ISSN 0361-7688, Programming and Computer Software, 2015, Vol. 41, No. 3, pp. 162–169. © Pleiades Publishing, Ltd., 2015.
Original Russian Text © P.G. Emelyanov, D.K. Ponomaryov, 2015, published in Programmirovanie, 2015, Vol. 41, No. 3.

Algorithmic Issues of AND-Decomposition of Boolean Formulas

P. G. Emelyanov[a, b*] **and D. K. Ponomaryov**[a, c**]

[a] *Ershov Institute of Informatics Systems, Siberian Branch, Russian Academy of Sciences, pr. Lavrentiev 6, Novosibirsk, 630090 Russia*

[b] *Novosibirsk State University, ul. Pirogova 2, Novosibirsk, 630090 Russia*

[c] *Institut für Künstliche Intelligenz, Universität Ulm, James-Franck-Ring, O27, D-89069 Ulm, Deutschland*

*e-mail: *emelyanov@iis.nsk.su, **ponom@iis.nsk.su*

Received February 01, 2015

Abstract—AND-decomposition of a boolean formula means finding two (or several) formulas such that their conjunction is equivalent to the given one. Decomposition is called disjoint if the component formulas do not have variables in common. In the paper, we show that deciding AND-decomposability is intractable for boolean formulas given in CNF or DNF and prove tractability of computing disjoint AND-decomposition components of boolean formulas given in positive DNF, Full DNF, and ANF. The latter result follows from tractability of multilinear polynomial factorization over the finite field of order 2, for which we provide a polytime factorization algorithm based on identity testing for partial derivatives of multilinear polynomials.

DOI: 10.1134/S0361768815030032

1. INTRODUCTION

Decomposition of boolean functions is an important research topic having a long history and a wide range of applications. Apart from combinatorial optimization, game and hypergraph theory, it has attracted the most attention in the logic circuit synthesis. Decomposition is related to the key parameters of electronic circuits such as size, time delay, and energy consumption. The report [Perkowski and Grygiel(1995)] contains an extensive survey of decomposition methods till the mid-1990's. The results of the next fifteen years of research are presented in [Steinbach and Lang (2003), Bioch (2010), Khatri and Gulati (2011)]. To position our paper in this research landscape, let us consider a number of key questions related to decomposition and indicate the specifics of our work.

Representations of Boolean Functions. It is known that the representation size depends on the normal form in which a boolean function is given, while translating between normal forms can be computationally hard. In this paper, we consider several well-known normal forms for boolean functions, CNF, DNF, Full DNF, and ANF,[1] but omit questions of translation between different normal forms. These representations are widely used in the circuit synthesis [Perkowski and Grygiel (1995), Sasao and Butler (2001), Kuon et al. (2008), Mishchenko and Sasao (2003)]. In the paper, we use the notions *formula* and *function* interchangeably, since for the mentioned classes of formulas, the difference is of no importance for decomposition.

Among the above mentioned normal forms, CNF and DNF were first to find application in the logic circuit synthesis. Implementations of boolean functions in full DNF, given as lookup tables, are used in the last generations of FPGA-based circuits. These have a number of advantages in comparison to non-programmable circuits and, as a consequence, enjoy wide commercial perspectives (e.g. see [Kuon et al. (2008)]).

From the algebraic point of view, ANF is a linear multivariate polynomial over the finite field of order 2. In comparison to DNF, the Algebraic Normal Form allows for a more compact representation of some classes of boolean functions used e.g. in implementation of arithmetic schemes, coders, or cyphers. Some researchers conjecture that, in general, this basis can be more cheap than DNF in the SOP-design [Sasao and Besslich (1990)]. Decomposition of positive boolean formulas (known also as monotone formulas) given in CNF/DNF attracted a particular attention in game and optimization theory (see the introduction in [Bioch (2010)] for a summary of literature). Positive DNF has numerous set-theoretic and hypergraph interpretations, which makes this form interesting in combinatorial research. Binary Decision Diagrams (BDDs) are out of the scope of this paper, but we note that they are also considered in tasks of decomposition.

Types of Decomposition. Typically one is interested in decompositions of the form $F = F_1 \odot \ldots \odot F_k$, where

[1] Algebraic Normal Form, also known as the Reed-Muller expansion, or Zhegalkin polynomial

$\odot \in \{\text{OR, AND, XOR}\}$. Decomposition into two components (bi-decomposition) is the most important case of decomposition of boolean functions. Even though it may not be stated explicitly, this case is considered in many papers: [Mishchenko et al. (2001), Sasao and Butler (2001), Bengtsson et al. (2002), Mishchenko and Sasao (2003), Chen et al. (2012), Choudhury and Mohanram (2010), Bioch (2010)], and [Khatri and Gulati(2011), Ch. 36]). Bi-decomposition has the form:

$$F(X) = \pi(F_1(\Sigma_1, \Delta), F_2(\Sigma_2, \Delta)),$$

where $\pi \in \{\text{OR, AND, XOR}\}$, $\Delta \subseteq X$, and $\{\Sigma_1, \Sigma_2\}$ is a partition of the variables $X \backslash \Delta$. As a rule, a decomposition into more than two components can be obtained by applying bi-decomposition iteratively to the already obtained components. We note that the representation form of the components may differ from that of the input function F. If $\Delta = \emptyset$ then decomposition is called disjoint and considered as optimal for many reasons. Sometimes requirements to decompositions are strengthened with additional optimality criteria, with balancedness of the variable partition being the most popular. Quite often, these additional requirements imply solving computationally hard problems.

