Upper Bounds for Gentle Branching Programs

Žák, Stanislav
1999

INSTITUTE OF COMPUTER SCIENCE

ACADEMY OF SCIENCES OF THE CZECH REPUBLIC

Upper bounds for gentle branching programs

Stanislav Žák

Technical report No. 788

November 17, 1999

Institute of Computer Science, Academy of Sciences of the Czech Republic
Pod vodárenskou věží 2, 182 07 Prague 8, Czech Republic
phone: +4202 66053690 fax: +4202 8585789
e-mail: stan@uivt.cas.cz

INSTITUTE OF COMPUTER SCIENCE

ACADEMY OF SCIENCES OF THE CZECH REPUBLIC

Upper bounds for gentle branching programs

Stanislav Žák[1]

Technical report No. 788
November 17, 1999

Abstract

In the theory of branching programs there are many results giving lower bounds for restricted branching programs. In [?] (and also in [?]) - among other results - it is demonstrated that the functions used for proving lower bounds are in fact easy functions in many cases. "Easy" means that the functions in question are computable on polynomially sized so-called gentle branching programs ("gentle" is a restriction based on a more careful observation of the nature of computations).

The present report continues this trend and proves that also the functions used for proving the latest lower bounds (Beame et al. [?], Ajtai [?]) are easy in the sense above. Some other results concerning upper bounds are added.

Keywords
branching programs, upper bounds

[1]The research was supported by the GA CR grant no. 201/98/0717.

1 Introduction

Branching programs are a general model of sequential computation with small (sublinear) memory. It is a well-known fact that the logarithm of any lower bound on the number of nodes of branching programs implies a lower bound on the space complexity on Turing machines. Especially, any superpolynomial lower bound on branching programs for a function in P would solve the $P =?LOG$ problem.

Many lower bounds were obtained for so-called restricted branching programs. The lower bounds for the basic restriction - the read-once branching programs - are ranging from $2^{c\sqrt{n}}$ in 1983 (1981) [?] (also [?]) to $2^{n-O(logn)^2}$ for a function in P and $2^{n-O(logn)}$ for a function in $DTIME(2^{O(logn)^2}$ in 1999 [?]. (The read-once branching programs are such that during each computation each variable is tested at most once.)

Some results were obtained for (1,+k)-branching programs (during any computation each variable is tested at most once with exception of k of them) and for syntactic k-branching programs (along any path in the program each variable is tested at most k-times). The latest lower bounds for branching programs of length $(1 + \epsilon)n$ [?] and for branching programs of length $k.n$ [?] were long-standing problems.

In [?], the authors have considered the nature of computations more carefully and they have obtained a new restriction based on the balancing the uncertainty - so called gentle branching programs. Using a new method they proved two lower bounds for such b.p..

This new restriction is interesting also from the point of view of upper bounds. In [?] it is proved that each (polynomially sized) read-once branching program is a (polynomially sized) gentle one. Further it is proved that many functions which are superpolynomially hard for restrictions mentioned above are polynomially easy for gentle branching programs. It seems that gentleness captures something like simplicity of functions or computations.

In this paper we add some new results to the study of relation between gentleness and simplicity. Especially we demonstrate that the functions used for proving the latest lower bounds [?], [?] are in fact very easy. To make the collection of upper bounds from [?] , [?] more complete we prove also a small upper bound for D_2 (Dyck) language which was used for the first lower bound of the form $2^{n/c}$ on read-once branching programs [?]. Moreover we demonstrate that by a change of the definition of gentle b.p. the pointer function used in [?] for the proof of the lower bound becomes an easy function.

