Using Relevance Queries for Identification of Read-Once Functions

Dmitry V. Chistikov

Faculty of Computational Mathematics and Cybernetics
Moscow State University, Russia
dch@cs.msu.ru

Abstract. A Boolean function is called read-once if it can be expressed by a formula over $\{\wedge, \vee, \neg\}$ where no variable appears more than once. The problem of identifying an unknown read-once function f depending on a known set of variables $x_1, \ldots, x_n$ by making queries is considered. Algorithms are allowed to perform standard membership queries and queries of two special types, allowing to reveal the relevance of variables to projections of f. Two exact identification algorithms are developed: one makes $O(n^2)$ yes—no queries, and the other makes $O(n \log^2 n)$ queries with logarithmically long answers. Information-theoretic lower bound on the number of bits transferred from oracles to identification algorithms in the worst case is $\Omega(n \log n)$.

Keywords: query learning, exact identification, read-once Boolean function, relevant variable.

1 Introduction

We consider query complexity of exact identification problem for read-once Boolean functions in a non-standard learning model. A function is called *read-once* if it can be expressed by a formula over $\{\wedge, \vee, \neg\}$ where no variable appears more than once (a read-once formula, sometimes referred to as μ-formula). Our algorithms are allowed to perform standard membership queries (MQ), which reveal the value of the unknown (target) function on a given input vector, and queries of two special types. The latter answer the following questions about projections of the target function f: given a projection f_p induced by a partial assignment of variables p,

(RQ) is a variable x_i relevant to f_p?
(CRQ) how many relevant variables does f_p have?

Here the standard definition of relevance for Boolean functions is used — a variable x_i is called *relevant* if there exists an input vector such that a change in the value of x_i leads to the change in the function's output. Queries of the first type (denoted RQ for *relevance queries*) take arguments p and x_i, and queries of the second type (denoted CRQ for *counting relevance queries*), take only one

argument p. It is easily seen that queries of each type can be simulated with queries of the other type.

Our main results are as follows. We develop two algorithms (one for each type of relevance queries) running in polynomial time and identifying an unknown n-variable read-once function. The first algorithm makes $O(n^2)$ membership and relevance queries (MQ and RQ), and the second algorithm makes $O(n \log^2 n)$ membership and counting relevance queries (MQ and CRQ). For each n, both algorithms can be expressed by deterministic decision trees. The corresponding information-theoretic lower bounds on query complexity (the number of queries performed in the worst case) are $\Omega(n \log n)$ and $\Omega(n)$, so the second algorithm is suboptimal only by a factor of $O(\log^2 n)$. If we count the number of bits received from oracles, which answer the queries, instead of the total number of queries, the second algorithm achieves $O(n \log^3 n)$ and the lower bound is $\Omega(n \log n)$.

One can think of several reasons for the study of learning problems in this model. First of all, relevance queries seem natural to the process of learning. Imagine a student learning a rule which assigns one of two labels to each conceivable combination of features characterizing possible situations (examples). In this interpretation, membership queries are requests for a correct classification of a particular example (combination of features): "If these features are present and those are not, how do I classify the example?" Relevance queries are questions of the form: "Suppose that the presence of several features is known; can this feature affect the correct decision?" Since the teacher (the supervisor) knows how to apply the rule, he or she can answer the student's question promptly. For counting queries, however, it appears more natural to request the whole list of relevant features instead of their number.

The notion of relevance plays a major role in the study of read-once Boolean functions. One can observe, for instance, that all read-once functions f having at least two relevant variables possess the following property: for each relevant x_i there exist a value $\sigma \in \{0, 1\}$ and another variable x_j that is relevant to f but not to its projection $f_{x_i \leftarrow \sigma}$ obtained by substituting σ for x_i. In 1963, Subbotovskaya proved a variant of the converse: a function f is read-once if and only if all its projections f_p (including f itself) possess this property [10]. The list of all minimal non-read-once projections was obtained by Stetsenko [9], who built upon the criterion of Subbotovskaya.

For various learning problems involving read-once functions, the notion of relevance also turns out to be of crucial importance. Consider generalized read-once formulae, in which function symbols (gates) are taken from an arbitrary set of Boolean functions, called a basis (standard read-once functions are then read-once over the basis $\{\wedge, \vee, \overline{\ }\}$). It turns out that for any finite basis the problem of distinguishing an individual (generalized) read-once function from all other (generalized) read-once functions over the same basis can be solved by checking its value on a polynomial number of input vectors, if the set of all relevant variables is known in advance [12,5]. If, however, this information is not available a priori, then the complexity of the problem is exponential: for

instance, all 2^n input vectors must be tested to distinguish $f(x_1, \ldots, x_n) \equiv 0$ from all read-once conjunctions of literals.

For exact identification problems, similar results are known [14,13]. Suppose that, in addition to standard membership queries (requests for the value of the unknown function on given input vectors), learning algorithms are allowed to ask whether a given projection has at least one irrelevant variable. For the standard basis $\{\wedge, \vee, \overline{\ }\}$, the problem of exact identification with these queries can be solved polynomially if the set of relevant variables is known a priori, otherwise the task requires an exponential number of queries in the worst case.

Exact identification problem for read-once Boolean functions has been studied since 1984, when Valiant described an algorithm performing the task using a polynomial number of queries of three types [11]. Angluin, Hellerstein and Karpinski [3] developed an algorithm using queries of only one of these three types (namely, relevant possibility queries, which answer questions of type, "Are literals x_i^σ and x_j^τ contained in a single minterm of f?") and an algorithm using membership and equivalence queries (such a query basically asks if f can be expressed by a given read-once formula; we also refer to the paper [2] for the more general original definition). Both algorithms run in polynomial time; the first algorithm makes $O(n^2)$ relevant possibility queries, and the second algorithm makes $O(n^3)$ membership and n equivalence queries (for a particular case of monotone read-once functions just $O(n^2)$ membership queries are sufficient, and equivalence queries are not needed at all).

Note that for the standard basis $\{\wedge, \vee, \overline{\ }\}$ considered in this paper, a read-once function $f(x_1, \ldots, x_n)$ has an irrelevant variable x_i if and only if its number of *ones* (true points, i. e., vectors $\alpha \in \{0,1\}^n$ such that $f(\alpha) = 1$) is even (see, e. g., [14]). This means that the question of whether f has at least one irrelevant variable is equivalent to the request for the parity of the arithmetic sum $S(f)$ of $f(\alpha_1, \ldots, \alpha_n)$ over all 2^n vectors $(\alpha_1, \ldots, \alpha_n)$. Furthermore, the number of variables not relevant to f is equal to the largest number k such that 2^k divides $S(f)$.

2 Preliminaries

Let us first define a *read-once formula* over $\{\wedge, \vee, \overline{\ }\}$. If x is a Boolean variable, then x is also a read-once formula. If $\mathcal{F}_1$ and $\mathcal{F}_2$ are read-once formulae, then a formal expression $\overline{\mathcal{F}}_1$ is also a read-once formula, and so are $(\mathcal{F}_1 \wedge \mathcal{F}_2)$ and $(\mathcal{F}_1 \vee \mathcal{F}_2)$ whenever sets of variables that occur in $\mathcal{F}_1$ and $\mathcal{F}_2$ are disjoint. Each (well-formed) formula $\mathcal{F}$ expresses a Boolean function f, defined in the usual way. A function f is called a *read-once function* if it can be expressed by a read-once formula. By convention we shall usually assume that Boolean constants 0 and 1 are also read-once functions expressed by corresponding zero-variable formulae.

Literals of a variable x are $x^1 = x$ and $x^0 = \overline{x}$. Symbols $\wedge$ and $\vee$ are said to be *dual* to each other, and so are constants 0 and 1.

The structure of read-once formulae can be represented by rooted trees. Such trees are, in fact, Boolean circuits over the basis of negation and arbitrary fan-in conjunction and disjunction, and we shall depict them with leaves on top and root at the bottom. More formally, consider a rooted tree T satisfying the following conditions:

1) any node labeled with 0 or 1 must be the only node in T;
2) leaves are labeled with literals of different variables;
3) non-leaf nodes are labeled with symbols $\wedge$ and $\vee$ and have arbitrary indegree $s \geq 2$;
4) adjacent nodes are labeled with different symbols.

Each read-once function f can be represented by such a tree. This representation can be shown to be unique [6,7], so we usually talk about *the tree* of a read-once function.

In any rooted tree T, the *parent* (direct ancestor) of a non-root node v is denoted by $\mathrm{anc}(v)$. A subtree T_v contains v as its root and all descendants of v in T. We write $w \in T_v$ if w is a node contained in the subtree T_v. We shall often identify a leaf u labeled with a literal of a variable x_i with this variable itself, and by doing so, in a sense, not distinguish between a variable and its negation. Sets of leaves and sets of variables will therefore be usually used as synonyms.

By $l(v)$ we denote the number of leaves contained in T_v, i. e., the number of leaves that are descendants of v (if v is a leaf itself, we put $l(v) = 1$). Non-leaf nodes are also referred to as *internal* nodes, and direct descendants of a node are called its *children*. For an internal node v, by $\lambda(v)$ we shall sometimes denote the number of leaves that are children of v.

A variable x_i of a Boolean function f is called *relevant* if there exist two input vectors α and α' disagreeing only on x_i (in ith position) such that $f(\alpha) \neq f(\alpha')$. Relevant variables are sometimes called essential, and all other variables are called *irrelevant*, or fictitious. In this paper, by $R(f)$ we denote the set of all variables relevant to f, and by $r(f)$ the cardinality of $R(f)$.