The well-known example of bi-decomposition is Shannon's Expansion:

$$F = xF_{x=1} \vee \neg xF_{x=0} = (x \vee F_{x=0})(\neg x \vee F_{x=1}).$$

In some sense, a complete solution to bi-decomposition of arbitrary functions is given in series of papers by Steinbach et al. [Steinbach and Lang (2003)]. It allows to verify whether a given boolean function is decomposable wrt a given variable partition and to compute its components. Finding a partition is a difficult task on its own and apart from that, the solution implies a number of steps which may be intractable.

In [Bioch (2005), Bioch (2010)], Bioch studies computational properties of modular decompositions based on a generalization of Shannon's Expansion. A set of variables A is called modular set of a boolean function $F(X)$ if F can be represented as $F(X) = H(G(A), B)$, where $\{A, B\}$ is a partition of X and H, G are some boolean functions. The function $G(A)$ is called component of F and a modular decomposition is obtained from iterative decomposition into such components. It is shown that in general it is coNP-complete to decide whether a subset of variables is modular, however for positive DNF this problem is tractable. The complexity of finding a modular tree representing all modular sets of a monotone boolean function is $O(n^5N)$ where n is the number of variables and N is the number of products in DNF.

We note that a function may have a modular or bi-decomposition, but may not be AND-decomposable, since this form of decomposition assumes representation of a function as a conjunction. Thus, AND-decomposition can be viewed as a special case of modular and bi-decomposition. We demonstrate that deciding even this special case of decomposability is coNP-complete for formulas given in CNF and DNF. On the other hand, we show tractability of computing AND-decompositions of formulas given in positive CNF and DNF, Full DNF, and ANF. It is not obvious, whether the technique used by Bioch for positive DNF is applicable to the case of AND-decomposition. We note however that in our Lemma 2, the idea of computing decomposition components resembles the final step of constructing components in [Bioch (2010), Sect. 2.9].

How to Partition the Variables. This is the principal problem in decomposition of boolean functions. For many representations of functions, if a variable partition is given then finding decomposition components (if decomposition exists) is simple. Constructing modular sets [Bioch (2005), Bioch (2010)] is one way to solve this problem. In general, there are several approaches possible. Either the variable sets Δ, Σ_1, Σ_2 are guessed before the algorithm is run [Mishchenko et al. (2001), Sasao and Butler (2001), Mishchenko and Sasao(2003)], and hence, the algorithm may fail if the choice was wrong, or these sets are constructed during the runtime [Khatri and Gulati (2011), Ch. 5, 6] and [Chen et al. (2012)].

Logic vs. Algebraic Decomposition. Approaches to decomposition of boolean functions can be classified into logic and algebraic. The first are based on equivalent transformations of formulas in propositional logic. In the latter, boolean functions are considered as algebraic objects, with the corresponding transformation rules. Informally, these approaches can be characterized as syntactic and semantic, respectively. Although propositional formulas can be treated as algebraic objects, this distinction appears to be useful from the methodological point of view. As it is often the case, semantic methods have a higher potential than syntactic techniques. In general, logic-based approaches to decomposition are more powerful and achieve better results than algebraic ones: a boolean function can be decomposable logically, but not algebraically, since boolean divisors of a boolean function may be different from its algebraic factors [Khatri and Gulati (2011), Ch. 4].

A standard algebraic representation of boolean functions is polynomials, usually over finite fields, among which $\mathbb{F}_2$ (the Galois field of order 2) is the best known. Then AND-decomposition corresponds to factorization of multivariate polynomials over $\mathbb{F}_2$. We note that in general, one distinguishes between decomposition and factorization of polynomials, if they are not multilinear. The state of the research on this problem is well presented in [von zur Gathen and Gerhard (2013)], although it does not contain the key result by Shpilka and Volkovich [Shpilka and Volkovich (2010)] reported in 2010. Probably, this was because the result was obtained by an original technique of complexity analysis for arithmetic circuits and was published in proceedings of a colloquium on automata and languages. The authors noted the strong connection between polynomial factorization and identity testing. It follows from their results that a multilinear polynomial over $\mathbb{F}_2$ can be factored in time $O(l^3)$, where l is the size of the polynomial given as a symbol sequence. In this paper, we provide a factorization algorithm for multilinear polynomials over $\mathbb{F}_2$ which runs in $O(l^3)$ and is based on identity testing for partial derivatives of a product of polynomials obtained from the input one. Although the algorithm has the same O-complexity, the size of auxiliary data used by the algorithm is smaller, which is significant on large inputs. For instance, the product of polynomials is computed only once, in comparison to the approach of [Shpilka and Volkovich(2010)]. Moreover, we show that the algorithm can be implemented without computing the product explicitly, which contributes to efficiency of factorization of large input polynomials.

The paper is organized as follows. In Section 2, we introduce definitions and notations used in the text and formulate the computational problems considered in the paper. In Section 3, we study the complexity of these problems: first, for boolean formulas given in CNF, then in DNF, and finally in ANF. Omitted proofs can be found in the extended version of this paper available at: http://persons.iis.nsk.su/en/person/ponom/papers/

2. PRELIMINARIES

2.1. AND-Decomposability

For a formula φ, let var (φ) be a set of its variables. If Σ is a set of variables and var $(\varphi) \subseteq \Sigma$, then we say that φ is defined *over variables* Σ (or *over* Σ for short). taut(Σ) denotes a tautology over Σ. A literal is either a variable (a positive literal) or negation of a variable (a negative literal). A formula φ is called positive if it does not contain negative literals. If ξ and ξ' are clauses (or conjuncts, respectively), then $\xi' \subseteq \xi$ means that ξ' is a subclause (sub-conjunct) of ξ, i.e. ξ' is defined by a nonempty subset of literals from ξ. If φ is in DNF, then a conjunct ξ of the formula φ is called *redundant* in φ, if there is another conjunct ξ' in φ such that $\xi' \subseteq \xi$.

