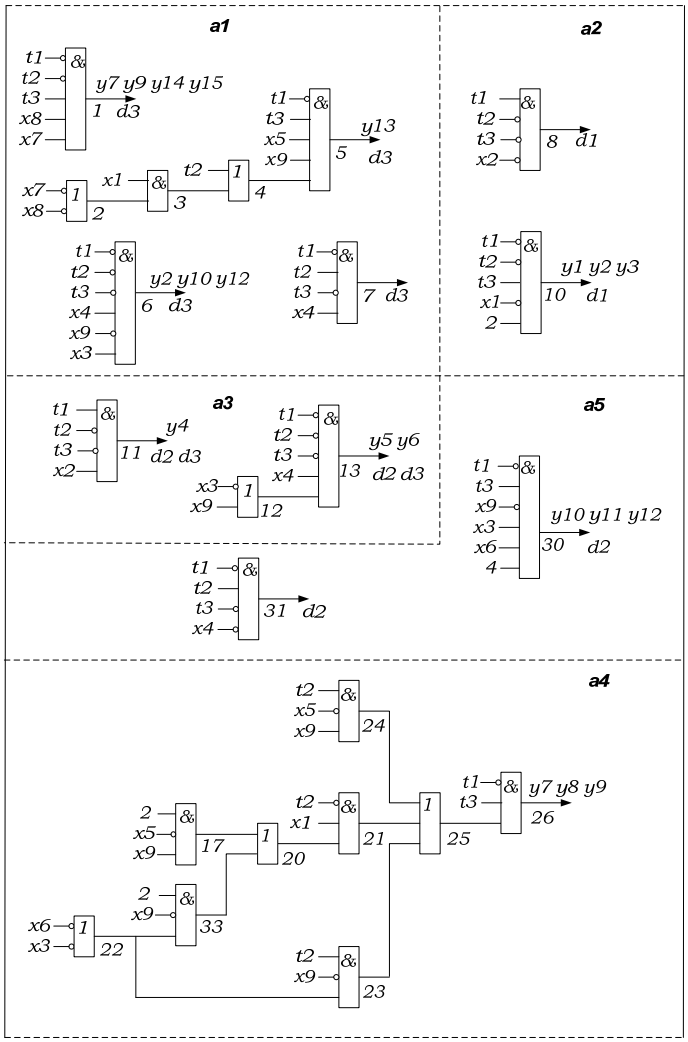

Figure 26. Circuit after the second removal equal gates

The result of term decomposition in our example is shown in Fig. 27,b. Fig. 29 contains the total circuit after this step.

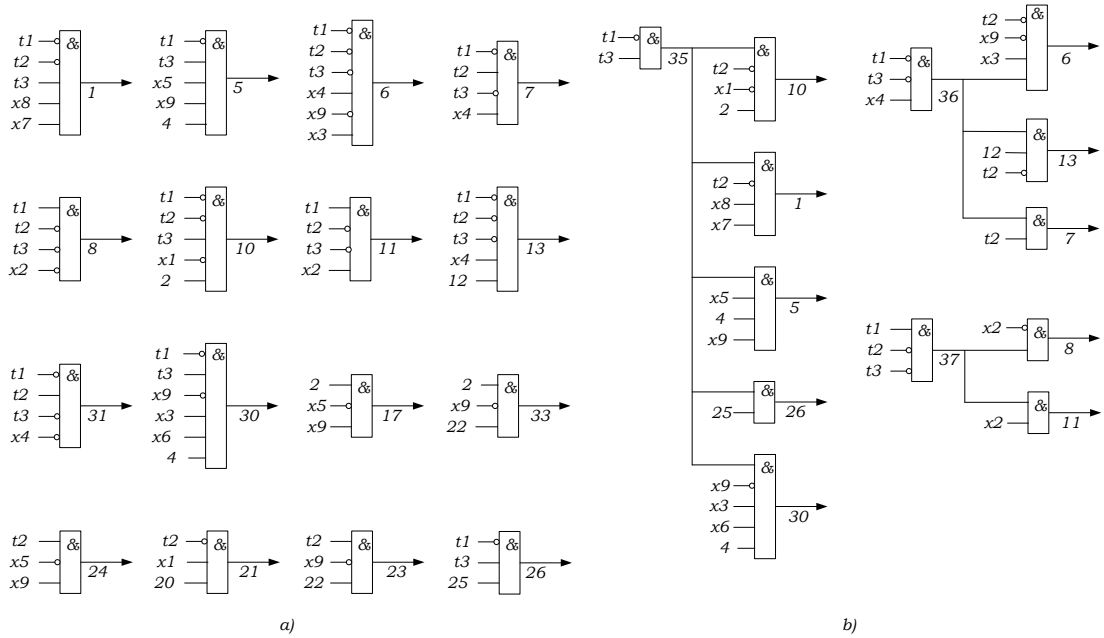

Figure 27. Term decomposition in our example

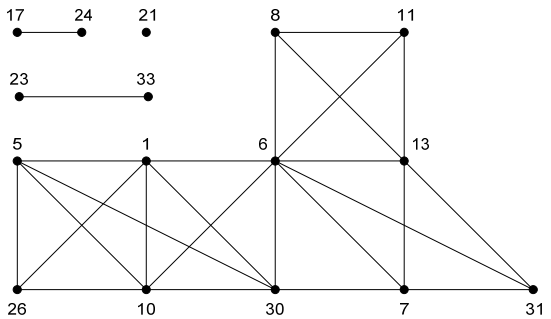

Figure 28. The graph G_ω of relation ω

Step 9. Construct OR-gate for each output variable y_n (n = 1, …, 15 in our example) and for each input memory function d_r (r = 1, …, 3 in our example) which occur more than once in the circuit after step 8. If we look at Fig. 29 we will find that several outputs appear more than once in this circuit. For example, y_2 is written at the outputs of gates AND_6 and AND_{10}, d_2 is written at the outputs AND_{11}, AND_{13}, AND_{30} and AND_{31}. It is evident that FSM has only one output y_2, so, first – the circuit in Fig. 29 is not the final circuit and, second – output y_2 will be equal to one when the output of AND_6 or the output of AND_{10} are equal to one. Thus, for y_2 and for each output that appears more than once in Fig. 29, we should construct OR-gate with the inputs connected to the outputs of gates where these signals are written.

To formalize this process we constructed Table 10 where each row contains the list of gates for each output. Now we can immediately construct OR-gates for the outputs which occur more than once in the logic circuit (Fig. 30,a).

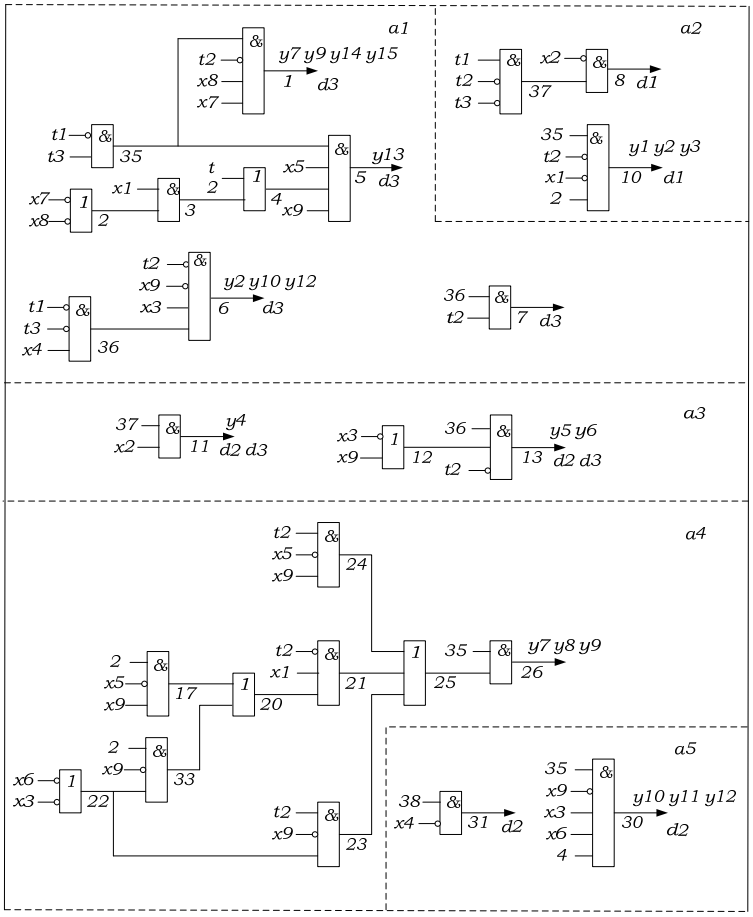

Figure 29. Logic circuit after term decomposition

Table 10. Gates for outputs

Outputs	*Gates*	*Outputs*	*Gates*	*Outputs*	*Gates*
y1	*e10*	*y7*	*e1 e26*	*y13*	*e5*
y2	*e6 e10*	*y8*	*e26*	*y14*	*e1*
y3	*e10*	*y9*	*e1 e26*	*y15*	*e1*
y4	*e11*	*y10*	*e6 e30*	*d1*	*e8 e10*
y5	*e13*	*y11*	*e30*	*d2*	*e11 e13 e30 e31*
y6	*e13*	*y12*	*e6 e30*	*d3*	*e1 e5 e6 e7 e11 e13*

Step 10. Find equal OR-gates among the gates constructed at step 9. Leave only one gate in each set of equal gates. The logic circuit after removal of equal gates is shown in Fig. 30,b.

Step 11. Find the inclusion of OR-gates into other gates among the gates constructed at steps 10. Unfortunately, we do not have any such cases in our rather simple example.

Step 12. Make term decomposition for all OR-gates. Just as at Step 8 for AND-gates, we consider here only the gates with not less than three inputs since the minimal number of inputs in a common term in term decomposition is equal to two (after removal equal gates and full inclusion). Similar to step 8, we should construct the

graph of the relation ω for OR-gates. The problem of term decomposition for OR-gates is divided into as many subproblems as the number unconnected components in the graph of ω. In our rather simple example, we have only two OR-gates with more than two inputs (see *d2* and *d3* in Fig. 30,b).

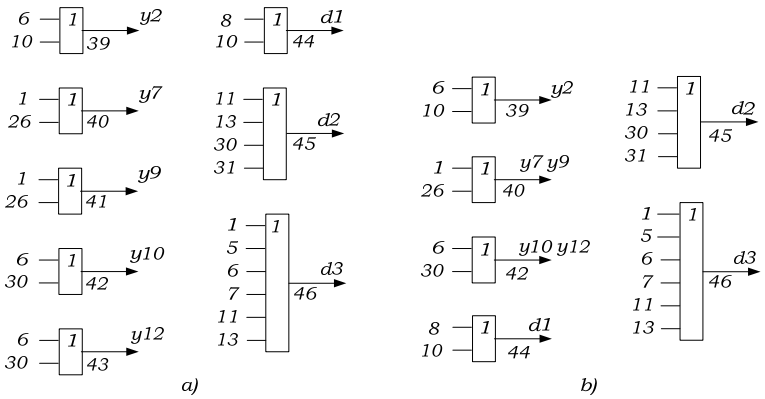

Figure 30. OR-gates before (a) and after (b) removal equal gates

The final logic circuit is shown in Fig. 31. We have placed the circuit from Fig. 30,b at the bottom of Fig. 31. Of course, we should remove appearances of the outputs y_2, y_7, y_9, y_{10}, y_{12} and input memory functions d_1, d_2, d_3 from other parts of the logic circuit. Thus, only the outputs that have one entry in the column "*Gates*" of Table 10 will be in the part of the circuit that is above the "*OR for outputs and input memory functions*" in Fig. 31.

Step 13. Relax and drink your coffee.

Really, the reason, that we cannot demonstrate gate inclusion and term decomposition for OR-gates, can be explained not only by the simplicity of our example, but also a very effective optimization at the previous steps that allows to decrease the number of inputs in the most gates of our circuit. To overcome some dissatisfaction of our last steps, let us discuss one more example, from the synthesis another FSM, presented in Table 11 and Fig. 32 with OR-gates for output variables and input memory functions which occur more than once in the circuit after Step 9.

