národní
úložiště
šedé
literatury

A Lower Bound on Branching Programs Reading Some Bits Twice

Savický, Petr
1995

Dostupný z http://www.nusl.cz/ntk/nusl-33631

Dílo je chráněno podle autorského zákona č. 121/2000 Sb.

Tento dokument byl stažen z Národního úložiště šedé literatury (NUŠL).

Datum stažení: 28.09.2024

Další dokumenty můžete najít prostřednictvím vyhledávacího rozhraní nusl.cz .

INSTITUTE OF COMPUTER SCIENCE

ACADEMY OF SCIENCES OF THE CZECH REPUBLIC

A lower bound on branching programs reading some bits twice

Petr Savický, Stanislav Žák

Technical report No. 647

Institute of Computer Science, Academy of Sciences of the Czech Republic
Pod vodárenskou věží 2, 182 07 Prague 8, Czech Republic
phone: (+422) 66414244 fax: (+422) 8585789
e-mail: {savicky,stan}@uivt.cas.cz

INSTITUTE OF COMPUTER SCIENCE

ACADEMY OF SCIENCES OF THE CZECH REPUBLIC

A lower bound on branching programs reading some bits twice

Petr Savický, Stanislav Žák [1]

Technical report No. 647

Abstract

By $(1,+k(n))$-branching programs (b. p.) we mean those b. p. which during each of their computations are allowed to test at most $k(n)$ input variables repeatedly. For a Boolean function computable within polynomial time a trade-off is presented between the number of repeatedly tested variables and the size of any b. p. P computing the function. Namely, if at most n^{α} repeated tests are allowed, where α is a constant satisfying $0 < \alpha < 1$, then the size of P is at least $\exp(\Omega(\sqrt{n^{1-\alpha}/\log_2 n}))$.

The presented result is a step towards a superpolynomial lower bound for 2-b. p. which is an open problem since 1984 when the first superpolynomial lower bounds for 1-b. p. were proven [8], [10]. The present result is an improvement on [12].

Keywords

branching programs, complexity, Boolean functions

[1] The research of both authors was supported by GA CR, Grant No. 201/95/0976

1 Introduction

A branching program (b. p.) is a computation model for representing the Boolean functions. The input of a branching program is a vector consisting of the values of n Boolean variables. The branching program itself is a directed acyclic graph with one source. The out-degree of each node is at most 2. Every branching node, i.e. a node of out-degree 2, is labeled by the index of an input variable and one of its out-going edges is labeled by 0, the other one by 1. The sinks (out-degree 0) are labeled by 0 and 1. A branching program determines a Boolean function as follows. The computation starts at the source. If a node of out-degree 1 is reached, the computation follows the unique edge leaving the node. In each branching node the variable assigned to the node is tested and the out-going edge labeled by the actual value of the variable is chosen. Finaly, a sink is reached. Its label determines the value of the function for the given input. By the size of a branching program we mean the number of its nodes.

The branching programs were introduced because of their relation to $P=?LOG$ problem. Namely, the polynomial size b. p. represent a nonuniform variant of LOG. Hence, a superpolynomial lower bound on b. p. for a Boolean function computable within polynomial time would imply $P \neq LOG$.

In order to investigate the computing power of branching programs, restricted models were suggested. Besides the hope that these restricted models help to solve the general problem, some kinds of the restricted branching programs are important also as a data structure for representing the Boolean functions in some CAD applications, e.g. design or verification of Boolean circuits, see e.g. [9].

One possible kind of restriction of the b. p. is to bound the number of repeated tests of each variable. In a k-b. p., each computation can test each variable at most k times. Even more restrictive assumption is the following. In a syntactic k-b. p., on each path from the source to a sink, each variable is tested at most k times. This is more restrictive, since in a b. p., there may be pathes that are not followed by any computation.

In 1984 the first superpolynomial lower bounds for 1-b. p. were proven [8], [10]. The first steps towards the case of 2-b. p. were made with real-time b. p., which perform at most n steps during each computation on any input of length n. The results were a quadratic lower bound [3], a subexponential lower bound [11] and an exponential lower bound [5]. For (even nondeterministic) syntactic k-b. p., exponential lower bounds have been proven [2], [4]. However the problem for 2-b. p. remains open.