We now define the main property of boolean formulas studied in this paper, the definition is adopted from [Ponomaryov (2008)], where it is given in a general form.

Definition 1 (Decomposability). A boolean formula φ is called AND-decomposable into components with disjoint supports (or decomposable, for short) if it is equivalent to the conjunction $\psi_1 \wedge \psi_2$ of some formulas ψ_1 and ψ_2 such that:

1. var $(\psi_1) \cup$ var $(\psi_2) =$ var (φ);
2. var $(\psi_1) \cap$ var $(\psi_2) = \varnothing$;
3. var $(\psi_i) \neq \varnothing$, for $i = 1, 2$.

We say that φ is decomposable with a variable partition $\{\Sigma_1, \Sigma_2\}$, if φ has some decomposition components ψ_1 and ψ_2 over the variable sets Σ_1 and Σ_2; respectively.

Note that a similar definition could be given for OR-decomposability, i.e. for decomposition into the disjunction of ψ_1 and ψ_2. Clearly, a formula φ is AND-decomposable iff $\neg\varphi$ is OR-decomposable.

Example 1. The formula $(x \wedge y) \vee (x \wedge \neg y)$ is equivalent to the conjunction of x and taut(y) and hence decomposable.

The positive formula in *DNF* $(x \wedge u) \vee (x \wedge v) \vee (y \wedge u) \vee (y \wedge v)$ is decomposable with the components $x \vee y$ and $u \vee v$.

Note that **Definition 1** is formulated with the two components ψ_1 and ψ_2 which in turn can be decomposable formulas. Since at each decomposition step the variable sets of the components must be proper subsets of the variables of the original formula φ, the decomposition process necessarily stops and gives formulas which are non-decomposable. The obtained formulas define some partition of var(φ) and the fact below (which follows from a property of a large class of logical calculi shown in [Ponomaryov (2008)]) says that this variable partition is unique.

Fact 1 (Uniqueness of Decompositions—Corollary of Thm. 1 in [Ponomaryov (2008)])

If φ is decomposable, then there is a unique partition $\{\pi_1, \dots, \pi_n\}$ of var(φ), such that φ is equivalent to $\bigwedge\{\psi_i | \text{var}(\psi_i) = \pi_i, i = 1, \dots, n\}$, where each formula ψ_i is not decomposable.

This means that any possible algorithm[2] for decomposing a formula into components could be applied iteratively to obtain from a given φ some formulas ψ_i, $i = 1, \dots, n$, which are non-decomposable and *uniquely* define a partition of the variables of φ.

2.2. Computational Problems Considered in the Paper

In the text, we omit subtleties related to efficient encoding of input sets of variables and boolean formulas (given in CNF, DNF, or ANF) assuming their standard representation as symbol sequences. The complexity of each computational problem below will be defined wrt the size of the input formula.

$\boxed{\varnothing\text{Dec}}$ *Given a formula* φ, *decide whether* φ *is decomposable.*

$\boxed{\varnothing\text{DecPart}}$ *Given a formula* φ *and a partition* $\{\Sigma_1, \Sigma_2\}$ *of* `var`(φ), *decide whether* φ *is decomposable.*

It turns out that the problem $\varnothing$Dec for formulas in DNF is closely related to the problem of multilinear polynomial factorization (Dec$\mathbb{F}_2$) which we formulate below. The connection is, in particular, due to the fact that taking a conjunction of two formulas in DNF is

[2] Existence and complexity of decomposition algorithms in various logics have been studied in [Morozov and Ponomaryov (2010), Konev et al. (2010), Ponomaryov (2014), Ponomaryov (2008)].

quite similar to taking a product of two multivariate polynomials. We recall that a multivariate polynomial F is *linear* (*multilinear*) if the degree of each variable in F is 1. We denote a finite field of order 2 by $\mathbb{F}_2$ and say that a polynomial is *over the field* $\mathbb{F}_2$ if it has coefficients from $\mathbb{F}_2$. A polynomial F is *called factorable* over $\mathbb{F}_2$ if $F = G_1 \cdot G_2$, where G_1 and G_2 are non-constant polynomials over $\mathbb{F}_2$.

Example 2. Consider the multilinear polynomial $F = xu + xv + yu + yv$ (cf. the positive DNF from Example 1). We have $F = (x + y) \cdot (u + v)$, thus F is factorable.

The following important observation shows further connection between polynomial factorization and the problem $\varnothing$Dec:

Fact 2 (Factoring over $\mathbb{F}_2$) If a multilinear polynomial F is factorable over $\mathbb{F}_2$, then its factors do not have variables in common.

Clearly, if some factors G_1 and G_2 of F have a common variable, then the polynomial $G_1 \cdot G_2$ is not linear and thus, is not equal to F in the ring of polynomials over $\mathbb{F}_2$.

