Isomorphism testing of read-once functions and polynomials

Raghavendra Rao B.V.[1] and Jayalal Sarma M.N.[2]

1 **Department of Computer Science, Saarland University**
`bvrr@cs.uni-saarland.de`

2 **Department of Computer Science and Engineering, Indian Institute of Technology Madras, Chennai, India**
`jayalal@cse.iitm.ac.in`

Abstract

In this paper, we study the isomorphism testing problem of formulas in the Boolean and arithmetic settings. We show that isomorphism testing of Boolean formulas in which a variable is read at most once (known as read-once formulas) is complete for log-space. In contrast, we observe that the problem becomes polynomial time equivalent to the graph isomorphism problem, when the input formulas can be represented as OR of two or more monotone read-once formulas. This classifies the complexity of the problem in terms of the number of reads, as read-3 formula isomorphism problem is hard for coNP.

We address the polynomial isomorphism problem, a special case of polynomial equivalence problem which in turn is important from a cryptographic perspective [19, 16]. As our main result, we propose a deterministic polynomial time canonization scheme for polynomials computed by constant-free read-once arithmetic formulas. In contrast, we show that when the arithmetic formula is allowed to read a variable twice, this problem is as hard as the graph isomorphism problem.

1998 ACM Subject Classification F.2.1 [Numerical Algorithms and Problems] – Computations on polynomials, F.1.3 [Complexity Measures and Classes], F.2.3 [Tradeoffs between Complexity Measures]

Keywords and phrases Computational Complexity, Boolean Isomorphism, Read Once Polynomials

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2011.115

1 Introduction

Computational isomorphism problems between various mathematical structures has intriguing computational complexity (see [6] for a survey). An important example, the Graph Isomorphism(GI) problem asks : given two graphs $G_1(V_1, E_1)$ and $G_2(V_2, E_2)$ decide if there is a bijection $\sigma : V_1 \to V_2$ such that $(u, v) \in E_1 \iff (\sigma(u), \sigma(v)) \in E_2$. This study becomes more important when the structures are computational models by themselves. Checking equivalence between programs is undecidable in general, but has useful special cases with respect to other computational models. We consider isomorphism testing of two important computational structures: Boolean and arithmetic circuits.

A Boolean formula (also known as an expression) is a natural non-uniform model of computing a Boolean function. The corresponding isomorphism question is to decide if the given boolean functions (input as formulas) are equivalent via a bijective transformation of the variables. This problem is known as the Formula isomorphism (FI for short) problem

31st Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011).
Editors: Supratik Chakraborty, Amit Kumar; pp. 115–126
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

in the literature. In general FI is in Σ_2^P (*i.e.* , the second level of the polynomial hierarchy), and unlikely to be Σ_2^P-hard unless the polynomial hierarchy collapses to the third level [2]. Goldsmith *et al.* [13] showed that FI for monotone formulas is as hard as general case. (See also [7, 11] for more results on the structure of FI.) Though it can be easily seen that FI is coNP-hard, an exact complexity characterization for FI is unknown to date.

This situation motivates one to look for special cases of FI that admit efficient algorithms. The number of reads of each variable in the formula is a restriction. A formula ϕ is not satisfiable if and only if it is isomorphic to the constant formula 0. By duplicating variables and introducing appropriate equivalence clauses, it follows that even when the number of reads is bounded by 3, FI(in CNF form) is coNP-hard.

We now address the intermediate cases; that is when the number of reads is bounded by 1 and 2 respectively. The first case, also known as read-once formulas, is a model that has received a lot of attention in the literature in various contexts, *e.g.,*[4] obtained efficient learning algorithms for read-once formulas. We show:

▶ **Theorem 1.** *Formula isomorphism for read-once formulas is complete for deterministic logarithmic space.*

However, the bound above seems to be tight. If we allow variables to be read twice, then FI becomes GI-hard even in the most primitive case:

▶ **Theorem 2.** *Isomorphism testing of OR of two monotone read-once DNF formulas is complete for* GI*.*

A natural analogue of the formula isomorphism question in the arithmetic world is about polynomials : given two polynomials $p(x_1, x_2, \ldots, x_n)$ and $q(x_1, x_2, \ldots, x_n)$ decide if there is a non-trivial permutation of the variables such that the polynomials are identical under the permutation. We denote this problem by PI. We assume that polynomials are presented in the form of arithmetic circuits in the non-black-box setting.

The polynomial isomorphism problem can also be seen as a special case of the well-studied polynomial equivalence problem (PE for short), where given two polynomials $p(x_1, x_2, \ldots, x_n)$ and $q(x_1, x_2, \ldots, x_n)$, decide if there is a non-singular matrix $A \in \mathbb{F}^{n\times n}$ such that $q(X) = p(AX)$, where $AX = (\sum_j A_{1,j}x_j, \ldots, \sum_j A_{n,j}x_j)$. The equivalence problem has survived intense efforts to give deterministic polynomial time algorithms (See [21, 16]). The lack of progress was explained by a result in [1], which reduces graph isomorphism problem to equivalence testing of cubic polynomials. The polynomial equivalence problem is expected to be very challenging and there are cryptographic schemes which are based on polynomial equivalence problems[19]. More recently Kayal [16, 15] developed efficient randomized algorithms for equivalence testing for several special classes of polynomials. Indeed, in the case of isomorphism problem, the matrix A is restricted to be a permutation matrix. Our next result shows that this specialization does not really simplify the problem when degree is 3, and in fact the polynomial isomorphism problem for a constant degree d polynomial also reduces to that of degree 3 polynomials.

▶ **Theorem 3.** *For any constant d, the polynomial isomorphism problem for degree d polynomials is polynomial time many-one equivalent to testing isomorphism of degree-3 polynomials, which in turn, is polynomial time many-one equivalent to* GI*.*

This shows that the polynomial isomorphism problem is also likely to be hard even when the polynomial is given explicitly listing down the monomials, and is harder than graph isomorphism problem. In general, we show that the isomorphism problem of polynomials is easier than the equivalence problem (over $\mathbb{Z}, \mathbb{Q}, \mathbb{R}, \mathbb{C}$).

▶ **Theorem 4.** PI *polynomial time many-one reduces to* PE*.*

A naive algorithm for this problem would be to guess the permutation and then verify whether the polynomials are the same under this permutation, which is an instance of the well-studied polynomial identity testing and can be solved in coRP. Thus the isomorphism testing problem is in MA. Indeed, the problem is also harder than the polynomial identity testing problem, because a polynomial is isomorphic to a zero polynomial if and only if it is identically zero. Thus, derandomizing the above MA algorithm in general to NP will imply circuit lower bounds[14]. From the above discussion it also follows that polynomial isomorphism problem is in NP if and only if polynomial identity testing is in NP. Also, Thierauf [24] showed that if PI(over $\mathbb{Q}$) is NP-hard then PH collapses to Σ_2^p.

