Programming Calculations in Many-Dimensional Boolean Space

Arkadij Zakrevskij

Abstract — **The set of macro operations POBS over Boolean 2^n-component vectors is offered, which essentially facilitates programming calculations in many-dimensional Boolean space. The application of that set is illustrated by examples of the analysis of partial Boolean functions on a monotony and presence of functional regularities, solving problems of sequential composition and decomposition. An important role is played by operations of interaction between adjacent units of the space.**

Index Terms — **Boolean space, programming combinatorial problems, efficient macro operations.**

I. Introduction

A set composed of 2^n Boolean n-component vectors is called the Boolean n-dimensional space. The relation of neighborhood is defined there – two vectors are called *adjacent*, if they differ by values exactly in one component. This relation can be represented by a graph, where nodes correspond to elements of the Boolean space, and edges join nodes corresponding to adjacent elements. Such graphs are widely used in the educational literature for the description of methods of Boolean functions minimization and solution of other logical design tasks. However, already at n>5 the graph image becomes too complicated and inconvenient for practical usage which is illustrated by Fig. 1.

More acceptable from the programming point of view is the representation of n-dimensional Boolean space as a Boolean 2^n-vector, i.e. a vector with 2^n components corresponding to elements of the space. These components are numbered starting with zero: component with number k corresponds to an element of the considered Boolean space which represents n-component Boolean vector specifying the binary code of number k. Assigning to components of 2^n-vector the values from set {0, 1}, it is possible to set any Boolean function of n variables. For example, the Boolean vector

$$\boldsymbol{f} = 10000000\ 00000000\ 00001000\ 00000001\ 10000000\ 00000000\ 00110000\ 00000001$$

defines a Boolean function f of six variables x_1, x_2, x_3, x_4, x_5, x_6, receiving the value 1 on the following sets of their values: 000000, 010100, 011111, 100000, 110010, 1100011, 111111.

For convenience of visual perception the vector $\boldsymbol{f}$ is divided into eight fragments corresponding to intervals of the Boolean space with internal variables x_4, x_5, x_6. These fragments represent coefficients of disjunctive Shannon decomposition of the function $\boldsymbol{f}$ by variables x_1, x_2, x_3.

Boolean 2^n-vectors serve as main objects of conversions performed at solution of manifold logic combinatorial tasks, which arise at design of discrete devices and developing systems of artificial intelligence [1]. With the purpose of raising the efficiency of their programming a basic set of macro operations over such vectors is offered in this paper. It is called POBS (Parallel Operations in Boolean Space).

As experience shows, the set POBS appears rather efficient, allowing fast operating on a modern PC with long Boolean vectors representing arbitrary Boolean functions of many variables, up to 27 including. A row presenting such a vector (containing 2^{27} = 134 217 728 characters) would need a paper strip of length more than 250 kilometers.

II. Component-wise operations over Boolean vectors

The elementary operations of the set POBS are component-wise operations: the operation of inverting over one vector and arbitrary two-place Boolean operations over two vectors of the same size. We illustrate them by the following examples.

Designate as $\boldsymbol{g}$ and $\boldsymbol{h}$ the Boolean vectors representing Boolean functions $g(\boldsymbol{x})$ and $h(\boldsymbol{x})$, where $\boldsymbol{x} = (x_1, x_2, x_3, x_4, x_5)$. Let

$$\boldsymbol{g} = 10001100\ 00100010\ 11100001\ 00101010,$$
$$\boldsymbol{h} = 00110010\ 10001111\ 01100011\ 01100011.$$

Then

$$\overline{\boldsymbol{g}} = 01110011\ 11011101\ 00011110\ 11010101,$$
$$\boldsymbol{g} \vee \boldsymbol{h} = 10111110\ 10101111\ 11100011\ 01101011,$$
$$\boldsymbol{g} \wedge \boldsymbol{h} = 00000000\ 00000010\ 01100001\ 00100010,$$
$$\boldsymbol{g} \oplus \boldsymbol{h} = 10111110\ 10101101\ 10000010\ 01001001, \text{ etc.}$$

Manuscript received February 20, 2008.

This work was supported in part by the Belarusian Republican Fond of Fundamental Researches (Project Ф07МС-034).

Arkadij Zakrevskij is with the United Institute of Informatics Problems of the National Academy of Sciences of Belarus, 220023, Minsk; e-mail: zakr@newman. bas-net. by.