2 Branching programs and windows

Let us fix some (standard) notation concerning branching programs. A branching program (b.p.) is a directed acyclic graph with one source. The out-degree of each (non-sink) node is 2. Every node is labeled by an input variable x_i (or equivalently: by a bit i) and the two out-going edges are labelled by tests "$x_i = 0$" and "$x_i = 1$". The (two) sinks (out-degree 0 nodes) are labeled by 0 and 1. By the *size* of a branching program P we mean the number $|P|$ of its nodes. The *computation* $comp(a)$ on an input $a \in \{0,1\}^n$ is a sequence of nodes of P which starts in the source of P and at each node v labelled by i, $comp(a)$ follows that edge going from v which corresponds to the test $x_i = a(i)$. If the computation $comp(a)$ goes through a node v, then we also say that the input a *reaches* this node. By $comp_v(a)$ we will denote the part of the computation $comp(a)$ starting from the node v. The program *computes* a Boolean function f if, for every input $a \in \{0,1\}^n$, the computation $comp(a)$ reaches a sink labelled by $f(a)$.

By a branching program of length l we mean such a b.p. that each its computation is of length at most l.

We want to catch what information about the input bits is remembered at each moment of any computation on any branching program. For this purpose in [**?**] the following definition of windows is introduced (the first attempt to define windows is in [**?**]).

Let P be a branching program and v be a node in P. Let $F \subseteq \{0,1\}^n$ be an arbitrary subset of inputs, each of which reach the node v.

Definition 2.1 *The* window $w(a,v,F)$ *of input* $a \in F$ *at the node* v *with respect to the set* F *is a string of length* n *in the alphabet* $\{0,1,+,\#\}$ *which is defined according to the following three rules. Let* $F(a)$ *be the set of those inputs* $b \in F$ *for which* $comp_v(b) = comp_v(a)$.

1. *We assign a simple-cross (+) to the* i*-th bit of* $w(a,v,F)$ *if*
   - *either there is a* $b \in F$ *such that the first divergency of* $comp_v(a)$ *and* $comp_v(b)$ *is caused by a test on* i *(in this case we call that cross the* down-cross*),*
   - *or the bit* i *is not tested along any computation* $comp(b)$ *for* $b \in F(a)$ *(in this case we call that cross the* up-cross*).*
2. *We assign a* double-cross *(#) to the* i*-th bit of* $w(a,v,F)$ *if it was not crossed according to the first rule, and if* $a(i) \neq b(i)$ *for some input* $b \in F(a)$.
3. *The remaining bits of* $w(a,v,F)$ *are non-crossed (i.e. specified) and their values are the same as in* a.

By the same way it is possible to define windows at an edge (instead of at a node).

If some bit is double-crossed (#) in a window of $a \in F$ at some node v with respect to F, then this bit is not tested and remains double-crossed in the windows of a at *each* subsequent node w of $comp(a)$ with respect to the set of all inputs from F reaching w. (Double-crossed bits are "forgotten forever".)

The larger is F the smaller is the number of non-crossed bits in the windows relative to F.

If $a, b \in F$ and $comp_v(a) = comp_v(b)$ then the windows of a and b (at v with respect to F) have the same sets of down-crosses, of up-crosses (+) and of double-crosses (#), and all non-crossed bits and down-crossed bits have the same contents in both a and b.

From the intuitive point of view we expect that , in some sense, the window $w(a, v, F)$ says what bits are remembered (=non-crossed bits) and what bits are forgotten (= crossed - +,# - bits) at the moment when the computation $comp(a)$ reaches the node v (with respect to the set F).

In [?], [?] we introduce three arguments that the definition is reasonable.

1. The test on bit i at a node v of $comp(a)$ causes exactly removing of the + on the bit i of $w(a, v, F)$. It means that we have an analogy with the real world where a test on a variable gives an information about exactly this variable.

2. Each branching program recognizing symmetric words (uu^R) has the natural property that during the computation on any symmetric word each pair of symmetric positions is non-crossed (it means remembered or known at the same moment) at least once.

3. Moreover we have a theorem saying that if in a b.p. P many computations have long windows then P must be large. This theorem corresponds with the property of the real world that for remembering large information about many objects a large memory is needed. The theorem is used as a method of proving lower bounds on branching programs in [?], [?].

3 Gentle programs

In [?], via windows we have defined an interesting restriction as follows.

