

## COMPLEXITY CLASSIFICATIONS FOR DIFFERENT EQUIVALENCE AND AUDIT PROBLEMS FOR BOOLEAN CIRCUITS

ELMAR BÖHLER<sup>a</sup>, NADIA CREIGNOU<sup>b</sup>, MATTHIAS GALOTA<sup>c</sup>, STEFFEN REITH<sup>d</sup>,  
HENNING SCHNOOR<sup>e</sup>, AND HERIBERT VOLLMER<sup>f</sup>

<sup>a</sup> Theoretische Informatik, Universität Würzburg, Am Hubland, D-97030 Würzburg, Germany  
*e-mail address:* boehler@informatik.uni-wuerzburg.de

<sup>b</sup> Aix-Marseille Université, CNRS, LIF UMR 7279, 13 288 Marseille, France  
*e-mail address:* creignou@lif.univ-mrs.fr

<sup>c</sup> Elektrobit, Am Wolfsmantel 46, D-91058 Erlangen, Germany  
*e-mail address:* Matthias.Galota@elektrobit.com

<sup>d</sup> Theoretische Informatik, FB DCSM, Hochschule RheinMain, Kurt-Schumacher-Ring 18, D-65197  
Wiesbaden, Germany  
*e-mail address:* Steffen.Reith@hs-rm.de

<sup>e</sup> Institut für Informatik, Christian-Albrechts-Universität zu Kiel, Christian-Albrechts-Platz 4, 24118  
Kiel  
*e-mail address:* schnoor@ti.informatik.uni-kiel.de

<sup>f</sup> Institut für Theoretische Informatik, Leibniz Universität Hannover, Appelstraße 4, 30167 Han-  
nover, Germany  
*e-mail address:* vollmer@thi.uni-hannover.de

---

ABSTRACT. We study Boolean circuits as a representation of Boolean functions and consider different equivalence, audit, and enumeration problems. For a number of restricted sets of gate types (bases) we obtain efficient algorithms, while for all other gate types we show these problems are at least NP-hard.

### 1. INTRODUCTION

The study of Boolean functions is an active research topic since more than one hundred years. Since the early papers of Shannon [RS42, Sha38] and Lupanov [Lup58] in the 1940s and 1950s, Boolean circuits (then called *switching circuits*) have been used as a computation model for Boolean functions. The computational complexity theory of Boolean circuits developed rapidly, see Savage’s textbook [Sav76]. In the meantime many beautiful results have

---

1998 ACM Subject Classification: F.2.2.

Key words and phrases: Boolean circuits, complexity classification, isomorphism.

<sup>b</sup> Supported by the Agence Nationale de la Recherche under grant ANR-09-BLAN-0011-01.

<sup>f</sup> Supported by DFG VO 630/6-2.

been proven, e. g., in the area of lower bounds or of algebraic and logical characterizations of small circuit classes, cf. [Weg87, Vol99].

Another development of equal importance is the search for different representations (sometimes also called *data structures*, see, e.g., the books [MT98, Weg00]) for Boolean functions that may facilitate solving presumably hard problems. Let us explain this with an example. The well-known satisfiability problem for propositional logic is known to be NP-complete. This immediately implies that the problem, given a Boolean circuit  $C$ , to decide if there is an input for which  $C$  outputs 1 is NP-complete as well. Thus, using Boolean circuits as a representation for a Boolean function  $f$ , to determine if  $f^{-1}(1)$  is not empty appears to be a computationally hard problem. However, if we represent  $f$  by a decision tree, satisfiability can be solved in polynomial time (in the size of the decision tree). The same holds for ordered binary decision diagrams and different further types of so called *branching programs*, see [Weg00]. This advantage of course has its price: generally, Boolean circuits are a much more succinct way of representing Boolean functions. Nevertheless, since the pioneering work by R. E. Bryant, branching programs and in particular ordered binary decision diagrams have turned out to be a suitable representation for many application areas such as model checking, VLSI design, computer-aided design, etc; we refer the interested reader to [Weg00] for a discussion.

In this paper, a different approach is advocated. While it is known that in general satisfiability for Boolean circuits is NP-complete, there are prominent easy special cases: For example, if we consider only circuits over a monotone base, the satisfiability problem admits an efficient solution. Another example is that of linear circuits (i. e., circuits with a base of linear functions). This phenomenon was studied systematically by H. R. Lewis in 1979, who showed that satisfiability is NP-complete if the base contains or can implement the negation of implication, i. e., the function  $x \wedge \neg y$ . In all other cases, satisfiability has a polynomial-time algorithm. This *dichotomy result* holds for Boolean circuits as well as for propositional formulas. The work of Lewis has been taken up by Reith and Wagner [RW05] who examined further algorithmic problems such as the circuit value problem and the problem of counting the number of satisfying assignments.

Here we study further important algorithmic tasks for the representation of Boolean functions by Boolean circuits: First we examine the *equivalence problem*, i. e., the question if two given Boolean circuits represent logically equivalent Boolean functions, and the *isomorphism problem*, i. e., the problem if two given circuits can be made equivalent through a permutation of their input variables. While these problems are of enormous interest in the area of verification and model-checking, it should be remarked that also from a theoretical viewpoint they have a long history: they were studied by Jevons and Clifford in the 19th century and in particular the isomorphism problem became known as the “Jevons-Clifford Problem”. The isomorphism problem admittedly gains its importance from a more theoretical point of view. In complexity theory, isomorphism problems in general are notorious since often they resist a precise complexity theoretic classification. Most famous of course is graph isomorphism, a candidate for an “intermediate problem” between P and the NP-complete problems. Here we obtain a dichotomy distinguishing the easy from the hard cases for isomorphism of circuits, but for the hard problems we only have a hardness result, we are not able to prove completeness for a complexity class.

A second group of problems we study concerns so called *frozen variables*. A variable  $x$  is frozen in a Boolean circuit  $C$  if  $C$  is satisfiable and all its satisfying assignments give the same Boolean value to  $x$ . We study the problem to determine if a given circuit has a variable

that is frozen. We also consider a variant that has become known recently under the name *audit problem*: this is the problem to decide if a given circuit has a frozen variable or is unsatisfiable. Originally the audit problem stems from the database area. One can view the value of a frozen variable as having been compromised by the results of the query expressed by the circuit. This is considered problematic with respect to data security questions (see [KPR03]). The audit problem has further practical importance also in VLSI design and testing: here, a frozen variable is a hint for a stuck-at fault and hence a manufacturing defect within the circuit.

Finally, we study a variant of the counting problem that is also relevant in practice: Instead of just determining the number of satisfying assignments we are interested in an efficient way of producing (*enumerating*) all such assignments. Different notions of “efficient” enumeration have been considered in a paper by Johnson et al. [JYP88]. We recall these notions here (e. g., polynomial total time, polynomial delay) and study them in the context of enumerating solutions of Boolean circuits.

For all these problems we obtain complete complexity classifications: We determine exactly those circuit bases that make the problems hard (NP-complete or even harder) and for all remaining bases we present efficient algorithms solving these problems.

The organization of the paper is as follows: In the next section we define Boolean functions and Boolean circuits. We also introduce Post’s lattice of all closed classes of Boolean functions; this lattice will be our main technical tool to obtain the desired complexity results. In Sect. 3 we formally introduce all algorithmic problems that we will classify. In Sect. 4 we then turn to equivalence and isomorphism while in Sect. 5 we study all audit-like problems; Sect. 6 contains our results on enumeration. Finally, Sect. 7 contains a conclusion and presents some open problems and future research directions.

## 2. PRELIMINARIES

**2.1. Boolean functions and Post’s lattice.** A *Boolean function* is an  $n$ -ary function  $f : \{0, 1\}^n \rightarrow \{0, 1\}$ . In the following we will often use well-known Boolean functions as 0, 1,  $\wedge$ ,  $\vee$ ,  $\neg$ ,  $\oplus$ ,  $\rightarrow$ , the implication function, and the  $(k + 1)$ -ary  $k$ -threshold function  $t_k$  verifying  $t_k(x_1, \dots, x_{k+1}) = 1$  if and only if  $\sum_{i=1}^{k+1} x_i \geq k$ .

A *clone* is a set of Boolean functions that is closed under superposition, *i.e.*, it contains all projections (that is, the functions  $f(a_1, \dots, a_n) = a_k$  for  $1 \leq k \leq n$  and  $n \in \mathbb{N}$ ) and is closed under arbitrary composition [PK79, Sze86, Pip97, Lau06]. Let  $B$  be a finite set of Boolean functions. We denote by  $[B]$  the smallest clone containing  $B$  and call  $B$  a *base* for  $[B]$ . The set  $[B]$  corresponds to the set of all Boolean functions that can be computed by  $B$ -circuits (as defined below). All closed classes of Boolean functions are known, as is their inclusion structure, which forms a lattice. This lattice is named after its discoverer E. Post [Pos41].