This motivates looking at special cases of polynomial isomorphism problem for making progress. A *read-once polynomial* is a polynomial $f(X) \in \mathbb{Z}[x_1, \ldots, x_n]$ that can be computed by a read-once arithmetic formula. Read-once polynomials have been studied in various contexts in the literature. Bshouty et. al [10] developed efficient learning algorithms for read-once polynomials with membership and equivalence queries. (See also [8, 9].) More recently, Shpilka and Volkovich [22, 23] developed deterministic black-box sub exponential time algorithms for identity testing of read-once polynomials. In the non-black-box setting they give a polynomial time algorithm. We show the following for the isomorphism problem (which is harder than identity testing problem) as our main result.

▶ **Theorem 5.** *Isomorphism testing for constant-free read-once polynomials can be done in deterministic polynomial time.*

We then extend this to the case of arbitrary coefficients but still constant-free (see Theorem 14). As in the case of FI, we show that if we allow variables to be read twice then the polynomial isomorphism problem becomes GI-hard(see Theorem 16.). The structure of the rest of the paper is as follows. We introduce the basics and prove Theorem 4 in section 2. We prove Theorem 1 and 2 in section 3, and the main Theorem 5 in section 4.

2 Preliminaries

All the complexity theory notions used in this paper are standard. For more details, reader is referred to any standard complexity theory book. (See *e.g.,* [5].)

A Boolean formula ϕ is a directed acyclic graph, where out-degree of every node is bounded by 1, and the non-leaf nodes are labeled by $\{\vee, \wedge, \neg\}$ and the leaf nodes are labels by $\{x_1, \ldots, x_n, 0, 1\}$, where $x_1, \ldots, x_n$ are Boolean variables. Without loss of generality, we assume that ϕ has at most one node of out-degree zero, called the *output gate* of the formula. Naturally, with every formula ϕ, we can associate a Boolean function $f_\phi : \{0,1\}^n \to \{0,1\}$ defined as the function computed at the output node of the formula. A Boolean circuit is a generalization of formula wherein the out-degree of every node can be unbounded.

An arithmetic circuit C over a ring $\mathbb{F}$, is a directed acyclic graph where the non-leaf nodes are labeled by $\{+, \times\}$, and the leaf nodes are labeled by $\{x_1, \ldots, x_n\} \cup \mathbb{F}$, where $x_1, \ldots, x_n$ are variables that take values from $\mathbb{F}$, where $\mathbb{F}$ is a ring. In this paper we restrict our attention to cases where $\mathbb{F} \in \{\mathbb{Z}, \mathbb{R}, \mathbb{Q}\}$. Naturally, we can associate a polynomial $p_g \in \mathbb{F}[x_1, \ldots, x_n]$ with any gate g of the arithmetic circuit C. The polynomial computed by C is the polynomial associated with the output gate of C. An arithmetic formula is an arithmetic circuit where the out-degree of every node can be at most one.

A *read-once* formula (ROF for short), is a Boolean formula in which every variable x_i appears at most once as a leaf label, *i.e.* , every variable is read at most once. Similarly we can define read-once arithmetic formulas, *i.e.* , arithmetic formulas where a variable appears at most once. Polynomials computed by read-once arithmetic formulas are also known as *read-once polynomials* (ROPs for short).

A *constant-free* read-once arithmetic formula is a read-once arithmetic formula, where the only allowed leaf labels are x_i or $-x_i$. A constant-free ROP is a polynomial that can be computed by constant-free read-once arithmetic formula. A *general-constant-free* ROP is an ROP computed by arithmetic read-once formulas with the leaves labeled by $a_i x_i$, where $a_i \in \mathbb{Z} \setminus \{0\}$. For computational purposes, we assume that a constant-free ROP is given as a constant-free read-once formula in the input.

Now we define some notations that are used in Section 4. Let $C_1, \ldots, C_k$ denote a collection of ordered tuples. Then $\mathsf{sort}(C_1, \ldots, C_k)$ denotes the lexicographic sorted list of $C_1, \ldots, C_k$. For $k > 0$, Σ_k denotes the set of all permutations of a k element set. Let $S_1, \ldots, S_n \in \{0, 1\}$, then $\mathsf{parity}(S_1, \ldots, S_n) \stackrel{\triangle}{=} (\sum_{i=1}^n S_i \mod 2)$; and $\mathsf{binary}(S_1, \ldots S_n) \stackrel{\triangle}{=} \sum_{i=1}^n S_i 2^{n-i}$. For $a \in \mathbb{Z} \setminus \{0\}$, $\mathsf{sgn}(a) = 1$ if $a < 0$, and $\mathsf{sgn}(a) = 0$ otherwise.

Isomorphism testing problems : We now define the problems we address in the paper.

Formula Isomorphism(FI): Given two Boolean formulas $F_1(x_1, \ldots, x_n)$, and $F_2(x_1, \ldots, x_n)$ on n variables : $X = \{x_1, \ldots, x_n\}$, test if there exists a permutation $\pi \in S_n$, such that the functions computed by $F_1(x_1, \ldots, x_n)$ and $F_2(x_{\pi(1)}, \ldots, x_{\pi(n)})$ are the same.

Polynomial Isomorphism(PI): Given two polynomials $P, Q \in \mathbb{F}[x_1, \ldots, x_n]$, test if there exists a permutation $\pi \in S_n$ such that $P(x_1, \ldots, x_n) = Q(x_{\pi(1)}, \ldots, x_{\pi(n)})$. $\mathrm{PI}_d(\mathbb{F})$ denotes the special case when P, and Q are of degree at most d. A notion related to isomorphism is *canonization.* A *canonical code* for polynomials is a function $\mathcal{C} : \mathbb{F}[x_1, \ldots, x_n] \to \{0, 1\}^*$ such that $\mathcal{C}(f) = \mathcal{C}(g)$ if and only if the polynomials f and g are isomorphic.

Polynomial Equivalence(PE): Given two polynomials $P, Q \in \mathbb{F}[x_1, \ldots, x_n]$, test if there is a non-singular matrix $A = (a_{i,j}) \in GL(n, \mathbb{F})$ such that the polynomials $P(x_1, \ldots, x_n) = Q(y_1, \ldots, y_n)$ where $y_1, \ldots, y_n$ are obtained by applying the linear transformation defined by the row-vectors of A, *i.e.* , $y_i = \sum_{j=1}^n a_{i,j} x_j$.

