Complexity of DNF Minimization and Isomorphism Testing for Monotone Formulas ⋆

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

[a] *University of Kentucky, Dept. of Computer Science, Lexington KY 40506–0046*

[b] *Friedrich-Schiller-Universität Jena, Institut für Informatik, D–07737 Jena, Germany*

Abstract

We investigate the complexity of finding prime implicants and minimum equivalent DNFs for Boolean formulas, and of testing equivalence and isomorphism of monotone formulas. For DNF related problems, the complexity of the monotone case differs strongly 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 the equivalent problem for monotone formulas is in L. We show PP-completeness of checking if the minimum size of a DNF for a monotone formula is at most k, and for k in unary, we show that the complexity of the problem drops to coNP. In [Uma01] a similar problem for arbitrary formulas was shown to be Σ_2^{P}-complete. We show that calculating the minimum equivalent 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.

Key words: Computational complexity; Monotone formulas; Prime implicants; Formula isomorphism

⋆ A preliminary version of this work was presented at MFCS 2005 [GHM05]

* Corresponding author.

Email addresses: goldsmit@cs.uky.edu (Judy Goldsmith), hagen@cs.uni-jena.de (Matthias Hagen), mundhenk@cs.uni-jena.de (Martin Mundhenk).

[1] Partially supported by NSF grant ITR-0325063.

[2] Partially supported by a Landesgraduiertenstipendium Thüringen.

1 Introduction

Monotone formulas are Boolean formulas that contain only conjunction and disjunction, but not negation, as connectives (cf. Section 2 for formal definitions). To solve the satisfiability problem for monotone formulas is trivial: the only unsatisfiable monotone formulas are the constant 0 and equivalent formulas, as every other monotone formula can be satisfied by setting all variables to `true`. Hence, to check whether a given (arbitrarily nested) monotone formula (potentially including the constant 0) is satisfiable it suffices to check only one assignment. the computational complexity of the satisfiability problem for monotone formulas is thus much simpler than the NP-complete satisfiability problem for arbitrary Boolean formulas. Counting the number of satisfying assignments, however, has the same complexity for monotone and for arbitrary formulas [Val79]. Hence, it is interesting to compare the complexity of problems for arbitrary formulas and for monotone formulas.

In the first part of this paper (Sections 3–7), we investigate the complexity of problems related to calculating smallest equivalent disjunctive normal forms (DNFs), which consist of prime implicants of the formula. For arbitrary Boolean formulas, a smallest equivalent DNF must be constructed from the set of prime implicants, and the problem of deciding if a given DNF is a smallest equivalent DNF of a given formula is not known to be computable in polynomial space.

Note that for monotone formulas, there is no choice as the smallest DNF consists of all prime implicants. In Sections 3–5 we consider problems of checking, finding, and counting prime implicants (cf. the first four questions in Figure 1 for more precise problem statements and respective results). 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 of 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. Counting satisfying assignments is shown to have the same complexity for monotone formulas as for arbitrary formulas. Counting prime implicants for monotone formulas we show to be PP-complete, whereas for arbitrary formulas, an upper bound for this problem is $\mathsf{PP}^{\mathsf{NP}}$, but the exact complexity — in terms of completeness — is open.

Next, in Section 6, we consider the complexity of calculating the size of a smallest DNF (cf. the fifth question in Figure 1), which, it turns out, 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 (excepting trivial cases). It would seem that this is necessary to allow the problem to be decided within an alternating nondeterministic polynomial time bound, because the smallest DNF of a (monotone) formula may have size exponential in the length

question about a formula φ	complexity for	
	arbitrary φ	monotone φ
Is C a prime implicant of φ ?	DP-complete (Theorem 5)	in L (Theorem 6)
Does φ have a prime implicant of size $\leq k$?	Σ_2^{P}-complete [Uma01]	NP-complete (Theorem 7)
Does φ have $\geq k$ prime implicants ?	in $\mathsf{PP}^{\mathsf{NP}}$	PP-complete (Theorem 8)
Does φ have an odd number of prime implicants ?	in $\oplus\mathsf{P}^{\mathsf{NP}}$	$\oplus$P-complete (Theorem 13)
Does φ have a DNF of size $\leq k$?		
k in unary	Σ_2^{P}-complete (Theorem 17)	coNP-complete (Theorem 18)
k in binary	in EXPTIME	PP-complete (Theorem 16)
Is S a set of prime implicants of φ and $S \not\equiv \varphi$?	DP-complete (Theorem 22)	NP-complete (Theorem 19)

Fig. 1. Summary of complexity results of Sections 3–7

of the formula. The exact complexity of this problem for arbitrary formulas is open, though it is Σ_2^{P}-hard (which follows from [Uma01]) and in EXPTIME. For monotone formulas, however, 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 7 we investigate the problem of checking whether a set S of prime implcants of a formula φ contains all prime implicants of φ (cf. the last question in Figure 1) and show it to be DP-complete for arbitrary formulas whereas it is NP-complete for monotone formulas. Furthermore, we consider the complexity of calculating the smallest DNF for a monotone formula, and it becomes clear that the smallest DNF is not polynomial time computable as the output might be too large. 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 [JPY88]. We show that the smallest DNF for a monotone formula is output-polynomial time computable if and only if $\mathsf{P} = \mathsf{NP}$. Note that even calculating the size of the smallest DNF is shown to be PP-complete in Section 6.

In the second part of the paper (Section 8) we turn to considering 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 DNF and ψ is in CNF (conjunctive normal form), the equivalence problem remains coNP-complete for arbitrary formulas, but for monotone formulas an upper bound between P and coNP-complete was proven in [FK96, GK99]. In the case that one of the input formulas consists only of terms of bounded length, it is known that the problem is in P [EG95, MP97], even in RNC [BEGK00]. We give an L-algorithm improving these results. Finally, we refute a conjecture from [Rei03], by showing that checking whether two given formulas are isomorphic has exactly the same complexity for arbitrary formulas 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 Boolean 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 a *monomial*, and a disjunction is called a *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 Boolean 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 $C \rightarrow \varphi$ is valid. A monomial C is a *prime implicant* of φ if and only if (1) C is an implicant of φ and (2) for every proper subset $S \subset C$ it holds that S is not an implicant of φ. Notice that case (1) is easy to decide for monotone formulas, but is coNP-complete for arbitrary formulas. 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 $S \subseteq C - \{\ell_i\}$ it holds that $(C - \{\ell_i\}) \rightarrow S$ is valid. If S is an implicant of φ, then $S \rightarrow \varphi$ is valid. Both valid formulas together yield that $(C - \{\ell_i\}) \rightarrow \varphi$ is valid, 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.

In our proofs, we will define reductions that transform formulas into monotone formulas, such that satisfying assignments of the basic formula induce prime implicants of the monotone formula.

