Complexity of DNF and Isomorphism of Monotone Formulas

Judy Goldsmith[1], Matthias Hagen[2]*, Martin Mundhenk[2]

[1] University of Kentucky, Dept. of Computer Science, Lexington, KY 40506–0046, goldsmit@cs.uky.edu

[2] Friedrich-Schiller-Universität Jena, Institut für Informatik, D–07737 Jena, {hagen,mundhenk}@cs.uni-jena.de

Abstract. We investigate the complexity of finding prime implicants and minimal equivalent DNFs for Boolean formulas, and of testing equivalence and isomorphism of monotone formulas. For DNF related problems, the complexity of the monotone case strongly differs from the arbitrary case. We show that it is DP-complete to check whether a monomial is a prime implicant for an arbitrary formula, but checking prime implicants for monotone formulas is in L. We show PP-completeness of checking whether the minimum size of a DNF for a monotone formula is at most k. For k in unary, we show the complexity of the problem to drop to coNP. In [Uma01] a similar problem for arbitrary formulas was shown to be Σ_2^p-complete. We show that calculating the minimal DNF for a monotone formula is possible in output-polynomial time if and only if P = NP. Finally, we disprove a conjecture from [Rei03] by showing that checking whether two formulas are isomorphic has the same complexity for arbitrary formulas as for monotone formulas.

1 Introduction

Monotone formulas are Boolean formulas that contain only conjunction and disjunction as connectives, but no negation. To solve the satisfiability problem for monotone formulas is trivial. Every satisfiable monotone formula is satisfied by the assignment that sets all variables to true. Hence, the computational complexity of the satisfiability problem for monotone formulas is very much simpler than the NP-complete satisfiability problem for arbitrary formulas. On the other hand, counting the number of satisfying assignments has the same complexity for monotone and for arbitrary formulas [Val79]. Hence, it is interesting to compare the complexity of problems for arbitrary and for monotone formulas.

In the first part of this paper, we investigate the complexity of calculating smallest equivalent Disjunctive Normal Forms (DNF). The smallest equivalent DNF for a formula consists of prime implicants of the formula. For arbitrary formulas, it is hard to find the smallest choice of prime implicants. It is still open whether a smallest equivalent DNF can be calculated in polynomial space.

* The author is supported by a Landesgraduiertenstipendium Thüringen.

For monotone formulas, the smallest DNF consists of all prime implicants. We consider problems of checking and finding prime implicants (Section 3). We show that checking whether a monomial is a prime implicant for a formula is DP-complete for arbitrary formulas, whereas it is in L for monotone formulas. DP [PY84] contains both NP and coNP and is contained in Σ_2^{p} in the Polynomial Time Hierarchy. The question, whether a prime implicant of a certain size exists for a given formula, was shown to be Σ_2^{p}-complete in [Uma01]. We show that the same question is only NP-complete for monotone formulas. The complexity of calculating the size of a smallest DNF depends on the representation of the problem. Umans [Uma01] showed that given a formula φ in DNF and an integer k, it is Σ_2^{p}-complete to decide whether φ has a DNF of size at most k. Notice that the length of the input DNF is greater than the size of the DNF that is searched for (except trivial cases). This seems necessary to allow the problem to be decided within a non-deterministic polynomial time bound, because the smallest DNF of a (monotone) formula may have size exponential in the length of the formula. The exact complexity of this problem for arbitrary formulas is open. It is Σ_2^{p}-hard (which follows from [Uma01]) and in EXPTIME. For monotone formulas, we exactly characterize the complexity of this problem by showing it to be PP-complete. If one encodes the upper bound of the DNF length in unary instead — i.e. given formula φ and string 1^k, decide whether φ has a DNF of size $\leq k$ — we prove the problem to be Σ_2^{p}-complete for arbitrary formulas, and coNP-complete for monotone formulas.

In Section 4 we consider the hardness of calculating the smallest DNF for a monotone formula. It is clear that the smallest DNF is not polynomial time computable. Therefore, we consider the notion of output-polynomial time. A function is in output-polynomial time if it can be computed in time polynomial in the length of the input plus the length of the function value [Pap97]. We show that the DNF for a monotone formula is output-polynomial time computable if and only if P = NP. Even calculating the size of a minimal DNF is shown to be PP-complete. In Section 5 we consider equivalence and isomorphism problems. The problem of deciding whether monotone formulas φ and ψ are equivalent is known to be coNP-complete [Rei03]. For arbitrary formulas the same completeness holds. If φ is in Conjunctive Normal Form (CNF) and ψ is in DNF, the equivalence problem remains coNP-complete for arbitrary formulas, but for monotone formulas an upper bound between P and coNP-complete was settled in [FK96]. In the case that one of the input formulas consists only of terms of bounded length, we give an L-algorithm improving results from [EG95, BEGK00]. Finally, we refute a conjecture from [Rei03], by showing that checking whether two given formulas are isomorphic has exactly the same complexity for arbitrary as for monotone formulas.