$\boxed{\text{Dec}\mathbb{F}_2}$ *Given a non-constant multilinear polynomial F over $\mathbb{F}_2$, decide whether F is factorable over $\mathbb{F}_2$.*

Since every monomial can be viewed as a set of variables and the whole polynomial as a family of such sets, the problem Dec$\mathbb{F}_2$ can be reformulated as a variant of the well-known NP-complete Set Splitting Problem (known also as the Hypergraph 2-Coloring Problem):

$\boxed{\text{Cartesian Splitting Problem}}$ *Given a family $\mathcal{F}$ of subsets of a set S, decide whether there exists a partition $\{\Sigma_1, \Sigma_2\}$ of S such that $\mathcal{F} = \{S_1 \cup S_2 | S_i \in \mathcal{F}_i, i = 1, 2\}$ where each $\mathcal{F}_i$ is a family over the elements from Σ_i.*

It turns out that the above requirement of splitting into a Cartesian union introduces enough structure into the Set Splitting Problem to obtain tractability.

3. MAIN RESULTS

3.1. Complexity of Decomposition for CNF

We show that in general, decomposition is a hard problem for CNF, but in the case of positive CNF it allows for an efficient solution. Note that decomposition itself is conceptually closer to the CNF representation, since it gives a conjunction of formulas. The situation with positive DNF and full DNF is more complicated, because decomposable formulas in DNF have a Cartesian structure which can be recognized in polytime, but the proof of this fact relies on polynomial factorization in $\mathbb{F}_2$.

Theorem 1 (Complexity for CNF). *For boolean formulas given in* CNF,

1. *the problem* $\varnothing$DecPart *is coNP-complete*;

2. *the problem*, $\varnothing$Dec *is coNP-hard and is in P^{NP}*.

Proof Sketch. We prove coNP-hardness in point 1 by showing that the set of formulas in CNF which are valid or unsatisfiable (denote it by Ω) is Karp-reducible to the set of decomposable formulas in CNF. For a given formula φ in CNF we consider the following formula constructed for φ, where p, $q \notin \text{var}(\varphi)$ are "fresh" variables:

$$\psi = (\varphi \vee p) \wedge \bigwedge_{x \in \text{var}(\varphi)} (q \vee x)$$

Clearly, ψ can be converted into CNF in linear time (in the size of φ). In the proof, we show the following equivalences: ψ is decomposable $\Leftrightarrow$ ψ is decomposable with the variable partition $\{\{p\}, \text{var}(\varphi) \cup \{q\}\} \Leftrightarrow \varphi \in \Omega$. This shows coNP-hardness of the problems $\varnothing$Dec, $\varnothing$DecPart.

The containment of $\varnothing$DecPart in coNP is shown by a Karp-reduction to the set of valid boolean formulas. Let φ be a boolean formula and $\{\Sigma_1, \Sigma_2\}$ an arbitrary partition of var (φ). We consider the formulas $\varphi^*_{\Sigma_1}$ and $\varphi^*_{\Sigma_2}$ such that $\text{var}(\varphi^*_{\Sigma_1}) \cap \text{var}(\varphi^*_{\Sigma_1}) = \varnothing$ and for $i = 1, 2$, each formula $\varphi^*_{\Sigma_i}$ is obtained from φ by renaming the variables from Σ_{3-i} into "fresh" ones, not present in φ. Then we show that that φ is decomposable with the partition $\{\Sigma_1, \Sigma_2\}$ iff the formula $\varphi^*_{\Sigma_1} \wedge \varphi^*_{\Sigma_2} \longrightarrow \varphi$ is tautology.

For the proof of point 2 of the Theorem it remains to provide a P^{NP}-algorithm for solving the problem $\varnothing$Dec. We show that if a formula φ in CNF is decomposable and contains a clause ξ with variables from both decomposition components, then ξ must contain a sub-clause $\xi' \subset \xi$ such that φ entails ξ'. This property gives an algorithm for deciding $\varnothing$Dec, which tries to "eliminate" literals one-by-one from clauses of a given formula in CNF (by quering an NP-oracle) and gives an equivalent formula, for which decomposability can be decided (and the corresponding components can be computed) in polytime. □

Theorem 2 (Complexity for Positive CNF). *For positive boolean formulas in* CNF, *the problem* $\varnothing$Dec *is in P. Moreover, decomposition components can be computed in polynomial time* (*if decomposition exists*).

This theorem follows from observation that a positive formula does not have "too many equivalent reformulations" and entailment of positive formulas is tractable (we provide a formal justification in the extended version of this paper). This allows us to easily compute a "canonical form" of a positive formula as a set of clauses, which uniquely defines decomposition components.

3.2. Decomposition of Formulas in DNF and ANF

We recall that the Algebraic Normal Form (ANF) of a boolean formula can be viewed as a multilinear polynomial over $\mathbb{F}_2$. Due to **Fact 2**, the notion of decomposability for formulas in ANF can be defined in terms of polynomial factorability over $\mathbb{F}_2$. For this reason, we use the terminology of polynomials when talking about algebraic results in this section. We start with the complexity of decomposition for formulas in Full DNF (i.e. formulas explicitly given by the set of satisfying assignments) and then formulate results on positive DNF and polynomial factorization over $\mathbb{F}_2$. Interestingly, the latter problem is also related to decomposition of formulas in Full DNF, even though such formulas contain negative literals. We show that negative literals can be encoded as "fresh" variables giving a positive DNF.

Theorem 3 (Complexity for Full DNF). *For boolean formulas in Full* DNF,

1. *the problem* $\varnothing$DecPart *is in P*;

2. *the problem* $\varnothing$Dec *is reducible to* Dec$\mathbb{F}_2$ *and hence is in P.*

In every case, the corresponding decomposition components can be computed in polynomial time.