The following properties are crucial for the below definitions of the clones:

- $f$  is  *$c$ -reproducing* if  $f(c, \dots, c) = c$ ,  $c \in \{0, 1\}$ . The functions  $\wedge$  and  $\vee$  are 0- and 1-reproducing, the binary exclusive or,  $\oplus$ , is 0-reproducing, but not 1-reproducing, whereas the unary negation ( $\neg$ ) is neither 1- nor 0-reproducing.
- $f$  is *monotonic* if  $a_1 \leq b_1, \dots, a_n \leq b_n$  implies  $f(a_1, \dots, a_n) \leq f(b_1, \dots, b_n)$ . Boolean functions built up on composition of only  $\wedge, \vee, 0, 1$  are monotonic, like for instance  $g(x, y, z) \equiv x \wedge (1 \wedge (y \vee z))$ .

- $f$  is  $c$ -separating of degree  $k$  if for all  $A \subseteq f^{-1}(c)$  of size  $|A| \leq k$  there exists an  $i \in \{1, \dots, n\}$  such that  $(a_1, \dots, a_n) \in A$  implies  $a_i = c$ ,  $c \in \{0, 1\}$ . The  $(k+1)$ -ary  $k$ -threshold function  $t_k$  is 1-separating of degree  $k$ , but not 1-separating of degree  $k+1$ . For instance  $t_2(x, y, z) \equiv (x \wedge y) \vee (x \wedge z) \vee (y \wedge z)$ , which is the ternary majority function, is 1-separating of degree 2.
- $f$  is  $c$ -separating if  $f$  is  $c$ -separating of degree  $|f^{-1}(c)|$ . The implication  $(x \rightarrow y) \equiv \neg x \vee y$  is 0-separating.
- $f$  is *self-dual* if  $f(x_1, \dots, x_n) \equiv \neg f(\neg x_1, \dots, \neg x_n)$ . The function  $g(x, y, z) \equiv (x \wedge \neg y) \vee (x \wedge \neg z) \vee (\neg y \wedge \neg z)$  is self-dual.
- $f$  is *affine* if  $f \equiv x_1 \oplus \dots \oplus x_n \oplus c$  with  $c \in \{0, 1\}$ . The function  $g(x, y, z) \equiv x \oplus y \oplus z \oplus 1$  is affine and self-dual.

For a list of all Boolean clones see Table 1 and for their inclusion structure see Figure 2. For an extensive introduction to superposition, Post's Lattice and related problems see [BCRV03]. In the naming of the clones the semantic of single indexes is as follows. Index 2 indicates that the clone contains no constants at all. Index 0 (resp. 1) indicates that the clone contains only the constant 0 (resp. 1) but not 1 (resp. 0). Clones with no index contain both constants 0 and 1. The only exceptions to this convention are the clones  $D$  and  $D_1$  which do not contain any constants at all. The index  $*$  stands for all valid indexes. Clones of particular importance in this paper are:

- the clone of all Boolean functions  $BF = [\wedge, \neg] = [\wedge, \vee, \neg, 0, 1]$
- the monotonic clones  $M_*$ , e.g.,  $M_2 = [\wedge, \vee]$ ,  $M = [\wedge, \vee, 0, 1]$
- the affine clones  $L_*$ , e.g.,  $L_2 = [x \oplus y \oplus z]$ ,  $L = [x \oplus y, 0, 1]$
- the disjunctive clones  $V_*$ , e.g.,  $V_2 = [\vee]$ ,  $V = [\vee, 0, 1]$
- the conjunctive clones  $E_*$ , e.g.,  $E_2 = [\wedge]$ ,  $E = [\wedge, 0, 1]$
- the  $c$ -reproducing clones  $R_1$  (the clone of all 1-reproducing functions),  $R_0$  (0-reproducing functions),  $R_2$  (functions that are both 1- and 0-reproducing)
- the implication clone  $S_0 = [\rightarrow]$
- the negated-implication clone  $S_1 = [x \wedge \neg y]$
- the self-dual clones:  $D$  self-dual,  $D_1 = D \cap R_2$ ,  $D_2 = D \cap M$
- the clones  $S_{00} = S_0 \cap R_2 \cap M = [x \vee (y \wedge z)]$ ,  $S_{10} = S_1 \cap R_2 \cap M = [x \wedge (y \vee z)]$ ,  $S_{12} = S_1 \cap R_2 = [x \wedge (y \vee \neg z)]$  and  $S_{02} = S_0 \cap R_2 = [x \vee (y \wedge \neg z)]$ .
- the clones  $I_*$  containing only the identity and some constant functions, e.g.,  $I_0 = [id, 0]$

In the following we will often implicitly refer to the inclusion structure of Post's lattice. Here are some facts that we will use.

- The function  $x \oplus y \oplus z$  is a function of  $D_1$  since it is in  $L_2$  (see the base given in Table 1) and  $L_2 \subset D_1$ .
- Similarly the ternary majority function  $t_2(x, y, z) \equiv (x \wedge y) \vee (x \wedge z) \vee (y \wedge z)$  is a function of  $D_1$  since it is in  $D_2$  and  $D_2 \subset D_1$ .
- For all  $B$  such that  $S_{12} \subset [B] \subseteq R_1$  there exists a  $k \geq 2$  such that the threshold function  $t_k \in [B]$ . Indeed in this case  $[B]$  is either  $S_{12}^k$  for some  $k \geq 2$ , or  $R_1$  or  $R_2$  (which both contain  $S_{12}^2$ ).

We will often add some constant  $c = 0$  or  $1$  to a clone  $C$  and consider the clone  $C' = [C \cup \{c\}]$  generated out of  $C$  and  $c$ . With Post's lattice one can determine this  $C'$  quite easily: It is

the lowest clone above  $C$  that contains  $c$ , i.e., the lowest clone above both  $C$  and  $I_c$ . As a consequence a base of  $C'$  is obtained by a base of  $C$  to which we add the constant  $c$ . The following list contains identities we will frequently use.

- $\text{BF} = [\text{S}_1 \cup \{1\}]$ , thus  $\{x \wedge \neg y, 1\}$  is a base of  $\text{BF}$ .
- $\text{S}_1 = [\text{S}_{12} \cup \{0\}]$ , thus  $\{x \wedge (y \vee \neg z), 0\}$  is a base of  $\text{S}_1$ .
- $\text{R}_1 = [\text{S}_{12} \cup \{1\}]$ , thus  $\{x \wedge (y \vee \neg z), 1\}$  is a base of  $\text{R}_1$ .
- $\text{R}_0 = [\text{S}_{02} \cup \{0\}]$ , thus  $\{x \vee (y \wedge \neg z), 0\}$  is a base of  $\text{R}_0$ .

**2.2. Boolean circuits.** Let us now define the central objects that we deal with in this paper, namely *Boolean circuits* (see also [Vol99]):

**Definition 2.1.** Let  $B$  be a finite set of Boolean functions. A *Boolean circuit* over  $B$ , or a *B-circuit* is a tuple

$$C = (V, E, \alpha, \beta, o),$$

where  $(V, E)$  is a finite, acyclic, directed graph,  $\alpha: E \rightarrow \mathbb{N}$  is an injective function,  $\beta: V \rightarrow B \cup \{x_i \mid i \in \mathbb{N}\}$ , and  $o \in V$ , such that the following conditions hold:

- If  $v \in V$  has in-degree 0, then  $\beta(v) \in \{x_i \mid i \in \mathbb{N}\}$ , or  $\beta(v)$  is a 0-ary function from  $B$ ,
- if  $v \in V$  has in-degree  $k > 0$ , then  $\beta(v)$  is a  $k$ -ary function in  $B$ .

Nodes in  $V$  are also called *gates*. A gate  $v$  with  $\beta(v) \in \{x_i \mid i \in \mathbb{N}\}$  is called an *input-gate*, and  $o$  is called *output-gate*. Later the function  $\alpha$  will be used to specify the order of the predecessors of a gate.

With  $\text{Var}(C)$  we denote the variables appearing in the circuit  $C$ , i.e., the set  $\{\beta(v) \mid v \in V\} \cap \{x_i \mid i \in \mathbb{N}\}$ .

