ISSN 0032-9460

Eng

Russian Original Vol. 23, No. 1, January-March, 1987

July, 1987

COP. 2

PRITA9 23(1) 1-90 (1987)

PROBLEMS OF INFORMATION TRANSMISSION

ПРОБЛЕМЫ ПЕРЕДАЧИ ИНФОРМАЦИИ

(PROBLEMY PEREDACHI INFORMATSII)

TRANSLATED FROM RUSSIAN

c/b CONSULTANTS BUREAU, NEW YORK

8

BINARY PROGRAMS AND THEIR REALIZATION BY ASYNCHRONOUS AUTOMATA

Yu. L. Sagalovich and A. A. Shalyto

UDC 621.391.1-503.5

We consider coding of states of asynchronous automata designed to evaluate systems of Boolean functions by means of binary programs. We show that for a fairly wide class of systems of Boolean functions race-proof state coding is irredundant.

1. INTRODUCTION

Recent advances in microelectronics have focused the attention on program realization of systems of Boolean functions (SBF).

Binary programs [1] for one Boolean function and for a whole SBF can be realized in the form of discrete automata. This is particularly important, for instance, when the entire value set of the variables is not known in advance and the values appear successively one after another.

Realization of a binary program (BP) for SBF by an asynchronous automaton raises the question of race-proof coding of the automaton states. This paper presents a preliminary analysis of this issue. In Sec. 2 we construct the transition graph and the transition table of an automaton realizing the BP for a given SBF. Section 3 is devoted in its entirety to race-proof coding of automaton states, relying on tghe terminology and the system of concepts introduced in Sec. 2.

2. THE BINARY PROGRAM GRAPH AND THE REALIZING AUTOMATON

Consider a system of m Boolean functions

$$y_j = f_j(x_0, x_1, \ldots, x_{l-1}), \quad j = 1, 2, \ldots, m, \tag{1}$$

such that for any $i = 1, \ldots, \ell$, there is at least one function in the system which is essentially dependent on x_{i-1}.

Definition. Following [1], we define the binary program for the system (1) as a program which sequentially computes the coefficients $f_i(\sigma_0, \sigma_1, \ldots, \sigma_{i-1}, x_i, \ldots, x_{\ell-1})$ of the conjunctions $x_0^{\sigma_0} x_1^{\sigma_1} \ldots x_{i-1}^{\sigma_{i-1}}$, $i = 1, 2, \ldots, \ell$ in the expansion

$$f_j = \bigvee_{\substack{\text{over all} \\ \text{combinations } \sigma}} x_0^{\sigma_0} x_1^{\sigma_1} \ldots x_{i-1}^{\sigma_{i-1}} f_j(\sigma_0, \sigma_1, \ldots, \sigma_{i-1}, x_i, \ldots, x_{l-1}),$$
$$j = 1, 2, \ldots, m.$$

of the system (1) on an arbitrary successively generated combination $\sigma = (\sigma_0, \sigma_1, \ldots, \sigma_{\ell-1})$ of the variable values.*

The binary program for the system (1) is representable as a graph with one initial node and at most 2^m final nodes. This graph is constructed in the following way [1, 2].

Take a dichotomous tree, i.e., a tree with precisely two outgoing edges from each node (except the final nodes) and one incoming edge into each node (except the initial node). One of the outgoing edges of each node is labeled 0 and the other 1. The tree nodes are assigned to $\ell + 1$ levels indexed 0, 1, ..., ℓ. Each node belongs precisely to one level. Each node of level i is reached by a unique path of i − 1 edges from the initial tree node. The i-th level nodes thus may be indexed by binary numbers $\sigma_0\sigma_1\ldots\sigma_{i-1}$, where $\sigma_j = 0, 1$,

*In general, a binary program is a program which consists of conditional jumps.

Translated from Problemy Peredachi Informatsii, Vol. 23, No. 1, pp. 89-96, January-March, 1987. Original article submitted February 27, 1985; revision submitted June 12, 1985.

