KYBERNETIKA — VOLUME 27 (1991), NUMBER 6

BRANCHING PROGRAMS PROVIDE LOWER BOUNDS ON THE AREA OF VLSI CIRCUITS

JURAJ HROMKOVIČ

Branching programs that were studied as nonuniform computing model providing lower bounds on the space of deterministic sequential computations are considered. We show that they can be used for providing lower bound on the parallel VLSI computations.

1. INTRODUCTION AND DEFINITIONS

The branching programs represent a tool for proving lower bounds on the space complexity of sequential computations (see [10, 12, 14, 19] for details), and they were studied in several papers for this reason [1–5, 9–14, 17–21].

In this paper we find a strong connection between the branching programs and VLSI circuits that represent a model of parallel computations. To be more precise we prove the following results:

(i) The capacity of the levelled branching programs provides lower bounds on the area complexity of the multilective VLSI circuits.

(ii) The capacity of the levelled one-time-only branching programs provides lower bounds on the area complexity of the basic model of VLSI circuits.

These results enables us to regard the several nontrivial lower bounds on the capacity of branching programs (especially the exponential lower bounds on the size of the one-time-only branching programs) proved in [1, 9, 11, 13, 17, 21] for lower bounds on the area complexity of VLSI computations. Thus, we obtain several new, nontrivial lower bounds on VLSI computation by relating the complexity of branching programs to VLSI computations.

Usually, an informal or partially formal description of VLSI circuit models is used in the current literature [15, 16]. We use the definition of the basic VLSI model in the form of a mathematical structure from [6, 7]. Now, we define the model of multilective VLSI circuits in this formal way because it helps us to find a clear idea connecting the complexity of multilective VLSI circuits with the capacity of branching programs.

Definition 1.1. Let $X = \{x_1, \ldots, x_n\}$, $Y = \{y_1, \ldots, y_m\}$ be sets of Boolean variables. A *problem instance* P from the input variables X to the output variables Y is a nonempty set of Boolean functions $f_1, \ldots, f_m$ such that $f_i\colon \{0, 1\}^n \to \{0, 1\}$ and $y_i = f_i(x_1, \ldots, x_n)$ for $i = 1, \ldots, m$. The positive integer n is called the *size* of P.

Definition 1.2. A *problem* is an infinite sequence of problem instances, where each two instances in the sequence have a different size parameter n.

The recognition of a language $L \subseteq \{0, 1\}^+$ can be considered as an example of a problem. To see this fact we associate with each language $L \subseteq \{0, 1\}^+$ the infinite sequence of Boolean functions $\{h_i^L\}_{i=1}^{\infty}$, where $h_i^L\colon \{0, 1\}^i \to \{0, 1\}$ and $h_i^L(a_1, \ldots, a_i) = 1$ iff $a_1 a_2, \ldots, a_i \in L \cap \{0, 1\}^i$. So, the ith problem instance is the Boolean function h_i^L.

Definition 1.3. A *4-graph* is such a directed graph $G = (V, E)$ that, for each $v \in V$, the sum of the number of output edges of v (the *outdegree* of v) and of the number of input edges of v (the *indegree* of v) is bounded by 4.

Definition 1.4. A *grid-graph* M_G is a 4-graph G embedded in the lattice in such a way that each square of the lattice has one of the following contents:

(a) a vertex of the graph
(b) a straightforward part of a line going in the horizontal or in the vertical direction (this line is part of the layout of an edge of the graph)
(c) a broken line coming in the lattice square in one of two vertical (horizontal) directions and coming out in one of two horizontal (vertical) directions,
(d) a crossing of two lines, one going in the horizontal direction, the other in the vertical direction (this depicts the place of two crossing edges),
(e) the empty content

Definition 1.5. The area complexity of a grid-graph is the area of a minimal rectangle involving all non-empty squares of the lattice. The *area complexity of a 4-graph* is the minimum of area complexities of all grid-graphs that are embeddings of G in the lattice.

