Branching Programs: Paradoxes in the Method

Žák, Stanislav
2009

Institute of Computer Science
Academy of Sciences of the Czech Republic

Branching programs: paradoxes in the method

Stanislav Žák

Technical report No. 1052

Pod Vodárenskou věží 2, 182 07 Prague 8, phone: +420 266 051 111, fax: +420 286 585 789,
e-mail:ics@cs.cas.cz

Institute of Computer Science
Academy of Sciences of the Czech Republic

Branching programs: paradoxes in the method[1]

Stanislav Žák[2]

Technical report No. 1052

Abstract:

We present three paradoxes in the theory of branching programs. They are of the same type – an absolutely negligible phenomenon produces a massive proof machinery.

The first paradox concerns read-once branching programs. Many lower bound results are in fact based on the small phenomenon that two parts arising from the cube $\{0,1\}^k$ by fixing a variable to 0 or to 1 are (sub)cubes, too.

For the second paradox the phenomenon consists of two simple facts. The first one says (on the intuitive level of reasoning) that any application of one instruction of any Turing machine means reading (and possibly remembering) the content of one input bit. The second one concerns convex functions. It says that the values of any secant in its inner interval (resp. of any tangent) are above (resp. below) the values of the (convex) function in question. This phenomenon gives a new possibility of classifying branching programs, and it implies a general lower bound method for branching programs without any restriction, in particular. Based on this phenomenon new-type classes of branching programs are known such that in polynomial they cover all polynomial read-once branching programs, of course, and moreover, they cover in polynomial most of witness functions for superpolynomial lower bounds known till now. For the so strong classes of branching programs the new general method mentioned above implies superpolynomial lower bounds.

The third phenomenon is also easy. We know that in any branching program the computations on any two inputs of different function values have the last common inner node. Our phenomenon says that the same -the last common node- is true for three inputs of different function values, too. In this paper from this phenomenon we derive a new lower bound proof for branching programs. We prove a superpolynomial lower bound for a nontraditional but reasonable class of branching programs connected with some special function considerably large in that context than the class of read-once branching programs.

Keywords:
Branching programs, lower bound techniques

[1]Research partially supported by the "Information Society" project 1ET100300517 and the Institutional Research Plan AV0Z10300504.

[2]Institute of Computer Science, Academy of Sciences of the Czech Republic, P. O. Box 5, 18207 Prague 8, Czech Republic,

1 Introduction

First we introduce some usual definitions for completeness of the text and then we present three phenomenon mentioned in the abstract in a more detailed way.

1.1 Technical preliminaries

By a branching program (b.p.) P (over binary inputs of length n) we mean a finite, oriented, acyclic graph with one source (in-degree = 0) where all nodes have out-degree = 2 (so-called branching or inner nodes) or out-degree = 0 (so-called sinks). The branching nodes are labeled by variables x_i, $i = 1, ..., n$, one out-going edge is labeled by 0 and the other by 1, the sinks are labeled by 0 or by 1. If a node v is labeled by x_i we say that x_i is tested at v.

For an input $a = a_1...a_n \in \{0,1\}^n$ by $comp(a)$ we mean the sequence of nodes starting at the source of P and ending in a sink. In the sequence for each i, $1 \leq i \leq n$, at any node with label x_i the next node is pointed by the edge with label a_i.

A special case of b.p. with in-degree = 1 in each node (with exception of the source) is called decision tree.

If a node $v \in comp(a)$ we say that a reaches v. If a and b reach v and immediately below v they reach different nodes we say that $comp(a)$ and $comp(b)$ diverge in v (or shortly a and b diverge at v). Similarly for more than two inputs.

P computes function f_P which on each $a \in \{0,1\}^n$ outputs the label of the sink reached by a.

We say that P computes in time $t(n)$ if each its computation is of the length at most $t(n)$.

The well-known class of restricted b.p.'s are so-called read-once branching programs in which along each computation each variable is tested at most once. Read-once b.p.'s compute in time n, of course.

If $comp(a)$ has a common part with a path p in P we say that a follows p.

By a distribution we mean any mapping D of $\{0,1\}^n$ to (the set of nodes of) P with the property that for each a $D(a)$ is a node of $comp(a)$ ($D(a) \in comp(a)$). The class of the distribution at node v is the set of all a's mapped to v.

Let v be a node of P and let A be a set of some (not necessarily all) inputs reaching v. We say that T is a tree developed in v according to P with respect to A iff the branches of T simply follow (only) the paths of P starting at v and followed by inputs from A till the sinks (in T no joining of paths is allowed, of course). Moreover each edge pointing to a node with out-degree = 1 in T is repointed to its successor. Hence in T each node has out-degree = 2 with exception of its leaves.

By the size of P we mean the number of its nodes. By the complexity of a Boolean function f we mean the size of the minimal b.p.'s computing f.

It is a well-known fact that superpolynomial lower bound on the size of b.p.'s implies superlogarithmic lower bound for space complexity of Turing machines [9].

1.2 Some comments on lower bound problems

The main goal of the theory of branching programs is to find Boolean functions superpolynomially hard for the general branching programs. In the ideal case to find these functions in P or at least in

NP. This would solve the famous problem $LOG \subset ?\ P$ or respectively $LOG \subset ?\ NP$.

At the origin of the theory(last seventies) two possible ways of the next development of the theory were considered. The first possible way was to work with the general b. p.'s and gradually step by step to prove higher and higher lower bounds with the hope that one day a superpolynomial bound will be achieved. However, the history has chosen the another possible way - to prove superpolynomial lower bounds for restricted b. p.'s gradually less and less restricted with the hope that one day we obtain a superpolynomial lower bound for b.p.'s without any restriction, i.e. for general b.p.'s. As an example we may follow the line of the research consisting in the next sequence of restrictions: read-once b.p.'s - real-time b.p.'s - (1,+k)-b.p.'s - syntactic k-b.p.'s - k-b.p.'s - time restricted b.p.'s cf. [7] (with superpolynomial lower bound for $n.log\ n$-time restricted b.p.'s as a consequence of [1]). In general it seems that the proof of a superpolynomial lower bound for a class of b.p.'s is more valuable than the proof of a higher lower bound but for a class of b.p.'s which is not so large.

In the past, the definitions of new classes of restricted b.p.'s and the proofs of the respective superpolynomial lower bounds were in any case great successes, millstones in the development of the theory. On the other hand taken from the critical point of view with the full respect to the past efforts we may say that the proofs of lower bounds achieved in the last 30 years are in some sense also witnesses - memory-stones - of our incapability to develop our ideas to the more general proofs, especially to the proofs of $LOG \neq P$ resp. $LOG \neq NP$.