$j = 0, 1, \ldots, i - 1$; each of these numbers precisely describes the path from the initial node to the given node. Each node of level i is at the same time the initial node of a subtree with $\ell + 1 - i$ levels. Subtrees with initial nodes located on the same level are nonintersecting. To the node $\sigma_0\sigma_1\ldots\sigma_{i-1}$ we allocate the system of expansion coefficients of the SBF (1) in the variables $x_0, x_1, \ldots, x_{i-1}$ corresponding to the same conjunction $x_0^{\sigma_0}x_1^{\sigma_1}\ldots x_{i-1}^{\sigma_{i-1}}$. A tree with allocated systems of coefficients is called filled. By assumption the system (1) is essentially dependent on the variables $x_0, x_1, \ldots, x_{\ell-1}$, and so different systems of coefficients are allocated to different first-level nodes. Having considered the nodes of level $(i - 1)$, let us pass to the i-th level nodes ($i = 2, \ldots, \ell - 1$). If the systems of coefficients in two nodes

$$\sigma_0\sigma_1\ldots\sigma_{i-2}\sigma_{i-1} \text{ and } \sigma_0\sigma_1\ldots\sigma_{i-2}\bar{\sigma}_{i-1} \tag{2}$$

are the same (this means that the system of coefficients in the node $\sigma_0\sigma_1\ldots\sigma_{i-2}$ is independent of x_{i-1}), then the node $\sigma_0\sigma_1\ldots\sigma_{i-2}$ and the two nodes (2) are combined into one i-th level node. The incoming edge of the node $\sigma_0\sigma_1\ldots\sigma_{i-2}$ now enters the new node. After combining nodes of the form (2), the i-th level may still contain nodes with identical systems of coefficients. Combine all these nodes, so that their incoming edges now enter the combined node. At the same time, also combine the filled subtrees attached to these nodes, since filled subtrees with the same number of levels and the same systems of coefficients in the initial nodes are totally identical.

Each path from the initial to a final node represents a combination of variable values on which the SBF (1) takes the system of values assigned to the given final node. Therefore systems of coefficients should be saved only for the final nodes and can be deleted from all the other nodes.

Label all the i-th level nodes by the same symbol x_i. This concludes the construction of the graph. Given our indexing of the variables x_i, the graph is unique and its configuration may be varied only depending on the indexing of the variables. The system (1) is uniquely reconstructed from the last-level systems of coefficients and the graph.†

As an example, Fig. 1 shows the BP graph for the SBF

$$\begin{aligned} y_1 &= \bar{x}_3(\bar{x}_1\bar{x}_2 \vee x_1x_2x_3) \vee x_0x_2(x_1 \oplus x_3), \\ y_2 &= \bar{x}_0(\bar{x}_1 \vee x_2\bar{x}_3) \vee x_0x_1\bar{x}_2x_3 \vee \bar{x}_1x_2\bar{x}_3. \end{aligned} \tag{3}$$

If the nodes of the BP graph are indexed arbitrarily, and the system of functions is reconstructed from this graph in the same sequence in which the coefficients of expansion in the variables x_i, $i = 0, 1, \ldots, \ell - 1$ are determined, then the BP may be realized in the form of a discrete automaton. The automaton states are indexes of graph nodes; the input state are the indexes of the variables $x_0, x_1, \ldots, x_{\ell-1}$ and their values. The transition of the automaton from state a of level i to state b (always on a level with a higher index) for a given value σ_i of the variable x_i is interpreted as finding a system of coefficients corresponding to the node a. The output of the automaton may be a state index, and its output in the terminal state is a combination of function values. If the output is the index of one of the final nodes, then this means that the functions of the system (1) have been evaluated on the corresponding combination of variable values. One of the useful automaton realizations of binary programs is the Wilks microprogrammed automaton which allows parallel output of computation results.

When binary programs are realized by synchronous automata, the time to evaluate the system (1) may become excessively long under certain conditions. Therefore, the asynchronous mode is apparently the only way for increasing the access speed.

3. RACE-PROOF CODING OF THE STATES OF AN AUTOMATON REALIZING A BINARY PROGRAM

Asynchronous operation of a BP-realizing automaton requires race-proof coding of its states.