Proof Sketch. The proof is based on the following important characterization:

Lemma 1 (Semantic Criterion of Decomposability). *Let* φ *be a boolean formula and V be the set of satisfying assignments. The formula* φ *is decomposable with a variable partition* $\{\Sigma_1, \Sigma_2\}$ *iff* $V = V|_{\Sigma_1} \uplus V|_{\Sigma_2}$, *where for* $i = 1, 2$, $V|_{\Sigma_i}$ *is the restriction of V onto the variables from* Σ_i *and* $V|_{\Sigma_1} \uplus V|_{\Sigma_2}$ *is the set of all assignments* v *such that the restriction of* v *onto* Σ_i *belongs to* $V|_{\Sigma_i}$.

To prove point 1 of Theorem 3, let φ be a formula in Full DNF and $\{\Sigma_1, \Sigma_2\}$ be a partition of var(φ). Since Full DNF is the explicit representation of all satisfying assignments, one can compute the sets $V|_{\Sigma_i}$, for $i = 1, 2$ and check in polynomial time, whether the condition in Lemma 1 holds.

In the proof of point 2, for a given formula φ, we consider a positive formula φ' obtained by injective substitution of negative literals with "fresh" variables, not occurring in φ. Then we construct a multilinear polynomial F as a sum of monomials which correspond to conjuncts of φ' and show that φ is decomposable iff F is factorable over $\mathbb{F}_2$. □

It turns out that for a positive formula φ in DNF without redundant conjuncts, decomposability is equivalent to factorability over $\mathbb{F}_2$ of the multilinear polynomial corresponding to φ. The polynomial is obtained as the sum of monomials (products of variables) corresponding to the conjuncts of φ Observe that the positive formula $\varphi = x \vee (x \wedge y) \vee z$ with the redundant conjunct $x \wedge y$ is equivalent to $(x \vee z) \wedge$ taut($\{y\}$) and thus, decomposable. However, the polynomial $x + xy + z$ corresponding to φ is non-factorable. Also note that if a polynomial has a factor with the constant monomial, e.g. $xy + y = (x + 1) \cdot y$, then the corresponding boolean formula in DNF contains a redundant conjunct.

Theorem 4 (Decomposition of Positive DNF and Factorization). *For positive boolean formulas in DNF without redundant conjuncts, the problem* $\varnothing$Dec *is equivalent to* Dec$\mathbb{F}_2$. □

We formulate the main result on formulas given in DNF in the following corollary which is a consequence of Theorems 4, 5, and the constructions mentioned in the proof sketch to Theorem 1.

Corollary 1 (Complexity for DNF).

1. *for formulas in DNF, the problem* $\varnothing$DecPart *is coNP-complete*;

2. *for positive boolean formulas in DNF, the problem* $\varnothing$Dec *is in P and the corresponding decomposition components can be computed in polynomial time.*

Proof Sketch. The first point of the corollary follows from the construction used in the proof sketch to Theorem 1. Note that the formula ψ considered in the proof sketch can be converted into DNF in polynomial time, if the input formula φ is given in DNF, which shows coNP-hardness for the first point of the Corollary. The containment in coNP can be shown by reduction to the validity of boolean formulas as in the proof sketch to Theorem 1, since the construction there does not depend on the normal form in which the formula φ is given. The second point of the Corollary follows from Theorems 4 and 5, and the fact that redundant conjuncts can be found and eliminated efficiently from an input formula. □

We now turn to tractability of the problem Dec$\mathbb{F}_2$, to which the decomposition problems in Theorem 3 and Corollary 1 are reduced. As already noticed, tractability of Dec$\mathbb{F}_2$ follows from the results of [Shpilka and Volkovich (2010)], where the authors propose two solutions for decomposition of a polynomial over an arbitrary finite field F. The first one is a decomposition algorithm, which has a subroutine for computing a justification assignment for an input polynomial, and relies on a procedure for identity testing in $\mathbb{F}$. It is proved that the complexity of this algorithm is $O(n^3 \cdot d \cdot IT)$, where n is the number of variables, d is the maximal individual degree of variables in the input polynomial, and IT is the complexity of identity testing in $\mathbb{F}$. It follows that this gives a decomposition algorithm of quartic complexity for factoring multilinear polynomials over $\mathbb{F}_2$. The second solution proposed by the authors is a decomposition algorithm which constructs for every variable of an input polynomial f, a combination $f \cdot f_1 - f_2 \cdot f_3$ of four polynomials, where each f_i is a "copy" of f under a renaming of some variables. Every combination is tested for equality to the zero polynomial. It can be seen that this gives an algorithm of cubic complexity for factoring multilinear polynomials over $\mathbb{F}_2$.

In Theorem 5 below, we provide a solution to factorization of multilinear polynomials over $\mathbb{F}_2$, which is different from the both algorithms proposed in [Shpilka and Volkovich (2010)]. The only common feature between the approaches is application of identity testing, which seems to be inevitable in factorization. Our solution is based on computation of partial derivatives of polynomials obtained from the input one and gives an algorithm of cubic complexity. More precisely, the product $f_1 \cdot f_2$ is computed, where f_i are polynomials obtained from the input, and then for each variable x, the partial derivative of $f_1 \cdot f_2$ is tested for equality to zero. In particular, our algorithm operates polynomials which are smaller than the ones considered in [Shpilka and Volkovich (2010)]. Moreover, we note in the extended version of the paper that the same can be achieved without computing the product $f_1 \cdot f_2$ explicitly, which is particularly important on large inputs. We present the factorization algorithm as the theorem below to follow the complexity oriented style of exposition used in this paper.