If f is a Boolean function and p is a set of substitutions of the form $x_i \leftarrow g^i$, where all g^i are Boolean functions, by f_p we denote a function obtained from f by substituting functions g^i for inputs x_i. To avoid unnecessary complications, for each substitution $x_i \leftarrow g^i$ it is always ensured that all x_i are different and no x_i is relevant to any of g^j for $j \neq i$. If all substitutions in p have the form $x_i \leftarrow \sigma_i$, where $\sigma_i \in \{0, 1\}$, then p is called a *partial assignment*, and the function f_p is a *projection* of f induced by p. Subscripts at function symbols will usually be interpreted as partial assignments, and sequences will be indexed by superscripts, as in $f^n, \ldots, f^1$.

In settings of the problems considered in this paper, the task is exact identification of an unknown read-once function f of a known set of variables $\{x_1, \ldots, x_n\}$. The relevance of these variables is not known a priori. In both settings, learning (identification) algorithms are allowed to perform *membership queries*, i. e., to request the value of f on arbitrary input vectors α picked by algorithms.

In this paper, we are primarily interested in *query complexity* (the number of queries performed by the algorithms in the worst case) rather than time

complexity (maximum running time) of our algorithms. One can easily check that both suggested algorithms run in polynomial time and do not involve any brute-force search subroutines.

In order to obtain lower bounds on the complexity of our identification problems, observe that the logarithm of the number of read-once functions of variables $x_1, \ldots, x_n$ is $\Theta(n \log n)$ (see, e. g., [8]). Suppose that only yes—no queries (for instance, membership and relevance ones) are allowed. Then every algorithm identifying an unknown read-once function of these variables must perform at least $\Omega(n \log n)$ queries in the worst case, since the algorithm's structure can be represented by an oriented (decision) tree. Now suppose that answers to the queries can have logarithmic length (this is the case for counting relevance queries); then this bound is relaxed to $\Omega(n)$ (for the number of bits received as answers, the former bound still holds). Also note that a well-known statement on the average depth of a leaf in a tree (see, e. g., [1, section 8.6]) shows that these bounds also hold for the average complexity.

For the sake of simplicity, we shall assume in the description of our algorithms that all n input variables are relevant to the unknown function f. Since the identification of the set of all variables relevant to f can be performed with either n relevance queries (RQ) or with $n+1$ counting relevance queries (CRQ), this does not affect the upper bounds of $O(n^2)$ and $O(n \log^2 n)$ on the total number of queries performed by our algorithms.

3 Relevance queries: quadratic algorithm

In the setting considered in this section, *relevance queries* are available. Each relevance query supplies a partial assignment p and a variable x_i. The query returns 1 if $x_i \in R(f_p)$, that is, if x_i is relevant to f_p, and 0 otherwise.

Recall that our goal is exact identification of an unknown read-once function f. As explained above, we can safely assume from the start that all input variables are relevant to f. We shall present an algorithm that builds upon the following fact (see, e. g., [10]):

Claim 1. *For any read-once function f with $r(f) \geq 2$, for each variable x relevant to f there exists a unique $\sigma \in \{0, 1\}$ such that $R(f_{x \leftarrow \sigma}) = R(f) \setminus \{x\}$.*

Claim 1 gives an overall scheme of the future algorithm: construct a sequence of projections $f^n, f^{n-1}, \ldots, f^1$ such that $f^n = f$, each f^k is a projection of f^{k+1} and has exactly k relevant variables (here $k = n-1, \ldots, 1$). In other words, the algorithm will construct a sequence of variables $x_n, \ldots, x_1$ such that $R(f^k) = \{x_1, \ldots, x_k\}$ and each f^{k-1} is obtained from f^k by substituting some Boolean constant α_k for the variable x_k. This construction can be represented by the following diagram:

$$f^n \xrightarrow{x_n \leftarrow \alpha_n} f^{n-1} \xrightarrow{x_{n-1} \leftarrow \alpha_{n-1}} f^{n-2} \longrightarrow \cdots \longrightarrow f^2 \xrightarrow{x_2 \leftarrow \alpha_2} f^1.$$

The projection f^1 that depends on x_1 is either x_1 or $\overline{x}_1$, so it can be identified with a single membership query. The main issue we need to address is that of

going in the backward direction, i. e., reconstructing f^k given its projection f^{k-1}. Our plan is to choose the variable x_k in such a way that this reconstruction can be carried out without additional effort, that is, without making any queries at all.

Let us introduce the following notation. For a function g and variables x and y relevant to g, we write $x \trianglerighteq y$ iff there exists $\tau \in \{0, 1\}$ such that y is not relevant to $g_{x \leftarrow \tau}$. Observe that $x \trianglerighteq x$ for all variables x, and put $x \triangleright y$ iff $x \trianglerighteq y$ and $x \neq y$. Now note that the original (reflexive) relation $\trianglerighteq$ is transitive [10]:

Claim 2. *For any Boolean function f and any variables x, y, z relevant to f, if $x \trianglerighteq y$ and $y \trianglerighteq z$, then $x \trianglerighteq z$.*

For read-once functions with two or more relevant variables, it follows that a cycle of length two can be found in the graph of $\trianglerighteq$ [10]:

Claim 3. *For any read-once function f with $r(f) \geq 2$, there exist two variables x, y relevant to f such that $x \triangleright y$ and $y \triangleright x$.*

Now return to a projection f^k of the unknown function f. Note that once a pair x, y such that $x \triangleright y$ and $y \triangleright x$ for f^k has been found, we are basically done. Indeed, suppose that σ and τ are chosen in such a way that y is not relevant to $f^k_{x \leftarrow \sigma}$ and x is not relevant to $f^k_{y \leftarrow \tau}$. Denote by f^{k-1} the projection $f^k_{x \leftarrow \overline{\sigma}}$, and observe that by Claim 1 we have $R(f^{k-1}) = R(f^k) \setminus \{x\}$. From the definition of a relevant variable we deduce that $f^k_{x \leftarrow \sigma, y \leftarrow \tau} = f^k_{x \leftarrow \overline{\sigma}, y \leftarrow \tau} = f^k_{x \leftarrow \sigma, y \leftarrow \overline{\tau}}$. Since $f^{k-1} = f^k_{x \leftarrow \overline{\sigma}}$, it follows that $f^k = f^{k-1}_{y \leftarrow (x^\sigma \vee y^\tau)^\tau}$.

Therefore, the only remaining issue is that of finding an appropriate pair x, y for each (k-variable) function f^k. An exhaustive search would use $\Theta(k^2)$ queries in the worst case, but our algorithm will use Claim 2 and make only $O(k)$ queries on this step. Details are provided further in this section.

The ideas presented above are incorporated in the procedure of Algorithm $\mathcal{A}_1$ outlined in Fig. 1. In the description of steps 2(b)i and 2(b)ii, the equalities $x_k = x$, $y_k = y$ and $x = y$ are not assignments, but notation: $x_k = x$, for instance, means that the *name* x_k will henceforth represent the variable previously referred to as x. The same holds for the equalities $f^n = f$ and $f = f^n$ in 1 and 5. The equality in 2c simply fixes the value of the variable x_k for further queries during the subsequent iterations of 2 and 3. Finally, the equality in the description of step 4 indicates how to obtain a read-once formula for f^k from a known read-once formula for f^{k-1}. We also remark that for each iteration of step 2, the set $R(f^k) \setminus \{x\}$ of possible values for y is evaluated only once in 2b, regardless of possible changes in the meaning of x.

Theorem 1. *The algorithm $\mathcal{A}_1$ correctly identifies any read-once function f and performs one membership query and $O(n^2)$ relevance queries, where n is the number of input variables of f.*

Proof. First estimate the number of performed queries. Observe that two relevance queries are sufficient to check the relation $x \triangleright y$ for any fixed projection

Fig. 1. Algorithm $\mathcal{A}_1$: membership and relevance queries

1. Identify all n relevant variables. Put $f^n = f$.
2. For $k = n, \ldots, 2$:
   (a) Take any $x \in R(f^k)$.
   (b) For each $y \in R(f^k) \setminus \{x\}$:
      i. For f^k, if $x \rhd y$ and $y \rhd x$, denote $x_k = x$, $y_k = y$, choose σ_k and τ_k such that $y_k \notin R(f^k_{x_k \leftarrow \sigma_k})$ and $x_k \notin R(f^k_{y_k \leftarrow \tau_k})$, and go to step 2c.
      ii. For f^k, if $x \rhd y$ and $y \not\rhd x$, put $x = y$.
   (c) Obtain $f^{k-1} = f^k_{x_k \leftarrow \overline{\sigma}_k}$.
3. Identify the projection f^1 depending on the remaining variable.
4. For $k = 2, \ldots, n$: reconstruct $f^k = f^{k-1}_{y_k \leftarrow (x_k^{\sigma_k} \vee y_k^{\tau_k})^{\tau_k}}$.
5. Terminate with $f = f^n$.

f^k and a given pair of relevant variables x, y. It follows that the algorithm makes $O(n^2)$ relevance queries, and a membership query is only needed for step 3.

We now prove the correctness of the algorithm, which clearly requires special consideration. We already know that the choice of x_k, y_k on step 2(b)i guarantees the correct reconstruction on step 4. What we need to show is that the loop of step 2b always finds an appropriate pair x_k, y_k and terminates on 2(b)i.

To prove this, assume the converse. Let $y_1, \ldots, y_{k-1}$ be the variables tested on step 2b. Denote by y_0 the initial x, and by $y_{i_0}, y_{i_1}, \ldots, y_{i_s}$ all the variables represented by the name x during different iterations of the loop (here $i_0 = 0$). It follows that $y_{i_0} \rhd y_{i_1} \rhd \ldots \rhd y_{i_s}$ and $y_{i_s} \not\rhd y_j$ for all $j < i_s$ (by Claim 2). By Claim 1, there exists an index $j > i_s$ such that $y_{i_s} \rhd y_j$, which is a contradiction. This concludes the proof.