2 Definitions

We consider Boolean formulas with connectives $\wedge$ (conjunction), $\vee$ (disjunction), and $\neg$ (negation). We assume that the negations appear directly in front of variables. (Other connectives are used as abbreviations, whereas we use the $\leftrightarrow$ only once because of the doubling of the formula length.) Actually this is no limitation because every formula may be transformed in polynomial time to fulfill these assumptions. A *monotone formula* is a Boolean formula without negations. A *term* is a conjunction or a disjunction of *literals*, i.e. of variables and negated variables; a conjunction is called *monomial*, and a disjunction is called *clause*. The empty clause is unsatisfiable, and the empty monomial, denoted λ, is valid. A *monotone term* is a term without negations. Terms are also considered as sets of literals. Term T_1 *covers* term T_2 if $T_1 \subseteq T_2$.

An *assignment* $\mathcal{A}$ for a formula φ is a mapping of the variables of φ to the truth values true and false. An assignment $\mathcal{A}$ is said to *satisfy* formula φ if φ evaluates to true under $\mathcal{A}$. For monotone formulas we regard $\mathcal{A}$ also as a set $\mathcal{A}_m$ where variable x is in $\mathcal{A}_m$ if and only if x gets value true under $\mathcal{A}$. Notice that in this way every monotone monomial can also be interpreted as an assignment.

An *implicant* of a formula φ is a monomial C such that $C \to \varphi$ is valid. A monomial C is a *prime implicant* of φ iff (1) C is an implicant of φ and (2) for every proper subset $S \subset C$ holds that S is not an implicant of φ. Notice that in order to check condition (2) it suffices to check for $C = \{\ell_1, \ell_2, \ldots, \ell_k\}$ whether for each $\ell_i \in C$ it holds that $C - \{\ell_i\}$ is not an implicant of φ. Every proper subset S of C is a subset of $C - \{\ell_i\}$ for some i. For $S \subseteq C - \{\ell_i\}$ holds that $(C - \{\ell_i\}) \to S$ is valid. If S is an implicant of φ, then $S \to \varphi$ is valid. Both valid formulas together yield that $(C - \{\ell_i\}) \to \varphi$ is valid too, inducing that $C - \{\ell_i\}$ is an implicant of φ. Hence, if no $C - \{\ell_i\}$ is an implicant of φ, then no proper subset of C is an implicant of φ.

A formula is in *conjunctive normal form (CNF)* if it is a conjunction of clauses. Similarly a formula is in *disjunctive normal form (DNF)* if it is a disjunction of monomials. It is said to be in k-CNF (k-DNF), if all clauses (monomials) consist of at most k literals. A monotone formula φ in normal form is *irredundant* if and only if no term of φ covers another term of φ. For a monotone formula, the disjunction of all its prime implicants yields an equivalent monotone DNF. On the other hand, every prime implicant must appear in every equivalent DNF for a monotone formula. Hence, the smallest DNF for a monotone formula is unique and equals the disjunction of all its prime implicants. This is not the case for non-monotone formulas, where the smallest DNF is a subset of the set of all prime implicants. It is NP-hard to select the right prime implicants [Mas79]. See also [Czo99] for an overview on the complexity of calculating DNFs.

We use complexity classes L (logarithmic space), P, NP, coNP, DP (difference polynomial time, which appears to be the class for "exact cost" optimization), Σ_2^{p} (NP with NP oracle), PP (probabilistic polynomial time), and PSPACE. The inclusion structure is $\mathsf{L} \subseteq \mathsf{P} \subseteq \begin{smallmatrix}\mathsf{NP}\\ \mathsf{coNP}\end{smallmatrix} \subseteq \mathsf{DP} \subseteq \begin{smallmatrix}\Sigma_2^{\mathsf{p}}\\ \mathsf{PP}\end{smallmatrix} \subseteq \mathsf{PSPACE}$. All considered classes except L are closed downwards under $\leq_m^p$-reduction, and PP is closed under complement. Closely related to PP is the function class $\#\mathsf{P}$. See [Pap94] for definitions of these classes. As natural complete problems for NP, coNP and PP we consider SAT (is the Boolean formula φ satisfiable?), UNSAT (is φ unsatisfiable?), and MajSat (do at least half of the assignments satisfy φ?). We assume that the input formulas for these problems contain only $\wedge$, $\vee$, and $\neg$ (placed right in front of variables) as connectives.

3 Size of Disjunctive Normal Forms

In this section we concentrate on computing the size of minimal DNFs. Therefore we first analyze the complexity of finding prime implicants.

A valid formula has the empty monomial λ as its only prime implicant. An unsatisfiable formula has no prime implicant at all. In general, a formula φ has a prime implicant if and only if φ is satisfiable. Therefore, the question of whether a formula has a prime implicant is NP-complete, and it is in L for monotone formulas.

Next we consider the problem of deciding whether a monomial is a prime implicant of a formula.

IsPrimi : *instance:* Boolean formula φ and monomial C
question: is C a prime implicant of φ ?

The complexity of IsPrimi is intermediate between $\mathsf{NP} \cup \mathsf{coNP}$ and Σ_2^{p}.