Theorem 5 (Tractability of Linear Polynomial Factorization over $\mathbb{F}_2$). *The problem* Dec$\mathbb{F}_2$ *is in P and for any factorable multilinear polynomial its factors can be computed in polynomial time.*

Proof. Let F be a non-constant multilinear polynomial over $\mathbb{F}_2$. We describe a number of important properties which hold if F is factorable over $\mathbb{F}_2$. Based on these properties, we derive a polynomial procedure for partitioning the variables of F into disjoints sets Σ_1 and Σ_2 such that if F is factorable, then it must have factors which are polynomials over these variable sets. Having Σ_1 and Σ_2, it suffices to check whether F is indeed factorable wrt this partition: if the answer is "no", then F is non-factorable, otherwise we obtain the corresponding factors.

Checking whether F is factorable wrt a variable partition can be done efficiently due the following fact:

Lemma 2 (Factorization Under a Given Variable Partition). *In the notations above, for $i = 1, 2$, let S_i be the set of monomials obtained by restricting every monomial of F onto Σ_i (for instance, if $F = xy + y$ and $\Sigma_1 = \{x\}$; then $S_1 = \{x, 1\}_j$). Let, F_i be the polynomial consisting of the monomials of S_i, for $i = 1, 2$. Then F is factorable into some polynomials with the sets of variables Σ_1 and Σ_2 iff $F = F_1 \cdot F_2$.*

Proof of the Lemma. The "if" direction is obvious, since for $i = 1, 2$, each F_i necessarily contains all the variables from Σ_i. Now assume that F has a factorization $F = G_1 \cdot G_2$ which corresponds to the partition Σ_1, Σ_2. Then every monomial of F is a product of some monomials from G_1, G_2, i.e. it either contains variables of both Σ_1 and Σ_2, or only from Σ_i for some $i = 1, 2$ iff G_{3-i} contains the constant monomial. This means that S_i is the set of monomials of G_i, for $i = 1, 2$, i.e. $F_i = G_i$. □

Let us proceed to properties of factorable polynomials over $\mathbb{F}$. Let $F_{x=v}$ be the polynomial obtained from F by setting x equal to v. Note that $\frac{\partial F}{\partial x} = F_{x=1} + F_{x=0}$ and every multilinear polynomial can be represented in the form

$$F = \frac{\partial F}{\partial x} \cdot x + F_{x=0} = (F_{x=1} + F_{x=0})x + F_{x=0}$$

(the last expression is also called Reed's expansion).

First of all, note that if some variable x is contained in every monomial of F, then F is either nonfactorable (in case $F = x$), or trivially factorable, i.e. $F = x \cdot \frac{\partial F}{\partial x}$. We further assume that there is no such variable in F. We also assume that $F \neq x + 1$, i.e. F contains at least two variables[3].

Let F be a polynomial over the set of variables $\{x, x_1, \ldots, x_n\}$. If F is factorable, then it can be represented as $F = (x \cdot Q + R) \cdot H$, where

—the polynomials Q, R, and H do not contain x;

—Q and R do not have variables with H in common;

—R is a non-empty polynomial (since F is not trivially factorable);

—the left cofactor of this product is a non-factorable polynomial.

Then we have $F_{x=0} = R \cdot H$ and also $\frac{\partial F}{\partial x} = Q \cdot H$. Obviously, the both polynomials can be computed in polynomial time.

Let y be a variable of F, different from x, and consider the following derivative of the product of these polynomials:

$$\frac{\partial}{\partial y}(Q \cdot R \cdot H^2) = \frac{\partial Q}{\partial y}RH^2 + Q\frac{\partial}{\partial y}(RH^2)$$
$$= \frac{\partial Q}{\partial y}RH^2 + \frac{\partial R}{\partial y}QH^2 + 2\frac{\partial H}{\partial y}QRH$$

since for all z, $2z = z + z = 0$ holds in $\mathbb{F}_2$, we have:

$$= H^2 \cdot \left(\frac{\partial Q}{\partial y}R + \frac{\partial R}{\partial y}Q\right) = H^2 \cdot \frac{\partial}{\partial y}(Q \cdot R).$$

It follows that in case y is a variable from H, we have $\frac{\partial}{\partial y}(Q \cdot R) = 0$ and thus,

$$\frac{\partial}{\partial y}(Q \cdot R \cdot H^2) = 0.$$

Let us now show the opposite, assume that the variable y does not belong to H and prove that the H derivative is not equal to zero.

Since y does not belong to H, in general, Q and R have the form

$$Q = Ay + B, \quad R = Cy + D,$$

[3] We note that besides the factors of the form x and $x + 1$, there is a number of other simple cases of factorization that can be recognized easily.

for some polynomials A, B, C, D not containing y. Then $Q \cdot R = ACy^2 + (AD + BC)y + BD$ and hence, $\frac{\partial}{\partial y}(Q \cdot R) = AD + BC$.

Thus, we need to show that $AD + BC \neq 0$. Assume the contrapositive, i.e. that $AD + BC = 0$. Note that AD and BC can not be zero, because otherwise at least one of the following holds: $A = B = 0$, $A = C = 0$, $D = B = 0$, or $D = C = 0$. The first two conditions are clearly not the case, since we have assumed that x and y are not contained in H, while the latter conditions yield that F is trivially factorable (wrt the variable y or x, respectively).