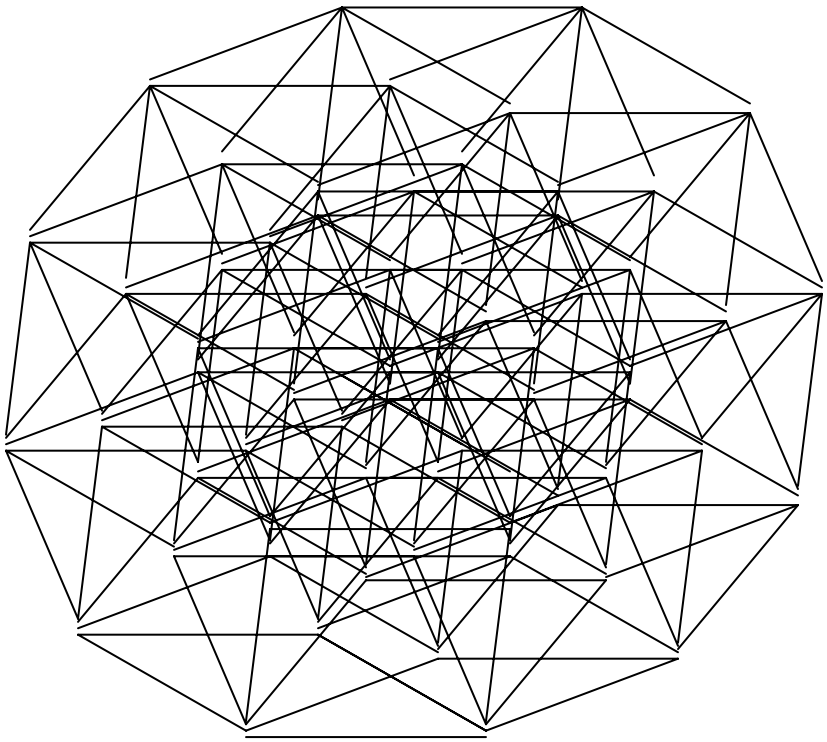

Fig. 1. Six-dimensional cube

More complicated is the operation of *permutation of arguments* of a Boolean function, which is presented by a Boolean 2^n-vector. It is defined by a permutation on the set of variable numbers and results in the appropriate permutation of components of the vector.

For example, as a result of permutation of numbers (4, 2, 1, 3), the variables of the set $\boldsymbol{x} = (x_1, x_2, x_3, x_4)$ will rearrange in a new sequence (x_4, x_2, x_1, x_3). It leads to the appropriate permutation of components of vector $\boldsymbol{f}$, representing the Boolean function $f(\boldsymbol{x})$. The component f_k is relocated in place with number i, if the binary code $\boldsymbol{k}$ of number k represents the result of multiplying the permutation matrix $\boldsymbol{P}$ by the binary code $\boldsymbol{i}$ of number i. In other words, the vector $\boldsymbol{k}$ is equal to the component-wise disjunction of columns of matrix $\boldsymbol{P}$, marked with ones in vector $\boldsymbol{i}$. It is illustrated below by the example of substitution of the component f_{13} by f_{14}:

	$\boldsymbol{P}$		$\boldsymbol{i}$		$\boldsymbol{k}$
1	**0 0 0 1**		**1**		**1**
2	**0 1 0 0**		**1**		**1**
3	**1 0 0 0**	×	**0**	=	**1**
4	**0 0 1 0**		**1**		**0**
	1 2 3 4		13		14

Thus, permutation (4, 2, 1, 3) on the set of components of vector $\boldsymbol{x}$ results in the following permutation on the set of components of vector $\boldsymbol{f}$:

(0, 8, 1, 9, 4, 12, 5, 13, 2, 10, 3, 11, 6, 14, 7, 15).

Let $\boldsymbol{f}$ = 0111 1010 0100 1001. Then the considered permutation on the set of components of this vector results in its new value $\boldsymbol{f}^*$ =0100 1110 1110 0001, corresponding to the new order (x_4, x_2, x_1, x_3) of arguments.

III. Operations over adjacent elements of Boolean space

The set POBS contains also operations of interaction between different components of one Boolean 2^n-vector $\boldsymbol{f}$, specifying a function $f(\boldsymbol{x}) = f(x_1, x_2, \ldots, x_n)$. Most important are the *operations over adjacent elements* of Boolean space. Boolean n-vectors are adjacent if they differ by their values only in one component. For representation of Boolean space as a many-dimensional Boolean cube such vectors are presented by nodes joint with edges.

Usage of such structure allows to perform parallel logical calculations and by that to accelerate them considerably. This idea is not new. So, a special supplement to the universal computer, called L-machine, was developed in 1961-1962 in the Siberian Physical-Technical Institute, which essentially accelerates the process of solving logical design problems [2]. The basic idea consists in executing of distributed in a Boolean space logic operations on a series of information fields playing the role of some registers. The fields are structurally similar to ten-dimensional Boolean cubes and allow to represent immediately Boolean functions of ten variables and complete component-wise operations over them. One of the fields is named main, and is used for conversions limited by one function. The circuit implementation of the main field provides simultaneous execution of any two-place Boolean operation within of each from 512 couples of elements adjacent by some selected variable.