Table 11. Gates for outputs in one more example

Outputs	*Gates*	*Outputs*	*Gates*
y1	*e5 e31 e36 e39*	*y8*	*e4 e44*
y2	*e3 e5 e36*	*y9*	*e16 e30*
y3	*e39 e56 e60*	*y10*	*e3 e16 e43*
y4	*e4 e31 e36 e42*	*d1*	*e56 e60*
y5	*e3 e16 e43*	*d2*	*e3 e4 e31 e36 e39 e44*
y6	*e16 e30*	*d3*	*e3 e4 e31 e42 e43*
y7	*e16 e44 e56*	*d4*	*e3 e4 e5 e42 e43 e44 e56*

Step 10a. From Table 11 or Fig. 32 we immediately get that $OR_{75} = OR_{68}$ and $OR_{76} = OR_{67}$. We remove OR_{75} and OR_{76} and get y_9 together with y_6 from OR_{68} and y_{10} together with y_5 from OR_{67} (Fig. 33).

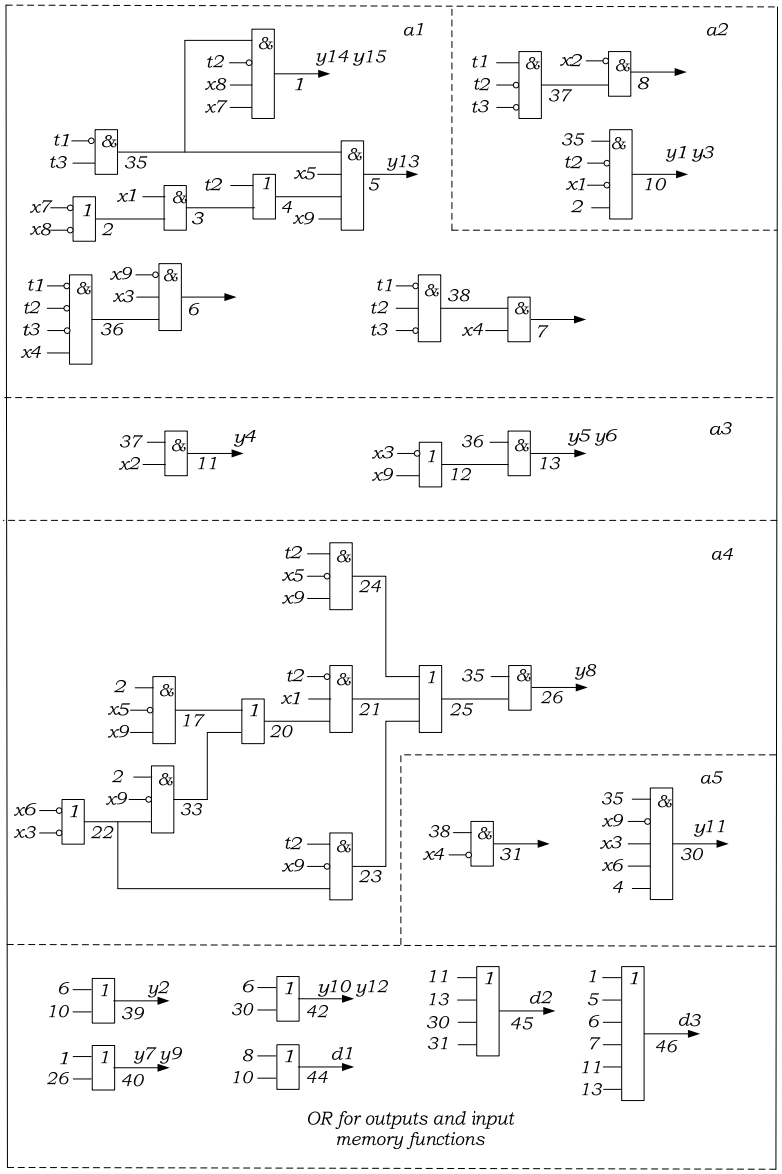

Figure 31. The final logic circuit

Step 11a. We checked full inclusion for OR gates and found that $OR_{70} \subset OR_{72}$, $OR_{70} \subset OR_{74}$ and $OR_{71} \subset OR_{65}$. The circuit after this step is presented in Fig. 34.

Step 12a. For term decomposition, we constructed the graph of relation ω for OR-gates with three and more inputs (Fig. 35). The problem of term decomposition for OR-gates is divided into as many subproblems as the number of unconnected components in the graph of ω. In our example we have two subgraphs and one of them is nontrivial.

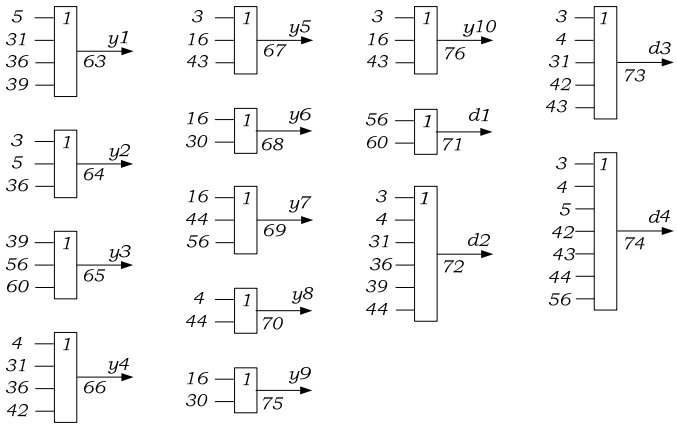

Figure 32. OR-gates for y_n and d_r in one more example

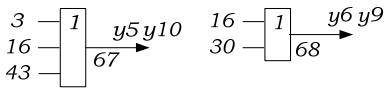

Figure 33. OR-gate transformation after removal of equal gates

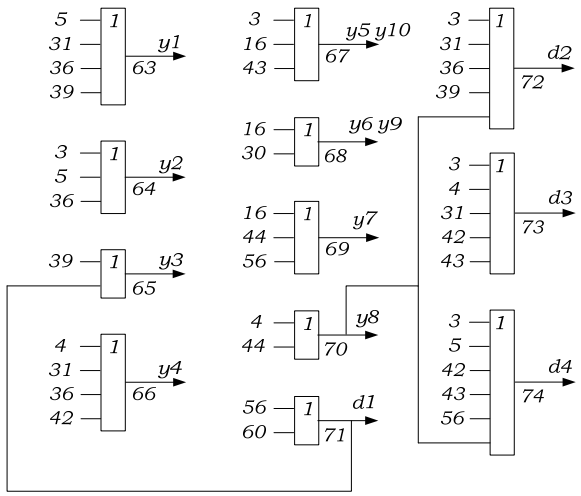

Figure 34. OR-gates after full inclusion

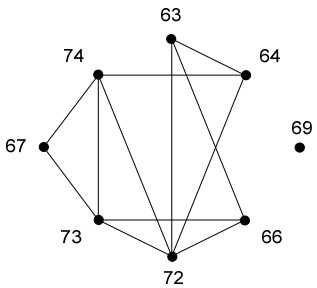

Figure 35. The graph of relation ω for OR-gates

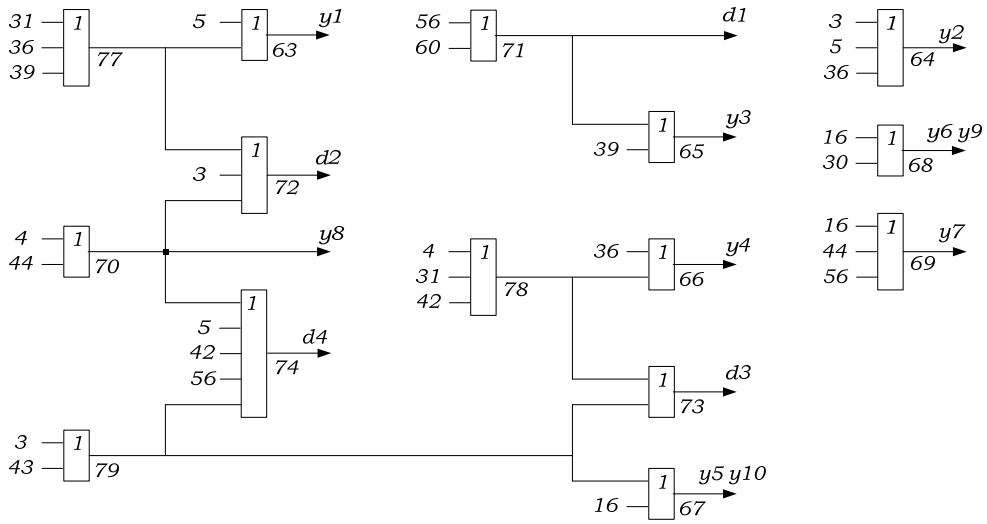

Figure 36. Logic circuit after term decomposition

The logic circuit after term decomposition is presented in Fig. 36. Its cost is 10 inputs lower than in the initial circuit in Fig. 32.

Chapter 6 Transformation of Algorithmic State Machines

In this Chapter, we consider two more representations of Algorithmic State Machine – System of transition formulae and Matrix scheme of algorithms. After this, we will discuss transformation of Algorithmic State Machine – minimization of conditional and operator vertices and combining of Algorithmic State Machines. We will use these transformations in the next chapter dedicated to the high level synthesis of digital systems.

6.1 Various representations of Algorithmic State Machine

6.1.1 System of transition formulae. As an example, we use ASM Γ in Fig. 1. Let us look at operators following operator Y_b. The facts that operator Y_3 is implemented after Y_b when $x_1x_4x_3 = 1$, operator Y_1 is implemented after Y_b when $x_1x'_4 = 1$ and operator Y_5 is implemented after Y_b when $x_1x_4x'_3 = 1$ or $x'_1 = 1$, can be represented as the formula

$$Y_b \rightarrow x_1x_4x_3Y_3 + x_1x_4x'_3Y_5 + x_1x'_4Y_1 + x'_1Y_5. \qquad (1)$$

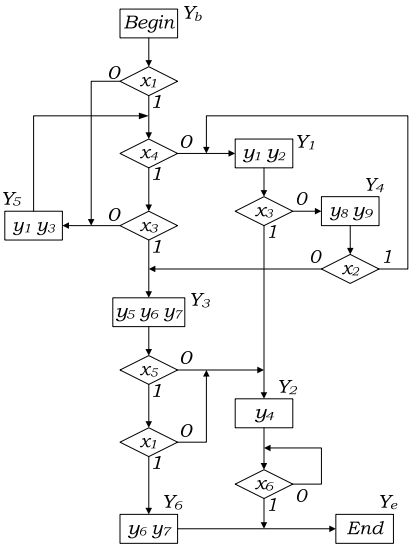

Figure 1. ASM Γ

In general, the transition formula for the operator Y_i is

$$Y_i \rightarrow \sum_{t=1}^{T+1} \alpha_{it} Y_t \ , \quad i = b, 1, \ldots, T.$$

Here $Y_{T+1} = Y_e$ is the operator, corresponding to the final vertex "*End*" and a_{it} is the *transition function* from Y_i to Y_t (see Section 4.1.2 in Chapter 4). The term $a_{it}Y_t$ is equal to Y_t if $a_{it} = 1$, and is equal to 0 if $a_{it} = 0$.

The transition formulae can be transformed according to the rules of Boolean algebra and the following additional rules:

1. $(\alpha + \beta)Y_i = \alpha Y_i + \beta Y_i$;
2. $\alpha\beta Y_i + \alpha\gamma Y_j = \alpha(\beta Y_i + \gamma Y_j)$;
3. If $\alpha = \beta$ then $\alpha Y_i = \beta Y_i$.

