Electronic Colloquium on Computational Complexity, Report No. 150 (2019)

A Logical Characteristic of Read-Once Branching Programs

Stanislav Žák*

Institute of Computer Science of the Czech Academy of Sciences,
P. O. Box 5, 182 07 Prague 8, Czech Republic, zakknin@seznam.cz

Abstract. We present a mathematical model of the intuitive notions such as the knowledge or the information arising at different stages of computations on branching programs (b.p.). The model has two appropriate properties:
i) The "knowledge" arising at a stage of computation in question is derivable from the "knowledge" arising at the previous stage according to the rules of the model and according to the local arrangement of the b.p.
ii) The model confirms the intuitively well-known fact that the knowledge arising at a node of a computation depends not only on it but in some cases also on a "mystery" information. (I. e. different computations reaching the same node may have different knowledge(s) arisen at it.)
We prove that with respect to our model no such information exists in read-once b.p.'s but on the other hand in b. p.'s which are not read-once such information must be present. The read-once property forms a frontier.
More concretely, we may see the instances of our models as a systems $S = (U, D)$ where U is a universe of knowledge and D are derivation rules. We say that a b.p. P is compatible with a system S iff along each computation in P S derives F ($false$) or T ($true$) at the end correctly according to the label of the reached sink. This key notion modifies the classic paradigm according to which the computational complexity is defined with respect to different classes of restricted b.p.'s (e.g. read-once b.p.'s, k-b.p.'s, b.p.'s computing in limited time etc.). Now, the restriction is defined by a subset of systems and only these programs are taken into account which are compatible with at least one of the chosen systems.
Further we may understand the sets U of knowledge(s) as a sets of admissible logical formulae. More rich sets U's imply the larger (= more free) restrictions on b.p.'s and consequently the smaller complexities of Boolean functions are detected. More rich logical equipment implies stronger computational effectiveness.

Key words: branching programs, computational complexity, logic

* S.Ž.'s research was supported by RVO: 67985807.

ISSN 1433-8092

1 Introduction

Let us follow a way starting with a simple intuitive question to the rich world of questions connecting logic, complexity and combinatorics.

1.1 An informal intuition transformed to a formal model

Before the period of 45 years a young student was confronted with the definition of Turing machine. Inspired by the intuition that at the starting configuration of a computation the machine in question knows probably nothing and at the final configuration the machine probably knows "YES" or "NO" he posed the naive question "And what the machine knows at the configurations in the middle of computation?".

In some way our paper answers this question in that sense that it presents a model developing knowledge along the computation step-by-step which has a reasonable properties and which is interesting for many reasons. The model is formulated in terms of branching programs (b.p.).

In the intermediate time a model for the more simple question "what the branching program knows about the contents of the input bits?" was developed. This model provided interesting results including a lower bound method for *general* branching programs, interesting restrictions on branching programs and the corresponding lower bounds [1],[2],[3],[4],[5].

Concretely an instance of our present model is a so-called deductive system which is a pair $S = (U, D)$ where U is a universe of knowledge (possibly a set of logic formulae) and D is a partial mapping from $P(U)$ to $P(U)$ which substitutes the deduction rules. For each branching program and for each its computation each deductive system assigns a subset of U to each node and to each edge of the computation. At the source the system assigns $\emptyset$ and to the next edges and nodes step-by-step it assigns different subsets of U depending i) on D, ii) on the subset assigned to the previous edge or node and iii) on the local arrangement. At sinks the derived subclass of U may (sometimes) contain F or T. In case that F (resp. T) is derived iff the label of the sink is 0 (resp. 1) we say that the system S is compatible with the program in question.

1.2 The read-once property forms a barrier against the mystery information

Our model confirms the intuitive evidence which is well-known to people familiar with branching programs. It is possible that the computations reaching the same node in b.p. have (at this node) different knowledge(s) assigned by the system. Hence the information is not defined only by the node, these computations must have some additional - mystery - information at this node. We prove that in read-once branching programs no such information exists and on the other hand in branching programs's which are not read-once such mystery information must be present. The read-once property forms a frontier.

1.3 A new paradigm of creating restrictions on branching programs

The usual way of studying the complexity of Boolean functions was to study this complexity with respect to the restricted classes of programs such as programs of constant width, read-once programs, programs computing within limited time and so on. The complexity of a function f is given by the size of the smallest program in the class computing f. Hence the more free restrictions (= larger classes of programs) provide the smaller complexities.

The new paradigm of creating restrictions uses the concept of compatibility. Let us have a class **S** of some systems. Let **P** be the class of all programs which are compatible with at least one system from **S**. **P** is our restricted class of programs and we are able to consider the complexity of functions with respect to it.

1.4 Logic versus complexity

This new paradigm gives us an exciting possibility to observe the links between logic and complexity. Suppose that U's are the sets of logic formulae of certain lengths which are build up on e.g. a number objects, on a number of predicates with certain arity and on a number of variables. (Each number characteristic depends on the number of input bits.) If we change the number characteristics up we obtain larger classes U's. Hence we obtain a larger class of restricted branching programs and therefore we obtain smaller (=not larger) complexities of Boolean functions. The possibility to use a richer logic equipment along each computations of b. p. make the computation more effective - it works with smaller (=not larger) demands on the sources. At the end of paper we introduce an example where a richer logic equipment implies a strictly smaller demands on sources.