In this paper we want to prolong and to modify the second way of research mentioned above (superpolynomial lower bounds for restricted b.p.'s). Taking into account the line of restrictions mentioned above (read-once b.p.'s - .... - time restricted b.p.s) we see that each of these restrictions is in some sense easy to understand. The question arises whether the restrictions which are so easy for our thinking reflect the wonderful world of b.p.'s in a sufficient way, sufficient for doing effective proofs. In this sense in the paper we want to do some small nonconventional steps outside of the practice used up to now. We introduce a restriction which is closely related to a function which is suspected to be a witness function for an important class of programs ($n.(log\ n)^2$-time restricted b.p.'s) and which in that context in polynomial covers read-once b. p.'s, of course. For this restriction we prove a superpolynomial lower bound . We hope that this is the way how to achieve the desired result.

1.3 Paradox 1: Read-once branching programs

The first paradox is based on a small phenomenon as follows: Let us divide the cube $\{0,1\}^n$ into two disjoint parts by fixing a variable $x_i = 0$ and $x_i = 1$. The phenomenon is that both parts are cubes, too, (with dimension $n-1$). The property "to be a cube" is a hereditary one under the operation of fixing the variables. This property has a deep impact for read-once b.p.'s.

Let a node of a b.p. be reached by a set of inputs which contains a subcube and let this node test a variable which is free for our subcube. After the test both edges out-going the node are followed by a subset of the subcube each, and moreover according to the phenomenon above both these subsets are subcubes, too.

Let P be a read-once b.p. and let v be its inner node. Let p be a computational path leading from the source to v. For p there is a corresponding subcube S such that its fixed bits are those tested along p. Since P is a read-once b.p. at v one of only free variables of S may be tested. According to the phenomenon above both edges out-going v are followed by two subcubes both with one fixed variable more. We repeat this operation on the next nodes. Since in each such node a free variable of some subcube is tested both next nodes are reached (by some subcubes). As a consequence we have that each node of b.p. P_v induced by v is reached by some (small) subcube of S. Hence in each node of P_v there are some inputs which have followed p (and tested the variables which are fixed for S). Since P is a read-once b.p. we obtain that in the whole P_v there is no test on any variable which is

fixed for S. Further we deduce that in P_v there is no test on any bit tested on any path from the source to v. This fact was crucial for the first lower bound proofs [9, 6] and then for many other proofs of lower bounds for read-once branching programs in the history. The concrete formal expression of this observation can be viewed in the definition of the so-called k-mixed functions and the lemma saying that each such function requires read-once b.p.'s of size at least 2^{k-1}([7] refers to [2]). [7] also mentions many read-once lower bound results.

Hence, starting with a trivial consideration of so simple property of the cube we achieved the point of departure of many lower bound proofs for read-once b.p.'s. This is a paradox!

1.4 Paradox 2: Windows and Trees

Let us introduce the second paradox mentioned in the Abstract. From the human point of view on the prematematical intuitive level of thinking it seems that at the moment when some instruction of a Turing machine is performed the content of the tape cell scanned by the head of the machine is read and maybe remembered. This commonly used intuitive idea has a strong impact in the theory of branching programs as we want to demonstrate in this subsection.

Let us have a Turing machine M which has two-way read-only input tape and a sublinear read-write work tape. By a configuration of M we mean a pair consisting of the position of the input head and of the situation on the rest of the machine (the state of the finite control, the content of the work tape and the position of its head). The resulting graph (the configurations in the role of nodes and transitions in the role of edges) has some significant properties. It is a finite non-empty acyclic graph which has one source (= the starting configuration), with sinks labelled by 1 (=YES) and 0 (=NO) (= the ending configurations). Each non-sink node has out-degree 2 and it is labelled by the input bit scanned by the head on the input tape (seeing the node in question as the configuration). The out-going edges are labelled by 0 and 1 according to the fact to what next configuration the input tape symbols 0 or 1 lead.

The definition of branching programs is based on a small step in our reasoning, simply to abandon Turing machines and to investigate the graphs with these properties above - the branching programs. Now firstly from the log-exp relation between the length of the work tape and the number of configurations of a TM we see why the superpolynomial lower bound on the number of nodes of branching programs implies the superlogarithmic lower bound on the space complexity of Turing machines [9]. And secondly, the question what the machine has read and remembered is now translated into the context of branching programs and it remains an actual one. Intuitively the role of our phenomenon plays now the suspicion that the test at a node of any branching program is in fact an action of remembering a bit of information (=the content of the tested bit).

Now we start to apply the usual mathematical routine to investigate our phenomenon. Firstly we see that we must in some sense define what is "remembering" and consequently also what is "forgetting". More concretely before us there is a problem to define what the branching program has remembered and forgotten about the input word a along $comp(a)$, i.e. in each node and in each edge of $comp(a)$. Such a definition is a great task, of course. Fortunately, there is also a small task more appropriate to the ability of the present author. We want to define what the program has remembered or forgotten about the values 0 or 1 of the input bits along $comp(a)$. We shall investigate this small problem, of course.

So, we are in the situation that (for a given b.p.) we want to catch the desired information concerning the contents of the bits of any input word a of length n. We shall proceed in such a way that we shall assign a word w of length n over the ternary alphabet $\{0, 1, +\}$ to each node and to each edge of $comp(a)$. Each such w will have the following property: for each i, $1 \leq i \leq n$, $w_i = a_i$ *or* $w_i = +$. The sign "+" will be called "a cross", and on the intuitive level of reasoning it will stand for "unknown" or "forgotten". The assigned word w will be called window on a at the respective node or edge of

$comp(a)$. By its length we shall mean the number of its non-crossed bits. On the intuitive level of our reasoning these non-crossed bits will represent the remembered information.

Before creating the formal definition of windows we have two simple ideas at our disposition. Firstly, on the intuitive level a test in b.p. means remembering of (the content of) one bit. Hence our next formal definition of windows should respect the rule "one test, (exactly) one cross is removed". Secondly, on the intuitive level it is difficult to say what is "remembered" but it is easy to say the complementary thing what is "forgotten" or "unknown". We see intuitively that "the bit which will be tested in the future is unknown or forgotten one now, and it should be a crossed one, now". - Our intuition reflects in the next formal definition of window.