Theorem 3.1. IsPrimi *is* DP-*complete.*

Proof. The standard DP-complete problem is Sat-Unsat $= \{(\varphi, \psi) \mid \varphi \in$ Sat$, \psi \in$ Unsat$\}$. We show that Sat-Unsat $\leq_m^p$ IsPrimi. The reduction function is the mapping $(\varphi, \psi) \mapsto (\neg\varphi \vee (\neg\psi \wedge z), z)$, where z is a new variable that neither appears in φ nor in ψ. It is clear that this mapping is polynomial time computable.

If $(\varphi, \psi) \in$ Sat-Unsat, then $\neg\psi$ is valid, and therefore $\neg\psi \wedge z$ has z as prime implicant. Hence z is an implicant of $\neg\varphi \vee (\neg\psi \wedge z)$. Because $\varphi \in$ Sat, its negation $\neg\varphi$ is not valid. Therefore, the empty monomial λ is not an implicant of $\neg\varphi$. Hence, z is a prime implicant of $\neg\varphi \vee (\neg\psi \wedge z)$. Next we consider the case that $(\varphi, \psi) \notin$ Sat-Unsat. If $\varphi \notin$ Sat, then $\neg\varphi \vee (\neg\psi \wedge z)$ is valid and λ is the only prime implicant of this formula. If $\varphi \in$ Sat and $\psi \notin$ Unsat, then $\neg\psi$ is not valid and therefore z is not an implicant of $\neg\psi \wedge z$. Because $\neg\varphi$ is not valid, it follows that z is neither an implicant of $\neg\varphi \vee (\neg\psi \wedge z)$.

This proves the DP-hardness of IsPrimi. We now show that IsPrimi is in DP, by proving IsPrimi $\leq_m^p$ Sat-Unsat. Let (φ, C) be an instance for IsPrimi, where $C = \{\ell_1, \ell_2, \dots, \ell_k\}$ is a monomial. Let $C(i) = C - \{\ell_i\}$. First, a necessary condition for C being a prime implicant for φ is that no proper subset of C is an implicant for φ (C is called *prime* in this case). This condition is equivalent to every $C(i)$ being not an implicant for φ (as argued in Section 2). I.e., for every i, $C(i) \to \varphi$ is not valid, and equivalently $\neg(C(i) \to \varphi)$ is in Sat. Summarized, if C is a prime implicant for φ, then $p(\varphi, C) = \bigwedge_{i=1}^{k} \neg(C(i) \to \varphi)$ is in Sat. Second, the other necessary condition for C being a prime implicant for φ is that C is an implicant for φ. I.e. $C \to \varphi$ is valid, and equivalently $n(\varphi, C) = \neg(C \to \varphi)$ is in Unsat.

C is a prime implicant for φ if and only if C is prime and C is an implicant for φ. This is equivalent to $p(\varphi, C) \in$ SAT and $n(\varphi, C) \in$ UNSAT. Eventually, this yields that $(\varphi, C) \in$ ISPRIMI if and only if $(p(\varphi, C), n(\varphi, C)) \in$ SAT-UNSAT. Since p and n are both polynomial time computable, the function f with $f(\varphi, C) = ((p(\varphi, C), n(\varphi, C))$ is a polynomial time reduction function for ISPRIMI $\leq_m^p$ SAT-UNSAT. Since DP is closed downwards under polynomial time many-one reduction, ISPRIMI $\in \mathsf{DP}$ follows. $\square$

For monotone formulas, the same problem is much easier. A monomial is an implicant of a monotone formula, if and only if the assignment that corresponds to the monotone monomial satisfies the formula. It can be checked in logarithmic space whether an assignment satisfies a monotone formula.

ISPRIMI$_{\text{mon}}$:	*instance:*	monotone formula φ and monotone monomial C
	question:	is C a prime implicant of φ ?

Theorem 3.2. ISPRIMI$_{\text{mon}}$ *is in* L.

The problem of checking whether a formula φ has a prime implicant of size at most k was shown to be Σ_2^{p}-complete [Uma01]. We show, that the same problem for monotone formulas is NP-complete only.

PRIMISIZE$_{\text{mon}}$:	*instance:*	monotone Boolean formula φ and integer k
	question:	does φ have a prime implicant consisting of at most k variables?

In the following, we will define reductions that transform formulas into monotone formulas, such that satisfying assignments of the basic formula are similar to prime implicants of the monotone formula.

Definition 3.3. *Let φ be a Boolean formula with connectives $\wedge$, $\vee$ and $\neg$, and variables $x_1, \ldots, x_n$. Remember that all negation signs appear directly in front of variables. Then $r(\varphi)$ denotes the formula obtained by replacing all appearances of $\neg x_i$ in φ by the new variable y_i (for $i = 1, 2, \ldots, n$). Let $c(\varphi)$ denote the conjunction $\bigwedge_{i=1}^{n}(x_i \vee y_i)$ and $d(\varphi)$ denote the disjunction $\bigvee_{i=1}^{n}(x_i \wedge y_i)$. The formulas φ^c and φ^{cd} are defined as $\varphi^c = r(\varphi) \wedge c(\varphi)$ and $\varphi^{cd} = \varphi^c \vee d(\varphi) = (r(\varphi) \wedge c(\varphi)) \vee d(\varphi)$.*

Since φ^c and φ^{cd} do not contain any negation signs, they are monotone formulas. Let $\mathcal{A}$ be an assignment for φ. Then $\mathcal{A}'_m$ denotes the assignment $\mathcal{A}'_m = \{x_i \mid \mathcal{A}$ maps x_i to true$\} \cup \{y_i \mid \mathcal{A}$ maps x_i to false$\}$. Such an assignment, which contains exactly one from x_i and y_i, is called *conform.* Notice that there is a one-to-one relation between assignments to φ and conform assignments to φ^c and φ^{cd}.