†The method proposed by V. I. Rubinov and A. A. Shalyto may be applied in order to construct the BP graph of a SBF. This method is a development of Blokh's canonical method [3] and it is based on a representation of the SBF by a truth table: identical rows of function values in this table are assigned the same index, while different rows are assigned different indexes. Then the graph is constructed as in the canonical method [3].

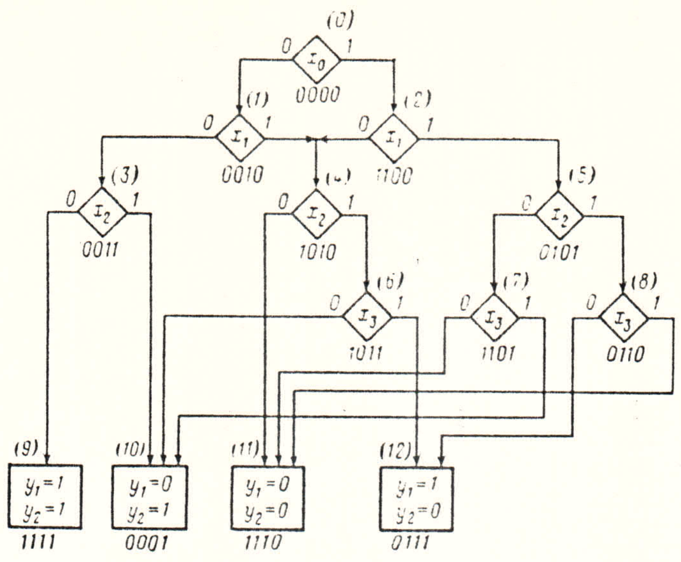

Fig. 1

TABLE 1

State	x_0		x_1		x_2		x_3	
	0	1	0	1	0	1	0	1
0	1	2	–	–	–	–	–	–
1	–	–	3	4	–	–	–	–
2	–	–	5	6	–	–	–	–
3	–	–	–	–	7	8	–	–
4	–	–	–	–	9	10	–	–
5	–	–	–	–	11	12	–	–
6	–	–	–	–	13	14	–	–
7	–	–	–	–	–	–	15	16
8	–	–	–	–	–	–	17	18
9	–	–	–	–	–	–	19	20
10	–	–	–	–	–	–	21	22
11	–	–	–	–	–	–	23	24
12	–	–	–	–	–	–	25	26
13	–	–	–	–	–	–	27	28
14	–	–	–	–	–	–	29	30
15	–	–	–	–	–	–	–	–
.								
.								
.								
30	–	–	–	–	–	–	–	–

In race-proof coding, 1) all the automaton states should be encoded by different binary code vectors (we denote them by indexed letters); 2) any two transitions $S_1 \rightarrow S_2$ and $S_3 \rightarrow S_4$ occurring for the same input state should have no common intermediate states which may arise due to races between automaton memory elements.

If the transitions $S_1 \rightarrow S_2$ and $S_3 \rightarrow S_4$ have no common intermediate states, we say that the transitions are separated. The transitions $S_1 \rightarrow S_2$ and $S_3 \rightarrow S_4$ are separated if and only if the state vectors S_1, S_2, S_3, S_4 contains at least one component with S_1 and S_2 equal to 1 and S_3 and S_4 equal to 0, or vice versa.

The BP graphs have highly specific properties, which largely determine the choice of the race-proof coding method.

Let us consider a system of functions (1) in which all the systems of expansion coefficients are different. This means that we do not have to combine any nodes and edges in the graph, and therefore the BP graph of this system of functions is a complete tree with 2^i nodes on each level i (here $m \geq \ell$).

If the graph nodes are indexed on each level from left to right, starting with level 0, then the nodes of level i carry indexes from $2^i - 1$ to $2^{i+1} - 2$.

Let the states of the automaton realizing the BP with this graph be arranged in the transition table in the order in which the corresponding graph nodes are indexed. (For convenience, the table gives the index i instead of S_i.)