Definition 1 *Let φ be a Boolean formula with variables $x_1, \ldots, x_n$ and connectives $\wedge$, $\vee$, and $\neg$, in which 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 $\mathcal{A}'_m = \{x_i \mid \mathcal{A} \text{ maps } x_i \text{ to } \texttt{true}\} \cup \{y_i \mid \mathcal{A} \text{ maps } x_i \text{ to } \texttt{false}\}$. 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}.

Proposition 2 *Let $\mathcal{A}$ be an assignment for φ. The following are equivalent.*

(1) $\mathcal{A}$ satisfies φ.
(2) $\mathcal{A}'_m$ is a prime implicant of φ^c.
(3) $\mathcal{A}'_m$ is a prime implicant of φ^{cd}.

PROOF. If $\mathcal{A}$ satisfies φ, then $\mathcal{A}'_m$ satisfies $r(\varphi)$. Since $\mathcal{A}'_m$ is conform, it satisfies $c(\varphi)$ too. Let z be any variable in $\mathcal{A}'_m$. Then for some a, $z \in \{x_a, y_a\}$. Since $\mathcal{A}'_m$ is conform, $\mathcal{A}'_m - \{z\}$ does not satisfy $x_a \wedge y_a$, and hence $\mathcal{A}'_m - \{z\}$ does not satisfy φ^c. Hence, $\mathcal{A}'_m$ is a prime implicant of φ^c.

Since prime implicant $\mathcal{A}'_m$ satisfies φ^c, it also satisfies φ^{cd}. By the same argument as above it follows that removing any variable z from $\mathcal{A}'_m$ leaves an assignment $\mathcal{A}'_m - \{z\}$ that satisfies neither $x_a \vee y_a$ nor $x_a \wedge y_a$ for some a, and hence it does not satisfy φ^{cd}.

Finally, take any conform prime implicant $\mathcal{A}'_m$ for φ^{cd}. Since it is conform, it does not satisfy $d(\varphi)$. Hence, it satisfies $r(\varphi)$. By construction of $\mathcal{A}'_m$ and $r(\varphi)$ it follows that $\mathcal{A}$ satisfies φ. $\square$

Proposition 3 *Let φ be a formula with variables $x_1, x_2, \ldots, x_n$. φ has m satisfying assignments if and only if φ^{cd} has $n + m$ prime implicants.*

PROOF. Every non-conform prime implicant of φ^{cd} has the form $\{x_i, y_i\}$, and there are n of these non-conform prime implicants. It follows from Proposition 2 that m is the number of conform prime implicants of φ^{cd}. Hence, $n + m$ is the number of prime implicants of φ^{cd}. $\square$

Proposition 4 *Let φ be a formula with variables $x_1, x_2, \ldots, x_n$. φ has m satisfying assignments if and only if φ^{cd} has $2^{2n} - 3^n + m$ satisfying assignments.*

PROOF. Consider any conform assignment for φ^{cd}. It sets exactly one of x_i and y_i to `true`. Every "larger" assignment — i.e., an assignment that sets additional variables to `true` — is a non-conform assignment satisfying φ^{cd}, because for some i, both x_i and y_i are set to `true` by such an assignment. Using this observation, the assignments that satisfy φ^{cd} can be split into two disjoint sets: the set A_c of assignments that are conform to satisfying assignments of φ, and the set S of non-conform assignments that satisfy at least one pair $x_i \wedge y_i$. The set A_c has the same size as the set of satisfying assignments of φ. The set of assignments that, for each i, contain at most one variable of x_i and y_i, has size 3^n. Since φ^{cd} has $2n$ variables, there are $2^{2n} - 3^n + |A_c|$ satisfying assignments for φ^{cd}. $\square$

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 [Qui53]. This is not the case for non-monotone formulas, where the smallest DNF is a subset of the set of all prime implicants, and it is NP-hard to select the prime implicants which give the smallest equivalent DNF [Mas79]. See also [Czo99] for an overview on the complexity of calculating equivalent 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), $\oplus$P (parity P), and PSPACE (polynomial space). The inclusion structure is

$$\mathsf{L} \subseteq \mathsf{P} \subseteq \begin{matrix}\mathsf{NP}\\ \mathsf{coNP}\end{matrix} \subseteq \mathsf{DP} \subseteq \begin{matrix}\Sigma_2^{\mathsf{P}}\\ \mathsf{PP}\end{matrix} \subseteq \mathsf{PSPACE}$$

and $\mathsf{P} \subseteq \oplus\mathsf{P} \subseteq \mathsf{PSPACE}$. All considered classes except L are closed downwards under $\leq_m^p$-reduction, and both PP and $\oplus$P are closed under complement. Closely related to PP is the function class #P. See [Pap94] for definitions of these classes. As natural complete problems for NP, coNP, DP, PP, and $\oplus$P we consider Sat (is the Boolean formula φ satisfiable?), Unsat (is φ unsatisfiable?), Sat-Unsat (given (φ, ψ), is $\varphi \in$ Sat and $\psi \in$ Unsat?), MajSat (do at least half of the assignments satisfy φ?), and $\oplus$Sat (is the number of satisfying assignments for φ odd?), respectively.

3 Checking prime implicants

Prime implicants are the basics of minimum equivalent DNFs. In this section we concentrate on checking whether a monomial is a prime implicant of a formula and consider the following problems.

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

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

For arbitrary formulas, this problem is shown to be DP-complete (Theorem 5), whereas for monotone formulas it is much easier, namely in L (Theorem 6).

Theorem 5 IsPrimi *is* DP*-complete.* [3]

PROOF. 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 not an implicant of $\neg\varphi \vee (\neg\psi \wedge z)$. This proves the DP-hardness of IsPrimi.

A set A is in DP [PY84] if and only if A is the intersection of a set in NP and a set in coNP. The set

$$L_1 = \{(\varphi, M) \mid \text{ for all literals } \ell \in M\text{: } M - \{\ell\} \text{ is not an implicant of } \varphi\}$$

is clearly in NP, and the set

$$L_2 = \{(\varphi, M) \mid \ M \text{ is an implicant of } \varphi\}$$

is clearly in coNP. It is straightforward that IsPrimi is $L_1 \cap L_2$. □

[3] This result was independently shown in [UVS06].

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

Theorem 6 IsPrimi$_{\text{mon}}$ *is in* L*.*

PROOF. On input φ and M, there are two things to do: check whether M is an implicant of φ — i.e., check whether *assignment* M satisfies φ — and check for all $x_i \in M$ that $M - \{x_i\}$ is not an implicant of φ — i.e., check whether *assignment* $M - \{x_i\}$ does not satisfy φ. Evaluating a formula under a given assignment can be performed in logarithmic space [Lyn77]. It follows directly that IsPrimi$_{\text{mon}}$ is in L. □

4 Existence of 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.

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?