There are also other models that are more powerfull than 1-b. p., but probably not more powerfull than 2-b. p. Namely, $(1,+k(n))$-b. p. is a b. p., where for each computation, the number of variables tested more than ones is at most $k(n)$. There is no restriction on the number of tests of these at most $k(n)$ variables. Again, syntactic $(1,+k(n))$-b. p. are b. p., where the above restriction is applied to any path from the source to a sink, not only to the computations.

For syntactic $(1,+k(n))$-b. p., where $k(n)$ is bounded by $cn^{1/3}/\log_2^{2/3} n$ for an appropriate $c>0$, exponential lower bound and tight hierarchies (in $k(n)$) are presented in [6] and [7]. In the present paper, we prove an exponential lower bound for $(1,+n^{\alpha})$-b. p., where α is any fixed real number in $(0,1)$, computing a function f_n. The function

f_n is computable in polynomial time and, in fact, it is in ACC. A preliminary result implying an exponential lower bound for another function and for $(1, +n^{\alpha})$-b. p., where α is any constant satisfying $\alpha < 1/2$, is presented in [12].

2 The lower bound

First, we shall discuss some notation. As an input to a b. p. we shall consider also partial inputs. This means that the input variables may have values 0, 1 and $*$. In the last case, we say that the corresponding input variable is not specified by the (partial) input. For an input u let *comp*(u) denotes the sequence of nodes of the b. p. visited during the computation on u. For a partial input u, we say that the computation for u leads to a node w, if w is the first branching node on the computation for u, which tests a variable not specified by u.

If u is a vector, then its i-th coordinate is denoted by $u(i)$. By subscripts, we distinguish different vectors. If $u \in \{0,1\}^n$, then $|u|$ denotes the number of ones in u.

Assume, P is a b. p. computing a function f and let f' be a subfunction of f obtained by setting some variables of f to constants. By changing every branching node of P labeled by a variable involved in the setting to a node with out-degree one and with an appropriate successor, we may obtain a b. p. P' computing f'. The b. p. P' will be called a restriction of P.

Definition 2.1 *Let $I = \{1, ..., n\}$ be the set of indices of the input variables. Let $u_1, u_2, \ldots, u_s$ be (partial) inputs and let for all $i = 1, 2, \ldots, s$, $A_i \subseteq I$ be the set of indices of the variables specified in u_i. Let A_i be pairwise disjoint. Then, let $[u_1, u_2, \ldots, u_s]$ be the (partial) input specifying the variables with the indices from $\bigcup_{i=1}^{s} A_i$ such that if $j \in A_i$, then $[u_1, u_2, \ldots, u_s](j) = u_i(j)$.*

Definition 2.2 *Let a, b be (partial) inputs with the same set of specified variables and $D = \{i : a(i) \neq b(i)\}$. The pair a, b is called a forgetting pair, if there is a node w in the branching program such that w belongs to both comp(a) and comp(b) and both computations read all the variables with indices in D at least ones before reaching w.*

Lemma 2.3 *Let c be the size of a branching program P and let every computation of P reads at least d different variables. Let $s \geq 1$ be a natural number such that $(2\log_2 c + 1)s \leq d$. Then there exist pairwise disjoint sets $A_i \subset \{1, 2, ..., n\}$ for $i = 1, ..., s$ and partial inputs $a_i : A_i \to \{0,1\}$ and $b_i : A_i \to \{0,1\}$, $a_i \neq b_i$ such that for all $i = 1, 2, ..., s$ we have*

(i) $|A_i| \leq 2\log_2 c + 1$,

(ii) the inputs $[a_1, ..., a_s]$ and $[a_1, ..., a_{i-1}, b_i, a_{i+1}, ..., a_s]$ form a forgetting pair.

Proof: Let $r = \lfloor \log_2 c \rfloor + 1 > \log_2 c$. By our assumption, every computation of P reads at least r variables, since $r \leq 2\log_2 c + 1 \leq d$. We shall construct a sequence $U_0, U_1, \ldots, U_r$ of sets of partial inputs in such a way that for every $i = 0, 1, \ldots, r$, the set U_i consists of 2^i inputs with exactly i specified bits. In particular, U_0 contains only the totaly undefined input. Given U_i, the set U_{i+1} is constructed as follows. For each

$u \in U_i$, follow the computation for u until the first bit not specified by u is required. Let j be its index. Then, include into U_{i+1} the inputs u_0 and u_1 extending u by setting the j-th bit to 0 and 1 respectively. Now, since $2^r > c$, there are at least two distinct inputs in U_r, say c_1 and c_2, such that the computation for both these inputs lead to the same node. Note that, by construction of U_r, there is a bit specified by both c_1 and c_2, but with different values in c_1 and c_2. Moreover, for both $i = 1, 2$, the computation for c_i reads all the bits specified by c_i.

