Spectral Methods for Testing Membership in Certain Post Classes and the Class of Forcing Functions

Ilya Shmulevich, *Member, IEEE*, Harri Lähdesmäki, and Karen Egiazarian, *Senior Member, IEEE*

***Abstract*—Forcing functions represent an important class of Boolean functions that have been extensively studied in the analysis of the dynamics of random Boolean networks as models of genetic regulatory systems. Several other so-called Post classes of Boolean functions are closely related to forcing functions and have been used in learning theory as well as in control systems. We develop novel spectral algorithms to test membership of a Boolean function in these classes. These algorithms are highly efficient and are essential in learning problems, especially in the context of genetic regulatory networks, where the same learning procedures are applied repeatedly.**

***Index Terms*—Forcing function, Post class, spectral algorithm.**

I. Introduction

FORCING functions constitute a special type of Boolean function in which at least one of the input variables is able to determine the value of the output of the function. For example, the function $f(x_1, x_2, x_3) = x_1 + x_2 x_3$, where the addition symbol stands for disjunction and the multiplication for conjunction, is a forcing function, since setting x_1 to 1 guarantees that the value of the function becomes 1 regardless of the value of x_2 or x_3. Although their defining property is quite simple, forcing functions have been implicated in a number of phenomena related to discrete dynamical systems as well as nonlinear filters.

For example, they have been used to study the convergence behavior of an important class of nonlinear digital filters called *stack filters* [1]–[3]. Stack filters are a class of sliding window nonlinear digital filters, defined by monotone Boolean functions.[1] Due to the so-called threshold decomposition property, they are able to operate in the real-valued domain. This filter class includes standard and weighted median filters, some morphological and soft morphological filters, weighted order statistic filters, and several other filter classes [4]. They also perform well in many situations where linear filters fail, such as when impulsive noise is present and sharp edges need to be preserved.

Manuscript received December 9, 2002; revised April 11, 2003. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Steven M. Kay.

I. Shmulevich is with the Cancer Genomics Laboratory, University of Texas M. D. Anderson Cancer Center, Houston, TX 77030 USA.

H. Lähdesmäki is with the Cancer Genomics Laboratory, University of Texas M. D. Anderson Cancer Center, Houston, TX 77030 USA and also with the Institute of Signal Processing, Tampere University of Technology, Tampere 33720, Finland.

K. Egiazarian is with the Institute of Signal Processing, Tampere University of Technology, Tampere 33720, Finland.

Digital Object Identifier 10.1109/LSP.2003.821725

[1]A Boolean function $f(x_1, \ldots, x_n)$ is called monotone if for any two vectors $\alpha = (\alpha_1, \ldots, \alpha_n)$ and $\beta = (\beta_1, \ldots, \beta_n)$ such that $\alpha_i \leq \beta_i$, the relation $f(\alpha) \leq f(\beta)$ holds.

A *root signal* of a given filter is a signal that is invariant to the application of that filter, i.e., the signal remains unchanged. Root signals are important for characterizing nonlinear filters because they represent the 'pass-band' characteristics of a filter, very much like the frequencies that are passed by a linear filter. An important problem is determining whether a given filter will drive any input signal to a root signal after a finite number of passes. Since stack filters can be viewed as discrete dynamical systems, the convergence to root signals can be viewed as the convergence to the fixed points of the system. The set of root signals of a stack filter can also be considered to be its associative memory [5], [6]. Another important task involves designing optimal stack filters with specified associative memories. Forcing functions have been shown to play a considerable role in both of these problems. For example, stack filters defined by forcing functions are known to possess the convergence property [2]. Also, in [5], some learning schemes were proposed to find minimal filters defined by forcing functions. In both of these contexts, it is important to be able to test whether a candidate stack filter, defined by a Boolean function, has the forcing property. As the number of stack filters grows super-exponentially relative to the size of the window, it is imperative to devise fast and highly efficient algorithms to test whether a Boolean function possesses the forcing property.