From this we obtain that $AD + BC = 0$ holds iff $AD = BC$ (since we are in $\mathbb{F}_2$).

Let $B = f_1 \cdot \ldots \cdot f_m$ and $C = g_1 \cdot \ldots \cdot g_n$ be the (unique) factorizations of B and C into nonfactorable polynomials. We have $AD = f_1 \cdot \ldots \cdot f_m \cdot g_1 \cdot \ldots \cdot g_n$, thus we may assume that $A = f_1 \cdot \ldots \cdot f_k \cdot g_1 \cdot \ldots \cdot g_l$ for some $0 \le k \le m$ and $0 \le l \le n$ (when $k = l = 0$ we assume that $A = 1$).

The polynomials B, C, D can be represented in the same form. Let us denote for polynomials U, V by (U, V) the greatest common divisor of U and V. Then $A = (A, B) \cdot (A, C)$, $B = (A, B) \cdot (D, B)$, similarly for C and D, and we obtain

$$\begin{aligned} x \cdot Q + R &= x \cdot (Ay + B) + (Cy + D) \\ &= x \cdot ((A, B)(A, C)y + (A, B)(D, B)) \\ &\quad + ((A, C)(D, C)y + (D, B)(D, C)) \\ &= ((A, B)x + (D, C))((A, C)y + (D, B)), \end{aligned}$$

which is a contradiction, because we have assumed that $x \cdot Q + R$ is non-factorable.

We have obtained a procedure for partitioning the variables of F into disjoint sets Σ_1 and Σ_2 in the following way. Having chosen some initial variable x from F, we first assign $\Sigma_1 = \{x\}$, $\Sigma_2 = \varnothing$ and compute the polynomial $Q \cdot R \cdot H^2$ (which equals $\frac{\partial F}{\partial x} \cdot F_{x=0}$). Then for every variable y from F (distinct from x), we compute the derivative $\frac{\partial}{\partial y}(Q \cdot R \cdot H^2)$. If it equals to zero, we put y into Σ_2, otherwise we put y into Σ_1. If at the end we have $\Sigma_2 = \varnothing$, then the polynomial F is non-factorable. Otherwise it remains to apply Lemma 2 to verify whether the obtained sets Σ_1 and Σ_2 indeed correspond to a factorization of F. If the answer is "no", then F is non-factorable, otherwise the polynomials F_1 and F_2 defined in Lemma 2 are the required factors.

If n is the size of the input polynomial as a symbol sequence, then it takes $O(n^2)$ steps to compute the polynomial $G = Q \cdot R \cdot H^2$ and check whether the derivative $\frac{\partial G}{\partial y}$ equals zero for a variable y (since both, identity testing and computing the derivative can be done in time linear in the size of a polynomial over $\mathbb{F}_2$, given as a symbol sequence). As we must check this for every variable $y \neq x$, we have a procedure that computes a candidate variable partition in $O(n^3)$ steps. Then it takes $O(n^2)$ time to verify by Lemma 2 whether this partition indeed corresponds to factors of F. □

3.3 Testing AD = BC without Multiplication

In practice, computing the product of polynomials having sizes comparable to the input polynomial is not reasonable, especially in the case when this product is not needed explicitly. Let us demonstrate that testing $AD = BC$ can be done without explicitly computing the products AD and BC.

Consider expansions of the polynomials A, D, B, C with respect to a variable x such that the monomial x does not divide AD and BC:

$$\begin{aligned} (A_1x + A_2)(D_1x + D_2) &= (B_1x + B_2)(C_1x + C_2), \\ A_1D_2x^2 + (A_1D_2 + A_2D_1)x + A_2D_2 & \\ &= B_1C_1x^2 + (B_1C_2 + B_2C_1)x + B_2C_2. \end{aligned}$$

The equality is true if the coefficients at the corresponding degrees of x are equal:

$$\begin{cases} A_1D_1 = B_1C_1 \\ A_2D_2 = B_2C_2 \\ A_1D_2 + A_2D_1 = B_1C_2 + B_2C_1 \end{cases}$$

Thus, in the first two equalities, we have the same problem to be solved for smaller polynomials. Let us show that the third equality can be treated similarly.

Assume that the first two equalities are already checked to be true; otherwise, the original equality does not hold. In addition, we can assume further that $A_1, A_2 \neq 0$.

Let us multiply the both sides of the third equality by A_1A_2 (other combinations are possible). We have

$$A_1^2A_2D_2 + A_1A_2^2D_1 = A_1A_2B_1C_2 + A_1A_2B_2C_1.$$

Since the first two equalities hold, we can apply suitable substitutions and regrouping

$$A_1^2B_2C_2 + A_1A_2B_2C_1 = A_2^2B_1C_1 + A_1A_2B_1C_2,$$

extract the common factor

$$A_1B_2(A_1C_2 + A_2C_1) = A_2B_1(A_2C_1 + A_1C_2),$$

and next, rewrite as follows:

$$(A_1B_2 + A_2B_1)(A_1C_2 + A_2C_1) = 0.$$

Therefore, verifying the third equality reduces to testing two equalities with polynomials having smaller sizes:

$$A_1B_2 = A_2B_1 \quad \text{or} \quad A_1C_2 = A_2C_1.$$

As a result, verifying the original equality is reduced to testing at most four smaller equalities. By applying this procedure recursively, the problem can be reduced to either constant polynomials or small ones, for which multiplication is faster. The number of recursive calls is estimated by $O(N)$ (this is the number of vertices in a rooted, at least binary and at most quaternary, tree having N leaves).