Let P be a branching program and v be a node in P. Throughout this section, let $F \subseteq \{0,1\}^n$ be an arbitrary (but fixed) set of inputs which reach the node v, i.e. the computations on inputs from F go through the node v; in this case we say also that F is *classified at* v. We will always assume that the set F is *closed* in the following natural sense: $a \in F$, $b \in \{0,1\}^n$ and $comp(b) = comp(a)$ imply $b \in F$.

Let a be an input from F. Depending on what is the window $w(a,v,F)$ for this input a at the node v with respect to F, we define the following subsets of $\{1,\ldots,n\}$.

$N(a)$ $=_{df}$ the set of all non-crossed bits;

$D(a)$ $=_{df}$ the set of all double-crossed (#) bits;

$S(a)$ $=_{df}$ the set of those bits $i \in D(a)$, which were non-crossed in the window for a *immediately before* the node v. i.e. which were non-crossed in the window for a at the corresponding edge, feeding into v.

Let also

N $=_{df}$ the set of all bits which are non-crossed and have the same value in the windows at v of *all* inputs from F (the *common specified part* of F), and

D $=_{df}$ the set of all bits which are double-crossed in the windows at v of *all* inputs from F (the *core* of F)

Definition 3.1 *We say that F is classified at v* in a regular manner *with* fluctuation γ *and* deviation δ *if its core $D \neq \emptyset$ and, for every input $a \in F$, $|N(a) \setminus N| \leq \gamma$ and $max\{|D(a) \setminus D|, |D(a) \setminus S(a)|\} \leq \delta$.*

The fluctuation tells that the "mixed" non-crossed part of $N(a)$ has at most γ bits, whereas the deviation ensures that at least $|D(a)| - \delta$ bits of a where double-crossed at the node v for the first time.

Definition 3.2 *A branching program P is* gentle *on a set of inputs $A \subseteq \{0,1\}^n$ with fluctuation γ and deviation δ if there is a distribution $\varphi : A \to V$ of these inputs among the nodes of P such that each (non-empty) class $F = \{a \in A : \varphi(a) = v\}$ of this distribution is classified at the corresponding node v in a regular manner with the fluctuation γ and deviation δ. We also say that a program is* α-gentle *if it is such on some set of at least $2^{n-\alpha}$ inputs.*

Parameters α, γ and δ range between 0 and n, and reflect the 'degree of gentleness': the smaller they are the more gentle the program is. In [**?**] we show that read-once branching programs (1-b.p.) are very gentle (with $\alpha \leq 1$ and $\gamma = \delta = 0$).

4 Upper bounds for the latest witness functions

The latest results in the theory of branching programs are the exponential lower bound for branching programs of length $(1+\epsilon)n$ (Beame et al. [**?**]) and the exponential lower bound for branching programs of length $k.n$ (Ajtai [**?**]). We want to prove that the witness function from the both papers are very easy for gentle b.p.s. We start with Ajtai's function.

Definition 4.1 *A is a Bolean function such that for all $a \in \{0,1\}^n$, $a = a_1...a_n$ $A(a) = 1$ iff the cardinality of the set $\{(i,j,k) \mid 1 \leq i < j < k$ and $i+j = k$ and $a_i = a_j = a_k = 1\}$ is an even number.*

Ajtai [**?**] has proved that A is exponentially difficult ($2^{\epsilon_k \cdot n}$) for each branching program of length at most $k.n$.

Theorem 4.2 *A is computable on a 1-gentle branching program with deviation 3 and fluctuation 3 of size $8.n^3$.*

Proof:

We will describe a branching program P computing A of size $8.n^3$ and then we will prove that P is a gentle branching program.

Informally speaking, P starts its computation from the right end of the input. Whenever P finds the symbol one on the position, say, k, P computes the parity of the number of pairs (i,j) such that $i+j = k$ and $a_i = a_j = (a_k =) 1$. P combines this parity with the parities for the previous positions k', $k' > k$.