In general we assume that the input polynomials are given as arithmetic circuits. PE_d denotes the restriction of PE where the input polynomials are of degree d. (See [21] for a detailed exposition on this problem). The following equivalence was proved in [21].

▶ Proposition 6 ([21, 20]). GI poly time many-one reduces to PE_3.

In general, though PI is a special case of PE where A is restricted to be a permutation matrix, it is unclear a priori whether PI is easier than PE. We give a reduction from PI to PE over $\mathbb{Z}, \mathbb{Q}, \mathbb{R}$, and $\mathbb{C}$, this proves Theorem 4.

Proof of Theorem 4: Let $f(X)$ and $g(X)$ be the two polynomials given as an input instance of PI, where $X = \{x_1, \ldots, x_n\}$. Let $d = \max\{\deg(f), n\}$, $m > \max\{2n, n + d + 4\}$, such that $\mathsf{g}cd(m - 2n, d + n + 4) = 1$, and $X' = X \cup \{y, z\}$. Define

$$\begin{aligned} f'(X, y, z) &\stackrel{\triangle}{=} f(X) + y^{d+1} x_1 \cdots x_n + z^{d+n+2}(x_1 + \cdots + x_n) + z^{d+n+4} + y^{d+1} z^m; \quad \text{and} \\ g'(X, y, z) &\stackrel{\triangle}{=} g(X) + y^{d+1} x_1 \cdots x_n + z^{d+n+2}(x_1 + \cdots + x_n) + z^{d+n+4} + y^{d+1} z^m \end{aligned}$$

Suppose $f(X) \cong g(X)$, then clearly $f'(X') \cong g'(X')$. Suppose $f'(X') = g'(A'X')$ for some non-singular matrix A'. We claim that A' has to be a permutation matrix. By the degree conditions, A' sends y to by , and z to cz, where $c^{d+n+4} = 1$, and $b^{d+1} c^m = 1$. Also note that y and z both have zero coefficients in $A'x_i$ for all i, by the unique factorization of $x_1 \cdots x_n$. Similarly, for all i, $A'x_i$ cannot have two non-zero coefficients, again by the degrees of y and z, and the unique factorization of $x_1 \cdots x_n$. The only possibility is, A' could be the product of a permutation matrix P and a diagonal matrix D with determinant equal to 1. Let the i th entry in the diagonal D be λ_i. Then, $z^{d+n+2}A'x_i$ will have coefficient $\lambda_i c^{d+n+2}$, but in the target polynomial f', it has coefficient 1, so $\lambda_i = c^2$. This implies $b^{d+1}c^{2n} = 1$, and hence $c^{m-2n} = 1$. As $\mathsf{gcd}(m-2n, d+n+4) = 1$, $c = 1$, and hence $b = 1$, $\lambda_i = 1$ $1 \leq i \leq n$. Thus, $f(X) \cong g(X)$ if and only if the polynomials $f'(X')$ and $g'(X')$ are equivalent. Note that $f'(X')$ can be computed by a circuit of size $s + 4d + 4n + m + 9$, where s is the size of a circuit computing $f(X)$. This completes the proof.

3 Isomorphism testing of Boolean read-once formulas

For a Boolean read-once formula ϕ, let $G(\phi) = (V_\phi, E_\phi)$ denote the formula graph of ϕ as defined in [4], *i.e.* , $V_\phi = \{x_1, \ldots, x_n\}$, and $E_\phi = \{(x_i, x_j) \mid \mathsf{LCA}(x_i, x_j) \text{ is labeled } \wedge\}$, where $\mathsf{LCA}(x, y)$ denotes the least common ancestor of the leaves labeled x and y in ϕ.

3.1 Logspace characterization : Proof of Theorem 1

Proof. We first argue the upper bound. We argue for the special case of monotone read-once formulas. Let ϕ_1 and ϕ_2 be two minimal monotone read-once formulas. First observe that $G(\phi_1) \cong G(\phi_2) \iff \phi_1 \cong \phi_2$. Let F_1 (resp. F_2) be the minimum read-once formula computing the same function as ϕ_1 (resp. ϕ_2) by merging consecutive gates of the same type into one gate of larger fan-in. Construct two trees T_1 and T_2 from F_1 and F_2 respectively as follows. We describe the construction for T_1. Treat the formula F_1 as a undirected tree with $\wedge$ gates colored as RED, $\vee$ gates colored as BLUE and the leaf nodes colored as GREEN.

▶ Claim 1. $G(\phi_1) \cong G(\phi_2) \iff T_1 \cong T_2$.

Assuming the claim, testing whether $\phi_1 \cong \phi_2$ is equivalent to isomorphism testing of colored trees. As the latter can be done in deterministic logarithmic space [18], it is enough to prove the claim.

Proof of the claim. ($\Rightarrow$) Suppose $G(\phi_1) \cong G(\phi_2)$, and σ be such a bijection between the vertices of $G(\phi_1)$ and $G(\phi_2)$. Fix the corresponding map between the leaves of T_1 and T_2. For any two leaves x, y of T_1, let $\mathsf{LCA}(x, y)$ denote the least common ancestor of x and y in T_1. Colors and degrees of $\mathsf{LCA}(x, y)$ and $\mathsf{LCA}(\sigma(x), \sigma(y))$ are the same. (This follows from the property of the graphs $G(\phi_1)$ and $G(\phi_2)$.) So σ induces a color-preserving isomorphism between T_1 and T_2.

($\Leftarrow$) Let σ be a color preserving isomorphism between T_1 and T_2. Let π denote the corresponding bijection between the leaves of T_1 and T_2 induced by σ. It is sufficient to argue that $G(\phi_1) = \pi(G(\phi_2))$. Consider two variables x and y. As $\mathit{color}(\mathsf{LCA}(x, y)) = \mathit{color}(\mathsf{LCA}(\pi(x), \pi(y)))$, we have $(x, y) \in E(G(\phi_1)) \iff (\pi(x), \pi(y)) \in E(G_2)$. This completes the proof of the Claim. ◀

The argument above can be extended to the non-monotone case by coloring the leaves of T_f (resp. T_g) that correspond to positive literals as YELLOW, and those corresponding to negative literals as RED.

