I N F S Y S

R E S E A R C H

R E P O R T

INSTITUT FÜR INFORMATIONSSYSTEME

ABTEILUNG WISSENSBASIERTE SYSTEME

NEW RESULTS ON MONOTONE DUALIZATION AND GENERATING HYPERGRAPH TRANSVERSALS

Thomas Eiter Georg Gottlob Kazuhisa Makino

INFSYS RESEARCH REPORT 1843-02-05

APRIL 2002

Institut für Informationssysteme
Abtg. Wissensbasierte Systeme
Technische Universität Wien
Favoritenstraße 9-11
A-1040 Wien, Austria
Tel: +43-1-58801-18405
Fax: +43-1-58801-18493
sek@kr.tuwien.ac.at
www.kr.tuwien.ac.at

TECHNISCHE UNIVERSITÄT WIEN

INFSYS RESEARCH REPORT

INFSYS RESEARCH REPORT 1843-02-05, APRIL 2002

NEW RESULTS ON MONOTONE DUALIZATION AND GENERATING HYPERGRAPH TRANSVERSALS

Thomas Eiter,[1] Georg Gottlob,[2] and Kazuhisa Makino[3]

Abstract. We consider the problem of dualizing a monotone CNF (equivalently, computing all minimal transversals of a hypergraph), whose associated decision problem is a prominent open problem in NP-completeness. We present a number of new polynomial time resp. output-polynomial time results for significant cases, which largely advance the tractability frontier and improve on previous results. Furthermore, we show that duality of two monotone CNFs can be disproved with limited nondeterminism. More precisely, this is feasible in polynomial time with $O(\chi(n) \cdot \log n)$ suitably guessed bits, where $\chi(n)$ is given by $\chi(n)^{\chi(n)} = n$; note that $\chi(n) = o(\log n)$. This result sheds new light on the complexity of this important problem.

Keywords: Dualization, hypergraphs, transversal computation, output-polynomial algorithms, combinatorial enumeration, treewidth, hypergraph acyclicity, limited nondeterminism.

[1]Institut für Informationssysteme, Technische Universität Wien, Favoritenstraße 9-11, 1040 Wien, Austria. E-mail: eiter@kr.tuwien.ac.at.

[2]Institut für Informationssysteme, Technische Universität Wien, Favoritenstraße 9-11, 1040 Wien, Austria. E-mail: gottlob@dbai.tuwien.ac.at.

[3]Division of Systems Science, Graduate School of Engineering Science, Osaka University, Toyonaka, Osaka 560-8531, Japan. E-mail: makino@sys.es.osaka-u.ac.jp

Acknowledgements: This work was supported in part by the Austrian Science Fund (FWF) project Z29-INF, by TU Wien through a scientific collaboration grant, and by the Scientific Grant in Aid of the Ministry of Education, Science, Sports and Culture of Japan.

A shorter version of this paper appears in: Proceedings of the 34th ACM Symposium on Theory of Computing (STOC-02), May 19-21, 2002, Montreal, Quebec, Canada.

Contents

1 Introduction

Recall that the prime CNF of a monotone Boolean function f is the unique formula $\varphi = \bigwedge_{c \in S} c$ in conjunctive normal form where S is the set of all prime implicates of f, i.e., minimal clauses c which are logical consequences of f. In this paper, we consider the following problem:

	Problem DUALIZATION
Input:	The prime CNF φ of a monotone Boolean function $f = f(x_1, \ldots, x_m)$.
Output:	The prime CNF ψ of its dual $f^d = \overline{f}(\overline{x}_1, \ldots, \overline{x}_m)$.

It is well known that DUALIZATION is equivalent to the TRANSVERSAL COMPUTATION problem, which requests to compute the set of all minimal transversals (i.e., minimal hitting sets) of a given hypergraph $\mathcal{H}$, in other words, the *transversal hypergraph* $Tr(\mathcal{H})$ of $\mathcal{H}$. Actually, these problems can be viewed as the same problem, if the clauses in a monotone CNF φ are identified with the sets of variables they contain. DUALIZATION is a search problem; the associated decision problem DUAL is to decide whether two given monotone prime CNFs φ and ψ represent a pair (f, g) of dual Boolean functions. Analogously, the decision problem TRANS-HYP associated with TRANSVERSAL COMPUTATION is deciding, given hypergraphs $\mathcal{H}$ and $\mathcal{G}$, whether $\mathcal{G} = Tr(\mathcal{H})$.

DUALIZATION and several problems which are like transversal computation known to be computationally equivalent to problem DUALIZATION (see [15]) are of interest in various areas such as database theory (e.g. [38, 49]), machine learning and data mining (e.g., [6, 7, 12, 22]), game theory (e.g. [26, 42, 43]), artificial intelligence (e.g., [21, 28, 29, 44]), mathematical programming (e.g., [5]), and distributed systems (e.g., [18, 27]) to mention a few.

While the output CNF ψ can be exponential in the size of φ, it is currently not known whether ψ can be computed in *output-polynomial* (or *polynomial total*) *time*, i.e., in time polynomial in the combined size of φ *and* ψ. Any such algorithm for DUALIZATION (or for TRANSVERSAL COMPUTATION) would significantly advance the state of the art of several problems in the above application areas. Similarly, the complexity of DUAL (equivalently, TRANS-HYP) is open since more than 20 years now (cf. [3, 15, 30, 31, 33]).

Note that DUALIZATION is solvable in polynomial total time on a class $\mathcal{C}$ of hypergraphs iff DUAL is in PTIME for all pairs $(\mathcal{H}, \mathcal{G})$, where $\mathcal{H} \in \mathcal{C}$ [3]. DUAL is known to be in co-NP and the best currently known upper time-bound is quasi-polynomial time [17, 19, 47]. Determining the complexities of DUALIZATION and DUAL, and of equivalent problems such as the transversal problems, is a prominent open problem. This is witnessed by the fact that these problems are cited in a rapidly growing body of literature and have been referenced in various survey papers and complexity theory retrospectives, e.g. [30, 34, 40].

Given the importance of monotone dualization and equivalent problems for many application areas, and given the long standing failure to settle the complexity of these problems, emphasis was put on finding tractable cases of DUAL and corresponding polynomial total-time cases of DUALIZATION. In fact, several relevant tractable classes were found by various authors; see e.g. [4, 8, 9, 10, 12, 14, 15, 20, 35, 36, 39, 41] and references therein. Moreover, classes of formulas were identified on which

DUALIZATION is not just polynomial total-time, but where the conjuncts of the dual formula can be enumerated with *incremental polynomial delay*, i.e., with delay polynomial in the size of the input plus the size of all conjuncts so far computed, or even with *polynomial delay*, i.e., with delay polynomial in the input size only. On the other hand, there are also results which show that certain well-known algorithms for DUALIZATION are not polynomial-total time. For example, [15, 39] pointed out that a well-known sequential algorithm, in which the clauses c_i of a CNF $\varphi = c_1 \wedge \cdots \wedge c_m$ are processed in order $i = 1, \ldots, m$, is not polynomial-total time in general. Most recently, [46] showed that this holds even if an optimal ordering of the clauses is assumed (i.e., they may be arbitrarily arranged for free).

Main Goal. The main goal of this paper is to present important new polynomial total time cases of DUALIZATION and, correspondingly, PTIME solvable subclasses of DUAL which significantly improve previously considered classes. Towards this aim, we first present a new algorithm DUALIZE and prove its correctness. DUALIZE can be regarded as a generalization of a related algorithm proposed by Johnson, Yannakakis, and Papadimitriou [31]. As other dualization algorithms, DUALIZE reduces the original problem by self-reduction to smaller instances. However, the subdivision into subproblems proceeds according to a particular order which is induced by an arbitrary fixed ordering of the variables. This, in turn, allows us to derive some bounds on intermediate computation steps which imply that DUALIZE, when applied to a variety of input classes, outputs the conjuncts of ψ with polynomial delay or incremental polynomial delay. In particular, we show positive results for the following input classes:

- **Degenerate CNFs.** We generalize the notion of k-degenerate graphs [50] to hypergraphs and define *k-degenerate monotone CNFs* resp. *hypergraphs*. We prove that for any constant k, DUALIZE works with polynomial delay on k-degenerate CNFs. Moreover, it works in output-polynomial time on $O(\log n)$-degenerate CNFs.

- **Read-k CNFs.** A CNF is *read-k*, if each variable appears at most k times in it. We show that for read-k CNFs, problem DUALIZATION is solvable with polynomial delay, if k is constant, and in total polynomial time, if $k = O(\log(\|\varphi\|)$. Our result for constant k significantly improves upon the previous best known algorithm [12], which has a higher complexity bound, is not polynomial delay, and outputs the clauses of ψ in no specific order. The result for $k = O(\log \|\varphi\|)$ is a non-trivial generalization of the result in [12], which was posed as an open problem [11].

- **Acyclic CNFs.** There are several notions of hypergraph resp. monotone CNF acyclicity [16], where the most general and well-known is α-acyclicity. As shown in [15], DUALIZATION is polynomial total time for β-acyclic CNFs; β-acyclicity is the hereditary version of α-acyclicity and far less general. A similar result for α-acyclic prime CNFs was left open. (For non-prime α-acyclic CNFs, this is trivially as hard as the general case.) In this paper, we give a positive answer and show that for α-acyclic (prime) φ, DUALIZATION is solvable with polynomial delay.

- **Formulas of Bounded Treewidth.** The *treewidth* [45] of a graph expresses its degree of cyclicity. Treewidth is an extremely general notion, and bounded treewidth generalizes almost all other notions of near-acyclicity. Following [13], we define the treewidth of a hypergraph resp. monotone CNF φ as the treewidth of its associated (bipartite) variable-clause incidence graph. We show that DUALIZATION is solvable with polynomial delay (exponential in k) if the treewidth of φ is bounded by a constant k, and in polynomial total time if the treewidth is $O(\log\log\|\varphi\|)$.

- **Recursive Applications of** DUALIZE **and k-CNFs.** We show that if DUALIZE is applied recursively and the recursion depth is bounded by a constant, then DUALIZATION is solved in polynomial total time. We apply this to provide a simpler proof of the known result [8, 15] that monotone k-CNFs (where each conjunct contains at most k variables) can be dualized in output-polynomial time.

After deriving the above results, we turn our attention (in Section 5) to the fundamental computational nature of problems DUAL and TRANS-HYP in terms of complexity theory.

Limited nondeterminism. In a landmark paper, Fredman and Khachiyan [17] proved that problem DUAL can be solved in quasi-polynomial time. More precisely, they first gave an algorithm A solving the problem in $n^{O(\log^2 n)}$ time, and then a more complicated algorithm B whose runtime is bounded by $n^{4\chi(n)+O(1)}$ where $\chi(n)$ is defined by $\chi(n)^{\chi(n)} = n$. As noted in [17], $\chi(n) \sim \log n/\log\log n = o(\log n)$; therefore, duality checking is feasible in $n^{o(\log n)}$ time. This is the best upper bound for problem DUAL so far, and shows that the problem is most likely not NP-complete.

A natural question is whether DUAL lies in some lower complexity class based on other resources than just runtime. In the present paper, we advance the complexity status of this problem by showing that its complement is feasible with *limited nondeterminism*, i.e, by a nondeterministic polynomial-time algorithm that makes only a poly-logarithmic number of guesses. For a survey on complexity classes with limited nondeterminism, and for several references see [23]. We first show by using a simple but effective technique, which succinctly describes computation paths, that testing non-duality is feasible in polynomial time with $O(\log^3 n)$ nondeterministic steps. We then observe that this approach can be improved to obtain a bound of $O(\chi(n) \cdot \log n)$ nondeterministic steps. *This result is surprising, because most researchers dealing with the complexity of* DUAL *and* TRANS-HYP *believed so far that these problems are completely unrelated to limited nondeterminism.*

We believe that the results presented in this paper are significant, and we are confident that they will be prove useful in various contexts. First, we hope that the various polynomial/output-polynomial cases of the problems which we identify will lead to better and more general methods in various application areas (as we show, e.g. in learning and data mining [12]), and that based on the algorithm DUALIZE or some future modifications, further relevant tractable classes will be identified. Second, we hope that our discovery on limited nondeterminism provides a new momentum to complexity research on DUAL and TRANS-HYP, and will push it towards settling these longstanding open problems.