Theorem 3.4. PRIMISIZE$_{\text{mon}}$ *is* NP*-complete.*

Proofsketch: PRIMISIZE$_{\text{mon}}$ is easily seen to be in NP. NP-hardness follows from a reduction SAT $\leq_m^p$ PRIMISIZE$_{\text{mon}}$. The reduction function maps every SAT instance φ with variables $x_1, \ldots, x_n$ to (φ^c, n). This reduction is polynomial time computable. $\square$

Since minimal DNFs of a formula are disjunctions of prime implicants, a natural question arises. How hard is it to calculate the size of a minimal DNF? The respective problem

$\textsc{MinDnfSize}_{\mathrm{dnf}}$: *instance:* Boolean formula φ in DNF and integer k
question: does φ have a DNF with at most k occurences of variables?

for arbitrary DNF formulas was shown to be Σ_2^{p}-complete in [Uma01].

The monotone version — input is a monotone DNF-formula φ — is in L since counting the length of the irredundant part of φ suffices and testing irredundancy can be managed in logarithmic space.

If the input is an arbitrary formula, the problem is Σ_2^{p}-hard (which follows from the latter result from [Uma01]). It is clear that the problem is in $\mathsf{EXPTIME}$, but it is even not known whether the problem is in PSPACE. We show PP-completeness when the input is monotone.

$\textsc{MinDnfSize}_{\mathrm{mon}}$: *instance:* monotone Boolean formula φ and integer k
question: does φ have a DNF with at most k occurences of variables?

Theorem 3.5. $\textsc{MinDnfSize}_{\mathrm{mon}}$ *is* PP*-complete.*

Proof. A set A is in PP, if there exists a polynomial time bounded non-deterministic machine M that on input x has at least as many accepting as rejecting computation paths iff $x \in A$. The machine M is allowed to have accepting, rejecting, and non-deciding computation paths. Our polynomial time machine M that decides $\textsc{MinDnfSize}_{\mathrm{mon}}$ roughly works as follows. Consider input (φ, k). Let l be the maximum length of a monomial with variables from φ. Then M guesses a sequence w of $l+1$ bits. If the first bit of w equals 0, then it accepts, if the remaining bits encode an integer $< k$ — otherwise it halts non-deciding. In this way, k accepting computation paths are produced. If $w = 1v$ has first bit 1, then M checks in polynomial time (Theorem 3.2) whether v encodes a prime implicant (with variables in increasing order) for φ. If not, then it halts undecided. If yes, then this computation path splits in that many rejecting paths as the monomial v has variables. The smallest DNF of a monotone formula consists of all prime implicants of the formula. Hence, M on input (φ, k) has at least as many accepting as rejecting computation paths if and only if φ has a DNF with at most k occurences of variables. This shows that $\textsc{MinDnfSize}_{\mathrm{mon}}$ is in PP.

To show PP-hardness, we give a reduction $\textsc{MajSat} \leq_m^p \textsc{MinDnfSize}_{\mathrm{mon}}$. Consider an instance φ of $\textsc{MajSat}$ where φ has n variables, and let φ^{cd} be as in Definition 3.3. Observe that every prime implicant of φ^{cd} either is conform and hence consists of n variables, or it is not conform and consists of two variables x_i, y_i. If $\varphi \in \textsc{MajSat}$, then there are at least 2^{n-1} satisfying assignments to φ. Every satisfying assignment of φ induces a conform prime implicant of φ^{cd}. Every $i \in \{1, 2, \ldots, n\}$ induces the non-conform prime implicant $x_i \wedge y_i$. Hence, there are at least 2^{n-1} conform and n non-conform prime implicants of φ^{cd}. Because the minimum DNF consists exactly of all prime implicants, it follows that the minimum DNF of φ^{cd} has size at least $n \cdot 2^{n-1} + 2 \cdot n$. If $\varphi \notin$ MajSat, then the minimum DNF of φ^{cd} has size at most $n \cdot (2^{n-1} - 1) + 2 \cdot n$. The function that maps φ to $(\varphi^{cd}, n \cdot (2^{n-1} - 1) + 2 \cdot n)$ is polynomial time computable, and by the above observations it reduces $\overline{\text{MajSat}}$ to MinDnfSize$_{\text{mon}}$. Since PP is closed under complement, the PP-hardness of MinDnfSize$_{\text{mon}}$ follows. □

Accordingly, we can show that the function that on input a monotone formula φ outputs the size of the smallest DNF of φ is #P-complete. In [Val79] it is shown that computing the number of prime implicants of a monotone formula is #P-complete. Our result extends the latter since it additionally takes the size of the prime implicants into account.