Following this line we shall obtain a new insight into the world of complexity and moreover into the world of logic, and links between them. The dynamics of measuring complexity with respect to restrictions mentioned above will be more rich and of course more inspiring due to its logical foundations.

1.5 New horizons of a possible future research

We can create complexity classes of Boolean functions simply by defining a set L_P of some limits on the properties of branching programs such as size, time, shape, number of repeated tests etc. Newly we may also define a set L_S of properties of deductive logical system such that as the number of predicates and their arities, number of objects, length and forms of logical formulae, forms of deduction rules etc. The corresponding complexity class of Boolean functions are those ones computed by branching programs with limits from L_P which are compatible with at least one deductive system satisfying limits from L_S.

The simple fact holds that making the limits from L_P or from L_S or from both more free we obtain classes at least the same or larger. The classical hierarchy question arises about the minimal increase of limits which causes that classes are

strictly larger. For the limits from L_S this is a very new thing. The other question is so-called trade-off when a decrease in some limit(s) is saturated by an increase in other limit(s). How a decrease of the size or time can be saturated by reacher number of predicate or reacher forms of logical formulae? We may formulate a general hypothesis: "More logic, less complexity (= stronger effectiveness), and vice versa".

A new question is that one about the sensitivity on complexity for different types of Boolean functions when we vary the logical bounds. Are there Boolean functions which are very sensitive in complexity on the changes of some logical bounds but sensitive on changes on other one limits are only a little? Would it be reasonable to define classes of Boolean functions with respect to the sensitivity to the changes of different logical bounds?

Another question arises: given a set of Boolean functions (e.g. codes of graphs of some type) what logical equipment has the strongest impact on effectiveness of computation?

As an marginal note we can introduce a new way of computation of partial Boolean functions. Suppose we have a branching program P which we consider without the labels of its sinks and further we have a deductive system S non necessarily compatible with P. Let a be an input and s_a be the sink of P reached by a. If $F \in V_{S,P}(a, s_a)$ and $T \notin V_{S,P}(a, s_a)$ then $f_{S,P}(a) =_{df} 0$. If $T \in V_{S,P}(a, s_a)$ and $F \notin V_{S,P}(a, s_a)$ then $f_{S,P}(a) =_{df} 1$. Otherwise $f_{S,P}(a)$ is undefined. How is the relation between the impact by P and that by S on the final form of $f_{S,P}$?

We are able to formutate many questions connecting complexity, logic and combinatorics within the lines mentioned above and similar ones. And what about the corresponding answers?

2 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 the computation on a ($comp(a)$) we mean the sequence of nodes (and edges) 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. By the length of a computation we mean the number of its inner nodes.

For 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 at v (or shortly a and b diverge at v). Similarly for more than two inputs. If $comp(a)$ has a common part with a path p in P we say that a follows p (in this part). 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 its each computation is of the length at most $t(n)$.
A special case of b.p. with in-degree = 1 in each node (with exception of the source) is called decision tree. Another 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.
By a distribution we mean any mapping D of a subset 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. (Similarly, we can work with distribution to edges.)

Let v be a node of P. By the tree T_v unfolded in v according to P we mean the decision tree the branches of which are given by the paths in P starting at v and ending at sinks. Let A be a set of some (not necessarily all) inputs reaching v. By the tree $T_{v,A}$ unfolded in v according to P with respect to A we mean the decision tree which results from tree T_v after application of the following operations:

a) From T_v we omit all branches which are not followed by any input from A.

b) Each edge pointing to a node with out-degree = 1 (after a)) is repointed to its successor.

By the tree $T^{c)}_{v,A}$ we mean a tree which results from not only a),b) but also from the next operation c)

c) at each leaf l of $T_{v,A}$ reached by the set of inputs $B_l \subseteq A$, $|B_l| > 1$ a subtree T_l is added which totally splits B_l and which tests variables in some ordering ϕ.

Similarly we can define the trees $T_{e,A}, T^{c)}_{e,A}$ for any edge e.

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 [6].

3 Deductive systems

Definition 1. *By a deductive system we mean a pair* (U, D)

where U *is a set (a universe of knowledge) which must contain also the elements* F *and* T *and the elements* $o^0_1, ..., o^0_n, o^1_1, ..., o^1_n$.

D *is a partial mapping from* $P(U)$ *to* $P(U)$ *which satisfies the next conditions:*

i) If D *is defined on* $A \subseteq U$ *then* $A \subseteq D(A)$ *and moreover* D *is defined also on* $D(A)$ *and* $D(A) = D(D(A))$.

ii) If D *is defined on* A, B *subsets of* U *and* $A \subseteq B$ *then* $D(A) \subseteq D(B)$.

iii) If for an i $o^0_i \in A$ *and* $o^1_i \in A$ *(and* D *is defined on* A*) then* $D(A) = U$. *Similarly for* $F \in A$ *and* $T \in A$.

iv) For $i = 1, ..., n$ and $j = 0, 1$ and $A \subseteq U$, D is defined on A, if $o_i^j \notin A$ then $o_i^j \notin D(A)$ (except of case iii)).