The rest of this paper is structured as follows. The next section provides some preliminaries and introduces notation. In Section 3, we present our algorithm DUALIZE for dualizing a given monotone prime CNF. After that, we exploit this algorithm in Section 4 to derive a number of polynomial instance classes of the problems DUALIZATION and DUAL. In Section 5 we then show that DUAL can be solved with limited nondeterminism.

2 Preliminaries and Notation

A *Boolean function* (in short, *function*) is a mapping $f : \{0,1\}^n \to \{0,1\}$, where $v \in \{0,1\}^n$ is called a *Boolean vector* (in short, *vector*). As usual, we write $g \leq f$ if f and g satisfy $g(v) \leq f(v)$ for all $v \in \{0,1\}^n$, and $g < f$ if $g \leq f$ and $g \neq f$. A function f is *monotone* (or *positive*), if $v \leq w$ (i.e., $v_i \leq w_i$ for all i) implies $f(v) \leq f(w)$ for all $v, w \in \{0,1\}^n$. Boolean variables $x_1, x_2, \ldots, x_n$ and their complements $\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n$ are called *literals*. A *clause* (resp., *term*) is a disjunction (resp., conjunction) of literals containing at most one of x_i and $\bar{x}_i$ for each variable. A clause c (resp., term t) is an *implicate* (resp., *implicant*) of a function f, if $f \leq c$ (resp., $t \leq f$); moreover, it is *prime*, if there is no implicate $c' < c$ (resp., no implicant $t' > t$) of f, and *monotone*, if it consists of positive literals only. We denote by $PI(f)$ the set of all prime implicants of f.

A *conjunctive normal form* (CNF) (resp., disjunctive normal form, DNF) is a conjunction of clauses (resp., disjunction of terms); it is *prime* (resp. *monotone*), if all its members are prime (resp. *monotone*). For any CNF (resp., DNF) ρ, we denote by $|\rho|$ the number of clauses (resp., terms) in it. Furthermore, for any formula φ, we denote by $V(\varphi)$ the set of variables that occur in φ, and by $\|\varphi\|$ its *length*, i.e., the number of literals in it. We occasionally view CNFs φ also as sets of clauses, and clauses as sets of literals, and use respective notation (e.g., $c \in \varphi$, $\overline{x}_1 \in c$ etc).

As well-known, a function f is monotone iff it has a monotone CNF. Furthermore, all prime implicants and prime implicates of a monotone f are monotone, and it has a unique prime CNF, given by the conjunction of all its prime implicates. For example, the monotone f such that $f(v) = 1$ iff $v \in \{(1100), (1110), (1101), (0111), (1111)\}$ has the unique prime CNF $\varphi = x_2(x_1 \vee x_3)(x_1 \vee x_4)$.

Recall that the *dual* of a function f, denoted f^d, is defined by $f^d(x) = \overline{f}(\overline{x})$, where $\overline{f}$ and $\overline{x}$ is the complement of f and x, respectively. By definition, we have $(f^d)^d = f$. From De Morgan's law, we obtain a formula for f^d from any one of f by exchanging $\vee$ and $\wedge$ as well as the constants 0 and 1. For example, if f is given by $\varphi = x_1x_2 \vee \overline{x}_1(\overline{x}_3 \vee x_4)$, then f^d is represented by $\psi = (x_1 \vee x_2)(\overline{x}_1 \vee \overline{x}_3x_4)$. For a monotone function f, let $\psi = \bigwedge_{c \in C}(\bigvee_{x_i \in c} x_i)$ be the prime CNF of f^d. Then by De Morgan's law, f has the (unique) prime DNF $\rho = \bigvee_{c \in C}(\bigwedge_{x_i \in c} x_i)$; in the previous example, $\rho = x_1x_2 \vee x_2x_3x_4$. Thus, we will regard DUALIZATION also as the problem of computing the prime DNF of f from the prime CNF of f.

3 Ordered Transversal Generation

In what follows, let f be a monotone function and

$$\varphi \quad = \quad \bigwedge_{i=1}^{m} c_i \tag{1}$$

its prime CNF, where we assume without loss of generality that all variables x_j ($j = 1, 2, \ldots n$) appear in φ. Let φ_i ($i = 0, 1, \ldots, n$) be the CNF obtained from φ by fixing variables $x_j = 1$ for all j with $j \geq i+1$. By definition, we have $\varphi_0 = 1$ (truth) and $\varphi_n = \varphi$. For example, consider $\varphi = (x_1 \vee x_2)(x_1 \vee x_3)(x_2 \vee x_3 \vee x_4)(x_1 \vee x_4)$. Then we have $\varphi_0 = \varphi_1 = 1$, $\varphi_2 = (x_1 \vee x_2)$, $\varphi_3 = (x_1 \vee x_2)(x_1 \vee x_3)$, and $\varphi_4 = \varphi$. Similarly, for the prime DNF

$$\psi \quad = \quad \bigvee_{t \in PI(f)} t \tag{2}$$

of f, we denote by ψ_i the DNF obtained from ψ by fixing variables $x_j = 1$ for all j with $j \geq i+1$. Clearly, we have $\varphi_i \equiv \psi_i$, i.e., φ_i and ψ_i represent the same function denoted by f_i.

Proposition 3.1 *Let φ and ψ be any CNF and DNF for f, respectively. Then, for all $i \geq 0$,*

(a) $\|\varphi_i\| \leq \|\varphi\|$ and $|\varphi_i| \leq |\varphi|$, and

(b) $\|\psi_i\| \leq \|\psi\|$ and $|\psi_i| \leq |\psi|$.

Denote by Δ^i ($i = 1, 2, \ldots, n$) the CNF consisting of all the clauses in φ_i but not in φ_{i-1}. For the above example, we have $\Delta^1 = 1$, $\Delta^2 = (x_1 \vee x_2)$, $\Delta^3 = (x_1 \vee x_3)$, and $\Delta^4 = (x_2 \vee x_3 \vee x_4)(x_1 \vee x_4)$. Note that $\varphi_i = \varphi_{i-1} \wedge \Delta^i$; hence, for all $i = 1, 2, \ldots, n$ we have

$$\psi_i \quad \equiv \quad \psi_{i-1} \wedge \Delta^i \;\; \equiv \bigvee_{t \in PI(f_{i-1})} (t \wedge \Delta^i). \tag{3}$$

Let $\Delta^i[t]$, for $i = 1, \ldots, n$ denote the CNF consisting of all the clauses c such that c contains no literal in t_{i-1} and $c \vee x_i$ appears in Δ^i. For example, if $t = x_2x_3x_4$ and $\Delta^4 = (x_2 \vee x_3 \vee x_4)(x_1 \vee x_4)$, then $\Delta^4[t] = x_1$. It follows from (3) that for all $i = 1, 2, \ldots, n$

$$\psi_i \quad \equiv \bigvee_{t \in PI(f_{i-1})} \Big((t \wedge \Delta^i[t]) \;\vee\; (t \wedge x_i) \Big). \tag{4}$$

Lemma 3.2 *For every term $t \in PI(f_{i-1})$, let $g_{i,t}$ denote the function represented by $\Delta^i[t]$. Then $|PI(g_{i,t})| \leq |\psi_i| \leq |\psi|$.*

Proof. Let $V = \{x_1, x_2, \ldots, x_n\}$ and let $s \in PI(g_{i,t})$. Then by (4), $t \wedge s$ is an implicant of ψ_i. Hence, some $t^s \in PI(f_i)$ exists such that $t^s \geq t \wedge s$. Note that $V(t) \cap V(\Delta^i[t]) = \emptyset$, t and $\Delta^i[t]$ have no variable in common, and hence we have $V(s) \subseteq V(t^s)$ ($\subseteq V(s) \cup V(t)$), since otherwise there exists a clause c in $\Delta^i[t]$ such that $V(c) \cap V(t^s) = \emptyset$, a contradiction. Thus $V(t^s) \cap V(\Delta^i[t]) = V(s)$. For any $s' \in PI(g_{i,t})$ such that $s \neq s'$, let $t^s, t^{s'} \in PI(f_i)$ such that $t^s \geq t \wedge s$ and $t^{s'} \geq t \wedge s'$, respectively. By the above discussion, we have $t^s \neq t^{s'}$. This completes the proof. $\Box$

We now describe our algorithm DUALIZE for generating $PI(f)$. It is inspired by a similar graph algorithm of Johnson, Yannakakis, and Papadimitriou [31], and can be regarded as a generalization.

Algorithm DUALIZE

Input: The prime CNF φ of a monotone function f.
Output: The prime DNF ψ of f, i.e. all prime implicants of f.

Step 1: Compute the smallest prime implicant t_{min} of f and set $Q := \{ t_{min} \}$;

Step 2: **while** $Q \neq \emptyset$ **do begin**
Remove the smallest t from Q and output t;
for each i with $x_i \in V(t)$ and $\Delta^i[t] \neq 1$ **do begin**
Compute the prime DNF $\rho_{(t,i)}$ of the function represented by $\Delta^i[t]$;
for each term t' in $\rho_{(t,i)}$ **do begin**
if $t_{i-1} \wedge t'$ is a prime implicant of f_i **then begin**
Compute the smallest prime implicant t^* of f such that $t^*_i = t_{i-1} \wedge t'$;
$Q := Q \cup \{t^*\}$
end{if} **end**{for} **end**{for}
end{while}

Here, we say that term s is *smaller* than term t if $\sum_{x_j \in V(s)} 2^{n-j} < \sum_{x_j \in V(t)} 2^{n-j}$; i.e., as vector, s is lexicographically smaller than t.

Theorem 3.3 *Algorithm* DUALIZE *correctly outputs all* $t \in PI(f)$ *in increasing order.*

Proof. (Sketch) First note that the term t^* inserted in Q when t is output is larger than t. Indeed, t' ($\neq 1$) and t_{i-1} are disjoint and $V(t') \subseteq \{x_1, \ldots, x_{i-1}\}$. Hence, every term in Q is larger than all terms already output, and the output sequence is increasing. We show by induction that, if t is the smallest prime implicant of f that was not output yet, then t is already in Q. This clearly proves the result.

Clearly, the above statement is true if $t = t_{min}$. Assume now that $t \neq t_{min}$ is the smallest among the prime implicants not output yet. Let i be the largest index such that t_i is not a prime implicant of f_i. This i is well-defined, since otherwise $t = t_{min}$ must hold, a contradiction. Now we have (1) $i < n$ and (2) $i + 1 \notin V(t)$, where (1) holds because t_n $(= t)$ is a prime implicant of f_n $(= f)$ and (2) follows from the maximality of i. Let $s \in PI(f_i)$ such that $V(s) \subseteq V(t_i)$, and let $K = V(t_i) - V(s)$. Then $K \neq \emptyset$ holds, and since $x_{i+1} \notin V(t)$, the term $t' = \bigwedge_{x_j \in K} x_j$ is a prime implicant of $\Delta^{i+1}[s]$. There exists $s' \in PI(f)$ such that $s'_i = s$ and $x_{i+1} \in V(s')$, since $s \wedge x_{i+1} \in PI(f_{i+1})$. Note that $\Delta^{i+1}[s] \neq 0$. Moreover, since s' is smaller than t, by induction s' has already been output. Therefore, $t' = \bigwedge_{x_j \in K} x_j$ has been considered in the inner for-loop of the algorithm. Since $s'_i \wedge t'$ $(= t_i = t_{i+1})$ is a prime implicant of f_{i+1}, the algorithm has added the smallest prime implicant t^* of f such that $t^*_{i+1} = t_{i+1}$. We finally claim that $t^* = t$. Otherwise, let k be the first index in which t^* and t differ. Then $k > i + 1$, $x_k \in V(t)$ and $x_k \notin V(t^*)$. However, this implies $t_k \notin PI(f_k)$, contradicting the maximality of i. $\Box$

Remark 3.1 (1) The decomposition rule (4) was already used in [33].
(2) In step 1, we could generate any prime implicant t of f, and choose then a lexicographic term ordering inherited from a dynamically generated variable ordering. In step 2, it is sufficient that any monotone DNF $\tau_{(t,i)}$ of the function represented by $\Delta^i[t]$ is computed, rather than its prime DNF $\rho_{(t,i)}$. This might make the algorithm faster.