The same idea was put in the basis of the commutative computer offered by W.D. Hillis in 1978 and designed by Thinking Machine Corporation in 1985. This computer provides information exchange in a multiprocessor computer, which components are immediately connected with each other similarly to nodes of the n-dimensional Boolean cube. Transfer of a portion of information between any two processors in such computer takes no more than n time clicks. Several commutative computers where created and used by the researches working in the field of artificial intelligence to solve the logic inference problems [3].

When the number of arguments n exceeds 5 it is convenient to set vector $\boldsymbol{f}$ by a Boolean matrix of size $2^5 \times 2^{n-5}$, representing its 32-element rows by words in the computer memory (what is adequate for the majority of modern computers). In this case any two units of the space M adjacent by the variable x_k belong to the same word if $k <$

6, and belong to different words otherwise, that should be taken into account at programming. Let's remark, that in presented below examples it is more convenient for visual perception to use matrices by the size $2^4 \times 2^{n-4}$.

We introduce the following elementary operations of conversion of a Boolean function $f(\mathbf{x}) = f(x_1, x_2, \ldots, x_n)$ presented with vector $\boldsymbol{f}$, by interaction of adjacent units:

$\boldsymbol{f} - k$ – assignment of value 0 to argument x_k,
$\boldsymbol{f} + k$ – assignment of value 1 to argument x_k.

These operations are illustrated by the following matrices, splitting the set of elements of vector $\boldsymbol{f}$ into two parts corresponding to different values of the selected variable x_k and called below conditionally *left* (marked bold font for x_4) and *right*:

$\boldsymbol{f}$	$\boldsymbol{f} - 4$
0110110101011110	**0110011001010101**
0010010000010110	**0010001000010001**
1100101001110001	**1100110001110111**
0100010111010011	**0100010011011101**
1101110111101110	**0110110101011110**
0100010001100110	**0010010000010110**
1010101000010001	**0110110101011110**
0101010100110011	**0010010000010110**
1 2 $\boldsymbol{f} + 4$	$\boldsymbol{f} - 1$

At execution of the operation $\boldsymbol{f} - k$ both elements in each couple adjacent by the variable x_k accept value from the left part, at execution of the operation $\boldsymbol{f} + k$ – from the right part. If $n < 6$ (in the given example if $n < 5$) this operation is implemented by means of appropriate shift of columns in the matrix, otherwise – of rows.

By way of generalization we shall enter the following operations, in which instead of the scalar k the n-component Boolean vector $\boldsymbol{u}$ is used:

$\boldsymbol{f} - \boldsymbol{u}$ – assignment of value 0 to all arguments x_k, which correspond to 1-components (having value 1) u_k of vector $\boldsymbol{u}$,
$\boldsymbol{f} + \boldsymbol{u}$ – assignment of value 1 to all arguments x_k, which correspond to 1-components u_k of vector $\boldsymbol{u}$.

The first of these operations can be interpreted as obtaining the initial coefficient f_0 of disjunctive Shannon decomposition of function f by all variables of the set $\boldsymbol{u}$ (in this case all variables receive value 0), the second – as obtaining the finite coefficient f_1 (when all variables receive value 1).

For example, if $n = 8$ and $\boldsymbol{u} = 01100010$, the operation $\boldsymbol{f} - \boldsymbol{u}$ is equivalent to the composition $((\boldsymbol{f} - 2) - 3) - 7$, and the operation $\boldsymbol{f} + \boldsymbol{u}$ is equivalent to the composition $((\boldsymbol{f} + 2) + 3) + 7$. Thus the same value is assigned to variables x_2, x_3 and x_7.

Let's introduce also the operation of symmetrization $S\,\boldsymbol{f} * k$, at which execution the both adjacent by variable x_k elements in each couple gain an identical value, as a result of application of the operator *, selected from the set $\{\vee, \wedge, \rightarrow, \oplus, \ldots\}$, to the initial values of the elements of a couple. This operation also is reduced to the surveyed above, as

$$S\,\boldsymbol{f} * k = (\boldsymbol{f} - k) * (\boldsymbol{f} + k).$$

For example,

$\boldsymbol{f}$	$S\,\boldsymbol{f} \oplus 1$
0110110101011110	**1010011100101111**
0010010000010110	**0110000111000101**
1100101001110001	**1010011100101111**
0100010111010011	**0110000111000101**
0111111101111111	**0000110000001100**
0011011000110110	**0000000000000000**
1111101111111011	**1100000000110000**
1101011111010111	**0000000011000011**
1 2 $S\,\boldsymbol{f} \vee 3$	$S\,\boldsymbol{f} \wedge 6$

In particular, operation $S\,\boldsymbol{f} \oplus k$ represents the well known operation of derivation of a Boolean function by the variable x_k.

The operation of symmetrizing $S\,\boldsymbol{f} * k$ also is generalized by usage of a vector $\boldsymbol{u}$ instead of a scalar k. It is represented by expression $S\,\boldsymbol{f} * \boldsymbol{u}$ and is equivalent to the sequence of operations $S\,\boldsymbol{f} * k_i$, in which scalars k_i represent the numbers of 1-components of vector $\boldsymbol{u}$. In this case operator * is selected from the set $\{\vee, \wedge, \oplus\}$.