This definition of a Boolean circuit corresponds to the intuitive idea that a circuit consists of a set of gates which are either input gates, or compute some Boolean function (in our case, functions from  $B$ ) with arguments taken from the predecessor gates. The set  $B$  is also called a *base*. The distinguished gate  $o$  is the output-gate, i.e., the value computed by the circuit is the result computed in this gate. The *size* of a circuit is the number of non-input gates. The function computed by a circuit is defined in the canonical way: Once we know the values for the input-gates, we can inductively (since the graph is acyclic) compute the value for each gate  $g \in V$ . For non-commutative functions in  $B$ , the ordering  $\alpha$  on the edges in the graph gives a well-defined function value. The following definition captures this formally:

**Definition 2.2.** Let  $C = (V, E, \alpha, \beta, o)$  be a Boolean circuit with  $\text{Var}(C) = \{x_1, \dots, x_n\}$ , and let  $a_1, \dots, a_n \in \{0, 1\}$ . Let  $v$  be a gate in  $C$ . We define the function  $f_v$  computed by the gate  $v$  on input  $(a_1, \dots, a_n)$  as follows:

- If  $v$  is an input-gate, i.e.,  $\beta(v) = x_i$  for  $i \in \{1, \dots, n\}$ , we define  $f_v(a_1, \dots, a_n) = a_i$ .
- If  $v$  has in-degree  $k$ , and  $v_1, \dots, v_k$  are the predecessor gates of  $v$  in  $C$  such that  $\alpha((v_1, v)) < \dots < \alpha((v_k, v))$ , then

$$f_v(a_1, \dots, a_n) = \beta(v)(f_{v_1}(a_1, \dots, a_n), \dots, f_{v_k}(a_1, \dots, a_n)).$$

We define the function  $f_C: \{0, 1\}^n \rightarrow \{0, 1\}$ , the *function computed by  $C$* , as  $f_o$ .

| Class                        | Definition  | Base(s)  |
|------------------------------|---|--|
| BF                           | all Boolean functions   | $\{\wedge, \neg\}$   |
| R <sub>0</sub>               | $\{f \in \text{BF} \mid f \text{ is 0-reproducing}\}$                       | $\{\wedge, \oplus\}$   |
| R <sub>1</sub>               | $\{f \in \text{BF} \mid f \text{ is 1-reproducing}\}$                       | $\{\vee, x \oplus y \oplus 1\}$  |
| R <sub>2</sub>               | $R_1 \cap R_0$  | $\{\vee, x \wedge (y \oplus z \oplus 1)\}$                                     |
| M                            | $\{f \in \text{BF} \mid f \text{ is monotonic}\}$                           | $\{\wedge, \vee, 0, 1\}$   |
| M <sub>1</sub>               | $M \cap R_1$  | $\{\wedge, \vee, 1\}$  |
| M <sub>0</sub>               | $M \cap R_0$  | $\{\wedge, \vee, 0\}$  |
| M <sub>2</sub>               | $M \cap R_2$  | $\{\wedge, \vee\}$   |
| S <sub>0</sub> <sup>n</sup>  | $\{f \in \text{BF} \mid f \text{ is 0-separating of degree } n\}$           | $\{\rightarrow, \text{dual}(t_n)\}$  |
| S <sub>0</sub>               | $\{f \in \text{BF} \mid f \text{ is 0-separating}\}$                        | $\{\rightarrow\}$  |
| S <sub>1</sub> <sup>n</sup>  | $\{f \in \text{BF} \mid f \text{ is 1-separating of degree } n\}$           | $\{x \wedge \bar{y}, t_n\}$  |
| S <sub>1</sub>               | $\{f \in \text{BF} \mid f \text{ is 1-separating}\}$                        | $\{x \wedge \bar{y}\}$   |
| S <sub>02</sub> <sup>n</sup> | $S_0^n \cap R_2$  | $\{x \vee (y \wedge \bar{z}), \text{dual}(t_n)\}$                              |
| S <sub>02</sub>              | $S_0 \cap R_2$  | $\{x \vee (y \wedge \bar{z})\}$  |
| S <sub>01</sub> <sup>n</sup> | $S_0^n \cap M$  | $\{\text{dual}(t_n), 1\}$  |
| S <sub>01</sub>              | $S_0 \cap M$  | $\{x \vee (y \wedge z), 1\}$   |
| S <sub>00</sub> <sup>n</sup> | $S_0^n \cap R_2 \cap M$   | $\{x \vee (y \wedge z), \text{dual}(t_n)\}$                                    |
| S <sub>00</sub>              | $S_0 \cap R_2 \cap M$   | $\{x \vee (y \wedge z)\}$  |
| S <sub>12</sub> <sup>n</sup> | $S_1^n \cap R_2$  | $\{x \wedge (y \vee \bar{z}), t_n\}$   |
| S <sub>12</sub>              | $S_1 \cap R_2$  | $\{x \wedge (y \vee \bar{z})\}$  |
| S <sub>11</sub> <sup>n</sup> | $S_1^n \cap M$  | $\{t_n, 0\}$   |
| S <sub>11</sub>              | $S_1 \cap M$  | $\{x \wedge (y \vee z), 0\}$   |
| S <sub>10</sub> <sup>n</sup> | $S_1^n \cap R_2 \cap M$   | $\{x \wedge (y \vee z), t_n\}$   |
| S <sub>10</sub>              | $S_1 \cap R_2 \cap M$   | $\{x \wedge (y \vee z)\}$  |
| D                            | $\{f \mid f \text{ is self-dual}\}$   | $\{(x \wedge \bar{y}) \vee (x \wedge \bar{z}) \vee (\bar{y} \wedge \bar{z})\}$ |
| D <sub>1</sub>               | $D \cap R_2$  | $\{(x \wedge y) \vee (x \wedge \bar{z}) \vee (y \wedge \bar{z})\}$             |
| D <sub>2</sub>               | $D \cap M$  | $\{(x \wedge y) \vee (y \wedge z) \vee (x \wedge z)\}$                         |
| L                            | $\{f \mid f \text{ is linear}\}$  | $\{\oplus, 1\}$  |
| L <sub>0</sub>               | $L \cap R_0$  | $\{\oplus\}$   |
| L <sub>1</sub>               | $L \cap R_1$  | $\{\leftrightarrow\}$  |
| L <sub>2</sub>               | $L \cap R_2$  | $\{x \oplus y \oplus z\}$  |
| L <sub>3</sub>               | $L \cap D$  | $\{x \oplus y \oplus z \oplus 1\}$   |
| V                            | $\{f \mid f \text{ is an } \vee\text{-function or a constant function}\}$   | $\{\vee, 0, 1\}$   |
| V <sub>0</sub>               | $[\{\vee\}] \cup [\{0\}]$   | $\{\vee, 0\}$  |
| V <sub>1</sub>               | $[\{\vee\}] \cup [\{1\}]$   | $\{\vee, 1\}$  |
| V <sub>2</sub>               | $[\{\vee\}]$  | $\{\vee\}$   |
| E                            | $\{f \mid f \text{ is an } \wedge\text{-function or a constant function}\}$ | $\{\wedge, 0, 1\}$   |
| E <sub>0</sub>               | $[\{\wedge\}] \cup [\{0\}]$   | $\{\wedge, 0\}$  |
| E <sub>1</sub>               | $[\{\wedge\}] \cup [\{1\}]$   | $\{\wedge, 1\}$  |
| E <sub>2</sub>               | $[\{\wedge\}]$  | $\{\wedge\}$   |
| N                            | $[\{\neg\}] \cup [\{0\}] \cup [\{1\}]$                                      | $\{\neg, 1\}, \{\neg, 0\}$   |
| N <sub>2</sub>               | $[\{\neg\}]$  | $\{\neg\}$   |
| I                            | $[\{id\}] \cup [\{1\}] \cup [\{0\}]$  | $\{id, 0, 1\}$   |
| I <sub>0</sub>               | $[\{id\}] \cup [\{0\}]$   | $\{id, 0\}$  |
| I <sub>1</sub>               | $[\{id\}] \cup [\{1\}]$   | $\{id, 1\}$  |
| I <sub>2</sub>               | $[\{id\}]$  | $\{id\}$   |

Figure 1: The list of all Boolean clones with definitions and bases, where  $t_n := \bigvee_{i=1}^{n+1} \bigwedge_{j=1, j \neq i}^{n+1} x_j$  and  $\text{dual}(f)(a_1, \dots, a_n) = \neg f(\neg a_1, \dots, \neg a_n)$ .

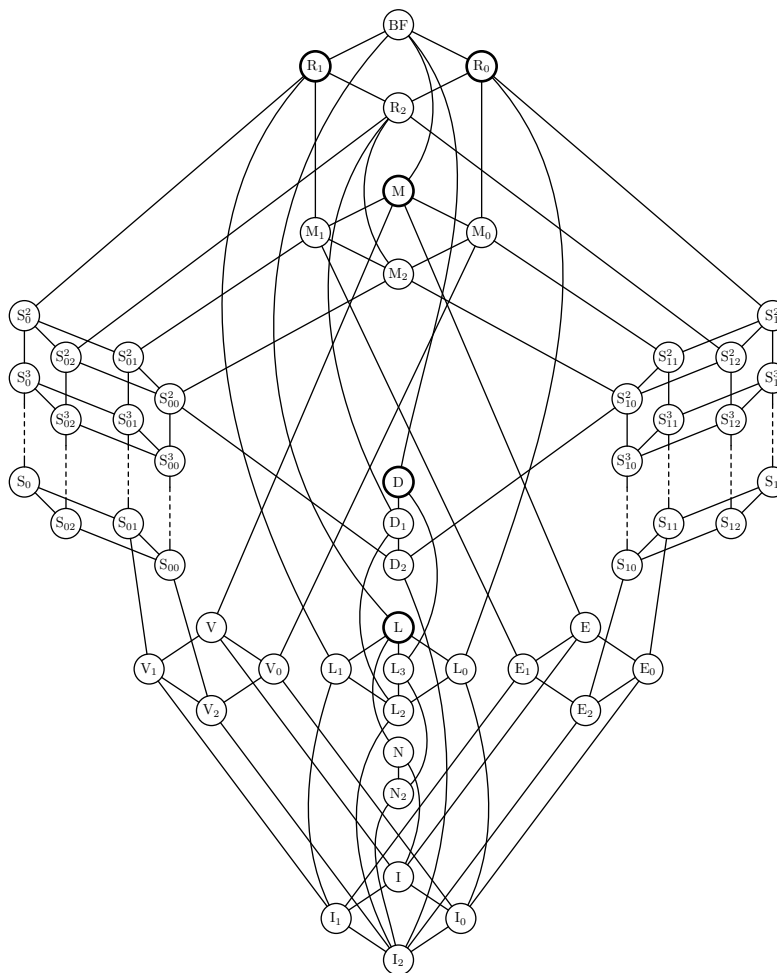


Figure 2: Lattice of all Boolean clones

If the function  $f_C$  associated with a circuit  $C$  does not depend on its  $i$ th argument, we say that  $x_i$  is a *fictive* or *irrelevant* variable for the circuit.