Let us consider the time complexity of algorithm DUALIZE. We store Q as a binary tree, where each leaf represents a term t and the left (resp., right) son of a node at depth $j - 1 \geq 0$, where the root has depth 0, encodes $x_j \in V(t)$ (resp., $x_j \notin V(t)$). In Step 1, we can compute t_{min} in $O(\|\varphi\|)$ time and initialize Q in $O(n)$ time.

As for Step 2, let $T_{(t,i)}$ be the time required to compute the prime DNF $\rho_{(t,i)}$ from $\Delta^i[t]$. By analyzing its substeps, we can see that each iteration of Step 2 requires $\sum_{x_i \in V(t)}(T_{(t,i)} + |\rho_{(t,i)}| \cdot O(\|\varphi\|))$ time.

Indeed, we can update Q (i.e., remove the smallest term and add t^*) in $O(n)$ time. For each t and i, we can construct $\Delta^i[t]$ in $O(\|\varphi\|)$ time. Moreover, we can check whether $t_{i-1} \wedge t'$ is a prime implicant of f_i and if so, we can compute the smallest prime implicant t^* of f such that $t^*_i = t_{i-1} \wedge t'$ in $O(\|\varphi\|)$ time; note that t^* is the smallest prime implicant of the function obtained from f by fixing $x_j = 1$ if $x_j \in V(t_i \wedge t')$ and 0 if $x_j \notin V(t_i \wedge t')$ for $j \leq i$.

Hence, we have the following result.

Theorem 3.4 *The output delay of Algorithm* DUALIZE *is bounded by*

$$\max_{t \in PI(f)} \Big(\sum_{x_i \in V(t)} (T_{(t,i)} + |\rho_{(t,i)}| \cdot O(\|\varphi\|)) \Big) \tag{5}$$

time, and DUALIZE *needs in total time*

$$\sum_{t \in PI(f)} \sum_{x_i \in V(t)} (T_{(t,i)} + |\rho_{(t,i)}| \cdot O(\|\varphi\|)). \tag{6}$$

If the $T_{(t,i)}$ are bounded by a polynomial in the input length, then DUALIZE becomes a polynomial delay algorithm, since $|\rho_{(t,i)}| \leq T_{(t,i)}$ holds for all $t \in PI(f)$ and $x_i \in V(t)$. On the other hand, if they are bounded by a polynomial in the combined input and output length, then DUALIZE is a polynomial total time algorithm, where $|\rho_{(t,i)}| \leq |\psi|$ holds from Lemma 3.2. Using results from [3], we can construct from DUALIZE an incremental polynomial time algorithm for DUALIZATION, which however might not output $PI(f)$ in increasing order. Summarizing, we have the following corollary.

Corollary 3.5 *Let* $T = \max\{ T_{(t,i)} \mid t \in PI(f), x_i \in V(t) \}$. *Then, if* T *is bounded by a*

(i) polynomial in n *and* $\|\varphi\|$, *then* DUALIZE *is an* $O(n\|\varphi\|T)$ *polynomial delay algorithm;*

(ii) polynomial in n, $\|\varphi\|$, *and* $\|\psi\|$, *then* DUALIZE *is an* $O(n \cdot |\psi| \cdot (T + |\psi| \cdot \|\varphi\|))$ *polynomial total-time algorithm; moreover,* DUALIZATION *is solvable in incremental polynomial time.*

In the next section, we identify sufficient conditions for the boundedness of T and fruitfully apply them to solve open problems and improve previous results.

4 Polynomial Classes

4.1 Degenerate CNFs

We first consider the case of small $\Delta^i[t]$. Generalizing a notion for graphs (i.e., monotone 2-CNFs) [50], we call a monotone CNF φ *k-degenerate*, if there exists a variable ordering $x_1, \ldots, x_n$ in which

$|\Delta^i| \leq k$ for all $i = 1, 2, \ldots, n$. We call a variable ordering $x_1, \ldots, x_n$ *smallest last* as in [50], if x_i is chosen in the order $i = n, n-1, \ldots, 1$ such that $|\Delta^i|$ is smallest for all variables that were not chosen. Clearly, a smallest last ordering gives the least k such that φ is k-degenerate. Therefore, we can check for every integer $k \geq 1$ whether φ is k-degenerate in $O(\|\varphi\|)$ time. If this holds, then we have $|\rho_{(t,i)}| \leq n^k$ and $T_{(t,i)} = O(kn^{k+1})$ for every $t \in PI(f)$ and $i \in V(t)$ (for $T_{(t,i)}$, apply the distributive law to $\Delta^i[t]$ and remove terms t where some $x_j \in V(t)$ has no $c \in \Delta^i[t]$ such that $V(t) \cap V(c) = \{x_j\}$). Thus Theorem 3.4 implies the following.

Theorem 4.1 *For k-degenerate CNFs φ,* DUALIZATION *is solvable with $O(\|\varphi\| \cdot n^{k+1})$ polynomial delay if $k \geq 1$ is constant.*

Applying the result of [37] that log-clause CNF is dualizable in incremental polynomial time, we obtain a polynomiality result also for non-constant degeneracy:

Theorem 4.2 *For $O(\log \|\varphi\|)$-degenerate CNFs φ, problem* DUALIZATION *is solvable in polynomial total time.*

In the following, we discuss several natural subclasses of degenerate CNFs.

4.1.1 Read-bounded CNFs

A monotone CNF φ is called *read-k*, if each variable appears in φ at most k times. Clearly, read-k CNFs are k-degenerate, and in fact φ is read-k iff it is k-degenerate under every variable ordering. By applying Theorems 4.1 and 4.2, we obtain the following result.

Corollary 4.3 *For read-k CNFs φ, problem* DUALIZATION *is solvable*

(i) with $O(\|\varphi\| \cdot n^{k+1})$ polynomial delay, if k is constant;

(ii) in polynomial total time, if $k = O(\log(\|\varphi\|))$.

Note that Corollary 4.3 (i) trivially implies that DUALIZATION is solvable in $O(|\psi| \cdot n^{k+2})$ time for constant k, since $\|\varphi\| \leq kn$. This improves upon the previous best known algorithm [12], which is only $O(|\psi| \cdot n^{k+3})$ time, not polynomial delay, and outputs $PI(f)$ in no specific order. Corollary 4.3 (ii) is a non-trivial generalization of the result in [12], which was posed as an open problem [11].

4.1.2 Acyclic CNFs

Like in graphs, acyclicity is appealing in hypergraphs resp. monotone CNFs from a theoretical as well as a practical point of view. However, there are many notions of acyclicity for hypergraphs (cf. [16]), since different generalizations from graphs are possible. We refer to α-, β-,γ-, and *Berge*-acyclicity as stated in [16], for which the following proper inclusion hierarchy is known:

$$\text{Berge-acyclic} \subseteq \gamma\text{-acyclic} \subseteq \beta\text{-acyclic} \subseteq \alpha\text{-acyclic}.$$

The notion of α-acyclicity came up in relational database theory. A monotone CNF φ is *α-acyclic* iff $\varphi = 1$ or reducible by the GYO-reduction [25, 51], i.e., repeated application of one of the two rules:

(1) If variable x_i occurs in only one clause c, remove x_i from c.
(2) If distinct clauses c and c' satisfy $V(c) \subseteq V(c')$, remove c from φ.

to 0 (i.e., the empty clause). Note that α-acyclicity of a monotone CNF φ can be checked, and a suitable GYO-reduction output, in $O(\|\varphi\|)$ time [48]. A monotone CNF φ is *β-acyclic* iff every CNF consisting of clauses in φ is α-acyclic. As shown in [15], the prime implicants of a monotone f represented by a β-acyclic CNF φ can be enumerated (and thus DUALIZATION solved) in $p(\|\varphi\|) \cdot |\psi|$ time, where p is a polynomial in $\|\varphi\|$. However, the time complexity of DUALIZATION for the more general α-acyclic prime CNFs was left as an open problem. We now show that it is solvable with polynomial delay, by showing that α-acyclic CNFs are 1-degenerate.

Let $\varphi \neq 1$ be a prime CNF. Let $a = a_1, a_2, \dots, a_q$ be a GYO-reduction for φ, where $a_\ell = x_i$ if the ℓ-th operation removes x_i from c, and $a_\ell = c$ if it removes c from φ. Consider the unique variable ordering $b_1, b_2, \dots, b_n$ such b_i occurs after b_j in a, for all $i < j$. For example, let $\varphi = c_1c_2c_3c_4$, where $c_1 = (x_1 \vee x_2 \vee x_3)$, $c_2 = (x_1 \vee x_3 \vee x_5)$, $c_3 = (x_1 \vee x_5 \vee x_6)$ and $c_4 = (x_3 \vee x_4 \vee x_5)$. Then φ is α-acyclic, since it has the GYO-reduction

$$a_1 = x_2,\ a_2 = c_1,\ a_3 = x_4,\ a_4 = x_6,\ a_5 = c_4,\ a_6 = c_3,\ a_7 = x_1,\ a_8 = x_3,\ a_9 = x_5.$$

From this sequence, we obtain the variable ordering

$$b_1 = x_5,\ b_2 = x_3,\ b_3 = x_1,\ b_4 = x_6,\ b_5 = x_4,\ b_6 = x_2.$$

As easily checked, this ordering shows that φ is 1-degenerate. Under this ordering, we have $\Delta^1 = \Delta^2 = 1$, $\Delta^3 = (x_1 \vee x_3 \vee x_5)$, $\Delta^4 = (x_1 \vee x_5 \vee x_6)$, $\Delta^5 = (x_3 \vee x_4 \vee x_5)$, and $\Delta^6 = (x_1 \vee x_2 \vee x_3)$. This is not accidental.

Lemma 4.4 *Every α-acyclic prime CNF is 1-degenerate.*

Note that the converse is not true, i.e., there exists a 1-degenerate CNF that is not α-acyclic. For example, $\varphi = (x_1 \vee x_2 \vee x_3)(x_1 \vee x_2 \vee x_4)(x_2 \vee x_3 \vee x_4 \vee x_5)$ is such a CNF. Lemma 4.4 and Theorem 4.1 imply the following result.

Corollary 4.5 *For α-acyclic CNFs φ, problem* DUALIZATION *is solvable with $O(\|\varphi\| \cdot n^2)$ delay.*

Observe that for a prime α-acyclic φ, we have $|\varphi| \leq n$. Thus, if we slightly modify algorithm DUALIZE to check $\Delta^i = 1$ in advance (which can be done in linear time in a preprocessing phase) such that such Δ^i need not be considered in step 2, then the resulting algorithm has $O(n \cdot |\varphi| \cdot \|\varphi\|)$ delay. Observe that the algorithm in [15] solves, minorly adapted for enumerative output, DUALIZATION for β-acyclic CNFs with $O(n \cdot |\varphi| \cdot \|\varphi\|)$ delay. Thus, the above modification of DUALIZE is of the same order.

4.1.3 CNFs with bounded treewidth

A *tree decomposition (of type I)* of a monotone CNF φ is a tree $T = (W, E)$ where each node $w \in W$ is labeled with a set $X(w) \subseteq V(\varphi)$ under the following conditions:

1. $\bigcup_{w \in W} X(w) = V(\varphi)$;
2. for every clause c in φ, there exists some $w \in W$ such that $V(c) \subseteq X(w)$; and
3. for any variable $x_i \in V$, the set of nodes $\{w \in W \mid x_i \in X(w)\}$ induces a (connected) subtree of T.

The *width* of T is $\max_{w \in W} |X(w)| - 1$, and the *treewidth* of φ, denoted by $Tw_1(\varphi)$, is the minimum width over all its tree decompositions.

Note that the usual definition of treewidth for a graph [45] results in the case where φ is a 2-CNF. Similarly to acyclicity, there are several notions of treewidth for hypergraphs resp. monotone CNFs. For example, tree decomposition of type II of CNF $\varphi = \bigwedge_{c \in C} c$ is defined as type-I tree decomposition of its incident 2-CNF (i.e., graph) $G(\varphi)$ [13, 24]. That is, for each clause $c \in \varphi$, we introduce a new variable y_c and construct $G(\varphi) = \bigwedge_{x_i \in c \in \varphi}(x_i \vee y_c)$ (here, $x_i \in c$ denotes that x_i appears in c). Let $Tw_2(\varphi)$ denote the type-II treewidth of φ.