Assume that the 2ℓ table columns are arranged in the order of increasing indexes of the variables x_i and, for definiteness, in each pair of columns corresponding to the same variable the column $x_i = 0$ precedes the column $x_i = 1$.

Then the transition table has the standard form. It comprises $\ell + 1$ horizontal bands, and the i-th band contains 2^i rows ($i = 0, 1, \ldots, \ell$). At the intersection of columns $x_i = 0$ and $x_i = 1$ with the i-th band are entered the indexes of the final states of the transitions. In the other cells of the table, the transitions are undefined. The final states of the transitions in the i-th band are the initial states of the transitions in the (i + 1)-th band. Along the diagonal of the table, from left to right and from top to bottom, we have "boxes" of common elements. The width of each "box" is two columns and its height is 2^i rows. In the last band ℓ the transitions are undefined. Therefore this band may be omitted. The table in this band may be defined, say, by assuming that the states pass into themselves.

Table 1 is a transition table for an automaton with 31 states which realizes the BP of a system of functions of four variables.

A path is a sequence of states which pass into one another as the input states change. For instance, the sequences 0, 1, 3, 8, 17 or 0, 2, 6, 14, 30 are paths.

For convenience, instead of the term "separation of transitions" we will use the term "separation of rows," implying that the final states of the transitions lie in two different rows.

The following proposition holds.

Proposition. All $2^{\ell+1} - 1$ states of the automaton realizing the BP of a system of functions whose expansion coefficients in the variables $x_0, x_1, \ldots, x_{\ell-1}$ are all different may be coded by irredundant race-proof code, i.e., a code of length $k = \ell + 1$.

The proof is constructive, i.e., we actually perform race-proof encoding of states. We do not require separation of rows in the zeroth band of the table.

Separation of the first-band rows requires one bit. Therefore, we enter the symbol $\bar{\sigma}_1$ in the first bit of all the state vectors of all the paths starting in state 1 and the symbol σ_1 in the first bit of all the state vectors of all the paths starting in state 2.

In order to separate the four rows of the second band, we need one additional bit of the code vector, since two pairs of initial states of this band are already separated as the final states of the transitions with different initial states in the preceding band. In the second bit of the vectors of states 3 and 5 we enter the symbol $\bar{\sigma}_2$, and in the second bit of the states 4 and 6 the symbol σ_2. The same symbols are entered in the second bit of the vectors of those states which belong to paths with the corresponding initial state. The same procedure is applied to each band. Each successive band (with a "box") requires one new variable for separation of previously unseparated rows. There are $\ell - 1$ such bands. Therefore, in order to separate all the transitions occurring for identical input states we need $\ell - 1$ bits. Successive separation of transitions implies partial discrimination of states: all the states which belong to the path starting in some state of band i are distinguished from all the states which belong to paths starting in another initial state of band i in the table are coded by different code vectors of legnth i. However, by construction, together with each initial state of band i, the same combination of the first i bits is found in $2^{\ell+1-i} - 2$ additional states. This means that each combination of i symbols occurs precisely $2^{\ell+1-i} - 1$ times, and therefore the remaining $\ell + 1 - i$ bits are sufficient for distinguishing between the states with these identical combinations.

The last remark allows us to complete the coding procedure. The remaining symbols up to $\ell + 1$ are entered so as to satisfy a single requirement, namely ensuring that all the code combinations are distinct. This is feasible given the quantitative relationships discussed above. The additional symbols can be entered in the order of decreasing band index in the transition table, since in the process of separation of transitions bands with higher index are coded by longer code combinations.

Having encoded all the states 1, 2, ..., $2^{\ell+1} - 2$ by different code vectors, we are left with two unused combinations of length $\ell + 1$. One of them should be used to encode the state 0. This completes the coding. As a result, all the transitions originating for the same input state are separated and all the states are distinct. This completes the proof of the proposition.