By writing  $C(x_1, \dots, x_n)$ , we mean that  $C$  is a circuit such that  $\text{Var}(C) \subseteq \{x_1, \dots, x_n\}$ . For constant values  $a_1, \dots, a_n \in \{0, 1\}$ , we also denote  $f_C(a_1, \dots, a_n)$  by  $C(a_1, \dots, a_n)$ . An *assignment* for the variables in  $C(x_1, \dots, x_n)$  is a function  $I: \{x_1, \dots, x_n\} \rightarrow \{0, 1\}$ . Such an assignment is also called *compatible with  $C$* . We will write  $I \models C$  if  $C(I(x_1), \dots, I(x_n)) = 1$ . In this case we also say that  $I$  is a *satisfying assignment* or a *solution* for  $C$ . We denote by  $\text{Sat}(C)$  the set of assignments satisfying  $C$  and by  $\#\text{Sat}(C)$  the cardinality of this set. A circuit is *satisfiable* if it has a satisfying assignment. When the order of variables is clear from the context, we write an assignment simply as the tuple of binary values, i.e., with  $(a_1, \dots, a_n)$  we denote the corresponding assignment  $I$  where  $I(x_i) = a_i$  for all relevant  $i$ . For convenience we also often write  $C(a_1 \dots a_n)$  when we mean  $C(a_1, \dots, a_n)$ . Hence  $C(1^n)$  denotes the value  $C(1, \dots, 1)$ . We sometimes view the circuit as a function of its assignments, and write  $C(I) = 1$  if  $I \models C$ , and  $C(I) = 0$  otherwise. In the following, let  $n \in \mathbb{N}$ , let  $f$  be an  $n$ -ary Boolean function,  $a \in \{0, 1\}$ , and  $I_1, I_2$  be

functions  $I_1, I_2: \{x_1, \dots, x_n\} \rightarrow \{0, 1\}$ . We define  $\#_a(I) = \#\{i : 1 \leq i \leq n \text{ and } I(x_i) = a\}$ . For assignments  $I_1$  and  $I_2$ , we write  $I_1 \leq I_2$  if  $I_1(x_i) \leq I_2(x_i)$  for  $1 \leq i \leq n$ . Finally let  $\text{dual}(I)$  be the assignment  $\text{dual}(I)(x_i) = 1 - I(x_i)$ .

For a circuit  $C$  and a variable  $x \in \text{Var}(C)$ , the variable  $x$  is said to be *frozen* in  $C$  if  $C$  is satisfiable and there is a constant  $c \in \{0, 1\}$  such that for all assignments  $I$ ,  $I \models C$  implies  $I(x) = c$ . Similarly we define that  $V \subseteq \text{Var}(C)$  is *frozen* in  $C$  if every variable in  $V$  is frozen in  $C$ .

### 3. PROBLEMS FOR PROPOSITIONAL CIRCUITS AND COMPLEXITY CLASSES

We now define the list of computational problems involving Boolean circuits that we study in this paper. All our problems are connected to the satisfiability problem and the circuit value problem defined as follows—in the following, let  $B$  be a base, i.e., a finite set of Boolean functions.

*Problem:*  $\text{SAT}_C B$

*Instance:* A  $B$ -circuit  $C(x_1, \dots, x_n)$

*Question:* Is  $C$  satisfiable?

*Problem:*  $\text{VAL}_C(B)$

*Instance:* A  $B$ -circuit  $C(x_1, \dots, x_n)$  and an assignment  $(a_1, \dots, a_n)$

*Question:* Is  $f_C(a_1, \dots, a_n) = 1$ ?

The complexity of these problems is well known:

**Proposition 3.1.** ([Lew79]) *Let  $B$  be a finite set of Boolean functions. Then  $\text{SAT}_C(B)$  is NP-complete if  $S_1 \subseteq [B]$ , and solvable in P otherwise.*

**Proposition 3.2** ([Lad75],[RW05]). *Let  $B$  be a finite set of Boolean functions, then  $\text{VAL}_C(B) \in \text{P}$ .*

We will be interested in equivalence and isomorphism problems. Let us define precisely these two notions.

**Definition 3.3.** Let  $\pi: \{x_1, \dots, x_n\} \rightarrow \{x_1, \dots, x_n\}$  be a permutation and  $I: \{x_1, \dots, x_n\} \rightarrow \{0, 1\}$  be a truth assignment. We define the permuted assignment  $\pi(I)$  by  $\pi(I)(x_i) = I(\pi(x_i))$  for  $i = 1, \dots, n$ .

**Definition 3.4.** Let  $C_1(x_1, \dots, x_n)$  and  $C_2(x_1, \dots, x_n)$  be  $B$ -circuits.

The two circuits are equivalent, denoted by  $C_1 \equiv C_2$ , if for all truth assignments  $I: \{x_1, \dots, x_n\} \rightarrow \{0, 1\}$ ,  $I \models C_1$  if and only if  $I \models C_2$ .

The two circuits are isomorphic, denoted by  $C_1 \cong C_2$  if there exists a permutation  $\pi: \{x_1, \dots, x_n\} \rightarrow \{x_1, \dots, x_n\}$  such that for all truth assignments  $I: \{x_1, \dots, x_n\} \rightarrow \{0, 1\}$ ,  $I \models C_1$  if and only if  $\pi(I) \models C_2$ .

Using these equivalence relations, we define the *Boolean equivalence* and *Boolean isomorphism* problem for  $B$ -circuits:

*Problem:*  $\text{EQ}_C(B)$

*Instance:* Two  $B$ -circuits  $C_1$  and  $C_2$

*Question:* Is  $C_1 \equiv C_2$ ?



The equivalence problem for propositional circuits or formulas is one of the standard coNP-complete problems. The complexity of the next problem we consider, the isomorphism problem, is not completely determined. It is clearly coNP-hard and lies in the second level of the polynomial hierarchy, more precisely in  $\Sigma_2^P$ . However, it is not known to be solvable in coNP, and is not complete for  $\Sigma_2^P$ , unless the polynomial hierarchy collapses [AT00]. We study the version of this problem where the inputs are restricted to  $B$ -circuits:

*Problem:*  $\text{ISO}_C(B)$   
*Instance:* Two  $B$ -circuits  $C_1$  and  $C_2$   
*Question:* Is  $C_1 \cong C_2$ ?

The next two problems are concerned with frozen variables. As defined earlier a variable  $x$  is frozen in a satisfiable circuit if all solutions of the circuit assign  $x$  the same Boolean value. The problem of recognizing frozen variables in Boolean formulas was first studied by Jon Kleinberg, Christos Papadimitriou, and Prabhakar Raghavan in [KPR03]; their motivation to consider this problem was to ensure that database queries do not reveal information that should be kept secret. Again, we consider the version of two problems in this context where we restrict the propositional gates allowed to appear in the input circuits:

*Problem:*  $\text{FV}_C(B)$   
*Instance:* A  $B$ -circuit  $C$  over a set of variables  $V$  and  $V' \subseteq V$  such that  $|V'| \geq 1$   
*Question:* Is  $V'$  frozen in  $C$ ?

If we restrict the problem  $\text{FV}_C(B)$  to instances with  $V' = V$ , then we obtain the generalized *Unique Satisfiability problem* over circuits. This is a natural complete problem for the class US (see [BG82]). We define the problem  $\text{Unique SAT}_C(B)$  to be the restriction of this problem to  $B$ -circuits as input.

*Problem:*  $\text{Unique SAT}_C(B)$   
*Instance:* A  $B$ -circuit,  $C$   
*Question:* Does  $C$  have exactly one satisfying assignment?

The question of the existence of such a frozen variable is the following problem

*Problem:*  $\exists \text{FV}_C(B)$   
*Instance:* A  $B$ -circuit  $C$   
*Question:* Is there a frozen variable in  $C$ ?

Note that in the above problem  $\exists \text{FV}_C(B)$  it is necessary that the circuit is satisfiable. If we drop the restriction of being satisfiable, we have the definition of the so called *Audit problem*: Does  $C$  have a frozen variable or is  $C$  unsatisfiable?

Besides these decision problems we are also interested in the enumeration problem which asks, for a given Boolean circuit to generate the set of its satisfying assignments with no repetition.