Proposition 4.6 *For every monotone CNF φ, it holds that $Tw_2(\varphi) \leq Tw_1(\varphi) + 2^{Tw_1(\varphi)+1}$.*

Proof. Let $T = (W, E)$, $X : W \to 2^V$ be any tree decomposition of φ having width $Tw_1(\varphi)$. Introduce for all $c \in \varphi$ new variables y_c, and add y_c to every $X(w)$ such that $V(c) \subseteq X(w)$. Clearly, the result is a type-I tree decomposition of $G(\varphi)$, and thus a type-II tree decomposition of φ. Since at most $2^{|X(w)|}$ many y_c are added to $X(w)$ and $|X(w)| - 1 \leq Tw_1(\varphi)$ for every $w \in W$, the result follows. □

This means that if $Tw_1(\varphi)$ is bounded by some constant, then so is $Tw_2(\varphi)$. Moreover, $Tw_1(\varphi) = k$ implies that φ is a k-CNF; we discuss k-CNFs in Section 4.2 and only consider $Tw_2(\varphi)$ here. The following proposition states some relationships between type-II treewidth and other restrictions of CNFs from above.

Proposition 4.7 *The following properties hold for type-II treewidth.*

(i) There is a family of monotone prime CNFs φ such that $Tw_2(\varphi)$ is bounded by a constant, but φ is not k-CNF for any constant k.

(ii) There is a family of monotone prime CNFs φ such that $Tw_2(\varphi)$ is bounded by a constant, but φ does not have bounded read.

(iii) There is a family of α-acyclic prime CNFs φ such that $Tw_2(\varphi)$ is not bounded by any constant. (This is a contrast to the graph case that a graph is acyclic if and only if its treewidth is 1.)

Proof. (i): For example, $\varphi = (\bigvee_{x_i \in V} x_i)$ has $Tw_2(\varphi) = 1$, since it has a tree decomposition $T = (W, E)$ with $X : W \to 2^V$ defined by $W = \{1, 2, \ldots, n\}$, $E = \{(w, w+1), w = 1, 2, \ldots, n-1\}$, and

$X(w) = \{x_w, y_c\}$, $w \in W$, where $c = (\bigvee_{x_i \in V} x_i)$. However, it is not an $(n-1)$-CNF (but an n-CNF). On the other hand, by Lemma 4.8, we can see that there is a family of monotone prime CNFs φ such that $Tw_2(\varphi)$ is not bounded by any constant, but φ is k-CNF for some constant k.

(ii): For example, let φ be a CNF containing $n-1$ clauses $c_i = (x_1 \vee x_i)$, $i = 2, 3, \ldots, n$. Then φ has $Tw_2(\varphi) = 1$, since it has a tree decomposition $T = (W, E)$ with $X : W \to 2^V$ defined by $W = \{(c_i, x_1), (c_i, x_i), i = 2, 3, \ldots, n\}$, $E = \{((c_i, x_1), (c_{i+1}, x_1)), i = 2, 3, \ldots, n-1\} \cup \{((c_i, x_1), (c_i, x_i)), i = 2, 3, \ldots, n$, and $X((c_i, x_k)) = \{y_{c_i}, x_k\}$, $(c_i, x_k) \in W$. However, it is not read-$(n-2)$ (but read-$(n-1)$).

(iii): For example, let φ be a CNF on $V = \{x_1, x_2, \ldots, x_{2n}\}$ containing n clauses $c_i = (x_i \vee \bigvee_{j \geq n+1} x_j)$, for $i = 1, \ldots, n$. Then φ is α-acyclic. We claim that $Tw_2(\varphi) \geq n-1$. Let us assume that there exists a tree $T = (W, E)$ with $X : E \to 2^V$ that shows $Tw_2(\varphi) \leq n-2$, where T is regarded as a rooted tree. Let $T_i = (W_i, E_i)$ be the subtree of T induced by $W_i = \{w \in W \mid y_{c_i} \in X(w)\}$, and let r_i be its root. Consider the case in which W_i and W_j are disjoint for some i and j. Suppose that r_j is an ancestor of r_i. Since $|X(r_i)| \leq Tw_2(\varphi) + 1 \leq n-1$, there exists a node $x_{n+k} \in V$ such that $1 \leq k \leq n$ and $x_{n+k} \notin X(r_i)$. However, since the incident graph of φ contains two edges (x_{n+k}, y_{c_i}) and (x_{n+k}, y_{c_j}), we have $x_{n+k} \in \bigcup_{w \in W_i - \{r_i\}} X(w)$ and $x_{n+k} \in \bigcup_{w \in W_j} X(w)$. This is a contradiction to the condition that $\{w \in W \mid x_{n+k} \in X(w)\}$ is connected. Similarly, we can prove our claim when T_i and T_j are disjoint, but r_j is not an ancestor of r_i.

We thus consider the case in which $W_i \cap W_j \neq \emptyset$ holds for any i and j. Since T_i's are trees, the family of W_i, $i = 1, 2, \ldots, n$, satisfies the well-known Helly property, i.e., there exists a node w in $\bigcap_{i=1}^n W_i$. $X(w)$ must contain all y_{c_i}'s. This implies $|X(w)| \geq n$, a contradiction. □

As we show now, bounded-treewidth implies bounded degeneracy.

Lemma 4.8 *Let φ be any monotone CNF with $Tw_2(\varphi) = k$. Then φ is 2^k-degenerate.*

Proof. Let $T = (W, E)$ with $X : W \to 2^V$ show $Tw_2(\varphi) = k$. From this, we reversely construct a variable ordering $a = a_1, \ldots, a_n$ on $V = V(\varphi)$ such that $|\Delta^i| \leq 2^k$ for all i.

Set $i := n$. Choose any leaf w^* of T, and let $p(w^*)$ be a node in W adjacent to w^*. If $X(w^*) \setminus X(p(w^*)) \subseteq \{y_c \mid c \in \varphi\}$, then remove w^* from T. On the other hand, if $(X(w^*) \setminus X(p(w^*))) \cap V = \{x_{j_1}, \ldots, x_{j_\ell}\}$ where $\ell \geq 1$ (in this case, only $X(w^*)$ contains $x_{j_1}, \ldots, x_{j_\ell}$), then define $a_{i+1-h} = x_{j_h}$ for $h = 1, \ldots, \ell$ and update $i := n - \ell$, $X(w^*) := X(w^*) \setminus \{x_{j_1}, \ldots, x_{j_\ell}\}$, and $X(w) := X(w) \setminus \{y_c \mid c \in \varphi, V(c) \cap \{x_{j_1}, \ldots, x_{j_\ell}\} \neq \emptyset\}$ for every $w \in W$. Let a be completed by repeating this process.

We claim that a shows that $|\Delta^i| \leq 2^k$ for all $i = 1, \ldots, n$. To see this, let w^* be chosen during this process, and assume that $a_i \in X(w^*) \setminus X(p(w^*))$. Then, by induction on the (reverse) construction of a, we obtain that for each clause $c \in \Delta^i$ we must have either (a) $y_c \in X(w^*)$ or (b) $V(c) \subseteq X(w^*)$. The latter case may arise if in previous steps of the process some descendant $d(w^*)$ of w^* was removed which contains y_c such that y_c does not occur in w^*; however, in this case $V(c) \subseteq X(w)$ must be true on every node on the path from $d(w^*)$ to w^*.

Now let $q = |X(w^*) \setminus V|$. Since $|X(w^*) \setminus \{a_i\}| \leq k$, we have

$$|\Delta^i| \ \leq \ q + 2^{k-q} \ \leq \ 2^k.$$

This proves the claim. □

Corollary 4.9 *For CNFs φ with $Tw_2(\varphi) \leq k$,* DUALIZATION *is solvable (i) with $O(\|\varphi\| \cdot n^{2^k+1})$ polynomial delay, if k is constant; and (ii) in polynomial total time, if $k = O(\log\log\|\varphi\|)$.*

4.2 Recursive application of algorithm DUALIZE

Algorithm DUALIZE computes in step 2 the prime DNF $\rho_{(t,i)}$ of the function represented by $\Delta^i[t]$. Since $\Delta[t]$ is the prime CNF of some monotone function, we can recursively apply DUALIZE to $\Delta^i[t]$ for computing $\rho_{(t,i)}$. Let us call this variant R-DUALIZE. Then we have the following result.

Theorem 4.10 *If its recursion depth is d,* R-DUALIZE *solves* DUALIZATION *in $O(n^{d-1} \cdot |\psi|^{d-1} \cdot \|\varphi\|)$ time.*

Proof. If $d = 1$, then $\Delta^i[t_{min}] = 1$ holds for t_{min} and every $i \geq 1$. This means that $PI(f) = \{t_{min}\}$ and φ is a 1-CNF (i.e., each clause in φ contains exactly one variable). Thus in this case, R-DUALIZE needs $O(n)$ time. Recall that algorithm DUALIZE needs, by (6), time $\sum_{t\in PI(f)} \sum_{x_i\in V(t)} (T_{(t,i)}+|\rho_{(t,i)}| \cdot O(\|\varphi\|))$. If $d = 2$, then $T_{(t,i)} = O(n)$ and $|\rho_{(t,i)}| \leq 1$. Therefore, R-DUALIZE needs time $O(n \cdot |\psi| \cdot \|\varphi\|)$. For $d \geq 3$, Corollary 3.5.(ii) implies that R-DUALIZE needs $O(n^{d-1} \cdot |\psi|^{d-1} \cdot \|\varphi\|)$ time. □

Recall that a CNF φ is called *k-CNF* if each clause in φ has at most k literals. Clearly, if we apply algorithm R-DUALIZE to a monotone k-CNF φ, the recursion depth of R-DUALIZE is at most k. Thus we obtain the following result; it re-establishes, with different means, the main positive result of [8, 15].

Corollary 4.11 R-DUALIZE *solves* DUALIZATION *in $O(n^{k-1} \cdot |\psi|^{k-1} \cdot \|\varphi\|)$ time, i.e., in polynomial total time for monotone k-CNFs φ where k is constant.*

5 Limited Nondeterminism

In the previous section, we have discussed polynomial cases of monotone dualization. In this section, we now turn to the issue of the precise complexity of this problem. For this purpose, we consider the decision problem DUAL, i.e., decide whether given monotone prime CNFs φ and ψ represent dual Boolean functions, instead of the search problem DUALIZATION.

It appears that problem DUAL can be solved with limited nondeterminism, i.e., with poly-log many guessed bits by a polynomial-time non-deterministic Turing machine. This result might bring new insight towards settling the complexity of the problem.

We adopt Kintala and Fischer's terminology [32] and write $g(n)$-P for the class of sets accepted by a nondeterministic Turing machine in polynomial time making at most $g(n)$ nondeterministic steps on every input of length n. For every integer $k \geq 1$, define $\beta_k\mathrm{P} = \bigcup_c (c\log^k n)$-P. The βP *Hierarchy* consists of the classes

$$\mathrm{P} = \beta_1\mathrm{P} \subseteq \beta_2\mathrm{P} \subseteq \cdots \subseteq \bigcup_k \beta_k\mathrm{P} = \beta\mathrm{P}$$

and lies between P and NP. The $\beta_k\mathrm{P}$ classes appear to be rather robust; they are closed under polynomial time and logspace many-one reductions and have complete problems (cf. [23]). The complement class of $\beta_k\mathrm{P}$ is denoted by co-$\beta_k\mathrm{P}$.

We start in Section 5.1 by recalling algorithm A of [17], reformulated for CNFs and by analyzing A's behavior. The proof that A can be converted to an algorithm that uses $\log^3 n$ nondeterministic bit guesses, and that DUAL is thus in co-$\beta_3\mathrm{P}$, is rather easy and should give the reader an intuition of how our new method of analysis works. In Section 5.2, we use basically the same technique for analyzing the more involved algorithm B of [17]. Using a modification of this algorithm, we show that DUAL is in co-$\beta_2\mathrm{P}$. We also prove the stronger result that the complement of DUAL can be solved in polynomial time with only $O(\chi(n) \cdot \log(n))$ nondeterministic steps (=bit guesses). Finally, Section 5.3 shows that membership in co-$\beta_2\mathrm{P}$ can alternatively be obtained by combining the results of [17] with a theorem of Beigel and Fu [2].

5.1 Analysis of Algorithm A of Fredman and Khachiyan

The first algorithm in [17] for recognizing dual monotone pairs is as follows.

Algorithm A (reformulated for CNFs[1]).