Definition 1 *Let P be a branching program, v be its node. Let A be a subset of the set of all inputs reaching v. From v we develop a tree $T_{v,A}$ according to P with respect to A. From the level of sinks we arbitrarily(!) test appropriate bits in such a way that in these tests both out-going edges are followed by inputs from A (till the moment when in each leaf of the resulting tree $T_{v,A}$ there is exactly one input from A).*

For each $a \in A$ we define the window $w(a, v, A)$ on a at v with respect to A in such a way that $w(a, v, A)_i = +$ if and only if in $T_{v,A}$ there is a test on bit i along the branch followed by a. (On the other -non-crossed- bits $w(a, v, A)$ equals a.)

The length of a window is the number of its non-crossed bits.

The window $w(a, v, A)$ is said to be a natural one iff A is the set of all inputs reaching v.

Comments.

i) We see that the definition of windows is ambiguous due to the arbitrary last part of the construction of $T_{v,A}$ beginning at the level of sinks. We may do so since the next theorems concerning windows will be independent on this part of the construction.

ii) In the definition if we replace "node v" by "edge e" we obtain the window assigned to the edge e.

iii) For each a in a given set A comparing the window on a at v with the respect to A and the window on a at an out-going edge e leaving v with respect to the subset of A corresponding to e we see that the rule "one test, (exactly) one cross is removed" mentioned above is satisfied.

iv) It is clear that the simple thing holds: "The larger A, the larger number of branches in the tree, the larger number of crosses, the shorter windows".

Another confirmation of our intuition is given by a small theorem in [10] saying that for each (general) branching programs computing symmetric words it holds that during the computation on such a word each pair of symmetric positions must be non-crossed at least in one natural window. In other words branching programs computing symmetric words must compute in a human-like way. In the history the analysis of this phenomenon lead to the definition of multisyms (below).

For the theory of windows the following theorem ([10]) is very important.

Theorem 1 *Let P be a branching program and A be a set of inputs of length n distributed in (the set of nodes of) P. Let $A_1, \ldots A_r$ be all classes of this distribution. Then*
$log_2\ (size\ of\ P) \geq log_2\ r \geq log_2\ |A| - n + avelw$
where $avelw$ is the average length of windows of inputs from A according to A_i's, $i = 1, \ldots r$.

We postpone the proof for a while till the moment when we will have made some planned lemmas on binary trees.

Comments.
i) Now is the point to check the relation between our intuition on one hand and the formal theory on the second hand. Firstly the intuitive rule "one test, (exactly) one cross is removed" is in the theory satisfied. Secondly, for symmetric words the intuitive strategy to compare symmetric bits is in the theory unique one possible. And now thirdly, Theorem 1 confirms our intuition that remembering many information about many inputs requires a large memory, i. e. a large branching program. We see that our construct -windows- is closely related to our intuition.

ii) Moreover Theorem 1 gives a general method for proving large lower bounds. For proving a lower bound for a Boolean function it suffices to prove that on any b.p. this function requires large windows on many inputs.

The next theorem will be useful in the sequel.

Theorem 2 *Let P be a branching program, A be a set of inputs of length n and z be a number.*
Then there is a set $A_z \subseteq A$ of cardinality at least $(1-\frac{1}{z}).|A|$ such that for each $a \in A_z$ all natural windows on a along $comp(a)$ are of length at most $z.(log_2\ (size\ of\ P) - log_2\ |A| + n)$.

Proof: We distribute each $a \in A$ to the node of its (first) maximal natural window. Let r be the number of classes of this distribution. We see that for each $a \in A$ the window according to (the corresponding class of) this distribution is not shorter than the maximal natural window on a and therefore it is not shorter than any natural window along whole $comp(a)$.

What concerns of windows according to our distribution the previous theorem says that
$avelw \leq log_2\ (size\ of\ P) - log_2\ |A| + n$.

Further at most $\frac{1}{z}.|A|$ inputs from A have windows according to our distribution of length at least $z.avelw$. Hence at least $(1-\frac{1}{z}).|A|$ of inputs from A have windows according to our distribution of length at most $z.avelw$. Hence - as stated above - $(1-\frac{1}{z}).|A|$ of inputs from A have each their natural windows (along the whole computations) of length at most $z.avelw$. Q.E.D.

□

Comments.
i) Informally, the theorem says that for any large A all a's from A but a fraction of them have short natural windows.

Now, it is the time to investigate the second part of the considered phenomenon, especially the simple statements concerning convex functions and their secants and tangents.

First let us take into account the function $f(x) = x.\log\ x$ on interval $[1;2]$. Its second derivative is $log_2\ e.\frac{1}{x} > 0$. Hence f is convex on $[1;2]$. The straight line $s(x) = 2x - 2$ is the secant of f, their common points are $(1;0)$ and $(2;2)$. Therefore on [1;2] $s(x) \geq f(x)$.

Let k be an arbitrarily chosen natural number and let m be a number, $m \in [0;2^k]$. For $x = 1 + \frac{m}{2^k}$ from the inequality $s(x) \geq f(x)$ above we may obtain the inequality $\frac{(2^k+m).k+2.m}{2^k+m} \geq \log(2^k+m)$.

The semantics of the last inequality is as follows: Let us have a binary tree (with a root)in which some branches are of the length k and the others are of the length $k+1$. Let m be the number of all nodes of our tree which are in the distance k from the root and which are not leaves. Now, we see that the inequality in question says that the average length of branches in our tree is at least binary

logarithm of their numbers.

Let us take a (general) binary tree with a root. Let us consider its transformation into a tree such that the maximal difference of lenghts of its branches equals 1 (this is the case of the tree above). Our transformation works step by step. In one step in the maximal depth a pair of leaves with a common predecessor are cut, in the minimal depth one leaf obtains two successors. In each step the number of branches remains the same, their average length becomes shorter (and shorter). Hence from the trivial fact concerning secants of convex functions we obtain the well known fact that the average length of branches of binary tree is at least logarithm of their number.

Now, let f be a function convex on an interval and let t be any its tangent. Our simple fact says that that for each x from the mentioned interval $t(x) \leq f(x)$ holds. Hence for any $x_1, ..., x_r$ from the interval the following holds:

$\frac{1}{r}\sum_{i=1}^{r} f(x_i) \geq \frac{1}{r}\sum_{i=1}^{r} t(x_i)$

$= t(\frac{\sum_{i=1}^{r} x_i}{r})$ since t is a linear function

$= f(\frac{\sum_{i=1}^{r} x_i}{r})$ for t being the tangent in the point $(\frac{\sum_{i=1}^{r} x_i}{r}; f(\frac{\sum_{i=1}^{r} x_i}{r}))$.