Perhaps even more prevailing is the role of forcing functions in studying transitions between order and chaos in random Boolean networks [7]–[10], the latter being a well-known model class of genetic regulatory networks. Boolean networks have been one of the most intensively studied models of discrete dynamical systems, enjoying a sustained interest from both the biology and physics communities. Analytical and numerical studies of the relationships between structural organization and dynamical behavior of Boolean networks have yielded important insights into the overall behavior of large genetic networks [7], evolutionary principles [11], [12], and the development of chaos [13], [14]. Although structurally simple, these systems are capable of displaying a remarkably rich variety of complex behavior, having much in common with other dynamical system models. In this model, gene expression is quantized to only two levels: ON and OFF. The expression level (state) of each gene is functionally related to the expression states of some other genes, using Boolean functions. Dynamics are introduced by synchronous updating of all genes. Forcing functions, when used as regulatory control rules, are one of the few known mechanisms capable of preventing chaotic behavior in Boolean networks [7]. In fact, there is overwhelming evidence that forcing functions, which are often called canalizing functions in the genetic network community, are abundantly utilized in

higher vertebrate gene regulatory systems [7]. Indeed, a recent large-scale study of the literature on transcriptional regulation in eukaryotes demonstrated an overwhelming bias toward canalizing rules [15].

Forcing functions exhibit a close relationship to several other classes of Boolean functions, first described by Post [16]. These classes of functions are closed under superposition (or composition) in the sense that a composition of functions all belonging to this class results in another function belonging to the same class. These classes have been applied for synthesis and reliability of control systems [17], [18] as well as in learning theory [19]. Although the relationship between forcing functions and the aforementioned Post classes will become clear in the following sections, we should like to mention that strong experimental evidence exists in support of the latter being intimately related to the phase transition between order and chaos in random Boolean networks [20]. The results of a recent paper [21] can be used to test membership of Boolean functions in all Post classes by means of checking solutions to equations involving the Boolean functions in question. Although the methods are mathematically elegant, they are not computationally efficient. Our approach is based on novel spectral-type algorithms for testing whether a Boolean function belongs to one of the above classes.

In order to study the properties of the various classes of functions discussed above, it is necessary to have efficient algorithms for testing whether a given function belongs to a given class. For example, in the context of learning where we are inferring functions from a specific class as described in [19], in order to test inferential algorithms, efficient methods for deciding whether a given function belongs to such a class are indispensable. In the context of random Boolean networks, a certain proportion of (random) functions in a network is chosen to belong to a specific class and the network is simulated to study its global dynamical behavior. This is repeated in a Monte Carlo-type fashion with many large random networks. This typically involves testing hundreds of thousands of Boolean functions. As we will see, spectral methods enjoy an enormous advantage over direct methods because the same transform matrix can be used over and over again. Spectral algorithms are also considerably more efficient, from the point of view of time complexity, than direct methods [22].

II. Preliminaries

A Boolean function is a mapping $f : \{0,1\}^n \to \{0,1\}$.

Definition 1: A function f is said to be forcing if there exists an $i \in \{1,\ldots,n\}$ and $u,\ v \in \{0,1\}$ such that for all $x_1,\ldots,x_n \in \{0,1\}$, if $x_i = u$ then $f(x_1,\ldots,x_n) = v$. The input variable x_i is called the forcing variable with forcing value u and forced value v.

Equivalently, all vectors $x \in \{0,1\}^n$ for which $f(x) = \overline{v}$ must have $x_i = \overline{u}$, where $\overline{u}$ is the negation of u. When $u = v = 0$, this condition reduces to the Post class,[2] which we will denote as A^∞. Thus, we can define A^∞ as follows.

Definition 2: A Boolean function f belongs to class A^∞ if all vectors on which the function takes on the value 1 have a common component equal to 1. Similarly, we define the Post class a^∞ by replacing ones by zeros in the previous sentence.

For example, the function $f(x_1,x_2,x_3) = x_1 + x_2x_3$ belongs to a^∞, since all the vectors on which the function is equal to 0, namely (000), (001), and (010) have a common component equal to 0 (the first one). For simplicity, we will also say "f is a^∞."a

If we denote the set of forcing functions by $\mathcal{C}$, then it is clear from the above definitions that $A^\infty \cup a^\infty \subseteq \mathcal{C}$. We now define the class A^μ as follows.

Definition 3: A Boolean function f belongs to class A^μ, $\mu \geq 2$, if any μ vectors on which the function takes on the value 1 have a common component equal to 1 (some of these μ vectors may be repeated). The class a^μ is defined by replacing ones by zeros in the previous sentence.

As an example, the function

$$f(x_1,x_2,x_3) = x_1x_2 + x_2x_3 + x_1x_3 \tag{1}$$

is A^2, since any two vectors on which the function is equal to 1 have a common unity component.