More formally. P consists of parts P_n, ... , P_3. Each part P_l has two output nodes $v_{l,0}, v_{l,1}$ which are the input nodes of P_{l-1}. The output nodes of P_3 are sinks of the whole program P. The output node $v_{l,0}$ ($v_{l,1}$, resp.) (of the part P_l) represents (it means: $v_{l,0}$ ($v_{l,1}$, resp.) is reached by) all inputs $a = a_1...a_n$ where the parity of the number of triads (i,j,k) such that $i+j = k, a_i = a_j = a_k, k \geq l$ is equal to zero (one, resp.).

The two subprograms of P_l starting from the two input nodes of P_l have no common nodes. The edges going to two output nodes of P_l (= the input nodes of P_{l-1}) are combined in such a way that the condition on $v_{l,0}, v_{l,1}$ (above) is satisfied. In each from the both input nodes of P_l we compute as follows: we test a_l . From $v_{l-1,0}$ ($v_{l-1,1}$, resp.) the 0-edge leads immediately to $v_{l,0}$ ($v_{l,1}$, resp.). The 1-edge leads to the consequent tests on pairs (i,j) such that $i+j = l$. After each test the partial parity is remembered.

At this moment we see that P computes A and that the size of P is at most $8.n^3$.

For the purposes of the proof that P is gentle we need a more detailed description of P_n. In the source of P_n there is a test on the variable x_n. The 0-edge of this test goes immediately to the output node $v_{n,0}$. The 1-edge goes to the chain of subparts $Q_{n,i,j}$ where $i > j, i+j = n$ and $Q_{n,i,j}$ checks whether $x_i = x_j = 1$. Each $Q_{n,i,j}$ has two output nodes which reflect the parity of the number of pairs (i',j') such that $i' \geq i, j' \leq j, i'+j' = n, x_{i'} = x_{j'} = 1$. These two output nodes are the input nodes of $Q_{n,i-1,j+1}$.

In each input node of $Q_{n,i,j}$ a full binary tree of depth 2 starts where the first test is on x_i and second one is on x_j. Leaves of both trees are sticked to two output nodes of $Q_{n,i,j}$ according to parity requirements.

We have described P_n. The next P_l, $n-1 \geq l \geq 3$ are such that from both their input nodes a subprogram similar to P_n starts where only the index n is replaced by l. The output nodes of both subprograms are sticked to the output nodes according to parity requirements.

We introduce a small irregularity to the part P_{n-1}. The subprogram starting from the 0-sink $v_{n,0}$ of P_n tests firstly x_j and only then x_i in his subparts $Q_{n-1,i,j}$. This change gives us the fact that the sequence of tests $(x_n)x_{n-1}x_1x_{n-2}x_2x_{n-3}x_3...$ in P_n is the same as the sequence of tests $x_{n-1}x_1x_{n-2}x_2....$ in the subprogram starting from the 0-sink $v_{n,0}$ of P_n.

Now, we start the proof that P is a gentle branching program.

We define the set $F =_{df} \{a \in \{0,1\}^n | (a_n = 1 \ and \ a \ reaches \ v_{n,0}) \ or (a_n = 0 \ and \ in \ v_{n,0} \ a \ joints \ an \ a' = a_1...a_{n-1}1)\}$.

We distribute F to $v_{n,0}$. It is easy to check that $|F| \geq 2^{n-1}$. Hence, if we prove that F is regular we obtain that P is a 1-gentle branching program.

For each $a \in F$ $D(a) = D(a, v_{n,0}, F)$ contains the variable x_n. This follows from the definition of F and from the fact that below (after) $v_{n,0}$ there is no test on x_n. Hence $D =_{df} \bigcap_{a \in F} D(a) \neq \emptyset$.

For each $a \in F$ comp(a) goes through the input tests of parts P_l, $n-1 \geq l \geq 3$ where the test is always on x_l. Therefore some other double-crosses $\#$ in $w(a, v_{n,0}, F)$ may be at most on x_2, x_1. Thus we have $|D(a)| \leq 3$ which suffices for $|D(a) - D| \leq 3$ and $|D(a) - S(a)| \leq 3$. Hence the deviation is at most 3.