For example, if $\boldsymbol{u} = 010011$, the operation $S\,\boldsymbol{f} \wedge \boldsymbol{u}$ is equivalent to the expression

$$S(S(S\,\boldsymbol{f} \wedge 2) \wedge 5) \wedge 6.$$

It can be interpreted in such a way: all elements of the fragment of vector $\boldsymbol{f}$, corresponding to conjunction $\bar{x}_1 \bar{x}_3 \bar{x}_4$, gain value 0, if even one of them was equal to 0.

IV. Operations of Conversion of Dimension of Boolean Vectors

Such operations allow to implement interaction between Boolean vectors of different dimension.

Consider two Boolean vectors:

n-vector $\boldsymbol{u}$ with k ones marking some k variables from the set $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$,

2^k-vector $\boldsymbol{h}$, specifying the Boolean function h of the marked variables.

We introduce into set POBS the operation $\boldsymbol{h} \times \boldsymbol{u}$ of transfer of the function h into a fragment of the Boolean space of n variables, which corresponds to the conjunction of inversions of not marked in $\boldsymbol{u}$ variables. By that all elements of remaining fragments gain value 0.

Let, for example, $n = 5$, $\boldsymbol{u} = 01101$ and $\boldsymbol{h} = 10010011$.

Then a Boolean 2^5-vector is created, in which the fragment corresponding to the conjunction $\bar{x}_1 \bar{x}_4$ is selected (marked with bold):

000000**00** **00**0000**00** 00000000 00000000,

and the vector $\boldsymbol{h}$ is inscribed in it. As a result, the following vector is obtained:

$\boldsymbol{h}\times\boldsymbol{u}$= **10**0000**1**00 **00**0011**0**0 00000000 00000000

The operation $\boldsymbol{f} : \boldsymbol{u}$ is introduced by analogy: it realizes a return carry of the information from the selected fragment of 2^n-vector to the vector $\boldsymbol{h}$, specifying obtained by that Boolean function h of k variables. So, if $\boldsymbol{u}$ = 01010 and

$\boldsymbol{f}$ = 00110010 11100000 11100110 00011101

then the fragment is found, which corresponds to conjunction $\bar{x}_1 \bar{x}_3 \bar{x}_5$

00110010 **11**100000 11100110 00011101

and the information contained in it is used for build-up of the required vector:

$$\boldsymbol{h} = \boldsymbol{f} : \boldsymbol{u} = 0111.$$

If it is known, that function f represented by vector $\boldsymbol{f}$ depends only on variables of the set $\boldsymbol{u}$ (the rest variables appear fictitious), then by means of the operation $\boldsymbol{f} : \boldsymbol{u}$ the latter are deleted from the function and the result is represented by vector $\boldsymbol{h}$.

V. Operations over partial Boolean functions

Let's pass to reviewing *partial* (not completely defined) Boolean functions widely used when solving problems of logical design.

Any arbitrary partial Boolean function f of n variables can be represented by a corresponding 2^n-component ternary vector $\boldsymbol{f}^-$. For programming it is more convenient to set it by a couple of Boolean vectors $\boldsymbol{f}^1$ and $\boldsymbol{f}^0$, also 2^n-component. By that ones in the vector $\boldsymbol{f}^1$ mark components, on which the function f receives value 1, and in the vector $\boldsymbol{f}^0$ they mark components, where the function is equal to zero. In other words, the vectors $\boldsymbol{f}^1$ and $\boldsymbol{f}^0$ represent accordingly characteristic sets M^1 and M^0 of the function f.

Let, for example,

$$\boldsymbol{f}^- = 1\text{-}001010\ 011\text{--}01\text{-}1.$$

Then

$$\boldsymbol{f}^1 = 10001010\ 011000101,$$
$$\boldsymbol{f}^0 = 00110101\ 100001000.$$

In this case the operation of assignment of value 0 to argument x_k will be represented by the couple of operations $\boldsymbol{f}^1 - k$, $\boldsymbol{f}^0 - k$, and that of value 1 – by the couple $\boldsymbol{f}^1 + k$, $\boldsymbol{f}^0 + k$.

Similarly to $\boldsymbol{f}$, the Boolean vector $\boldsymbol{u}$ also can appear ternary, for example when representing some elementary conjunction Then it also should be replaced by a pair of Boolean vectors $\boldsymbol{u}^1$ and $\boldsymbol{u}^0$, in this case n-component. For example, in the operation of disjunctive decomposition of a partial Boolean function f by all variables of the united set $\boldsymbol{u}^1 \cup \boldsymbol{u}^0$ the obtaining of the coefficient at that conjunction is carried out by the series of operations

$$\boldsymbol{f}^1 - \boldsymbol{u}^0,\ \boldsymbol{f}^0 - \boldsymbol{u}^0,\ \boldsymbol{f}^1 + \boldsymbol{u}^1,\ \boldsymbol{f}^0 + \boldsymbol{u}^1.$$

VI. Programming in basis POBS

Let's show some examples of using a software technology in basis POBS to solve tasks of the theory of Boolean functions.