As an example, let us encode the transition table in Table 1. After separation of transitions, the states have the following coding (the corresponding parts of the code vectors are underscored):

0. 0 0000;	8. $\underline{001}11$	16. $\underline{000}$ 01	24. $\underline{100}$ 01
1. $\underline{0}$ 1000	9. $\underline{010}11$	17. $\underline{001}$ 10	25. $\underline{101}$ 10
2. $\underline{1}$ 1000	10. $\underline{011}11$	18. $\underline{001}$ 01	26. $\underline{101}$ 01
3. $\underline{00}$ 100	11. $\underline{100}11$	19. $\underline{010}$ 10	27. $\underline{110}$ 10
4. $\underline{01}$ 100	12. $\underline{101}11$	20. $\underline{010}$ 01	28. $\underline{110}$ 01
5. $\underline{10}$ 100	13. $\underline{110}11$	21. $\underline{011}$ 10	29. $\underline{111}$ 10
6. $\underline{11}$ 100	14. $\underline{111}11$	22. $\underline{011}$ 01	30. $\underline{111}$ 01
7. $\underline{00\ 0}11$	15. $\underline{000}10$	23. $\underline{100}$ 10	

Figure 2 is the graph corresponding to Table 1, with the state coding shown.

Examining the underscored parts of the code vectors, we conclude that the states 7, 15, 16, say, have the same combination 000 of the first three symbols. Two symbols are sufficient to distinguish between these three states, and the same is true for the states 8, 17, 18. The states 3, 7, 8, 15-18 have the same combination 00 of the first two symbols. Three symbols are sufficient to distinguish between these seven states. However, the states 7, 16, 15, just like 8, 17, 18, have been previously distinguished using three symbols. For the seventh state 3 we may take any previously unused combination of three symbols, 100, say.

The same technique is applied to all the paths with initials state 4. Then we see from Table 1 that all the states 1, 3, 4, 7-10, 15-22 have the same symbol in the first place. But all the states 3, 7, 8, 15-18 have been separated from the states 4, 9, 10, 19-22 by the second symbol already in the stage of separation of transitions, and also within each of these groups of states by the preceding construction.

Thus, 14 states have already been distinguished by 14 different combinations of four symbols. Two combinations remain, one of which, 1000, is assigned to the first symbol 0 of state 1. The same technique is applied to all the paths with intiial state 2. After that, all the states which belong to paths starting in state 1 and in state 2 are distinguished form one another and from states which belong to paths with different initial states. Thus, thirty states have been distinguished using thirty different combinations of five symbols, i.e., the smallest possible number of symbols.

Note that transitions inevitably produce intermediate states which are precisely equal to other states of the automaton. For instance, the transition from state 3 to state 7 may produce all the remaining six states with two zeros in the first two bits — the states 0, 8, 15, 16, 17, 18. However, by construction, none of these states appears in any of the transitions with a different initial or a different final state for the same input state.

Note that both Table 1 and the given code are suitable for any system of Boolean functions of four variables with different expansion coefficients in these variables. The difference between systems of functions is reflected only in the combinations of constants corresponding to states 15 to 30.

Also note that if the BP graph does not have the regular structure described above and some of its nodes and edges are combined, the transition table is again made up of "boxes' with indeterminacy outside the boxes. Therefore separation of transitions for identical input states is also sufficient to eliminate critical races. The process of race-proof coding, however, is no longer as transparent and algorithmic as for the case of a complete graph considered above.