Let P be a branching program, $S = (U, D)$ be a deductive system and $a \in \{0,1\}^n$ be an input. To each w - a node or an edge of $comp(a)$ - S assigns a set $V_{S,P}(a, w)$ which is a subset of U. The process of assigning starts in the source of program in question, $V_{S,P}(a, source_P) =_{df} \emptyset$.

Let a be an input and let v be a node of $comp(a)$ with two outgoing edges e_0, e_1 (labeled by $0, 1$, resp.) testing a bit i. Let $V_{S,P}(a, v)$ be given. Let moreover $e_0 \in comp(a)$. Then we define $V_{S,P}(a, e_0) = D(V_{S,P}(a, v) \cup \{o_i^0\})$ (we assume that D is defined). Similarly for e_1 and $\{o_i^1\}$.

Let us have an edge $e \in comp(a)$ ending in a node v. Let $V_{S,P}(a, e)$ be given. We define $V_{S,P}(a, v)$ as a subset of $V_{S,P}(a, e)$.

We proceed as follows. Let M_e be the set of all inputs reaching e and M_v be the set of all inputs reaching v. Let us take the trees T_{e,M_e} a T_{v,M_v}.

Let $A_{a,e}$ be the set of all "answers" along the branch of T_{e,M_e} followed by a, i.e. the set of all $o_i^{a_i}$ such that x_i is tested on this branch. Similarly we define $A_{a,v}$ as the set of all "answers" along the branch of T_{v,M_v} followed by a. Clearly, $A_{a,e} \subseteq A_{a,v}$ (since $M_e \subseteq M_v$).

$V_{S,P}(a, v)$ will be the minimal subset V of $V_{S,P}(a, e)$ such that for each $G \in \{F, T\}$ if $G \in D(V_{S,P}(a, e) \cup A_{a,e})$ then $G \in D(V \cup A_{a,v})$. It is easy to see that $D(V_{S,P}(a, e)$ itself fulfils this axiom, hence the existence of V is ensured. From the minimality of V it follows that $V \cap A_{a,v} = \emptyset$.

(We see that there is an ambiguity in the definitions of V. We did not eliminate the possibility of more than one minimal sets and we choose it arbitrarily. Though this ambiguity the definition and proof machinery works well.)

3.1 Compatible systems

Definition 2. *Let $S = (U, D)$ be a deductive system. We say that S is a nonconflict system iff for each input $a = a_1...a_n \in \{0,1\}^n$ the set $D(\{o_i^{a_i} | i = 1...n\})$ never contains both F, T.*

In the next definition we introduce the basic notion of our paper.

Definition 3. *Let P be a branching program and let S be a nonconflict deductive system. We say that P and S are (mutually) compatible if for each input a reaching the sink s_a in P T (F, resp.) is derived in $V_S(a, s_a)$ iff the label of s_a is 1 (0, resp.).*

We intuitively consider the compatible systems as entities which give us to understand how the programs compute (think) or how their inherent logics work.

4 Read-Once Branching Programs and the mystery information

Given a program P and a deductive system S we see that along any computation in P, S assigns the sets $V \subseteq U$ to each node and to each edge. In principle, at any w - a node or an edge - the sets V assigned to w may be different for other different computations also reaching w. In other words the memory arising at a node (or an edge) along a computation is not given by the reached node (or the reached edge) only but it is given also by some other mystery things. This is the key message of this section. Moreover we prove that in read-once programs there is no mystery information on one hand and on the other hand in branching programs which are not read-once the mystery information must be present. The read-once property forms a frontier.

Definition 4. *Let P be a program, w be a node or an edge of P, and T_w be the tree developed from w. Let $S = (U, D)$ be a deductive system, and V_1, V_2 be some subsets of U. Then we say that V_1, V_2 are T_w-equivalent iff for each $G \in \{F, T\}$ and for each branch B in T_w $G \in D(V_1 \cup A_B)$ iff $G \in D(V_2 \cup A_B)$ where A_B is the set of all answers along B.*

This formal property of "T_w-equivalence " stands as the formal opposite of the intuitive notion of equality of information.

Theorem 1. *Let P be a read-once branching program and S be any deductive system compatible with P. Let w be a node or an edge of P. Then for each inputs a, b reaching w sets $V_{S,P}(a, w), V_{S,P}(b, w)$ are T_w-equivalent.*

Proof. By topdown induction. Let w be the source of P. By definition for each input a, b $V_{S,P}(a, w) = \emptyset = V_{S,P}(b, w)$.

Let v be a node of P such that for all inputs a reaching v sets $V_{S,P}(a, v)$ are mutually T_v-equivalent. Let e_0, e_1 be the edges leaving v. Let a, b be inputs reaching e_0. We want to prove that the sets $V_S(a, e_0), V_S(b, e_0)$ are T_{e_0}-equivalent. Let B be any branch in T_{e_0} and $G \in \{F, T\}$. We see that

$G \in D(V_{S,P}(a, e_0) \cup A_B)$ iff

$G \in D(D(V_{S,P}(a, v) \cup \{o_i^0\}) \cup A_B)$ (where i is the bit tested at v) iff

$G \in D(V_{S,P}(a, v) \cup A_{B'})$ (where B' is branch in T_v consisting from e_0 followed by B) iff

$G \in D(V_{S,P}(b, v) \cup A_{B'})$ (since $V_{S,P}(a, v), V_{S,P}(b, v)$ are T_v-equivalent by induction) iff

$G \in D(V_{S,P}(b, e_0) \cup A_B)$.