Definition 1.6. A *multilective VLSI circuit* is a 6-tuple $R = \langle M_G, P, p, X, Y, r\rangle$, where:

(1) M_G is a grid-graph which is an embedding if the graph $G = (V, E)$,
(2) P is a finite, nonempty set (called a *processor set*) of the functions from $\{0, 1\}^i$ to $\{0, 1\}^j$, where $i + j \leqq 4$, $i, j \in \{0, 1, \ldots, 4\}$,
(3) $X = \{x_1, \ldots, x_n\}$ is the set of *input variables*
(4) $Y = \{y_1, \ldots, y_m\}$ is the set of *output variables*
(5) r is a function (called an *I/O function*) from $X \cup Y$ to the set of nonempty,

finite subsets of $V \times N$ such that for all $x, y \in X \cup Y$, $|r(y)| \geqq 1$; $x \neq y$ implies $r(x) \cap r(y) = \emptyset$; and if $(v, t) \in r(x)$ for an $x \in X(Y)$ then v is called an *input* (*output*) vertex and v has the indegree (outdegree) zero and the outdegree (indegree) at most three.

(6) p is a function from V to P such that:
  (i) for each $v \in V$ which is not an input or output vertex, if v has the indegree i and the outdegree j, then $p(v) = f_v$, where f_v: $\{0, 1\}^i \to \{0, 1\}^j$;
  (ii) for each input vertex $v \in V$, $p(v) = f_v$, where f_v: $\{0, 1\} \to \{0, 1\}^j$ and j is the outdegree of v;
  (iii) for each output vertex $v \in V$, $p(v) = f_v$, where f_v: $\{0, 1\}^i \to \{0, 1\}$ and i is the indegree of v.

The vertices with functions assigned from P are called *processors*.

Definition 1.7. A *VLSI circuit* is a multilective VLSI circuit $R = \langle M_G, P, p, X, Y, r\rangle$ with the property $|r(x)| = 1$ for any $x \in X$ (each input variable is available exactly ones).

Now, let us define the computation of a multilective VLSI ciscuit $R = \langle M_G, P, p, X, Y, r\rangle$ with $G = (V, E)$. Let $v_1, \ldots, v_m$ be a fixed order of all vertices in V. Let, for $j_i \in \{1, 2, 3, 4\}$, $e_{i1}, \ldots, e_{ij_i}$ be the sequence of all input edges of v_i from E for all v_i's that are not input vertices, and let e_{i1} be an "input edge" (not in E) if v_i is an input processor. For each time unit t one can associate a Boolean value a^t_{ik} to each e_{ik}, $i \in \{1, \ldots, m\}$, $k \in \{1, \ldots, j_i\}$. Thus, we can define a *state* s_t of the VLSI circuit R in the time unit t as the sequence

$$a^t_{11}, \ldots, a^t_{1j_1}, a^t_{22}, \ldots, a^t_{2j_2}, \ldots, a^t_{mm}, \ldots, a^t_{mj_m}$$

of values on the edges

$$e_{11}, \ldots, e_{1j_1}, e_{22}, \ldots, e_{2j_2}, \ldots, e_{mm}, \ldots, e_{mj_m}\,.$$

The subsequence of s_t consiststing only from the values on the edges from E is called the *internal state* of R. In the time unit $t = 0$ all input edges from E have the value 0. For all $x \in X$, if $(v_i, 0) \in r(x)$, the e_{i1} has the value of the input variable x. If for all x's $(v_i, 0) \notin r(x)$ then e_{i1} has the value 0 $(a^0_{i1} = 0)$.

Clearly, knowing the values of all input edges in the time unit t we obtain values on edges from E in the time unit $t + 1$ as output values of all processors (vertices) with given inputs. The input edge of the input processor (vertex) v has in the time unit $t + 1$ either the value 0 (if no input variable is inputing in v in the time unit $t + 1$) or the value of an input variable comming in v in the time unit t.

The *computation* of R on an input α is a sequence of states $s_0, s_1, \ldots, s_t, s_{t+1}, \ldots$ such that for each t, s_t is the state of R working on α in the time unit t. Taking the output values from output processors in the time units determined by r one can associate exactly one problem instance to R.

Definition 1.8. The *time complexity* $I(R)$ of a multilective VLSI circuit $R =$

$= \langle M_G, P, p, X, Y, r\rangle$ is $\max\{t \mid (v, t) \in r(y)$ for a $y \in Y\}$. The *area complexity* $A(R)$ of R is the area complexity of M_G. The *time (area) complexity* $T(P)$ $(A(P))$ of a problem instance P is the time (area) complexity of an R solving P with minimal time (area) complexity.

2. RESULTS

We shall show that the branching programs provide lower bounds on the area complexity of VLSI computations. To do it we define m-branching programs (m-BP) as a generalization of branching programs (BP). We use m-BPs to simulate the work of multilective VLSI circuits. Then simulating m-BP by the original BP we obtain the connection between the complexity of VLSI computations and the complexity of BPs.