Using a similar approach, one can show that counting satisfying assignments for monotone formulas and counting prime implicants for monotone formulas both are PP-complete. Notice that in [Val79] it is shown that given a monotone formula in 2CNF (all clauses consist of at most two variables) the function that calculates the number of satisfying assignments is #P-complete. From this result, it only follows that the problem to decide whether a monotone formula in 2CNF with n variables has at least 2^{n-1} satisfying assignments is PP-complete under polynomial time *Turing* reductions. Our approach yields PP-completeness under the stronger polynomial time many-one reduction.

One of the main reasons that an analogue to Theorem 3.5 for arbitrary formulas is unknown is the fact that polynomial time does not allow on input φ, k to guess a candidate for a DNF of length k. Therefore, we consider a variant of MinDnfSize where k is given in unary.

MinDnfSize′ : *instance:* Boolean formula φ and string 1^k
question: does φ have a DNF with at most k occurences of variables?

Theorem 3.6. MinDnfSize′ *is* Σ_2^{p}*-complete.*

Proof. MinDnfSize$_{\text{dnf}}$ reduces to MinDnfSize′ by the following function f. Let $|\varphi|$ denote the number of occurences of variables in φ. If $k \geq |\varphi|$, then $(\varphi, k) \in$ MinDnfSize$_{\text{dnf}}$ and $f(\varphi, k)$ is some fixed element in MinDnfSize′. If $k < |\varphi|$, then $f(\varphi, k) = (\varphi, 1^k)$. Clearly, f is polynomial time computable and reduces the problem MinDnfSize$_{\text{dnf}}$ to MinDnfSize′. MinDnfSize′ $\in \Sigma_2^{\mathsf{p}}$ can be shown using the standard guess-and-check approach. □

If we restrict the input to be monotone the complexity is lower.

MinDnfSize′$_{\text{mon}}$: *instance:* monotone Boolean formula φ and string 1^k
question: does φ have a DNF with at most k occurences of variables?

Theorem 3.7. MinDnfSize′$_{\text{mon}}$ *is* coNP*-complete.*

Proof. MinDnfSize′$_{\text{mon}}$ is coNP-hard: A formula φ is unsatisfiable if and only if φ^{cd} has $(x_i \wedge y_i)$ as its only prime implicants (where $i = 1, 2, \ldots, n$ for $x_1, \ldots, x_n$

are the variables of φ). Hence, φ is unsatisfiable if and only if $(\varphi^{cd}, 1^{2n}) \in$ $\textsc{MinDnfSize}'_{\text{mon}}$. This shows that $\textsc{MinDnfSize}'_{\text{mon}}$ is coNP-hard.

$\textsc{MinDnfSize}'_{\text{mon}} \in \mathsf{coNP}$: Consider the problem $A = \{(\varphi, 1^k) |$ the monotone formula φ has a minimal DNF of size $> k\}$. Note that A is the complement of $\textsc{MinDnfSize}'_{\text{mon}}$. A is in NP since one has to guess a disjunction D of monomials of size greater than k and less than $k + |\varphi|$ and check that all are different prime implicants for φ. If so, then the minimal DNF for φ has at least the size of D. Both the guess and the check are polynomial time computable. Hence, $\textsc{MinDnfSize}'_{\text{mon}} \in \mathsf{coNP}$. □

4 Computing DNFs

A DNF of a formula is a disjunction of (prime) implicants. For monotone formulas, the minimal DNF is unique and it is the disjunction of *all* prime implicants. In order to investigate the complexity of the search for all prime implicants, we use the following problem $\textsc{MorePrimi}_{\text{mon}}$. It has instances (φ, S), where φ is a formula and S is a set of monomials. A pair (φ, S) belongs to $\textsc{MorePrimi}_{\text{mon}}$ if S is a proper subset of a minimal DNF of φ. I.e., every monomial in S is a prime implicant for φ, but there is at least one more prime implicant for φ that must be added to S in order to make S a DNF for φ.

$\textsc{MorePrimi}_{\text{mon}}$:

instance: monotone Boolean formula φ and set S of monomials
question: is S a set of prime implicants of φ and $\varphi \not\equiv S$?

Theorem 4.1. $\textsc{MorePrimi}_{\text{mon}}$ *is* NP*-complete.*

There are monotone formulas whose minimal DNF have size exponential in the size of the formula. Therefore it is clear that the DNF cannot be computed in time polynomial in the length of the *input*. For such problems one would like to have algorithms that run in time polynomial in the length of the input plus the length of the output.

Definition 4.2. *[Pap97] A function f can be computed in* output-polynomial time*, if there is an algorithm A that for all x on input x outputs $f(x)$ and there is a polynomial q such that for all x, A on input x has running time $q(|x| + |f(x)|)$.*

An algorithm that cycles through all monomials and outputs those that are prime implicants of the monotone input formula, eventually outputs the minimal DNF of its input. For the special case of formulas that have long DNFs, this algorithm can be seen to have running time polynomial in the length of the output. For formulas with short DNFs, the running time of this straightforward algorithm is exponential in the length of the output. Anyway, we show that we cannot expect to find an algorithm that behaves significantly better than this straightforward approach.