Here a, β and γ are Boolean functions and Y_i and Y_j are operators. The set of transition formulae for all $i = b, 1, 2, \ldots, T$ is called the *system of transition formulae*. This system contains seven formulae for ASM Γ in Fig. 1:

$$\begin{aligned}
&Y_b \rightarrow x_1x_4x_3Y_3 + x_1x_4x'_3Y_5 + x_1x'_4Y_1 + x'_1Y_5;\\
&Y_1 \rightarrow x_3Y_2 + x'_3Y_4;\\
&Y_2 \rightarrow x_6Y_e;\\
&Y_3 \rightarrow x_5x_1Y_6 + x_5x'_1Y_2 + x'_5Y_2;\\
&Y_4 \rightarrow x_2Y_1 + x'_2Y_3;\\
&Y_5 \rightarrow x_4x_3Y_3 + x_4x'_3Y_5 + x'_4Y_1;\\
&Y_6 \rightarrow Y_e.
\end{aligned} \tag{2}$$

The following representation of the transition formula

$$Y_i \rightarrow x_mA + x'_mB \tag{3}$$

is called the *expansion of transition formula* Y_i by variable x_m. Here x_m is one of the logical conditions and transition subformulae A and B do not depend on x_m. For example, let us expand formula (1) by variable x_1:

$$Y_b \rightarrow x_1(x_4x_3Y_3 + x_4x'_3Y_5 + x'_4Y_1) + x'_1Y_5\,. \tag{4}$$

If some terms of the transition formula do not depend on x_m, then these terms must be multiplied by expression $(x_m + x'_m)$ before the expansion of the transition formula by x_m. Thus, any transition formula may be expanded by any variable. For example, expanding the transition formula (1) by the variable x_4 we obtain

$$\begin{aligned}
&Y_b \rightarrow x_1x_4x_3Y_3 + x_1x_4x'_3Y_5 + x_1x'_4Y_1 + x'_1(x_4 + x'_4)Y_5 =\\
&= x_4(x_1x_3Y_3 + x_1x'_3Y_5 + x'_1Y_5) + x'_4(x_1Y_1 + x'_1Y_5).
\end{aligned} \tag{5}$$

In expression (3), we can continue expanding subformulae A and B by other variables until the terms in the internal brackets are as follows:

$$(x_pY_m + x'_pY_n) \tag{6}$$

where $Y_m, Y_n \in \{Y_1, \ldots, Y_T, Y_{T+1}\}$. This expression corresponds to subgraph G_1 in Fig. 2. The resulting transition formula and the system of such formulae are called the *bracket transition formula* and the *system of transition formulae in the bracket form* correspondingly.

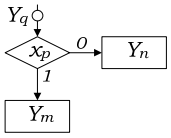

Figure 2. Subgraph G_1 corresponding to expression (6)

Let us continue to expand the transition formula for Y_b in expression (4):

$$Y_b \rightarrow x_1(x_4(x_3Y_3 + x'_3Y_5) + x'_4Y_1) + x'_1Y_5. \tag{7}$$

Expression (7) presents the transition formula for Y_b in a bracket form. It is easy to show that there is a one-to-one correspondence between the bracket transition formula and the ASM subgraph obtained from this formula. To illustrate this,

consider the construction of *ASM* subgraph G_2 (Fig. 3,a) for transition formula Y_b in (7).

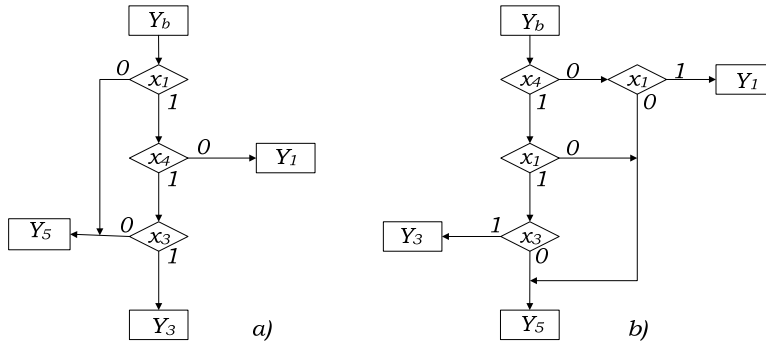

Figure 3. Subgraphs G_2 and G_3 corresponding to expansions (7) and (8)

First, draw operator vertex Y_b. Then construct conditional vertex with x_1 and link its input with the output of operator Y_b, since first we expanded transition formula Y_b with this variable x_1. Next, construct conditional vertex x_4 and link its input with the output *"1"* of the preceding conditional vertex x_1, since the expansion by the variable x_4 is implemented in brackets just after x_1. Further, construct operator vertex Y_5 and link its input with the output *'0'* of conditional vertex x_1, as there are no expansions by any variables after x'_1. In exactly the same way construct conditional vertex x_3, link its input with the output *"1"* of vertex x_4, and link the zero output of vertex x_4 with operator vertex Y_1 etc. Thus, at each step we have no alternative and can construct only one subgraph for a given bracket transition formula.

The ASM subgraph G_3 in Fig. 3,b is constructed in a similar way for another expansion (8) of the same unexpanded transition formula Y_b in (1). We changed the order of variables in this expansion:

$$Y_b \to x_4(x_1x_3Y_3 + x_1x'_3Y_5 + x'_1Y_5) + x'_4(x_1Y_1 + x'_1Y_5) =$$
$$= x_4(x_1(x_3Y_3 + x'_3Y_5) + x'_1Y_5) + x'_4(x_1Y_1 + x'_1Y_5). \qquad (8)$$

Thus, to obtain a ASM subgraph from a bracket transition formula it is sufficient to construct chains of conditional vertices in accordance with a sequence of the transition formula expansion. As seen from Fig. 3,a and Fig. 3,b, the number of conditional vertices in the subgraph for some transition formula depends on the order of the expansion of this transition formula. Now it is evident that to transit from a system of transition formulae to ASM it is necessary to expand this system to the bracket form and to construct the ASM subgraph for each bracket transition formula.

We will show now that the number of conditional vertices in the ASM, thus constructed, depends not only on the separate expansion of each transition formula but also on their joint expansion. Let us rewrite here expression (7) and expand transition formula for Y_5 beginning from variable x_4:

$$Y_b \to x_1\underline{(x_4(x_3Y_3 + x'_3Y_5)} + x'_4Y_1) + x'_1Y_5; \qquad (7)$$

$$Y_5 \to \underline{x_4(x_3Y_3 + x'_3Y_5)} + x'_4Y_1; \qquad (9)$$

For example, let us construct ASM subgraph G_4 for bracket transition formulae Y_b and Y_5 in (7) and (9). The identical subformulae are underlined in these expressions. Obviously, one subgraph corresponds to the same subformulae so only one subgraph is constructed for these subformulae (Fig. 4,a).

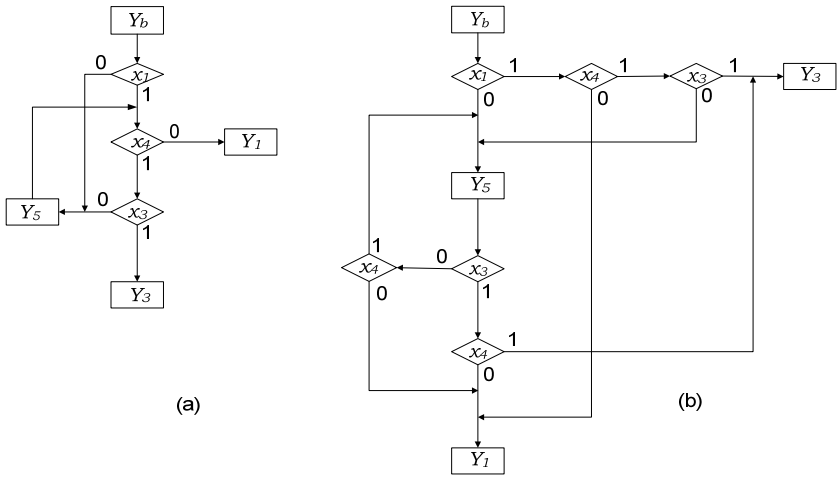

Figure 4. Subgraphs G_4 (a) and G_5 (b) corresponding to expansions (7, 9) and (7, 10)

Now we construct the ASM subgraph for the same bracket transition formula for Y_b (7) and the new bracket transition formula Y_5, obtained by its expansion beginning with variable x_3:

$$Y_5 \to x_3(x_4Y_3 + x'_4Y_1) + x'_3(x_4Y_5 + x'_4Y_1). \qquad (10)$$

Since there are no identical subformulae in bracket transition formulae (7) and (10) we derive subgraph G_5 (Fig. 4,b) with three additional conditional vertices, compared to subgraph G_4 in Fig. 4,a.

At the end of this section, we expand the system of transition formulae, presented in (2):

$$\begin{aligned}
&Y_b \to \underline{x_1(x_4(x_3Y_3 + x_3'Y_5) + x_4'Y_1)} + x_1'Y_5;\\
&Y_1 \to x_3Y_2 + x_3'Y_4;\\
&Y_2 \to x_6Y_e;\\
&Y_3 \to x_5(x_1Y_6 + x_1'Y_2) + x_5'Y_2; \qquad (11)\\
&Y_4 \to x_2Y_1 + x_2'Y_3;\\
&Y_5 \to \underline{x_4(x_3Y_3 + x_3'Y_5) + x_4'Y_1};\\
&Y_6 \to Y_e.
\end{aligned}$$

6.1.2. Matrix schemes of algorithms. The matrix scheme of algorithm (MSA) is a square matrix with rows $Y_b, Y_1, \ldots, Y_T$, and columns $Y_1, Y_2, \ldots, Y_T, Y_e$. We write the transition function from operator Y_i to operator Y_j at the intersection of row Y_i and column Y_j in this matrix. The MSA M for ASM Γ from Fig. 1 is presented in Table 1. For simplicity, we do not write the transition functions that are equal to zero in the matrix scheme of algorithm.

The transition from ASM to MSA is obvious. We find all transition functions in ASM and place them into the corresponding entries of MSA. To transit from MSA to ASM we should obtain a system of transition formulae, a system of bracket transition formulae and then ASM. Just as for ASM (Section 4.1.3), it is possible to define an execution of MSA and a value of MSA for any sequence of vectors of logical conditions.

Table 1. MSA M

	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_e
Y_b	$x_1x'_4$		$x_1x_4x_3$		$x_1x_4x'_3$ x'_1		
Y_1		x_3		x'_3			
Y_2							x_6
Y_3		$x_5x'_1$ x'_5				x_5x_1	
Y_4	x_2		x'_2				
Y_5	x'_4		x_4x_3		$x_4x'_3$		
Y_6							*1*

6.2 Minimization of conditional vertices in Algorithmic state machines

In this section, we will discuss minimization of conditional vertices in ASMs. The minimization algorithm consists of three steps. In the first step, the initial ASM is divided into such subgraphs $G^1,...,G^Q$, that for obtaining the minimal ASM it is sufficient to minimize the number of conditional vertices in each subgraph independently of one another. In the second step, for each subgraph $G^q(q=1,...,Q)$ we will find a set of equivalent ones that contains the minimized subgraph $G^q_{\min}$. In the third step, the subgraph $G^q_{\min}$ will be obtained by solving the covering problem on the set of subgraphs found in the second step.

6.2.1 ASM partitioning into subgraps. As an example for ASM minimization, we will use ASM Γ_{nonmin} in Fig. 5. To partition ASM, choose *any* operator, except the final one Y_e, from the set of operators $\{Y_b, Y_1,...,Y_T, Y_e\}$ and find the set of operators towards which there is a path

$$Y_i \tilde{x}_{i1}...{}_i\, \tilde{x}_{iR} Y_j \qquad (12)$$

from operator vertex Y_i (i = *b,1,2, …,T*) passing only through conditional vertices with the logical conditions $x_{i1},...,x_{iR}$. Here, as before, $\tilde{x}_{ir} = x_{ir}$ if the path proceeds through the conditional vertex with x_{ir} via output '1' $(r=1,...,R)$ and $\tilde{x}_{ir} = x'_{ir}$ if the path proceeds through the conditional vertex via output '0'. Since we can start from any operator, let us begin with the initial operator Y_b. For ASM Γ_{nonmin} in Fig. 5, there are paths to Y_1, Y_4, Y_5 from Y_b. The arrows from operator Y_b (Fig. 6,a) designate these paths. Next, go to the right side of this subgraph and find the operators, excluding Y_b, from which there are paths (12) leading to operators Y_1, Y_4, Y_5. We place these operators (Y_2, Y_3) under Y_b on the left side. For all operators on the left, continue finding such operators to which there are paths (12), etc., until the set on the left (A^1) and that on the right (B^1) are no longer increasing. Next, choose a new operator

$Y_m \notin A^1$ (Y_1 in our example) and construct (A^2) and (B^2) in a similar way (see Fig. 6,b). The algorithm consists of Q steps, $Q \le T+1$, where T is the number of operator vertices.