A.. Testing a partial Boolean function on monotony

Consider a partial Boolean function $f(\boldsymbol{x}) = (x_1, x_2, \ldots, x_n)$, given by two sets of argument values collections: by the set M^1, on which it receives value 1, and the set M^0, where it receives value 0. Let's term it as *monotone* or, in particular, a *positive* function, if for any couple of collections $\boldsymbol{p} \in M^1$ and $\boldsymbol{q} \in M^0$ condition $\boldsymbol{p} > \boldsymbol{q}$ (vector $\boldsymbol{p}$ is greater than vector $\boldsymbol{q}$) is satisfied, i.e. for any couple (p_i, q_i) of vector components $p_i \geq q_i$ and at least for one couple $p_i > q_i$.

A simple method of checking the function for monotony could be applied, which is based on exhaustive search of all couples $(\boldsymbol{p}, \boldsymbol{q})$ and testing each of them on satisfying the condition $\boldsymbol{p} > \boldsymbol{q}$. However, with increase of the number of variables n and corresponding growth of power of sets M^1 and M^0 such method appears too labour-consuming. The offered below method using operations from the set POBS is more efficient.

Let's set the function $f(\boldsymbol{x})$ by two Boolean 2^n-vectors: $\boldsymbol{f}^1$ and $\boldsymbol{f}^0$. The elements of set M^1 are represented by ones in vector $\boldsymbol{f}^1$, the elements of set M^o - by ones in vector $\boldsymbol{f}^0$. Designate as M^* the set of such elements of Boolean space, each of which is greater than some element from set M^1 or equals it, and present this set by vector $\boldsymbol{f}^*$.

The affirmation 1. The function $f(\boldsymbol{x})$ is monotone, if and only if $\boldsymbol{f}^* * \boldsymbol{f}^0 = \boldsymbol{0}$.

The vector $\boldsymbol{f}^*$ can be found with the help of introduced above operations of the set POBS, by a sequence of n steps. At first we receive the vector $\boldsymbol{f}^2 = (\boldsymbol{f}^1 - 1) \vee \boldsymbol{f}^1$, presenting set M^1, supplemented with elements of Boolean space, adjacent "from above" to some elements from M^1 by variable x_1. Then the obtained set is expanded similarly by the next variable x_2: $\boldsymbol{f}^3 = (\boldsymbol{f}^2 - 1) \vee \boldsymbol{f}^2$. After iterating this operation by all remaining variables we receive the required vector $\boldsymbol{f}^{n+1} = \boldsymbol{f}^*$.

Let, for example, n = 5,

$$\boldsymbol{f}^1 = 00010000\ 00100000\ 00000001\ 00001010,$$
$$\boldsymbol{f}^0 = 11000010\ 00000100\ 10100000\ 10000000.$$

In this case the process of the sequential extension of set M_1, resulting in obtaining vector $\boldsymbol{f}^*$ representing set M^*, can be demonstrated by the following sequence of vectors obtained on the next steps:

$$\boldsymbol{f}^2 = 00010000\ 00100000\ 00010001\ 00101010,$$

$\boldsymbol{f}^3$ = 00010000 00110000 00010001 00111011 ,
$\boldsymbol{f}^4$ = 00010001 00110011 00010001 00111011 ,
$\boldsymbol{f}^5$ = 00010001 00110011 00010001 00111011 ,
$\boldsymbol{f}^6$ = 00010001 00110011 00010001 00111111= $\boldsymbol{f}$*

Component-wise conjunction of the obtained vector with vector

11000010 00000100 10100000 10000000 = $\boldsymbol{f}^0$

is equal to zero ($\boldsymbol{f}^* \wedge \boldsymbol{f}^0 = \boldsymbol{0}$), therefore, the considered Boolean function is monotone.

B. Search for functional regularities

An important role in modern information technologies is played by procedures of data mining, i.e. extraction of knowledge from the dataflow, search of regularities allowing discovering right decisions at solution of the intellectual tasks [4]. A special but important case of regularities is considered below, namely functional regularities, often encountered in natural sciences.

The following formal task was considered in [5]. We assume that a set of objects is preset, each of which is characterized by some combination of n binary values (indicating if the corresponding signs are present or not present). The question is, whether it is possible always to define uniquely the value of some selected sign, if the values of remaining ones are known? And if possible, how to define it?

The initial information in this task can be presented by a collection R of some elements in n-dimensional Boolean space $M = \{0, 1\}^n$ of signs. These elements set known objects and can be considered as the roots of some Boolean equation $F = 1$, where $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$.