Now it suffices to estimate the fluctuation. Let us take a branch b in the tree T_F. (We know that the windows are the same for all inputs following b.) There are no up-crosses since $x_n...x_3$ are tested at the input nodes of $P_n...P_3$, x_1, x_2 are tested in P_n. Above we have argued that double crosses are at most 3 - on x_n and then maybe on x_1, x_2.

Let us take any variable y different from the variables tested in the last $Q_{n,i,j}$ and from $x_{n/2}$ for n even. Let us take a partial path in P_n which goes from the source (the test on x_n) by the 1-edge and which ends in the test on y in P_n. There are two inputs which follow this path, on y they branch and they join in F. In T_F they follow the same branch until the test on y since the path in P_n and P_{n-1} are the same (due to the irregularity we have introduced to the definition of P_{n-1}.

Therefore there are at most three non-crossed bits.
P is a 1-gentle branching program with deviation 3 and fluctuation 3. Q.E.D.
□

We see that though A is very difficult for branching programs of length $k.n$ A is easy for gentle programs even with small parameters (= very gentle).

Now let us consider one of the witness function of Beame et al. [?].

Let M be an $n \times n$-matrix over GF(3). Let $M_{i,j} =_{df} (-1)^{<i,j>}$. For $\sigma = \sigma_1 ... \sigma_n \in \{0,1\}^n$ we define $f(\sigma_1 ... \sigma_n) = 1$ iff $\sigma^T M \sigma \equiv 0 \ mod\ 3$.

They have proved that f is exponentially difficult for branching programs of length $(1+\epsilon)n$.

Theorem 4.3 *The function f s computable on 3-gentle branching program P of size $10n^2$ with deviation = 0 and fluctuation = 2.*

Proof:

Before constructing the desired program P let us notice that

$\sigma^T M \sigma = a + b + c + d$ where

$a = \sigma_1\sigma_1 M_{1,1} + \sigma_1\sigma_2 M_{2,1} + \sigma_2\sigma_1 M_{1,2} + \sigma_2\sigma_2 M_{2,2} = -\sigma_1\sigma_1 - \sigma_1\sigma_2 - \sigma_2\sigma_2$,

$b = \sigma_1.(\sum_{i=3}^n \sigma_i M_{i,1} + \sum_{j=3}^n \sigma_j M_{1,j})$,

$c = \sigma_2(\sum_{i=3}^n \sigma_i M_{i,2} + \sum_{j=3}^n \sigma_j M_{2,j})$,

$d = \sum_{i=3}^n \sum_{j=3}^n \sigma_j \sigma_i M_{i,j}$.

The program P computes and adds the values a, b, c, d. At the top P starts with a tree of depth 2 with tests on σ_1, σ_2. The four leaves of the tree are the input nodes of the second part of P where b and c are computed and which have three output nodes v_0, v_1, v_2. Each output nodes v_i represents one value of the sum $a+b+c$. These nodes are input nodes of the last part where d is computed.

It is clear that P computes f and that the size of P is at most $10n^2$.

We are going to prove that P is a gentle branching program. Let us take into account the node v_0 and all inputs σ reaching v_0 via the branch $\sigma_1 = 0, \sigma_2 = 0$ or the branch $\sigma_1 = 1, \sigma_2 = 0$. Let F be the union of all pairs of inputs which are the same on $\sigma_3 ... \sigma_n$ and which reach v_0 by different branches above. Since below v_0 there is no test on σ_1 we have double-crosses on σ_1 for each $\sigma \in F$. Hence $D \neq \emptyset$.

Now let us derive the value $|F|$. The computation of b starts in the node which represents the value -1 (of a). To reach v_0 it must have the value 1 on b. We see that $b = \sigma_1.(\sum_{i=3}^n \sigma_i M_{i,1} + \sum_{j=3}^n \sigma_j M_{1,j}) = \sum_{i=3}^n \sigma_i.2.M_{i,1} = \sum_{i=3}^n \sigma_i(-1).(-1)^{<i,1>}$.

