A Superpolynomial Lower Bound for a Class of Restricted Branching Programs

Žák, Stanislav
2007

Institute of Computer Science
Academy of Sciences of the Czech Republic

A superpolynomial lower bound for a class of restricted branching programs

Stanislav Žák

Technical report No. 1012

November 2007

Pod Vodárenskou věží 2, 182 07 Prague 8, phone: +420 266 053 111, fax: +420 286 585 789,
e-mail:ics@cs.cas.cz

Institute of Computer Science
Academy of Sciences of the Czech Republic

A superpolynomial lower bound for a class of restricted branching programs[1]

Stanislav Žák[2]

Technical report No. 1012

November 2007

Abstract:

We considerably improve the lower bound result (Jukna, Žák 2000) concerning read-once branching programs. We prove a superpolynomial lower bound for a considerably larger class of branching programs. The new lower bound technique is based on a tight observation concerning separation of positive and negative input strings.

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, stan@cs.cas.cz

1 Introduction

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 nodes) or out-degree = 0 (so-called sinks). The branching nodes are labelled by variables x_i, $i = 1, ..., n$, one out-going edge is labelled by 0 and the other by 1, the sinks are labelled by 0 or by 1.

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 each 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).

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 restriction of 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 in P we say that a follows this path.

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. 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.

There is a long history of proving superpolynomial lower bounds for restricted b.p.‘s, especially for read-once b.p.‘s (see Wegener 2000). The strike was done by Ajtai (1999) with proof of superpolynomial lower bound for b.p.‘s computing in time $n.log\ n$. We are searching for superpolynomial lower bound for less restricted b.p.‘s computing in time $n.(log\ n)^2$. In this text we present a technique based on a new principle as follows.

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 $f(a) = f(b) \neq f(x)$.

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 a non-sink node v which is the last node in P reached by all three a, b, x.

Let us consider the situation in v. In v (according to definition of v) a, b, x must diverge. Let the i-th variable 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 1-edge , too. Since v is "last", the computational paths starting in v and given by a and b must not meet until the moment when b, x diverge. This is our new principle.

For its application it remains to imagine that in one node many inputs with rich mutual relations can be distributed with the effects that many paths till some significant depths must not have any common nodes. As a consequence we can obtain lower bounds.

Motivation for the presented research is the great challenge to prove superpolynomial lower bound for branching programs computing in time more than $n.log\ n$ (cf. Ajtai 1999). We have created a new lower bound technique based on the principle stated above. The first application of this technique gives a lower bound for a class of restricted branching programs computing in time $n.(log\ n)^2$ probably incomparable with that of Ajtai and , moreover, containing the class of polynomially sized read-once branching programs. From the last fact it follows that the presented result is a considerable improvement of that one from (Jukna, Žák 2000) giving a lower bound for read-once branching programs. The main success of this paper is traversing a long way starting by the formulation of a very simple principle above and continuing by transformation of it to an effective lower bound giving a reasonable result.

2 Multisyms

For appropriate n's we understand the binary inputs of length n as matrices $m \times k$ where $m.k = n$. We say that such a matrix is a t-multisym if for each choice of t columns there is a row r monochromatic on them (we say that r covers this choice of columns) . For the purposes of this text we shall use only 2-multisyms, simply multisyms. This function was used to prove superpolynomial lower bound for so-called read-once branching programs (see Jukna, Žák 2000).
We often use notation $m = \epsilon(n).log\ n$ and $k = \frac{n}{\epsilon(n).log\ n}$. It is easy to see that for $\epsilon(n) \geq 3$ the number of multisyms is at least 2^{n-1}. Indeed the number of multisyms is at least
$2^n - \binom{k}{2}.2^m.2^{n-2m} \geq$
$2^n - n^2.2^{n-m} \geq$
$2^n - 2^{n+2.log\ n-\epsilon(n).log\ n} \geq$
$2^n - 2^{n-log\ n} \geq 2^{n-1}$.
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). Each $P_{i,j}$ has two sinks - simply to separate the matrices with covered (C_1, C_2) from the others. The first sink of $P_{i,j}$ is the source of the next subprogram $P_{i',j'}$, the second sink of $P_{i,j}$ is one of sinks of P. Such a $P_{i,j}$ is a chain of microprograms M_r for each row r. M_r is responsible for covering of (C_1, C_2) on 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 v be a node of P, a, b be positive inputs, x be a negative one, C_1, C_2 be columns, all a, b, x reach v. 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, the test in which x leaves a for the first time (below v) is in C_2,
iv) b never meets a before x leaving a.
Of course, this definition is inspired by multisyms, positive and negative inputs means inputs reaching sinks with different labels.
We say that P is a reasonable branching program iff P satisfies the next restriction :
For at least 2^{n-2} inputs a's and for all C_1, C_2 there is a node v of P such that
R1 there are b, x satisfying $R(v, a, C_1, C_2, b, x)$ or $R(v, a, C_2, C_1, b, x)$.
and
R2 if some b' reaches v and $test_v b' \neq test_v a$ (a and b' diverge at v) then $R(v, a, C_1, C_2, b', x)$.