(We have used the fact that $D(D(X \cup Y) \cup Z)$ is in between $D(X \cup Y \cup Z)$ and $D(D(X \cup Y \cup Z))$ which are equal. Similarly for $D(D(X) \cup Y \cup Z)$.

Similarly for e_1.

Let a, b be two inputs reaching v via edges e_a, e_b.

If $e_a = e_b$ then a, b are T_{e_a}-equivalent by induction and further a, b are T_v-equivalent since $T_v = T_{e_a}$ by the read-once property.

If $e_a \neq e_b$ let us take any branch B of T_v. By the read-once property there is an input a_B such that a_B follows a till v and then a_B follows B till the sink. F or T are in $D(V_{S,P}(a_B, v) \cup A_B)$ if 0 or 1 is the label of the sink of B (according to *Theorem 9 in Section 7*).

The same holds for a since a and a_B are T_{e_a}-equivalent (by induction).

The same arguments hold for b. We obtain that F or T is in $D(V_{S,P}(a, v) \cup A_B)$ and in $D(V_{S,P}(b, v) \cup A_B)$ according to the label 0 or 1 of the sink of B. Therefore a, b are T_v-equivalent.

□

Corollary 1. *Let P be a read-once branching program and P_1 be any its subprogram. Let S be any system compatible with P_1. Then for each w a node or an edge of P_1 and for each inputs a, b reaching w (in P_1) the sets $V_{S,P_1}(a, w)$ and $V_{S,P_1}(b, w)$ are T_w-equivalent in P_1.*

Proof. P_1 is a read-once branching program, too. □

Theorem 2. *Let P be a not-read-once branching program which is a minimal one. Then there are a system S and a subprogram P_1 compatible with S such that in P_1 there is a node or an edge w of P_1 and there are two inputs a and b both reaching w (in P_1) such that the sets $V_{S,P_1}(a, w)$ and $V_{S,P_1}(b, w)$ are not T_w-equivalent in P_1.*

Proof. Since P is a not-read-once program then there is a computation *comp*(a) which tests a variable x_i (with result $x_i = 0$ wlog) at least two times, on a node v_1 and then on v_2. Between v_1, v_2 there is no other test on x_i. Moreover wlog *comp*(a) ends in the sink with label 0.

Let b be an input such that *comp*(b) leaves v_2 by edge $x_i = 1$.

CASE 1. Assume that b reaches both v_1 and v_2.

In P on a-path or on b-path from v_1 to v_2 there is at least one other test (from minimality). Say, it is on *comp*(a) (wlog) in a node u_1 and it tests a variable j, $a_j = 1$.

Since v_2 is a test on i immediately next after v_1 it must be $i \neq j$.

Let P_1 be a (sub)program based on inner nodes v_1, u_1, v_2 and on three sinks reached by inputs a, b, c where $c_i = 0, c_j = 0$.

If d is an input then v_d denotes F or T according the value 0 or 1 of the label of the sink reached by *comp*(d).

Now, let us create an appropriate deductive system $S = (U, D)$. D is minimal such that:

$D(\{o_i^0\}) = \{o_i^0\}$, $D(\{o_i^1\}) = \{o_i^1, v_b\}$, $D(\{o_j^0\}) = \{o_j^0\}$, $D(\{o_j^1\}) = \{o_j^1\}$,
$D(\{o_i^0, o_j^1\}) = \{o_i^0, o_j^1, v_a\}$, $D(\{o_i^0, o_j^0\}) = \{o_i^0, o_j^0, v_c\}$

We see that S is a non-conflict system.

Let us introduce a notation: If some edge leads from a node u to a node v we denote it $e(u, v)$.

Let us verify that S is compatible with P_1.

Let us derive the sets $V_{S,P_1}(a, w)$ when w is going along *comp*(a) in P_1 (using the rules from Section 3).

We see that at the source $V_{S,P_1}(a, v_1) = \emptyset$. Then $V_{S,P_1}(a, e(v_1, u_1)) = \{o_i^0\}$, $V_{S,P_1}(a, u_1) = \emptyset$, $V_{S,P_1}(a, e(u_1, v_2)) = \{o_j^1\}$, $V_{S,P_1}(a, v_2) = \{o_j^1\}$ and $V_S(a, e(v_2, sink_a)) = \{o_i^0, o_j^1, v_a\}$. Hence $V_{S,P_1}(a, sink_a) = \{v_a\}$.

For the input b: $V_{S,P_1}(b, v_1) = \emptyset$, $V_{S,P_1}(b, e(v_1, v_2)) = \{o_i^1\}$, $V_{S,P_1}(b, v_2) = \emptyset$ and $V_{S,P_1}(b, e(v_2, sink)) = \{o_i^1, v_b\}$. Hence $V_{S,P_1}(sink_b) = \{v_b\}$.