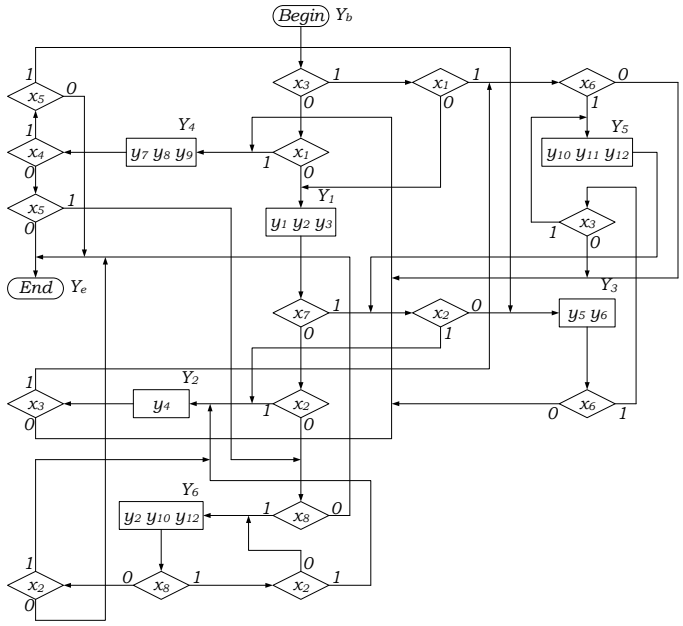

Figure 5. Nonminimal ASM Γ_{nonmin}

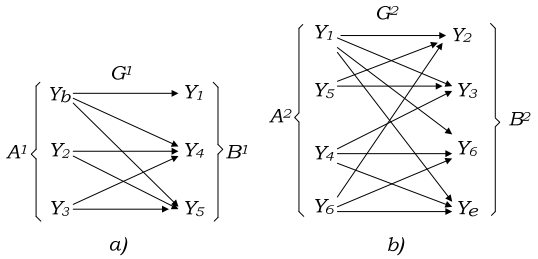

Figure 6. Partition ASM Γ_{nonmin}

In the q th step $(q = 2,\ldots,Q)$, choose a vertex $Y_f \notin \bigcup_{i=1}^{q-1} A^i$, and continue constructing A^q and B^q etc. until, after Q steps, it is impossible to find an operator, which is not included in the sets $A^1,\ldots,A^Q$. As a result, ASM is divided into subgraphs $G^1, \ldots, G^Q$ where the subgraph $G^q(A^q, B^q, P^q)$ is defined by sets A^q, B^q, P^q and by the arcs between the vertices of these sets. Here P^q is the set of conditional vertices between operators of the sets A^q and B^q. Two subgraphs G^1 and G^2 constructed by partition of ASM Γ_{nonmin} in Fig. 5 is shown in Fig. 7.

From the partition of ASM it follows:

1. For any two subgraphs G^1 and G^2:

$$A^i \cap A^j = \varnothing \; ; \; B^i \cap B^j = \varnothing ; \;\; P^i \cap P^j = \varnothing .$$

2. The partition is independent of the choice of the first operator at every step.

Since the partition is unique and the sets of conditional vertices in different subgraphs do not intersect ($P^i \cap P^j = \varnothing$), the following statement results immediately:

$$R = \sum_{q=1}^{Q} R^q \; .$$

Here R is the number of conditional vertices in ASM and R^q is the number of conditional vertices in the subgraph G^q *(q = 1, …, Q)*. In one of my previous works I have proved that if ASM is partitioned into subgraphs $G^1, \ldots, G^q$ and the number $R^q_{\min}$ of conditional vertices is minimal in each subgraph G^q *(q = 1, …, Q)*, then the number of conditional vertices in ASM is also minimal and equal to

$$R_{\min} = \sum_{q=1}^{Q} R^q_{\min} .$$

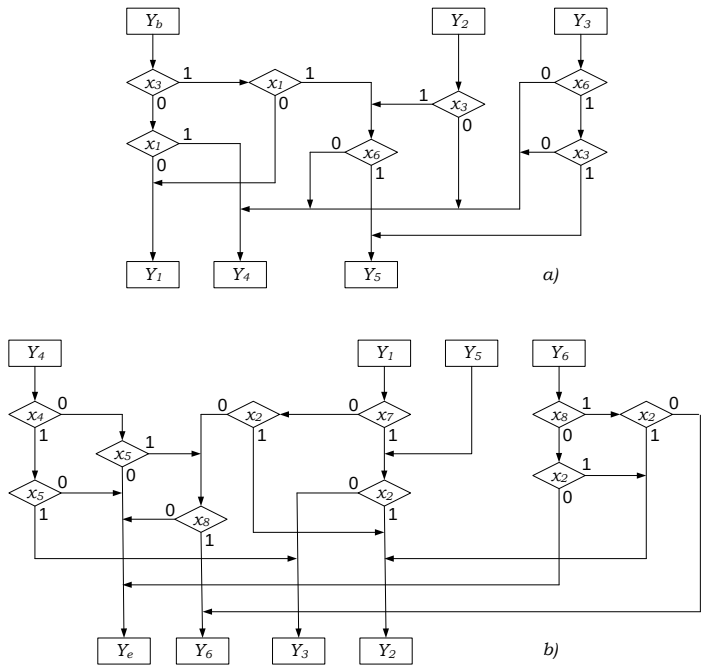

Figure 7. Two subgraphs G^1 and G^2 as a result of partition of ASM Γ_{nonmin}

From the last statement, it follows that the problem of expansion for the system of transition formulae can be also divided into Q independent subproblems in accordance with the introduced partition of ASM. In our example for ASM Γ_{nonmin}, we have two such subsystems:

1. $$Y_b \to x_3 x_1 x_6 Y_5 + x_3 x_1 x'_6 Y_4 + x_3 x'_1 Y_1 + x'_3 x_1 Y_4 + x'_3 x'_1 Y_1 ;$$
$$Y_2 \to x_3 x_6 Y_5 + x_3 x'_6 Y_4 + x'_3 Y_4 ; \qquad (13)$$
$$Y_3 \to x_6 x_3 Y_5 + x_6 x'_3 Y_4 + x'_6 Y_4 .$$

2. $Y_1 \to x_7 x_2 Y_2 + x_7 x'_2 Y_3 + x'_7 x_2 Y_2 + x'_7 x'_2 x_8 Y_6 + x'_7 x'_2 x'_8 Y_e;$
$Y_4 \to x_4 x_5 Y_3 + x_4 x'_5 Y_e + x'_4 x_5 x_8 Y_6 + x'_4 x_5 x'_8 Y_e + x'_4 x'_5 Y_e;$ (14)
$Y_5 \to x_2 Y_2 + x'_2 Y_3;$
$Y_6 \to x_8 x_2 Y_2 + x_8 x'_2 Y_6 + x'_8 x_2 Y_2 + x'_8 x'_2 Y_e.$

6.2.2 Constructing a set of equivalent subgraphs. At this stage, we expand each subsystem of transition formulae, obtained at the previous stage, into the bracket form. To minimize a subgraph of ASM corresponding to the subsystem of transition formulae, M versions of expansion for each transition formula Y_t should be constructed:

$$Y_t \to x_{im} A_m + x'_{im} B_m. \quad (15)$$

Here $x_{i1}, \ldots, x_{iM}$ are the first M variables contained maximal times in Y_t. Then, we again find M variables x_{nm} and x_{pm} which occur maximal times in A_m and B_m. After that, we continue expansion of A_m and B_m with x_{nm} and x_{pm}:

$$Y_t \to x_{im}(x_{nm} C_{nm} + x'_{nm} D_{nm}) + x'_{im}(x_{pm} E_{pm} + x'_{pm} F_{pm}).$$

Continuing expansion we get not more than $M^{2^K - 1}$ version of expansion, here K is the number of expansion levels. In our experiments with very complicated ASMs, we have seen that it is sufficient to take $M = K = 3$ so as not to lose the minimal subgraph of ASM.

Now we return to our example and begin with transition formula Y_b from the first subsystem (13). Let us find how many times each variable occurs in the transition formula for Y_b:

x_1 – 5 times; x_3 – 5 times; x_6 – 2 times.

In our simple example we use $M = 2$, that is we make four expansions for Y_b – two starting from x_1 and two starting from x_3 (two variables with the maximal number of appearances in transition formula Y_b). Let us begin with x_1:

$$Y_b \to x_1(x_3 x_6 Y_5 + x_3 x'_6 Y_4 + x'_3 Y_4) + x'_1 Y_1.$$

Since the expansion after x'_1 is impossible, find how many times each variable occurs in the transition formula after x_1:

x_3 – 3 times; x_6 - 2 times.

In our example, we choose x_3, x_6 and get two expansion versions with x_1 in the first step. First of them is

$$Y_b \to x_1(x_3(x_6 Y_5 + x'_6 Y_4) + x'_3 Y_4) + x'_1 Y_1. \quad (16)$$

The result of the second expansion with x_6 after x_1 is:

$$Y_b \to x_1(x_6(x_3 Y_5 + x'_3 Y_4) + x'_6 Y_4) + x'_1 Y_1. \quad (17)$$

Now let us return to Fig. 2. In this figure, we marked the input of the conditional vertex with x_p by operator Y_q. The following transition formula

$$Y_q \to x_p Y_m + x'_p Y_n$$

corresponds to operator Y_q in Fig. 2. We call Y_q a *derivative operator* and denote it by a circle to distinguish it from primary rectangular operators $Y_b, Y_1, \ldots, Y_T, Y_e$ in the graph of ASM. For expansion (16) we can write the following derivative operators beginning with the internal brackets:

$$Y_{10} \to x_6 Y_5 + x'_6 Y_4;$$
$$Y_{11} \to x_3 Y_{10} + x'_3 Y_4;$$
$$Y_{12} \to x_1 Y_{11} + x'_1 Y_1.$$

Derivative operator Y_{12} coincides with operator Y_b. We write the numbers of derivative operators over the corresponding opening brackets in (16):

$$Y_b \to \overset{12}{x_1}(\overset{11}{x_3}(\overset{10}{x_6 Y_5 + x'_6 Y_4}) + x'_3 Y_4) + x'_1 Y_1 \qquad (18)$$

and list these operators in the special Table 2. Operators Y_{10}, Y_{11}, Y_{12} are the first three operators in this table (we use number t instead of Y_t in Table 2).

Table 2. The derivative operators for Y_b, Y_2, Y_3

10	*x6*	*5*	*4*	*16*	*x1*	*4*	*1*
11	*x3*	*10*	*4*	*17*	*x1*	*10*	*1*
12	*x1*	*11*	*1*	*18*	*x3*	*17*	*16*
13	*x3*	*5*	*4*	*19*	*x1*	*5*	*1*
14	*x6*	*13*	*4*	*20*	*x6*	*19*	*16*
15	*x1*	*14*	*1*	*21*	*x3*	*20*	*16*

It is obvious that there is a one-to-one correspondence between the bracket transition formula and the ASM subgraph obtained from this formula. To illustrate this, consider the construction of ASM subgraph (Fig. 8) for transition formula Y_b in (18). We see at once that such a construction is reduced to the successive exposure of derivative operators Y_{12}, Y_{11}, Y_{10} from Table 2. It is possible to present expansion (18) or the subgraph in Fig. 8 as the first column in Table 3 (*v1* means version1). The number of derivative operators in this column is equal to the number of conditional vertices in Fig. 8. For now, disregard the parenthesis in the columns of this table.