Let C_1 be the set of bits tested by $comp(c_1)$ and C_2 be the set of bits tested by $comp(c_2)$. Let $A_1 = C_1 \cup C_2$. Since $C_1 \cap C_2 \neq \emptyset$, $|A_1| \leq 2r - 1 \leq 2\log_2 c + 1$. Let a partial input a_1 extends c_1 in such a way that the bits with indices in A_1 not determined in c_1 are equal to the values of these bits in c_2. Similarly, b_1 extends c_2 so that the bits undefined in c_2 have the same value as in c_1. Hence, a_1 and b_1 may differ on the bits with indices from $C_1 \cap C_2$ and the bits with indices from the symmetric difference of C_1 and C_2 are specified in both a_1 and b_1 and equal. One can easily see that $comp(a_1)$ follows $comp(c_1)$ and $comp(b_1)$ follows $comp(c_2)$ until $comp(a_1)$ and $comp(b_1)$ join in some node w. All the bits, where a_1 and b_1 differ, are read on both computations before reaching w. Hence, a_1 and b_1 form a forgetting pair.

If A_j, a_j and b_j for all $j = 1, \ldots, i$, where $i < s$, are already constructed, we continue in the following way: consider only those inputs which equal a_j on A_j for all $j = 1, \ldots, i$. These inputs define a restriction P_i of P. Since $|\bigcup_{j=1}^{i} A_j| \leq (2\log_2 c + 1)(s - 1) \leq d - 2\log_2 c - 1$ and $r \leq 2\log_2 c + 1$, each computation of P_i tests at least r variables. If not, then during a computation of P less than d bits are tested. This would be a contradiction to the assumptions of the lemma. Now, the construction of $U_0, U_1, \ldots, U_r$ applied in the first part of the proof to P is applied to P_i. Using this, a_{i+1}, b_{i+1} and A_{i+1} are defined in the same way as a_1, b_1 and A_1 above. Note that, by construction, A_{i+1} containes only variables not in $\bigcup_{j=1}^{i} A_j$.

It follows from the construction that (i) is satisfied. Moreover, the construction of a_i and b_i implies that the computations for the partial inputs $[a_1, ..., a_i]$ and $[a_1, ..., a_{i-1}, b_i]$ lead to the same node. Hence, the inputs $[a_1, \ldots, a_s]$ and $[a_1, \ldots, a_{i-1}, b_i, a_{i+1}, \ldots, a_s]$ form a forgetting pair. □

Definition 2.4 *For every nonzero natural numbers m, t let $g_{m,t}$ be the Boolean function of $(m + 1)t$ variables defined as follows. The input of $g_{m,t}$ is treated as an m by t 0-1 matrix A and a 0-1 vector u of length t. Then, let $g_{m,t}(A, u) = 1$ if and only if $Au = 0$.*

In order to prove that $g_{m,t}$ is hard for $(1, +s)$-b. p. for some s, we will find an appropriate m by t matrix A and then we prove a lower bound on the size of any $(1, +s)$-b. p. computing $g_{m,t}(A, u)$ as a function of u only. The properties of the matrix A needed for this are stated in the following definition.

Definition 2.5
An m by t matrix A will be called bad, if the two following conditions are satisfied
(a) For every nonzero $u \in \{0,1\}^t$, if $|u| < m/\log_2 t$, then $Au \neq 0$.
(b) For every partial input u with at most $\lfloor t/4 \rfloor$ specified coordinates, there exists a total input u' extending u such that $Au' = 0$.

Since the multiplication of a vector u by a fixed matrix A is a linear operation, the property (a) implies also the following. If u, v are distinct vectors, $Au = 0$ and the Hamming distance of u and v is less than $m/\log_2 t$, then $Av \neq 0$. In terms of the linear codes, this means that the code $\{u : Au = 0\}$ has the minimum distance at least $m/\log_2 t$.