This equation is called solvable in regard to some variable, if this variable can be presented by a Boolean function of the remaining variables, which is defined on the set R [5]. We consider the task of detection of such variables in the equation $F = 1$ and finding the appropriate functions.

The affirmation 2. The necessary and sufficient condition of solvability of the equation $F = 1$ in regard to the variable x_i is the absence in the set R of couples of collections, adjacent by x_i.

Proof by contradiction (by the rule *modus tollens*): if there exists such a couple, the variable x_i receives in it different values on identical sets of values of remaining variables, which contradicts the definition of the functional relation.

Let's designate through $f(\boldsymbol{x})$ the characteristic Boolean function of set R, where $f(q_j) = 1$ if $q_j \in R$ and $f(q_j) = 0$ if $\neg (q_j \in R)$. Through $f(x_i = 0)$ and $f(x_i = 1)$ we denote the result of replacement in the function $f(\boldsymbol{x})$ the variable x_i with constant 0 or 1, accordingly.

The affirmation 3. The equation $F = 1$ is solvable in regard to variable x_i, if and only if $f(x_i = 0) \wedge f(x_i = 1) = 0$.

This affirmation allows to apply introduced above vector operations $\boldsymbol{f} - i$ and $\boldsymbol{f} + i$ for checking the equation for solvability in regard to variable x_i. Affirmation 3 can be reformulated in terms of these operations in the following way: the necessary and sufficient condition of solvability of the equation $F = 1$ in regard to variable x_i is the satisfaction of the relation

$$(\boldsymbol{f} - i) \wedge (\boldsymbol{f} + i) = \boldsymbol{0}.$$

In case if this condition is satisfied there arises a task of finding an appropriate Boolean function, which generally appears to be partial, and its optimal determination. The optimization can consist both in minimization of the number of arguments of the function, and in simplification of its algebraic representation, for example in DNF.

Let's consider the first of these tasks. It is similar to the task of minimization of unconditional diagnostic test and can be solved by the same method. Some argument x_k can be defined as fictitious, if after its deleting the equation remains solvable in regard to the variable x_i. The operation of deleting the argument x_k can be presented as the extension of set R by this variable, i.e. as the following conversion of its characteristic function f

$$f := f(x_k = 0) \vee f(x_k = 1).$$

In terms of introduced above vector operations it is defined as $\mathrm{S}\,\boldsymbol{f} \vee k$, whence follows

The affirmation 4. The argument x_k can be deleted from the set of arguments of the variable x_i defined as a function of remaining variables, if and only if

$$((\mathrm{S}\,\boldsymbol{f} \vee k) - i) \wedge ((\mathrm{S}\,\boldsymbol{f} \vee k) + i) = \boldsymbol{0}.$$

C. Sequential composition of Boolean functions

Let's consider the following task. The set of arguments $\boldsymbol{x}=(x_1, x_2, \ldots, x_n)$ is divided by the Boolean n-vectors $\boldsymbol{u}$, $\boldsymbol{w}$ and $\boldsymbol{v}$ into three not intersected subsets $\boldsymbol{u}$, $\boldsymbol{w}$ and $\boldsymbol{v}$: $\boldsymbol{x} = \boldsymbol{u} \cup \boldsymbol{w} \cup \boldsymbol{v}$. Two Boolean functions $h(\boldsymbol{u}, \boldsymbol{w})$ and $g(x, \boldsymbol{w}, \boldsymbol{v})$, presented with corresponding Boolean vectors $\boldsymbol{h}$ and $\boldsymbol{g}$ are given also. It is required to calculate their composition under condition $x = h(\boldsymbol{u}, \boldsymbol{w})$ and to present the obtained Boolean function $f(\boldsymbol{x})$ by a 2^n-vector $\boldsymbol{f}$.

Such composition called non-disjoint sequential two-block, is illustrated by an example on fig. 2, where $n = 6$ and the sets $\boldsymbol{u} = (x_1, x_2)$, $\boldsymbol{w} = (x_3, x_4)$ and $\boldsymbol{v} = (x_5, x_6)$ are presented by six-dimensional Boolean vectors $\boldsymbol{u}$ = 110000, $\boldsymbol{w}$ = 001100 and $\boldsymbol{v}$ = 000011.

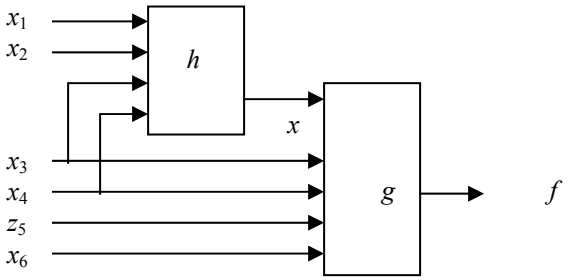

Fig. 2. An example of non-disjoint sequential two-block composition

Let's assume, that the functions $h(\boldsymbol{u}, \boldsymbol{w})$ and $g(x, \boldsymbol{w}, \boldsymbol{v})$ are preset by corresponding vectors:

```
h = 1101001001101100
g = 0011010011001001 1010010110101011
           g0                g1
```

For convenience, the vector $\boldsymbol{g}$ is broken in two halves, specifying values of the function $g(x, \boldsymbol{w}, \boldsymbol{v})$ at values 0 and 1 of binary variable x.