For the input c: $V_{S,P_1}(c, v_1) = \emptyset$, $V_{S,P_1}(c, e(v_1, u_1)) = \{o_i^0\}$, $V_{S,P_1}(c, u_1) = \{o_i^0\}$, $V_{S,P_1}(c, e(u_1, sink_c)) = \{o_i^0, o_j^0, v_c\}$ and $V_{S,P_1}(c, sink_c) = \{v_c\}$.

We see that S derives the correct values on sinks. Therefore S is compatible with P_1.

Now, let $w = v_2$. We want to prove that $V_{S,P_1}(a, w) = \{o_j^1\}$ and $V_{S,P_1}(b, w) = \emptyset$ are not T_w-equivalent.

Let us take the branch in T_w consisting from edge e_0 outgoing w with test $i = 0$. We see that $D(V_{S,P_1}(a, w) \cup \{o_i^0\}) = \{o_i^0, o_j^1, v_a\}$ on one hand , and on the other hand $D(V_{S,P_1}(b, w) \cup \{o_i^0\}) = \{o_i^0\}$. Hence the sets $V_{S,P_1}(a, w)$ and $V_{S,P_1}(b, w)$ are not T_w-equivalent.

CASE 2. Taking into account CASE 1 $comp(b)$ must leave $comp(a)$ in a node v_0 (with test, say, $j = 0$) before v_1 and join $comp(a)$ again before (or in) v_2 and leaves v_2 by edge $x_i = 1$.

Further there is a computation on an input c which reach v_1 and leaves it by the edge $x_i = 1$. Let P_1 be defined on inner nodes v_0, v_1, v_2 and on sinks for $\{a, b, c\}$.

Now we define $S = (U, D)$ as follows:

$D(\{o_i^0, o_j^1\}) = \{o_i^0, o_j^1, v_a\}$, $D(\{o_i^1, o_j^0\}) = \{o_i^1, o_j^0, v_b\}$, $D(\{o_i^1, o_j^1\}) = \{o_i^1, o_j^1, v_c\}$
On other arguments D is *identity*, especially $D(\{o_i^0, o_j^0\}) = \{o_i^0, o_j^0\}$.

We see that S is nonconflict.

Compatibility: On a the desired v_a is assigned after v_1 and after v_2. The desired v_b for b is assigned after v_0 and after v_2. And the desired v_c for c is assigned after v_1.

Let $w = v_2$. Then $V_{S,P_1}(a, w) = \{o_j^1\}$ and $V_{S,P_1}(b, w) = \{o_j^0\}$. Now, let us take a branch of T_w consisting from the edge leaving w and ending in the sink with label 0. The sets $D(V_{S,P_1}(a, w) \cup \{o_i^0\}) = \{o_j^1, o_i^0, v_a\}$ and $D(V_{S,P_1}(b, w) \cup \{o_i^0\}) = \{o_i^0, o_j^0\}$ are not $\{F, T\}$-equivalent. Hence the sets $V_{S,P_1}(a, w)$ and $V_{S,P_1}(b, w)$ are not T_w-equivalent.

□

5 Complexity based on compatibility

Definition 5. *Let $S_1 = (U_1, D_1)$ and $S_2 = (U_2, D_2)$ be systems. We say that S_1 is a part of S_2, $S_1 \sqsubseteq S_2$ iff $U_1 \subseteq U_2$ and moreover for each $A \subseteq U_1$ if D_1 is defined on A then D_2 is also defined on A and $D_1(A) \subseteq D_2(A)$.*

Lemma 1. *Let P be a program and let S_1, S_2 be systems which are P-sound. If and S_1 is compatible with P and $S_1 \sqsubseteq S_2$ then also S_2 is compatible with P.*

Proof. Let a be an input with $f_P(a) = 0$. If S_1 is compatible with P then $D_1(V_{S_1}(a, source) \cup A_{a,source}) = D_1(A_{a,source})$ must contain F because the set of answers from the source to the sink is the maximum what is at the disposition for deriving F along $comp(a)$ and S_1 is compatible with P.

Further also $D_2(A_{a,source})$ must contain F since $S_1 \sqsubseteq S_2$.

During the derivation along $comp(a)$ from the source to the sink S_2 must preserve F. At the sink s_a the unique possibility is that $F \in V_{S_2}(a, sink_a)$. Hence S_2 is compatible with P.

□

Definition 6. *Let f be a Boolean function and S be a system. Then, by S-complexity of f we mean the size of the smallest branching program P computing f and compatible with S if such P exists (otherwise formally S-complexity equals ∞).*

Theorem 3. *Let f be a Boolean function and S_1, S_2 be systems. Let S_2 be P-sound for each P computing f compatible with S_1. If $S_1 \sqsubseteq S_2$ then S_2-complexity of f is not larger than S_1-complexity of f.*

Proof. It suffices to prove that each program P compatible with S_1 is compatible also with S_2. This follows from the lemma above. So, the minimum for S_2-complexity is taken over the same or larger set of programs than in case of S_1-complexity. □

The theorem confirms our intuitive idea that the branching programs which may compute in a more complicated way (i. e. using our terminology, which are compatible with richer deductive systems) can compute more effectively, i. e. within a smaller complexity bound. If the internal dialogue along each computation (i. e. node-edge-node) may use more rich means then it is more effective (= with smaller demands on memory, on complexity resp.). In the next section we demonstrate this fact convincingly. In our example, a small increase in the richness of deductive systems produces a dramatic drop in the need of the computation source (memory).