Input: Monotone CNFs φ, ψ representing monotone f, g s.t. $V(c) \cap V(c') \neq \emptyset$, for all $c \in \varphi$, $c' \in \psi$.
Output: `yes` if $f = g^d$, otherwise a vector w of form $w = (w_1, \ldots, w_m)$ such that $f(w) \neq g^d(w)$.

Step 1:

Delete all redundant (i.e., non-minimal) clauses from φ and ψ.

Step 2:

Check that (1) $V(\phi) = V(\psi)$, (2) $\max_{c \in \varphi} |c| \leq |\psi|$, (3) $\max_{c' \in \psi} |c'| \leq |\varphi|$, and (4) $\Sigma_{c \in \varphi} 2^{-|c|} + \Sigma_{c' \in \psi} 2^{-|c'|} \geq 1$.

If any of conditions (1)-(4) fails, $f \neq g^d$ and a witness w is found in polynomial time (cf. [17]).

Step 3:

If $|\varphi| \cdot |\psi| \leq 1$, test duality in $O(1)$ time.

Step 4:

If $|\varphi| \cdot |\psi| \geq 2$, find some x_i occurring in φ or ψ (w.l.o.g. in φ) with frequency $\geq 1/\log(|\varphi| + |\psi|)$. Let

$$\varphi_0 = \{c - \{x_i\} \mid x_i \in c,\, c \in \varphi\}, \quad \varphi_1 = \{c \mid x_i \notin c,\, c \in \varphi\},$$
$$\psi_0 = \{c' - \{x_i\} \mid x_i \in c',\, c' \in \psi\}, \quad \psi_1 = \{c' \mid x_i \notin c',\, c' \in \psi\}.$$

Call algorithm A on the two pairs of forms:

$$\text{(A.1)} \ (\varphi_1, \psi_0 \wedge \psi_1) \quad \text{and} \quad \text{(A.2)} \ (\psi_1, \varphi_0 \wedge \varphi_1)$$

If both calls return `yes`, then return `yes` (as $f = g^d$), otherwise we obtain w such that $f(w) \neq g^d(w)$ in polynomial time (cf. [17]).

---

[1] In [17], duality is tested for DNFs while our problem DUAL speaks about CNFs; this is insignificant, since DNFs are trivially translated to CNFs for this task and vice versa (cf. Section 2).

We observe that, as noted in [17], the binary length of any standard encoding of the input φ, ψ to algorithm A is polynomially related to $|\varphi| + |\psi|$, if step 3 is reached. Thus, for our purpose, we consider $|\varphi| + |\psi|$ to be the input size.

Let φ^*, ψ^* be the original input for A. For any pair (φ, ψ) of CNFs, define its *volume* by $v = |\varphi| \cdot |\psi|$, and let $\epsilon = 1/\log n$, where $n = |\varphi^*| + |\psi^*|$. As shown in [17], step 4 of algorithm A divides the current (sub)problem of volume $v = |\varphi| \cdot |\psi|$ by self-reduction into subproblems (A.1) and (A.2) of respective volumes (assuming that x_i frequently occurs in φ):

$$|\varphi_1| \cdot |\psi_0 \wedge \psi_1| \quad \leq \quad (1 - \epsilon) \cdot v \tag{7}$$

$$|\varphi_0 \wedge \varphi_1| \cdot |\psi_1| \;\leq\; |\varphi| \cdot (|\psi| - 1) \quad \leq \quad v - 1 \tag{8}$$

Let $T = T(\varphi, \psi)$ be the recursion tree generated by A on input (φ, ψ). In T, each node u is labeled with the respective monotone pair, denoted by $I(u)$; thus, if r is the root of T, then $I(r) = (\varphi, \psi)$. The *volume* $v(u)$ of node u is defined as the volume of its label $I(u)$.

Any node u is a leaf of T, if algorithm A stops on input $I(u) = (\varphi, \psi)$ during steps 1-3; otherwise, u has a left child u_l and a right child u_r corresponding to (A.1) and (A.2), i.e., labeled $(\varphi_1, \psi_0 \wedge \psi_1)$ and $(\psi_1, \varphi_0 \wedge \varphi_1)$ respectively. That is, u_l is the "high frequency move" by the splitting variable.

We observe that every node u in T is determined by a *unique path* from the root to u in T and thus by a unique sequence $seq(u)$ of right and left moves starting from the root of T and ending at u. The following key lemma bounds the number of moves of each type for certain inputs.

Lemma 5.1 *Suppose $|\varphi^*| + |\psi^*| \leq |\varphi^*| \cdot |\psi^*|$. Then for any node a in T, $seq(a)$ contains at most v^* right moves and at most $\log^2 v^*$ left moves, where $v^* = |\varphi^*| \cdot |\psi^*|$.*

Proof. By (7) and (8), each move decreases the volume of a node label. Thus, the length of $seq(u)$, and in particular the number of right moves, is bounded by v^*. To obtain the better bound for the left moves, we will use the following well-known inequality:

$$(1 - 1/y)^y \quad \leq \quad 1/e, \qquad \text{for } y \geq 1. \tag{9}$$

In fact, the sequence $(1 - 1/y_i)^{y_i}$, for any $1 \leq y_1 < y_2 < \ldots$ monotonically converges to $1/e$ from below. By (7), the volume $v(u)$ of any node u such that $seq(u)$ contains $\log^2 v^*$ left moves is bounded as follows:

$$v(u) \leq v^* \cdot (1 - \epsilon)^{\log^2 v^*} = v^* \cdot (1 - 1/\log n)^{\log^2 v^*}.$$

Since $n = |\varphi^*| + |\psi^*| \leq |\varphi^*| \cdot |\psi^*| = v^*$, and because of (9) it follows that:

$$\begin{aligned} v(u) \quad &\leq \quad v^* \cdot \left((1 - 1/\log v^*)^{\log v^*}\right)^{\log v^*} \\ &\leq \quad v^* \cdot (1/e)^{\log v^*} \;=\; v^*/(e^{\log v^*}) \;<\; v^*/(2^{\log v^*}) \;=\; 1. \end{aligned}$$

Thus, u must be a leaf in T. Hence for every u in T, $seq(u)$ contains at most $\log^2 v^*$ left moves. □

Theorem 5.2 *Problem* DUAL *is in* co-β_3P.

Proof. Instances such that either $c \cap c' = \emptyset$ for some $c \in \varphi^*$ and $c' \in \psi^*$, the sequence $seq(u)$ is empty, or $|\varphi^*| + |\psi^*| > |\varphi^*| \cdot |\psi^*|$ are easily recognized and solved in deterministic polynomial time. In the remaining cases, if $f \neq g^d$, then there exists a leaf u in T labeled by a non-dual pair (φ', ψ'). If $seq(u)$ is known, we can compute, by simulating A on the branch described by $seq(u)$, the entire path $u_0, u_1, \ldots, u_l = u$ from the root u_0 to u with all labels $I(u_0) = (\varphi^*, \psi^*)$, $I(u_1)$, …, $I(u_l)$ and check that $I(u_l)$ is non-dual in steps 2 and 3 of A in polynomial time. Since the binary length of any standard encoding of (φ^*, ψ^*) is polynomially related to $n = |\varphi^*| + |\psi^*|$ if $seq(u)$ is nonempty, to prove the result it is sufficient to show that $seq(u)$ can be constructed in polynomial time from $O(\log^3 v^*)$ suitably guessed bits. To see this, let us represent every $seq(u)$ as a sequence $seq^*(u) = [\ell_0, \ell_1, \ell_2 \ldots, \ell_k]$, where ℓ_0 is the number of leading right moves and ℓ_i is the number of consecutive right moves after the i-th left move in $seq(u)$, for $i = 1, \ldots, k$. For example, if $seq(u) = [\mathtt{r}, \mathtt{r}, \mathtt{l}, \mathtt{r}, \mathtt{r}, \mathtt{r}, \mathtt{l}]$, then $seq^*(u) = [2, 3, 0]$. By Lemma 5.1, $seq^*(u)$ has length at most $\log^2 v^* + 1$. Thus, $seq^*(u)$ occupies in binary only $O(\log^3 v)$ bits; moreover, $seq(u)$ is trivially computed from $seq^*(u)$ in polynomial time. □

5.2 Analysis of Algorithm B of Fredman and Khachiyan

The aim of the above proof was to exhibit a new method of algorithm analysis that allows us to show with very simple means that duality can be polynomially checked with limited nondeterminism. By applying the same method of analysis to the slightly more involved algorithm B of [17] (which runs in $n^{4\chi(n)+O(1)}$ time, and thus in $n^{o(\log n)}$ time), we can sharpen the above result by proving that deciding whether monotone CNFs φ and ψ are non-dual is feasible in polynomial time with $O(\chi(n) \cdot \log n)$ nondeterministic steps; consequently, the problem DUAL is in co-β_2P.

Like algorithm A, also algorithm B uses a recursive self-reduction method that decomposes its input, a pair (φ, ψ) of monotone CNFs, into smaller inputs instances for recursive calls. Analogously, the algorithm is thus best described via its *recursion tree* T, whose root represents the input instance (φ^*, ψ^*) (of size n), whose intermediate nodes represent smaller instances, and whose leaves represent those instances that can be solved in polynomial time. Like for algorithm A, the nodes u in T are labeled with the respective instances $I(u) = (\varphi, \psi)$ of monotone pairs. Whenever there is a branching from a node u to children, then $I(u)$ is a pair of dual monotone CNFs iff $I(u')$ for *each* child u' of u in T is a pair of dual monotone CNFs. Therefore, the original input (φ^*, ψ^*) is a dual monotone pair iff all leaves of T are labeled with dual monotone pairs.

Rather than describing algorithm B in full detail, we confine here to recall those features which are relevant for our analysis. In particular, we will describe some essential features of its recursion tree T.

For each variable x_i occurring in φ, the *frequency* ϵ_i^φ *of* x_i *w.r.t.* φ is defined as $\epsilon_i^\varphi = \frac{|\{c \in \varphi : x_i \in c\}|}{|\varphi|}$, i.e., as the number of clauses of φ containing x_i divided by the total number of clauses in φ. Moreover, for each $v \geq 1$, let $\chi(v)$ be defined by $\chi(v)^{\chi(v)} = v$.

Let $v^* = |\varphi^*||\psi^*|$ denote the volume of the input (=root) instance (φ^*, ψ^*). For the rest of this section, we assume that $|\varphi^*| + |\psi^*| \leq |\varphi^*| \cdot |\psi^*|$. In fact, in any instance which violates this inequality, either φ^* or ψ^* has at most one clause; in this case, DUAL is trivially solvable in polynomial time.

Algorithm B first constructs the root r of T and then recursively expands the nodes of T. For each node u with label $I(u) = (\varphi, \psi)$, algorithm B does the following.

The algorithm first performs a polynomial time computation, referred to here as LCHECK(φ, ψ), which is as follows. LCHECK(φ, ψ) first eliminates all redundant (i.e., non-minimal) clauses from φ and ψ and then tests whether some of the following conditions is violated:

1. $V(\varphi) = V(\psi)$;
2. $\max_{c \in \varphi} |c| \leq |\psi|$ and $\max_{c \in \psi} |c| \leq |\varphi|$;
3. $\min(\, |\varphi|,\, |\psi|\,) > 2$.

If LCHECK$(\varphi, \psi) = \mathit{true}$, then u is a leaf of T (i.e., not further expanded); whether $I(\varphi, \psi)$ is a dual monotone pair is then decided by some procedure TEST(φ, ψ) in polynomial time. In case TEST(φ, ψ) returns *false*, the original input $(\varphi^*,\, \psi^*)$ is not a dual monotone pair, and algorithm B returns *false*. Moreover, in this case a counterexample w to the duality of φ^* and ψ^* is computable in polynomial time from the path leading from the root r of T to u.

If LCHECK(φ, ψ) returns *false*, algorithm B chooses in polynomial time some appropriate variable x_i such that $\epsilon_i^{\varphi} > 0$ and $\epsilon_i^{\psi} > 0$, and creates two or more children of u by deterministically choosing one of three alternative decomposition rules **(i)**, **(ii)**, and **(iii)**. Each rule decomposes $I(u) = (\varphi, \psi)$ into smaller instances, whose respective volumes are summarized as follows. Let, as for algorithm A, $\varphi_0 = \{c - \{x_i\} \mid x_i \in c,\, c \in \varphi\}$, $\varphi_1 = \{c \mid x_i \notin c,\, c \in \varphi\}$, $\psi_0 = \{c' - \{x_i\} \mid x_i \in c',\, c' \in \psi\}$, and $\psi_1 = \{c' \mid x_i \notin c',\, c' \in \psi\}$. Furthermore, define $\epsilon(v) = 1/\chi(v)$, for any $v > 0$.