The next section demonstrates that R1, R2 are taken from life.
The restriction "reasonable b.p." may seem be very special since it is formulated in close connection to multisyms and therefore it can be reasonably used only for b.p.'s computing this function. 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 in the next development of our ideas to remove our assumption with both R1, R2. Hence our approach to start with R1, R2 is legitimate.

3 Reasonability of R1, R2

In the next lemma we demonstrate that R1, R2 are satisfied by canonical branching programs.

Lemma 1 *Each canonical branching program (computing multisyms) is reasonable.*

Proof. Let P be a canonical branching program computing multisyms. Let us verify R1.

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 microprogram M_r (responsible for a row r) in which 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 b, x. Let x be a nonmultisym which at v follows a and which ends at sink of $P_{1,2}$. (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 then diverges a).

Let b be a multisym which at v follows the opposite edge.Clearly
$R(v, a, C_1, C_2, b, x)$. R1 is verified.

R2 is satisfied clearly.Q.E.D. □

Further we want to prove that also all polynomially sized read-once branching programs computing multisyms are reasonable, i. e. that 2^{n-2} a's and all C_1, C_2 satisfy our assumption with R1, R2. This task is not trivial. We shall use the technique of so called windows which originated in year 1981 (Žák 1983). This technique is two-faced. One face is represented by a lemma which is in a strictly classical combinatorial form on one hand, but which is relatively sterile from the point of intuition and inspiracy on the other hand. The second face of our technique is represented by definitions and lemma's which may seem to be bizarre but they are still strictly mathematically correct on one hand, and very closed to intuition concerning flows of information along computational paths, and very inspirative in any case on the other hand. First let us present the first face of our technique - the combinatorial lemma.

Lemma 2 *Let us have r binary trees. Let S be the sum of (the numbers of) their leaves and l be the average length of their branches. Then $l \geq log_2\ S\ -\ log_2\ r$.*

One possible proof can be found in (Jukna, Žák 2003). 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. The second face of our technique is motivated by an attempt to catch the information concerning the contents of input bits remembered at different stages of computation. To each node of computation $comp(a)$ on an input a of length n we assign a word w of length n over ternary alphabet $0, 1, +$ ($+$ is called a cross) such that for all i, $1 \leq i \leq n$ $w_i = a_i$ or $w_i = +$. The number of noncrossed bits in w is called the length of the window w. Intuitively, the noncrossed bits represent the remembered information.

Definition 1 *Let P be a branching program, v be its node. Let A be a set of some inputs of length n reaching v. From v we develop a tree $T_{v,A}$ according to P with respect to A. From level of sinks we arbitrarily test appropriate bits in such a way that in each leaf of $T_{v,A}$ 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.

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

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".
Let us take into account the situation described in the definition and let us imagine two dummy nodes at the middle of both edges out-going v. For each $a \in A$ in the window at the dummy node exactly one cross on the bit tested at v is removed in comparison with the window at v. This property (one test - one cross is removed) corresponds very well to the intuition that the test at a node is in fact acquisition of the information about the content of the tested bit. In the history (1981) of this matter the requisition of this property lead to our definition of windows (Žák 1983, Žák 1997).
Another confirmation of our intuition is given by a small theorem in (Žák 1997) 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 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.
For the theory of windows the following theorem (Žák 1997) is very important.