*Problem:*  $\text{Enum SAT}_C(B)$   
*Input:* A  $B$ -circuit,  $C$   
*Output:* All satisfying assignments of  $C$

In the following in establishing the complexity of the decision problems defined above we need notions of the following complexity classes: Let  $\mathcal{P}$  (NP resp.) be the class of languages which are decidable (acceptable, resp.) by deterministic (nondeterministic, resp.) Turing machines in polynomial time. For an arbitrary complexity class  $\mathcal{K}$ , let  $\text{co}\mathcal{K} = \{\overline{A} : A \in \mathcal{K}\}$ .

Recall that  $D^P = \{L \cap L' : L \in \text{NP}, L' \in \text{coNP}\}$ , which is the second level of the Boolean hierarchy and contains both NP and coNP (see [CGH<sup>+</sup>88, CGH<sup>+</sup>89]).

For our hardness results we mostly employ *logspace many-one reductions*, defined as follows: A language  $A$  is logspace many-one reducible to some language  $B$  (written  $A \leq_m^{\log} B$ ) if there exists a logspace-computable function  $f$  such that  $x \in A$  if and only if  $f(x) \in B$ . We write  $A \equiv_m^{\log} B$  if  $A \leq_m^{\log} B$  and  $B \leq_m^{\log} A$ . *Polynomial-time many-one reductions* (written as  $A \leq_m^p B$  and  $A \equiv_m^p B$ ) are defined in the same way, except that the function  $f$  is only required to be computable in polynomial time.

For the enumeration problem polynomial time is not a suitable notion of efficiency, since the number of solutions may be exponential in the length of the circuit. For the notion of an “efficient” enumeration algorithm, we use the definitions from [JPY88]. An algorithm for the enumeration problem has *polynomial total time*, if the running time of the algorithm is polynomial in the length of the input circuit and in the number of its satisfying solutions. This notion is also referred to as *output polynomial*. An important feature of an enumeration algorithm is the ability to start generating solutions as soon as possible, and more generally to generate solutions in a regular way with a limited delay between two successive outputs. It has *polynomial delay* if the time needed by the algorithm between its start and the printing of the first solution, the time between the printing of each two consecutive solutions, and the time between printing the last solution and the termination of the algorithm is bounded by a polynomial in the length of the input circuit. In [JPY88] the authors exhibited polynomial-delay algorithms that used exponential space and therefore distinguished polynomial-delay algorithms using only polynomial space. In our paper polynomial-delay enumeration algorithms all work with polynomial space, hence we do not mention it explicitly. An enumeration algorithm with polynomial delay can be further required to output the elements in some order (e.g. lexicographic order) (see [JPY88]).

Let us now make explicit the main tool that we will use in order to get complexity classifications.

**Proposition 3.5.** *Let  $\mathbf{Prob}$  be one of the decision problems introduced above, and let  $B_1, B_2$  be finite sets of Boolean functions such that  $B_1 \subseteq [B_2]$ . Then  $\mathbf{Prob}(B_1) \leq_m^{\log} \mathbf{Prob}(B_2)$ . In particular, if  $[B_1] = [B_2]$ , then  $\mathbf{Prob}(B_1) \equiv_m^{\log} \mathbf{Prob}(B_2)$ .*

*Proof.* Since all of the problems that we study in this paper only consider the function computed by the corresponding input circuits, it is clear that a transformation converting a circuit into an equivalent one leaves the properties considered in these decision problems invariant. Observe that if  $B_1$  and  $B_2$  are finite sets of Boolean functions such that  $B_1 \subseteq [B_2]$ , then every function from  $B_1$  can be expressed as a  $B_2$ -circuit, its so-called  $B_2$ -representation. Thus we can, in logarithmic space, convert any  $B_1$ -circuit into an equivalent  $B_2$ -circuit in replacing every gate of the original circuit (which is a function from  $B_1$ ) by its  $B_2$ -representation. This concludes the proof. Note that since  $B_1$  and  $B_2$  are not part of the input the cost of computing the  $B_2$ -representations of the functions of  $B_1$  is not taken into account.  $\square$

Since the above result shows that the complexity of the problems we study does not depend on the particular base of a clone that we consider, we sometimes write  $\mathbf{Prob}(C)$  for a clone  $C$ . For example, we write  $\text{SAT}_C(\text{BF})$  to denote the satisfiability for a set  $B$  with

$[B] = \text{BF}$ , e.g., for  $\text{SAT}_C(\{\wedge, \vee, \neg\})$ . Due to the above, choosing a different base  $B$  of BF results in a problem with the same complexity.

As a consequence in order to get a complete classification for  $\mathbf{Prob}(B)$  for every finite set  $B$  it is enough to examine all possible clones. When we show a hardness result for  $\mathbf{Prob}(C)$  for some clone  $C$ , then hardness also holds for every finite set  $B$  such that  $C \subseteq [B]$ . Also when we show tractability of  $\mathbf{Prob}(C)$ , then tractability also holds for every finite set  $B$  such that  $B \subseteq [C]$ .

We also note that a similar result as Proposition 3.5 applies to the enumeration problem. For example, if  $B_1 \subseteq [B_2]$ , and there is a polynomial-delay enumeration algorithm for  $B_2$ -circuits, then there also is a polynomial-delay enumeration algorithm for  $B_1$ -circuits.

#### 4. EQUIVALENCE- AND ISOMORPHISM PROBLEMS

In this section we use the inclusion structure of all closed classes (see Figure 2) to determine the complexity of  $\text{EQ}_C(B)$  step by step. Similarly we are able to give lower bounds for the isomorphism problem of  $B$ -circuits.

The following lemma will be a useful fact in our proofs—the lemma follows from the simple observation that for the equivalence- or isomorphism problems, consistently swapping 0s and 1s does not change the complexity. Let us first introduce some notation. If  $f$  is an  $n$ -ary Boolean function, then  $\text{dual}(f)$  denotes the Boolean function such that  $\text{dual}(f)(x_1, \dots, x_n) = \neg f(\neg x_1, \dots, \neg x_n)$ . For a set  $B$  of Boolean functions, let  $\text{dual}(B) = \{\text{dual}(f) \mid f \in B\}$ .

**Lemma 4.1.** *Let  $B$  be a finite set of Boolean functions. Then  $\text{EQ}_C(B) \equiv_m^{\log} \text{EQ}_C(\text{dual}(B))$  and  $\text{ISO}_C(B) \equiv_m^{\log} \text{ISO}_C(\text{dual}(B))$ .*

Let us first identify the tractable cases. The next proposition says that when besides constants only  $\vee$ -functions or only  $\wedge$ -functions or only  $\oplus$ -functions are allowed, equivalence and isomorphism are easily checkable.

**Proposition 4.2.** *Let  $B$  be a finite set of Boolean functions. If  $B \subseteq \text{E}$  or  $B \subseteq \text{V}$  or  $B \subseteq \text{L}$  then  $\text{EQ}_C(B)$  and  $\text{ISO}_C(B)$  are tractable.*

*Proof.* If  $B$  only contains to  $\vee$ -functions ( $\wedge$ -functions,  $\oplus$ -functions resp.), the basic idea is that we can first compute an explicit normal form for the functions computed by such a  $B$ -circuit. This normal form then easily allows to determine equivalence or isomorphism.

First let  $B \subseteq \text{V}$ . Let  $C_1(x_1, \dots, x_n)$  and  $C_2(x_1, \dots, x_n)$  be two  $B$ -circuits. The Boolean functions described by  $C_1$  and  $C_2$  can be expressed as follows:  $f_{C_1}(x_1, \dots, x_n) = a_0 \vee (a_1 \wedge x_1) \vee \dots \vee (a_n \wedge x_n)$  and  $f_{C_2}(x_1, \dots, x_n) = b_0 \vee (b_1 \wedge x_1) \vee \dots \vee (b_n \wedge x_n)$ , where  $a_1, \dots, a_n, b_1, \dots, b_n \in \{0, 1\}$ .

The values of  $a_i$  and  $b_i$ , where  $0 \leq i \leq n$ , can be determined easily by using the following simple facts:  $a_0 = 0$  ( $b_0 = 0$ , resp.) iff  $f_{C_1}(0^n) = 0$  ( $f_{C_2}(0^n) = 0$ , resp.) and  $a_i = 0$  ( $b_i = 0$ , resp.) for  $1 \leq i \leq n$  iff  $a_0 = 0$  ( $b_0 = 0$ , resp.) and  $f_{C_1}(0^{i-1}10^{n-i}) = 0$  ( $f_{C_2}(0^{i-1}10^{n-i}) = 0$ , resp.). This can be checked in polynomial time with the help of  $\text{VAL}_C(B)$  as an oracle. Since  $\text{VAL}_C(B)$  is tractable (see Proposition 3.2) we conclude that the normal forms can be computed efficiently. Now, clearly  $(C_1, C_2) \in \text{EQ}_C(B)$  iff either  $a_0 = b_0 = 1$  or  $a_0 = b_0 = 0$  and  $a_i = b_i$  for  $1 \leq i \leq n$ , and similarly,  $(C_1, C_2) \in \text{ISO}_C(B)$  iff either  $a_0 = b_0 = 1$  or  $|\{i \mid a_i = 1, 1 \leq i \leq n\}| = |\{i \mid b_i = 1, 1 \leq i \leq n\}|$  and  $a_0 = b_0 = 0$ . Thus we conclude that  $\text{EQ}_C(B)$  and  $\text{ISO}_C(B)$  are tractable.