Definition 2.1. Let $X = \{x_1, \ldots, x_n\}$ be a set of Boolean variables. The *m-branching program, m-BP,* for a X and a positive integer $m \leqq n$, is an acyclic directed graph with the following properties:
1) there is exactly one source,
2) every node has the outdegree 2^k for a $k \in \{0, 1, \ldots, m\}$,
3) every node v is labelled by a set $\{x_{i_1}, \ldots, x_{i_r}\} \subseteq X$ for an $r \leqq m$, each edge e of 2^r edges leading from v is labelled by $l(e) = (k_1, \ldots, k_r)$, $k_i \in \{0, 1\}$ for $i \in$, $\in \{1, \ldots, r\}$, and $l(e) \neq l(e')$ for different e and e' leading from v
4) every sink is labelled by 0 or 1.

Definition 2.2. Let U be an m-BP for $X = \{x_1, \ldots, x_n\}$. Given an input $a =$ $= (a_1, \ldots, a_n) \in \{0, 1\}^n$, U computes a function value $f_u(a)$ in the following way. The computation starts at the source. If the computation has reached a node v labelled by $\{x_{i_1}, \ldots, x_{i_r}\}$ then computation proceeds via the edge labelled by $(a_{i_1}, \ldots, a_{i_r})$. Once the computation reaches a sink, the computation ends and $f_u(a)$ is defined to be the label of that sink.

The *length* of an m-BP U is the length of the longest path in U, and *the complexity of* U, BP(U) is the number of nodes in U. The *capacity* of U is $\log_2$ BP (U).

It is easy to see that 1-BPs are the original BPs as defined in [10, 12], and that each Boolean function can be computed by an m-BP of length n/m and complexity $O(2^n)$.

Now, let us introduce some special types of branching programs.

Definition 2.3. Let m be a positive integer. Let U be an m-BP. If, for each path $v_1, \ldots, v_k$ of vertices in U and each $i \neq j$, $l(v_i) \cap l(v_j) \neq \emptyset$ then we say the U is a *one-time-only* m-BP, shortly m-BP$_1$.

A m-BP (m-BP$_1$) U is *levelled* if the vertices of U are partitioned into some blocks $A_1, \ldots, A_k$ so that every edge goes from A_i to A_{i+1} for some $i \leqq k - 1$. If, for each $z \in \{1, \ldots, k\}$, the vertices in A_z are labelled by the same subset of input variables then we say that U is a well-levelled m-BP (m-BP$_1$). $W(U) = \max\{|A_i| \mid i = 1, \ldots, k\}$ is the width of levelled m-BP U.

Definition 2.4. Let f be a Boolean function from $\{0, 1\}^n$ to $\{0, 1\}$. Let m be a positive integer. *m-BP* (f) [m-BP$_1$ (f)] denotes the minimum of sizes of m-branching programs [one-time-only m-branching programs] which compute f. 1-BP (f) [1-BP$_1$ (f)] will be denoted by $BP(f)$ [$BP_1(f)$] in what follows too.

Now, let us establish the relation between m-BPs and VLSI computations.

Lemma 2.5. Let, for a Boolean function f: $\{0, 1\}^n \to \{0, 1\}$, $P = \{f\}$ be a problem instance with the set of input variables X. Let R be a multilective VLSI circuit [VLSI circuit] solving P. There is an $A(R)$-BP [$A(R)$-BP$_1$] U such that

(i) U computes f,
(ii) U is well levelled,
(iii) the depth of U is at most $T(R)$,
(iv) the size of U is at most $2^{2A(R)}\, T(R)$,
(v) the width of U is at most $2^{2A(R)}$.

Proof. Let $R = \langle M_G, P, p, X, Y, r\rangle$, $G = (V, E)$. Since R has the area $A(R)$, $|E| \leqq 2A(R)$. So, there are at most $2^{2A(R)}$ internal states of R. Let $t_1, \ldots, t_k$ be time in which R "reads" some input values.

We construct the m-BP U simulating the work of R in the following way. U will have $k + 1$ levels, each level containing at most $2^{2A(R)}$ nodes. Every node in the ith level (for $i \in \{1, \ldots, k\}$) will correspond to an internal state of R that is reachable in the time unit t_i.