Theorem 4.3. *The function that on input a monotone formula φ outputs the smallest DNF for φ is in output-polynomial time if and only if* $\mathsf{P} = \mathsf{NP}$.

Proof. Assume that A is an output-polynomial time algorithm for the considered problem, and let q be the polynomial bounding the run time of A. We show how to solve MorePrimi$_{\text{mon}}$ in polynomial time. For an instance (φ, S) of MorePrimi$_{\text{mon}}$, first check whether S is a set of prime implicants for φ, and reject if this is not the case. Then start A on input φ for $q(|\varphi| + |S|)$ steps. If A does not halt after $q(|\varphi|+|S|)$ steps, then S does not contain all prime implicants of φ, and our algorithm accepts. If A halts after $q(|\varphi|+|S|)$ steps, then accept if and only if S is a proper subset of the output of A. It is clear that this algorithm decides MorePrimi$_{\text{mon}}$. Its run time is bounded by the polynomial q, plus some polynomial overhead. Since MorePrimi$_{\text{mon}}$ is NP-complete (Theorem 4.1), it solves an NP-complete problem in polynomial time, and therefore $\mathsf{P} = \mathsf{NP}$.

For the other proof direction, assume that $\mathsf{P} = \mathsf{NP}$. The set $V = \{(w, S, \varphi) \mid w \text{ is a prefix of a prime implicant } C \text{ for } \varphi \text{ and } C \notin S\}$ is in NP. Our algorithm that computes a minimal DNF of a monotone input formula φ starts with S being the empty set, and uses V as an oracle to make S the set of all prime implicants of φ — and hence the minimal DNF of φ — using a prefix search technique. Intuitively spoken, every query to V yields one bit for the output. From $\mathsf{P} = \mathsf{NP}$ it then follows that the algorithm runs in output-polynomial time. ☐

Notice that a similar result is not known for arbitrary formulas.

As a final remark we return to the complexity of MorePrimi$_{\text{mon}}$ (Theorem 4.1). We have seen that the complexities of IsPrimi$_{\text{mon}}$ (Theorem 3.2) and MorePrimi$_{\text{mon}}$ differ. This is not the case for the corresponding non-monotone problems IsPrimi (Theorem 3.1) and MorePrimi.

Theorem 4.4. MorePrimi *is* DP-*complete.*

5 Equivalence and Isomorphism of Monotone Formulas

Deciding equivalence for arbitrary Boolean formulas is coNP-complete. The same holds for monotone formulas [Rei03]. However, if the monotone input formulas are given in k-CNF and DNF it is known that the problem is in P [EG95], even in RNC [BEGK00]. We improve these results by showing that logarithmic space suffices.

MonE$_{\text{const}}$: *instance:* irredundant, monotone Boolean formulas φ in k-CNF for a constant k and ψ in DNF
question: are φ and ψ equivalent?

Theorem 5.1. MonE$_{\text{const}} \in \mathsf{L}$.

Two Boolean formulas φ and ψ are *isomorphic* if and only if there exists a permutation — a bijective renaming — π of the variables such that φ and $\pi(\psi)$ are equivalent. Two Boolean formulas are *congruent* if they are isomorphic after negating some of the variables. For example $x_1 \wedge x_2$ and $\neg x_3 \wedge x_4$ are congruent. Such a negation of some variables with the bijective renaming of the variables is called *n-permutation.* A witness for the congruence of the above example is the n-permutation π that exchanges $\neg x_3$ and x_1 as well as x_4 and x_2.

We want to compare the problem of testing isomorphism for monotone Boolean formulas to the case of abitrary Boolean formulas. This provides a negative answer to a conjecture from [Rei03].

BoolIso$_{\text{mon}}$:	*instance:* monotone Boolean formulas φ and ψ
	question: are φ and ψ isomorphic?
BoolIso:	*instance:* Boolean formulas φ and ψ
	question: are φ and ψ isomorphic?
BoolCon:	*instance:* Boolean formulas φ and ψ
	question: are φ and ψ congruent?

Note that BoolCon is polynomially equivalent to BoolIso [BRS98].

Theorem 5.2. BoolIso$_{\text{mon}} \equiv_m^p$ BoolIso.

Proof. To show BoolIso$_{\text{mon}} \leq_m^p$ BoolIso we can choose the identy function as reduction function. We now show BoolIso $\leq_m^p$ BoolIso$_{\text{mon}}$. In [BRS98] it was shown that BoolIso $\leq_m^p$ BoolCon. Therefore, it suffices to show that BoolCon $\leq_m^p$ BoolIso$_{\text{mon}}$. The reduction function maps the instance (φ, ψ) of BoolCon to the pair $(\varphi^{cd}, \psi^{cd})$ (cf. Definition 3.3). We have to show $(\varphi, \psi) \in$ BoolCon $\Leftrightarrow (\varphi^{cd}, \psi^{cd}) \in$ BoolIso$_{\text{mon}}$.