Remark. It is worth pointing out that the only membership query made by the algorithm is used to decide if the obtained projection f^1 is equal to x_1 or to $\overline{x}_1$ (here we assume $R(f) = \{x_1, \ldots, x_n\}$). Since the formula for f is then obtained from the formula for f^1 by substitutions of the form $y_k \leftarrow (x_k^{\sigma_k} \vee y_k^{\tau_k})^{\tau_k}$, it follows that different answers to the membership query lead to read-once functions that are negations of each other. This means that our algorithm does not really need a membership query per se, and can instead be supplied with any valid pair of the form $\langle \alpha, f(\alpha) \rangle$, where α is any input vector from $\{0,1\}^n$. The reconstruction phase of the new algorithm will then produce two hypotheses for f, one of which is inconsistent with the given sample. The other hypothesis is equal to f and output by the algorithm.

4 Counting queries: algorithm overview

In the setting considered in this section, *counting relevance queries* are available. The only parameter provided by such a query is a partial assignment p. The

answer to the query is the value $r(f_p)$, i. e., the total number of variables relevant to the projection f_p.

The high-level scheme of our identification algorithm will be the same as in the previous section. We shall still construct a sequence of projections f^k, $k = n, \ldots, 1$, of the unknown function f such that each f^k has exactly k relevant variables and is obtained from f^{k+1} by substituting a constant for some variable x_{k+1}.

However, our strategy will be different from the one applied in Section 3. Recall that for each f^k we used $O(k)$ queries to find an appropriate variable x_k with a corresponding Boolean constant. This resulted in zero cost of the reconstructing step, that is, f^k could be obtained from f^{k-1} without any additional queries. Unfortunately, this gave us $\Theta(n^2)$ queries in total for n-variable functions in the worst case. In this section we shall use any relevant variable as x_k and make $O(1)$ queries to find an appropriate constant. Thus the overall complexity of the algorithm will be determined by the number of queries needed for reconstructing all f^k from their predecessors f^{k-1}. The whole process will be based on the following lemma, which is used implicitly in [14].

Lemma 1. *Let g be a read-once function with $k \geq 3$ relevant variables. Suppose that $x \in R(g)$ and $\sigma \in \{0, 1\}$ are chosen in such a way that $0 < r(g_{x \leftarrow \overline{\sigma}}) < r(g_{x \leftarrow \sigma})$. Then the tree T of g can be obtained from the tree T' of $g_{x \leftarrow \sigma}$ with one of the following operations (see Fig. 2):*

1) *attaching a new leaf x to some internal node v;*
2) *replacing a leaf y with a new internal node u, labeling it with a symbol dual to that of the former parent v of y and attaching x and y as children of u;*
3) *splitting an internal node v into two nodes v_0 and v_1 and injecting a new node u between them, i. e., if v is labeled with $\vee$ and $g_1, \ldots, g_s$ are functions represented by its children, then T is obtained by replacing T'_v with a subtree representing a function*

$$((g_{i_1} \vee \ldots \vee g_{i_t}) \wedge x^\sigma) \vee g_{i_{t+1}} \vee \ldots \vee g_{i_s},$$

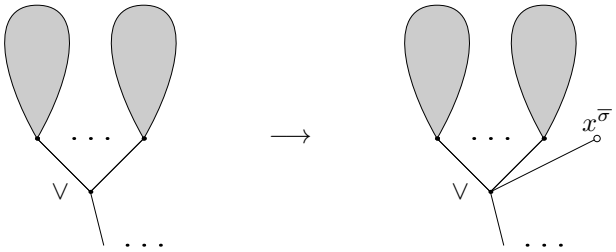
where $\{i_1, \ldots, i_s\} = \{1, \ldots, s\}$, $s \geq 3$, and $t \geq 2$ (if v is labeled with $\wedge$, then all symbols in the formula are replaced with dual ones).

Proof. By Claim 1 we have $R(g) \setminus R(g_{x \leftarrow \sigma}) = \{x\}$. Let the tree T of g be known and consider the parent w of the leaf x in T. Denote by d the number of children (direct descendants) of w. If $d \geq 3$, then substituting σ for x eliminates one of them and the rest of the tree is left untouched. This corresponds to case 1 in the list above. If $d = 2$, then w has exactly one child besides x. If this child is a leaf, we observe case 2. Otherwise, this child is labeled with the same symbol as the parent of w, and these two nodes are glued together in T'. This situation is described in case 3. Note that w cannot be the root of T, since $r(g_{x \leftarrow \overline{\sigma}}) > 0$. This concludes the proof.

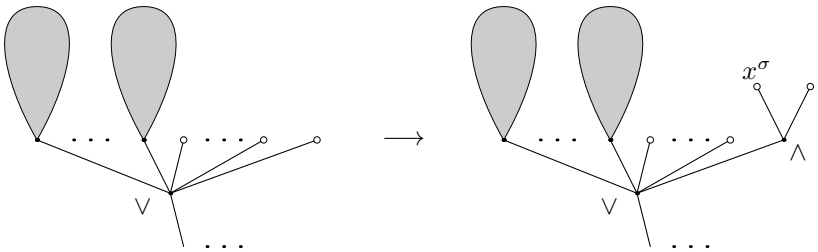
We remark that the case $r(g_{x \leftarrow \overline{\sigma}}) = 0$ is simple and considered separately.

Fig. 2. Tree transformations
(in the dual case $\vee$ and $\wedge$ are exchanged and σ is negated)

(a) Attaching a new leaf to an existing node



(b) Adding a new internal node with two descendants



(c) Splitting an existing node

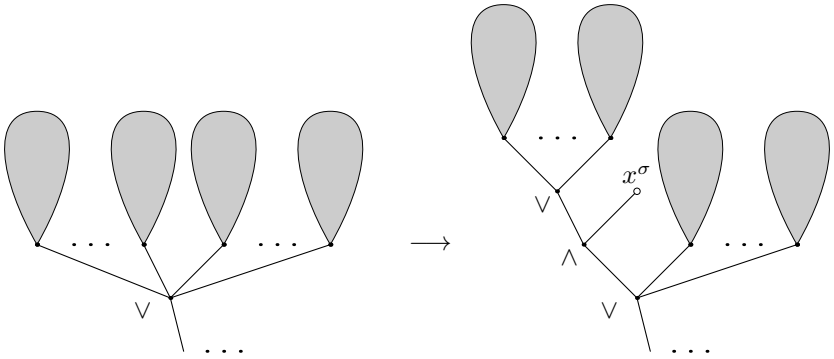


Fig. 3. Algorithm $\mathcal{A}_2$: membership and counting relevance queries

1. Identify all relevant variables. Denote them by $x_1, \ldots, x_n$.
2. Construct a sequence of constant values $\sigma_n, \ldots, \sigma_2$ from $\{0,1\}$ such that functions $f^n, \ldots, f^1$ defined by $f^n = f$ and $f^{k-1} = f^k_{x_k \leftarrow \sigma_k}$ have $R(f^k) = \{x_1, \ldots, x_k\}$.
3. Decide whether $f^1 = x_1$ or $f^1 = \overline{x}_1$.
4. For $k = 2, \ldots, n$, reconstruct f^k from its projection f^{k-1}:
   (a) If $r(f^k_{x_k \leftarrow \overline{\sigma}_k}) = 0$: choose $\circ \in \{\wedge, \vee\}$ and $\tau \in \{0,1\}$ such that $f^k = f^{k-1} \circ x_k^\tau$, reconstruct the tree of f^k and proceed to the next k.
   (b) Find a node a in the known tree of f^{k-1} such that $a = \mathrm{anc}(\mathrm{anc}(x_k))$ in an unknown tree of f^k.
   (c) Identify the set of variables $R(f^k_{x_k \leftarrow \sigma_k}) \setminus R(f^k_{x_k \leftarrow \overline{\sigma}_k})$.
   (d) Transform the tree of f^{k-1} into the tree of f^k, and proceed to the next k, if any.
5. Terminate with $f = f^n$.

Suppose that $f^{k-1} = f^k_{x \leftarrow \sigma}$, where f^k, x and σ satisfy the conditions of Lemma 1. Also assume that T' is a known tree of f^{k-1} and T is an unknown tree of f^k. We shall split the task of reconstructing T from T' into two parts. First, we locate the node $a \in T'$ such that $a = \mathrm{anc}(\mathrm{anc}(x))$ in T. Second, we identify the set of variables relevant to $f^k_{x \leftarrow \sigma}$ but not to $f^k_{x \leftarrow \overline{\sigma}}$. Once the second part is done, we can easily insert in T' a new leaf labeled with an appropriate literal of x. The obtained tree will then be guaranteed to represent f^k, which is exactly what we need.

We are now ready to present a sketch of our algorithm $\mathcal{A}_2$ for exact identification with membership and counting relevance queries. The main steps of the algorithm are given in Fig. 3. For now, we do not focus on implementation details, however crucial they may be, and leave out the rigorous description of the actions. Details for steps 2, 3, 4a and 4d are explained in the proof of Theorem 2 in this section, and implementation of steps 4b and 4c is postponed until Section 5 and Section 6, respectively. Our final goal is to prove that all the steps can be implemented in such a way that the total query complexity is bounded by $O(n \log^2 n)$.

Observe that the overall complexity depends primarily on the implementation of steps 4b and 4c, since all other computations can be performed with a moderate number of queries, compared to the announced value. We claim that these two steps can also be done with an appropriate number of queries. We split this claim into two lemmas.

Lemma 2. *Computation on step 4b in the algorithm $\mathcal{A}_2$ can be done by a subroutine that performs at most $\log_2 k + O(1)$ counting relevance queries.*

Lemma 3. *Computation on step 4c in the algorithm $\mathcal{A}_2$ can be done by a subroutine performing counting relevance queries and having the following property: the total number of queries it makes when $\mathcal{A}_2$ is invoked on a read-once function f with n relevant variables is $O(n \log^2 n)$.*

Proofs of these two lemmas are given in the sequel. Namely, Section 5 is devoted to the proof of Lemma 2, and in Section 6 the proof of Lemma 3 is given.