The first level of U contains only one node that is labelled by the subset of input variables that contains exactly the variables read by R in the time unit t_1. Clearly, the cardinality of this subset is at most $A(R)$ because the number of input vertices of R is bounded by $A(R)$. For $i \in \{2, \ldots, k\}$, the ith level contains exactly one node for each internal state of R reachable in the time unit t_i. All nodes in the ith level are labelled with the same set of at most $A(R)$ input variables inputting in the circuit R in the time unit t_i. The last level contains only two nodes. One labelled by 1 corresponds to states of R in which R gives output 1, the other labelled by 0 corresponds to states of R in which R gives output 0 in the time unit $T(R)$. Let v labelled by $\{x_{i_1}, \ldots, x_{i_k}\}$ be a node of the jth level for some $j \in \{1, \ldots, k\}$, and let v correspond to an internal state s_v of R. Then, for each $(a_1, \ldots, a_k) \in \{0, 1\}^k$, U contains an edge leading from v to a node u in the $(j + 1)$th level that corresponds to an internal state s_u that is the internal state of R reached in the time unit t_{j+1} from the state s_v by inputing the values $a_1, \ldots, a_k$ of input variables $x_{i_1}, \ldots, x_{i_k}$ in the time unit t_j.

Clearly, U fulfils (i), (ii), (iii), (iv), and (v), and if R is a VLSI circuit then U is a one-time-only $A(R)$-BP. □

Now, we simulate m-BPs by BPs.

Lemma 2.6. Let, for a Boolean function $f\colon \{0, 1\}^n \to \{0, 1\}$, U be an well-levelled m-BP computing f. Then there is a well-levelled BP V computing f such that

$$c(V) \leqq 2^m\, c(U)\,.$$

Proof. Let U be an m-BP. Starting from the source and going from one level to the next one we replace each node and the edges leading from it by a tree of the depth at most $m - 1$, and some futher edges leading from the leaves of the tree. If v is labelled by $\{x_{i_1}, \dots, x_{i_k}\}$ the tree T_v is the full binary tree of the depth $k - 1$. The root of T_v is labelled by $\{x_{i_1}\}$, and each node of an rth level of T_v is labelled by $\{x_{i_r}\}$. Two edges labelled by 0 and 1 lead from any node of T_v. If the path from the root of T_v to a leaf z of T_v is labelled (on edges) by $a_1, a_2, \dots, a_{k-1}$ then we give an edge labelled by 0 from z to a node u connected by the edge $(a_1, a_2, \dots, a_{k-1}, 0)$ with v in U, and an edge labelled by 1 from z to a node w connected by the edge $(a_1, a_2, \dots, a_{k-1}, 1)$ with v in U.

Clearly, the constructed BP is well-levelled and it computes the same Boolean function as U. □

Combining Lemmas 2.5 and 2.6 we obtain the main results of this paper.

Theorem 2.7. Let R be a multilective VLSI circuit computing a Boolean function f. Then:

$$3A(R) + \log_2 T(R) \geqq \log_2 (\mathrm{BP}(f))$$

Proof. Let us prove this assertion by contradiction. Let $3\,A(R) + \log_2 T(R) < \log_2 (\mathrm{BP}(f))$ for a multilective VLSI circuit R computing f. Using Lemma 2.5 we can construct an $A(R)$-BP U computing f and fulfilling $\mathrm{BP}(U) \leqq 2^{2A(R)}\, T(R)$. Applying Lemma 2.6 we can construct a BP (V) computing f such that

$$\mathrm{BP}(V) \leqq 2^{A(R)}\, \mathrm{BP}(U) \leqq 2^{3A(R)}\, T(R)\,.$$

So, we obtain $\mathrm{BP}(V) < \mathrm{BP}(f)$ which is a contradiction. □

Theorem 2.8. Let R be a VLSI circuit computing a Boolean function f. Then

$$3\,A(R) \geqq \log_2 (\mathrm{BP}_1(f)) - \log_2 n\,.$$

Proof. The proof is similar to that of the previous theorem. The only difference is that every BP_1 has at most n levels (i.e. there are at most n time units in which R reads an input). □

Now, let us consider the usefulness of the introduced lower bound technique. It is very hard to prove a nonlinear lower bound on a given Boolean function. The highest known lower bound on BPs computing a Boolean function sequence $\mathscr{P} = \{f_i\}_{i=1}^{\infty}$ is $\Omega(n^2/(\log_2 n)^2)$ due to Nečiporuk [11]. So, using BPs were able to give only logarithmic lower bounds on the area and $\log_2 T(R)$ of multilective VLSI circuit. But, note that we do not know any other lower bound technique that is able to bring more for multilective VLSI circuits.