Tractability for  $B \subseteq E$  now follows immediately from the above using Lemma 4.1.

Finally let  $B \subseteq L$  and let  $C_1$  and  $C_2$  be  $B$ -circuits. The Boolean functions described by the  $B$ -circuits  $C_1(x_1, \dots, x_n)$  and  $C_2(x_1, \dots, x_n)$  can be expressed as follows:  $f_{C_1}(x_1, \dots, x_n) = a_0 \oplus (a_1 \wedge x_1) \oplus \dots \oplus (a_n \wedge x_n)$  and  $f_{C_2}(x_1, \dots, x_n) = b_0 \oplus (b_1 \wedge x_1) \oplus \dots \oplus (b_n \wedge x_n)$ , where  $a_1, \dots, a_n, b_1, \dots, b_n \in \{0, 1\}$ . Similar to the above cases the values  $a_i$  and  $b_i$  for  $0 \leq i \leq n$  can be determined by a  $\oplus$ -L-calculation, since we know that  $\text{VAL}_C(B)$  is tractable. In particular  $a_0 = f_{C_1}(0, \dots, 0)$ ,  $b_0 = f_{C_2}(0, \dots, 0)$ ,  $a_i = f_{C_1}(0^{i-1}10^{n-i}) \oplus a_0$  and  $b_i = f_{C_2}(0^{i-1}10^{n-i}) \oplus b_0$ , where  $1 \leq i \leq n$ . Now, clearly  $(C_1, C_2) \in \text{EQ}_C(B)$  iff  $a_i = b_i$  for  $0 \leq i \leq n$ , and  $(C_1, C_2) \in \text{ISO}_C(B)$  iff  $a_0 = b_0$  and  $|\{i \mid a_i = 1, 1 \leq i \leq n\}| = |\{i \mid b_i = 1, 1 \leq i \leq n\}|$ . Again, both problems are tractable.  $\square$

The main step in obtaining hardness for the remaining cases now is to show that both equivalence and isomorphism are hard for monotone functions. This is the statement of the next lemma.

**Lemma 4.3.**  $\text{EQ}_C(\{\vee, \wedge\})$  and  $\text{ISO}_C(\{\vee, \wedge\})$  are  $\leq_m^{\log}$ -hard for coNP.

*Proof.* We prove that 3-TAUT, the problem of deciding whether a 3-DNF formula is a tautology, is logspace reducible to  $\text{EQ}_C(\{\vee, \wedge\})$  and  $\text{ISO}_C(\{\vee, \wedge\})$ . Since 3-TAUT is well known to be coNP-hard this will complete the proof.

Let  $H(x_1, \dots, x_n)$  be a 3-DNF formula with  $\text{Var}(H) = \{x_1, \dots, x_n\}$ . Let  $C(x_1, \dots, x_n, y_1, \dots, y_n)$  be the circuit obtained from  $H$  in replacing every occurrence of a negated variable  $\neg x_i$  by the fresh variable  $y_i$ . Note that  $C$  is a  $\{\vee, \wedge\}$ -circuit that can have fictive variables. Define  $C_1(x_1, \dots, x_n, y_1, \dots, y_n) = \bigwedge_{i=1}^n (x_i \vee y_i)$  and  $C_2(x_1, \dots, x_n, y_1, \dots, y_n) = C_1 \wedge C$ . Observe that  $\text{Sat}(C_2) \subseteq \text{Sat}(C_1)$ .

We claim that  $H$  is a tautology if and only if  $C_1 \equiv C_2$  if and only if  $C_1 \cong C_2$ . Suppose first that  $H$  is a tautology. We prove that every assignment that sets at least one of  $x_i$  and  $y_i$  to true for every  $i = 1, \dots, n$  satisfies  $C$ , thus proving  $C_1 \equiv C_2$  and *a fortiori*  $C_1 \cong C_2$ . Let  $I$  be such an assignment. Consider the assignment  $\tilde{I}$  defined by  $\tilde{I}(x_i) = I(x_i)$  and  $\tilde{I}(y_i) = 1 - I(x_i)$  for  $i = 1, \dots, n$ . Observe that  $\tilde{I}$  satisfies  $C$  since  $H$  is a tautology. Moreover since for every  $i$ ,  $I(x_i) + I(y_i) \geq 1$  we have  $I \geq \tilde{I}$ . Therefore by monotonicity  $I$  satisfies  $C$  as well.

Conversely, suppose that  $H$  is not a tautology. We prove that  $\#\text{Sat}(C_1) \neq \#\text{Sat}(C_2)$ , thus proving  $C_1 \not\equiv C_2$  and *a fortiori*  $C_1 \not\cong C_2$ . Let  $I$  be an assignment that does not satisfy  $H$ . Consider  $I'$  defined by  $I'(x_i) = I(x_i)$  and  $I'(y_i) = 1 - I(x_i)$  for  $i = 1, \dots, n$ . Observe that  $I'$  satisfies  $C_1$  but not  $C_2$ . Since  $\text{Sat}(C_2) \subseteq \text{Sat}(C_1)$ , this proves that  $\#\text{Sat}(C_2) < \#\text{Sat}(C_1)$ .  $\square$

The next three propositions generalize the hardness result from Lemma 4.3.

**Proposition 4.4.** Let  $B$  be a set of Boolean functions such that  $S_{10} \subseteq [B]$  or  $S_{00} \subseteq [B]$ , then  $\text{EQ}_C(B)$  and  $\text{ISO}_C(B)$  are  $\leq_m^{\log}$ -hard for coNP.

*Proof.* First let  $S_{10} \subseteq [B]$ . By Figure 1 we know that  $g(x, y, z) = x \wedge (y \vee z)$  is a base of  $S_{10}$ . Since  $g(x, y, y) = x \wedge y$  and  $g(1, x, y) = x \vee y$  we know that  $\wedge \in [B]$  and  $\vee \in [B \cup \{1\}]$ . Therefore according to Proposition 3.5 and Lemma 4.3 we get that  $\text{EQ}_C(B \cup \{1\})$  and  $\text{ISO}_C(B \cup \{1\})$  are coNP-hard. We now reduce these problems respectively to  $\text{EQ}_C(B)$  and  $\text{ISO}_C(B)$ . Let  $C_1(x_1, \dots, x_n, 1)$  and  $C_2(x_1, \dots, x_n, 1)$  be two  $B \cup \{1\}$ -circuits. Let  $v$  be a fresh variable that will be used to replace the constant 1. Let  $C'_1(x_1, \dots, x_n, v) = C_1(x_1, \dots, x_n, v) \wedge v$  and  $C'_2(x_1, \dots, x_n, v) = C_2(x_1, \dots, x_n, v) \wedge v$ . Since  $\wedge \in [B]$ ,  $C'_1$  and  $C'_2$  can be represented as  $B$ -circuits, and their  $B$ -representation can be computed in logarithmic

space, see proof of Proposition 3.5. It is obvious that  $C_1 \equiv C_2$  if and only if  $C'_1 \equiv C'_2$ . If  $C_1 \cong C_2$  then clearly  $C'_1 \cong C'_2$ . Conversely if  $C'_1 \cong C'_2$  then there is a permutation  $\pi: \{x_1, \dots, x_n, v\} \rightarrow \{x_1, \dots, x_n, v\}$  such that for all assignment  $I$ ,  $I \models C'_1$  if and only if  $\pi(I) \models C'_2$ . Since the value of  $v$  is fixed to 1 in every satisfying assignment one can suppose w.l.o.g. that  $\pi(v) = v$ . In this case we clearly have that for all assignment  $I$ ,  $I \models C_1$  if and only if  $\pi(I) \models C_2$ , thus showing that  $C_1 \cong C_2$ .

Now let  $S_{00} \subseteq [B]$ . By inspecting Figure 2 we obtain that  $S_{10} \subseteq [B]$  as well, or  $S_{10} \subseteq \text{dual}([B])$ . According to Lemma 4.1 the proof is then completed.  $\square$

**Proposition 4.5.** *Let  $B$  be a finite set of Boolean functions such that  $D_2 \subseteq [B] \subseteq D$ , then  $\text{EQ}_C(B)$  is  $\leq_m^{\log}$ -hard for coNP.*

*Proof.* Due to Lemma 4.3, we know that  $\text{EQ}_C(\{\wedge, \vee\})$  is  $\leq_m^{\log}$ -hard for coNP. Hence let  $C_1$  and  $C_2$  be  $\{\wedge, \vee\}$ -circuits, where  $\text{Var}(C_1) \cup \text{Var}(C_2) = \{x_1, \dots, x_n\}$ . By Figure 1 we know that  $t_2(x, y, z) = (x \wedge y) \vee (x \wedge z) \vee (y \wedge z)$ , the ternary majority function, is a base for  $D_2$ . Using the equalities  $t_2(x, y, 0) = x \wedge y$  and  $t_2(x, y, 1) = x \vee y$ , we can transform  $C_1$  and  $C_2$  into equivalent  $\{t_2, 0, 1\}$ -circuits in logarithmic space. For ease of notation, we denote these (equivalent) circuits with  $C_1(x_1, \dots, x_n, 0, 1)$  and  $C_2(x_1, \dots, x_n, 0, 1)$  again. We note that due to the above transformation, every application of  $t_2$  in  $C_1$  or  $C_2$  has exactly one constant argument.