$(\varphi, \psi) \in$ BoolCon $\Rightarrow (\varphi^{cd}, \psi^{cd}) \in$ BoolIso$_{\text{mon}}$: Let $(\varphi, \psi) \in$ BoolCon by an n-permutation π. Hence, φ and $\pi(\psi)$ are equivalent. We derive a permutation $\tilde{\pi}$ for $(\varphi^{cd}, \psi^{cd})$ from the n-permutation π in an elementary way. If π exchanges x_i with x_j, then $\tilde{\pi}$ exchanges x_i with x_j as well as y_i with y_j. And if π exchanges x_i with $\neg x_j$, then $\tilde{\pi}$ exchanges x_i with y_j as well as y_i with x_j. Note that $\tilde{\pi}$ does not make any remarkable changes on the $c(\psi)$- and $d(\psi)$-part of ψ^{cd} other than rearranging the terms in $c(\psi)$ and $d(\psi)$. We have to prove that φ^{cd} and $\tilde{\pi}(\psi^{cd})$ are equivalent and proceed by case differentiation of all possible monotone assignments.

$\exists i[x_i, y_i \in \mathcal{A}_m]$: Such assignments satisfy φ^{cd} and $\tilde{\pi}(\psi^{cd})$ by satisfying the conjunction $(x_i \wedge y_i)$.

$(\neg\exists i[x_i, y_i \in \mathcal{A}_m]) \wedge (\exists j[x_j, y_j \notin \mathcal{A}_m])$: None of the conjunctions of $d(\varphi)$ and $d(\psi)$ are satisfied by $\mathcal{A}_m$. Furthermore the disjunction $(x_j \vee y_j)$ in $c(\varphi)$ and $c(\psi)$ is not satisfied by $\mathcal{A}_m$ and consequently φ^{cd} and $\tilde{\pi}(\psi^{cd})$ are not satisfied.

It remains to verify the conform assignments: These are assignments that contain only one of the variables x_i and y_i for every $i \leq n$. They do not satisfy $d(\varphi)$ and $d(\psi)$ but do satisfy $c(\varphi)$ and $c(\psi)$. It remains to check $r(\varphi)$ and $\tilde{\pi}(r(\psi))$. From the facts that φ and $\pi(\psi)$ are equivalent and a conform assignment for φ^{cd} and $\tilde{\pi}(\psi^{cd})$ just simulates an assignment for φ and $\pi(\psi)$ it follows that the truth tables of φ^{cd} and $\tilde{\pi}(\psi^{cd})$ are identical in this case. Thus the truth tables of φ^{cd} and $\tilde{\pi}(\psi^{cd})$ are identical with respect to all possible assignments and therefore φ^{cd} and ψ^{cd} are isomorphic.

$(\varphi, \psi) \in$ BoolCon $\Leftarrow (\varphi^{cd}, \psi^{cd}) \in$ BoolIso$_{\text{mon}}$: A permutation $\tilde{\pi}$ for $(\varphi^{cd}, \psi^{cd}) \in$ BoolIso$_{\text{mon}}$ is called *proper* if and only if (1) whenever x_i and

x_j are exchanged, then so are y_i and y_j, and (2) whenever x_i and y_j are exchanged, then so are y_i and x_j.

Claim. For all $(\varphi^{cd}, \psi^{cd}) \in$ BOOLISO$_{\text{mon}}$ with more than two x-variables there is a proper permutation $\tilde{\pi}_p$ that ensures the equivalence of φ^{cd} and $\tilde{\pi}_p(\psi^{cd})$.

Proof. Suppose that the proposition of the claim does not hold. Then there exists a pair of formulas $(\varphi^{cd}_{im}, \psi^{cd}_{im}) \in$ BOOLISO$_{\text{mon}}$ with more than two x-variables for which no proper permutation exists. As a consequence φ^{cd}_{im} and $\tilde{\pi}_{im}(\psi^{cd}_{im})$ are equivalent for some improper permutation $\tilde{\pi}_{im}$. We distinguish between the two cases of $\tilde{\pi}_{im}$ being improper.

$\exists i[\tilde{\pi}_{im}$ exchanges x_i with x_j but not y_i with $y_j]$: Hence, $\tilde{\pi}_{im}$ exchanges y_i with $b \in \{x_k : k \leq n, k \neq j\} \cup \{y_k : k \leq n, k \neq j\}$. We examine the assignment $\mathcal{A}_m = \{x_j, b\}$. The conjunction $(x_j \wedge b)$ in $\tilde{\pi}_{im}(d(\psi))$ is satisfied by $\mathcal{A}_m$ and so is $\tilde{\pi}_{im}(\psi^{cd}_{im})$. But $\mathcal{A}_m$ does not satisfy φ^{cd}_{im}. Note that the conjunction $(x_j \wedge b)$ is not present in $d(\varphi)$ and therefore $\mathcal{A}_m$ cannot satisfy $d(\varphi)$. Furthermore not all of the disjunctions of $c(\varphi)$ contain x_j or b because there are more than two x-variables in φ^{cd}_{im} and ψ^{cd}_{im}. Thus the two formulas φ^{cd}_{im} and $\tilde{\pi}_{im}(\psi^{cd}_{im})$ cannot be equivalent. This is a contradiction to our assumption.