Further from above we know that the function $x.\log\ x$ is a convex one on $((0;\infty)$. Therefore we have

$\frac{1}{r}\sum_{i=1}^{r} x_i \log\ x_i \geq \frac{\sum_{i=1}^{r} x_i}{r}.(log\ (\sum_{i=1}^{r} x_i) - \log\ r)$

and then

$\frac{\sum_{i=1}^{r} x_i \log\ x_i}{\sum_{i=1}^{r} x_i} \geq\ log \sum_{i=1}^{r} x_i - \log\ r.$

The semantics of the last inequality is as follows: Let x_i be the number of leaves (and branches) of a tree T_i. From above we know that the sum of lengths of branches of T_i is at least $x_i.log\ x_i$. Hence we have the following lemma.

Lemma 1 *Let us have r binary trees. Let l be the average length of their branches and S be the sum of (the numbers of) their leaves. Then $l \geq log_2\ S\ -\ log_2\ r$.*

One possible proof can be found in [5]. For $r = 1$ we obtain the well-known fact that the average length of branches of any binary tree is at least the binary logarithm of their numbers.

Now we are able to prove Theorem 1.

Proof: We use our tree lemma above. Let us take into account the trees developed according to A_i's. Let S be the number of their leaves and l be the average length of their branches. According to lemma we have $l \geq log_2\ S - log_2\ r$. We see that $S = |A|$ and that $l = n - avelw$. Hence we have $n - avelw \geq log_2\ |A| - log_2\ r$. Q.E.D.

□

Using simple facts concerning secants and tangents of convex functions we have proven a theorem giving general lower bound method for general b.p.'s.

In the history (1981) of this matter the requisition of this property lead to our definition of windows ([8,10]).

We see that windows and trees are complementary in some sense. At each node long windows are the same as short branches in the respective tree and vice versa. The situation is discussed in detail in [5]. Windows (and trees) can be used for classification of branching programs. For original definition of windows [8],[10] we have that in read-once b.p.'s at each node each inputs reaching it have the same natural windows in some sense. The same observation can be made for tree-like analysis in [5]. Now we have had the possibility to define a larger class of b.p's simply by allowing that at each node the natural windows could be somewhat different. Now, there has been a task before us firstly to demonstrate that in this way we have obtained a class of b.p.'s which is computationally stronger than read-once b.p.'s, and secondly to prove superpolynomial lower bounds for our new class of b.p.'s. This question is investigated in [10,3,5].

The deepest analysis of the question is made in [5]. There so called Δ-balanced b.p.'s are introduced. The definition means that at least $2^{n-\Delta}$ inputs can be distributed among nodes of the program in question in such a way that each class of the distribution induces a decision tree without the branches of length shorter about Δ than the average length (i.e. without too long windows by complementarity of trees and windows) and with some additional special properties.

In [5] it is proven that read-once b. p.'s are covered by Δ-b.p.'s with $\Delta = 0$ and that many functions which are superpolynomially hard for b.p.'s restricted in the classical way are easy for Δ-balanced b.p.'s. including Ajtai's function [1]. Moreover a superpolynomial lower bound was proven for a class of so called strongly k-stable functions. These functions include among others the classical clique function (which is in NP) and a type of pointer function (which is in AC^0). The question is open to achieve similar results for restrictions more free than Δ b.p.'s.

Moreover using windows a superpolynomial lower bound for the function of so called multisyms was proven in [4] . This function was defined after an analysis of the result concerning symmetric words mentioned above. Multisyms will play the key role in the next sections.

We may conclude that a very small phenomenon (remembering of the content of an input cell scanned by the head of a Turing machine when one instruction of TM is performed , and the very simple facts about tangents and secants of convex functions) has induced a deep massive proof machinery. However intuitive experience with this machinery has induced a desire for to find another lower bound method more emphasizing separation of inputs with opposite function values. Such a method is demonstrated in the next subsection and in the rest of the paper.

1.5 Paradox 3: A new principle for proving lower bounds

Let us introduce our third small phenomenon (mentioned in the abstract) in a more technical form.

Let us have a b.p. P which computes a function f. Let $a = a_1...a_n,\ b = b_1...b_n,\ x = x_1...x_n$ be three inputs. Let for all $i = 1...n$ $x_i = a_i$ or $x_i = b_i$, and moreover $f(a) = f(b) \neq f(x)$.

We see that there is at least one node in P reached by all three a, b, x - e.g. the source of P. On the other hand $f(a), f(b)$ differ from $f(x)$ - therefore in P there is an inner node v which is the last node in P reached by all three a, b, x.

Now on the intuitive level let us argue how this phenomenon could imply lower bounds.

Let us consider the situation in v. In v (according to the definition of v) a, b, x must diverge. Let bit i be tested in v. It must be $a_i \neq b_i$ otherwise $a_i = b_i = x_i$ and no divergence is possible.

Wlog we may assume that a follows 0-edge outgoing from v, b follows 1-edge and x follows 0-edge , too. Since v is "last", the computational paths starting in v and given by a and given by b must

not have any common node until the moment when a, x diverge (b never meets a before x leaving a). This observation implies a new principle for proving lower bounds.

For its intuitive description it suffices to imagine that in one node many inputs with rich mutual relations can be distributed with the effects that many paths must not have any common nodes till some significant depths. If this is true also in the nodes below we obtain a binary tree embedded in the branching program in question. As a consequence we can prove a lower bound.

The last section introduces a proof of a lower bound based on this phenomenon.

2 Multisyms

Considering the windows on symmetric words (above in Subsection 1.4) and then on the set of words $\{u^k | u \in \{0,1\}^{n/k}\}$ which have windows of length k but only on few inputs we have obtained the following definitions of so-called multisyms. This function was already used in [4] for testing lower bound techniques based on the notion of windows for the case of read-once b.p.'s. A superpolynomial lower bound has been achieved.

For appropriate n's we understand the binary inputs of length n as matrices $m \times k$ where $m.k = n$. We say that some t columns are covered by a row r if the bits of these t columns on row r have the same value 0 or 1 (they are monochromatic on r). We say that such a matrix is a t-multisym if each choice of t columns is covered by a row.

It is easy to see that for any constant t t-multisyms are in P and that for any unbounded and reasonably constructible function $t(n)$ $t(n)$-multisyms are in co-NP.

For the purposes of this text we shall use only 2-multisyms, simply multisyms.