On the other hand there are many results providing exponential lower bounds on BP_1s computing specific Boolean functions (see for example Jukna [9], Wegener [17], Žák [21], Babai et. al. [1]). The highest 2^{cn} (for some constant c) lower bounds on one-time-only branching programs computing $\{h_n^{\Delta}\}_{n=1}^{\infty}$ for

$\Delta = \{w \in \{0, 1\}^+ \mid |w| = \binom{k}{2}$ for a $k \in \mathbb{N}$, w codes and undirected graph with k vertices having an odd number of triangles$\}$

was obtained in Babai et al. [1]. All lower bounds on BP_1s in [9, 17, 21] bring new lower bounds of the form $\Omega(n^a)$, $0 < a \leqq 1$, for VLSI circuits. An interesting fact follows from $\Omega(n)$ lower bound on the area of VLSI circuits recognizing the language Δ. One can simply construct a sequence of multilective VLSI circuits recognizing Δ in constant space and $0(n^{3/2})$ time (one after other – each triple of vertices is checked) and we see that there is no efficient area simulation of multilective VLSI circuits by VLSI circuits in some sense.

Open problem 2.9. Is there an efficient area simulation of multilective VLSI circuits with bounded multilectivity by VLSI circuits? We note that we have assumed the use of each input variable $0(n^{1/2})$ times for the recognition of Δ.

An argument for a negative answer is the following result of Žák [21]. He proved an exponential lower bound on BP_1s recognizing a specific language that can be recognized by two-time-only BP (each input variable occurs at most two times at any path of BP) with polynomial size.

3. CONCLUSION

We have found a simple way how to show that a well-known tool (BPs) for proving lower bounds on sequential computations can be used for proving lower bounds on VLSI circuits – a model of parallel computations. In fact we have introduced the well-levelled BPs as a special type of BPs that provides lower bounds on VLSI computations. We note that our lower bound technique is independent on the area-layout of VLSI circuits, i.e., the complexity of BPs is related directly to the number of processors working in parallel (not to the topology of circuits). Thus, some lower bounds for one-time-only branching programs can be interpreted as lower bounds for VLSI circuits. Unfortunaly, there are no lower bounds on general BPs which can bring something for proving lower bounds on the area complexity of multilective VLSI circuits. On the other hand there are already known, successful techniques [6, 7, 8, 15, 16] for sublinear and linear lower bounds on the area complexity of VLSI circuits. So, the method presented in this paper brings a new point of view on proving lower bounds for VLSI circuits, but it provides no break in proving lower bounds for VLSI computations. These facts imply that the main problem formulated in this paper has to be the following one.

Open problem 3.1. Improve the BP-method for proving lower bounds on VLSI computations in such a way that it brings a break in proving lower bounds for some VLSI computation model.

To solve the above stated problem we have at least two ways. One very hard way is to prove exponential lower bounds on the general BPs. Another way, which may be easier than the above mentioned one, is to find a new connection between a special type of branching programs and a special type of VLSI computation model, for which no $\Omega(n^a)$ lower bounds are known, and to prove an exponential lower bound for the given, special type of branching programs.

For example, one can conjecture that there is a connection between branching programs and where-and-when-indeterminite VLSI circuits. To find such a connection is of importance because there is no lower bound technique providing lower bounds on area of where- and when-indeterminite VLSI circuits. We have tried to find such a connection without any success. We conjecture that the indeterminism in reading input in BPs is of another nature (state dependent) as the where- and when-indeterminism in VLSI computations. More about where- and when-indeterminism and about lower bounds techniques for it can be found in Hromkovič [8].

Finally, we call attention to the fact that our contribution brings something new in the understanding of hardness of some problems in complexity theory. We know that to prove an exponential lower bound on BPs and to prove $\Omega(n^a)$, $a > 0$, lower bound on the area of multilective VLSI circuits are very hard problems. Now, we know that these problems have a similar nature (we see better why they are so hard), and that the first problem is at least so hard as the second one. The question whether lower bounds on multilective VLSI circuits can bring some lower bounds on BPs remains open.

ACKNOWLEDGEMENT

I would like to thank Ondrej Sýkora for a few interesting discussions.

(Received March 28, 1989.)

REFERENCES

[1] L. Babai, P. Hajnal, E. Szemerédi and G. Turán: A lower bound for read-once only branching programs. J. Computer System Sci. *35* (1987), 153—162.