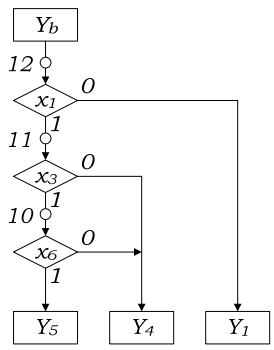

Figure 8. Subgraph for expression (18)

Table 3. The table of versions for Y_b, Y_2, Y_3

Y_b				Y_2		Y_3	
v1	*v2*	*v3*	*v4*	*v5*	*v6*	*v7*	*v8*
12	*15*	*18*	*21*	*(11)*	*(14)*	*(14)*	*(11)*
(11)	*(14)*	*17*	*20*	*(10)*	*(13)*	*(13)*	*(10)*
(10)	*(13)*	*16*	*19*				
		(10)	*16*				

Exactly in the same way, we implement expansion for each version of Y_b, Y_2, Y_3:

$$Y_b \overset{15}{\to} x_1 \overset{14}{(} x_6 \overset{13}{(} x_3 Y_5 + x'_3 Y_4) + x'_6 Y_4) + x'_1 Y_1;$$

$$Y_b \overset{18}{\to} x_3 \overset{17}{(} x_1 \overset{10}{(} x_6 Y_5 + x'_6 Y_4) + x'_1 Y_1) + x'_3 \overset{16}{(} x_1 Y_4 + x'_1 Y_1);$$

$$Y_b \overset{21}{\to} x_3 \overset{20}{(} x_6 \overset{19}{(} x_1 Y_5 + x'_1 Y_1) + x'_6 \overset{16}{(} x_1 Y_4 + x'_1 Y_1)) + x'_3 \overset{16}{(} x_1 Y_4 + x'_1 Y_1);$$

$$Y_2 \overset{11}{\to} x_3 \overset{10}{(} x_6 Y_5 + x'_6 Y_4) + x'_3 Y_4;$$

$$Y_2 \overset{14}{\to} x_6 \overset{13}{(} x_3 Y_5 + x'_3 Y_4) + x'_6 Y_4;$$

$$Y_3 \overset{14}{\to} x_6 \overset{13}{(} x_3 Y_5 + x'_3 Y_4) + x'_6 Y_4;$$

$$Y_3 \overset{11}{\to} x_3 \overset{10}{(} x_6 Y_5 + x'_6 Y_4) + x'_3 Y_4.$$

Having these expressions, we can complete the *table for derivative operators* (Table 2) and the *table of versions* (Table 3). The last table contains four versions for implementation of transition formula Y_b, two versions for transition formula Y_2 and two versions for transition formula Y_3. In Table 3, the derivative operator for Y_j is enclosed in brackets if it appears at least in one version $Y_f (f \neq j)$. Obviously, having chosen one version for each Y_b, Y_2, Y_3 we obtain the subgraph of ASM for the first subsystem of transition formulae. The number of conditional vertices in such a subgraph is equal to the number of derivative operators (excluding similar ones) in the chosen versions. For example, having chosen versions v_2, v_5 and v_7, we will obtain the subgraph with five conditional vertices (Fig. 9). *To obtain the subgraph with a minimal number of conditional vertices in the case of* m *operators* $Y_1, \ldots, Y_m$, *we have to choose one version for each* Y_j *so, that the total number of derivative operators should be minimal.*

We call this problem the problem of finding a *minimal cover for the table of versions*. The total number of possible subgraphs that can be constructed from such a table is equal to $R_1 \times \ldots \times R_m$. Here R_j is the number of versions for operator Y_j. In our simple example (Table 3), this number is equal to $4 \times 2 \times 2 = 16$.

Without detailed explanations, we give expansions for the second subsystem (14) of transition formulae, the list of derivative operators (Table 4) and the table of versions (Table 5):

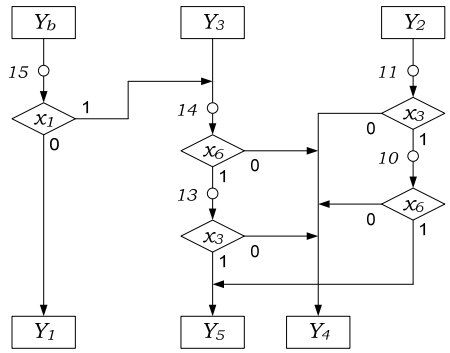

Figure 9. Subgraph for versions v_2, v_5 and v_7

$$Y_1 \to x_2 Y_2 + x'_2 \overset{31}{(} x_7 Y_3 + x'_7 \overset{30}{(} x_8 Y_6 + x'_8 Y_e));$$

$$Y_1 \to x_2 Y_2 + x'_2 \overset{35}{(} x_8 \overset{34}{(} x_7 Y_3 + x'_7 Y_6) + x'_8 \overset{33}{(} x_7 Y_3 + x'_7 Y_e));$$

$$Y_1 \to \overset{39}{x_7} \overset{38}{(} x_2 Y_2 + x'_2 Y_3) + x'_7 \overset{37}{(} x_2 Y_2 + x'_2 \overset{30}{(} x_8 Y_6 + x'_8 Y_e));$$

$$Y_1 \to \overset{43}{x_7} \overset{38}{(} x_2 Y_2 + x'_2 Y_3) + x'_7 \overset{42}{(} x_8 \overset{41}{(} x_2 Y_2 + x'_2 Y_6) + x'_8 \overset{40}{(} x_2 Y_2 + x'_2 Y_e));$$

$$Y_4 \to \overset{45}{x_5} \overset{44}{(} x_4 Y_3 + x'_4 \overset{30}{(} x_8 Y_6 + + x'_8 Y_e)) + x'_5 Y_e;$$

$$Y_4 \to \overset{49}{x_5} \overset{48}{(} x_8 \overset{47}{(} x_4 Y_3 + x'_4 Y_6) + x'_8 \overset{46}{(} x_4 Y_3 + x'_4 Y_e)) + x'_5 Y_e;$$

$$Y_4 \to \overset{52}{x_4} \overset{51}{(} x_5 Y_3 + x'_5 Y_e) + x'_4 \overset{50}{(} x_5 \overset{30}{(} x_8 Y_6 + x'_8 Y_e) + x'_5 Y_e);$$

$$Y_4 \to \overset{55}{x_4} \overset{51}{(} x_5 Y_3 + x'_5 Y_e) + x'_4 \overset{54}{(} x_8 \overset{53}{(} x_5 Y_6 + x'_5 Y_e) + x'_8 Y_e);$$

$$Y_5 \to \overset{38}{x_2} Y_2 + x'_2 Y_3;$$

$$Y_6 \to \overset{37}{x_2} Y_2 + x'_2 \overset{30}{(} x_8 Y_6 + x'_8 Y_e);$$

$$Y_6 \to \overset{42}{x_8} \overset{41}{(} x_2 Y_2 + x'_2 Y_6) + x'_8 \overset{40}{(} x_2 Y_2 + x'_2 Y_e).$$

Table 4. The derivative operators for Y_1, Y_4, Y_5 and Y_6

30	x8	6	e	39	x7	38	37	48	x8	47	46
31	x7	3	30	40	x2	2	e	49	x5	48	e
32	x2	2	31	41	x2	2	6	50	x5	30	e
33	x7	3	e	42	x8	41	40	51	x5	3	e
34	x7	3	6	43	x7	38	42	52	x4	51	50
35	x8	34	33	44	x4	3	30	53	x5	6	e
36	x2	2	35	45	x5	44	e	54	x8	53	e
37	x2	2	30	46	x4	3	e	55	x4	51	54
38	x2	2	3	47	x4	3	6				

6.2.3 Finding a minimal cover for the table of versions. We will show that a table of version may be compressed without a loss of the minimal cover. On the set of versions $\{v_{j1},\ldots,v_{jR_j}\}$ for operator Y_j, we define the partial ordering relation. Assume

Table 5. The table of versions for Y_1, Y_4, Y_5 and Y_6

Y_1				Y_4				Y_5	Y_6	
$\upsilon 1$	$\upsilon 2$	$\upsilon 3$	$\upsilon 4$	$\upsilon 5$	$\upsilon 6$	$\upsilon 7$	$\upsilon 5$	$\upsilon 6$	$\upsilon 7$	$\upsilon 8$
32	36	39	43	45	49	52	55	(38)	(37)	(42)
31	35	(38)	(38)	44	48	51	51		(30)	(41)
(30)	34	(37)	(42)	(30)	47	50	54			(40)
	33	(30)	(41)		46	(30)	53			
			(40)							

that $\upsilon_s \le \upsilon_t$ if the use of υ_s instead of υ_t does not increase the number of derivative operators in any cover of the table of versions. It is clear that if $\upsilon_s \le \upsilon_t$, version υ_t may be eliminated from the table of versions. It is easy to show that $\upsilon_s \le \upsilon_t$ if two following conditions are true:

$$L_s \le L_t;$$
$$(L_t - L_s) - |B_t \setminus B_s| \ge 0.$$

Here, L_t and L_s are the lengths of versions υ_t and υ_s (the number of derivative operators in these versions); B_t and B_s are the sets of operators in brackets in versions υ_t and υ_s; $|B_t \setminus B_s|$ is the number of elements in the difference between set B_t and B_s. Let us compare some versions from Table 3.

(a) υ_1 and υ_3: $L_1 = 3$; $L_3 = 4$; $L_1 < L_3$;
$(L_3 - L_1) - |B_3 \setminus B_1| = (4 - 3) - |\{10\} \setminus \{11,10\}| = 1 - 0 = 1 > 0$;

$\upsilon_1 \le \upsilon_3$.

(b) υ_1 and υ_4: $L_1 = 3$; $L_4 = 4$; $L_1 < L_4$;
$(L_4 - L_1) - |B_4 \setminus B_1| = (4 - 3) - |\varnothing \setminus \{11,10\}| = 1 - 0 = 1 > 0$;

$\upsilon_1 \le \upsilon_4$.

(c) υ_1 and υ_2: $L_1 = 3$; $L_2 = 3$; $L_1 = L_2$;
$(L_2 - L_1) - |B_2 \setminus B_1| = (3 - 3) - |\{14,13\} \setminus \{11,10\}| = 0 - 2 = -2 < 0$;

υ_1 *not* $\le \upsilon_2$.

(d) Check the inverse: υ_2 and υ_1: $L_1 = 3$; $L_2 = 3$; $L_1 = L_3$;
$(L_1 - L_2) - |B_1 \setminus B_2| = (3 - 3) - |\{11,10\} \setminus \{14,13\}| = 0 - 2 = -2 < 0$;

υ_2 *not* $\le \upsilon_1$; these two versions are incomparable.

(e) Versions (υ_5, υ_6) for Y_2 and (υ_7, υ_8) for Y_3 are incomparable as well (check it yourself).

The algorithm for compressing a table of versions is obvious. Comparing the versions within the columns for each Y_j $(j = 1,\ldots,m)$, we leave only those versions that are not worse than the others. Thus, after such a compression, only incomparable versions will remain in the table of versions. Table 6, which is the result of the first compression, contains only two versions for Y_b, Y_2 and Y_3.

Table 6. The result of the first compression

Yb		*Y2*		*Y3*	
υ1	*υ2*	*υ5*	*υ6*	*υ7*	*υ8*
12 *(11)* *(10)*	*15* *(14)* *(13)*	*(11)* *(10)*	*(14)* *(13)*	*(14)* *(13)*	*(11)* *(10)*

If there is at least one common version for two operators, we can combine their versions and consider these two operators as one operator (see Table 7).

Table 7. Y_2 and Y_3 as one operator

Yb		*Y2,Y3*	
υ1	*υ2*	*υ5*	*υ6*
12 *(11)* *(10)*	*15* *(14)* *(13)*	*(11)* *(10)*	*(14)* *(13)*

The compression procedure should be repeated several times. If after the final compression more than one version is available for one or several operators Y_j *(j = 1, …, T)* we can apply any well-known method to obtain a minimal cover for the prime implicant chart, because the complexity of the problem is essentially reduced. In our example, we can present the table of version in the sum-of-products form. For Table 7, we present each version as a product of its derivative operators and the set of version for each operator – as the sum-of-products:

(12*11*10 + 15*14*13) * (11*10 + 14*13) =
= 12*11*10 + 15*14*13*11*10 + 12*11*10*11*10 + 15*14*13 = 12*11*10 + 15*14*13.

After absorption, we get two products with the same length equal to three. It means that after minimization, the number of derivative operators or, which is the same, the number of conditional vertices in the first subgraph is equal to three. We use the first product in the final Table 8. The minimal subgraph is presented in Fig. 10,a. The corresponding derivative operators are taken from Table 8.

Table 8. The minimal cover of the table of versions

Yb	*Y2,Y3*
υ1	*υ3*
12 *(11)* *(10)*	*(11)* *(10)*

Without detailed explanations, we will present the process of minimization for subgraph G^2. Table 9 contains the results of the first compressing of the table of versions. The next compressing is not possible, so we use sum-of-products to find a minimal subgraph:

(32*31*30 + 39*38*37*30 + 43*38*42*41*40) * (45*44*30) * (38) * (37*30 + 42*41*40) =
= (32*31*30 + 39*38*37*30 + 43*38*42*41*40) * (45*44*30*38*37 + 45*44*30*38*42*41*40)=
= 32*31*30*45*44*38*37 + 32*31*30*45*44*38*42*41*40 + 39*38*37*30*45*44 +
+ 39*38*37*30*45*44*42*41*40 + 43*38*42*41*40*45*44*30*37 + 43*38*42*41*40*45*44*30 =
45*44*38*37*32*31*30 + 45*44*42*41*40*38*32*31*30 +
+ 45*44*39*38*37*30 + 45*44*43*42*41*40*38*30.