Now we argue the L-hardness. We reduce directed forest reachability (which is known to be L-complete[12]) to FI. Given the instance (G, s, t) of directed forest reachability where the task is to check if there is a directed path from s to t, we construct the formula(F) as follows. Ignore the incoming edges to s and outgoing edges from t. Replace s with a variable x and label every other leaf node with the constant 1. Replace all intermediate nodes by $\wedge$ gates. Label t as the output node. Since G is a directed forest, F will be a formula and is a read-once formula by construction. Moreover, F will evaluate to x (and hence isomorphic to the trivial formula x) if and only if there is a directed path from s to t. ◀

3.2 Larger Number of Reads : Proof of Theorem 2

Naturally, one could hope to extend theorem 1 to boolean formulas that read a variable at most a constant number of times. Surprisingly, it turns out that if the input formulas are represented as OR of two monotone read-once formulas, then isomorphism testing becomes GI hard.

▶ **Lemma 7.** GI *polynomial time many-one reduces to testing isomorphism of OR of two monotone read-once formulas given in DNF form.*

Proof. The reduction is from GI for bipartite graphs which is as hard as the general GI[17]. For a simple undirected bipartite graph $G = (U, V, E)$, define a formula $\phi(G)$ on variables $\{x_e \mid e \in E\}$ as follows. For every $v \in U \cup V$, $\phi(G)$ contains the term $x_{e_1} \wedge x_{e_2} \wedge \ldots \wedge x_{e_\ell}$ as a minterm, where $e_1, e_2, \ldots, e_\ell$ are the edges that are incident on v in G. *i.e.* ,

$$\phi_G = \bigvee_{v \in U \cup V} \bigwedge_{e \text{ incident on } v} x_e = (\bigvee_{u \in U} \bigwedge_{e \text{ incident on } u} x_e) \vee (\bigvee_{v \in V} \bigwedge_{e \text{ incident on } v} x_e) \tag{1}$$

So $\phi(G)$ can be written as an OR of two monotone read-once formulas. , $G_1 \cong G_2 \iff \phi(G_1') \cong \phi(G_2')$. This concludes the proof. ◀

Observe that a monotone boolean formula ϕ given in DNF form, can also be represented as a bipartite graph with vertices of one side corresponding to variables of ϕ and the terms of ϕ as vertices on the other side, edge relations is defined with respect to inclusion. Combined with Lemma 7, this proves Theorem 2.

4 Isomorphism testing of Read-once polynomials

As starting point, observe that the deterministic polynomial identity testing algorithm for read-once formulas [22] gives an NP upper bound for isomorphism testing of read-once polynomials. A natural question is to see if the NP upper bound above can be improved to a polynomial time algorithm. In the following, we provide a polynomial time algorithm for the isomorphism testing of certain special classes of read-once polynomials. We begin with the toy case of monotone read-once polynomials f such that $f(0) = 0$.

▶ **Lemma 8.** *Isomorphism testing of monotone read-once polynomials that can be computed by monotone read-once arithmetic formulas with leaves labeled from $\{x_1, \ldots, x_n\}$, can be done in deterministic logarithmic space.*

Proof. Let f be a monotone read-once polynomial computed by a monotone read-once formula ϕ_f. Without loss of generality assume that ϕ_f is in the minimal form, i.e., inputs of a $\times$ gate are either $+$ gates or variables and that of a $+$ gate are either $\times$ gates or variables. Let $G_f = (V_f, E_f)$ be the undirected graph with $V_f = \{x_1, \ldots, x_n\}$ and $E_f = \{(x_i, x_j) \mid \mathsf{LCA}_{\phi_f}(x_j, x_j) \text{ is a } \times \text{ gate}\}$. By the definition of G_f, a monomial $M = \prod_{j=1}^{k} x_{i_j}$ has coefficient 1 in f if and only if the vertices $x_{i_1}, \ldots, x_{i_k}$ form a maximal clique in G_f. Let T_f denote the underlying (undirected) tree of ϕ_f, where a node corresponding to a $+$ gate is colored BLUE and that corresponding to a $\times$ gate is colored RED.

Let f and g be two monotone read-once formulas as above. Clearly, $f \cong g \iff G_f \cong G_g$. Now we show that $G_f \cong G_g$ if and only if T_f is isomorphic to T_g as a colored tree.

Suppose $T_f \cong T_g$ via a bijection π between the vertices of T_f and T_g. Let σ be the bijection between the leaves of T_g and T_f induced by π. Then $\forall i \neq j$, $\mathsf{LCA}_{\phi_f}(x_i, x_j)$ is a $\times$ gate if and only if $\mathsf{LCA}_{\phi_g}(x_{\sigma(i)}, x_{\sigma(j)})$ is a $\times$ gate. So σ defines an isomorphism between G_f and G_g.

For the converse direction, suppose $f \cong g$. The proof is by induction on the structure of f and g. The base case is when f and g are single variables, in which case the claim follows. There are two cases:

Case 1: f and g can be written uniquely as $f = f_1 + \ldots + f_k$, $g = g_1 + \ldots + g_k$, where $f_i's$ and $g_i's$ cannot be written as sum of two or more variable disjoint monotone ROP's. Then, $f \cong g$ if and only if there is a permutation $\sigma \in \Sigma_k$ such that $f_i \cong g_{\sigma(i)} \iff G_{f_i} \cong G_{g_{\sigma(i)}} \iff T_{f_i} \cong T_{g_{\sigma(j)}}$, where the last equivalence is available from induction. So, $T_f \cong T_g$.

Case 2: $f = f_1 \times \ldots \times f_k$ and $g = g_1 \times \ldots \times g_k$, where $f_i's$ and $g_i's$ cannot be decomposed into products of two or more variable disjoint ROPs. Then, $f \cong g$ implies there is a permutation $\sigma \in \Sigma_k$ such that $f_i \cong g_{\sigma(i)}$. By induction. This implies $T_{f_i} \cong T_{g_{\sigma(i)}}$, which in turn implies $T_f \cong T_g$. Now the algorithm is obvious: given f and g, compute T_f, and T_g. Then test if $T_f \cong T_g$, using the log-space algorithm for testing isomorphism for trees [18]. ◀

Our goal now is to extend Lemma 8 to the case of non-monotone read-once polynomials. Consider a constant-free read-once formula, *i.e.* , a read-once formula where a leaf is labeled from $\{-x_i, x_i\}$ for some i. An obvious approach would be to use Lemma 8 with an additional coloring of -ve terms. Then the two representations: $f = f_1 \times f_2 \times \ldots \times f_k$, and $g = (-f_1) \times (-f_2) \times f_3 \times \ldots \times f_k$ will give rise to two non-isomorphic trees whereas f and g are identical polynomials.