4. CONCUSIONS

We have proved that AND-decomposability is intractable in general for boolean formulas given in CNF or DNF. On the other hand, we have shown the existence of polytime algorithms for computing decomposition components of positive formulas in DNF and formulas given in Full DNF, and the Algebraic Normal Form. Since AND-decomposability and OR-decomposability are the dual notions, our results are also applicable to the latter case.

Efficient AND-decomposition is important in optimization of logic circuits, since it can substantially reduce the representation size. The result on decomposition of positive DNF can contribute to improving efficiency of existing model counting techniques, while the result on Full DNF can be applied to optimization of lookup tables. It is straightforward to transform a DNF into a full DNF, thus, under a moderate increase of the formula size, one can decompose the obtained full DNF and then minimize the formulas having a smaller size, in order to obtain a compact conjunctive decomposition of the source DNF.

The factorization algorithm for multilinear polynomials over $\mathbb{F}_2$, described in this paper, can be used for efficient disjoint AND-decomposition of formulas given in DNF or ANF. It remains an open question, whether this algorithm can be used to obtain non-disjoint decompositions for boolean formulas.

Further research questions include implementation of the proposed decomposition algorithms and their evaluation on industrial benchmarks for boolean circuits.

5. ACKNOWLEDGMENTS

The first author was supported by the Russian Foundation for Humanities, grant no. 13-01-12003B. The second author was supported by the German Research Foundation within the Transregional Collaborative Research Center SFB/TRR 62 “Companion-Technology for Cognitive Technical Systems” the Russian Academy of Sciences (project no. 15/10), and the Russian Foundation for Basic Research (grant no. 15-07-03410). .

REFERENCES

1. Perkowski, M.A. and Grygiel, S., *PSU Electrical Engineering Department Report*, Portland, Oregon, USA: Department of Electrical Engineering, Portland State University, 1995.
2. Steinbach, B. and Lang, C., *Artificial Intelligence Review*, 2003, vol. 20, p. 319.
3. Bioch, J.C., in *Boolean Models and Methods in Mathematics, Computer Science, and Engineering*, Crama, Y. and Hammer, P.L., Eds., New York, NY, USA: Cambridge University Press, 2010, vol. 134 of *Encyclopedia of Mathematics and Its Applications*, pp. 39–78.
4. Khatri, S.P. and Gulati, K., Eds., *Advanced Techniques in Logic Synthesis, Optimizations and Applications*, New York Dordrecht Heidelberg London: Springer, 2011.
5. Sasao, T. and Butler, J.T., *Proc. 2001 Asia and South Pacific Design Automation Conference (ASP-DAC'01)*, New York, NY, USA: ACM, 2001, pp. 219–224.
6. Kuon, I., Tessier, R., and Rose, J., *FPGA Architecture: Survey and Challenges,* Boston–Delft: Now Publishers Inc., 2008.
7. Mishchenko, A. and Sasao, T., *Proc. 40th ACM/IEEE Design Automation Conference (DAC'03)*, New York, NY, USA: ACM, 2003, pp. 149–154.
8. Sasao, T. and Besslich, P., *IEEE Transactions on Computers*, 1990, vol. C-39, p. 262.
9. Mishchenko, A., Steinbach, B., and Perkowski, M.A., *Proc. 38th ACM/IEEE Design Automation Conference (DAC'01)*, New York, NY, USA: ACM, 2001, pp. 103–108.
10. Bengtsson, T., Martinelli, A., and Dubrova, E., *Proc. Notes of the 11th IEEE/ACM International Workshop on Logic & Synthesis (IWLS'02)*, 2002, pp. 51–55.
11. Chen, H., Janota, M., and Marques-Silva, J., *Proc. Design, Automation & Test in Europe Conference (DATE'12)*, IEEE, 2012, pp. 816–819.
12. Choudhury, M. and Mohanram, K., *Proc. 2010 IEEE/ACM International Conference on Computer-Aided Design (ICCAD'10)*, Piscataway, New Jersy, USA: IEEE Press, 2010, pp. 586–591.
13. Bioch, J.C., *Discrete Applied Mathematics,* 2005, vol. 149, p. 1.
14. von zur Gathen, J. and Gerhard, J., *Modern Computer Algebra*, New York, NY, USA: Cambridge University Press, 2013), 3rd ed.
15. Shpilka, A. and Volkovich, I., Proc. 37th International Colloquium on Automata, Languages and Programming. Part 1 (ICALP 2010), Springer, 2010, vol. 6198 of *Lecture Notes in Computer Science*, pp. 408–419.
16. Ponomaryov, D., Bulletin of the Novosibirsk Computing Center 2008, vol. 28, p. 111, URL http://persons.iis.nsk.su/files/persons/ pages/deIta-decomp.pdf.
17. Morozov, A. and Ponomaryov, D., *Siberian Mathematical Journal*, 2010, vol. 51, p. 667.
18. Konev, B., Lutz, C., Ponomaryov, D., and Wolter, F., *Proc. Twelfth International Conference on Principles of Knowledge Representation and Reasoning (KR'10)*, Palo Alto, California, USA: AAAI Press, 2010.
19. Ponomaryov, D., in *Research Note. Abstract appears in Proc. Logic Colloquium' 14*, 2014, URL http://persons.iis.nsk.su/files/persons/ pages/sigdecomp.pdf.