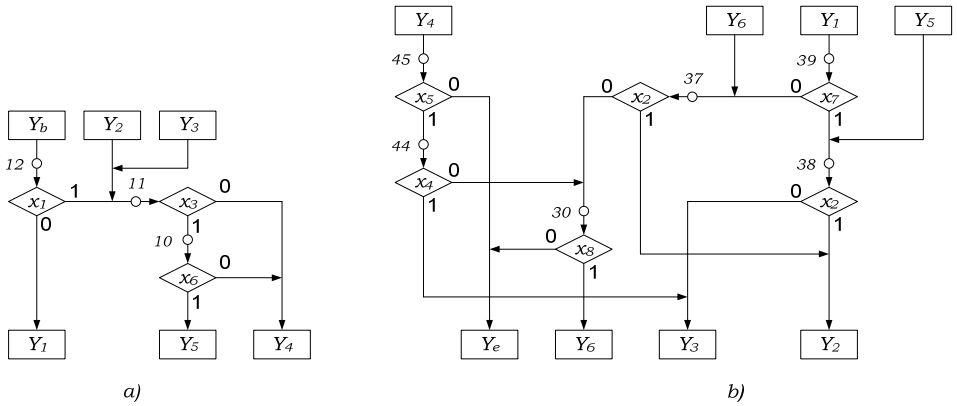

Figure 10. Subgraphs G^1 and G^2 after minimization

After absorption we get four products, the third of them contains the minimal number (six) of derivative operators. The final cover of the table of versions for the second subgraph G^2 is shown in Table 10. The minimal subgraph G^2 is shown in Fig. 10,b. The minimized ASM Γ_{min} is presented in Fig. 11. It contains only 9 conditional vertices whereas ASM Γ_{nonmin} in Fig. 5 has 17 conditional vertices.

Table 9. The result of the first compression

Y_1			Y_4	Y_5	Y_6	
$v1$	$v3$	$v4$	$v5$	$v6$	$v7$	$v8$
32 31 (30)	39 (38) (37) (30)	43 (38) (42) (41) (40)	45 44 (30)	(38)	(37) (30)	(42) (41) (40)

Table 10. The minimal cover of the table of versions

Y_1	Y_4	Y_5	Y_6
$v3$	$v5$	$v6$	$v7$
39 (38) (37) (30)	45 44 (30)	(38)	(37) (30)

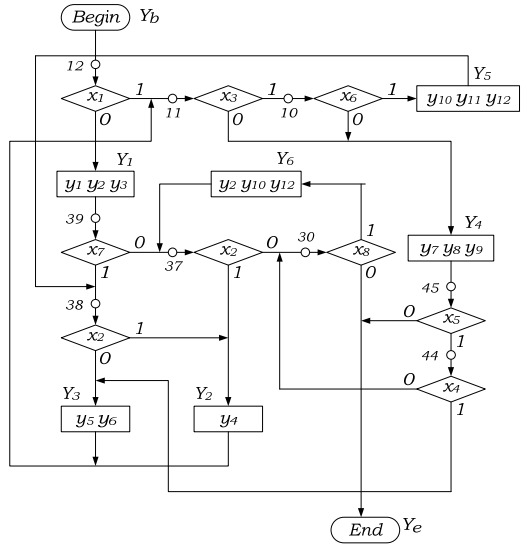

Figure 11. Mimimized ASM $\Gamma_{\min}$

6.3. Minimization of operator vertices

We will show that to minimize the number of operator vertices in ASM it is necessary to construct a Moore FSM implementing the given ASM, minimize this FSM and return from the minimal FSM to the minimal ASM. Thus, the minimization of operator vertices in ASM is reduced to the minimization of the corresponding Moore FSM. We will illustrate this technique by means of example of nonminimal ASM in Fig. 12.

As was shown in Chapter 4 (Section 4.3), Moore FSM can be synthesized in two steps:

1. The construction of a marked ASM. At this step, the vertices *Begin* and *End* are marked by the same symbol a_1 and all operator vertices are marked by symbols a_2, …, a_M. These marks are written in Fig. 12 near the corresponding vertices. Unlike to the marking in Chapter 4, we use symbol a_e to mark vertex *End* to distinguish the beginning and final vertices while returning from the minimal FSM to the minimal ASM.

2. To define the transitions in FSM Moore with the states a_1, …, a_M, a_e we find the following transition paths between operator vertices in the marked ASM:

$$a_m \tilde{x}_{m1} \ldots \tilde{x}_{mR} a_s .$$

Here $\tilde{x}_{mr} = x_{mr}$ if in the transition path, we leave the conditional vertex with x_{mr} via output ‘1’ and $\tilde{x}_{mr} = x'_{mr}$ if we leave the vertex with x_{mr} via output ‘0’.

The transition from state a_m to state a_s with input

$$X(a_m, a_s) = \tilde{x}_{m1} \ldots \tilde{x}_{mR}$$

corresponds to such a path. If a_m marks the operator vertex with operator Y_t, then the output function $\lambda(a_m) = Y_t$, i.e. we identify operator Y_t, written in the operator vertex with this state a_m.

As a result, we obtain Moore FSM with as many states as the number of symbols that is needed to mark ASM. In our example, the transition table of Moore FSM *S*, implementing ASM Γ in Fig. 12, is presented in Table 11. Since in ASM there are no paths from the final vertex into the other ones, there are no transitions from a_e into the other states (see the last row in this table).

To minimize Moore FSM (I advise you to reread Section 2.6.3 "Minimization of Moore automaton" from Chapter 2), we find successive partitions π_0, π_1, … until $\pi_{k+1} = \pi_k$ where π_k is a partition with blocks of equivalent states. In Table 11, the states with the same outputs are *0*-equivalent and they are in the same block of π_0. This partition is presented in Table 12. Column A_m of this table contains a block of partition π_0 with

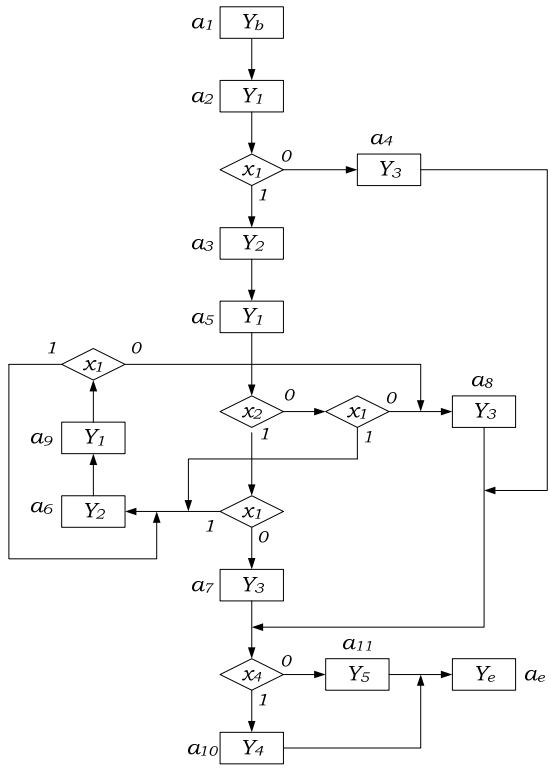

Figure 12. ASM Γ with redundant operator vertices

current states, column A_s contains a block of partition π_0 with next states from Table 11. In our example:

$$\pi_0 = \overline{a_1; a_2, a_5, a_9; a_3, a_6; a_4, a_7, a_8; a_{10}; a_{11}; a_e} = \{A_0, A_1, A_2, A_3, A_4, A_5, A_6\}.$$

Two states a_i and a_j are k-equivalent, if they are (k-1)-equivalent and they transit to the same blocks of π_{k-1} with the same inputs; π_{k-1} is the partition into the blocks of (k-1)-equivalent states. From Table 12

$$\pi_1 = \overline{a_1; a_2, a_5, a_9; a_3, a_6; a_4, a_7, a_8; a_{10}; a_{11}; a_e}.$$

Partition π_1 is equal to partition π_0. This means that π_1 is the partition with blocks of equivalent states. Taking one state from each block of π_1, we get the minimal set of states A_{min}:

$$A_{min} = \{a_1, a_2, a_3, a_4, a_{10}, a_{11}, a_e\}$$

and the minimal Moore FSM (Table 13). It is clear that minimal ASM Γ_{min} contains one beginning vertex (state a_1), one final vertex (state a_e) and five operator vertices (states a_2, a_3, a_4, a_{10} and a_{11}). To construct this minimal ASM we should divide the system of transition formulae into subsystems – we have four such subsystems in our example (Fig. 13) – and transform each subsystem into the bracket form:

1. $Y_b \rightarrow Y_1 \; ; \; Y_2 \rightarrow Y_1 ;$
2. $Y_1 \rightarrow x_1 Y_2 + x'_1 Y_3 ;$
3. $Y_3 \rightarrow x_4 Y_4 + x'_4 Y_5 ;$
4. $Y_4 \rightarrow Y_e \; ; \; Y_5 \rightarrow Y_e .$

ASM Γ_{min} with the minimal number of operator vertices (Fig. 14) was constructed by means of this system of bracket transition formulae.

Table 11. FSM Moore for ASM Γ

a_m	$Y(a_m)$	a_s	$X(a_m, a_s)$	h
a_1	Y_b	a_2	1	1
a_2	Y_1	a_3	x_1	2
		a_4	x'_1	3
a_3	Y_2	a_5	1	4
a_4	Y_3	a_{10}	x_4	5
		a_{11}	x'_4	6
a_5	Y_1	a_6	x_2x_1	7
		a_7	$x_2x'_1$	8
		a_6	x'_2x_1	9
		a_8	$x'_2x'_1$	10
a_6	Y_2	a_9	1	11
a_7	Y_3	a_{10}	x_4	12
		a_{11}	x'_4	13
a_8	Y_3	a_{10}	x_4	14
		a_{11}	x'_4	15
a_9	Y_1	a_6	x_1	16
		a_8	x'_1	17
a_{10}	Y_4	a_e	1	18
a_{11}	Y_5	a_e	1	19
a_e	Y_e	a_e	1	20

Table 12. Partition π_0

A_m	a_m	$Y(a_m)$	A_s	$X(a_m, a_s)$
A_0	a_1	Y_b	A_1	1
A_1	a_2	Y_1	A_2	x_1
			A_3	x'_1
	a_5	Y_1	A_2	x_2x_1
			A_3	$x_2x'_1$
			A_2	x'_2x_1
			A_3	$x'_2x'_1$
	a_9	Y_1	A_2	x_1
			A_3	x'_1
A_2	a_3	Y_2	A_1	1
	a_6	Y_2	A_1	1
A_3	a_4	Y_3	A_4	x_4
			A_5	x'_4
	a_7	Y_3	A_4	x_4
			A_5	x'_4
	a_8	Y_3	A_4	x_4
			A_5	x'_4
A_4	a_{10}	Y_4	A_6	1
A_5	a_{11}	Y_5	A_6	1
A_6	a_e	Y_e	A_6	1

Table 13. The minimal Moore FSM

a_m	$Y(a_m)$	a_s	$X(a_m, a_s)$	h
a_1	Y_b	a_2	1	1
a_2	Y_1	a_3	x_1	2
		a_4	x'_1	3
a_3	Y_2	a_2	1	4
a_4	Y_3	a_{10}	x_4	5
		a_{11}	x'_4	6
a_{10}	Y_4	a_e	1	7
a_{11}	Y_5	a_e	1	8
a_e	Y_e	a_e	1	9

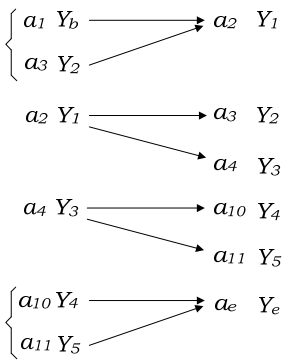

Figure 13. The partition of transition formulae for the minimal ASM

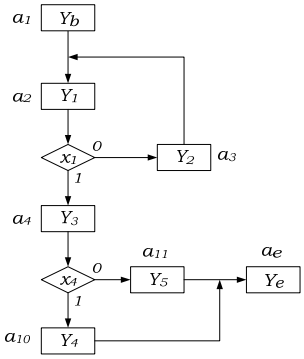

Figure 14. ASM Γ_{min}

6.4. ASM combining

When we would like to describe the behavior of a very complicated digital system, sometimes it is difficult to present it by just one ASM. In these cases, it is possible to describe separate subbehaviors with ASMs Γ_1, …, Γ_Q and then to combine them into one combined ASM Γ

At the beginning, we will show a trivial method for ASM combining. This method will not be used in the future, but it will help us to understand the combining process. As an example, we take four ASMs in Fig. 15.