Theorem 1 *Let P be a branching program and A be set of inputs distributed in (the set of nodes of) P. Let A_1, ... 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, $i = 1$, ... r.

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.

□

This theorem confirms our intuition that remembering many information about many inputs requires a large memory, i. e. a large branching program. Moreover this theorem gives a method of proving large lower bounds. For proving a lower bound for a Boolean function it suffices to prove that this function requires large windows on many inputs.

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 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 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.

□

Theorem 3 *Let P be a read-once branching program computing multisyms with $(log\ log\ n)^2 \leq \epsilon(n) \leq log\ n$. Let $size(P) \leq n^q$.*
Then there is a subset S of (the set of all) multisyms such that
a) $|S| \geq 2^{n-2}$,
b) For each $a \in S$ and for each pair of columns C_1, C_2 there is a node v such that R1,R2 hold.

Proof. By $\alpha(n)$-strong multisyms we mean multisyms with $\alpha(n).log\ n$ (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 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. 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 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 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. Proof of lemma is closed.

□

Lemma 4 *The number of pairs covering C_1, C_2 (in a) touched by (tests of) $comp(a)$ between v and 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 impossible according to the previous lemma. A contradiction. Q.E.D.

Let us verify that R1 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 us define b, x as follows. Outside of C_1, C_2 $a = b = x$. b differs a only on the bit tested at v. b is a multisym. 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 in such a way that to be nonmultisym arbitrarily. Now it suffices to prove that b never meets a before w nor in w. But this follows from the previous lemmas.
The same for R2.
□

□

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

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

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

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

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

$$\frac{n.(log\ n)^4.k}{\frac{1}{2}.\binom{k}{2}} \leq 6.(log\ n)^6.\ \square$$

□

5 Lower bound

Definition 2 *Let B be a full binary decision tree. Let M be a set of inputs, $M \subseteq \{0,1\}^n$. 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 leaves of T reached (in B) by inputs from M. By (M,B)-ratio of T $p_T^{M,B}$ we mean the number $\frac{L_T^{M,B}}{L_T}$.*

Theorem 4 *The polynomially sized reasonable branching programs cannot compute multisyms with $(loglog\ n)^2 \leq \epsilon(n) \leq log\ n$ 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 R1 it follows that on *comp*(a) there must be at least $\binom{k}{2}$ pebbles. Therefore according to Lemma 5, 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 $6.(log\ n)^6$

- see Corollary 1).
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. 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 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 that 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. In the middle of each edge below v till the depth $(log\ n)^2.6.(log\ n)^6$ 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
$p_B^M \geq \frac{|M|}{2^{n+\delta+1}}$.

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. □

Proof of Lemma.

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 in v_1 and followed by inputs from M don't meet till the depth $\frac{\sqrt{n}}{(log\ n)^2}$. This follows from the restriction R2. (Cf. c) of Lemma.)
For $i = 1, 2$ 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. 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-ratio. We notice 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. Let leaders be nodes (of this distribution) which have no such nodes as predecessors.

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 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 of depth $(log\ n)^2 + \delta$ in case when the modified R_i is chosen as the tree with maximal M-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 in at most $(log\ n)^2$ iterations of $Proc$.
From T' we construct the desired full binary tree T of depth $(log\ n)^2 + \delta$ as follows:
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 $(log\ n)^2 + \delta$. (In case of 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 one outgoing edge and the dummy tree rooted by the middle of the opposite out-going edge.

We chose that one with maximal number of inputs from M. In both cases the back reconstruction of B does not corrupt b) of Lemma.
(From construction of B it follows that the operation above is ensured till the depth $(log\ n)^2.6.(log\ n)^6 + \delta$. This is sufficient for 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. □

The proof above is the first application of the principle mentioned in Introduction. The result is modest but reasonable. The challenge is to remove restrictions R1, R2 and in this way to achieve a considerable improvement of Ajtai's result (Ajtai 1999).

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, 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

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

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

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

[6] 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.

[7] S. Žák - A lower bound technique for restricted branching programs (version G), Institute of Computer Science, Academy of Sciences, Prague, Technical report No. 991, March 2007