For arbitrary values on $\sigma_3, ..., \sigma_{n-2}$ by a choice on σ_{n-1}, σ_n it is always possible to reach the value $b = 1$ since with growing i the value $< i, 1 >$ changes $0, 1$ in a regular way. Hence the expression $(-1)(-1)^{<i,1>}$ regularly changes the values $-1, 1$. Thus the size of F is at least 2^{n-3}. (P will be 3-gentle.)

The computation starting at v_0 begins by dummy tests on $\sigma_3....\sigma_n$. This implies that for each $\sigma \in F$ $D(\sigma) = \{\sigma_1\}$. (And $D(\sigma) - S(\sigma) = 0$ since # appears at v_0 for the first time.) Therefore the deviation is equal to zero.

All inputs from F have 0 on σ_2. T_F is a complete tree on $\sigma_3...\sigma_{n-2}$. Therefore $N(\sigma)$ may differ from N only on σ_{n-1}, σ_n. Hence the fluctuation is at most two. Q.E.D.

□

We see that f is very easy for (very) gentle branching programs.

5 Other upper bounds

Let f be a Boolean function of n variables such that $f(x) = 1$ iff x is the binary code of a well-formed expression over two sorts of brackets : $(,)$ and $[,]$. (The code is a mapping of $\{(,),[,]\}$ to the length-two binary strings.)

We know that f is difficult for 1- and real-time branching programs ($2^{n/48}$) - Kriegel, Waack [**?**].

Theorem 5.1 *f is computable on a 9-gentle branching program with deviation 0 and fluctuation 0 of size $O(n^5)$.*

Proof:
We will formulate a condition equivalent to the fact that some expression over the alphabet $\{(,),[,],\}$ is a well-formed one.

We say that a prefix p of an expression x is a leftmost (the first) interval iff p is the shortest prefix such that the projection of p to the alphabet $\{(,)\}$ (if p starts with () or to $\{[,]\}$ (if p starts with [) is a well-formed expression in $\{(,)\}$ ($\{[,]\}$, resp.).

The second interval is the first interval of the rest and so on.

Let I be an interval starting with, say, [. We are able to say what are its subintervals in [,] and what are their dephts of nesting.

Our condition is:
1. The expression can be divided into intervals.
2. For each interval I (in "[,]") for each its subinterval S (in "[,]") of any depth l of nesting the symbols "(,)" contained in S and non-contained in any subinterval S' of S (in "[,]") of depth $> l$ form a well-formed expression. (Similarly for interval I starting with "(".)

It is clear that this condition is equialent to the fact that the expression is well-formed.

Further it is clear that the algorithmus verifying this condition can be implemented on a b.p. P of size at most $O(n^5)$. (n - interval, n - the level of nesting, n - the subinterval, n - the counting of subintervals more deeply nested, n - for the sum of (,) running in time.)

Now , let us construct a gentle branching program Q computing f. Q starts with a full tree of depth 10 with tests on $x_1, ..., x_{10}$. Its leaves are sticked into two nodes v_1, v_2. v_1 is reached by all inputs with prefixes [()() or [(()). The other inputs reach v_2. At v_1 the result does not depend on the prefix; so at v_1 we start the program $P_1 = P_{x_1...x_{10}=[()()}$. At v_2 we start the program P. Q computes f and its size is at most $O(n^5)$.

We choose F the set of all inputs reaching v_1 and distribute it to v_1. We see that $|F| \geq 2^{n-9}$. We want verify that F is a regular set with deviation 0 and fluctuation 0. At v_1 we develop the tree T_F. In each branch there are two inputs which differ only on the first ten bits ([()() and [(())) and on the other bits are the same and moreover they have there only down-crosses or up-crosses. Hence for each $a \in F$ we have non-crosses on bits $1, 2, 5$ and double-crosses on bits $3, 4$. So, $D(a) = S(a) = D$ - the deviation $= 0$. $N(a) = N$ - the fluctuation $= 0$. Q.E.D.