Rule (i) If $\epsilon_i^{\varphi} \leq \epsilon(v(u))$, then $I(u)$ is decomposed into:

- **a)** one instance $(\varphi_1, \psi_0 \wedge \psi_1)$ of volume $\leq (1 - \epsilon_i^{\varphi}) \cdot v(u)$;
- **b)** $|\psi_0|$ instances $I_1, \ldots, I_{|\psi_0|}$ of volume $\leq \epsilon_i^{\varphi} \cdot v(u)$ each. Each such instance I_j corresponds to one clause of ψ_0 and can thus be identified as the j-th clause of ψ_0 with an index $j \leq |\psi_0| < n$ (recall that n denotes the size of the original input).

Rule (ii) If $\epsilon_i^{\varphi} > \epsilon(v(u)) \geq \epsilon_i^{\psi}$, then $I(u)$ is decomposed into:

- **a)** one instance $(\psi_1, \varphi_0 \wedge \varphi_1)$ of volume $\leq (1 - \epsilon_i^{\psi}) \cdot v(u)$;
- **b)** $|\varphi_0|$ instances $I_1, \ldots, I_{|\varphi_0|}$ of volume $\leq \epsilon_i^{\psi} \cdot v(u)$ each. Each such instance I_j corresponds to one clause of φ_0 and can be identified by an index $j \leq |\varphi_0| < v^*$.

Rule (iii) If both $\epsilon_i^{\varphi} > \epsilon(v(u))$ and $\epsilon_i^{\psi} > \epsilon(v(u))$, then I is decomposed into:

- $\mathbf{c_0}$) one instance of volume $\leq (1 - \epsilon_i^{\varphi}) \cdot v(u)$, and
- $\mathbf{c_1}$) one instance of volume $\leq (1 - \epsilon_i^{\psi}) \cdot v(u)$.

Algorithm B returns *true* iff TEST$(I(u))$ returns *true* for each leaf u of the recursion tree. This concludes the description of algorithm B.

For each node u and child u' of u in T, we label the arc (u, u') with the precise type of rule that was used to generate u' from u. The possible labels are thus **(i.a)**, **(i.b)**, **(ii.a)**, **(ii.b)**, **(iii.c$_0$)**, and **(iii.c$_1$)**. We call **(i.a)** and **(ii.a)** *a-labels*, **(i.b)** and **(ii.b)** *b-labels*, and **(iii.c$_0$)** and **(iii.c$_1$)** *c-labels*. Any arc with a

b-label is in addition labeled with the index j of the respective instance I_j in the decomposition, which we refer to as the *j-label* of the arc.

Definition 5.1 *For any node u of the tree T, let $seq(u)$ denote the sequence of all edge-labels on the path from the root r of T to u.*

Clearly, if $seq(u)$ is known, then the entire path from r to u including all node-labels (in particular, the one of u) can be computed in polynomial time. Indeed, the depth of the tree is at most v^*, and adding a child to a node of T according to algorithm B is feasible in polynomial time.

The following lemma bounds the number of various labels which may occur in $seq(u)$.

Lemma 5.3 *For each node u in T, $seq(u)$ contains at most (i) v^* many a-labels, (ii) $\log v^*$ many b-labels, and (iii) $\log^2 v^*$ many c-labels.*

Proof. (i) Let us consider rule **(i.a)** first. Given that $\epsilon_i^\varphi > 0$, x_i effectively occurs in some clause of φ. Thus $|\varphi_1| < |\varphi|$. Moreover, by definition of ψ_0 and ψ_1, $|\psi_0 \wedge \psi_1| \leq |\psi|$. Thus we have $|\varphi_1| \cdot |\psi_0 \wedge \psi_1| < |\varphi| \cdot |\psi|$. It follows that whenever rule **(i.a)** is applied, the volume decreases (at least by 1). The same holds for rule **(ii.a)** by a symmetric argument. Since no rule ever increases the volume, there are at most v^* applications of an a-rule.
(ii) Assume that rule **(i.b)** is applied to generate a child t' of node t. By condition 3 of LCHECK, $v(t) > 4$. Therefore, $\chi(v(t)) > 2$ and thus $\epsilon_i^\varphi \leq \epsilon(v(t)) < 1/2$. It follows that $v(t') < v(t)/2$. The same holds if t' results from t via rule $(\mathbf{ii.b})$. Because no rule ever increases the volume, any node generated after (among others) $\log v^*$ applications of a b-rule has volume ≤ 1 and is thus a leaf in T.
(iii) If a c-rule is applied to generate a child t' of a node t, and since $\epsilon(v(t)) > \epsilon(v^*) > 1/\log v^*$, the volume of $v(t)$ decreases at least by factor $(1 - 1/\log v^*)$. Thus, the volume of any node u which results from t after $\log v^*$ applications of a c-rule satisfies $v(u) \leq v(t)(1 - 1/\log v^*)^{\log v^*} \leq v(t)/e$ by (9); i.e., the volume has decreased more than half. Thus, any node u resulting from the root of T after $\log^2 v^*$ applications of a c-rule satisfies $v(u) \leq v^* \cdot \left(\frac{1}{2}\right)^{\log v^*} = 1$; that is, u is a leaf in T. □

Theorem 5.4 *Deciding whether monotone CNFs φ and ψ are non-dual is feasible in polynomial time with $O(\log^2 n)$ nondeterministic steps, where $n = |\varphi| + |\psi|$.*

Proof. As in the proof of Theorem 5.2, we use a compact representation $seq^*(u)$ of $seq(u)$. However, here the definition of seq^* is somewhat more involved:

- $seq^*(u)$ contains all b-labels of $seq(u)$, which are the anchor elements of $seq^*(u)$. Every b-label is immediately followed by its associated j-label, i.e., the label specifying which of the (many) b-children is chosen. We call a b-label and its associated j-label a *bj-block*.

- At the beginning of $seq^*(u)$, as well as after each bj-block, there is an *ac-block*. The first ac-block in $seq^*(u)$ represents the sequence of all a- and c-labels in $seq(u)$ preceding the first b-label in $seq(u)$, and the i-th ac-block in $seq^*(u)$, $i > 1$, represents the sequence of the a and c labels (uninterrupted by any other label) following the $(i-1)$-st bj-block in $seq(u)$.

  Each ac-block consists of an α-block followed by a γ-block, where

– the α-block contains, in binary, the *number* of a-labels in the ac-block, and

– the γ-block contains all c-labels (single bits) in the ac-block, in the order as they appear.

For example, if s = "$(i.a), (ii.a), c_0, (ii.a), c_1, c_0, (i.a)$" is a maximal ac-subsequence in $seq(u)$, then its corresponding ac-block in $seq^*(u)$ is "$10, c_0, c_1, c_0$", where 10 (= 4) is the α-block (stating that there are four a-labels) and "c_0, c_1, c_0" is the γ-block enumerating the c-labels in s in their correct order.

The following facts are now the key to the result.

Fact A. Given ϕ^*,ψ^* and a string s, it is possible to compute in polynomial time the path $r = u_0, u_1, \ldots,$ $u_l = u$ from the root r of T to the unique node u in T such that $s = seq^*(u)$ and all labels $I(u_i)$, or to tell that no such node u exists (i.e., $s \neq seq^*(u)$ for every node u in T).

This can be done by a simple procedure, which incrementally constructs u_0, u_1, etc as follows.

Create the root node $r = u_0$, and set $I(u_0) = (\phi^*, \psi^*)$ and $t := 0$. Generate the next node u_{t+1} and label it, while processing the main blocks (ac-blocks and bj-blocks) in s in order, as follows:

ac-block: Suppose the α-block of the current ac-block has value n_α, and the γ-block contains labels $\gamma_1, \ldots, \gamma_k$. Set up counters $p := 0$ and $q := 0$, and while $p < n_\alpha$ or $q < k$, do the following.

If LCHECK$(I(u_t)) = true$, then flag an error and halt, as $s \neq seq^*(u)$ for every node u in T. Otherwise, determine the rule type $\tau \in \{(\mathbf{i}), (\mathbf{ii}), (\mathbf{iii})\}$ used by algorithm B to (deterministically) decompose $I(u_t)$.

- If $\tau \in \{(\mathbf{i}), (\mathbf{ii})\}$ and $p < n_\alpha$, then assign $I(u_{t+1})$ the a-child of $I(u_t)$ according to algorithm B, and increment p and t by 1.
- If $\tau = (\mathbf{iii})$ and $q < k$, then increment q by 1, assign $I(u_{t+1})$ the γ_q-child of $I(u_t)$ according to algorithm B, and increment t by 1.
- In all other cases (i.e., either $\tau \in \{(\mathbf{i}), (\mathbf{ii})\}$ and $p \geq n_\alpha$, or $\tau = (\mathbf{iii})$ and $q \geq k$), flag an error and halt, since $s \neq seq^*(u)$ for every node u in T.

bj-block: Determine the rule type $\tau \in \{(\mathbf{i}), (\mathbf{ii}), (\mathbf{iii})\}$ used by algorithm B to (deterministically) decompose $I(u_t)$. If $\tau = (\mathbf{iii})$, then flag an error and halt, since $s \neq seq^*(u)$ for every node u in T. Otherwise, assign $I(u_{t+1})$ the j'-th $(\tau.\mathbf{b})$-child of $I(u_t)$ according to rule $(\tau.\mathbf{b})$ of algorithm B, where j' is the j-label of the current bj-block.

Clearly, this procedure outputs in polynomial time the desired labeled path from r to u, or flags an error if $s \neq seq^*(u)$ for every node u in T.

Let us now bound the size of $seq^*(u)$ in terms of the original input size v^*.

Fact B. For any u in T, the size of $seq^*(u)$ is $O(\log^2 v^*)$.

By Lemma 5.3 (ii), there are $< \log v^*$ bj-blocks. As already noted, each bj-block has size $O(\log v^*)$; thus, the total size of all bj-blocks is $O(\log^2 v^*)$. Next, there are at most $\log v^*$ many ac-blocks and thus α-blocks. Each α-block encodes a number of $< v^*$ a-rule applications (see Lemma 5.3.(i)), and thus uses at most $\log v^*$ bits. The total size of all α-blocks is thus at most $\log^2 v^*$. Finally, by Lemma 5.3 (iii), the total size of all γ-blocks is at most $\log^2 v^*$. Overall, this means that $seq^*(u)$ has size $O(\log^2 v^*)$.

To prove that algorithm B rejects input (φ^*, ψ^*), it is thus sufficient to guess $seq^*(u)$ for some leaf u in T, to compute in polynomial time the corresponding path $r = u_0, u_1, \ldots, u_l = u$, and to verify that LCHECK$(I(u))$ =TEST$(I(u)) = true$. Therefore, non-duality of ϕ^* and ψ^* can be decided in polynomial time with $O(\log^2 v^*)$ bit guesses. Given that $v^* \leq n^2$, the number of guesses is $O(\log^2 n^2) = O(\log^2 n)$. □

The following result is an immediate consequence of this theorem.

Corollary 5.5 *Problem* DUAL *is in* co-β_2P *and solvable in deterministic* $n^{O(\log n)}$ *time, where* $n = |\varphi| + |\psi|$.

(Note that Yes-instances of DUAL must have size polynomial in n, since dual monotone pairs (φ, ψ) must satisfy conditions (2) and (3) in step 2 of algorithm A.) We remark that the proof of Lemma 5.3 and Theorem 5.4 did no stress the fact that $\epsilon(v) = 1/\chi(v)$; the proofs go through for $\epsilon(v) = 1/\log v$ as well. Thus, the use of the χ-function is not essential for deriving Theorem 5.4.

However, a tighter analysis of the size of $seq^*(u)$ stressing $\chi(v)$ yields a better bound for the number of nondeterministic steps. In fact, we show in the next result that $O(\chi(n) \cdot \log n)$ bit guesses are sufficient. Note that $\chi(n) = o(\log n)$, thus the result is an effective improvement. Moreover, it also shows that DUAL is most likely not complete for co-β_2P.