We present the Boolean space of variables $\boldsymbol{x} = (\boldsymbol{u}, \boldsymbol{w}, \boldsymbol{v})$ as follows:

```
                    ----  ----   w1
              ----        ----  w2
        --     --    --     --  v1
       - -   - -   - -   - -  v2
      0000 0000 0000 0000
    | 0000 0000 0000 0000
   |  0000 0000 0000 0000
   || 0000 0000 0000 0000
    u2
   u1
```

Then we sequentially map onto this space functions g, h_0 and h_1, introducing thus additional fictitious variables from the sets $\boldsymbol{v}$ and $\boldsymbol{u}$ and representing results by 2^n-vectors $\boldsymbol{a}$, $\boldsymbol{b}$ and $\boldsymbol{c}$:

$$\boldsymbol{h} \times (\boldsymbol{u}, \boldsymbol{w}) - \boldsymbol{v} = \boldsymbol{a}$$

```
1000 1000 0000 1000  1111 1111 0000 1111
0000 0000 1000 0000  0000 0000 1111 0000
0000 1000 1000 0000  0000 1111 1111 0000
1000 1000 0000 0000  1111 1111 0000 0000
```

$$\boldsymbol{g}_0 \times (\boldsymbol{w}, \boldsymbol{v}) - \boldsymbol{u} = \boldsymbol{b}$$

```
0011 0100 1100 1001  0011 0100 1100 1001
0000 0000 0000 0000  0011 0100 1100 1001
0000 0000 0000 0000  0011 0100 1100 1001
0000 0000 0000 0000  0011 0100 1100 1001
```

$$\boldsymbol{g}_1 \times (\boldsymbol{w}, \boldsymbol{v}) - \boldsymbol{u} = \boldsymbol{c}$$

```
1010 0101 1010 1011  1010 0101 1010 1011
0000 0000 0000 0000  1010 0101 1010 1011
0000 0000 0000 0000  1010 0101 1010 1011
0000 0000 0000 0000  1010 0101 1010 1011
```

In summary we discover the vector $\boldsymbol{f}$, representing the required composition of functions $h(\boldsymbol{u}, \boldsymbol{w})$ and $g(x, \boldsymbol{w}, \boldsymbol{v})$:

```
0011 0100 0000 1001
0000 0000 1100 0000     a b
0000 0100 1100 0000
0011 0100 0000 0000

0000 0000 1010 0000
1010 0101 0000 1011     ¬a c
1010 0000 0000 1011
0000 0000 1010 1011

0011 0100 1010 1001
1010 0101 1100 1011     f = a b ∨ ¬a c
1010 0100 1100 1011
0011 0100 1010 1011
```

D. Testing a partial Boolean function on decomposability at a given partition on the set of arguments

Suppose that a partial Boolean function $f(\boldsymbol{x})$ of n variables, represented by a ternary vector $\boldsymbol{f}^{-}$ is known. It is required to test it on decomposability at a given partition $\boldsymbol{u}/\boldsymbol{v}$ of the set $\boldsymbol{x}$, i.e. to find out, whether there exist such functions $h(\boldsymbol{u}, \boldsymbol{w})$ and $g(x, \boldsymbol{w}, \boldsymbol{v})$ of smaller number of variables, that $f(\boldsymbol{x}) = g(h(\boldsymbol{u}, \boldsymbol{w}), \boldsymbol{w}, \boldsymbol{v})$, where $\boldsymbol{w} = \boldsymbol{x} \setminus (\boldsymbol{u} \cup \boldsymbol{v})$.

At the positive answer to this question the logic circuit implementing function $f(\boldsymbol{x})$ can be simplified (for example, at logical synthesis in the basis of units LUT (look up tables), implementing functions of restricted number of variables).

The necessary and sufficient condition of decomposability of a completely defined Boolean function $f(\boldsymbol{x})$ at a partition $\boldsymbol{u}/\boldsymbol{v}$, which should be fulfilled for each coefficient $f_i(\boldsymbol{u}, \boldsymbol{v})$ of disjunctive Shannon decomposition of the function $f(\boldsymbol{x})$ by variables of the set $\boldsymbol{w}$ is the following. Each of the coefficients of alike decomposition of these coefficients by variables of the set $\boldsymbol{u}$ should receive no more than two different values.