Now the task of obtaining the announced complexity bound is basically reduced to the task of proving Lemmas 2 and 3. The proof of Theorem 2 below fills in the rest of the missing details.

Theorem 2. *The algorithm $\mathcal{A}_2$ correctly identifies any read-once function f and performs one membership query and $O(n \log^2 n)$ counting relevance queries, where n is the number of input variables of f.*

Proof. First observe that $y \in R(f)$ if and only if $r(f) \neq r(f_{y \leftarrow 0})$, so step 1 can be done with $n+1$ queries. Step 2 is done according to Claim 1, takes at most $2(n-1)$ queries, and reveals the values $r(f^k_{x_k \leftarrow 0})$, $r(f^k_{x_k \leftarrow 1})$ for $k = n, \ldots, 2$. On step 3 one membership query is performed.

Consider iterations of step 4. First suppose that $r(f^k_{x_k \leftarrow \overline{\sigma}_k}) = 0$. Then one counting relevance query is sufficient to reconstruct f^k. Indeed, observe that the projection $f^k_{x_k \leftarrow \overline{\sigma}_k}$ is equal to some Boolean constant β. Take arbitrary values $\alpha_1, \ldots, \alpha_{k-1} \in \{0,1\}$ and construct a partial assignment $p' = \{x_1 \leftarrow \alpha_1, \ \ldots \ , \ x_{k-1} \leftarrow \alpha_{k-1}\}$. Now $\beta' = f^{k-1}_{p'}$ is known, and $\beta = \beta'$ if and only if $r(f^k_{p'}) = 0$. (We remark that we could also use a membership query to reveal β.) If $\beta = 0$, then $f^k = f^{k-1} \wedge x_k^{\sigma_k}$, otherwise $f^k = f^{k-1} \vee x_k^{\overline{\sigma}_k}$.

Now suppose that $r(f^k_{x_k \leftarrow \overline{\sigma}_k}) > 0$. By Lemmas 2 and 3, steps 4b and 4c reveal the node $a = \mathrm{anc}(\mathrm{anc}(x_k))$ and the set of variables relevant to $f^k_{x_k \leftarrow \sigma_k}$ but not to $f^k_{x_k \leftarrow \overline{\sigma}_k}$. After that, step 4d transforms the tree of f^{k-1} into the tree of f^k according to Lemma 1 and chooses the literal of x_k such that all variables of f^k except x_k be relevant to $f^k_{x_k \leftarrow \sigma_k}$ (similarly to the case $r(f^k_{x_k \leftarrow \overline{\sigma}_k}) = 0$, although no additional queries are needed here).

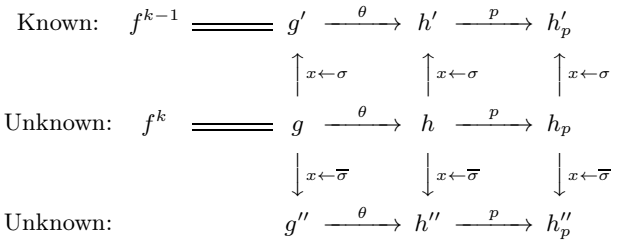
It remains to determine the total number of queries made by the algorithm. We have

$$O(n) + \sum_{k=2}^{n} (\log_2 k + O(1)) + O(n \log^2 n) = O(n \log^2 n),$$

which completes the proof.

Remark. The algorithm $\mathcal{A}_2$, similarly to $\mathcal{A}_1$, does not exploit the power of the membership query it makes, in the sense that this query can be replaced with any valid sample $\langle \alpha, f(\alpha) \rangle$ provided as input. As seen from Lemma 1, the tree structure of f can be reconstructed just by looking at consecutive differences $R(f^k_{x_k \leftarrow \sigma_k}) \setminus R(f^k_{x_k \leftarrow \overline{\sigma}_k})$, with the only exception being the case $r(f^k_{x_k \leftarrow \overline{\sigma}_k}) = 0$, where one counting relevance query still reveals all the needed information. One can prove by induction that without a membership query the algorithm will be able to provide two hypotheses, which are negations of one another. This effect is related to a simple observation that any types of queries dependent only on the information about relevance of input variables are able to distinguish Boolean values, but not to tell which is 0 and which is 1.

Fig. 4. Projections and partial assignments in Proof of Lemma 2



5 Proof of Lemma 2

This section is devoted to the proof of Lemma 2. We first introduce some notation, restate the lemma itself, and present main ideas of the proof. After that, we discuss the proof in detail.

Suppose that f^{k-1} is a known read-once function with $R(f^{k-1}) = \{x_1, \ldots, x_{k-1}\}$. Let f^k be an unknown read-once function such that $R(f^k) = R(f^{k-1}) \cup \{x_k\}$ and $f^k_{x_k \leftarrow \sigma_k} = f^{k-1}$, where σ_k is a constant from $\{0, 1\}$. By T' we denote the known tree of f^{k-1} and by T the unknown tree of f^k.

According to the description of the algorithm $\mathcal{A}_2$, we assume that $r(f^k_{x_k \leftarrow \overline{\sigma}_k}) > 0$. This implies that the leaf of T labeled with a literal of the variable x_k is not adjacent to the root of T. In other words, there exists a node a in T such that $a = \mathrm{anc}(\mathrm{anc}(x_k))$. Recall that T can be obtained from T' by means of one of the three operations described in Lemma 1, so a is also a node of T'. The goal of this section is to prove that this node $a \in T'$ can be identified with the aid of $\log_2 k + O(1)$ counting relevance queries.

Since throughout the proof the value of k is fixed, we shall use auxiliary notation to avoid unnecessary indexing. By definition, put $x = x_k$ and $\sigma = \sigma_k$. Also denote $g = f^k$ and $g' = f^k_{x \leftarrow \sigma} = f^{k-1}$, then T is the tree of g and T' is the tree of g'. Finally, put $g'' = f^k_{x \leftarrow \overline{\sigma}}$ and define $m = r(g') - r(g'')$. We note that links between the three projections of the function f mentioned in this paragraph are reflected in two leftmost columns in Fig. 4.

The main idea of the proof is to find a variable that belongs to the set difference $R(g') \setminus R(g'')$. At this stage, we do not need to identify the entire set of all such variables exactly, since the real goal of this step is just to identify the node a. These variables, however, are the key to the identification of a.

A straightforward algorithm searching for a variable $y \in R(g') \setminus R(g'')$ would try to assign constant values to variables of the unknown function g'' and analyze the number of variables relevant to the obtained projections. Unfortunately, this approach requires $\Theta(k)$ queries in the worst case, which only gives an $O(n^2)$ bound on the overall query complexity of the algorithm. The idea applied in this section is the use of simultaneous assignments to several variables. Let p be a partial assignment of constants to some s variables. Observe that if all these variables are relevant to g'', then the value $r(g''_p)$ is at most $r(g'') - s$. Now

suppose that at least one of these variables is irrelevant to g'', then this fact will be revealed if $r(g''_p) > r(g'') - s$. If we construct the assignment p carefully, then the last inequality will be guaranteed to hold whenever p assigns a constant value to some variable $y \in R(g') \setminus R(g'')$. After that, we shall be able to use binary search to find such a variable y and identify the node a.

We now begin to carry out the plan described above. We first need an auxiliary construction, which will help us throughout subsequent work. We shall use our knowledge of the gap $m = r(g') - r(g'')$ to reduce tree transformations of Lemma 1 to manipulations with a number of subtrees, whose set can be delineated beforehand. Recall that the number of leaves in a tree that are descendants of a node v is denoted by $l(v)$. Consider the set

$$N = \{\, u \in T' \mid u \neq v_0,\ l(u) \leq m,\ l(\mathrm{anc}(u)) > m \,\},$$

where by v_0 we denote the root of T'. The next claim basically restricts the set of all possible tree transformations taking T' to T to those leaving intact all subtrees T'_u, where $u \in N$.

Claim 4. *All variables in the set difference $R(g') \setminus R(g'')$ are contained in subtrees T'_u, where $u \in N$. Moreover, all subtrees T'_u, $u \in N$, containing variables from $R(g') \setminus R(g'')$ contain no variables from $R(g'')$ and have roots u such that $a = \mathrm{anc}(u)$.*

Proof. Let w be an internal node of T adjacent to x. Denote by l_w the number of leaves in T_w, then we have $m = l_w - 1$. Observe that the node $a = \mathrm{anc}(w) \in T$ has at least one child besides w. So in the tree T' it holds that $l(a) > l_w - 1 = m$. Furthermore, in the tree T' all variables from the set $R(g') \setminus R(g'')$ lie in subtrees T'_u, where nodes u are children of a (two possible cases are shown in Fig. 5). All these subtrees are left intact in T, as compared to T', and contain no variables from $R(g'')$. The total number of leaves in these subtrees equal m, so $l(u) \leq m$. This concludes the proof of Claim 4.

Remark. A possible perspective on Claim 4 is that once a variable $y \in R(g') \setminus R(g'')$ has been found, the task of identifying the node a can be carried out by taking the parent of a unique node $u \in N$ such that $y \in T'_u$. The reason for this is that for each variable $x_i \in R(g')$ there exists only one node u on the path from x_i to v_0 such that $l(u) \leq m$ and $l(\mathrm{anc}(u)) > m$.