We often use notation $m = \epsilon(n).log\ n$ and $k = \frac{n}{\epsilon(n).log\ n}$. It is easy to see that for $\epsilon(n) \geq 2$ the number of multisyms is at least 2^{n-1}. Indeed the number of non-multisyms is at most

$$\binom{k}{2}.2^m.2^{n-2m} \leq (\frac{n}{\epsilon(n).log\ n})^2.2^{n-m} \leq 2^{n+2.log\ (\frac{n}{\epsilon(n).log\ n})-\epsilon(n).log\ n} \leq 2^{n-1}.$$

By a canonical branching programs computing multisyms we mean any branching program P consisting from a chain of subprograms $(P_{i,j})$ for $i, j = 1...k,\ i \neq j$. Each program $P_{i,j}$ is responsible for verifying the covering of the pair of columns (C_i, C_j) by at least one row. Each $P_{i,j}$ has two sinks - simply to separate the matrices with covered (C_i, C_j) from the others. The first sink of $P_{i,j}$ ("the pair C_i, C_j is covered") is the source of the next subprogram $P_{i',j'}$, the second sink of $P_{i,j}$ ("the pair C_i, C_j is not covered") is one of sinks of P ("nonmultisyms"). Each such $P_{i,j}$ is a chain of microprograms M_r for each row r. M_r is responsible for testing of covering of (C_i, C_j) by row r. (M_r tests equality of two bits.)

Let P be a branching program and let the input words be viewed as matrices. Let a, b, x be inputs with the functional values $f_P(a) = f_P(b) \neq f_P(x)$. Let v be a node of P, C_1, C_2 be columns. By $R(v, a, C_1, C_2, b, x)$ we mean that

i) v is a node reached by all three a, b, x,
ii) in v a, b, x diverge, x follows a,
iii) the test in v is in C_1 or C_2, the test in which x leaves a for the first time (below v) is in the other column C_2 or C_1 respectively,
iv) b never meets a before x leaving a.

The point iv) reflects the main feature of our new principle from Subsection 1.5.

We say that P is a *reasonable* branching program iff P satisfies the next restriction R:

There is a set A of inputs with the same functional value $o \in \{0,1\}$, $|A| \geq 2^{n-2}$, such that for each $a \in A$ and for each two columns C_1, C_2, $C_1 \neq C_2$, there is a node $v \in comp(a)$ such that
i) the set $B_{a,v} =_{df} \{b | f_P(b) = o,\ b\ reaches\ v\ and\ a, b\ diverge\ at\ v\} \neq \emptyset$ and,
ii) there is an input x, $f_P(x) \neq o$, satisfying for each $b \in B_{a,v}$ $R(v, a, C_1, C_2, b, x)$.

The restriction R (*reasonable*) may seem to be very special since it is formulated in close connection to multisyms. On the other hand multisyms seem to be a very appropriate candidate for a function superpolynomially difficult for general b.p.'s computing in time $n.(log\ n)^2$. We plan to remove R in the next development of our ideas. R simply documents the level of proof we are able to achieve contemporarily. Hence our approach to start with R is legitimate.

Moreover the next section demonstrates that R is taken from life. We prove that the canonical branching programs are reasonable. Moreover what concerns superpolynomial lower bound for read-once b.p.'s starting the proof "by contradiction" by the (false) assumption that there is a read-once b.p. of polynomial size computing multisyms we obtain that this program is a reasonable one. Hence the proof of superpolynomial lower bound for reasonable programs computing multisyms - we produce in this paper - implies immediately the same superpolynomial lower bound for read-once programs and even for a larger class of programs. Therefore our task to prove a superpolynomial lower bound for reasonable branching programs has its sense also from the traditional point of view based on traditional definitions such as read-once etc., and the corresponding lower bounds.

3 Reasonability of R

In the next lemma we demonstrate that R is satisfied by canonical branching programs.

Lemma 2 *Each canonical branching program (computing multisyms) is reasonable.*

Proof: Let P be a canonical branching program computing multisyms. We want to verify R for the set of all multisyms.

Let a be a multisym, let C_1, C_2 be a pair of columns. Let us take the subprogram $P_{1,2}$ of P corresponding to the pair C_1, C_2 and within it the first microprogram M_r (responsible for a row r) such that r covers C_1, C_2 in a. In M_r a leaves $P_{1,2}$. Let v be the input node of M_r. We want to prove $R(v, a, C_1, C_2, b, x)$ for some nonmultisym x and for all $b \in B_{a,v}$.

Let x be a nonmultisym which at v follows a and which ends at sink of $P_{1,2}$, i.e. x does not cover the pair C_1, C_2. Such an x exists: e.g. x such that outside of C_1, C_2 x equals 1, on both C_1, C_2 x has at least one 1, x does not cover C_1, C_2, and on M_r at v x follows and immediately then diverges a. Let b equals the described x on all bits but the bit tested at v. b is a multisym and therefore $B_{a,v} \neq \emptyset$. For each $b' \in B_{a,v}$ $R(v, a, C_1, C_2, b, x)$ clearly. Q.E.D.

□

Comment. In the proof, v is the last common node of a, b', x for all $b' \in B_{a,v}$. The motif of our principle from Subsection 1.5 is present here.

The next theorem reduces the proof by contradiction of superpolynomial lower bound for read-once b.p.'s to the problem of superpolynomial lower bound for reasonable b.p.'s.

Theorem 3 *Let P be a read-once branching program computing multisyms with $(log\ log\ n)^3 \leq \epsilon(n) \leq log\ n$. Let $size(P) \leq n^q$.*

Then P is a reasonable branching program.

Proof: By $\alpha(n)$-strong multisyms we mean multisyms with at least $\alpha(n).log\ n$ (covering) monochromatic rows for each pair of columns. For $\alpha(n) \leq \sqrt{\epsilon(n)}$ the number of $\alpha(n)$-strong multisyms is at least 2^{n-1}. This follows from the fact that the number of strings which are not $\alpha(n)$-strong multisyms is at most

$\binom{k}{2}.2^m.\alpha(n).log\ n.\binom{m}{\alpha(n).log\ n}.2^{n-2m} \leq n^3.2^{n-m}.m^{\alpha(n).log\ n}$
$\leq 2^{n-\epsilon(n).log\ n+3+\alpha(n).log\ n.log\ (\epsilon(n).log\ n)} \leq 2^{n-1}$.

According to the previous theorem there are at least 2^{n-2} of $\alpha(n)$-strong multisyms such that each their natural window in P is of length at most $2.(q.log\ n+1)$. Let S be the set of such multisyms. For any multisym a and for each pair of columns C_1, C_2 we define a pair of nodes $v_{a,C_1,C_2}, w_{a,C_1,C_2}$ *in comp*(a) (in brief v, w) as follows: Let r be the first row monochromatic on C_1, C_2 such that both its bits (on C_1, C_2) are both tested along *comp*(a). v_{a,C_1,C_2} is the node of the first (in *comp*(a)) test in question, w_{a,C_1,C_2} is the node of the second one.