We overcome this by building a canonical code for general constant-free read-once polynomials along the lines of the well-known tree canonization algorithm [3]. Recall that a canonical code for a polynomial is an object that is unique for every isomorphism class. Also, note that efficient computation of canonical code for a class of polynomials implies efficient algorithm for isomorphism testing for that class, though the converse may not be true in general. For ease of exposition, we give details for the case of constant-free ROPs. We first observe some simple structural properties of constant-free read-once polynomials that serves as a foundation for our construction of canonization.

▶ Proposition 9. A constant-free read-once polynomial $f \neq 0$ has the following recursive structure:

- $f = a_i x_i$, where $a \in \{-1, 1\}$; or
- f is of Type-1, i.e., $f(X) = f_1(X_1) + f_2(X_2) + \ldots + f_k(X_k)$ for a unique $k \geq 2$, where f_i's are constant-free variable disjoint read-once polynomials and $X = X_1 \uplus X_2 \uplus \ldots \uplus X_k$. Also, f_i cannot be written as a sum of two or more variable disjoint constant-free ROPs; or
- f is of Type-2, i.e, $f(X) = f_1(X_1) \times f_2(X_2) \times \ldots \times f_t(X_t)$ for a unique $t \geq 2$, where f_i's are constant-free variable disjoint read-once polynomials and $X = X_1 \uplus X_2 \uplus \ldots \uplus X_t$. Also, f_i cannot be written as a product of two or more constant-free variable disjoint ROPs.

The following structural characterization of constant-free ROPs follows from Proposition 9.

▶ **Lemma 10.** *(a) If f, g are constant-free ROPs of Type-1, i.e. , $f = f_1 + \ldots + f_k$, $g = g_1 + \ldots + g_k$, where $f_i s$ and $g_i s$ are constant-free ROPs of Type-2. Then, $f \cong g \iff \exists \sigma \in \Sigma_k \;\; f_i \cong g_{\sigma(i)}$.*

(b) If f and g are constant-free ROPs of Type-2, i.e. , $f = f_1 \times \ldots \times f_k$, and $g = g_1 \times \ldots \times g_k$, where f_is and g_is are constant-free ROPs of Type-1. Then,

$$f \cong g \iff \begin{Bmatrix} \exists \sigma \in \Sigma_k, \text{ and } a_1, \ldots, a_k \in \{-1,1\} \text{ such that } f_i \cong a_{\sigma(i)} g_{\sigma(i)} \\ \text{and } \mathsf{parity}(a_1, \ldots, a_k) = 0. \end{Bmatrix}$$

Proof. For (a), suppose $f = f_1 + \ldots + f_k$, and $g = g_1 + \ldots + g_k$. As f_is (resp. g_is) are variable disjoint, there is no cancellation of monomials of f_is in f, *i.e.* , every monomial appearing in f_i also appears in f with the same coefficient as in f_i. Since each of the f_i's (and g_i's) cannot be written as a sum of two or more variable disjoint constant-free ROPs we have (a). For (b), note that converse direction is clear as $f_1, ..., f_k$ are variable disjoint. Suppose $f \cong g$, via a permutation σ of the variables. Then, $\sigma(f) = \sigma(f_1) \times \sigma(f_2) \times \ldots \times \sigma(f_k) = g_1 \times g_2 \times \ldots \times g_k$. Then, as $\sigma(f_1), \ldots, \sigma(f_k)$ are variable disjoint, there is a $\pi \in \Sigma_k$ such that $\sigma(f_i) = g_{\pi(i)}$ or $\sigma(f_i) = -g_{\pi(i)}$, and $\{i \mid \sigma(f_i) = -g_{\pi(i)}\}$ is even. ◀

A canonization for constant-free ROPs

Combining Lemma 10 with the standard canonization for trees [3], we propose a polynomial time canonization scheme for constant-free read-once polynomials.

We start with an informal description of code. As a toy example consider a linear polynomial f with coefficients in $\{-1, 1\}$. Let N_f be the number of variables with -ve coefficients and P_f be those with +ve coefficients. Clearly, a linear polynomial g is isomorphic to f if and only if $P_g = P_f$ and $N_g = N_f$. So, P_f, and N_f are the canonical values of f that are invariant under permutation of variables. Similarly, if $f = \prod_{i=1}^{k} a_i x_i$ with $a_i \in \{-1, 1\}$, then any $g = \prod_{i=1}^{k} b_i x_i$ is isomorphic to f if and only if the parity of the number of negative coefficients of g is equal to that of f. So, the number of variables, and the parity of the number of -ve coefficients would be an invariant set for f under permutations of variables.

By Lemma 10, if f is of Type-1, *i.e.* ,$f = f_1 + \ldots + f_k$, then any constant-free ROP isomorphic to f, will look like a permutation of f_is. So, a canonization of f would be a sorted ordering of those for f_is. If f is of Type-2, *i.e.* ,$f = f_1 \times \ldots \times f_k$, then, the canonization of f should be invariant when an even number of f_is are multiplied by -1. We handle these constraints by building the canonization for f, denoted by $\mathsf{code}(f)$ in a bottom-up fashion depending on the structure of the constant-free read-once arithmetic formula computing f.

For a constant-free read-once formula f, $\mathsf{code}(f)$ is a quadruple (C, P, N, S), where C is a string, $P, N \in \mathbb{N}$, and $S \in \{0, 1\}$. Here C stores information about the read-once polynomials computed by the sub-formulas at the root gate of the arithmetic formula computing f. The values of P, N, and S depend on the type of f (as in Proposition 9). If f is of Type-1, then $S = 0$, and N intuitively represents the number of "negative" polynomials f_i, and $P = k - N$. When f is of Type-2, $P = N = 0$, and S in some sense represents the parity of the number of "negative" polynomials in $f_1, \ldots, f_k$, where $f = f_1 \times \cdots \times f_k$. Here the term "negative" is used in a tentative sense.

Now we formally define code via induction based on the structure of f as given by Proposition 9. Abusing the notation we use the symbol $\emptyset$ also to denote empty string.