6 More logic - less complexity: An example

For our demonstration of the basic principle of complexity based on compatibility we use the simple classical parity function. We want to demonstrate the trade-off between the logical bounds on one hand and the memory bounds on the other hand. We shall construct a sequence $\{S_i\}_{i=1}^n$ of systems with increasing logical equipments, $S_i \sqsubseteq S_{i+1}$. We shall prove that the corresponding S_i-complexities of the parity function are decreasing. In informal words: More logic less complexity (and viceversa) or more logic more effectiveness.

For $i = 1...n$ we define $S_i =_{df} \{U_i, D_i\}$ where U_i contains the elements F, T and o_k^j, $k = 1, ..., n, j = 0, 1$ and moreover the elements which are formulae of

the type $Par(o)$. Par are predicates $odd_i, even_i$ applicable to the sets o of input bits of cardinality at least $n-i$.

D_i contains the deduction rules as follows:

Rule I). For any $\{o_{k_1}^{j_1}, ..., o_{k_{n-i}}^{j_{n-i}}\}$ where $k_1, ...k_{n-i}$ is a subsequence of $\{1, ..., n\}$ and $j_1, ...j_{n-i}$ are from $\{0,1\}$ D_i derives $Par(\{k_1, ..., k_{n-i}\})$ where $Par = odd_i$ or $Par = even_i$ according to the parity of the number of 1's in the chain $j_1, ..., j_{n-i}$.

Rule II). From $\{Par_L(A), o_j^k\}$ D derives $Par_R(A \cup \{j\})$ where A is a set of input bits, $0 < j < n+1$ and $j \notin A$, $k \in \{0,1\}$ Par_L is odd_i or $even_i$, and Par_R is odd_i or $even_i$ depending on Par_L and k in the obvious way which is given by the properties of the parity function.

Rule III). Moreover, in D_i there are two special rules
$odd_i(\{1, ..., n\})/T$ and $even_i(\{1, ..., n\})/F$.

Theorem 4. *For $i \in \{1, ..., n\}$, S_i-complexity of the parity function is at least 2^{n-i-1}.*

Proof. Let P be any branching program computing parity function which is compatible with S_i. We want to prove that $size(P)$ is at least 2^{n-i-1}, this will be sufficient. From the compatibility P and S_i it follows that at the sink of its computation each input has activated predicates F or T. Hence, during its computation each input has activated predicates odd_i or $even_i$ on the set $\{1, ..., n\}$ (cf. Rule III).
The unique way in which an input may have activated a parity predicate on a set of input bits of cardinality n is such that it has activated this predicate on the set of cardinality $n-1$ and used Rule II. Repeatedly till the cardinality $n-i$. For each input let us take into account the edge of its computation where the predicates odd_i or $even_i$ are activated on a set of elements of type o_j^k of size $n-i$ for the first time (cf. Rule I). This is the moment when the input in question has the natural window of length $n-i$ (see Lemma 4 in Section 8).
Now we distribute each input a to the edge of its computation where a has the natural window of length at least $n-i$ for the first time. The length of windows according to this distribution is of course of length at least $n-i$. According to Theorem 12 (Section 8) the number of classes of the distribution and hence the number of edges in P is at least 2^{n-i}. Hence, according to the fact that out-degree of nodes in P is at most 2 we have that $size(P)$ is at least 2^{n-i-1}. (Here we see how Theorem 12 works in practice.) □

Lemma 2. *For i, $i = 1, ..., n$, there is a program P which computes the parity function and which is compatible with system S_i such that $size(P) \leq 2^{n-i} + 2.(i+1)$.*

Proof. P starts as a decision tree of depth $n-i$ on the first $n-i$ bits. On the remaining i levels P is of width 2. The nodes are arranged in two columns, one column represents the value "even" and the other represents the value "odd". The zero-edge outgoing any node always preserves the column while the one-edge always changes the column.

We see that P indeed computes the parity function and that its size is below the desired bound. It remains to prove that P is compatible with S_i. On the level of leaves of the initial tree of depth $n-i$ in question we have for the first time derived the formula $Par(A)$ where A is the set of cardinality $n-i$ and Par is odd_i or $even_i$. Below in two column chain the cardinality of A is increasing step by step, by one in each step (cf. Rule II).

On the level of sinks the cardinality of A is n, and, therefore, correct F, T are derived here - cf. Rule III. So, P is compatible with S_i. □

Comment. The upper bound for programs compatible with S_{i+1} is at most $2^{n-i-1}+2i+4$. On the other hand the lower bound for programs compatible with S_{i-1} is at least 2^{n-i} which is in general more than the mentioned upper bound. Hence the decrease of the logical equipment from $S_i + 1$ to S_{i-1} considerably decreases the computational power.

7 Some more details on programs and systems

In general the deductive system can derive predicates F, T at the end of computations which do not correspond to the labels $1, 0$ of the reached sinks. This is the reason for the next definition.