Lemma 3 *Let $a \in S$, let C_1, C_2 be columns.*

Then there is no input c such that

a) c meets a at the moment when comp(a) still has not covered C_1, C_2 and still has touched (by any test) at most $2.(q.log\ n\ +\ 1)$ pairs covering C_1, C_2 in a.

b) there is a bit $i, i \in C_1 \cup C_2, a(i) \neq c(i)$, $a(i)$ tested by comp(a) before meeting comp(a) and comp(c).

Proof: By contradiction. Suppose such an c exists. We construct c' as a prolongation of c (it means we define values of the bits not tested by *comp*(c) before meeting with *comp*(a)). Outside of C_1, C_2 we give the same values which are in a. Hence c' and a cannot diverge by tests outside of C_1, C_2.

Fact 1. On C_1, C_2 there is no bit tested by *comp*(c) but not tested by *comp*(a) before their meeting.

By contradiction. Let i_1 be such a bit and let i'_1 be a bit associated with it on the same row on C_1, C_2. We construct a', a'' two prolongations of a. Outside of C_1, C_2 they equal a. On C_1, C_2 on pairs of bits on the same rows with exception of i_1, i'_1 we give values $01, 10$ so that each pair C, C_1 and C, C_2 is covered (we have at our disposition at least $(\alpha(n) - q).log\ n$ free pairs of bits). In i_1 we give different values for a', a'', in i'_1 we give the same value for both a', a''. a', a'' differ only on i_1 - therefore they will not diverge because on i_1 they must not test since i_1 was tested by c. They reach the same sink. On the other hand -due to the arrangement on i_1, i'_1- one of them is multisym and the other not. A contradiction.
On $6.log\ n$ pairs of bits on rows on C_1, C_2 we may give values $10, 01$ in such a way that both a and c' cover each C, C_1 and C, C_2 for each $C \neq C_1, C_2$.

Fact 2. c' covers all pairs C, C' for $C, C' \neq C_1, C_2$.

By contradiction. On C_1, C_2 on the remaining pairs of bits we give values 00. a, c' reach the same sink but only exactly one is a multisym. A contradiction.

Fact 3. c covers C_1, C_2.

By contradiction. Let c not cover C_1, C_2. On the remaining bits of C_1, C_2 with exception of j associated with i we construct a common prolongation in such a way that the resulting c' does not cover C_1, C_2. To j we give the value $a(i)$. a', c' reach the same sink but only exactly one of them is a multisym. A contradiction.

We see that c' covers all pairs of columns. On the other hand on C_1, C_2 there is a common prolongation a' of a such that a', c' reach the same sink with arrangement on bits i, j such that a' is not multisym. A contradiction. The proof of our lemma is closed.

□

Lemma 4 *The number of pairs covering C_1, C_2 (in a) touched by (tests of) $comp(a)$ before w is at most $2.(q.log\ n + 1)$.*

Proof: By contradiction. Let this number be larger. Then beginning at v many of bits from the pairs in question are tested by $comp(a)$ and therefore many bits are introducing natural windows on a between v and w. But $a \in S$, therefore each its natural window is of length at most $2.(q.log\ n + 1)$. Hence soon after v a c must meet a and close some bit in the window on a. But this is impossible according to the previous lemma. A contradiction. Q.E.D.

Let us verify that R holds. For $a \in S$ an $\alpha(n)$-multisym and for any pair of C_1, C_2 we have defined the nodes $v_{a,C_1,C_2}, w_{a,C_1,C_2} \in comp(a)$ in brief v, w.

Let b be a multisym which differs a only in the bit tested at v. We see that $B_{a,v} \neq \emptyset$.

Let us define x as follows. Outside of C_1, C_2 $a = x$. On C_1, C_2 x equals a on all bits tested by $comp(a)$ before w, on the bit tested at w x differs a and on the remaining bits x is defined arbitrarily in such a way that x is a nonmultisym. Now it suffices to prove that any $b' \in B_{a,v}$ never meets a before w nor in w. But this follows from the previous lemmas.

□

□

4 Combinatorics

Lemma 5 *Let T be a sequence of places, let $|T|$ be its length. Let k be a natural number.*

Let $\binom{k}{2}$ pebbles be distributed on places of T, at most $k-1$ pebbles on one place.

Let u, α be numbers, $u < \frac{\binom{k}{2}}{|T|}$ and $u < k-1$.

Then in T there is a subsequence S of α places such that

(i) on each of them more than u pebbles are distributed,

(ii) between each two places neighboring in S there is at most $d = \alpha.\frac{|T|-o}{o-1}$ nodes in T where $o = \frac{\binom{k}{2}-|T|.u}{k-1-u}$.

Proof: By a marked place we mean any place with at least $u+1$ pebbles. In T there is at least $o = \frac{\binom{k}{2}-|T|.u}{k-1-u}$ marked places. The average distance between marked places is at most p; $p = \frac{|T|-o}{o-1}$.

The number of intervals of length at least $\alpha.p+1$ of non-marked places is at most $\frac{|T|-o}{\alpha.p+1} = \frac{|T|-o}{\alpha.\frac{|T|-o}{o-1}+1} < \frac{o}{\alpha}$.

Hence there are at most $\frac{o}{\alpha}$ subsequences of marked places in which each two neighbors have distance at most $\alpha.p$ and moreover there is at least one such subsequence which contains at least α nodes. □

□

Corollary 1 *For $k = \frac{n}{\epsilon(n).\ log\ n}, \epsilon(n) \leq log\ n, T(n) = n.(log\ n)^2,\ \alpha = (log\ n)^2$ and $u = \frac{\sqrt{(n)}}{(log\ n)^2}$, $d(n)$ is at most $6.(log\ n)^6$.*

Proof: $d(n) = \alpha(n).\frac{T(n)-o}{o-1} =$

$\frac{\alpha(n).T(n)}{o-1} - \frac{\alpha(n).o}{o-1} \leq$

$\frac{n.(log\ n)^4}{o-1}$ (since $o > 1$)

$= \frac{n.(log\ n)^4}{\frac{\binom{k}{2}-T(n).u}{k-1-u}-1} \leq$

$\frac{n.(log\ n)^4.(k-1-u)}{\binom{k}{2}-T(n).u-k+1+u} \leq$
$\frac{n.(log\ n)^4.k}{\frac{1}{2}.\binom{k}{2}} \leq 6.(log\ n)^6.$ □

□

5 Lower bound

Definition 2 *Let B be a binary decision tree. Let M be a set of inputs, $M \subseteq \{0,1\}^n$. Let L_B be the number of all leaves of B and L_B^M be the number of all leaves of B reached by inputs from M. By M-ratio of B we mean the number $p_B^M = \frac{L_B^M}{L_B}$.*

Let T be a subtree of B. Let L_T be a number of all leaves of T and $L_T^{M,B}$ be the number of all leaves of T reached (in B !) by inputs from M. By (M,B)-ratio of T we mean the number $p_T^{M,B} = \frac{L_T^{M,B}}{L_T}$.