Remark 4: It is easy to see that if f is A^{μ_1}, then it is also A^{μ_2}, for any μ_2 satisfying $2 \leq \mu_2 \leq \mu_1$. A similar result holds for a^μ.

Another result, which will be utilized in this letter, is given in [23] as the following proposition.

Proposition 5: If the function $f : \{0,1\}^n \to \{0,1\}$ is A^μ, then for $\mu \geq n$ it is also A^∞.

It is easy to show that if f is A^μ, then any collection of $m \leq \mu$ terms in the disjunctive normal form (DNF)[3] must contain a variable that belongs to each one of those terms. Thus, there is once again an interesting relationship to forcing functions in the sense that the functions defined by those terms are forcing functions. For example, if we take any two terms from the function in (1), such as $x_1x_2 + x_2x_3$, then the Boolean function corresponding to these terms is forcing, with forcing variable x_2.

The remarkable property of the classes a^∞, A^∞, a^μ, A^μ as well as a number of other classes described by Post is that if $f(x_1,\ldots,x_m),\ f_1(x_1,\ldots,x_n),\ldots,f_m(x_1,\ldots,x_n)$ are all members of a given class, then the composition function $f(f_1(x_1,\ldots,x_n),\ldots,f_m(x_1,\ldots,x_n))$ is also a member of the same class. Some other examples include monotone functions, self-dual functions, and linear functions.

III. Testing the Forcing Property

We use the following matrix-based algorithm for testing whether a function $f : \{0,1\}^n \to \{0,1\}$ is forcing. Let $\mathbf{R}_n(0,1)$ be the *Rademacher* (0, 1) matrix of order n [22], whose rows contain all n-element binary vectors, arranged in the usual lexicographical order. As an example

$$\mathbf{R}_3^T(0,1) = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

[2]Typically, notation such as $\langle A^\infty \rangle$ is reserved to denote the *property* of a Boolean function. The original notation used by Post and subsequent authors [23] to denote the *set* of functions with such a property is, for example, F_8^∞. However, in this letter, we will simply use notation such as A^∞ to denote the set of functions with the property $\langle A^\infty \rangle$.

[3]The disjunctive normal form of a Boolean function is a disjunction (OR) of conjunctions (AND) of literals, which are variables or their negations.

where T denotes matrix transpose. The jth row of $\mathbf{R}_n(0,1)$ represents the configuration of input variables yielding the jth truth table value of a Boolean function. Let $\mathbf{f}$ denote the 2^n-element binary column-vector representing the truth table of the Boolean function. We define the *forcing transform* as

$$\mathbf{c}^{(1,1)} = \mathbf{R}_n^T(0,1) \cdot \mathbf{f}. \tag{2}$$

For convenience, let us define three other vectors

$$\mathbf{c}^{(0,1)} = |\mathbf{f}| \cdot \mathbf{1} - \mathbf{c}^{(1,1)} \tag{3a}$$

$$\mathbf{c}^{(0,0)} = \left(2^{n-1} - |\mathbf{f}|\right) \cdot \mathbf{1} + \mathbf{c}^{(1,1)} \tag{3b}$$

$$\mathbf{c}^{(1,0)} = 2^{n-1} \cdot \mathbf{1} - \mathbf{c}^{(1,1)} \tag{3c}$$

where $|\mathbf{f}| = \sum_{k=1}^{2^n} \mathbf{f}_k$ is the Hamming weight of $\mathbf{f}$ and $\mathbf{1}$ is a column vector containing all ones.

Proposition 6: The function f is a forcing function if and only if there exist $u, v \in \{0,1\}$ and $i \in \{1,\ldots,n\}$, such that $c_i^{(u,v)} = 2^{n-1}$. In that case, u is the forcing value of f for variable x_i and v is its forced value.