Lemma 2.6 *Let m and t be some nonzero natural numbers such that a bad m by t matrix exists. Let s be such that $4m(s+1) \leq t\log_2 t$. Then, every $(1,+s)$-b. p. computing $g_{m,t}$ has the size at least $2^{(m/\log_2 t-1)/2}$.*

Proof: Assume, A is a bad m by t matrix. Assume that a b. p. P computes $g_{m,t}$ and each of its computations reads at most s input bits repeatedly. Let P' be the restriction of P computing $g_A(u) = g_{m,t}(A,u)$. By definition, the function g_A is a function of t variables. Clearly, P' is not larger than P and it is also $(1,+s)$-b. p. Let the size of P' be c and let the minimum number of variables read on a computation path be d. Then $d > t/4$, because every partial input u of at most $t/4$ fixed variables may be extended to an input x with $g_A(x) = 1$ and also to x' with $g_A(x') = 0$. The existence of x follows from (b). The input x' may be chosen as an extension of u differing from x in one coordinate. By (a), we then have $g_A(x') = 0$.

The lower bound on the size of P' is proved by contradiction. Assume that

$$2\log_2 c + 1 < m/\log_2 t. \tag{2.1}$$

Multiplying this with the inequality $4m(s+1) \leq t\log_2 t$ from the assumptions of the lemma, we obtain

$$(2\log_2 c + 1)(s+1) \leq t/4. \tag{2.2}$$

This together with $d > t/4$ implies that the assumptions of Lemma 2.3 are satisfied for P', g_A and s replaced by $s+1$. Consider the partial inputs guaranteed by the lemma. By (b) from definition of a bad matrix and (2.2), it is possible to extend $[a_1, ..., a_{s+1}]$ to an input $x = [a_1, ..., a_{s+1}, a]$ with $g_A(x) = 1$. For $i = 1, ..., s+1$ let $y_i = [a_1, ..., b_i, ..., a_{s+1}, a]$. The inputs x and y_i form a forgetting pair. Moreover, the Hamming distance of x and y_i is at most $2\log_2 c + 1 < m/\log_2 t$. Hence, by (a) we have $g_A(y_i) = 0$. Consider the node w from Definition 2.2 used to x and y_i. This node is reached by the computation for x and also for y_i. However, since $g_A(x) \neq g_A(y_i)$, the computations for x and y_i reach different sinks. Hence, there is a node w' reached by both computations after w, where an input bit differing x and y_i is read. Since this bit was read also before reaching w, it is read twice in both computations. Since the sets of input bits, where x and y_i differ are disjoint for different $i's$, the computation for x reads at least $s+1$ input bits twice. This is a contradiction. Hence, (2.1) is not satisfied and we have $\log_2 c \geq (m/\log_2 t - 1)/2$. □

Lemma 2.7 *For every natural number t large enough and any natural number m satisfying $6\log_2 t \le m \le t/8$, there exists a bad m by t matrix A.*

Proof: For the proof of the existence of A we will replace (b) by the following stronger property.
(b') The linear span of any $t - \lfloor t/4 \rfloor$ columns of A is the whole space $\{0,1\}^m$.

By the following argument, (b') $\Rightarrow$ (b). If u' is a total input extending a partial input u, then Au' is a sum of two groups of columns in A. The first group are the columns corresponding to nonzero coordinates in the partial input u and the second group correspond to nonzero coordinates extending u to u'. Because of (b'), for any u of at most $\lfloor t/4 \rfloor$ specified coordinates it is possible to choose the extension u' in such a way that the two groups of columns have the same sum. Then, the total sum is zero. This implies (b).

The existence of A with (a) and (b') wil be established by proving that a matrix chosen at random satisfies these properties with positive probability. Assume, the entries of A are equal to 1 with probability 1/2 and that they are independent.

Let B_u be the event $Au = 0$. If u is a fixed nonzero vector, then Au is uniformly distributed over $\{0,1\}^m$. Hence, the probability of B_u is 2^{-m}. The probability of the disjunction of B_u over some set of vectors u is at most the sum of the probabilities of corresponding B_u. Hence, the probability that (a) is not satisfied is at most 2^{-m} times the number of nonzero u, $|u| < m/\log_2 t$. It is easy to verify that for every k at most $(t+1)/3$ we have, using also Stirling's formula,

$$\sum_{j=1}^{k} \binom{t}{j} \le 2\binom{t}{k} \le 2\left(\frac{te}{k}\right)^k .$$