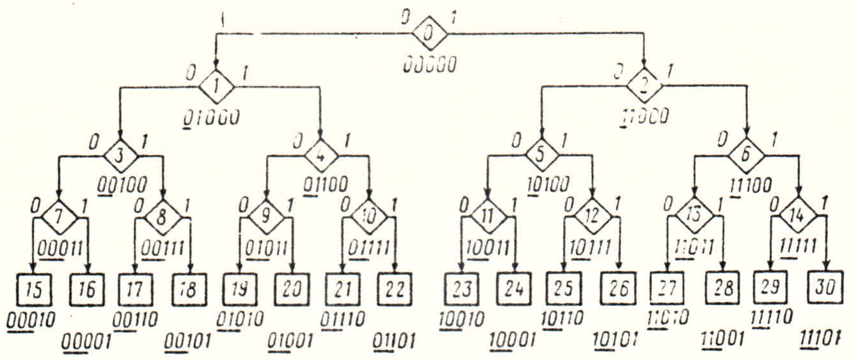

Fig. 2

TABLE 2

State	x_0		x_1		x_2		x_3	
	0	1	0	1	0	1	0	1
0	1	2	–	–	–	–	–	–
1	–	–	3	4	–	–	–	–
2	–	–	4	5	–	–	–	–
3	–	–	–	–	9	10	–	–
4	–	–	–	–	11	6	–	–
5	–	–	–	–	7	8	–	–
6	–	–	–	–	–	–	10	12
7	–	–	–	–	–	–	11	12
8	–	–	–	–	–	–	12	11
9	–	–	–	–	–	–	–	–
10	–	–	–	–	–	–	–	–
11	–	–	–	–	–	–	–	–
12	–	–	–	–	–	–	–	–

For the automaton realizing the BP graph of the system (3) shown in Fig. 1 with the transition table in Table 2, the race-proof code is obtained by dropping the extra columns of the Hadamard matrix of order 12 (see [4, p. 145]). The columns may be dropped by using the descent algorithm proposed by Yankovskaya [5].

The resulting race-proof code is irredundant and has the form

0. 0000	3. 0011	6. 1011	9. 1111	12. 0111.
1. 0010	4. 1010	7. 1101	10. 0001	
2. 1100	5. 0101	8. 0110	11. 1110	

If the automaton logic is now known beforehand, a universal code should be used. Yet the universal code may prove to be redundant. As a universal code with race-proof properties, regardless of the form of the particular 12-state automaton, we may use the code constructed from the Hadamard matrix of order 12 by dropping one bit without applying the descent method (which requires knowledge of the automaton logic). It is shown in [4] that for this code the minimum number of columns of the form 0011 and 1100 in any ordered 4-tuple of vectors S_1, S_2, S_3, S_4 is not less than 1, since the code is equidistant with distance $d = 6$. Note that the universal code length is a rough upper bound on code length.

Several techniques are available for combining race-proof coding with noise-tolerant coding. The first and simplest approach repeats each code combination $2t + 1$ times. This ensures tolerance to any t or fewer errors under conditions with races of memory elements. The second approach applies the universal race-proof and noise-tolerant code [4] with subsequent elimination of the extra columns. Both approaches involve using enumerative methods, which are described in [4, 6]. The impact of the specific features of the graph of the binary program automaton is apparently most pronounced in race-proof coding.

4. CONCLUSION

We have proposed minimum race-proof coding of the automaton realizing the binary program for a system of Boolean functions with a specific property: its graph is a complete tree.

The length of the race-proof code in the case of a complete tree is a trivial upper bound on the length of the race-proof code of the states of an arbitrary acyclic graph realizing the binary program for a system of Boolean functions in which the number of variables is equal to the number of levels.

We would like to propose the conjecture that, for any automaton graph (for any system of functions realized by this automaton), the race-proof code length is equal to the minimum length of the code which only performs state discrimination.

LITERATURE CITED

1. O. P. Kuznetsov, "On program realization of logical functions and automata. I," Avtomat. Telemekh., No. 7, 163-174 (1977).
2. V. A. Kuz'min, "Realization of Boolean functions by automata, normal algorithms, and Turing machines," in: Problems of Cybernetics [in Russian], No. 13, Fizmatgiz, Moscow (1965), pp. 75-96.
3. A. Sh. Blokh, Graph-Schemata and Their Application [in Russian], Visheishaya Shkola, Minsk (1975).
4. Yu. L. Sagalovich, State Coding and Reliability of Automata [in Russian], Svyaz', Moscow (1975).
5. A. E. Yankovskaya, "Descent algorithms for the solution of some problems of the design of discrete devices and their applications," in: Theory of Discrete Control Devices [in Russian], Nauka, Moscow (1982), pp. 206-214.
6. A. D. Zakrevskii and A. E. Yankovskaya, "Noise-tolerant coding of internal states of asymchronous automata," Information Papers [in Russian], No. 3(50), Nauchnyi Sovet po Kompleksnoi Problems "Kibernetika," AN SSSR, Moscow (1971), pp. 53-58.