Proof: The rows of $\mathbf{R}_n(0,1)$ correspond to the n variables of the Boolean function f. If x_i, $i = 1,\ldots,n$, is the forcing variable, then the truth table of f should have the same constant value (0 or 1) in all positions where x_i is equal to some value (0 or 1). Thus, there are four possibilities, corresponding to the four vectors $\mathbf{c}^{(u,v)}$, $u, v \in \{0,1\}$: two possible forcing values and two forced values. By multiplying row i ($i = 1,\ldots,n$) of $\mathbf{R}_n(0,1)$ by the truth table $\mathbf{f}$, as in (2), we essentially count the number of times the function is equal to 1 when $x_i = 1$. Similarly, for the case of $\mathbf{c}^{(0,1)}$, we count the number of times the function is equal to 1 when $x_i = 0$ (or equivalently, when $\overline{x}_i = 1$). The same arguments apply for the cases of $\mathbf{c}^{(0,0)}$ and $\mathbf{c}^{(1,0)}$, except that now we count the number of times the function is equal to 0. Since every row of the transform matrix $\mathbf{R}_n(0,1)$ contains exactly 2^{n-1} ones, a value of $c_i^{(1,1)} = 2^{n-1}$ would indicate that whenever $x_i = 1$, $f(x_1,\ldots,x_i,\ldots,x_n) = 1$. In light of the relationships given in (3a)–(3c), similar results hold for other values of u and v.

The above algorithm is quite efficient, as it can be implemented using binary matrix-vector multiplication algorithms, requiring no loops. In addition, information about all forcing variables along with their forcing values as well as the forced function values, is contained in the vector $\mathbf{c}^{(1,1)}$. Let us give an example.

Example: Let $f(x_1,x_2,x_3) = \overline{x}_1 + x_2x_3$ be a Boolean function. The truth table of f is equal to $\mathbf{f} = [1,1,1,1,0,0,0,1]^T$. Thus, taking the forcing transform, we have

$$\begin{aligned}\mathbf{R}_3^T(0,1) \cdot \mathbf{f} &= \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \\ &\quad \cdot [1 \quad 1 \quad 1 \quad 1 \quad 0 \quad 0 \quad 0 \quad 1]^T \\ &= [1 \quad 3 \quad 3] = \mathbf{c}^{(1,1)}.\end{aligned}$$

The other vectors are equal to $\mathbf{c}^{(0,1)} = [4\ 2\ 2]^T$, $\mathbf{c}^{(1,0)} = [3\ 1\ 1]^T$, and $\mathbf{c}^{(0,0)} = [0\ 2\ 2]^T$. It can be seen that $c_1^{(0,1)} = 4$. This means that f is a forcing function and the forcing variable is x_1 with forcing value 0 and forced value 1. As there are no other 4s in any of the other vectors, this is the only forcing variable.

IV. Testing the A^μ and a^μ Properties

We will concentrate only on property A^μ, since a^μ is similar by duality. Let us note that if f is A^μ, then $f(00\cdots0) = 0$, simply from the definition of A^μ. Thus, before testing f, it is suggested that the first component of $\mathbf{f}$ be checked. If it is not 0, then the function cannot be A^μ. The only exception is the trivial constant function $f(x) = 0$, which is A^μ by definition.

To set the stage, let S be an n-element set of natural numbers $\{1,2,\ldots,n\}$. For each family Ω of subsets of S, let f be the corresponding indicator function; that is, $f(x_1,\ldots,x_n) = 1$ if and only if Ω contains a $T \subseteq S$ such that

$$x_i = \begin{cases} 1, & \text{if the } i\text{th element of } S \text{ is in } T \\ 0, & \text{otherwise.} \end{cases}$$

That is, the x_i encode the sets T and the function f specifies which of the sets T are contained in Ω. It is easy to see that the function f is A^μ if any μ members of Ω have a nonempty intersection. Such sets are called μ-inseparable. Indeed, if there were μ subsets of S that have an empty intersection, then it would imply that there exist μ vectors on which the indicator function is equal to 1, but which do not share any common component equal to 1, thus violating the definition of A^μ. Let the rows of matrix $\mathbf{B}_{n,\mu}$, $\mu \geq 2$, contain all indicator functions of sets that are *not* μ-inseparable, with the exception of sets containing the empty set. In other words, the first column of $\mathbf{B}_{n,\mu}$ contains zeros. For example, for $n = 3$ and $\mu = 2$

$$\mathbf{B_{3,2}} = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \end{bmatrix}.$$

Note that the indicator functions (rows) are ordered such that the leftmost bit corresponds to the all-zero vector. Every row of $\mathbf{B}_{n,\mu}$ contains exactly μ ones. Define the A^μ *transform* as

$$\mathbf{c}_{n,\mu} = \mathbf{B}_{n,\mu} \cdot \mathbf{f}. \tag{4}$$

Proposition 7: Let $\mu \geq 2$ be given. Let $f : \{0,1\}^n \to \{0,1\}$ be a Boolean function that is equal to 0 on the all-zero vector and contains exactly $m \geq 2$ ones in its truth table $\mathbf{f}$. Let $k = \min(\mu, m, n)$. Then, f is A^μ if and only if $\mathbf{c}_{n,k}$ does not contain an element equal to k. If f has only a single unity component in its truth table ($m = 1$), then f is A^μ if and only if f is equal to 0 on the all-zero vector. The all-zero function, $f(x) \equiv 0$ is A^μ.