We now construct  $\{t_2\}$ -circuits  $C'_1$  and  $C'_2$  such that  $C_1 \equiv C_2$  if and only if  $C'_1 \equiv C'_2$ . Due to Proposition 3.5, this completes the proof.

Let  $u$  and  $v$  be fresh variables that will be used to replace the constants 0 and 1 that appear in  $C_1$  and  $C_2$ . Now define

$$C'_1(x_1, \dots, x_n, u, v) = t_2(v, C_1(x_1, \dots, x_n, u, v), u) \text{ and}$$

$$C'_2(x_1, \dots, x_n, u, v) = t_2(v, C_2(x_1, \dots, x_n, u, v), u).$$

By construction,  $C'_1$  and  $C'_2$  are  $\{t_2\}$ -circuits. We prove that  $C_1 \equiv C_2$  if and only if  $C'_1 \equiv C'_2$ .

First assume that  $C_1 \equiv C_2$ , and let  $I$  be an assignment for  $\{x_1, \dots, x_n, u, v\}$ . If  $I(u) = I(v)$ , then  $I \models C'_1$  iff  $I \models C'_2$  iff  $I(u) = I(v) = 1$ . Now consider the case that  $I(u) \neq I(v)$ . Since  $t_2$  is a self-dual function it is sufficient to consider the case  $I(u) = 0$  and  $I(v) = 1$ . In this case  $I \models C'_i$  iff  $I \models C_i$  for  $i = 1, 2$ , and hence  $I \models C'_1$  iff  $I \models C'_2$  since  $C_1 \equiv C_2$ .

Conversely, suppose that  $C_1 \not\equiv C_2$ . Then we can suppose that there exists an assignment  $I$  that satisfies  $C_1$  but not  $C_2$ . Extend  $I$  to  $I'$  by setting  $I'(u) = 0$  and  $I'(v) = 1$ . It is easy to see that  $I'$  satisfies  $C'_1$  but not  $C'_2$ , thus proving that  $C'_1 \not\equiv C'_2$ .  $\square$

We have a similar result for the isomorphism problem.

**Proposition 4.6.** *Let  $B$  be a finite set of Boolean functions such that  $D_2 \subseteq [B] \subseteq D$ , then  $\text{ISO}_C(B)$  is  $\leq_m^{\log}$ -hard for coNP.*

While the proof of this proposition uses essentially the same reduction as above, it is technically more involved and requires some technical results. We will reduce from the isomorphism problem for  $\{\wedge, \vee\}$ -circuits, which we know to be coNP-hard due to Lemma 4.3. However, in the proof of Proposition 4.6, we will need some special properties of the instances of  $\text{ISO}_C(\{\vee, \wedge\})$  that we reduce from. We therefore present a series of intermediate technical results that allow us to restrict the instances of  $\text{ISO}_C(\{\vee, \wedge\})$  as required.

First we need to introduce some new notion.

**Definition 4.7.** A pair of two variables  $\{s, t\}$  is *dominant* for a circuit  $C$ , if every truth assignment  $I$  with  $I(s) = I(t) = \alpha$  satisfies the circuit  $C$  if and only if  $\alpha = 1$ .

The following lemma gives some easy properties of dominant pairs. The proof of the lemma is straight-forward.

**Lemma 4.8.**

- (1) Let  $C$  be a circuit, and let  $\{s, t\}$  and  $\{s', t'\}$  be two dominant pairs for  $C$ . Then  $\{s, t\} \cap \{s', t'\} \neq \emptyset$ .
- (2) Let  $C_1$  and  $C_2$  be two circuits such that  $C_1 \cong C_2$  via a permutation  $\pi$ . If  $\{s, t\}$  is a dominant pair for  $C_1$ , then  $\{\pi(s), \pi(t)\}$  is a dominant pair for  $C_2$ .

*Proof.* For the first part, assume that  $\{s, t\} \cap \{s', t'\} = \emptyset$ . Then there is an assignment  $I$  with  $I(s) = I(t) = 0$ , and  $I(s') = I(t') = 1$ . Since  $\{s, t\}$  is dominant for  $C$ , it follows that  $I \not\models C$ . On the other hand, since  $\{s', t'\}$  is dominant for  $C$  as well, we know that  $I \models C$ , a contradiction. The second part is trivial.  $\square$

Next we need the following result, which says that the isomorphism problem remains hard for monotone functions even for some restricted instances.

**Lemma 4.9.**  $\text{ISO}_C(\{\vee, \wedge\})$  is  $\leq_m^{\log}$ -hard for coNP, even when instances are restricted to pairs of circuits  $(C_1, C_2)$  where neither  $C_1$  nor  $C_2$  implies or is implied by one variable, and further  $\#\text{Sat}(C_1) + \#\text{Sat}(C_2) < 2^n$  where  $n$  is the number of variables in  $C_1$ , which is the same as the number of variables in  $C_2$ .

*Proof.* We reduce the problem  $\text{ISO}_C(\{\vee, \wedge\})$ , which is coNP-hard due to Lemma 4.3, to the same problem with restrictions on the instances.

Let  $(C_1^0, C_2^0)$  be a pair of  $\{\vee, \wedge\}$ -circuits given as an instance of  $\text{ISO}_C(\{\vee, \wedge\})$ .

Without loss of generality one can suppose that they have the same number of variables,  $n$ , and that  $C_i^0$  is of the form  $C_i^{0'} \wedge (y \wedge z)$ . Thus  $C_i^0$  has fewer than  $2^{n-1}$  solutions. Since both  $C_i^0$  are monotone, we can, in polynomial time, verify whether  $C_1^0$  or  $C_2^0$  are constant. If one of them is, then  $C_1^0 \cong C_2^0$  is true if and only if they are equivalent to the same constant. Hence we assume that neither  $C_1^0$  nor  $C_2^0$  is constant. For  $i \in \{1, 2\}$ , we now rewrite  $C_i^0$  into

$$C_i = t_2(z_1, z_2, C_i^0),$$

where  $t_2$  is the ternary majority function.

We will show that the  $C_i$ 's have the desired properties, and that  $C_1^0 \cong C_2^0$  if and only if  $C_1 \cong C_2$ , thus concluding the proof.

Clearly, since the outmost operator of  $C_1$  and  $C_2$  is the majority function, and neither  $C_1^0$  nor  $C_2^0$  are constant, it follows that no variable implies or is implied by one of the  $C_i$ . Obviously,  $\#\text{Sat}(C_i) = 2 \cdot \#\text{Sat}(C_i^0) + 2^n$ , since the solutions of  $C_i$  are exactly those of  $C_i^0$  extended with  $z_1 \neq z_2$  (giving two solutions for each solution of  $C_i^0$ ), plus all  $2^n$  assignments setting  $z_1 = z_2 = 1$ . Since  $\#\text{Sat}(C_i^0) < 2^{n-1}$ , it follows that  $\#\text{Sat}(C_i) < 2^n + 2^n = 2^{n+1}$ . Therefore,  $\#\text{Sat}(C_1) + \#\text{Sat}(C_2) < 2^{n+2}$  as required (note that  $n+2$  is the number of variables appearing in  $C_1$  and  $C_2$ ).

It remains to show that  $C_1^0 \cong C_2^0$  if and only if  $C_1 \cong C_2$ . The left-to-right direction is trivial, by extending the permutation to be the identity on  $\{z_1, z_2\}$ . For the other direction, assume that  $C_1 \cong C_2$  via a permutation  $\pi$ . Observe that  $\{z_1, z_2\}$  is a dominant pair for both circuits. If  $\pi(\{z_1, z_2\}) = \{z_1, z_2\}$ , then since  $z_1$  and  $z_2$  are symmetric, we can assume that  $\pi(z_i) = z_i$ , and  $\pi$  restricted to the original variables establishes  $C_1^0 \cong C_2^0$ .

Hence assume  $\pi(\{z_1, z_2\}) \neq \{z_1, z_2\}$ . According to Lemma 4.8,  $\pi(\{z_1, z_2\}) \cap \{z_1, z_2\} \neq \emptyset$ . Since  $z_1$  and  $z_2$  are symmetric, we assume without loss of generality that  $\pi(\{z_1, z_2\}) = \{z_1, x\}$  for a variable  $x$  of  $C_2$ .