Using this and the fact that the last expression is nondecreasing in k, if k is considered a real variable between 0 and t, we obtain that the probability that (a) is not satisfied is at most

$$2\binom{t}{\lfloor m/\log_2 t \rfloor} 2^{-m} \le 2\left(\frac{et\log_2 t}{m}\right)^{m/\log_2 t} 2^{-m}. \tag{2.3}$$

By the assumptions of the lemma, $m \ge 2e\log_2 t$. This implies that the fraction inside the bracket in (2.3) is at most $t/2$. Hence, (2.3) is bounded by

$$2\left(\frac{t}{2}\right)^{m/\log_2 t} \cdot 2^{-m} = 2 \cdot 2^{-m/\log_2 t} \cdot 2^m \cdot 2^{-m} \le \frac{1}{32}. \tag{2.4}$$

Now, let us consider the property (b'). A group of columns generates the whole space $\{0,1\}^m$ iff it is not contained in any subspace of dimension $m-1$, i.e. in a set of the form $\{z : \langle v, z \rangle = 0\}$, where v is a nonzero vector from $\{0,1\}^m$ and $\langle .,. \rangle$ denotes the scalar product mod 2. Consider the following property.
(b") Every nonzero vector $v \in \{0,1\}^m$ has nonzero scalar product with at least $\lfloor t/4 \rfloor + 1$ columns of A.

We shall prove that (b") and (b') are equivalent. To prove (b') $\Rightarrow$ (b"), consider a nonzero $v \in \{0,1\}^m$. If (b") is not satisfied, then there is a group of $t - \lfloor t/4 \rfloor$ columns of A with zero scalar product with some nonzero v. This contradicts (b'), since these columns do not generate the whole $\{0,1\}^m$. To prove (b") $\Rightarrow$ (b'), consider some group of $t - \lfloor t/4 \rfloor$ columns of A. By (b") and the characterization of subspaces from above, for any subspace of $\{0,1\}$ of dimension $m-1$, there is a column in this group not contained in the subspace. This implies (b').

The probability of (b") may be estimated as follows. Fix some nonzero $v \in \{0,1\}^m$ and let X_i be the scalar product of v and the i-th column of A. The $X_i's$ are independent and $\Pr(X_i = 1) = 1/2$. Let $h = t/2 - \lfloor t/4 \rfloor \geq t/4$. The probability that (b") is not satisfied for the given particular v is, by Chernoff inequality (see e.g. [1])

$$\Pr\left(\sum_{i=1}^{t} X_i \leq \lfloor t/4 \rfloor = t/2 - h)\right) \leq e^{-2h^2/t} \leq e^{-t/8}$$

Since there are $2^m - 1$ nonzero vectors v, the probability that (b") is not satisfied for at least one of them is the probability of the disjunction of $2^m - 1$ events of probability at most $e^{-t/8}$. Hence, the probability that (b") is not satisfied is at most $(2^m-1)e^{-t/8}$. This is at most $(2/e)^{t/8}$, since $m \leq t/8$. Hence, the probability that (b") is not satisfied tends to zero with increasing t. Together with (2.4), this implies that the probability that a random matrix does not satisfy some of the conditions (a) and (b") is less than 1 for t large enough. This proves the existence the required bad matrix A. □

Definition 2.8 *For every natural number $n \geq 6$, let the Boolean function f_n be defined as follows. The first $2\lceil \log_2 n \rceil$ bits are considered the binary representation of natural numbers m and t. If $m = 0$, $t = 0$ or $(m+1)t > n - 2\lceil \log_2 n \rceil$, the function f_n is 0. Otherwise, f_n is equal to $g_{m,t}$ applied to the next $(m+1)t$ bits after the representations of m and t.*

Theorem 2.9 *For every $\alpha \in (0,1)$ and n large enough, any $(1,+n^\alpha)$-b. p. computing f_n has size at least $\exp(\Omega(\sqrt{n^{1-\alpha}/\log_2 n}))$.*

Proof: In order to prove the theorem, we choose appropriate numbers m and t depending on n and α. Then, we use the fact that every $(1,+n^\alpha)$-b. p. computing f_n may be transformed by appropriate setting of the first $2\lceil \log_2 n \rceil$ variables to a $(1,+n^\alpha)$-b. p. computing $g_{m,t}$. Then, we apply Lemma 2.6 to this new b. p. Since a setting of some variables does not increase the size of the b. p., the lower bound obtained for $g_{m,t}$ in this way is valid also for f_n.