Proof: The special cases where m equals to 0 or 1 are trivial. There are a total of three cases to consider. Whenever $k = \mu$, we use the definition of A^μ. Each row of $\mathbf{B}_{n,\mu}$ is the truth table of a Boolean function that is equal to 1 on μ vectors whose logical conjunction is the zero vector, i.e., they do not share a unity component. Thus, if f is also equal to 1 on those μ vectors, then the product of that row and $\mathbf{f}$ will be equal to μ, implying that f cannot be A^μ. Whenever $k = m$, if $\mu = m$, then the first case applies; otherwise, the definition of A^μ demands

that some of the m vectors on which f takes on the value 1 must be repeated. Thus, we only need to check whether f is A^m. If it is A^m, then f is also A^μ. If it is not A^m, then it cannot be A^μ according to Remark 4. Finally, whenever $k = n$, Proposition 5 implies that we need not check any $\mu > n$, since A^n already implies that the function is A^∞, which by Remark 4 in turn implies that it is A^μ for any $\mu \geq 2$.

Example: Let $f(x_1, x_2, x_3) = \overline{x}_1\overline{x}_2x_3 + \overline{x}_1x_2\overline{x}_3$ be a Boolean function. Then

$$\begin{aligned}\mathbf{B}_{3,2} \cdot \mathbf{f} &= \mathbf{B}_{3,2} \cdot [0 \quad 1 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0]^{\mathrm{T}} \\ &= [2 \quad 1 \quad 1 \quad 1 \quad 1 \quad 0]^T = \mathbf{c}_{3,2}.\end{aligned}$$

Since $\mathbf{c}_{3,2}$ contains an element equal to 2, function f cannot be A^2. Consequently, it cannot be A^μ for any $\mu > 2$.

The transform-based algorithm discussed above amounts to nothing more than an enumeration of all vectors having all-zero logical conjunctions and evaluating the Boolean function on these vectors. However, from an implementation standpoint, it is highly efficient, because the matrix $\mathbf{B}_{n,\mu}$ can be reused to check different functions by simply multiplying it by their truth tables. Thus, the computational effort to generate $\mathbf{B}_{n,\mu}$ is only required once. For fixed μ, this matrix is quite sparse, since every row contains exactly μ ones and zeros elsewhere.

V. Testing the A^∞ and a^∞ Properties

As above, we will only focus on A^∞, with a^∞ being similar by duality. Recall that a function f is A^∞ if all vectors on which the function equals 1 share a common unity component. Thus, if f is A^∞, then there must exist a variable x_i such that

$$f(x_1, \ldots, x_n) = x_i \cdot f(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n).$$

In other words, if we express f in its disjunctive normal form, then there must exist a variable that is contained in every term. Clearly, setting such a variable to 0 would force the function value to 0 as well. Thus, as discussed in Section II, an A^∞ function is also a forcing function. In fact, the forcing transform in (2) can be used to simultaneously check for A^∞ and a^∞. The relevant vectors to be checked are $\mathbf{c}^{(0,0)}$ and $\mathbf{c}^{(1,1)}$, respectively, as in Proposition 6. For instance, if $\mathbf{c}^{(0,0)}$ contains an element equal to 2^{n-1}, then f is A^∞. We should also note that although Proposition 7 can be used to test A^∞, we do not recommend using it, since the matrix $\mathbf{R}_n(0,1)$ is typically much smaller than matrix $\mathbf{B}_{n,m}$.

VI. Conclusion

We have described novel spectral algorithms to test membership of a Boolean function in several important classes of functions. The classes of forcing functions as well as the Post classes described in this letter are particularly important in nonlinear digital filters and discrete dynamical systems analysis. The proposed methods, therefore, represent a valuable tool. We should point out that spectral algorithms for testing other important Post classes, such as monotone functions, self-dual functions, and linear functions, are described in [22].