□

In the last theorem we work with the pointer function [**?**] which is difficult for gentle branching programs. We perform an experiment that by replacing "$D \neq \emptyset$" from the definition of regular sets by only "$D(a) \neq \emptyset$" we obtain more powerful 'gentle' branching programs.

Let s and k be such that $ks^2 = n$ and $k \geq \log n$. Arrange the n variables $X = \{x_1, x_2, ..., x_n\}$ into a $k \times s^2$ matrix; split the i-th row ($1 \leq i \leq k$) into s blocks of size s each, and let ϵ_i be the OR of ANDs of varibes in these blocks. The pointer function is defined by: $f(X) = x_j$ where j is the number (between 1 and n), whose binary code is $(\epsilon_1, ..., \epsilon_k)$.

Theorem 5.2 *f is computable on a 1-'gentle' branching program of size $O(n^2)$ with 'deviation' 3 and fluctuation 0.*

Proof: Let us describe a program P. P starts by a full tree of depth 3 on variables $x_{1,1}, x_{1,2}, x_{1,3}$. The leaves of the tree are sticked into two output nodes. The leaves of branches with at least one zero on the three variables in question are sticked to v_1; the branch with $x_{1,1}x_{1,2}x_{1,3} = 111$ ends in v_2. At v_1, v_2 we start two identical copies of the program P_1. P_1 trivially evaluates the conjunctions in the blocks of the row in question, then P_1 knows the value of the disjunction, hence it knows a bit of the pointer vector. Then P_1 computes the next bit. After evaluating of the last bit of the pointer vector P_1 tests the pointed input bit.

It is clear that P computes f and that $|P| \leq O(n^2)$.

Let us distribute the set $F =_{df} (\{0,1\}^3 - \{1^3\}).\{0,1\}^{n-3}$ to the node v_1. Since the variables $x_4, ..., x_n$ are tested only after reaching v_1 each $a \in F$ has only down- or up-crosses on them. In branches where the pointed input bit differs from $x_{1,1}, x_{1,2}, x_{1,3}$ these three bits have double-crosses. If one of them is the pointed bit then the remaining two have double-crosses. There are no non-crossed bits.

The conclusion: f is computable on 1-'gentle' branching program of size $O(n^2)$ with 'deviation' 3 and fluctuation 0.

Q.E.D.

□

Conclusions. Taking into account the upper bounds from [**?**], [**?**] and the new lower bounds from the present report it seems that the notion of gentleness catch something from the real world like simplicity of functions or human-like computation.

Acknowledgments. I thank Stasys Jukna for the cooperation in the research concerning information flow in branching programs -as expressed in terms windows and gentle- in the last years.

Bibliography

[1] M. Ajtai - A Nonlinear Time Lower Bound for Boolean Branching Programs, ECCC Trier, Report No.26, 1999

[2] A. Andreev, J. Baskakov, A. Clementi, J. Rolim - Small Pseudo-Random Sets Yield Hard Functions: New Tight Explicit Lower Bounds for Branching Programs, Proc. of ICALP'99, Prague, Springer, pp.179 - 189.

[3] P. Beame, M. Saks, J. Thathachar, Time-Space Tradeoffs for Branching Programs, ECCC Trier, Report No. 53, 1998

[4] S. Jukna, S. Žák, On Branching Programs with Bounded Uncertainty, Proc. of ICALP'98, LNCS Springer, Berlin, 1998, pp.259-270

[5] K. Kriegel, S. Waack, Exponential Lower Bounds for Real-Time Branching Programs, ???, pp. 263-267

[6] S. Žák, Information in Computation Structures, Acta Polytechnica, Prague, 20 (IV.4), 1983, pp. 47 - 54

[7] S. Žák, An exponential lower bound for one-time-only branching programs, Proc. of MFCS'84, Lect. Notes in Comput. Sci., 176 (Springer 1984), 562-566.

[8] S. Žák, A subexponencial lower bound for branching programs restricted with regard to some semantic aspects, ECCC Report Nr. 50, 1997