We consider the following four base cases:

base case 1: $f = x_i$, then $\mathsf{code}(f) = (\emptyset, 0, 1, 0)$.

base case 2: $f = -x_i$, then $\mathsf{code}(f) = (\emptyset, 1, 0, 0)$.

base case 3: $f = \sum_{i=1}^{k} a_i x_i$, for some $k > 2$, and $a_i \in \{-1, 1\}$. Let $C_i = \mathsf{code}(a_i x_i)$. Let $i_1, \ldots, i_k$ be such that $C_{i_1}, \ldots, C_{i_k}$ represents the lexicographical sorting of $C_1, \ldots, C_k$. Let $S_i = \mathsf{sgn}(a_i)$. Let $N = \mathsf{binary}(S_{i_1}, \ldots, S_{i_k})$, and $P = \mathsf{binary}(\bar{S}_{i_1}, \ldots, \bar{S}_{i_k})$. Then

$$\mathsf{code}(f) \triangleq ((\langle \emptyset, 0, 1 \rangle, \overset{\text{k times}}{\ldots} \langle \emptyset, 0, 1 \rangle), N, P, 0)$$

base case 4: $f = \prod_{i=1}^{k} a_i x_i$, $a_i \in \{-1, 1\}$. Let $S = 1$, if the number of -1's in $a_1, \ldots, a_k$ is odd, and $S = 0$ otherwise. Define

$$\mathsf{code}(f) \triangleq ((\langle \emptyset, 0, 1\rangle, \overset{\text{k times}}{\ldots} \langle \emptyset, 0, 1\rangle), 0, 0, S)$$

Inductively, assume that, $\mathsf{code}(g) = (C, N, P, 0)$ for a constant-free ROP g of Type-1 on at most $n-1$ variables, and $\mathsf{code}(g) = (C, 0, 0, S)$ for a constant-free ROP g of Type-2 in at most $n-1$ variables. Consider a constant-free ROP f on n variables. By Proposition 9, there are two cases

Type 1: Let $f = f_1 + f_2 + \ldots f_k$, where $f_1, \ldots, f_k$ are constant-free ROPs of Type-2. By induction, suppose $\mathsf{code}(f_i) = (C_i, 0, 0, S_i)$. If $f_i = ax_{j_i}$ for some $1 \leq j_i \leq n$, and $a \in \{-1, 1\}$ then we need to take $\mathsf{code}(f_i) = (\langle \emptyset, 0, 1\rangle, 0, 0, \mathsf{sgn}(a))$. Let $\langle C_{i_1}, \ldots, C_{i_k}\rangle = \mathsf{sort}(C_1, \ldots, C_k)$, $N = \mathsf{binary}(S_{i_1}, \ldots, S_{i_k})$, and $P = \mathsf{binary}(\bar{S}_{i_1}, \ldots, \bar{S}_{i_k})$. Then,

$$\mathsf{code}(f) \triangleq (\langle C_{i_1}, \ldots, C_{i_k}\rangle, N, P, 0) \tag{2}$$

Type 2: $f = f_1 \times f_2 \times \ldots \times f_k$, where $f_1, \ldots, f_k$ are constant-free ROPs of Type-1. By induction, suppose $\mathsf{code}(f_i) = (C_i, N_i, P_i, 0)$. Let $N'_i = \min\{N_i, P_i\}$, and $P'_i = \max\{N_i, P_i\}$. Let $\tilde{C_i} = \langle C_i, N'_i, P'_i\rangle$ and $\langle \tilde{C_{i_1}}, \ldots, \tilde{C_{i_k}}\rangle$ be the lexicographically sorted sequence of $\tilde{C_i}$'s, $S = |\{i \mid N'_i \neq N_i\}| \mod 2$. Then,

$$\mathsf{code}(f) = (\langle \tilde{C_{i_1}}, \ldots, \tilde{C_{i_k}}\rangle, 0, 0, S) \tag{3}$$

The following lemma describes some of the properties of the function code.

▶ **Lemma 11.** *(a) Let $f_1, \ldots, f_k$ be constant-free ROPs of Type-1, $a_1, \ldots, a_k, b_1, \ldots, b_k \in \{-1, 1\}$. Then*

$$\mathsf{code}(\prod_{i=1}^{k} a_i f_i) = \mathsf{code}(\prod_{i=1}^{k} b_i f_i) \iff \quad \mathsf{parity}(\mathsf{sgn}(a_1), \ldots, \mathsf{sgn}(a_k)) = \mathsf{parity}(\mathsf{sgn}(b_1), \ldots, \mathsf{sgn}(b_k)).$$

(b) code$(-\prod_{i=1}^{k} f_i) = (C, 0, 0, \bar{S})$, where $\mathsf{code}(\prod_{i=1}^{k} f_i) = (C, 0, 0, S)$.

(c) Let $f_1, \ldots, f_k$ be ROPs of Type-2 and suppose $\mathsf{code}(\sum_{i=1}^{k} f_i) = (C, N, P, 0)$. *Then* $\mathsf{code}(-\sum_{i=1}^{k} f_i) = (C, P, N, 0)$.

Proof. Proof is by induction on the number of variables in the constant-free read-once formula f. We consider two base cases. Let $f = \prod_{i=1}^{k} x_k$. Then by base case 4 in the definition of code,

$$\mathsf{code}(-f) = \big((\langle \emptyset, 1, 0\rangle, \ldots, \langle \emptyset, 0, 1\rangle), 0, 0, \bar{S}\big)$$

(a), (b) follow immediately now, and (c) is not relevant for this case. The second base case is when $f = \sum_i a_i x_i$. Note that only (c) is relevant here. Then $-f = \sum_i -a_i x_i$, and hence $\mathsf{code}(-f) = (C, P, N, 0)$, where $\mathsf{code}(f) = (C, N, P, 0)$. This proves (c) for the second base case.

Inductively suppose that statements (a)-(c) hold for all constant-free ROPs on $n' \leq n-1$ variables. Let f be a constant-free ROP of Type-2 on n variables, *i.e.* , $f = \prod_{i=1}^{k} f_i$, where f_is are constant-free ROPs of Type-1. Then, for $a = (a_1, \ldots, a_k) \in \{-1, 1\}^k$, $f_a = \prod_{i=1}^{k} a_i f_i$ is also a constant-free ROP of Type-2 on n variables. Suppose $\mathsf{code}(f_i) = (C_i, N_i, P_i, 0)$ for $1 \leq i \leq k$, then by (3),

$$\mathsf{code}(f) = ((\langle C_1, N'_1, P'_1\rangle, \ldots, \langle C_1, N'_k, P'_k\rangle), 0, 0, S).$$

By (c) of the induction hypothesis, we have $\mathsf{code}(-f_i) = (C_i, P_i, N_i, 0)$. Then, applying the construction given by (3),

$$\mathsf{code}(f_a) = (\mathsf{sort}(\langle C_1, N'_1, P'_1\rangle, \ldots, \langle C_k, N'_k, P'_k\rangle), 0, 0, S_a)$$

with $S_a = S$, if $\mathsf{parity}(\mathsf{sgn}(a_1), \ldots, \mathsf{sgn}(a_k)) = 0$, and $S_a = \bar{S}$ otherwise. This proves (a) and (b).