Theorem 5.6 *Deciding whether monotone CNFs* φ *and* ψ *are non-dual is feasible in polynomial time with* $O(\chi(n) \log n)$ *nondeterministic steps, where* $n = |\varphi| + |\psi|$.

Proof. In the proof of Theorem 5.4, our estimates of the components of $seq^*(u)$ were rather crude. With more effort, we establish the following.

Fact C. For any u in T, the size of $seq^*(u)$ is $O(\chi(v^*) \cdot \log(v^*))$.

Assume node u' in T is a child of u generated via a b-rule. The j-label of the arc (u, u') serves to identify one clause of $I(u)$. Clearly, there are no more than $v(u)$ such clauses. Thus $\log v(u)$ bits suffice to represent any j-label.

Observe that if u is a node of T, then any path π from u to a node w in T contains at most $v(u)$ nodes, since the volume always decreases by at least 1 in each decomposition step. Thus, the number of a-labeled arcs in π is bounded by $v(u)$ and not just by v^* $(= v(r))$.

For each node u and descendant w of u in T, let

$$f(u, w) = \sum_{u' \in B(u,w)} \log v(u'),$$

where $B(u, w)$ is the set of all nodes t on the path from u to w such that the arc from t to its successor on the path is b-labeled.

By what we have observed, the total size of all encodings of j-labels in $seq^*(u)$ is at most $f(v^*, u)$ and the size of all α-blocks in $seq^*(u)$ is at most $\log(v^*) + f(v^*, u)$, were the first term takes care of the first α-block and the second of all other α-blocks. Therefore, the total size of all α-blocks and all bj-blocks in $seq^*(u)$ is $O(f(v^*, u) + \log(v^*))$.

We now show that for each node u and descendant w of u in T, it holds that

$$f(u, w) \leq \log(v(u)) \cdot \chi(v(u)).$$

The proof is by induction on the number $|B(u, w)|$ of b-labeled arcs on the path π from u to w. If $|B(u, w)| = 0$, then obviously $f(u, w) = 0 \leq v(u)$.

Assume the claim holds for $|B(u', w)| \leq i$ and consider $|B(u, w)| = i + 1$. Let t be the first node on π contained in $B(u, w)$, and let t' be its child on π. Clearly, $f(u, w) = f(t, w)$, and thus we obtain:

$$\begin{aligned} f(u, w) &= \log(v(t)) + f(t', w) \\ &\leq \log(v(t)) + \log(v(t')) \cdot \chi(v(t')) && \text{(induction hypothesis)} \\ &\leq \log(v(t)) + (\log(v(t)) - \log(\chi(v(t)))) \cdot \chi(v(t)) && (\text{as } v(t') \leq \tfrac{v(t)}{\chi(v(t))},\ \chi(v(t')) \leq \chi(v(t))\,) \\ &= \log(v(t)) \cdot \chi(v(t)) && (\text{as } \log(\chi(y)) \cdot \chi(y) = \log y, \text{ for all } y). \end{aligned}$$

Thus, $f(u, w) \leq \log(v(u)) \cdot \chi(v(u))$. This concludes the induction and proves the claim.

Finally, we show that the total size of all γ blocks in $seq^*(u)$, i.e., the number of all c-labels in $seq(u)$, is bounded by $\chi(v^*) \cdot \log(v^*) < \log^2 v^*$. Indeed, assume a c-rule is applied to generate a child t' of any node t, and let $v = v(t)$, $v' = v(t')$. Since $\epsilon_i^\varphi > \epsilon(v)$ and $\epsilon_i^\psi > \epsilon(v)$, we have $v' < (1 - \epsilon(v)) \cdot v$. Since $\chi(v^*) > \chi(v)$, we have $\epsilon(v) = 1/\chi(v) > 1/\chi(v^*)$ and thus

$$v' < \left(1 - \frac{1}{\chi(v^*)}\right) \cdot v.$$

Hence, any node in T resulting after $\chi(v^*) \cdot \log(v^*)$ applications of a c-rule has volume at most

$$v^* \cdot \left(1 - \frac{1}{\chi(v^*)}\right)^{\chi(v^*) \cdot \log v^*} = v^* \cdot \left[\left(1 - \frac{1}{\chi(v^*)}\right)^{\chi(v^*)}\right]^{\log v^*} \leq v^* \cdot \left(\frac{1}{e}\right)^{\log v^*} \leq 1$$

(cf. also (9)). Consequently, along each branch in T there must be no more than $\chi(v^*) \cdot \log v^*$ applications of a c-rule. In summary, the total sizes of all α-blocks, all γ-blocks, and all encodings of j-labels in $seq^*(u)$ are all bounded by $\chi(v^*) \cdot \log v^*$. This proves Fact C.

As a consequence, non-duality of a monotone pair (φ^*, ψ^*) can be recognized in polynomial time with $O(\chi(v^*) \cdot \log v^*)$ many bit guesses. As already observed on the last lines of [17], we have $\chi(v^*) < 2\chi(n)$. Furthermore, $v^* \leq n^2$, thus $\log v^* \leq 2 \log n$. Hence, non-duality (φ^*, ψ^*) can be recognized in polynomial time with $O(\chi(n) \cdot \log(n))$ bit guesses. □

Corollary 5.7 *Problem* DUAL *is solvable in deterministic* $n^{O(\chi(n))}$ *time, where* $n = |\varphi| + |\psi|$.

Remark 5.1 Note that the sequence $seq(u)$ describing a path from the root of T to a "failure leaf" with label $I(u) = (\varphi', \psi')$ describes a choice of values for all variables in $V(\varphi \wedge \psi) \setminus V(\varphi' \wedge \psi')$. By completing it with values for $V(\varphi' \wedge \psi')$ that show non-duality of (φ', ψ'), which is possible in polynomial time, we obtain in polynomial time from $seq(u)$ a vector w such that $f(w) \neq g^d(w)$. It also follows from the proof of Theorem 5.6 that a witness w for $f \neq g^d$ (if one exists) can be found in polynomial time with $O(\chi(n) \cdot \log n)$ nondeterministic steps.

5.3 Application of Beigel and Fu's results

While our independently developed methods substantially differ from those in [1, 2], membership of problem DUAL in co-β_2P may also be obtained by exploiting Beigel and Fu's Theorem 8 in [1] (or, equivalently, Theorem 11 in [2]). They show how to convert certain recursive algorithms that use disjunctive self-reductions, have runtime bounded by $f(n)$, and fulfill certain additional conditions, into polynomial algorithms using $\log(f(n))$ nondeterministic steps (cf. [2, Section 5]).

Let us first introduce the main relevant definitions of [1]. Let $\|y\|$ denote the size of a problem instance y.

Definition 5.2 ([1]) *A partial order $\prec$ (on problem instances) is polynomially well-founded, if there exists a polynomial-bounded function p such that*

- $y_m \prec \cdots \prec y_1 \Rightarrow m \leq p(\|y_1\|)$ *and*
- $y_m \prec \cdots \prec y_1 \Rightarrow \|y_m\| \leq p(\|y_1\|)$.

For technical simplicity, [1] considers only languages (of problem instances) containing the empty string, Λ.

Definition 5.3 ([1]) *A disjunctive self-reduction (for short, d-self-reduction) for a language L is a pair $\langle h, \prec \rangle$ of a polynomial-time computable function $h(x) = \{x_1, \ldots, x_m\}$ and a polynomially well-founded partial order $\prec$ on problem instances such that*

- *Λ is the only minimal element under $\prec$;*
- *for all $x \neq \Lambda$, $x \in L \equiv h(x) \cap L \neq \emptyset$;*
- *for all x, $x_i \in h(x) \Rightarrow x_i \prec x$.*

Definition 5.4 ([1]) *Let $\langle h, \prec \rangle$ be a d-self-reduction and let x be a problem instance.*

- *$T_{h,\prec}(x)$ is the unordered rooted tree that satisfies the following rules: (1) the root is x; (2) for each y, the set of children of y is $h(y)$.*
- *$|T_{h,\prec}(x)|$ is the number of leaves in $T_{h,\prec}(x)$.*

Definition 5.5 ([1]) *Let T be a polynomial-time computable function. A language L is in $\mathrm{REC}(T(x))$, if there is a d-self-reduction $\langle h, \prec \rangle$ for L such that for all x*

1. *$|T_{h,\prec}(x)| \leq T(x)$, and*
2. *$T(x) \geq \sum_{x_i \in h(x)} T(x_i)$.*

Let $T(x)$-P denote the set of all (languages of) problems whose Yes-instances x are recognizable in polynomial time with $T(x)$ nondeterministic bit guesses.

Theorem 5.8 ([1]) $\mathrm{REC}(T(x)) \subseteq \lceil \log T(x) \rceil$-P

We now show that Theorem 5.8, together with Fredman's and Khachiyan's proof of the deterministic complexity of algorithm B, can be used to prove that problem DUAL is in co-β_2P.

Let L denote the set of all non-dual monotone pairs (φ, ψ) plus Λ. Let us identify each monotone pair (φ, ψ) which satisfies LCHECK(φ, ψ) but does not satisfy TEST(φ, ψ) with the "bottom element" Λ. Thus, if a node in the recursion tree T has a child labeled with such a pair, then the label is simply replaced by Λ.

Let us define the order $\prec$ on monotone pairs plus Λ as follows: $J \prec I$, if $I \neq J$ and either $J = \Lambda$ or J labels a node of the recursion tree generated by algorithm B on input I. It is easy to see that both conditions of Definition 5.2 apply; therefore, $\prec$ is polynomially well-founded. In fact, we may define the polynomial p by the identity function; since the sizes of the instances in the recursion tree strictly decrease on each path in T, the two conditions hold.

Define h as the function which associates with each monotone pair $I = (\varphi, \psi)$ those instances that label all children of the root by algorithm B on input I. Clearly h satisfies all three conditions of Definition 5.3, and hence $\langle h, \prec \rangle$ is a d-self-reduction for L.

Let T be the function which to each instance I associates $v(I)^{\log v(I)}$ (recall that $v(I)$ denotes the volume of I). It is now sufficient to check that conditions 1 and 2 of Definition 5.5 are satisfied, and to ensure that Theorem 5.8 can be applied.

That item 1 of Definition 5.5 is satisfied follows immediately from Lemma 5 in [17], which states that the maximum number of recursive calls of algorithm B on any input I of volume v is bounded by $v^{\chi(v)}$ $(\leq v^{\log v})$. Retain, however, that the proof of this lemma is noticeably more involved than our proof of the membership of DUAL in co-β_2P.

To verify item 2 of Definition 5.5, it is sufficient to prove that for a volume $v > 4$ of any input instance to algorithm B, it holds that

$$v^{\log v} \geq (v-1)^{\log(v-1)} + \frac{v}{3} \cdot \left(\frac{v}{2}\right)^{\log \frac{v}{2}}, \quad \text{and} \tag{10}$$

$$v^{\log v} \geq 2(\alpha \cdot v)^{\log(\alpha \cdot v)}, \quad \text{where } \alpha = 1 - 1/\log v; \tag{11}$$

here, (10) arises from the rules $(\mathbf{i})$, $(\mathbf{ii})$ and (11) from rule $(\mathbf{iii})$. As for (10), the child of u from $(\mathbf{i.a})$ resp. $(\mathbf{ii.a})$ has volume at most $v-1$, and there are at most $v/3$ many children from$(\mathbf{i.b})$ resp. $(\mathbf{ii.b})$, since $\min(|\varphi|, |\psi|) > 2$ (recall that $v = |\varphi| \cdot |\psi|$); furthermore, each such child has volume $\leq \epsilon(v) \cdot v \leq \frac{1}{2}v$. In case of (11), the volume of each child of u is bounded by $(1-\epsilon(v)) \cdot v \leq (1 - 1/\log v) \cdot v$; note also that $v^{\log v}$ monotonically increases for $v > 4$. To see (10), we have

$$\begin{aligned}
(v-1)^{\log(v-1)} + \tfrac{v}{3} \cdot \left(\tfrac{v}{2}\right)^{\log \frac{v}{2}} &\leq (v-1)^{\log v} + \tfrac{v}{3} \cdot \tfrac{v^{\log v - 1}}{2^{\log v - 1}} \\
&= v^{\log v} \cdot (1 - \tfrac{1}{v})^{\log v} + \tfrac{2 \cdot v^{\log v}}{3 \cdot v} \\
&\leq v^{\log v} \cdot (1 - \tfrac{1}{v} + \tfrac{2}{3 \cdot v}) \\
&= v^{\log v} \cdot (1 - \tfrac{1}{3 \cdot v}) \\
&< v^{\log v};
\end{aligned}$$

to show (11), note that

$$
\begin{aligned}
2(\alpha \cdot v)^{\log(\alpha \cdot v)} &= 2\alpha^{\log v + \log \alpha} \cdot v^{\log v + \log \alpha} \\
&\leq 2(\tfrac{1}{e} \cdot \alpha^{\log \alpha}) \cdot v^{\log v + \log \alpha} && (\alpha^{\log v} \leq 1/e, \text{ by (9)}) \\
&= \tfrac{2}{e} \cdot (\alpha \cdot v)^{\log \alpha} \cdot v^{\log v} \\
&\leq \tfrac{2}{e} \cdot v^{\log v} && (\alpha \cdot v)^{\log \alpha} \leq 1, \text{ i.e., } \log \alpha \cdot (\log \alpha + \log v) \leq 0, \\
& && \text{since } -1 < \log \alpha \leq 0 \text{ and } \log v > 2 \\
&< v^{\log v}.
\end{aligned}
$$

We can thus apply Theorem 5.8 and conclude that the complement of DUAL is in $\lceil \log T(x) \rceil$-P, and thus also in $\beta_2\mathrm{P}$.