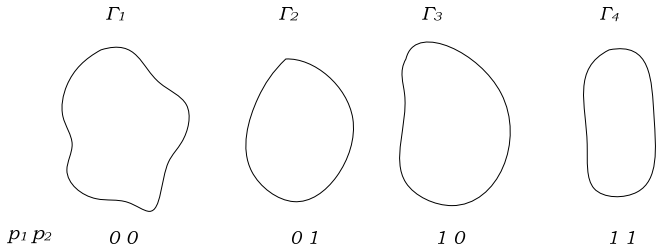

Figure 15. Four ASM to combine

First, we encode ASMs by vectors of values of new variables p_1 and p_2, these variables are not in the initial ASMs. The number of such variables depends on the number of ASMs and, in general case, is equal to $N =] \log_2 Q [$. Here Q is the number of ASMs to combine and $] a [$ is the nearest integer greater than a, or equal to a if a is integer. In our example, $Q = 4$ and $N = 2$.

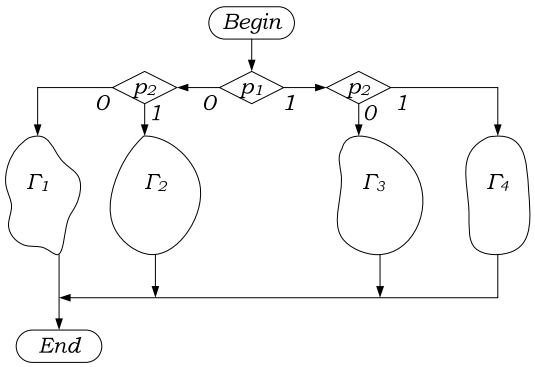

Figure 16. Trivial combining

The trivial combining method is shown in Fig. 16. We did not change ASMs in this figure; we only removed their vertices "*Begin*" and "*End*" because we can have only one pair of such vertices in the combined ASM. Paths running through conditional vertices with variables p_1, p_2, lead to ASMs with the corresponding codes. We use these codes here as instruction codes, or mode codes. For example, if $p_1p_2 = 00$ – ASM Γ_1 will be implemented, if $p_1p_2 = 01$ – ASM Γ_2 will be implemented etc.

Now we will present another method for ASM combining in which we minimize the numbers of operator and conditional vertices. Let us assume that we have ASMs Γ_1, …, Γ_Q and there are no equal operators within each ASM Γ_q but, of course, there may be such operators in different ASMs. If two or more equal operators are written in various operator vertices of one such ASM, we should rename them, beginning with the second one, by different numbers Y_{F+1}, Y_{F+2}, … Here F is the maximal number of operators in all ASMs that should be combined.

As an example we will combine ASMs Γ_1, …, Γ_4 in Fig. 17. The procedure for ASMs combining consists of several steps.

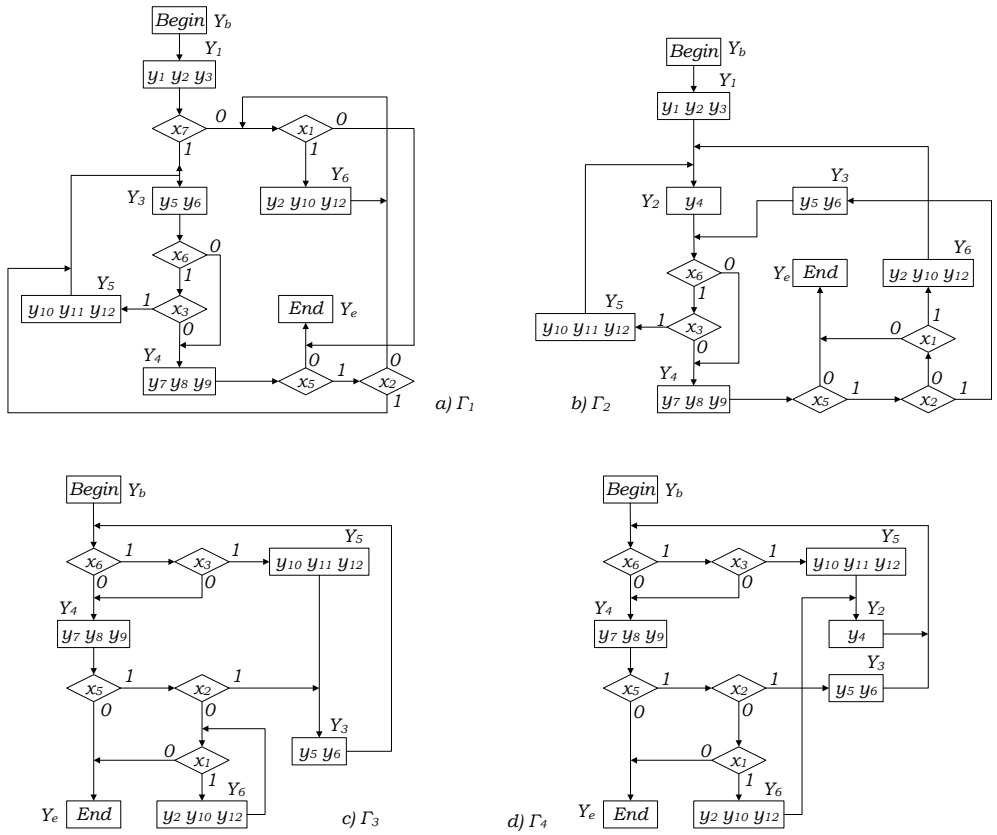

Figure 17. ASMs for combining

1. Construct MSA M_q for each ASM Γ_q. The MSAs M_1, …, M_4 are in Tables 14 – 17.

2. Encode each MSA M_q *(q = 1, …, 4)* by binary code $K(M_q)$. Since there are only four MSAs, two coding variables p_1 and p_2 are enough. Let

$K(M_1) = 00, K(M_2) = 01, K(M_3) = 10, K(M_4) = 11.$

Table 14. MSA M_1

$p'_1p'_2$	Y_1	Y_3	Y_4	Y_5	Y_6	Y_e
Y_b	1					
Y_1		x_7			x'_7x_1	$x'_7x'_1$
Y_3			$x_6x'_3 + x'_6$	x_6x_3		
Y_4		x_5x_2			$x_5x'_2x_1$	$x_5x'_2x'_1 + x'_5$
Y_5		1				
Y_6					x_1	x'_1

Table 15. MSA M_2

p'_1p_2	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_e
Y_b	1						
Y_1		1					
Y_2				$x_6x'_3 + x'_6$	x_6x_3		
Y_3				$x_6x'_3 + x'_6$	x_6x_3		
Y_4			x_5x_2			$x_5x'_2x_1$	$x_5x'_2x'_1 + x'_5$
Y_5		1					
Y_6		1					

Table 16. MSA M_3

$p_1p'_2$	Y_3	Y_4	Y_5	Y_6	Y_e
Y_b		$x_6x'_3 + x'_6$	x_6x_3		
Y_3		$x_6x'_3 + x'_6$	x_6x_3		
Y_4	x_5x_2			$x_5x'_2x_1$	$x_5x'_2x'_1 + x'_5$
Y_5	1				
Y_6				x_1	x'_1

Table 17. MSA M_4

p_1p_2	Y_2	Y_3	Y_4	Y_5	Y_6	Y_e
Y_b			$x_6x'_3 + x'_6$	x_6x_3		
Y_2			$x_6x'_3 + x'_6$	x_6x_3		
Y_3			$x_6x'_3 + x'_6$	x_6x_3		
Y_4		x_5x_2			$x_5x'_2x_1$	$x_5x'_2x'_1 + x'_5$
Y_5	1					
Y_6	1					

3. Write product $P(M_q)$ corresponding to $K(M_q)$ in the left upper corner of M_q:

$P(M_1) = p'_1p'_2; P(M_2) = p'_1p_2; P(M_3) = p_1p'_2; P(M_4) = p_1p_2.$

4. Construct combined MSA M. The set of operators $Y(M)$ in this MSA M is

$$Y(M) = \bigcup_{q=1}^{Q} Y(M_q),$$

where $Y(M_q)$ is the set of operators in MSA M_q and Q is the number of separate MSAs. This is the main idea of our combining. If the same operator Y_q occurs

in the several ASMs, there will be only one copy of this operator in the combined ASM. In our example:
$Y(M_1) = \{Y_b, Y_1, Y_3, Y_4, Y_5, Y_6, Y_e\}$; $Y(M_2) = \{Y_b, Y_1, Y_2, Y_3, Y_4, Y_5, Y_6, Y_e\}$;
$Y(M_3) = \{Y_b, Y_3, Y_4, Y_5, Y_6, Y_e\}$; $Y(M_4) = \{Y_b, Y_2, Y_3, Y_4, Y_5, Y_6, Y_e\}$.

Thus, in our example we have only eight operators in the combined ASM, including operators Y_b and Y_e:

$$Y(M) = Y(M_1) \cup Y(M_2) \cup Y(M_3) \cup Y(M_4) = \{ Y_b, Y_1, \ldots, Y_6, Y_e\}.$$

This combined ASM has seven rows and seven columns (Table 18).

To construct entries in the combined MSA M, we multiply each entry of each MSA M_q by its product $P(M_q)$ written in the left upper corner of this MSA M_q and insert it in the corresponding entry of MSA M – at the intersection of the same row and the same column. In other words, we multiply each MSA M_q by its product $P(M_q)$ and insert it in the combined MSA M.

To check that combined MSA M, thus constructed, implements MSA M_q when $P(M_q) = 1$, it is sufficient to substitute code $K(M_q)$ in each entry of MSA M. For example, if you substitute $K(M_1) = 00$ in MSA M in Table 18, you will get MSA M_1 in Table 14.

Table 18. Combined MSA M

	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_e
Y_b	$p'_1p'_2$ p'_1p_2			$p_1p'_2x_6x'_3 + p_1p'_2x'_6$ $p_1p_2x_6x'_3 + p_1p_2x'_6$	$p_1p'_2x_6x_3$ $p_1p_2x_6x_3$		
Y_1		p'_1p_2	$p'_1p'_2x_7$			$p'_1p'_2x'_7x_1$	$p'_1p'_2x'_7x'_1$
Y_2				$p'_1p_2x_6x'_3 + p'_1p_2x'_6$ $p_1p_2x_6x'_3 + p_1p_2x'_6$	$p'_1p_2x_6x_3$ $p_1p_2x_6x_3$		
Y_3				$p'_1p'_2x_6x'_3 + p'_1p'_2x'_6$ $p'_1p_2x_6x'_3 + p'_1p_2x'_6$ $p_1p'_2x_6x'_3 + p_1p'_2x'_6$ $p_1p_2x_6x'_3 + p_1p_2x'_6$	$p'_1p'_2x_6x_3$ $p'_1p_2x_6x_3$ $p_1p'_2x_6x_3$ $p_1p_2x_6x_3$		
Y_4			$p'_1p'_2x_5x_2$ $p'_1p_2x_5x_2$ $p_1p'_2x_5x_2$ $p_1p_2x_5x_2$			$p'_1p'_2x_5x'_2x_1$ $p'_1p_2x_5x'_2x_1$ $p_1p'_2x_5x'_2x_1$ $p_1p_2x_5x'_2x_1$	$p'_1p'_2x_5x'_2x'_1 + p'_1p'_2x'_5$ $p'_1p_2x_5x'_2x'_1 + p'_1p_2x'_5$ $p_1p'_2x_5x'_2x'_1 + p_1p'_2x'_5$ $p_1p_2x_5x'_2x'_1 + p_1p_2x'_5$
Y_5		p'_1p_2 p_1p_2	$p'_1p'_2$ $p_1p'_2$				
Y_6		p'_1p_2 p_1p_2				$p'_1p'_2x_1$ $p_1p'_2x_1$	$p'_1p'_2x'_1$ $p_1p'_2x'_1$