[2] D. A. Barrington: Bounded width polynomial-size branching programs recognize exactly those languages in NC^1. In: Proc. 18th ACM STOC, ACM 1986, pp. 1—5.

[3] A. Borodin, D. Dolev, F. E. Fich and W. Paul: Bounds for width two branching programs. In: Proc. 15th ACM STOC, ACM 1983, pp. 87—93.

[4] A. K. Chandra, M. L. Furst and R. J. Lipton: Multiparty protocols. In: Proc. 15th ACM STOC, ACM 1983, pp. 94—99.

[5] M. Ftáčnik and J. Hromkovič: Nonlinear lower bounds for real-time branching programs. Comput. Artificial Intelligence *4* (1985), 3, 353—359.

[6] J. Hromkovič: Some complexity aspects of VLSI computations. Part 1. A framework for the study of information transfer in VLSI circuits. Comput. Artificial Intelligence *7* (1988), 3, 229—252.

[7] J. Hromkovič: Some complexity aspects of VLSI computations. Part 2. Topology of circuits and information transfer. Comput. Artificial Intelligence *7* (1988), 4, 289—302,

[8] J. Hromkovič: Some complexity aspects of VLSI computations. Part 5. Nondeterministic and probabilistic VLSI circuits. Comput. Artificial Intelligence *8* (1989), 2, 169—188.

[9] S. P. Jukna: Lower bounds on the complexity of local circuits. In: Mathematical Foundation of Computer Science — Proc. 12th Symposium, Bratislava, Czechoslovakia, August 25—29, 1986 (J. Gruska, B. Rovan, J. Wiedermann, eds.). (Lecture Notes in Computer Science 233.) Springer-Verlag, Berlin—Heidelberg—New York—Tokyo 1986, pp. 440—448.

[10] W. Masek: A Fast Algorithm for String Editing Problem and Decision Graph Complexity. M. Sc. Thesis, MIT, May 1976.

[11] E. I. Nečiporuk: On a Boolean function. Dokl. Akad. Nauk SSSR *169* (1966), 4, 765—766. English translation: Soviet Math. Dokl. *7* (1966), 999—1000.

[12] P. Pudlák and S. Žák: Space complexity of computations. Unpublished manuscript, 1982.

[13] P. Pudlák: A lower bound on complexity of branching programs. In: Mathematical Foudations of Computer Science — Proc. 11th Symposium, Prague, Czechoslovakia, September 3—7, 1984 (M. P. Chytil, V. Koubek, eds.). (Lecture Notes in Computer Science 176.) Springer-Verlag, Berlin—Heidelberg—New York—Tokyo 1984, pp. 480—489.

[14] P. Pudlák: The hierarchy of Boolean circuits. Comput. Artificial Intelligence *6* (1987), 5, 449—468.

[15] C. D. Thompson: A Complexity Theory for VLSI. Doctoral Dissertation, CMU-CS-88-140, Computer Science Dept., Carnegie-Mellon University, Pittsburg, August 1980, 131 p.

[16] J. D. Ullman: Computational Aspects of VLSI. Computer Science Press, New York 1984.

[17] I. Wegener: On the Complexity of Branching Programs and Decision Trees for Qlique Functions. Universität Frankfurt, Fachbereich Informatik, Int. Rept. 5/84, 1984.

[18] I. Wegener: Time-space trade-offs for branching programs. J. Comput. System. Sci. *32* (1986), 1, 91—96.

[19] I. Wegener: Optimal decision trees and one-time only branching programs for symmetric Boolean functions. Inform. Control *62* (1984), 129—143.

[20] I. Wegener: The Complexity of Boolean Functions. Wiley-Teubner Series in Computer Science, B. G. Teubner, Stuttgart, John Wiley and Sons, New York 1987.

[21] S. Žák: An exponential lower bound for one-time-only branching programs. In: Mathematical Foundations of Computer Science — Proc. 11th Symposium, Prague, Czechoslovakia, September 3—7, 1984 (M. P. Chytil, V. Koubek, eds.). (Lecture Notes in Computer Science 176.) Springer-Verlag, Berlin—Heidelberg—New York—Tokyo 1984, pp. 562—566.

Doc. RNDr. Juraj Hromkovič, DrSc., katedra informatiky, Matematicko-fyzikálna fakulta Univerzity Komenského (Department of Computer Science, Faculty of Mathematics and Physics, Comenius University), 842 15 Bratislava. Czechoslovakia.