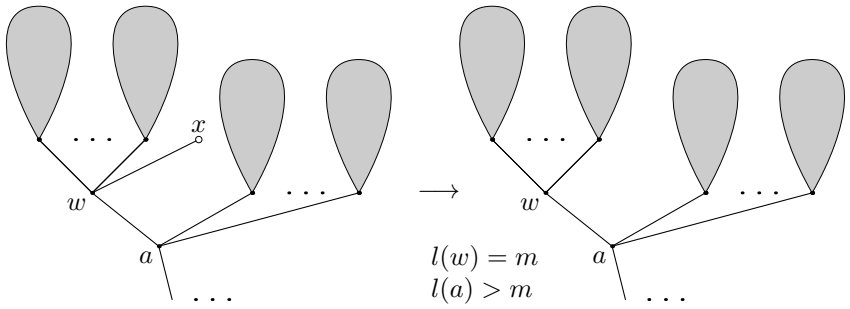
By Lemma 1, the tree T can be obtained from T' either by attaching a new leaf to one of the nodes $u \in N$ such that $l(u) = m$ (if $m = 1$, this means introducing a new internal node) or by "splitting" one of their parents, i. e., nodes from the set

$$A = \{\, \mathrm{anc}(u) \in T' \mid u \in N \,\},$$

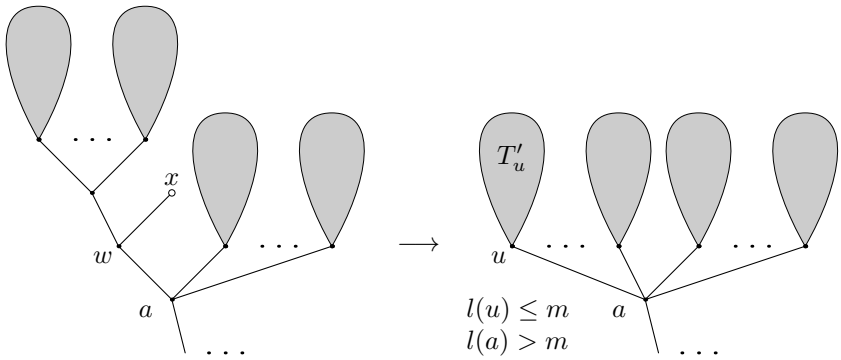
and "lifting" a subset of N. Now suppose that u is a non-leaf node contained in N. The corresponding subtree T'_u represents some read-once function g'_u, which has at least two relevant variables. Consider any single leaf in T'_u, labeled with a literal x_i^τ, for $\tau \in \{0, 1\}$. Since x_i is relevant to g'_u, there exists a partial

Fig. 5. Locating subtrees that are left intact

(a) The node w has indegree 3 or greater in T



(b) The node w has indegree 2 in T



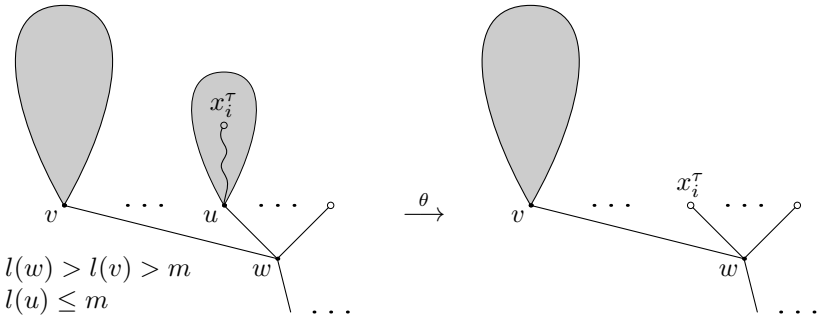
assignment θ_u to all other variables of g'_u such that $(g'_u)_{\theta_u} = x_i^\tau$. Let a partial assignment θ be the composition of such partial assignments θ_u for all non-leaf nodes u from N (see Fig. 6; if all the elements of N are leaves, then θ does nothing).

This assignment θ takes the functions g, g', g'' to h, h', h'', respectively (see left horizontal arrows in Fig. 4). The functions h and h'' are still unknown. Denote by T'_1 the tree of h', which is obtained from T' by replacing subtrees having roots from N with single leaves. The set N is transformed by θ into a set N_θ, which contains only leaves of T'_1.

Put $m_1 = r(h') - r(h'') > 0$. (Here one counting query is needed to reveal the value of $r(h'')$.) For each node v of T'_1, denote by $\lambda(v)$ the number of leaves of T'_1 which are children (direct descendants) of v. Put

$$A_1 = \{\, v \in A \mid \lambda(v) \geq m_1 + \delta(v) \,\},$$

Fig. 6. Partial assignment θ



where $\delta(v) = 0$ if the node v in T_1' has at least one non-leaf child and $\delta(v) = 1$ if all children of v are leaves. Finally, consider the set

$$N_1 = \{\, u \in N_\theta \mid \mathrm{anc}(u) \in A_1 \,\}.$$

We next show that this set contains all variables relevant to h' but not to h''.

Claim 5. *For any variable $y \in R(h') \setminus R(h'')$, it holds that $y \in N_1$ and $\mathrm{anc}(y) = a \in A_1$.*

Proof. First observe that $R(h') \setminus R(h'') \subseteq R(g') \setminus R(g'')$. By Claim 4, all variables of g' not relevant to g'' are contained in subtrees T'_u, where $u \in N$. Since T_1' is obtained from T' by replacing these subtrees with single leaf nodes, it follows that all variables from $R(h') \setminus R(h'')$ belong to the set N_θ. Furthermore, it also follows from Claim 4 that $\mathrm{anc}(u)$ is the same node $a \in A$ for all these subtrees T'_u. This node a in T_1' must have at least m_1 children that are leaves, so it remains to check the inequality $\lambda(a) \geq m_1 + \delta(a)$. Indeed, if all children of a are leaves, then their number is greater than or equal to $m_1 + 1$, because otherwise the tree of h would be obtained from T_1' by attaching a new leaf to a, and the same would hold for T and T' (which is impossible since some of the children of a in T' are contained in N, so $l(a) > m$ in T' and, therefore, $a \notin N$). This concludes the proof of Claim 5.

Claim 5 shows that the tree of h'' can be obtained from T_1' by eliminating some leaves that belong to N_1 and have the node a as their common parent, and possibly by suppressing an internal node of indegree 1 and glueing two adjacent nodes with identical labels in the resulting tree.

Remark. The overall approach to the task of locating the node a can be implemented without an explicit construction of the assignment θ. It seems, however, that the transformation described above makes the whole argument more transparent, since the problem appears reduced to a particular case.

Now all preliminary preparations are over and we move on to implementing our approach to the identification of the node a. Recall that we need to construct partial assignments to variables having some special properties. Let us introduce the following definition. Call a partial assignment $p = \{x_{i_1} \leftarrow \alpha_1, \ \dots \ , \ x_{i_s} \leftarrow \alpha_s\}$ *weak* if the following conditions are satisfied:

1) all leaves labeled with literals of $x_{i_1}, \dots, x_{i_s}$ in the tree of h' are adjacent to internal nodes from A_1;
2) for each j, all relevant variables of h' except x_{i_j} are relevant to $h'_{x_{i_j} \leftarrow \alpha_j}$;
3) for each internal node v of T'_1, if all children of v are leaves, then at most $(\lambda(v) - m_1)$ of them are labeled with literals of variables from $\{x_{i_1}, \dots, x_{i_s}\}$.

The concept of a weak partial assignment captures our idea of distinguishing the case when constants are substituted for relevant variables only from the case when an irrelevant variable is assigned some constant value. This is formalized by the following statement:

Claim 6. *If a weak partial assignment p assigns a value to a variable y not relevant to h'', then all relevant variables of h'' except those assigned by p are also relevant to h''_p.*

Proof. We first remark that key (although not all) projections of f discussed here can be found in Fig. 4. If $y \in R(h') \setminus R(h'')$, then in the tree T'_1 of h' it holds that $\mathrm{anc}(y) = a$. For an arbitrary node v of T'_1, denote by h'_v the function represented by the subtree $(T'_1)_v$ and by h''_v the function represented by the corresponding subtree of the tree of h''. (Note that although subscript here can still be interpreted as referring to a projection of the original function, this meaning is secondary.)

By condition 2 in the definition of a weak assignment, each separate sub-assignment of p having the form $x_{i_j} \leftarrow \alpha_j$ cannot make other variables of h' irrelevant. Recall that the tree of h'' can be obtained from T'_1 by eliminating several leaves. One can easily check that this elimination can be represented as a composition ξ of m_1 assignments of the form $x_i \leftarrow \tau_i$ having the same property (for this assignment ξ, it holds that $(h')_\xi = h''$). Therefore, it is sufficient to prove that under the conditions of the claim, for each internal (non-leaf) node $v \in T'_1$ the function h'_v is taken by the composition ξp to a non-constant projection $(h'_v)_{\xi p} = (h''_v)_p$. We shall do so using the fact that a function is non-constant iff it has at least one relevant variable.

First consider all nodes $v \in T'_1$ such that the node a is neither v nor a descendant of v. Use induction on the depth of $(T'_1)_v$, i. e., on the maximum number of edges on shortest paths from v to its descendants. For a node v, if all its children are leaves, then the desired follows directly from condition 3 in the definition of a weak partial assignment, since at least one variable is not assigned by p. If one of the children is a non-leaf node w, then the desired follows from the inductive assumption for w and from condition 2 in the definition of p.

Now consider the node a. If a has at least one non-leaf child v, then it follows that all variables relevant to $(h''_v)_p$ are also relevant to $(h''_a)_p$. Suppose

that the set W of all children of a contains only leaves. Recall that their number is $\lambda(a) \geq m_1 + 1$ and at most $(\lambda(a) - m_1)$ of them are assigned by p. Since ξ assigns values to m_1 elements of W, and at least one variable is assigned a value by both p and ξ, it follows that at least one variable from W is not assigned by either p or ξ. Therefore, this variable is relevant to $(h''_a)_p$.