Theorem 7 PrimiSize$_{\text{mon}}$ *is* NP*-complete.*

PROOF. PrimiSize$_{\text{mon}}$ is in NP: given a monotone formula φ with n variables and an integer k, guess a term of at most $\min\{k, n\}$ variables and check whether the guess is an implicant of φ. Both the guess and the check can be performed in polynomial time.

PrimiSize$_{\text{mon}}$ is NP-hard: we show that 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.

If $\varphi \in$ Sat, then there exists a satisfying assignment $\mathcal{A}$ for φ. By Proposition 2 it follows that $\mathcal{A}'_m$ is a prime implicant of φ^c. Because $\mathcal{A}'_m$ is conform, it contains exactly

one variable from each of the n pairs x_i, y_i. Hence, $(\varphi^c, n) \in \text{PRIMISIZE}_{\text{mon}}$.

If $(\varphi^c, n) \in \text{PRIMISIZE}_{\text{mon}}$, then there exists an implicant M of φ^c with exactly n variables. Because M then is an implicant of every $(x_i \vee y_i)$, exactly one of x_i and y_i is contained in M. Hence, M is conform to a satisfying assignment $\mathcal{A}$ for φ, and therefore $\varphi \in \text{SAT}$. $\square$

5 Counting prime implicants

We consider the complexity of counting the number (resp., the parity) of satisfying assignments and of prime implicants of monotone formulas. Counting satisfying assignments turns out to have the same complexity for monotone formulas as for arbitrary formulas (Theorems 8 and 10). Counting prime implicants seems easier for monotone formulas than for arbitrary ones (Theorem 11).

$\oplus\text{SAT}_{\text{mon}}$:	*instance:*	monotone Boolean formula φ
	question:	does φ have an odd number of satisfying assignments?
$\text{THRESHSAT}_{\text{mon}}$:	*instance:*	monotone Boolean formula φ and integer k
	question:	does φ have at least k satisfying assignments?

Theorem 8 $\text{THRESHSAT}_{\text{mon}}$ *is* PP*-complete.*

PROOF. $\text{THRESHSAT}_{\text{mon}}$ is in PP. Let (φ, k) be an instance of $\text{THRESHSAT}_{\text{mon}}$, where φ has n variables. The nondeterministic computation that guesses an assignment $\mathcal{A}$ and accepts it if and only if $\mathcal{A}$ satisfies φ is polynomial time bounded and has 2^n computation paths. If $k > 2^n$, reject, else, if $k \leq 2^{n-1}$, add $2^n - 2k$ accepting paths; otherwise, when $k > 2^{n-1}$, add $2k - 2^n$ rejecting paths. This nondeterministic computation has at least half of all computation paths accepting if and only if φ has at least k satisfying assignments.

To prove PP-hardness, we give a polynomial time reduction from MAJSAT. It maps φ to $(\varphi^{cd}, 2^{2n} - 3^n + 2^{n-1})$, where φ has variables $x_1, x_2, \ldots, x_n$. By Proposition 4, it follows for all instances φ of MAJSAT, that $\varphi \in \text{MAJSAT} \Leftrightarrow (\varphi^{cd}, 2^{2n} - 3^n + 2^{n-1}) \in \text{THRESHSAT}_{\text{mon}}$. $\square$

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 follows that deciding 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.

With the same proof idea we characterize the complexity of calculating the number of satisfying assignments for monotone formulas. Valiant [Val79] proved $\#\mathsf{P}$-completeness of computing the number of satisfying assignments for monotone formulas in 2CNF under polynomial time truth table reduction.

Corollary 9 *The function, which on input φ, a monotone formula, outputs the number of satisfying assignments of φ, is $\#\mathsf{P}$-complete under polynomial time many-one reduction.*

Theorem 10 $\oplus\text{SAT}_{\text{mon}}$ *is $\oplus\mathsf{P}$-complete.*

PROOF. It is straightforward to see that $\oplus\text{SAT}_{\text{mon}}$ is in $\oplus\mathsf{P}$. To prove $\oplus\mathsf{P}$-hardness, we use the construction from the proof of Theorem 8. For an instance φ of $\oplus\text{SAT}$ with variables $x_1, x_2, \ldots, x_n$ having m satisfying assignments, φ^{cd} has $2^{2n} - 3^n + m$ satisfying assignments by Proposition 4. This number is odd if and only if m is even, hence, the function that maps φ to φ^{cd} polynomial time reduces $\overline{\oplus\text{SAT}}$ to $\oplus\text{SAT}_{\text{mon}}$. Since $\oplus\mathsf{P}$ is closed under complement, the $\oplus\mathsf{P}$-hardness of $\oplus\text{SAT}_{\text{mon}}$ follows. □

As mentioned before, the problem of finding short prime implicants for Boolean formulas is Σ_2^{p}-complete [Uma01], mostly because checking whether a guessed term is an implicant requires a coNP oracle. This gives rise to the conjecture that counting the number of prime implicants is $\mathsf{PP}^{\mathsf{NP}}$-hard for Boolean formulas. We consider the problem for monotone formulas and prove that its complexity is lower.

$\oplus\text{PRIMI}_{\text{mon}}$:	*instance:*	monotone Boolean formula φ
	question:	does φ have an odd number of prime implicants?
$\text{THRESHPRIMI}_{\text{mon}}$:	*instance:*	monotone Boolean formula φ and integer k
	question:	does φ have at least k prime implicants?

Theorem 11 $\text{THRESHPRIMI}_{\text{mon}}$ *is PP-complete.*

PROOF. It is straightforward to see that $\text{THRESHPRIMI}_{\text{mon}}$ is in PP. To show PP-hardness of $\text{THRESHPRIMI}_{\text{mon}}$, we give a reduction from MAJSAT. For an instance φ of MAJSAT with variables $x_1, \ldots, x_n$, let φ^{cd} be the monotone formula as in Definition 1. The polynomial time reduction function maps φ to $(\varphi^{cd}, 2^{n-1} + n)$.

Consider $\varphi \in$ MajSat, where φ has n variables. It follows that there are at least 2^{n-1} satisfying assignments to φ. Every satisfying assignment of φ induces a conform prime implicant of φ^c and hence of φ^{cd}. Every $i \in \{1, 2, \ldots, n\}$ induces the non-conform prime implicant $x_i \wedge y_i$. Hence, φ^{cd} has at least $2^{n-1} + n$ prime implicants, i. e., $(\varphi^{cd}, 2^{n-1} + n) \in$ ThreshPrimi$_{\text{mon}}$.