Definition 7. *Let P be a program and S be a deductive system. We say that S is P-sound iff for each sink s of P with label 0 (1, resp.) S never derives T (F, resp.) for the computation on any input reaching s.*

Definition 8. *Let f be a Boolean function and S be a system. We say that S is f-sound iff S is P-sound for each P computing f.*

Theorem 5. *Let f be a Boolean function and let R be a full decision tree of depth n computing f. Let S be an R-sound system. Then S is an f-sound system.*

Proof. Let P be a program computing f and let a be an input reaching a sink s_a. The set $V_S(a, s_a)$ derived in P is a subset of these one derived in R along the branch of length n followed by a which does not contain any wrong F, T.

□

Theorem 6. *Let R be a decision tree. Then there is a system S compatible with R.*

Proof. Given R we define $S =_{df} (U, D)$ as follows. $U =_{df} \{o_i^j | i = 1 \ldots n, j = 0, 1\} \cup \{F, T\}$ and D is such that for each $A \subset U$ if A is the set of all answers along any whole branch b of R or large then $D(A) = A \cup \{F\}$ or $D(A) = A \cup \{T\}$ according to the label of the sink of b, and $D(A) = A$ otherwise. (And for A containing for some i both o_i^0 and o_i^1 D is undefined.)

We see that S is a nonconflict system. Moreover for each branch b of R and for each input a following b at the sink s of b $F \in V_S(a, s)$ or $T \in V_S(a, s)$ according to the label 0 or 1 of s. Hence S is compatible with R.

□

Corollary 2. *Let f be a boolean function. Then there is a branching program P computing f and a system S compatible with P.*

Proof. As P we may choose any decision tree for f.
□

Theorem 7. *Let T_0, T_1 be full binary decision trees of depth n computing the same function f. Let S be a system compatible with T_0. Then S is compatible with T_1, too.*

Proof. Let a be an input and $b_{0,a}$, $b_{1,a}$ be the branches in T_0, T_1 followed by a. Suppose that $f(a) = 0$. On the last edge $e_{p,0}$ of $b_{0,a}$ it holds $F \in D(\{o_1^{a_1}, ..., o_n^{a_n}\}) \subseteq V_{T_0,S}(a, e_{p,0})$
and $T \notin D(\{o_1^{a_1}, ..., o_n^{a_n}\}) \subseteq V_{T_0,S}(a, e_{p,0})$. ($S$ is nonconflict.).
On the last edge $e_{p,1}$ of $b_{1,a}$ it holds the same. Hence also on T_1 S derives F for a. S is compatible with T_1.
□

Theorem 8. *Let S be a system compatible with program P. Then S is f_P-sound.*

Proof. We want to prove that for any branching program Q computing f_P S is Q-sound. We see that S is compatible with T_P developed till the depth n. Then S is also compatible with T_Q developed till the depth n according to the theorem above. Then S is Q-sound since S deriving a false value on Q would derive a false value on T_Q, too.

□

Theorem 9. *Let P be a branching program and S be a system compatible with P. Let a be an input, w be a node or an edge, $w \in comp(a)$ and let M_w be the set of all inputs reaching w. Let $B_{a,w}$ be the branch in T_{w,M_w} followed by a, and $A_{a,w}$ be the set of all answers along $B_{a,w}$.*
Then $D(V_S(a, w) \cup A_{a,w})$ contains right F or T (only).

Proof. By induction from the source of P to sinks.
I. If w is the source then $V_S(a, source) = \emptyset$ and $A_{a,source}$ is the set of all answers till the leaf.
Since S is a nonconflict system in $D(V_S(a, w) \cup A_{a,w}) = D(A_{a,w})$ there is at most one element from F, T.
On the other hand along $comp(a)$ S derives correct F or T and $A_{a,w}$ is the maximum what S can have at its disposition. Therefore $D(V_S(a, w) \cup A_{a,w})$ contains right F or T (only).
II. a) Let w be an edge out-going a node v with test on i. Let by induction $D(V_S(a, v) \cup A_{a,v})$ contains right F or T (only).
$D(V_S(a, w) \cup A_{a,w}) = D(D(V_S(a, v) \cup \{o_i^{a_i}\}) \cup (A_{a,v} \setminus \{o_i^{a_i}\})) = D(V_S(a, v) \cup A_{a,v})$. Hence also $D(V_S(a, w) \cup A_{a,w})$ contains right F or T (only).
II. b) Let w be a node in $comp(a)$ and let e be the edge of $comp(a)$ ending in w. By induction $D(V_S(a, e) \cup A_{a,e})$ contains right F or T (only).

According to the definition $V_S(a,w)$ is a minimal subset V of $V_S(a,e)$ such that if F (T, resp.) is in $D(V_S(a,e) \cup A_{a,e})$ then F (T, resp.) is in $D(V \cup A_{a,w})$. From compatibility of S (especially from fact that S is nonconflict) it follows that in $D(V \cup A_{a,w})$ there is exactly one of the elements F, T.

□

Theorem 10. *Let S be a system compatible with a program P. Then S is compatible also with the tree T_P developed till the depth n.*

Proof. Since S is a nonconflict system S assigns at most one value F, T to each branch of T_P. On the other hand on each branch of T_P defined by an input a since T_P is developed till the depth n S derives at least the same or more than along nodes and edges of $comp(a)$. Hence S must derive the right F or T at the end of the branch in question. Hence S is compatible with T_P.