The same reasoning as in the inductive step above shows that for all nodes v such that a is a descendant of v the obtained projection $(h'_v)_{\xi p} = (h''_v)_p$ has at least one relevant variable. It then follows from the definition of ξ and from condition 2 in the definition of a weak partial assignment that all variables relevant to h' except those assigned by either ξ or p are also relevant to h''_p. This completes the proof of Claim 6.

The last auxiliary claim in this section shows that a constant number of weak assignments are sufficient for the identification of a variable from $R(h') \setminus R(h'')$:

Claim 7. *For each given read-once function g' and any possible values of $r(g'')$ and $r(h'')$, there exist two weak partial assignments p_1, p_2 such that for any read-once function g that is consistent with g' and these values at least one of these assignments assigns a value to a variable not relevant to h''.*

Proof. For each node $v \in A_1$, consider the set $W(v)$ of all children of v that belong to N_1. If v has at least one non-leaf child, then let p_1 assign constant values α_j to all variables x_{i_j} from $W(v)$. These values α_j are chosen according to the label of v so that $R(h'_{x_{i_j} \leftarrow \alpha_j}) = R(h') \setminus \{x_{i_j}\}$ (see also Claim 1 from Section 3). If all children of v are leaves, then let each of p_1 and p_2 assign constant values to $\lambda(v) - m_1 \geq 1$ variables from $W(v)$ so that the total number of variables assigned by the composition $p_1 p_2$ is maximized. Remark that if $v = a$, then exactly $\lambda(v) - m_1$ variables from $W(v)$ are relevant to h'', so in any subset of $W(v)$ having size $\lambda(v) - m_1 + 1$ there exists a variable irrelevant to h''. Observe that $(\lambda(v) - m_1 + 1)/(\lambda(v) - m_1) \leq 2$. It follows that if all variables from $W(v)$ assigned by p_1 and p_2 are relevant to h'', then so are all other variables from $W(v)$. This concludes the proof of Claim 7.

Claims 6 and 7 indicate an algorithm for the identification of the node a. During first (preliminary) steps the partial assignment θ is constructed as described above, and sets N_1 and A_1 are computed (one counting relevance query is needed to determine the value $m_1 = r(h') - r(h'')$). Then the algorithm constructs the assignments p_1, p_2 according to Claim 7 and runs counting relevance queries for both induced projections h''_{p_i}.

Denote by s_i the number of variables assigned by p_i. By Claim 6, if at least one of these variables is not relevant to h'', then each relevant variable of h'' is either also relevant to h''_{p_i} or assigned a value by p_i. Hence, $r(h''_{p_i}) + s_i > r(h'')$. On the other hand, if all variables assigned by p_i are relevant to h'', then $r(h''_{p_i}) + s_i \leq r(h'')$. It follows that comparison of sums $r(h''_{p_i}) + s_i$ to $r(h'')$ always reveals an assignment p_i whose existence is guaranteed by Claim 7.

The rest of the proof is straightforward. Since all sub-assignments of a weak partial assignment are also weak, a simple binary search identifies the node a

with at most $\log_2 k$ queries. The total number of queries needed on the step 4b is less than or equal to $\log_2 k + O(1)$. This concludes the proof of Lemma 2.

6 Proof of Lemma 3

This section is devoted to the proof of Lemma 3. Recall that for each fixed $k = 2, \ldots, n$ we are given a read-once function f^{k-1} such that $R(f^{k-1}) = \{x_1, \ldots, x_{k-1}\}$ and are asked to reconstruct an unknown read-once function f^k such that $R(f^k) = R(f^{k-1}) \cup \{x_k\}$ and $f^k_{x_k \leftarrow \sigma_k} = f^{k-1}$, where σ_k is a known constant. From the previous step of the algorithm we also learn the node a in the known tree T' of f^{k-1} such that $a = \mathrm{anc}(\mathrm{anc}(x_k))$ in the unknown tree T of f^k (since such a node exists, we deduce that $k \geq 3$).

Recall that the reconstruction of f^k can be performed according to Lemma 1, which guarantees that given the tree T', one can obtain the tree T with the aid of one of the three explicit operations (actually, there also exists a special degenerate case, but we shall put off its discussion until further need). What is more, Lemma 1 shows that the only information needed for the reconstruction of f^k, apart from T', is the set $R(f^k_{x_k \leftarrow \sigma_k}) \setminus R(f^k_{x_k \leftarrow \overline{\sigma}_k})$. Lemma 3, which we prove in this section, says that this set can be identified efficiently, provided that the tree T' is known and the node $a = \mathrm{anc}(\mathrm{anc}(x_k))$ is found in advance. More rigorously, for any read-once function f with n relevance variables, the total number of queries needed for the identification of sets $R(f^k_{x_k \leftarrow \sigma_k}) \setminus R(f^k_{x_k \leftarrow \overline{\sigma}_k})$, $k = 3, \ldots, n$, is at most $O(n \log^2 n)$.

We shall now describe an algorithm performing this identification. Following the lines of the previous section, we first reduce the general problem to its particular case, where additional structure is assumed. Observe that all variables from the difference $R(f^k_{x_k \leftarrow \sigma_k}) \setminus R(f^k_{x_k \leftarrow \overline{\sigma}_k})$ lie in some of the subtrees T'_u, where nodes u are children of the node a. Using a single partial assignment θ, we can ensure that all these subtrees consist of single leaves, that is, the tree of the unknown projection $(f^k_{x_k \leftarrow \overline{\sigma}_k})_\theta$ can be obtained from the tree of the projection $(f^{k-1})_\theta$ by eliminating some leaves with the common parent a.

An explicit construction of the assignment θ is provided in Section 5 and will not be duplicated here. However, we shall reuse the notation $h = f^k_\theta$, $h' = f^{k-1}_\theta$, and $h'' = (f^k_{x_k \leftarrow \overline{\sigma}_k})_\theta$ introduced previously. We note that the value $m_1 = r(h') - r(h'')$ is always positive and can be computed with the aid of a single counting relevance query. As before, T'_1 denotes the tree of h'.

We now summarize the obtained (reduced) setting. Given T'_1 and the node $a \in T'_1$, we wish to identify the set of leaves $u \in T'_1$ adjacent to a that are labeled with literals of variables not relevant to the function h''.

Main technique of doing this is the same as in the previous section. Let W be the set of leaves adjacent to a in T'_1. All the algorithm does is choose subsets of W, substitutes appropriate constants for the corresponding variables and performs queries to reveal the number of relevant variables of the induced projections. In short, for any subset $U \subseteq W$ the algorithm can learn the number of variables in

Fig. 7. Subroutine $\mathcal{S}$: identifying a subset of a set

Arguments: a set W and an integer m.
Oracle: an intersection function $q(\cdot)$.

1. If $m = 0$, return $\emptyset$. If $m = |W|$, return W.
2. Otherwise, choose any $U \subseteq W$ such that $|U| = \lfloor |W|/2 \rfloor$.
3. Make a query to the oracle to reveal $q_1 = q(U)$.
4. Make two recursive calls to compute $R = \mathcal{S}(U, q_1) \cup \mathcal{S}(W \setminus U, m - q_1)$.
5. Return R.

U which are relevant to h'' with a single counting query. A variant of a binary search algorithm can then be applied to identify the set $R(h') \setminus R(h'')$.

Note, however, that here we cannot directly apply the argument of Claim 6 from Section 5 to reveal the exact number of variables from U relevant to h''. It turns out that some partial assignments p to variables from U needed on this step are not weak in the sense of that section. Nevertheless, the fact that all variables from U have a common parent in T'_1 allows us to circumvent this difficulty:

Claim 8. *Suppose that $U = \{u_1, \ldots, u_s\}$ is a subset of W, and an assignment $p = \{u_1 \leftarrow \alpha_1, \ \ldots \ , \ u_s \leftarrow \alpha_s\}$ is such that $R(h'_{u_i \leftarrow \alpha_i}) = R(h') \setminus \{u_i\}$ for $i = 1, \ldots, s$. Then the value*

$$q(U) = |U| - \min\{\, r(h'') - r(h''_p),\ |W| - m_1 \,\},$$

where $m_1 = r(h') - r(h'')$, is equal to the number of leaves from U labeled with variables irrelevant to h''.

Proof. Consider the set $Q = R(h') \setminus R(h'')$ and observe that $m_1 = |Q|$. If $Q \cup U \neq W$, then $r(h'') - r(h''_p) < |W| - m_1$ and $q(U) = |U| - (r(h'') - r(h''_p))$, which is exactly the number of variables from U that are irrelevant to h''. If $Q \cup U = W$, then $r(h'') - r(h''_p) \geq |W| - m_1$. In this case $q(U) = |U| - (|W| - m_1) = |U| + |Q| - |Q \cup U| = |Q \cap U|$. This concludes the proof of Claim 8.

It follows that the whole task of identification of the set $R(h') \setminus R(h'')$ can be performed by Subroutine $\mathcal{S}$ defined in Fig. 7, supplied with the oracle for $q(U)$. After that, the reconstruction of the set $R(f^k_{x_k \leftarrow \sigma_k}) \setminus R(f^k_{x_k \leftarrow \overline{\sigma}_k})$ can be done without making any queries at all, since an arbitrary variable $y \in R(f^{k-1})$ is relevant to $f^k_{x_k \leftarrow \overline{\sigma}_k}$ iff it lies in a subtree T'_u such that $u \in R(h'')$.