If $\varphi \notin$ MajSat, then there are fewer than 2^{n-1} conform prime implicants of φ^{cd}. However, there may be implicants of φ^{cd} that are not conform to an assignment of φ. Consider such an implicant C; then C contains both x_i and y_i for some i. Then C is covered by the prime implicant $x_i \wedge y_i$ of φ^{cd}. Hence, φ^{cd} has fewer than $2^{n-1} + n$ prime implicants, i. e., $(\varphi^{cd}, 2^{n-1} + n) \notin$ ThreshPrimi$_{\text{mon}}$. □

The arguments of the above proof can also be used to show the following.

Corollary 12 ([Val79]) *The function, which on input φ, a monotone formula, yields the number of prime implicants of φ, is #P-complete.*

Theorem 13 $\oplus$Primi$_{\text{mon}}$ *is $\oplus$P-complete.*

PROOF. Containment of $\oplus$Primi$_{\text{mon}}$ in $\oplus$P is straightforward. To prove $\oplus$P-hardness, we give a reduction from $\oplus$Sat. The reduction function maps an instance φ of Sat with n variables to a monotone formula $f(\varphi)$ where

$$f(\varphi) = \begin{cases} \varphi^{cd}, & \text{if } n \text{ is even} \\ \varphi^{cd} \vee z, & \text{if } n \text{ is odd (for a new variable } z) \end{cases}$$

Consider φ with n variables and m satisfying assignments. If n is even, then $f(\varphi)$ has $n + m$ prime implicants (Proposition 3), and m is odd if and only if $n + m$ is odd. If n is odd, then $f(\varphi)$ has $n + m + 1$ prime implicants (Proposition 3), and m is odd if and only if $n + m + 1$ is odd. That is, $\varphi \in \oplus$Sat $\Leftrightarrow f(\varphi) \in \oplus$Primi$_{\text{mon}}$. □

6 Size of disjunctive normal forms

An arbitrary Boolean formula may have several different DNFs. Since minimum equivalent DNFs of a formula are disjunctions of prime implicants, a natural question arises. How hard is it to calculate the size of a minimum equivalent DNF?

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

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

Theorem 14 ([Uma01]) MinDnfSize$_{\text{dnf}}$ *is* Σ_2^{p}*-complete.*

For monotone formulas, the minimum equivalent DNF is unique, and can be obtained from any monotone DNF by removing monomials. This makes it possible to decide the problem under consideration in L.

MinDnfSize$_{\text{mondnf}}$:	*instance:*	monotone Boolean formula φ in DNF and integer k
	question:	does φ have a DNF with at most k occurrences of variables?

Lemma 15 MinDnfSize$_{\text{mondnf}}$ *is in* L*.*

PROOF. For a monotone DNF $\varphi = M_1 \vee M_2 \vee \ldots \vee M_m$ it holds that φ is not minimum if and only if two monomials are equal (i. e., $M_i = M_j$ for some $i \neq j$) or contained in another (i. e., $M_i \subset M_j$ for some i, j). Essentially, the minimum equivalent DNF for φ is $\bigvee_{i \in I} M_i$ for

$$I = \{j \mid 1 \leq j \leq m \wedge \forall i \neq j \colon (M_i \not\supset M_j \ \vee \ M_i = M_j \rightarrow i < j)\} \ .$$

The algorithm given as Algorithm 1 calculates the size of $\bigvee_{i \in I} M_i$ and compares it to the given upper bound.

The correctness of algorithm `minDNFSize` for monotone DNFs is straightforward, but we must check the space required. The for-loops require two logspace counters. The checks $M_i \supset M_j$ and $M_i = M_j$ can also be performed in logspace. The variable *sizesum* requires at most space logarithmic in the length of the input formula. □

If the input is an arbitrary formula, the problem of deciding whether there is an equivalent DNF with at most k variable occurences is Σ_2^{p}-hard (which follows from [Uma01]). It is clear that the problem is in $\mathsf{EXPTIME}$, but it is not known if the problem is in PSPACE. We show PP-completeness when the input is monotone.

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

Theorem 16 MinDnfSize$_{\text{mon}}$ *is* PP*-complete.*

Algorithm 1 `minDNFSize`(φ, k)

Input: a monotone formula $\varphi = M_1 \vee M_2 \vee \ldots \vee M_m$ in DNF and integer k
Output: Yes, if φ has a DNF of size $\leq k$, and No, otherwise

```
1: sizesum ← 0
2: for i ← 1, m do
3:     countthismonomial ← true
4:     for j ← 1, m; j ≠ i do
5:         if M_i ⊃ M_j then
6:             countthismonomial ← false
7:         else if M_i = M_j and i > j then
8:             countthismonomial ← false
9:         end if
10:    end for
11:    if countthismonomial then
12:        sizesum ← sizesum + |M_i|
13:    end if
14: end for
15: if sizesum ≤ k then
16:    output Yes and stop
17: end if
18: output No
```

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 on which the machine enters a "?" state that is distinct from the accept and reject states. Our polynomial-time machine M that decides MinDnfSize$_{\text{mon}}$ 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 $< k-1$, otherwise it halts undecided. This produces k accepting computation paths. If $w = 1v$ has first bit 1, then M checks in polynomial time (Theorem 6) whether v encodes a prime implicant (with variables in increasing order) for φ. If so, then this computation path splits into as many rejecting paths as the monomial v has variables, otherwise it halts undecided. 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 occurrences of variables. This shows that MinDnfSize$_{\text{mon}}$ is in PP.

To show PP-hardness, we give a reduction $\overline{\text{MajSat}} \leq_m^p$ MinDnfSize$_{\text{mon}}$. Consider an instance φ of MajSat with variables $x_1, \ldots, x_n$ and m satisfying assignments. Then φ^{cd} has $m+n$ prime implicants (Proposition 3), of which m are conform and contain n variables each, and n are not conform and contain 2 variables each. The minimum DNF of a monotone formula consists of all prime implicants. If $\varphi \in$ MajSat, 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 $\text{MinDnfSize}_{\text{mon}}$. Since PP is closed under complement, the PP-hardness of $\text{MinDnfSize}_{\text{mon}}$ follows. □

Accordingly, we can show that the function, which on input φ, a monotone formula, 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.

One of the main reasons that an analogue to Theorem 16 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 occurrences of variables?

Theorem 17 MinDnfSize′ *is* Σ_2^{p}*-complete.*

PROOF. $\text{MinDnfSize}_{\text{dnf}}$ ist Σ_2^{p}-complete [Uma01], and reduces to MinDnfSize′ by the following function f. Let $|\varphi|$ denote the number of occurrences of variables in φ. If $k \geq |\varphi|$, then $(\varphi, k) \in \text{MinDnfSize}_{\text{dnf}}$ and $f(\varphi, k)$ is some fixed element in MinDnfSize′. If $k < |\varphi|$, then $f(\varphi, k) = (\varphi, 1^k)$. Clearly, the reduction function f is polynomial time computable. MinDnfSize′ $\in \Sigma_2^{\text{p}}$ can be shown using the standard guess-and-check approach. □

If we restrict the input to monotone formulas, the complexity is lower.