Theorem 4 *The polynomially sized reasonable branching programs cannot compute multisyms with $(loglog\ n)^3 \leq \epsilon(n) \leq log\ n$ in time $n.(log\ n)^2$.*

Proof: By contradiction. Let P be a reasonable branching program computing multisyms in time $n.(log\ n)^2$ and with $|P| \leq n^q$ for some q. Since P is reasonable there is a set A of multisyms of cardinality at least 2^{n-2} with the corresponding properties.

For each multisym $a \in A$ we understand the nodes of $comp(a)$ as a sequence of places. Let $v \in comp(a)$ be a node (place) and let C_1 be the column of the variable tested in v. The number of pebbles on v is given by the number of columns C such that $R(v,a,C_1,C,b,x)$ for some b's, x's. Hence on each v there is at most $k-1$ pebbles where k is the number of columns in the matrices (multisyms).

From restriction R it follows that on $comp(a)$ there must be at least $\binom{k}{2}$ pebbles. Therefore according to Lemma 5 and Corollary 1, for each a , in $comp(a)$ there is a relatively long $((log\ n)^2)$ subsequence S_a of nodes with relatively many $(\frac{\sqrt{(n)}}{(log\ n)^2})$ pebbles and with the distances between their nodes relatively small (at most $6.(log\ n)^6$).

Let us distribute each multisym $a \in A$ to the first node of S_a. Let v be the node with a maximal class M of this distribution. Hence $|M| \geq \frac{2^{n-2}}{n^q}$.
From v we develop the syntactic tree B' according to P ("syntactic" means that we take into account also the branches which are not followed by any input e.g. in case of repeated tests on the same variable) till the depth $(log\ n)^2.6.(log\ n)^6$ in P (this number corresponds to $(\log\ n)^2$ nodes of S_a and the maximal distances $6.(log\ n)^6$ between them). Below this depth we continue the developing of the tree ignoring the repeated tests. On each branch b at the node corresponding to the related sink of P we add some subtree S_b which on each their branches tests all variables not tested below v till now. We know that the length of branches (constructed till now) is at least n and that each such branch is followed by at most one input (of length n). The branches (constructed till now) can be of different length (due to the possible different number of repeated tests in the first interval of length $(log\ n)^2.6.(log\ n)^6$ along different branches). To obtain a full tree B' of certain length we add some full tree T_b of appropriate length to each branch b shorter than the longest one of branches constructed till now. As a result we obtain a full binary decision tree B' of depth $n+\delta$. δ is at most $(log\ n)^2.6.(log\ n)^6$.
We modify B' as follows. Below v till the depth $(log\ n)^2.6.(log\ n)^6$ in the middle of each edge we insert a new node with the same test as in the node from which the edge in question out-goes. On the dead edge of this test we add a dummy full binary tree D of depth $log((log\ n)^2.6.(log\ n)^6) + (log\ n)^2 + \delta$. Below the depth $(log\ n)^2.6.(log\ n)^6$ we add dummy subtrees of depth $\delta - l$ where l is level of B' below $(log\ n)^2.6.(log\ n)^6$. Maximal l for this operation is δ.
Let B be the resulting tree. The number of its leaves is at most
$2^{n+\delta}$ +
$+2^{6.(log\ n)^8}.2^{log((log\ n)^2.6.(log\ n)^6)+(log\ n)^2+\delta}$ +

$+\sum_{l=0}^{\delta} 2^l.2^{\delta-l} \leq$
$\leq 2^{n+\delta+1}$.
It follows that M-ratio of B
$p_B^M \geq \frac{|M|}{2^{n+\delta+1}}$.

For the proof of our theorem it suffices to prove the next lemma.

Lemma 6 *There is a full binary decision tree T of depth $(log\ n)^2 + \delta$ rooted in v such that*
a) Each its branch (considered as a sequence of nodes) is a subsequence of a branch of B.
b) $p_T^{M,B} \geq p_B^M$.
c) The nodes of T reached by inputs from M in depth $(log\ n)^2$ are pairwise different in P.
d) The number of leaves of T reached by inputs from M is the same as the number of nodes of T reached by inputs from M on level $(log\ n)^2$.

Since P is small ($\leq n^q$) from c)and d) it follows that the number of leaves of T reached by inputs from M is small ($\leq n^q$) which implies that $p_T^{M,B} \leq \frac{n^q}{2^{(log\ n)^2+\delta}}$. According to b) it follows that $\frac{n^q}{2^{(log\ n)^2+\delta}} \geq p_T^{M,B} \geq p_B^M \geq \frac{|M|}{2^{n+\delta+1}}$. A contradiction. Q.E.D. □

The theorem is proven, it remains to prove the lemma.

Proof of Lemma 6.
The main point of construction of the desired tree T is recursive use of procedure $Proc$.
In the first step of $Proc$ we take into account the both subtrees of B rooted by the immediate successors v_0, v_1 of v. If one of these subtrees is reached by no input from M we add it to the constructed tree T'. We know that in the other case in P the branches starting in v_0 and that ones starting in v_1 and both followed by inputs from M don't meet till the depth $\frac{\sqrt{n}}{(log\ n)^2}$. This follows from the restriction R. (Cf. c) of Lemma.)
For $i = 0, 1$ in v_i and below v_i we distribute each input a from M to the node of B where a has the next (second) node of its special subsequence S_a. (Similarly as above in case of v for each node w of this distribution the branches followed by inputs from the corresponding class of the distribution starting at 0-successor of w do not meet the other ones starting at 1-successor till the depth $\frac{\sqrt{n}}{(log\ n)^2}$ by the restriction R.)
First let us take into account the easy case when on each branch there is at most one node of this distribution. Among the full subtrees with roots in the nodes of the distribution we chose that one which has maximal (M, B)-ratio. For case of back reconstruction of B from T from the point of view of b) of Lemma it is important that this ratio is at least the same or larger than the ratio of the tree rooted by corresponding v_i. In T' there will be an edge from v to the root of the chosen tree. On each tree chosen in the present iteration of $Proc$ we apply the next iteration.