To prove (c), suppose f is a constant-free ROP of Type-1 on n variables, i.e., $f = \sum_{i=1}^{k} f_i$. Suppose $\mathsf{code}(f_i) = (C_i, 0, 0, S_i)$, and $\langle C_1, \ldots, C_k\rangle$ be the lexicographically sorted order of C_i's, without loss of generality. Then, by the definition of code given in (2), $\mathsf{code}(f) = ((C_1, \ldots, C_k), N, P, 0)$, where $N = \mathsf{binary}(S_1, \ldots, S_k)$, and $P = \mathsf{binary}(\bar{S_1}, \ldots, \bar{S_k})$. Applying induction hypothesis (b) on f_i, $\mathsf{code}(-f_i) = (C_i, 0, 0, \bar{S_i})$. As $-f = \sum_{i=1}^{k} -f_i$, by (2) and the induction hypothesis, we have

$$\begin{aligned}
\mathsf{code}(f) &= (\langle C_1, \ldots, C_k\rangle, \tilde{N}, \tilde{P}, 0) \text{ where} \\
\tilde{N} &= \mathsf{binary}(\bar{S_1}, \ldots, \bar{S_k}) \text{ and} \\
\tilde{P} &= \mathsf{binary}(S_1, \ldots, S_k)
\end{aligned}$$

This implies $P = \tilde{N}$, and $N = \tilde{P}$, and hence (c) follows. ◀

Using these properties we prove that code is indeed a canonization for constant-free ROPs.

▶ **Lemma 12.** *Let f, and g be two constant-free ROPs. Then, $f \cong g \iff \mathsf{code}(f) = \mathsf{code}(g)$*

Proof. Proof is by induction on the structure and number of variables in f and g. For base Case, $f = \pm x_i, \sum_{i=1}^{k} a_i x_i$, or $\prod_{i=1}^{k} a_i x_i$, where $a_i \in \{-1, 1\}$. By examining the four base cases in the definition of code, the Lemma follows for these cases. For the induction step, we consider two cases depending on whether f is of Type-1 or Type-2.

Type 1: Let $f = f_1 + \ldots + f_k$ and $g = g_1 + \ldots + g_k$. First suppose $f \cong g$ via a bijection ϕ between the variables of f and g. As f_i's are variable disjoint, there exists a $\sigma \in \Sigma_k$ such that $\phi(f_i) = g_{\sigma(i)}$, and hence $f_i \cong g_{\sigma(i)}$, and by induction hypothesis, we have $\mathsf{code}(f_i) = \mathsf{code}(g_{\sigma(i)}) = (C_i, 0, 0, S_i)$. By (2), we can conclude that $\mathsf{code}(f) = \mathsf{code}(g)$. For the converse direction, suppose that $\mathsf{code}(f) = \mathsf{code}(g)$. Let $\mathsf{code}(f) = (\langle C_1, \ldots, C_k\rangle, \mathsf{binary}(S_1 \ldots S_k), \mathsf{binary}(\bar{S_1}, \ldots, \bar{S_k}), 0) = \mathsf{code}(g)$. Then by the structure of $\mathsf{code}(g)$ as in (2), we conclude $\mathsf{code}(f_i) = (C_i, 0, 0, S_i) = \mathsf{code}(g_i) \implies f_i \cong g_i$ (by induction hypothesis). Then, we have $g \cong f$ by Lemma 10.

Type 2: Let $f = f_1 \times f_2 \times \ldots f_k$ and $g = g_1 \times g_2 \times \ldots \times g_k$. Let $\mathsf{code}(f) = (C, 0, 0, S)$, and $\mathsf{code}(g) = (D, 0, 0, R)$, where $C = (\langle C_1, N'_1, P'_1\rangle, \ldots, \langle C_k, N'_k, P'_k\rangle)$, and $D = (\langle D_1, L'_1, M'_1\rangle, \ldots, \langle D_k, L'_k, M'_k\rangle)$. Suppose $\mathsf{code}(g) = \mathsf{code}(f)$. Then, by the definition of code, and Lemma 11, we have $\forall i \in [k]$, either $\mathsf{code}(f_i) = \mathsf{code}(g_i)$ or $\mathsf{code}(f_i) = \mathsf{code}(-g_i)$, and hence either $f_i \cong g_i$ or $f_i \cong -g_i$. As $S = R$, $|\{i \mid \mathsf{code}(f_i) = \mathsf{code}(-g_i)\}|$ must be even. Then by Lemma 10, we have $f \cong g$. For the converse direction, suppose $f \cong g$. Then by Lemma 10, there is a $\sigma \in \Sigma_k$, and $a_1, \ldots, a_k \in \{-1, 1\}$ with $\mathsf{parity}(\mathsf{sgn}(a_1), \ldots, \mathsf{sgn}(a_k)) = 0$ such that $f_i \cong a_i g_{\sigma(i)}$, and hence $\mathsf{code}(f_i) = \mathsf{code}(a_i g_{\sigma(i)})$. Then, by the definition of code, we have $\mathsf{code}(\prod_{i=1}^{k} f_i) = \mathsf{code}(\prod_{i=1}^{k} a_i g_{\sigma(i)})$. As $\mathsf{parity}(\mathsf{sgn}(a_1), \ldots, \mathsf{sgn}(a_k)) = 0$, by Lemma 11, $\mathsf{code}(\prod_{i=1}^{k} a_i g_{\sigma(i)}) = \mathsf{code}(\prod_{i=1}^{k} g_i)$, which completes the proof. ◀

▶ **Theorem 13.** *Isomorphism testing of constant-free read-once polynomials can be done in time polynomial in the number of variables in the input formulas.*

Proof. Given Lemma 12, the algorithm is obvious: on input f and g, compute $\mathsf{code}(f)$ and $\mathsf{code}(g)$, then check if $\mathsf{code}(f) = \mathsf{code}(g)$. Given f as an arithmetic constant-free read-once formula, $\mathsf{code}(f)$ can be computed in time polynomial in the size of the input formula. As size of $\mathsf{code}(.)$ as a collection of sets is at most the size of the input formula, we can test if $\mathsf{code}(f) = \mathsf{code}(g)$ in time linear in the size of the input formulas f and g. ◀