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

Theorem 18 $\text{MinDnfSize}'_{\text{mon}}$ *is* coNP*-complete.*

PROOF. $\text{MinDnfSize}'_{\text{mon}}$ is coNP-hard: Let φ be a formula with variables $x_1, \ldots, x_n$. Then φ is unsatisfiable if and only if φ^{cd} has $(x_1 \wedge y_1)$,…,$(x_n \wedge y_n)$ as its only prime

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

$\text{MinDnfSize}'_{\text{mon}} \in \text{coNP}$: Consider the problem $A = \{(\varphi, 1^k) |$ the monotone formula φ has a minimum equivalent DNF of size $> k\}$. Note that A is the complement of $\text{MinDnfSize}'_{\text{mon}}$. 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. Both the guess and the check are polynomial time computable. Hence, $\text{MinDnfSize}'_{\text{mon}} \in \text{coNP}$. □

7 Computing disjunctive normal forms

A DNF of a formula is a disjunction of (prime) implicants. For monotone formulas, the minimum equivalent 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, $\text{MorePrimi}_{\text{mon}}$. It has instances (φ, S), where φ is a formula and S is a set of monomials. A pair (φ, S) belongs to $\text{MorePrimi}_{\text{mon}}$ if S is a proper subset of a minimum equivalent DNF of φ. In other words, 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.

$\text{MorePrimi}_{\text{mon}}$:	*instance:*	monotone Boolean formula φ and set S of monomials
	question:	is S a set of prime implicants of φ and $\varphi \not\equiv S$?

In order to prove the NP-completeness of $\text{MorePrimi}_{\text{mon}}$, we just count the number of prime implicants of φ^{cd}.

Theorem 19 (see also [EG95]) $\text{MorePrimi}_{\text{mon}}$ *is* NP*-complete.*

PROOF. Given a monotone formula φ and a set S of monomials, it is an easy polynomial test to check whether S is a set of prime implicants for φ. If this holds, then $(\varphi, S) \in \text{MorePrimi}_{\text{mon}}$ iff there exists a prime implicant for φ that is not in S. Hence, $\text{MorePrimi}_{\text{mon}}$ is in NP. To prove NP-hardness, we give a reduction from Sat. For an instance φ of Sat with variables $x_1, \ldots, x_n$, let S be the set $S = \{(x_i \wedge y_i) \mid i = 1, 2, \ldots, n\}$. We show that $\varphi \in \text{Sat}$ if and only if $(\varphi^{cd}, S) \in \text{MorePrimi}_{\text{mon}}$.

Consider $\varphi \in \text{Sat}$. Then φ^{cd} has at least $n+1$ prime implicants, by Proposition 3. Since S consists only of n prime implicants of φ^{cd}, it follows that $(\varphi^{cd}, S) \in \text{MorePrimi}_{\text{mon}}$.

Consider $\varphi \notin \text{Sat}$. Then φ^{cd} has n prime implicants (Proposition 3). Since S contains all n prime implicants of φ^{cd}, it follows that $(\varphi^{cd}, S) \notin \text{MorePrimi}_{\text{mon}}$. □

There are monotone formulas whose minimum equivalent DNFs 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 it would be advantageous to have algorithms that run in time polynomial in the length of the input plus the length of the output.

Definition 20 ([JPY88]) *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 minimum equivalent DNF of its input. For the special case of formulas that have exponetially 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. We show that we cannot expect to find an algorithm that behaves significantly better than this straightforward approach.

Theorem 21 *The function, which on input φ, a monotone formula, 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 $\textsc{MorePrimi}_{\text{mon}}$ in polynomial time. For an instance (φ, S) of $\textsc{MorePrimi}_{\text{mon}}$, 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 $\textsc{MorePrimi}_{\text{mon}}$. Its run time is bounded by the polynomial q, plus some polynomial overhead. Since $\textsc{MorePrimi}_{\text{mon}}$ is NP-complete (Theorem 19), 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 $V = \{(w, S, \varphi) \mid w$ is a prefix of a prime implicant C for φ and $C \notin S\}$ is in NP. Our algorithm that computes a minimum equivalent 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 minimum equivalent 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. For monotone CNF one can use the algorithm from [FK96] in order to obtain the DNF in output-quasi-polynomial time.

As a final remark we return to the complexity of MorePrimi$_{\text{mon}}$ (Theorem 19). We have seen that the complexities of IsPrimi$_{\text{mon}}$ (Theorem 6) and MorePrimi$_{\text{mon}}$ differ. This is not the case for non-monotone formulas: the complexities of MorePrimi and IsPrimi (Theorem 5) for non-monotone formulas are equal.

MorePrimi :	*instance:*	Boolean formula φ and set S of monomials
	question:	is every monomial $C \in S$ a prime implicant of φ, and $\varphi \not\equiv S$?

Theorem 22 MorePrimi *is* DP*-complete.*

PROOF. The proof relies on the same ideas as the proof of Theorem 5. We show that IsPrimi $\leq_m^p$ MorePrimi, using the polynomial time reduction function given by the mapping

$$(\varphi, C) \mapsto \begin{cases} (\varphi \wedge (z \vee z'), \{z\}), & \text{if } C = \lambda \\ (\varphi \vee z, \{C\}), & \text{if } C \neq \lambda \end{cases}$$

where z and z' are new variables that do not appear in φ nor in C.

If $C = \lambda$ is a prime implicant of φ, then φ is valid, and therefore both z and z' are prime implicants for $\varphi \wedge (z \vee z')$. Hence, $(\varphi \wedge (z \vee z'), \{z\})$ is in MorePrimi. If $C \neq \lambda$ is a prime implicant of φ, then φ is not valid, and therefore z is a prime implicant for $\varphi \vee z$. Hence, $(\varphi \vee z, \{C\})$ is in MorePrimi.

If C is not a prime implicant for φ, then $(\varphi \vee z, \{C\})$ is not in MorePrimi. If $C = \lambda$, then φ is not valid, and therefore z is not a prime implicant for $(\varphi \wedge (z \vee z'), \{z\})$, which shows that $(\varphi \wedge (z \vee z'), \{z\})$ is not in MorePrimi.