We shall now prove the correctness of the algorithm described above and the claimed upper bound on the number of queries performed on this step (4c in Fig. 3 for Algorithm $\mathcal{A}_2$). From now on, by $\log c$ we denote the logarithm of c to base 2. For an individual call to Subroutine $\mathcal{S}$, we prove the following proposition:

Claim 9. *Suppose that W is a finite set and $Q \subseteq W$, $|Q| = m$. Assume that $q(U) = |Q \cap U|$ for all $U \subseteq W$. Then Subroutine $\mathcal{S}$ from Fig. 7, when supplied with W, m and $q(\cdot)$, always terminates, returns Q, and makes at most $\min\{m, |W| - m\}\lceil \log |W| \rceil$ queries.*

Proof. $\mathcal{S}$ is an implementation of a binary search variant on W. For $|W| \leq 1$, the algorithm always terminates without recursive invocations and returns Q. Otherwise, the cardinality of both U and $W \setminus U$ is strictly less than $|W|$. It then follows from the description of $\mathcal{S}$ that for any finite set W, Subroutine $\mathcal{S}$ terminates and returns Q.

To prove the bound on the number of oracle calls, denote by w the cardinality of W. The proof is by induction on the depth of recursive invocations of $\mathcal{S}$. If $m = 0$ or $m = w$, there is nothing to prove. Note that $w = 1$ also implies no queries at all. Now suppose that $w \geq 2$ and $0 < m < w$. By definition, put $w_1 = \lfloor w/2 \rfloor$ and $w_2 = \lceil w/2 \rceil$. Also put $q_2 = m - q_1$ and $m_i = \min\{q_i, w_i - q_i\}$, $i = 1, 2$. We now show that the following inequality always holds:

$$1 + m_1 \lceil \log w_1 \rceil + m_2 \lceil \log w_2 \rceil \leq \min\{m, w - m\} \lceil \log w \rceil.$$

First observe that $\lceil \log w_1 \rceil \leq \lceil \log(w/2) \rceil = \lceil \log w \rceil - 1$. Secondly, if $\lceil \log w_2 \rceil > \lceil \log w \rceil - 1$, then $\lceil \log(2\lceil w/2 \rceil) \rceil > \lceil \log w \rceil$, which means that w cannot be even. Assume that $w = 2l + 1$, then $\lceil \log(2l + 2) \rceil > \lceil \log(2l + 1) \rceil$. It follows that $2l + 1$ is a power of 2. In fact, the only possibility is $w = 1$, but that contradicts our assumption. Hence, $\lceil \log w_2 \rceil \leq \lceil \log w \rceil - 1$. We now obtain

$$\begin{aligned} 1 + m_1 \lceil \log w_1 \rceil + m_2 \lceil \log w_2 \rceil &\leq 1 + m_1(\lceil \log w \rceil - 1) + m_2(\lceil \log w \rceil - 1) \\ &= (m_1 + m_2)\lceil \log w \rceil - (m_1 + m_2 - 1). \end{aligned}$$

It suffices to prove, then, that $m_1 + m_2 \leq \min\{m, w - m\}$. Indeed, once we have done that, we notice that the desired inequality follows directly from the inequality $m_1 + m_2 - 1 \geq 0$. (If $m_1 + m_2 < 1$, then $m_1 = m_2 = 0$ and we are proving the inequality $1 \leq \min\{w - m, m\} \lceil \log w \rceil$, which obviously holds.) Since $m_i \leq q_i$ for $i = 1, 2$, we obtain that $m_1 + m_2 \leq q_1 + q_2 = m$. Similarly, from $m_i \leq w_i - q_i$ we get $m_1 + m_2 \leq w_1 + w_2 - m = w - m$. Thus, $m_1 + m_2 \leq \min\{m, w - m\}$, which concludes the proof of Claim 9.

Claim 9 proves the correctness of our algorithm identifying the set $R(f^k_{x_k \leftarrow \sigma_k}) \setminus R(f^k_{x_k \leftarrow \overline{\sigma}_k})$. It also shows that a relatively large number of queries can be made on each individual call to $\mathcal{S}$. Indeed, if $|W| = \Theta(k)$ and $m_1 = \Theta(k)$, then the obtained upper bound can be as large as $\Theta(k \log k)$. Furthermore, it cannot be improved by more than a polylogarithmic factor, since the number of all possible $\lfloor k/2 \rfloor$-sized subsets of a k-sized set equals $2^{k(1-o(1))}$.

For this reason, we shall prove the needed bound on the total number of queries performed during such invocations. Since the arguments of each subsequent invocation are somewhat dependant on the results of the previous ones, we develop a special technique to handle this dependency.

We shall need some notation related to directed graphs. In an directed graph, an arc from u to v is denoted by $u \to v$; we shall also say that v is a *successor* of u. Directed paths are written as $v_1 \to v_2 \to \ldots \to v_s$, where $v_1, \ldots, v_s$ are consecutive vertices of the path. A vertex with outdegree 0 is called a *sink*. A *branch* is a path that goes from a vertex with indegree 0 to a sink.

Recall that by Lemma 1, for each iteration of step 4 in $\mathcal{A}_2$, if the value $r(f^k_{x_k \leftarrow \overline{\sigma}_k})$ is strictly greater than 0, then there exist three types of operations which can transform the tree of f^{k-1} into that of f^k. In the special case $r(f^k_{x_k \leftarrow \overline{\sigma}_k}) = 0$, such a transformation is either attaching x_k to the root of the existing tree (case 1 in terms of Lemma 1) or introducing a new root having exactly two children.

Construct a sequence of directed graphs $G_2, \ldots, G_n$ with labeled vertices by the following rules. Let G_2 contain a single vertex labeled with 1. For each $k = 3, \ldots, n$, the graph G_k is obtained from G_{k-1} according to the type of transformation on trees performed by $\mathcal{A}_2$ when reconstructing f^k (see Fig. 8):

(a) Suppose T is obtained by attaching x_k to an existing node v (case 1 in Lemma 1, or $r(f^k_{x_k \leftarrow \overline{\sigma}_k}) = 0$ if v is the root of T'). Let $d+1$ be the number of children of v in T. Then take a sink w labeled with $d-1$ in G_{k-1} and add an arc from w to a new vertex labeled with d.
(b) Suppose that T is obtained from T' by introducing a new root (this implies $r(f^k_{x_k \leftarrow \overline{\sigma}_k}) = 0$) or injecting a new internal node with exactly two descendants (case 2 in Lemma 1). Here, add to G_{k-1} a new isolated vertex labeled with 1.
(c) Finally, suppose that T is obtained from T' by splitting an internal node v (case 3 in Lemma 1). As earlier, assume that v has s children in T' and x_k "lifts" t of them. In this case, take a sink w labeled with $(s-1)$ in G_{k-1} and add arcs from w to two new vertices labeled with $(t-1)$ and $(s-t)$, respectively. Moreover, add a new isolated vertex labeled with 1.

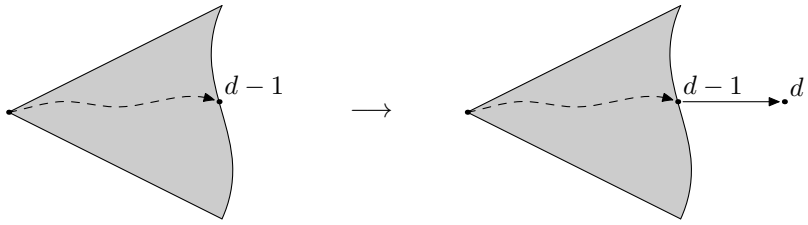
One can observe that the correctness of the described procedure is guaranteed by the fact that labels of sinks in G_k are exactly the numbers of children of internal nodes in T on step 4 of $\mathcal{A}_2$ (for this k) decreased by 1. Put $L_k = 0$ if (a) or (b) is performed on the iteration for this k, and $L_k = \min\{l', l''\}$ if (c) introduces arcs from a vertex labeled with $(l' + l'')$ to ones labeled with l' and l''.

Claim 10. *For all $k \geq 3$, the number of queries made on iteration k of step 4 in $\mathcal{A}_2$ does not exceed $(L_k + 1) \cdot \lceil \log(k-1) \rceil$.*

Proof. If $r(f^k_{x_k \leftarrow \overline{\sigma}_k}) = 0$, then Subroutine $\mathcal{S}$ is not invoked. Now suppose that $r(f^k_{x_k \leftarrow \overline{\sigma}_k}) \neq 0$. Then Subroutine $\mathcal{S}$ is asked to identify an m_1-sized subset of the set of nodes (leaves in T'_1) adjacent to a. Let w be the number of all these nodes. By Claim 9, $\mathcal{S}$ makes at most $\min\{m_1, w - m_1\}\lceil \log w \rceil$ queries. If $m_1 = 1$, then operation (a) or (b) is performed on G_{k-1}, $L_k = 0$ and the desired bound is obvious. If $m_1 \geq 2$, then operation (c) is performed, l' and l'' are $m_1 - 1$ and $w - m_1$, so $L_k + 1 = \min\{m_1, w - m_1 + 1\}$. This completes the proof of Claim 10.