Let

$$\begin{aligned} m_0 &= 1/4 \cdot n^{(1-\alpha)/2}\sqrt{\log_2 n} \\ t_0 &= 3\frac{n^{(1+\alpha)/2}}{\sqrt{\log_2 n}}. \end{aligned}$$

Moreover, let $m = \lfloor m_0 \rfloor$ and $t = \lfloor t_0 \rfloor$. It is easy to verify that $m = (1+o(1))m_0$, $t = (1+o(1))t_0$ and $\log_2 t = \log_2 t_0 + o(1) = (1+o(1))\log_2 t_0$.

Since $(m+1)t \leq 3/4 \cdot n + o(n) \leq n - 2\lceil \log_2 n \rceil$, the value of f_n for this choice of m and t is determined by $g_{m,t}$ applied to appropriate input bits. Since $m = o(t)$

and $\log_2 t = O(\log_2 n) = o(m)$, the assumption of Lemma 2.7 is satisfied. Hence, there exists an m by t bad matrix. In order to use Lemma 2.6 for $s = n^\alpha$, it remains to verify its assumption on s, namely $4m(s+1) \le t \log_2 t$. We have

$$\frac{t \log_2 t}{4m} = (1 + o(1)) \frac{t_0 \log_2 t_0}{4m_0} = (1 + o(1)) \cdot 3n^\alpha \cdot \frac{\log_2 t_0}{\log_2 n}. \tag{2.5}$$

Since

$$\frac{\log_2 t_0}{\log_2 n} = \frac{1+\alpha}{2} - o(1) \ge \frac{1}{2},$$

we have that (2.5) is at least $(1 + o(1)) \cdot 3/2 \cdot n^\alpha \ge n^\alpha + 1$.

Lemma 2.6 implies that the size of any $(1, +n^\alpha)$-b. p. computing $g_{m,t}$ and hence f_n is $\exp(\Omega(m/\log_2 t))$. This implies the theorem, since

$$\frac{m}{\log_2 t} \ge (1 - o(1)) \frac{m_0}{\log_2 n} = (1 - o(1)) \cdot \frac{1}{4} \cdot \frac{n^{(1-\alpha)/2}}{\sqrt{\log_2 n}}.$$

□

Bibliography

[1] B. Bollobás, *Random graphs*, Academic Press Inc., London, 1985.

[2] A. Borodin, A.Razborov and R. Smolensky, "On Lower Bounds for Read-k-times Branching Programs", *Computational Complexity* 3, pp. 1 - 18.

[3] M. Ftáčnik and J. Hromkovič, "Nonlinear Lower Bound for Real-Time Branching Programs".

[4] S. Jukna, "A Note on Read-k-times Branching Programs", Universität Dortmund - Forschungsbericht Nr. 448, 1992.

[5] K. Kriegel and S. Waack, "Exponential Lower Bounds for Real-time Branching Programs" *in* Proc. FCT'87, LNCS 278, pp. 263 - 267.

[6] D. Sieling, "New Lower Bounds and Hierarchy Results for Restricted Branching Programs", TR 494, 1993, Univ. Dortmund, to appear in *J. of Computer and System Sciences*.

[7] D. Sieling and I. Wegener, "New Lower bounds and hierarchy results for Restricted Branching Programs", *in* Proc. of Workshop on Graph-Theoretic Concepts in Computer Science WG'94, LNCS 903, Springer, 1994, pp. 359-370.

[8] I. Wegener, "On the Complexity of Branching Programs and Decision Trees for Clique Functions", *JACM* 35, 1988, pp. 461 - 471.

[9] I. Wegener, "Efficient data structures for the Boolean functions", *Discrete Mathematics* 136 (1994), pp. 347-372.

[10] S. Žák, "An Exponential Lower Bound for One-time-only Branching Programs", *in* Proc. MFCS'84, LNCS 176, pp. 562 - 566.

[11] S. Žák, "An Exponential Lower Bound for Real-time Branching Programs", *Information and Control*, Vol. 71, No 1/2, pp. 87 - 94.

[12] S. Žák, "A superpolynomial lower bound for $(1, +k(n))$- branching programs", *in* Proc. MFCS'95, to appear in LNCS 969, Springer, 1995.