To show that MorePrimi is in DP, consider an instance (φ, S) for MorePrimi, where $S = \{C_1, C_2, \ldots, C_k\}$ and $C_i = \{\ell_1^i, \ell_2^i, \ldots, \ell_{k_i}^i\}$. We express $\varphi \not\equiv S$ as a query $q(y)$ to a Sat oracle with

$$q(\varphi, S) = \neg(\varphi \leftrightarrow S) \ .$$

Moreover, we can express that the C_is are prime using the function p from the proof of Theorem 5 as

$$p'(\varphi, S) = \bigwedge_{i=1}^{k} p(\varphi, C_i) \ .$$

Finally, we can express that all monomials in S are implicants for φ as a query to an Unsat oracle using the function n from the proof of Theorem 5 by

$$t(\varphi, S) = \bigvee_{i=1}^{k} n(\varphi, C_i) \ .$$

It is not difficult to see that $(\varphi, S) \in$ MorePrimi iff $(q(\varphi, S) \wedge p'(\varphi, S), t(\varphi, S)) \in$

SAT-UNSAT. Since q, p', and t are polynomial time computable, MOREPRIMI $\in$ DP follows. $\square$

8 Equivalence and isomorphism of monotone formulas

Deciding equivalence for arbitrary Boolean formulas is coNP-complete. The same holds for monotone formulas [Rei03]. The problem MONET — MO(notone) N(ormal form) E(quivalence) T(est) — asks for the equivalence of two monotone formulas φ in DNF and ψ in CNF. MONET is decidable in time $O(n^{\log n})$ [FK96], and it belongs to coNP using only $\log^2 n$ many nondeterministic bits [EGM03, KS03]. We consider its restriction where the size of the monomials in the DNF is bounded. A formula is in k-DNF if it is in DNF and each monomial has at most k literals.

MONET$_k$:	*instance:*	irredundant, monotone Boolean formulas φ in k-DNF and ψ in CNF
	question:	are φ and ψ equivalent?

MONET$_k$ is known to be in P [EG95, MP97], even in RNC [BEGK00]. We improve these results by showing that MONET$_k$ can be decided in logarithmic space.

Theorem 23 MONET$_k$ *is in* L*, for every integer* k*.*

In order to proof Theorem 23 we use a property of transversal hypergraphs given as Lemma 24.

A *hypergraph* $\mathcal{H}$ is a family E of subsets of a finite set V. The elements of V are called *vertices*, the elements of E, *edges*. A set $T \subseteq V$ is called a *transversal* of $\mathcal{H}$ if T has a non-empty intersection with every edge of $\mathcal{H}$. The minimal transversals of $\mathcal{H}$ with respect to set inclusion form the *transversal hypergraph* $Tr(\mathcal{H})$. The problem of deciding if a given hypergraph $\mathcal{G}$ is the transversal hypergraph of another hypergraph $\mathcal{H}$ is called TRANS-HYP. MONET$_k$ is equivalent to the problem TRANS-HYP with bounded edge-size (see [EG95]), because the monomials of φ and the clauses of ψ can be seen as edges of two hypergraphs that are transversal hypergraphs of each other.

The following Lemma is proven in [EG95] as part of Theorem 5.2. Note that we only changed notation to better fit in the MONET setting.

Lemma 24 ([EG95]) *Let* φ *in* k*-DNF with the set of monomials* M_φ *and* ψ *in CNF with the set of clauses* C_ψ *be two irredundant, monotone Boolean formulas. If* $k \geq 2$*, then:*

$$(\varphi, \psi) \in \text{MONET}_k \Longleftrightarrow C_\psi \subseteq Tr(M_\varphi) \wedge E_1 \wedge E_2, \tag{1}$$

with

$$E_1 \equiv \neg\exists M \subseteq V, |M| \leq k : M \in Tr(C_\psi) \wedge M \notin M_\varphi$$
$$E_2 \equiv \neg\exists C'_\psi \subseteq C_\psi, |C'_\psi| = k+1 : \forall C \in C_\psi : C \not\subseteq \{x \in V : d(x, C'_\psi) > 1\},$$

where $d(x, C'_\psi)$ denotes the number of sets in C'_ψ that contain the variable x.

We now analyze the complexity of the test implicit in Lemma 24 with the intention of tightening the complexity bound for MONET_k. Therefore, we examine the complexity of checks that will be used in the proof of Theorem 23 below.

We first show that we can check in logarithmic space whether φ and ψ are irredundant.

$\text{IsIrred}_{\text{mon}}$:	*instance:*	monotone Boolean formula φ in CNF/DNF with variable set V
	question:	is φ irredundant?

Lemma 25 $\text{IsIrred}_{\text{mon}}$ *is in* L.

PROOF. An appropriate algorithm is given as Algorithm 2. The correctness of Algorithm 2 is straightforward, but we must check the space required. The for-loops require three logspace counters: the counters in lines 3 and 4 need only count to $|\varphi|$, the number of terms in φ. The counter in line 5 need only count to $|M_i|$. Another two logspace counters are used for checkvariable_2 and for counting to $|M_i|$ in line 10. The checkvariable_1 requires constant space. The terms of φ need not be copied to be compared. The counters add up to logarithmic space. □

We next have to examine the test whether a set of variables is contained in a family of variable sets.

IsIn:	*instance:*	a set M of subsets $M_i \subseteq V$ and a subset $S \subseteq V$
	question:	$S \in M$?

Lemma 26 IsIn *is in* L.

PROOF. An appropriate algorithm is given as Algorithm 3. The correctness of Algorithm 3 is straightforward, but we must check the space requirements. The for-loops require two logspace-counters to count to $|M|$ and $|S|$. The checkvariable requires constant space. This all adds up to logarithmic space. □

Algorithm 2 `is_irred`(φ)

Input: a monotone formula φ in normal form with the set V of variables
Output: Yes, if φ is irredundant, and No, otherwise

checkvariable$_1 \leftarrow 1$
checkvariable$_2 \leftarrow 0$
for all terms T_i of φ **do**
    for all other terms $T_j \neq T_i$ of φ **do**
        for all variables $x \in T_i$ **do**
            if $x \in T_j$ **then**
                checkvariable$_2 \leftarrow$ checkvariable$_2 + 1$
            end if
        end for
        if checkvariable$_2 = |T_i|$ **then**
            checkvariable$_1 \leftarrow 0$
        end if
    end for
end for
if checkvariable$_1 = 1$ **then**
    output Yes
else
    output No
end if

Algorithm 3 `is_in`(M, S)

Input: a set M of subsets $M_i \subseteq V$ and a subset $S \subseteq V$
Output: Yes, if $S \in M$, and No, otherwise

for all $M_i \in M$ **do**
    checkvariable $\leftarrow 1$
    for all $x \in S$ **do**
        if $x \notin M_i$ **then**
            checkvariable $\leftarrow 0$
        end if
    end for
    if checkvariable $= 1$ **then**
        output Yes and **stop**
    end if
end for
output No

Finally, we need a test to decide whether a monotone clause is contained in the irredundant CNF of a monotone DNF.