$\exists i[\tilde{\pi}_{im}$ exchanges x_i with y_j but not y_i with $x_j]$: An analogous argumentation as above shows that the formulas φ^{cd}_{im} and $\tilde{\pi}_{im}(\psi^{cd}_{im})$ cannot be equivalent. This is a contradiction to our assumption. Hence, the claim follows.

As a consequence, there is a proper permutation $\tilde{\pi}_p$ for every $(\varphi^{cd}, \psi^{cd}) \in$ BOOLISO$_{\text{mon}}$. A proper permutation only works on the $r(\psi)$-part of the ψ^{cd}-formula and only rearranges the terms in $c(\psi)$ and $d(\psi)$. Given a proper permutation $\tilde{\pi}_p$ we can easily derive an n-permutation π for (φ, ψ). If $\tilde{\pi}_p$ exchanges x_i with x_j as well as y_i with y_j, then π exchanges x_i with x_j. And if $\tilde{\pi}_p$ exchanges x_i with y_j as well as y_i with x_j, then π exchanges x_i with $\neg x_j$. Since the y-variables are placeholders for the negative literals, we see that π ensures $(\varphi, \psi) \in$ BOOLCON. This concludes the proof of BOOLCON $\leq^p_m$ BOOLISO$_{\text{mon}}$.

Thus we have established BOOLISO $\equiv^p_m$ BOOLISO$_{\text{mon}}$. □

In [AT00] it is shown that BOOLISO is not complete for Σ^{p}_2 unless the Polynomial Time Hierarchy collapses. As a consequence of Theorem 5.2, this holds for BOOLISO$_{\text{mon}}$ as well.

6 Concluding Remarks

We compared the complexity of problems related to the construction of Disjunctive Normal Forms for non-monotone and monotone formulas. We proved that finding an algorithm that computes a minimal DNF for a monotone formula in output-polynomial time is the same as solving $\mathsf{P} = \mathsf{NP}$. A similar result for arbitrary formulas is still open. Anyway, we assume that at least $\mathsf{P} = \mathsf{PSPACE}$ is the consequence. Although we proved that calculating the size of a minimal DNF for a monotone formula is PP-complete (resp. $\#\mathsf{P}$-complete), even a PSPACE upper bound for the non-monotone case is open.

Some problems for formulas are easier to decide in the monotone case than for arbitrary formulas. Among them are finding prime implicants (NP- vs. Σ_2^{p}-complete) and calculating the size of a smallest equivalent DNF (PP-complete vs. unknown). On the other hand, there are problems whose complexity stays the same for monotone formulas. We could show this polynomial time equivalence for isomorphism testing and counting satisfying assignments.

Deciding equivalence for monotone formulas is coNP-complete [Rei03] like it is for Boolean formulas. Nevertheless we were able to prove a log-space upper bound for the special case $\textsc{Mone}_{\mathrm{const}}$ of equivalence testing. The complexity of the general problem $\textsc{Mone}$ without a constant bound for the clause size (which is equivalent to $\textsc{MorePrimi}_{\mathrm{mon}}$ for instances (φ, S) with φ in CNF) remains open.

References

[AT00] M. Agrawal and T. Thierauf. The formula isomorphism problem. *SIAM Journal on Computing*, 30(3):990–1009, 2000.

[BEGK00] E. Boros, K. Elbassioni, V. Gurvich, and L. Khachiyan. An efficient incremental algorithm for generating all maximal independent sets in hypergraphs of bounded dimension. *Parallel Processing Letters*, 10(4):253–266, 2000.

[BRS98] B. Borchert, D. Ranjan, and F. Stephan. On the computational complexity of some classical equivalence relations on Boolean functions. *Theory of Computing Systems*, 31(6):679–693, 1998.

[Czo99] S.L.A. Czort. The complexity of minimizing Disjunctive Normal Form formulas. Technical Report IR-130, Dept. of Computer Science, University of Aarhus, January 1999.

[EG95] T. Eiter and G. Gottlob. Identifying the minimal transversals of a hypergraph and related problems. *SIAM Journal on Computing*, 24(6):1278–1304, 1995.

[FK96] M.L. Fredman and L. Khachiyan. On the complexity of dualization of monotone Disjunctive Normal Forms. *Journal of Algorithms*, 21:618–628, 1996.

[Mas79] W.J. Masek. Some NP-complete set covering problems. Unpublished manuscript, 1979.

[Pap94] C.H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

[Pap97] C.H. Papadimitriou. NP-completeness: a retrospective. In *Proceedings of 24th ICALP*, volume 1256 of *Lecture Notes in Computer Science*, pages 2–6, 1997.

[PY84] C.H. Papadimitriou and M. Yannakakis. The complexity of facets (and some facets of complexity). *Journal of Computer and System Sciences*, 28:224–259, 1984.

[Rei03] S. Reith. On the complexity of some equivalence problems for propositional calculi. In *Proceedings of MFCS 2003*, volume 2747 of *Lecture Notes in Computer Science*, pages 632–641, 2003.

[Uma01] C. Umans. The minimum equivalent DNF problem and shortest implicants. *Journal of Computer and System Sciences*, 63(4):597–611, 2001.

[Val79] L.G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979.