References

[1] P. Wendt, E. Coyle, and N. Gallagher, "Stack filters," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 34, pp. 898–911, ASSP-1986.

[2] M. Gabbouj, P.-T. Yu, and E. J. Coyle, "Convergence behavior and root signal set of stack filters," *Circ. Syst. Signal Processing*, vol. 11, no. 1, pp. 171–193, 1992.

[3] P.-T. Yu and E. J. Coyle, "Convergence behavior and N-roots of stack filters," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 38, pp. 1529–1544, Sept. 1990.

[4] J. Astola and P. Kuosmanen, *Fundamentals of Nonlinear Digital Filtering*. New York: CRC, 1997.

[5] P.-T. Yu and E. J. Coyle, "The classification and associative memory capability of stack filters," *IEEE Trans. Signal Processing*, vol. 40, pp. 2483–2497, Oct. 1992.

[6] ——, "On the existence and design of the best stack filter based associative memory," *IEEE Trans. Circuit Syst.*, vol. 39, pp. 171–184, Mar. 1992.

[7] S. A. Kauffman, *The Origins of Order: Self-Organization and Selection in Evolution*. New York: Oxford Univ. Press, 1993.

[8] ——, "Requirements for evolvability in complex systems: Orderly dynamics and frozen components," *Physica D*, vol. 42, pp. 135–152, 1990.

[9] J. F. Lynch, "On the threshold of chaos in random cellular automata," *Random Struct. Alg.*, vol. 6, no. 2/3, pp. 239–260, 1995.

[10] D. Stauffer, "On forcing functions in Kauffman's random Boolean networks," *J. Statist. Phys.*, vol. 46, no. 3/4, pp. 789–794, 1987.

[11] M. D. Stern, "Emergence of homeostasis and 'noise imprinting' in an evolution model," *Proc. Nat. Acad. Sci*, vol. 96, pp. 10 746–10 751, 1999.

[12] S. Bornholdt and K. Sneppen, "Robustness as an evolutionary principle," *Proc. R. Soc. London B*, vol. 266, pp. 2281–2286, 2000.

[13] L. Glass and C. Hill, "Ordered and disordered dynamics in random networks," *Europhys. Lett.*, vol. 41, pp. 599–604, 1998.

[14] A. Bhattacharjya and S. Liang, "Power-law distributions in some random Boolean networks," *Phys. Rev. Lett.*, vol. 77, pp. 1644–1647, 1996.

[15] S. E. Harris, B. K. Sawhill, A. Wuensche, and S. Kauffman, *Complexity*, vol. 7, no. 4, pp. 23–40, 2002.

[16] E. Post, "Introduction to a general theory of elementary propositions," *American Journal of Mathematics*, vol. 43, pp. 163–185, 1921.

[17] E. I. Nechiporuk, "On the complexity of circuits in some bases, containing nontrivial elements with zero weights," *Probl. Cybern.*, no. 8, 1962. In Russian.

[18] A. A. Muchnik and S. G. Gindikin, "On the completeness of systems of unreliable elements which realize functions of the algebra of logic," *Doclady AN SSSR*, vol. 144, no. 5, 1962. In Russian.

[19] I. Shmulevich, M. Gabbouj, and J. Astola, "Complexity of the consistency problem for certain Post classes," *IEEE Trans. Syst., Man, Cybern.*, pt. Part B, vol. 31, pp. 251–253, Apr. 2001.

[20] I. Shmulevich, H. Lähdesmäki,, E. R. Dougherty, J. Astola, and W. Zhang, "The role of certain Post classes in Boolean network models of genetic networks," *Proc. Nat. Acad. Sci.*, vol. 100, no. 19, pp. 10 734–10 739, 2003.

[21] V. S. Levchenkov, "Boolean equations and Post classes," *Doklady Akademii Nauk*, vol. 373, no. 3, pp. 316–319, 2000. In Russian.

[22] S. Agaian, J. Astola, and K. Egiazarian, *Binary Polynomial Transforms and Nonlinear Digital Filters*. New York: Marcel Dekker, 1995.

[23] S. V. Yablonsky, G. P. Gavrilov, and V. B. Kudryavtsev, *Functions of the Algebra of Logic and Post Classes*. Moscow: Nauka, 1966. In Russian.