IsClause:	*instance:*	a monotone clause C and an irredundant, monotone DNF φ with the set M_φ of monomials
	question:	Is C contained in the irredundant CNF of φ?

Lemma 27 IsClause *is in* L.

PROOF. An appropriate algorithm is given as Algorithm 4. It checks whether C has a non-empty intersection with every monomial of φ (lines 1 to 11 of the listing below), and whether no $C \setminus \{x\}$ for each $x \in C$ has a non-empty intersection with all monomials (lines 12 to 28). The correctness of Algorithm 4 is straightforward, but we must check the space requirements. Let the input size n be the number of variable occurences in φ and C.

To know the current monomial, the for-loops in line 1 and line 14 could manage counters that give the number of already checked monomials. These counters have to count till $|M_\varphi|$. Hence, they are logarithmically bounded in n. An analogous argumentation holds for the for-loops in line 3 and line 16. To know the current variable, they manage a counter that gives a variable index, which is clearly logarithmic in n. And again, the for-loop in line 12 is handled analogously.

It remains to check the variables *count* and *hit*. The maximal value of *count* is the size of a largest monomial of φ, which is logarithmic in n. The maximal value of *hit* is $|M_\varphi|$, which is also logarithmic in n. Altogether, logarithmic space suffices to run the described algorithm. □

Using the procedures for IsIrred, IsIn, and IsClause we can now prove Theorem 23.

PROOF. [of Theorem 23] Let n denote the number of variable occurrences in φ and ψ. We assume that every monomial of φ has size at most k for a constant k. An appropriate machine is able to determine this k by counting in logarithmic space. The irredundancy of φ and ψ can be checked in logarithmic space according using the procedure from the proof of Lemma 25. We will show that the right hand side of (1) can be verified in logarithmic space. Therefore, we describe the work of an appropriate machine T. The machine uses the logspace procedures `is_in` (cf. Lemma 26) and `is_clause` (cf. Lemma 27) as subroutines. Note that procedure calls can be space-efficiently simulated by using pointers to cells on input or working tapes of T where parameters needed for the procedure call start.

$C_\psi \subseteq Tr(M_\varphi)$: T calls `is_clause` systematically for φ and every clause in C_ψ. To know which clause is currently tested, T counts the number of tested clauses. This counter can be managed in logarithmic space in the size of C_ψ.

Algorithm 4 `is_clause`(C, φ)

Input: monotone clause C and monotone DNF φ with the set M_φ of monomials
Output: Yes, if C is contained in the irredundant CNF of φ, and No, otherwise

```
1: for all M_i ∈ M_φ do
2:     count ← 0
3:     for all x ∈ M_i do
4:         if x_j ∈ C then
5:             count ← count + 1
6:         end if
7:     end for
8:     if count = 0 then
9:         output No and stop
10:    end if
11: end for
12: for all x_i ∈ C do
13:    hit = |M_φ|
14:    for all M_j ∈ M_φ do
15:        count ← 0
16:        for all x_k ∈ M_j do
17:            if x_k ∈ C and k ≠ i then
18:                count ← count + 1
19:            end if
20:        end for
21:        if count > 0 then
22:            hit ← hit − 1
23:        end if
24:    end for
25:    if hit = 0 then
26:        output No and stop
27:    end if
28: end for
29: output Yes
```

E_1: Every constant-sized M has to be checked. To do this, T systematically generates the candidates. To know which candidate is the actual candidate, T counts the number of already checked candidates. The number of possible candidates is bounded by $1 + \binom{n}{1} + \binom{n}{2} + \ldots + \binom{n}{k} = O(n^k)$. Hence the counter needs $k \cdot \log(n)$ bits. Because of the constant size of M the machine T can write down the whole current candidate. For every candidate M a procedure described in the proof of Theorem 6 and analogous to `is_clause` answers the question whether M is a prime implicant of ψ. If the answer is Yes, then T calls `is_in` to know whether M is in φ. Altogether E_1 can be verified in logarithmic space.

E_2: Only a constant number of clauses form the current candidate set for the E_2-test.

By systemtically generating the candidate sets, T is able to know the monomials that form the current candidate by counting the candidates. The counter must count to $\binom{n}{k+1} = O(n^{k+1})$, hence, logarithmic space suffices. Because of the constant size of the candidates C'_ψ, the machine T can manage pointers to each clause in the current candidate set C'_ψ on some working tape. Hence, using `is_in`, T is able to check, for every variable in every clause, if the variable is contained in more than one element of C'_ψ.

Altogether we can conclude that logarithmic space suffices to decide $\textsc{Monet}_k$. □

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 maps $\neg x_3$ to x_1 and x_4 to x_2.

We want to compare the problem of testing isomorphism for monotone Boolean formulas to the case of arbitrary Boolean formulas. We provide a negative answer to a conjecture from [Rei03] by showing that testing isomorphism for monotone Boolean formulas is as hard as for arbitrary formulas.

$\textsc{BoolIso}_{\text{mon}}$:	*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?

In [BRS98] it was shown that (1) BoolCon is polynomial time equivalent to BoolIso, (2) BoolIso is coNP-hard, and (3) the graph isomorphism problem reduces in polynomial time to BoolIso.

Theorem 28 $\textsc{BoolIso}_{\text{mon}}$ *is polynomial time equivalent to* BoolIso.

PROOF. To show $\textsc{BoolIso}_{\text{mon}} \leq^p_m$ BoolIso we can choose the identity function as reduction function. We now show BoolIso $\leq^p_m$ $\textsc{BoolIso}_{\text{mon}}$. In [BRS98] it was shown that BoolIso $\leq^p_m$ BoolCon. Therefore, it suffices to show BoolCon $\leq^p_m$ $\textsc{BoolIso}_{\text{mon}}$. The reduction function maps the instance (φ, ψ) of BoolCon to the

pair $(\varphi^{cd}, \psi^{cd})$ (cf. Definition 1). It is therefore sufficient to show $(\varphi, \psi) \in \textsc{BoolCon} \Leftrightarrow (\varphi^{cd}, \psi^{cd}) \in \textsc{BoolIso}_{\text{mon}}$.

$(\varphi, \psi) \in \textsc{BoolCon} \Rightarrow (\varphi^{cd}, \psi^{cd}) \in \textsc{BoolIso}_{\text{mon}}$: Let $(\varphi, \psi) \in \textsc{BoolCon}$ 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 maps x_i to x_j, then $\tilde{\pi}$ maps x_i to x_j as well as y_i to y_j. And if π maps x_i to $\neg x_j$, then $\tilde{\pi}$ maps x_i to y_j as well as y_i to 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 must 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 $\tilde{\pi}(r(\psi))$. Given that φ and $\pi(\psi)$ are equivalent, and that a conform assignment for φ^{cd} and $\tilde{\pi}(\psi^{cd})$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.