Extension to constant-free ROPs with arbitrary coefficients: The function code defined for constant-free ROPs can be extended to include constant-free ROPs where leaf nodes are labeled with $a_i x_i$, where $a_i \in \mathbb{Z}$. We denote this extension by general-constant-free ROPs. There is one main bottleneck for general-constant-free ROPs of Type-2: suppose $f = f_1 \times \cdots \times f_k$, and if a_1 is the GCD of coefficients of f_1, then $f = (f_1/a_1)(a_1 f_2) \times \cdots \times f_k$. So, a canonical code has to be invariant under taking out GCD of the coefficients of some of the f_i's and multiplying out these values among the remaining f_j's, provided the polynomial remains general-read-once. This can be achieved by explicitly carrying the GCD of the coefficients. For the sake of notational convenience, we denote the canonical function for general-constant-free ROPs by code'. For a general-constant-free ROP f, we define $\mathsf{code}'(f)$ as a quintuple (C, N, P, S, α), where C is a string, $N, P, \alpha \in \mathbb{N}$, $S \in \{0, 1\}$. The values C, P, N, and S have the same meaning as in the definition of code, and α is the GCD of the coefficients of f. We generalize the properties of code (details are skipped) to show:

▶ **Theorem 14.** *Isomorphism testing of general-constant-free ROPs can be done in* P.

Extension to Pre-processed ROPs: Motivated by [23], we extend Theorem 14 to the case of pre-processed constant-free ROPs. (See [23] for more on pre-processed ROPs). In a pre-processed constant-free ROP, a leaf labeled by variable x_i computes an arbitrary univariate polynomial $f_i(x_i)$ with coefficients from $\mathbb{Z}$. By providing an efficient way of canonically encoding the univariate polynomials that appear at the leaves, we obtain the following:

▶ **Theorem 15.** *Canonization of a pre-processed general constant-free ROP can be done in time polynomial in the number of variables and the degree of the univariate polynomials at the leaves.*

5 Polynomials with higher reads

As a natural extension, one could ask if the canonization procedure presented in the previous section can be extended to arithmetic formulas that read a variable at most twice. However, as in the case of Boolean formulas, it turns out that allowing variables twice makes the problem as hard as GI. In fact, even for the most primitive classes of Read-2 polynomials 1) Sum of two depth two monotone ROPs and, 2) Read-2 polynomials given in the $\Pi\Sigma$ form, isomorphism testing is complete for GI. It is already known that PI is harder than GI[24, 16]). We provide a different reduction that optimizes the number of reads. We skip some details here.

▶ **Theorem 16.** GI *polynomial time many-one reduces to testing isomorphism of two polynomials when both of the polynomials are given in one of the following representations:*

(a) Sum of two monotone read-once depth-2 arithmetic formulas with a $+$ gate at the top.
(b) Read-2 monotone arithmetic formulas of depth two, with a $\times$ gate at the top.

Hardness of PI for the above special cases also forces one to ask whether the hardness given by Proposition 16 extends to the polynomial equivalence (PE) problem. Though we do not know the exact answer, we observe that PE for read-4 polynomials is hard for GI.

▶ Proposition 17. PE for the case of read-4 polynomials is hard for GI, for $\mathbb{F} \in \{\mathbb{Z}, \mathbb{Q}, \mathbb{R}\}$.

References

1 M. Agrawal and N. Saxena. Equivalence of f-algebras and cubic forms. In *STACS*, pages 115–126, 2006.

2 M. Agrawal and T. Thierauf. The Formula Isomorphism Problem. *SIAM J. Comput.*, 30(3):990–1009, 2000.

3 A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesly, 1974.

4 D. Angluin, L. Hellerstein, and M. Karpinski. Learning read-once formulas with queries. *J. ACM*, 40:185–210, January 1993.

5 S. Arora and B. Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.

6 V. Arvind and J. Torán. Isomorphism testing: Perspective and open problems. *Bulletin of the EATCS*, 86:66–84, 2005.

7 B. Borchert, D. Ranjan, and F. Stephan. On the complexity of some classical equivalence relations on boolean functions. *Theory of Computing Systems*, 31(6):679–693, 1998.

8 D. Bshouty and N. H. Bshouty. On learning arithmetic read-once formulas with exponentiation (extended abstract). In *COLT*, pages 311–317, 1994.

9 N. H. Bshouty and R. Cleve. Interpolating arithmetic read-once formulas in parallel. *SIAM J. Comput.*, 27(2):401–413, 1998.

10 N. H. Bshouty, T. R. Hancock, and L. Hellerstein. Learning arithmetic read-once formulas. *SIAM J. Comput.*, 24(4):706–735, 1995.

11 P. Clote and E. Kranakis. Boolean functions, invariance groups, and parallel complexity. *SIAM J. Comput.*, 20(3):553–590, 1991.

12 S. A. Cook and P. McKenzie. Problems complete for L. *Jl. of Algorithms*, 8:385–394, 1987.

13 J. Goldsmith, M. Hagen, and M. Mundhenk. Complexity of dnf minimization and isomorphism testing for monotone formulas. *Inf. Comput.*, 206(6):760–775, 2008.

14 V. Kabanets and R. Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. *Computational Complexity*, 13(1-2):1–46, 2004.

15 N. Kayal. Affine projections of polynomials. ECCC-Report TR11-061, 2011.

16 N. Kayal. Efficient algorithms for some special cases of the polynomial equivalence problem. In *SODA*. SIAM, 2011.

17 J. Köbler, U. Schöning, and J. Torán. *The graph isomorphism problem: its structural complexity*. Birkhauser Verlag, Basel, Switzerland, Switzerland, 1993.

18 S. Lindell. A logspace algorithm for tree canonization (extended abstract). In *STOC*, pages 400–404, 1992.

19 J. Patarin. Hidden fields equations (hfe) and isomorphisms of polynomials (ip): Two new families of asymmetric algorithms. In *EUROCRYPT'96*, pages 33–48, 1996.

20 B. V. R. Rao and J. M. N. Sarma. On the complexity of matroid isomorphism problems. In *CSR*, pages 286–298, 2009.

21 N. Saxena. *Morphisms of Rings and Applications to Complexity*. PhD thesis, Department of Computer Science, Indian Institute of Technology, Kanpur, India, 2006.

22 A. Shpilka and I. Volkovich. Read-once polynomial identity testing. In *STOC*, pages 507–516, 2008.

23 A. Shpilka and I. Volkovich. Improved polynomial identity testing for read-once formulas. In *APPROX-RANDOM*, pages 700–713, 2009.

24 T. Thierauf. The isomorphism problem for read-once branching programs and arithmetic circuits. *Chicago J. Theor. Comput. Sci.*, 1998, 1998.