The coefficients $f_i(\boldsymbol{u}, \boldsymbol{v})$ of disjunctive Shannon decomposition of a partial Boolean function $f(\boldsymbol{x})$ by variables of set $\boldsymbol{w}$ are represented by fragments $\boldsymbol{T}_i$ - ternary matrices, which rows correspond to different values of vector $\boldsymbol{u}$, and columns correspond to different values of vector $\boldsymbol{v}$. The corresponding components of the ternary vector $\boldsymbol{f}^{-}$ serve as elements of fragments. The condition of decomposability of the function $f(\boldsymbol{x})$ at the partition $\boldsymbol{u}/\boldsymbol{v}$ can be formulated now as follows: for each coefficient $f_i(\boldsymbol{u}, \boldsymbol{v})$ such predetermination of the appropriate matrix $\boldsymbol{T}_i$ is possible (replacement of values "–" by 0 or 1), at which its rows will receive no more than two different values.

It was shown in [6], that the check of this condition is reduced to finding out if the graph of orthogonality of rows of each matrix $\boldsymbol{T}_i$ is bichromatic. A heuristic algorithm was suggested there, which guarantees obtaining exact solutions under condition of connectivity of the considered graphs (this condition is usually fulfilled). The ternary vector $\boldsymbol{f}^{-}$ is represented in it by an appropriate couple of Boolean vectors $\boldsymbol{f}^{1}$ and $\boldsymbol{f}^{0}$, and the operations over the neighbors are effectively used providing simultaneous testing of all $2^{|\boldsymbol{w}|}$ fragments $\boldsymbol{T}_i$.

The algorithm tries to divide the set of rows in each fragment into two classes A and B of mutually compatible rows. A sequence of conversions is implemented over the initial vectors $\boldsymbol{f}^{1}$ and $\boldsymbol{f}^{0}$, which results are represented by Boolean 2^n-vectors $\boldsymbol{a}^{1}$ and $\boldsymbol{a}^{0}$.

The algorithm is iterated. The first iteration starts with build-up of the class A by inclosing in it the first row of the fragment. This operation is reduced to a sequence of substitutions of value 0 for the variables from set $\boldsymbol{u}$.

$$\boldsymbol{a}^{0} := \boldsymbol{f}^{0} - \boldsymbol{u}$$
$$\boldsymbol{a}^{1} := \boldsymbol{f}^{1} - \boldsymbol{u}$$

Then in each fragment the rows orthogonal to the first one are found and marked with 1 in the Boolean vector $\boldsymbol{b}$.

$$\boldsymbol{b} := \mathrm{S}\,(\boldsymbol{h}^{\,0}\boldsymbol{f}^{\,1} \vee \boldsymbol{h}^{\,1}\boldsymbol{f}^{\,0}) \vee \boldsymbol{v}.$$

The obtained sets constitute classes B and are checked for compatibility:

$$\boldsymbol{a}^{\,0} := \mathrm{S}\,(\boldsymbol{f}^{\,0}\boldsymbol{b}) \vee \boldsymbol{u}$$

$$\boldsymbol{a}^{\,1} := \mathrm{S}\,(\boldsymbol{f}^{\,1}\boldsymbol{b}) \vee \boldsymbol{u}$$

If by that $\boldsymbol{a}^{\,0}\boldsymbol{a}^{\,1} \neq \boldsymbol{0}$, some of the considered sets appear incompatible, whence follows, that the graph of orthogonality of rows of the corresponding fragment is not bichromatic and, therefore, the function $f(\boldsymbol{x})$ is not decomposable at the partition $\boldsymbol{u}/\boldsymbol{v}$.

On the other hand, if $\boldsymbol{a}^{\,0}\boldsymbol{a}^{\,1} = \boldsymbol{0}$, the following iteration is implemented. The classes A are supplemented by rows, orthogonal by some of rows of classes B and are checked for compatibility. Then the classes B can be similarly extended, etc. The algorithm terminates after execution of a sufficient number of iterations.

REFERENCES

[1] Zakrevskij A.D Computation in Boolean spaces. In "Logical structure of scientific knowledge". *Moscow: Nauka,* 1965, pp. 292-310 (in Russian).

[2] Zakrevskiy A.D. Machine for the solution of logical problems of the type of the synthesis of relay circuits. – Relay systems and finite automata. *Transl. proceedings., Burrough Corp.,* 1964. pp. 544-557.

[3] W. Daniel Hillis. Connection machine. – *Scientific American,* June 1987, Vol. 256, No 6.

[4] Data mining and knowledge discovery approaches based on rule induction techniques (E. Triantaphyllou and G. Felici, Eds.). – Massive Computing Series, Springer, Heidelberg, Germany, 2006.

[5] Zakrevskij A.D. About solvability of Boolean equations. - *Proceedings of NAS of Belarus*, 2007, Vol. 51, No 5, pp. 44-46 (in Russian).

[6] Arkadij Zakrevskij. A new heuristic algorithm for sequential two-block decomposition of Boolean functions. – *Proceedings of 3rd IFAC Workshop on Discrete Event System Design DESDes'06.* September 26-28, 2006, Rydzyna, Poland. University of Zielona Gora, pp. 13-17.