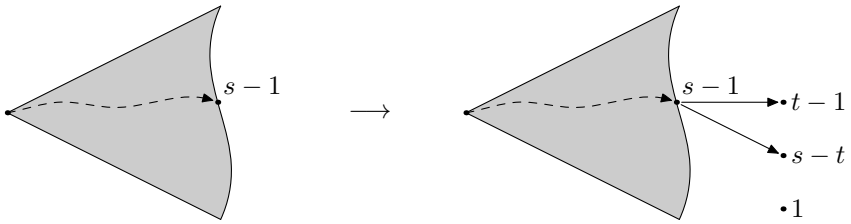
Fig. 8. Construction of graphs $G_2, \ldots, G_n$
(only transformed weakly connected components are shown)

(a) For attaching a new leaf to an existing node



(b) For introducing a new root or adding a new internal node with two descendants

(c) For splitting an existing node



For a graph G_n define

$$L(G_n) = \sum_{k=3}^{n} L_k \;= \sum_{\substack{v \to v_1 \\ v \to v_2}} \min\{L(v_1), L(v_2)\},$$

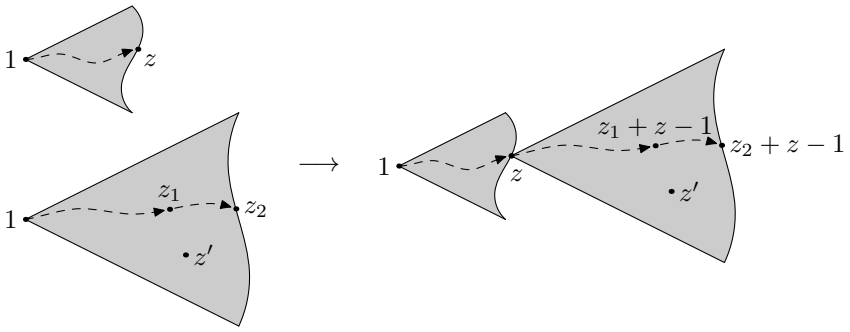
where $L(v)$ is the label of v in G_n and the sum in the last expression is over all $v \in G_n$ such that outdegree of v is 2 (for such a vertex v, by v_1 and v_2 we denote its successors).

Claim 11. $L(G_n) \leq n \log n$.

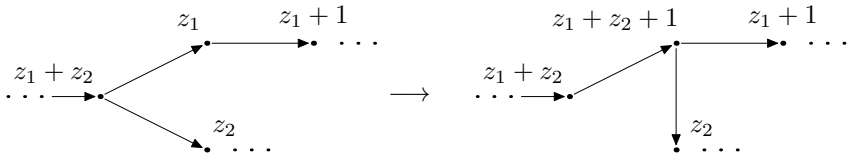
Proof. Observe that G_n is a forest that consists of rooted trees directed from the roots. If G_n contains more than one tree, then it can be transformed into a single tree G'_n by successive identification of trees' roots with other trees' leaves (see Fig. 9a). If in such an operation the label of the leaf u is $z > 1$, then one arbitrary branch $u \to u_1 \to u_2 \to \ldots \to u_s$ from u to a sink u_s in the obtained graph is selected and labels of all u_i are increased by $(z-1)$. This ensures that

Fig. 9. Graph transformations in Proof of Claim 11

(a) Identifying a root with a leaf



(b) Moving an arc



the label of each vertex v of outdegree 2 in the obtained tree G'_n is equal to the sum of labels of its direct successors, and the label of each vertex u of outdegree 1 is less by 1 than that of its successor. It can easily be seen that $L(G_n) \leq L(G'_n)$.

Now suppose that there exists a vertex v_0 in G'_n with outdegree 2 and arcs $v_0 \to v_1$, $v_0 \to v_2$ such that v_1 has only one successor v'_1. Suppose that labels of v_1 and v_2 are z_1 and z_2, then v_0 is labeled with $z_1 + z_2$, and v'_1 is labeled with $z_1 + 1$. Transform the tree by replacing the arc $v_0 \to v_2$ with the arc $v_1 \to v_2$ (see Fig. 9b). For consistency, the label of v_1 is changed to $z_1 + z_2 + 1$. Remark that this transformation either does not change $L(\cdot)$ or increases it by 1.

Continue transforming G'_n as long as possible. The directed tree G''_n obtained in the end has a simple path across the vertices of outdegree 1 from the root to some vertex u which is the root of a strictly binary subtree (this subtree contains no internal vertices of outdegree 1). By the argument above, $L(G'_n) \leq L(G''_n)$, and we shall now prove the required bound on $L(G''_n)$.

Let s be the label of u. We claim that $L(G''_n) \leq s \log s$. The proof is by induction on s. For $s = 1$, there is nothing to prove. If $s = 2$, then $L(G''_n) \leq 1 \leq 2 \log 2$. Suppose that $s \geq 3$. Assume that arcs $u \to u_1$, $u \to u_2$ are present in G''_n, and the labels of u_1, u_2 are x and $s - x$. Without loss of generality, assume that $x \leq s/2$. By definition, put $\phi(x) = x \log x$ and $\Phi(x) = x + \phi(x) + \phi(s - x)$. By the induction hypothesis, $L(G''_n) \leq \Phi(x)$. We now prove that $\Phi(x) \leq s \log s$

for all $x \in [1; s/2]$. Take the derivatives $\phi'(x) = (\ln x + 1)/\ln 2$ and

$$\Phi'(x) = 1 + \frac{(\ln x + 1) - (\ln(s-x)+1)}{\ln 2} = 1 + \log\frac{x}{s-x} = \log\frac{2x}{s-x}.$$

Since $x \le s/2$, the inequality $\Phi'(x) \ge 0$ is equivalent to $x \ge s/3$. This means that we only need to consider $x = 1$ and $x = s/2$. We have

$$\Phi(1) = 1 + 0 + (s-1)\log(s-1) \le \log s + (s-1)\log s = s\log s,$$
$$\Phi(s/2) = \frac{s}{2} + 2\cdot\frac{s}{2}\log\frac{s}{2} = \frac{s}{2} + s\log s - s \le s\log s.$$

This concludes the proof of Claim 11.

By Claims 10 and 11, the number of queries made by Algorithm $\mathcal{A}_2$ on step 4c is less than or equal to

$$\begin{aligned}\sum_{k=3}^{n}(L_k+1)\lceil\log(k-1)\rceil &\le \left(\sum_{k=3}^{n}L_k + n\right)\cdot(\log n + 1)\\ &\le (n\log n + n)\cdot(\log n + 1)\\ &= O(n\log^2 n).\end{aligned}$$

This completes the proof of Lemma 3.

7 Conclusions

We developed two algorithms for exact identification of read-once functions. Our first algorithm $\mathcal{A}_1$ makes $O(n^2)$ yes—no queries. Our second algorithm $\mathcal{A}_2$ makes $O(n\log^2 n)$ queries with logarithmically long and one-bit long answers. (In fact, the number of membership queries performed by each algorithm is one, and this query can be replaced with any valid sample $\langle\alpha, f(\alpha)\rangle$ supplied as input.) The second bound is close to the information-theoretic lower bound of $\Omega(n\log n)$ bits. The second algorithm is, in fact, allowed to query a special characteristic of the arithmetic sum $S(f_p)$ of all f_p's values for any partial assignment p — namely, the largest number k such that 2^k divides $S(f_p)$.

This characteristic $S(f_p)$ appears to play a key role in the process of exact identification, as performed by the algorithm. For a different setting where only the parity (sum modulo 2) of $S(f_p)$ is available [14], a known algorithm achieves only $n^2(1-o(1))$. If queries of the second least significant bit of $S(f_p)$ are additionally allowed [4], this value can be improved to $3n^2/4\cdot(1-o(1))$, which is still $\Theta(n^2)$, as compared to $O(n\log^3 n)$ bits achieved in this paper.

Acknowledgements. The author is grateful to Prof. Andrey A. Voronenko, who set up the original problem of exact identification with queries to the oracle that returns the value of $S(f_p)$. The author also wishes to thank his colleagues Maksim A. Bashov and Vladimir V. Lysikov for fruitful and inspiring discussions. This research was supported by Russian Foundation for Basic Research, project number 09-01-00817, and by Russian Presidential grant MD-757.2011.9.

References

1. A. V. Aho, J. E. Hopcroft, J. D. Ullman. Data structures and algorithms. Addison-Wesley, 1983.
2. D. Angluin. Queries and concept learning. *Machine Learning.* 2 (1987). P. 319–342.
3. D. Angluin, L. Hellerstein, M. Karpinski. Learning read-once formulas with queries. *Journal of the ACM.* 40 (1993). P. 185–210.
4. D. V. Chistikov. Learning read-once functions using two least significant bits of the number of true points of projections. In: *Proc. of 3rd Russian workshop "Syntax and semantics of logical systems"* (*in Russian*). Irkutsk, izd. VSGAO (2010). P. 114–118.
5. D. V. Chistikov. Checking tests for read-once functions over arbitrary bases. In: Proc. CSR 2012, *Lecture Notes in Computer Science*, vol. 7353 (2012). P. 52–63.
6. V. A. Gurvich. On read-once Boolean functions. *Uspehi matematicheskih nauk* (*in Russian*). Vol. 32 (1977), 1. P. 183–184.
7. M. Karchmer, N. Linial, I. Newman, M. Saks, A. Widgerson: Combinatorial characterization of read-once formulae. *Discrete Mathematics.* Vol. 114 (1–3). 1995. P. 275–282.
8. P. Savicky, A. R. Woods. The number of Boolean functions computed by formulas of a given size. In: *Random Structures and Algorithms: Proceedings of the Eighth International Conference.* Vol. 13, 3–4 (1998). P. 349–382.
9. V. A. Stetsenko. On almost bad Boolean bases. *Theoretical Computer Science.* Vol. 136, 2. 1994. P. 419–469.
10. B. A. Subbotovskaya. On comparing bases for implementing Boolean functions with formulae. *Dokl. AN SSSR* (*in Russian*). 1963. Vol. 149, 4. P. 784–787.
11. L. G. Valiant. A theory of the learnable. *Communications of the ACM.* 27, 1984. P. 1134–1142.
12. A. A. Voronenko. On checking tests for read-once functions. *Matematicheskie voprosy kibernetiki* (*in Russian*). Vol. 11. Moscow, Fizmatlit (2002). P. 163–176.
13. A. A. Voronenko. On global testing (deciphering) problems for read-once Boolean functions. In: *Proc. of 3rd Russian workshop "Syntax and semantics of logical systems"* (*in Russian*). Irkutsk, izd. VSGAO (2010). P. 17–22.
14. A. A. Voronenko, D. V. Chistikov. Learning read-once functions using subcube parity queries. *Computational Mathematics and Modeling.* Vol. 22, 1 (2011). P. 81–91.