$(\varphi, \psi) \in \textsc{BoolCon} \Leftarrow (\varphi^{cd}, \psi^{cd}) \in \textsc{BoolIso}_{\text{mon}}$: A permutation $\tilde{\pi}$ for $(\varphi^{cd}, \psi^{cd}) \in \textsc{BoolIso}_{\text{mon}}$ is called *proper* if and only if (1) whenever x_i is mapped to x_j, so is y_i to y_j, and (2) whenever x_i is mapped to y_j, so is y_i to x_j.

Claim 29 *For all $(\varphi^{cd}, \psi^{cd}) \in \textsc{BoolIso}_{\text{mon}}$ 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 $(\varphi^{cd}_{im}, \psi^{cd}_{im}) \in \textsc{BoolIso}_{\text{mon}}$ 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.

$\exists i[\tilde{\pi}_{im}$ maps x_i to x_j but not y_i to $y_j]$: Hence, $\tilde{\pi}_{im}$ maps y_i to some $b \in \{x_k : k \leq n, k \neq j\} \cup \{y_k : k \leq n, k \neq j\}$. 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_{im}^{cd} and ψ_{im}^{cd}. Thus, the two formulas φ_{im}^{cd} and $\tilde{\pi}_{im}(\psi_{im}^{cd})$ cannot be equivalent. This is a contradiction to our assumption.

$\exists i[\tilde{\pi}_{im}$ maps x_i to y_j but not y_i to $x_j]$: An analogous argument to the above shows that the formulas φ_{im}^{cd} and $\tilde{\pi}_{im}(\psi_{im}^{cd})$ 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 $(\varphi^{cd}, \psi^{cd}) \in \text{BoolIso}_{\text{mon}}$. 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 $\tilde{\pi}_p$ maps x_i to x_j and y_i to y_j, then π maps x_i to x_j. And if $\tilde{\pi}_p$ maps x_i to y_j as well as y_i to x_j, then π maps x_i to $\neg x_j$. Since the y-variables are placeholders for the negative literals, we see that π ensures $(\varphi, \psi) \in \text{BoolCon}$. This concludes the proof of $\text{BoolCon} \leq_m^p \text{BoolIso}_{\text{mon}}$.

Thus we have established $\text{BoolIso} \equiv_m^p \text{BoolIso}_{\text{mon}}$. □

In [AT00] it is shown that BoolIso is not complete for Σ_2^{p} unless the Polynomial Time Hierarchy collapses. As a consequence of Theorem 28, this holds for $\text{BoolIso}_{\text{mon}}$ as well.

9 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 the minimum equivalent DNF for a monotone formula in output-polynomial time is the same as solving $\mathsf{P} = \mathsf{NP}$, though a similar result for arbitrary formulas is still open, and we assume that at least $\mathsf{P} = \mathsf{PSPACE}$ is the consequence. Although we proved that calculating the size of a minimum equivalent DNF for a monotone formula is PP-complete (resp., #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 short 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] as for arbitrary Boolean formulas. Nevertheless we were able to prove a log-space upper bound for

the special case, MONET_k, of equivalence testing. The complexity of the general problem MONET without a constant bound for the clause size (which is equivalent to $\text{MOREPRIMI}_{\text{mon}}$ for instances (φ, S) with φ in CNF) remains open.

10 Acknowledgments

The authors thank the anonymous referee for his valuable comments and suggestions. We also gratefully acknowledge the editing done by Tom Dodson.

References

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

[BEGK00] Endre Boros, Khaled M. Elbassioni, Vladimir Gurvich, and Leonid 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] Bernd Borchert, Desh Ranjan, and Frank Stephan. On the computational complexity of some classical equivalence relations on Boolean functions. *Theory of Computing Systems*, 31(6):679–693, 1998.

[Czo99] Sebastian Lukas Arne Czort. The complexity of minimizing disjunctive normal formulas. Technical Report IR-130, University of Aarhus, Department of Computer Science, January 1999.

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

[EGM03] Thomas Eiter, Georg Gottlob, and Kazuhisa Makino. New results on monotone dualization and generating hypergraph transversals. *SIAM Journal on Computing*, 32(2):514–537, 2003.

[FK96] Michael L. Fredman and Leonid Khachiyan. On the complexity of dualization of monotone disjunctive normal forms. *Journal of Algorithms*, 21(3):618–628, 1996.

[GHM05] Judy Goldsmith, Matthias Hagen, and Martin Mundhenk. Complexity of DNF and isomorphism of monotone formulas. In *Mathematical Foundations of Computer Science 2005, 30th International Symposium, MFCS 2005, Gdansk, Poland, August 29 - September 2, 2005, Proceedings*, pages 410–421, 2005.

[GK99] Vladimir Gurvich and Leonid Khachiyan. On generating the irredundant conjunctive and disjunctive normal forms of monotone Boolean functions. *Discrete Applied Mathematics*, 96-97:363–373, 1999.

[JPY88] David S. Johnson, Christos H. Papadimitriou, and Mihalis Yannakakis. On

generating all maximal independent sets. *Information Processing Letters*, 27(3):119–123, 1988.

[KS03] Dimitris J. Kavvadias and Elias C. Stavropoulos. Monotone Boolean dualization is in $\mathsf{coNP}[\log^2 n]$. *Information Processing Letters*, 85(1):1–6, 2003.

[Lyn77] Nancy A. Lynch. Log space recognition and translation of parenthesis languages. *Journal of the ACM*, 24(4):583–590, 1977.

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

[MP97] Nina Mishra and Leonard Pitt. Generating all maximal independent sets of bounded-degree hypergraphs. In *Proceedings of the Tenth Annual Conference on Computational Learning Theory (COLT 1997), July 6-9, 1997, Nashville, Tennessee, USA*, pages 211–217, 1997.

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

[PY84] Christos H. Papadimitriou and Mihalis Yannakakis. The complexity of facets (and some facets of complexity). *Journal of Computer and System Sciences*, 28(2):244–259, 1984.

[Qui53] Willard van Orman Quine. Two theorems about truth functions. *Boletin de la Sociedad Matemática Mexicana*, 10:64–70, 1953.

[Rei03] Steffen Reith. On the complexity of some equivalence problems for propositional calculi. In *Mathematical Foundations of Computer Science 2003, 28th International Symposium, MFCS 2003, Bratislava, Slovakia, August 25-29, 2003, Proceedings*, pages 632–641, 2003.

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

[UVS06] Christopher Umans, Tiziano Villa, and Alberto L. Sangiovanni-Vincentelli. Complexity of two-level logic minimization. *IEEE Transactions on CAD of Integrated Circuits and Systems*, 25(7):1230–1246, 2006.

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