□

8 Systems and windows

We introduce a definition which will be useful for our lower bound proof.

Definition 9. *Let P be a branching program, v be its node or its edge. Let A be a set of some (not necessarily all) inputs reaching v. From v we totally develop the tree $T^c_{v,A}$ (see Section 2).*

For each $a \in A$ we define the window $w(a,v,A) \in \{0,1,+\}$ on a at v with respect to A in such a way that $w(a,v,A)_i = +$ if and only if in $T^{c)}_{v,A}$ 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.

A very similar definition is widely commented in [8].

We use the following lemma for the proof of the next theorem.

Lemma 3. *[5] 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, $l \geq log_2\ S - log_2\ r$.*

Theorem 11. *Let P be a branching program and A be a set of inputs of length n distributed in edges $e_1, ...e_r$ of P. Let $A_1,\ ...\ A_r$ be the classes of this distribution. Then, $log_2\ (2.size\ of\ P) \geq log_2\ r \geq log_2\ |A| - n + avelw$ where $avelw$ is the average length of windows on inputs from A each at corresponding e_i with respect to A_i, $i = 1,\ ...\ r$.*

Proof. Let us take the classes $A_1, ..., A_r$ distributed to edges $e_1, ..., e_r$. For each i, $1 \leq i \leq r$, in e_i let us totally develop the tree $T^c_{e_i,A_i}$ according to P. We obtain r binary trees and we apply lemma above. Let l, S be as in lemma. We have $log_2(2.size\ of\ P) \geq log_2\ r \geq log_2\ S - l \geq log_2\ |A| - l \geq log_2\ |A| - n + avelw$. □

Lemma 4. *Let P be a program, S be a system and a be an input. For each node or edge w of $comp(a)$ and for each bit i it holds that if $o_i^{a_i} \in V(a,w)$ then the bit i is non-crossed in the natural window $w(a,w)$ on a at w (symbolically $i \in ncw(a,w)$).*

Proof. By induction. Let a be an input. At the source we have $V_S(a, source) = \emptyset$.

Let v be a node of $comp(a)$ such that for each $i = 1, ..., n$ if $o_i^{a_i} \in V_S(a,v)$ then $i \in ncw(a,v)$. Let i_0 be the bit tested at v. On the outgoing edge followed by a the set $V_S(a,v)$ is enlarged only by one element of the type o_i^j, concretely by $o_{i_0}^{a_{i_0}}$. This enlargement is covered by enlargement of $w(a,v)$ about the non-crossed bit i_0.

Let e be an edge of $comp(a)$ ending in node v such that for each $i = 1, ..., n$ if $o_i^{a_i} \in V_S(a,e)$ then $i \in ncw(a,e)$.

We want to prove that for $i = 1, ..., n$ if $o_i^{a_i} \in V_S(a,v)$ then $i \in ncw(a,v)$.

If $o_i^{a_i} \in V_S(a,v)$ then $o_i^{a_i} \in V(a,e)$ and $i \in ncw(a,e)$. Moreover, since $o_i^{a_i} \in V_S(a,v)$ it holds that $o_i^{a_i} \notin A_{a,v}$ where $A_{a,v}$ is the set of all answers along the branch of T_{v,M_v} followed by a till the sink . We see that this branch has no test on i and therefore i remains a non-crossed bit, $i \in ncw(a,v)$.

□

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.

References

1. Žák, S.: Information in Computation Structures. Acta polytechnica. Vol. 20, no. 4 (1983), pp. 47-54. ISSN 1210-2709
2. Žák, S.: A Subexponential Lower Bound for Branching Programs Restricted with Regard to Some Semantic Aspects. Electronic Colloquium on Computational Complexity. Report Series 1997. ECCC TR97-50. Trier, 1997, http://www.eccc.uni-trier.de/report/1997/050
3. Jukna, S., Žák, S.: On Branching Programs with Bounded Uncertainty. Automata, Languages and Programming. Proceedings. Berlin : Springer, 1998 - (Larsen, K.; Skyum, S.; Winskel, G.), pp. 259-270 ISBN 3-540-64781-3. - (Lecture Notes in Computer Science. 1443). [ICALP'98 International Colloquium /25./. Aalborg (DK), 13.07.1998-17.07.1998.
4. Jukna, S., Žák, S.: Some Notes on the Information Flow in Read-Once Branching Programs. SOFSEM'2000: Theory and Practice of Informatics. Berlin : Springer, 2000 - (Hlav, V.; Jeffery, K.; Wiedermann, J.), pp. 356-364 ISBN 3-540-41348-0. ISSN 0302-9743. - (Lecture Notes in Computer Science. 1963).
5. Jukna, S., Žák, S.: On Uncertainty versus Size in Branching Programs. Theoretical Computer Science. 290 (2003), pp. 1851-1867.
6. Wegener, I.: Branching Programs and Binary Decisions Diagrams, SIAM Monographs on Discrete Mathematics and Applications, pp. 408, 2000.
7. Žák, S.: A Lower Bound Method for Branching Programs and Its Application. Prague : ICS AS CR, 2012. 19 pp., Technical Report, V-1171
8. Žák, S.: Inherent Logic and Complexity. Electronic Colloquium on Computational Complexity, Report No. 29 (2015).

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il