The advantage of Beigel and Fu's method is its very abstract formulation. The method has two disadvantages, however, that are related to the two items of Definition 5.5.

The first item requires that $T(x)$ is at least the number of leaves in the tree for x. In order to show this, one must basically prove a deterministic time bound for the considered algorithm (or at least a bound of the number of recursive calls for each instance, which is often tantamount to a time-bound). The method does not suggest how to do this, but presupposes that such a bound exists (in the present case, this was done by Fredman and Khachiyan in a nontrivial proof). The second item requires to prove that the T-value of any node x in the recursion tree is at least the sum of the T-values of its children. This may be hard to show in many cases, and does not necessarily hold for every upper bound T.

Our method instead does not require an a priori time bound, but directly constructs a nondeterministic algorithm from the original deterministic algorithm, which lends itself to a simple analysis that directly leads to the desired nondeterministic time bound. The deterministic time bound follows as an immediate corollary. It turns out (as exemplified by the very simple proof of Theorem 5.4) that the analysis involved in our method can be simpler than an analysis according to previous techniques.

6 Conclusion

We have presented several new cases of the monotone dualization problem which are solvable in output-polynomial time. These cases generalize some previously known output-polynomial cases. Furthermore, we have shown by rather simple means that non-dual monotone pairs (φ, ψ) can be recognized, using a nondeterministic variant of Fredman and Khachiyan's algorithm B [17], in polynomial time with $O(\log^2 n)$ many bit guesses, which places problem DUAL in the class co-$\beta_2\mathrm{P}$. In fact, a refined analysis revealed that this is feasible in polynomial time with $O(\chi(n) \cdot \log n)$ many bit guesses.

While our results document progress on DUAL and DUALIZATION and reveal novel properties of these problems, the question whether dualization of monotone pairs (φ, ψ) is feasible in polynomial time remains open. It would be interesting to see whether the amount of guessed bits can be further significally decreased, e.g., to $O(\log \log v \cdot \log v)$ many bits.

Acknowledgments

Georg Gottlob thanks José Balcázar for inviting him to an interesting workshop in Bellaterra, Barcelona, Spain, and to Leonard Pitt for giving an elucidating lecture on the dualization problem there.

References

[1] R. Beigel and B. Fu. Molecular computing, bounded nondeterminism, and efficient recursion. In: *Proc. 24th International Colloquium on Automata, Languages and Programming (ICALP)*, pp. 816-826, Springer LNCS 1256, 1997.

[2] R. Beigel and B. Fu. Molecular computing, bounded nondeterminism, and efficient recursion. *Algorithmica*, 25: 222–238, 1999.

[3] C. Bioch and T. Ibaraki. Complexity of identification and dualization of positive Boolean functions. *Information and Computation*, 123:50–63, 1995.

[4] 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.

[5] E. Boros, K. Elbassioni, V. Gurvich, L. Khachiyan and K. Makino. On generating all minimal integer solutions for a monotone system of linear inequalities. In: *Proc. 28th International Colloquium on Automata, Languages and Programming (ICALP)*, pp. 22–103, Springer LNCS 2076, 2001.

[6] E. Boros, V. Gurvich, L. Khachiyan and K. Makino. Dual-bounded generating problems: Partial and multiple transversals of a Hypergraph. *SIAM Journal on Computing*, 30:2036–2050, 2001.

[7] E. Boros, V. Gurvich, L. Khachiyan and K. Makino. On the complexity of generating maximal frequent and minimal infrequent sets. In: *Proc. 19th International Symposium on retical Aspects of Computer Science (STACS)*, pp. 133–141, Springer LNCS 2285, 2002.

[8] E. Boros, V. Gurvich, and P. L. Hammer. Dual subimplicants of positive Boolean functions. *Optimization Methods and Software*, 10:147–156, 1998.

[9] E. Boros, P. L. Hammer, T. Ibaraki and K. Kawakami, Polynomial time recognition of 2-monotonic positive Boolean functions given by an oracle, *SIAM Journal on Computing*, 26 (1997) 93-109.

[10] Y. Crama. Dualization of regular Boolean functions. *Discrete Applied Mathematics*, 16:79–85, 1987.

[11] C. Domingo. Private communication.

[12] C. Domingo, N. Mishra and L. Pitt. Efficient read-restricted monotone CNF/DNF dualization by learning with membership queries. *Machine Learning*, 37:89–110, 1999.

[13] C. Chekuri and A. Rajaraman. Conjunctive query containment revisited. In: *Proc. 6th International Conference on Database Theory (ICDT)*, Delphi, Greece, Springer LNCS 1186, pp. 56–70, 1997.

[14] T. Eiter. Exact transversal hypergraphs and application to Boolean μ-functions. *Journal of Symbolic Computation*, 17:215–225, 1994.

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

[16] R. Fagin. Degrees of acyclicity for hypergraphs and relational database schemes. *Journal of the ACM*, 30:514–550, 1983.

[17] M. Fredman and L. Khachiyan. On the complexity of dualization of monotone disjunctive normal forms. *Journal of Algorithms*, 21:618–628, 1996.

[18] H. Garcia-Molina and D. Barbara. How to assign votes in a distributed system. *Journal of the ACM*, 32(4):841–860, 1985.

[19] D.R. Gaur. *Satisfiability and self-duality of monotone Boolean functions*. Ph.D. thesis, School of Computing Science, Simon Fraser University, January 1999.

[20] D.R. Gaur and R. Krishnamurti. Self-duality of bounded monotone Boolean functions and related problems. In: *Proc. 11th International Conference on Algorithmic Learning Theory (ALT)*, pp. 209-223, Springer LNCS 1968, 2000.

[21] G. Gogic, C. Papadimitriou, and M. Sideri. Incremental recompilation of knowledge. *Journal of Artificial Intelligence Research*, 8:23–37, 1998.

[22] D. Gunopulos, R. Khardon, H. Mannila, and H. Toivonen. Data mining, hypergraph transversals, and machine learning. *Proc. 16th ACM Symp. on Principles of Database Systems (PODS)*, pp. 209–216, 1997.

[23] J. Goldsmith, M. Levy, and M. Mundhenk. Limited nondeterminism. *SIGACT News*, 27(2):20-29, 1996.

[24] G. Gottlob, N. Leone, and F. Scarcello. Hypertree decompositions and tractable queries. In: *Proc. 18th ACM Symp. on Principles of Database Systems (PODS)*, pp. 21-32, 1999. Full paper to appear in *Journal of Computer and System Sciences*.

[25] M. Graham. On the universal relation. Technical Report, University of Toronto, Canada, September 1979.

[26] V. Gurvich. Nash-solvability of games in pure strategies. *USSR Comput. Math and Math. Phys.*, 15(2):357–371, 1975.

[27] T. Ibaraki and T. Kameda. A theory of coteries: Mutual exclusion in distributed systems. *IEEE Transactions on Parallel and Distributed Systems*, 4(7):779–794, 1993.

[28] D. Kavvadias, C. H. Papadimitriou, and M. Sideri, On Horn envelopes and hypergraph transversals. In: *Proc. 4th International Symposium on Algorithms and Computation (ISAAC)*, pp. 399–405, Springer LNCS 762, 1993.

[29] R. Khardon. Translating between Horn representations and their characteristic models. *Journal of Artificial Intelligence Research*, 3:349-372, 1995.

[30] D. S. Johnson. Open and closed problems in NP-completeness. Lecture given at the International School of Mathematics "G. Stampacchia": Summer School "NP-Completeness: The First 20 Years", Erice (Sicily) , Italy, June 20-27, 1991.

[31] D. S. Johnson, M. Yannakakis, and C. H. Papadimitriou. On generating all maximal independent sets. *Information Processing Letters*, 27:119–123, 1988.

[32] C.M.R. Kintala and P. Fischer. Refining nondeterminism in relativized polynomial-time bounded computations. *SIAM Journal on Computing*, 9:46–53, 1980.

[33] E. Lawler, J. Lenstra, and A. Rinnooy Kan. Generating all maximal independent sets: NP-hardness and polynomial-time algorithms. *SIAM Journal on Computing*, 9:558–565, 1980.

[34] L. Lovász. Combinatorial optimization: Some problems and trends. DIMACS Technical Report 92-53, RUTCOR, Rutgers University, 1992.

[35] K. Makino and T. Ibaraki. The maximum latency and identification of positive Boolean functions. *SIAM Journal on Computing*, 26:1363–1383, 1997.

[36] K. Makino and T. Ibaraki, A fast and simple algorithm for identifying 2-monotonic positive Boolean functions. *Journal of Algorithms*, 26:291–305, 1998.

[37] K. Makino. Efficient dualization of $O(\log n)$-term monotone disjunctive normal forms. Technical Report 00-07, Discrete Mathematics and Systems Science, Osaka University, 2000; to appear in *Discrete Applied Mathematics*.

[38] H. Mannila and K.-J. Räihä. Design by example: An application of Armstrong relations. *Journal of Computer and System Sciences*, 22(2):126–141, 1986.

[39] N. Mishra and L. Pitt. Generating all maximal independent sets of bounded-degree hypergraphs. In: *Proc. Tenth Annual Conference on Computational Learning Theory (COLT)*, pp. 211–217, 1997.

[40] Ch. H. Papadimitriou. NP-completeness: A retrospective, In: *Proc. 24th International Colloquium on Automata, Languages and Programming (ICALP)*, pp.2–6, Springer LNCS 1256, 1997.

[41] U. N. Peled and B. Simeone. An $O(nm)$-time algorithm for computing the dual of a regular Boolean function. *Discrete Applied Mathematics* 49:309–323, 1994.

[42] K. G. Ramamurthy. *Coherent Structures and Simple Games*. Kluwer Academic Publishers, 1990.

[43] R. C. Read, Every one a winner, or how to avoid isomorphism when cataloging combinatorial configurations. *Annals of Discrete Mathematics* 2:107–120, 1978.

[44] R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32:57–95, 1987.

[45] N. Robertson and P. Seymour. Graph minors II: Algorithmic aspects of tree-width. *Journal of Algorithms*, 7:309–322, 1986.

[46] K. Takata. On the sequential method for listing minimal hitting sets. In *Proceedings Workshop on Discrete Mathematics and Data Mining, 2nd SIAM International Conference on Data Mining, April 11-13, Arlington, Virginia, USA*, 2002.

[47] H. Tamaki. Space-efficient enumeration of minimal transversals of a hypergraph. IPSJ-AL 75:29-36, 2000.

[48] R. E. Tarjan and M. Yannakakis. Simple linear time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM Journal on Computing*, 13:566–579, 1984.

[49] V. D. Thi. Minimal keys and antikeys. *Acta Cybernetica*, 7(4):361–371, 1986.

[50] B. Toft. Colouring, Stable sets and perfect graphs. *Handbook of Combinatorics*, Vol. 1 Chapter 4. Elsevier, 1995.

[51] C. T. Yu and M. Ozsoyoglu. An algorithm for tree-query membership of a distributed query. *Proceedings IEEE COMPSAC*, pp. 306–312, 1979.