The difficult case is when some nodes of this distribution are on the same branch of B. We will still construct trees rooted in the nodes of our distribution with the property that their sets of leaves are pairwise disjoint. This property implies that the maximal (M, B)-ratio of these trees is at least equal or larger than the (M, B)-ratio of the tree rooted in the corresponding v_i.
Let leaders be nodes (of this distribution) which have no such nodes as predecessors. (Each leader is the first on its branch.)
We take into account a partition of the set of nodes of the distribution according to the equivalence "to be below (or equal to) the same leader".

Let w_i's be a class of nodes (a class of the partition) where sets of multisyms M_i's are distributed.

We construct the corresponding trees R_i's rooted in w_i's. Each R_i contains all branches followed by inputs from M_i. The sets of leaves of R_i's are pairwise disjoint (since in each leaf there is at most one input - from the construction of B). In general the union of R_i's do not cover the whole subtree

rooted in the leader of w_i's because in general there are branches not followed by any input from M. To each R_i we potentially add some other branches to save the possibility to continue the construction of full tree T of depth $(log\ n)^2 + \delta$ in case when the modified R_i is chosen as the tree with maximal (M, B)-ratio.

We proceed according to the following rules:

Let us take R one of R_i's. In B let us follow its branches from its root to its leaves. Let u be a node on some branch b such that only one outgoing edge is followed by inputs from M_i. In case when u is in B in depth at least $d = (log\ n)^2.6.(log\ n)^6 + \delta$ we add the whole subtree of B rooted by the out-going edge in question to R or we do nothing if u plays its role for another R which consumes this subtree in question.
In case when u is in the depth at most d the most difficult case is such that the out-going edge in question is followed by inputs from some other M_i's.
We saturate the need of subtree in the direction of the out-going edge in question using the added subtree rooted in the middle of this edge. The added subtree is large enough to yield subtrees of desired depth $(log\ n)^2 + \delta$ for at most $6.(log\ n)^6$ R_i's (on one branch of B)in at most $(log\ n)^2$ iterations of *Proc.* Cf. the definition of B.

From T' we construct the desired full binary tree T of depth $(log\ n)^2 + \delta$ as follows:

To obtain T we modify T' only below the depth $(log\ n)^2$ (in T'). Moreover from the construction of T' we know that the nodes of T' reached by inputs from M in depth $(log\ n)^2$ are pairwise different in P. Hence c) of Lemma is satisfied.

From each node of T' in depth $(log\ n)^2$ in T' we follow the branches in B (!). If the node in question (on level $(log\ n)^2$ and below) of T' is not reached in B by any input from M we simply add a subtree of an appropriate depth to gain the desired depth $(log\ n)^2 + \delta$ of T. (In case of the back reconstruction of B from T b) of Lemma is not corrupted.)

If a node has only one out-going edge followed by inputs from M we prolong the dead edge by an subtree of appropriate depth to gain the desired depth $(log\ n)^2 + \delta$ and we follow the edge with inputs from M.

If in a node both outgoing edges are followed by inputs from M we consider two subtrees rooted in this node. Each of these two subtrees contains the subtree rooted by the outgoing edge in question and the dummy tree rooted by the middle of the opposite out-going edge (cut in the appropriate depth to gain the desired depth $(log\ n)^2 + \delta$ of T). We chose that one with maximal number of inputs from M. d) of Lemma is satisfied. In both cases the back reconstruction of B does not corrupt b) of Lemma.

From the construction of B it follows that the operations above are ensured till the depth $(log\ n)^2.6.$ $(log\ n)^6 + \delta$. This is sufficient for our construction of T.

The conditions a), b), c), d) of Lemma are satisfied.

It remains to verify that T is a full tree of depth $(log\ n)^2 + \delta$.
From the description of *Proc* it is easy to see that T is a full tree till the level (in T) at least $(log\ n)^2$. On this level the nodes not reached (in B) by any input from M have an appropriate prolongation by a subtree till the desired depth $(log\ n)^2 + \delta$ - this follows from the construction of B and from the description of *Proc.* Q.E.D. □

What concerns the contradiction constructed within two sentences introduced immediately after the formulation of Lemma 6 we see that the contradiction remains valid even for $q = \frac{log\ n}{2} - 1$. Hence the following Corollary holds.

Corollary 2 *The reasonable branching programs of size* $n^{\frac{log\ n}{2}-1}$ *cannot compute multisyms with* $(loglog\ n)^3 \leq \epsilon(n) \leq log\ n$ *in time* $n.(log\ n)^2$.

After a simple analysis of Theorem 3 and of its proof we can obtain the following lower bound for read-once b.p.'s.

Corollary 3 *The read-once branching programs of size* $n^{\sqrt{log\ n}-7}$ *cannot compute multisyms with* $\epsilon(n) = log\ n$.

What concerns our new and very simple principle (Subsection 1.5) we see that via its main feature catched in R it produces massive proofs and yields reasonable results. The third paradox mentioned in the Abstract has been demonstrated.

Bibliography

[1] M. Ajtai - A non-linear time lower bound for Boolean branching programs, Proc. of 40th IEEE Ann. Symp. on Foundations of Computer Science, 1999, pp. 60-70

[2] S. Jukna - Entropy of contact circuits and lower bounds on their complexity, Theoretical Computer Science 57, 1988, 113-129.

[3] S. Jukna, S. Žák - On branching programs with bounded uncertainty, Proc. of ICALP'98, LNCS 1443,Springer, Berlin, 259-270.

[4] S. Jukna, S. Žák - Some notes on the information flow in read-once branching programs, in Proc. of 27th Ann. Conf. on Current trends in Theory and Practice of Informatics, LNCS 1963, Springer, Berlin, 2000, pp. 356-364

[5] S. Jukna, S. Žák - On Uncertainty versus Size in Branching Programs, Theoretical Computer Science, Vol. 290, 2003, pp. 1851-1867 (ISSN: 0304-3975)

[6] I. Wegener - On the complexity of branching programs and decision trees for clique functions, Journal of the ACM 35, 461-471.

[7] I. Wegener - Branching programs and binary decision diagrams, SIAM, 2000

[8] S. Žák - Information in Computation Structures, Acta Polytechnica, Vol. 20, 1983, No. 4, pp.47-54,(ISSN:1210-2709)

[9] S. Žák - An exponential lower bound for one-time-only branching programs, in Proc. of MFCS'84, LNCS 176, 562-566.

[10] S. Žák - A subexponential lower bound for branching programs restricted with regard to some semantic aspects, Electronic Colloqium on Computational Complexity, Report Series 1997, ECCC TR97-50, Trier, 1997.