5. Construct combined ASM Γ.

5.1. Partition the system of transition formulae into independent subsystems. Here we should implement techniques similar to ASM partitioning into subgraphs (Secion 6.2.1). We begin with any operator, for example, from Y_b and find the set of operators towards which there are paths (12) in MSA M (the columns with entries distinct from zero in row Y_b). For row Y_b, they are operators Y_1, Y_4, Y_5 (see arrows from Y_b in Fig. 18,a). Next, we go to the right side of this subgraph and find operators, other than Y_b, from which there are paths (12) leading to operators Y_1, Y_4, Y_5. We place these operators (Y_2, Y_3) on the left side under Y_b. For all operators on the left, we continue picking such

operators to which there are paths in MSA *M*, etc., until the set on the left *(A^1)* and that on the right *(B^1)* are no longer increasing. Next we will choose a new operator $Y_m \notin A^1$ (Y_1 in our example) and construct *(A^2)* and *(B^2)* in a similar way (see Fig. 18,b). Thus, in our example we have two independent subsystems of transition formulae derived immediately from MSA *M*:

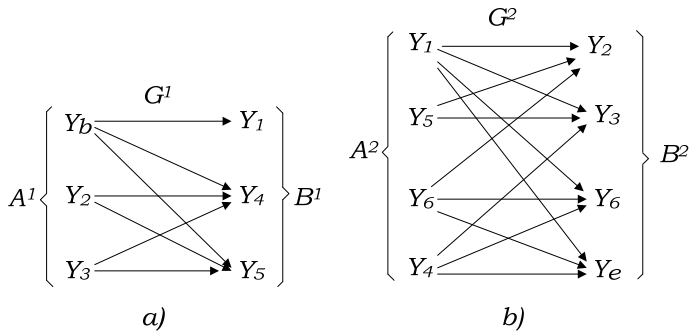

Figure 18. Partition of operators in MSA *M*

Subsystem 1

$Y_b \rightarrow p'_1p'_2Y_1 + p'_1p_2Y_1 + p_1p'_2x_6x'_3Y_4 + p_1p'_2x'_6Y_4 + p_1p_2x_6x'_3Y_4 + p_1p_2x'_6Y_4 + p_1p'_2x_6x_3Y_5 + p_1p_2x_6x_3Y_5;$

$Y_2 \rightarrow p'_1p_2x_6x'_3Y_4 + p'_1p_2x'_6Y_4 + p_1p_2x_6x'_3Y_4 + p_1p_2x'_6Y_4 + p'_1p_2x_6x_3Y_5 + p_1p_2x_6x_3Y_5;$

$Y_3 \rightarrow p'_1p'_2x_6x'_3Y_4 + p'_1p'_2x'_6Y_4 + p'_1p_2x_6x'_3Y_4 + p'_1p_2x'_6Y_4 + p_1p'_2x_6x'_3Y_4 + p_1p'_2x'_6Y_4 + p_1p_2x_6x'_3Y_4 + p_1p_2x'_6Y_4 + p'_1p'_2x_6x_3Y_5 + p'_1p_2x_6x_3Y_5 + p_1p'_2x_6x_3Y_5 + p_1p_2\ x_6x_3Y_5.$

Subsystem 2

$Y_1 \rightarrow p'_1p_2Y_2 + p'_1p'_2x_7Y_1 + p'_1p'_2x'_7x_1Y_6 + p'_1p'_2x'_7x'_1Y_e;$

$Y_4 \rightarrow p'_1p'_2x_5x_2Y_3 + p'_1p_2x_5x_2Y_3 + p_1p'_2x_5x_2Y_3 + p_1p_2x_5x_2Y_3 + p'_1p'_2x_5x'_2x_1Y_6 + p'_1p_2x_5x'_2x_1Y_6 + p_1p'_2x_5x'_2x_1Y_6 + p_1p_2x_5x'_2x_1Y_6 + p'_1p'_2x_5x'_2x'_1Y_e + p'_1p'_2x'_5Y_e + p'_1p_2x_5x'_2x'_1Y_e + p'_1p_2x'_5Y_e + p_1p'_2x_5x'_2x'_1Y_e + p_1p'_2x'_5Y_e + p_1p_2x_5x'_2x'_1Y_e + p_1p_2x'_5Y_e;$

$Y_5 \rightarrow p'_1p_2Y_2 + p_1p_2Y_2 + p'_1p'_2Y_3 + p_1p'_2Y_3;$

$Y_6 \rightarrow p'_1p_2Y_2 + p_1p_2Y_2 + p'_1p'_2x_1Y_6 + p_1p'_2x_1Y_6 + p'_1p'_2x'_1Y_e + p_1p'_2x'_1Y_e.$

5.2. Construct minimal subgraphs. Here we should implement techniques presented above in Section 6.2.2 "*Constructing a set of equivalent subgraphs"* and Section 6.2.3 "*Finding a minimal cover for the table of versions"*. Without detailed explanation (we advice you to make ASM expansion by yourself) we give the tables of derivative operators for the first subsystem (Table 19) and the tables of versions during two compressions (Tables 20 and 21). The minimal subgraph with operators Y_b, Y_2 and Y_3 is presented in Table 22.

Table 19. The derivative operators for Y_b, Y_2, Y_3

8	x3	5	4	16	p1	4	1
9	x6	8	4	17	x3	15	16
10	x6	5	4	18	x6	14	16
11	x3	10	4	19	x6	17	16
12	p1	9	1	20	p1	10	1
13	p1	11	1	21	x6	15	16
14	p1	8	1	22	x3	20	16
15	p1	5	1	23	x3	21	16

Table 20. The table of versions for Y_b, Y_2, Y_3

Y_b						Y_2, Y_3	
υ1	υ2	υ3	υ4	υ5	υ6	υ7	υ8
12	13	18	19	22	23	(9)	(11)
(9)	(11)	14	17	20	21	(8)	(10)
(8)	(10)	(8)	15	(10)	15		
		16	16	16	16		

Table 21. The table of versions for Y_b, Y_2, Y_3 after compression of Y_b

Y_b		Y_2, Y_3	
υ1	υ2	υ7	υ8
12	13	(9)	(11)
(9)	(11)	(8)	(10)
(8)	(10)		

Table 22. The final table of versions for Y_b, Y_2, Y_3

Y_b	Y_2, Y_3
υ1	υ7
12	(9)
(9)	(8)
(8)	

Tables 23 – 26 contain the same for the second subsystem of transition formulae. The last table presents the minimal subgraph with operators Y_1, Y_4, Y_5 and Y_6. The minimal combined ASM Γ is shown in Fig. 19.

Table 23. The derivative operators for Y_1, Y_4, Y_5 and Y_6

8	x1	6	e	22	p2	2	10	36	x5	3	e
9	x7	3	8	23	x7	15	17	37	x5	8	e
10	x7	3	6	24	p2	2	11	38	x5	6	e
11	x7	3	e	25	x7	15	18	39	x1	38	e
12	x1	10	11	26	x1	22	24	40	x2	36	37
13	p2	2	9	27	x1	22	25	41	x2	36	39
14	p2	2	12	28	x1	23	24	42	x5	31	e
15	p2	2	3	29	x1	23	25	43	x2	36	38
16	p2	2	8	30	x2	3	8	44	x5	32	e
17	p2	2	6	31	x2	3	6	45	x2	36	e
18	p2	2	e	32	x2	3	e	46	x1	42	44
19	x1	17	18	33	x1	31	32	47	x1	42	45
20	x7	15	16	34	x5	30	e	48	x1	43	44
21	x7	15	19	35	x5	33	e	49	x1	43	45

Table 24. The table of versions for Y_1, Y_4, Y_5 and Y_6

Y_1					Y_4							Y_5	Y_6	
$v1$	$v2$	$v3$	$v4$	$v5$	$v6$	$v7$	$v8$	$v9$	$v10$	$v11$	$v12$	$v13$	$v14$	$v15$
13	14	20	21	26	34	35	40	41	46	47	49	(15)	(16)	(19)
9	12	(15)	(15)	22	30	33	36	36	42	42	43		(8)	(17)
(8)	10	(16)	(19)	10	(8)	31	37	39	31	31	36			(18)
	11	(8)	(17)	24		32	(8)	38	44	45	38			
			(18)	11					32	36	45			

Table 25. The compressed table of versions for Y_1, Y_4, Y_5 and Y_6

Y_1			Y_4	Y_5	Y_6	
$v1$	$v3$	$v4$	$v6$	$v13$	$v14$	$v15$
13	20	21	34	(15)	(16)	(19)
9	(15)	(15)	30		(8)	(17)
(8)	(16)	(19)	(8)			(18)
	(8)	(17)				
		(18)				

Table 26. The final table of versions for Y_1, Y_4, Y_5 and Y_6

Y_1	Y_4	Y_5	Y_6
$v3$	$v6$	$v13$	$v14$
20	34	(15)	(16)
(15)	30		(8)
(16)	(8)		
(8)			

Note, that during expansion of transition formulae we can come across two don't care cases:

- If the number of ASMs Q is smaller than 2^N, i.e. not all possible vectors of coding variables $p_1, \ldots, p_N$ are used for encoding of MSAs $M_1, \ldots, M_Q$, all transition formulae are not specified on such vectors of variables $p_1, \ldots, p_N, x_1, \ldots, x_L$, in which first N components correspond to unused combination. We do not have such a case in our example.
- If operator Y_t is absent in some MSA M_q, then transition formula for Y_t is not specified on vectors of variables $p_1, \ldots, p_N, x_1, \ldots, x_L$, in which first N components correspond to the code K(M_q) of this MSA M_q. The operator Y_2 is not in MSAs M_1 and M_3 in our example, therefore the transition formula for Y_2 is not specified for vectors, in which $p_1 = p_2 = 0$ and $p_1 = 1$, $p_2 = 0$, since K(M_1) = *00* and K(M_3) = *10*.

$$Y_2 \rightarrow p'_1p_2x_6x'_3Y_4 + p'_1p_2x'_6Y_4 + p_1p_2x_6x'_3Y_4 + p_1p_2x'_6Y_4 + p'_1p_2x_6x_3Y_5 + p_1p_2x_6x_3Y_5 =$$

$$= x_6(p'_1p_2x'_3Y_4 + p_1p_2x'_3Y_4 + p'_1p_2x_3Y_5 + p_1p_2x_3Y_5) + x'_6(p'_1p_2Y_4 + p_1p_2Y_4) =$$

$$= x_6(x_3(p'_1p_2Y_5 + p_1p_2Y_5) + x'_3(p'_1p_2Y_4 + p_1p_2Y_4)) + x'_6(p'_1p_2Y_4 + p_1p_2Y_4).$$

In the internal brackets, we couldn't continue expanding with variable p_2, because this expression does not contain products with $p'_1p'_2$ and $p_1p'_2$ corresponding to the unused codes *00* and *10*. Using don't care we have

$$p'_1p_2 + p'_1p'_2 = p'_1; \qquad p_1p_2 + p_1p'_2 = p_1$$

and finally we get:

$$Y_2 \rightarrow x_6(x_3(p'_1Y_5 + p_1Y_5) + x'_3(p'_1Y_4 + p_1Y_4)) + x'_6(p'_1Y_4 + p_1Y_4) =$$

$$= x_6(x_3Y_5 + x'_3Y_4) + x'_6Y_4.$$

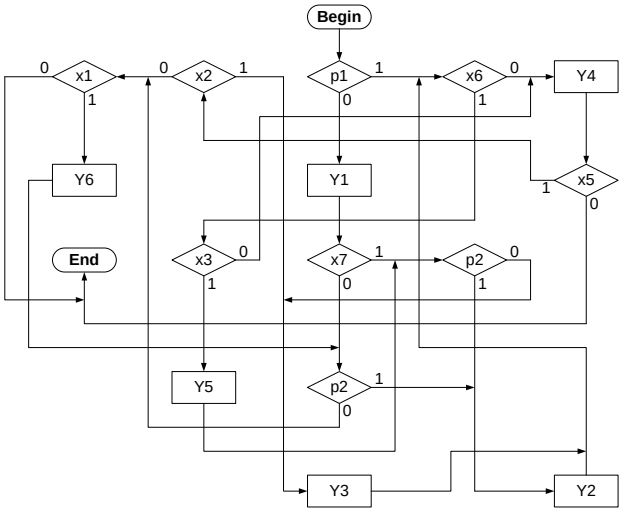

Figure 19. Combined ASM *Γ*