We prove that  $C_2^0$  is equivalent to  $x$ . First let  $I$  be an assignment to the variables in  $C_2^0$  with  $I(x) = 1$ , we prove that  $I \models C_2^0$ . For this, consider the assignment  $I^+$  which extends  $I$  by  $I^+(z_1) = 1$  and  $I^+(z_2) = 0$ . Since  $\{z_1, x\}$  is dominant, it follows that  $I^+ \models C_2$ , and thus  $I \models C_2^0$ . Therefore,  $x$  implies  $C_2^0$ . For the other direction, let  $I$  be an assignment with  $I(x) = 0$ , we show that  $I \not\models C_2^0$ . We extend  $I$  to  $I^+$  by setting  $I^+(z_1) = 0$  and  $I^+(z_2) = 1$ . Since  $\{z_1, x\}$  is dominant for  $C_2$ , it follows that  $I^+ \not\models C_2$ , hence we know that  $I \not\models C_2^0$ , and thus  $C_2^0$  is equivalent to  $x$  as claimed.

Therefore,  $C_2$  has exactly three relevant variables (recall that a variable  $x$  is *relevant* for a circuit  $C$ , if there are assignments  $I_1$  and  $I_2$  such that  $I_1(x') = I_2(x')$  for all variables  $x' \neq x$ , and  $I_1 \models C$  and  $I_2 \not\models C$ , i.e., if the value of the function computed by the circuit in fact depends on  $x$ ).

Since  $C_2 \cong C_1$ , we know that  $C_1$  also has exactly three relevant variables. Hence  $C_1^0$  has exactly one relevant variable, and since we also know that  $C_1^0$  is a monotone circuit, it follows that  $C_1^0$  is equivalent to a single variable. In particular,  $C_1^0 \cong C_2^0$  as claimed.  $\square$

We need a last technical result. This lemma allows us, in the later proof of Proposition 4.6, to use a similar argument as in the above proof of Lemma 4.9: In both proofs it is essential that we can control the possible dominant pairs of a circuit that is of the form  $t_2(x_1, x_2, C)$ , where  $t_2$  is the ternary majority function and  $C$  is some circuit. In the proof of Lemma 4.9, we knew the dominant sets of  $t_2(z_1, z_2, C_i^0)$  since  $z_1$  and  $z_2$  did not appear in  $C_i^0$ . In the proof of Proposition 4.6, the situation will be a bit more complicated, and we will need the following lemma to ensure that the dominant pairs in the circuits resulting from our reduction are exactly the ones that we need.

**Lemma 4.10.** *If  $C(x_1, \dots, x_n, 1, 0)$  is a circuit that does not imply a variable and is not implied by a variable, then  $\{u, v\}$  is the only dominant pair for*

$$C' = t_2(u, C_2(x_1, \dots, x_n, u, v), v),$$

where  $u$  and  $v$  are new variables.

We note that a circuit  $C$  implies a variable if and only if the function computed by  $C$  is 1-separating, and is implied by a variable if and only if the function computed by  $C$  is 0-separating. We note that for the ternary majority function  $t_2$ , all pairs of two distinct variables are dominant.

*Proof.* We use the following notation: For an assignment  $I$  for  $C$ , with  $I^+$  we denote the assignment  $I$  extended with  $I^+(u) = 1$  and  $I^+(v) = 0$ . By construction it follows that  $I \models C$  if and only if  $I^+ \models C'$ .

Clearly,  $\{u, v\}$  is dominant for  $C'$ . Let  $\{u', v'\} \neq \{u, v\}$  be dominant for  $C'$ . Due to Lemma 4.8, we know that  $\{u, v\} \cap \{u', v'\} \neq \emptyset$ .

First assume  $u \in \{u, v\} \cap \{u', v'\}$ , then  $\{u', v'\} = \{u, x\}$  for a variable  $x$  of  $C$ . We prove that  $x$  implies  $C$ . Hence let  $I(x) = 1$ . By construction, we have that  $I^+(u) = 1$ , and  $I^+(x) = I(x) = 1$ . Since  $\{u, x\}$  dominates  $C'$ , it follows that  $I^+ \models C'$ . Due to the above, this means that  $I \models C$ . Hence  $x$  implies  $C$ , a contradiction.

Similarly, assume that  $v \in \{u, v\} \cap \{u', v'\}$ , then  $\{u', v'\} = \{v, x\}$  for a variable  $x$  of  $C$ . We claim that  $C$  implies  $x$ . Hence let  $I$  be an assignment with  $I \models C$ , and assume that  $I(x) = 0$ . From the above it follows that  $I^+ \models C'$ . On the other hand, we have that

$I^+(x) = I^+(v) = 0$ , and since  $\{v, x\}$  dominates  $C'$ , this implies  $I^+ \not\models C'$ , a contradiction. Therefore,  $C$  indeed implies  $x$ , which is a contradiction to the prerequisites of the lemma.  $\square$

We are now in a position to prove Proposition 4.6.

*Proof.* As in the proof of Proposition 4.5 we get that  $\text{ISO}_C(B \cup \{0, 1\})$  is coNP-hard, and we reduce this problem to  $\text{ISO}_C(B)$ . Let  $C_1(x_1, \dots, x_n, 0, 1)$  and  $C_2(x_1, \dots, x_n, 0, 1)$  be two  $B \cup \{0, 1\}$ -circuits. According to Lemma 4.9 one can suppose that neither  $C_1$  nor  $C_2$  implies or is implied by one variable, and further that  $\#\text{Sat}(C_1) + \#\text{Sat}(C_2) < 2^n$  where  $n$  is the number of variables in  $C_1$ , which is the same as the number of variables in  $C_2$ .

Let  $u$  and  $v$  be fresh variables, and let

$$C'_1(x_1, \dots, x_n, u, v) = t_2(u, C_1(x_1, \dots, x_n, u, v), v)$$

and

$$C'_2(x_1, \dots, x_n, u, v) = t_2(u, C_2(x_1, \dots, x_n, u, v), v).$$

As mentioned in the earlier proof of Lemma 4.9, this construction is very similar to what we used there. The major difference lies in the role of the variables that are used in the application of the newly introduced majority function: In the proof of Lemma 4.9, we used new variables  $z_1$  and  $z_2$  that did not appear anywhere else, and whose role was symmetric. In fact, the proof of Lemma 4.9 only works since  $z_1$  and  $z_2$  did not appear in the circuits  $C_i^0$  considered in that proof.

In the current proof, the situation is different: Here, the variables  $u$  and  $v$  do appear in the circuits  $C_i(x_1, \dots, x_n, u, v)$ , and they are clearly not symmetric—they “simulate” the values 0 and 1, respectively. In the remainder of the current proof, we make crucial use of the facts established in Lemma 4.9, namely, that the circuits  $C_1$  and  $C_2$  are not implied by, or imply, a variable. This then allows us to apply Lemma 4.10 and ensure that  $\{u, v\}$  is the only dominant pair of  $C'_i$ .

Another difference is that in the current proof, the circuits  $C_i(x_1, \dots, x_n, u, v)$  are indeed  $D_2$ -circuits, where in the earlier result, the majority function was applied to (almost) arbitrary  $\{\wedge, \vee\}$ -circuits.

We prove  $C_1 \cong C_2$  if and only if  $C'_1 \cong C'_2$ . It is obvious that if  $C_1 \cong C_2$  then  $C'_1 \cong C'_2$ . Conversely, suppose that  $C'_1 \cong C'_2$ . Then there exists a permutation  $\pi: \{x_1, \dots, x_n, u, v\} \rightarrow \{x_1, \dots, x_n, u, v\}$  such that for every truth assignment  $I: \{x_1, \dots, x_n, u, v\} \rightarrow \{0, 1\}$  it holds that  $I \models C'_1$  if and only if  $\pi(I) \models C'_2$ . Observe that because of the majority function, the pair  $\{u, v\}$  is dominant for both circuits  $C'_1$  and  $C'_2$ . According to Lemma 4.10 we have then  $\{\pi(u), \pi(v)\} = \{u, v\}$ . Suppose that  $\pi(u) = v$  and  $\pi(v) = u$ . Let  $\#C'_1$  be the number of truth assignments satisfying  $C'_1$  that set  $u$  to 0 and  $v$  to 1, and  $\#C'_2$  be the number of truth assignments satisfying  $C'_2$  that set  $u$  to 1 and  $v$  to 0. Since  $C'_1$  and  $C'_2$  are isomorphic through a permutation  $\pi$  such that  $\pi(u) = v$  and  $\pi(v) = u$  we have  $\#C'_1 = \#C'_2$ . We have  $\#C'_1 = \#\{I \mid I(u) = 0, I(v) = 1 \text{ and } C_1(I(x_1), \dots, I(x_n), 0, 1) = 1\}$  and  $\#C'_2 = \#\{J \mid J(u) = 1, J(v) = 0 \text{ and } C_2(J(x_1), \dots, J(x_n), 1, 0) = 1\}$ . Observe that  $\#C'_1 = \#\text{Sat}(C_1)$ , while

$$\begin{aligned} \#C'_2 &= \#\{I: \{x_1, \dots, x_n\} \rightarrow \{0, 1\} \mid C_2(I(x_1), \dots, I(x_n), 1, 0) = 1\} \\ &= \#\{I: \{x_1, \dots, x_n\} \rightarrow \{0, 1\} \mid C_2(1 - I(x_1), \dots, 1 - I(x_n), 0, 1) = 0\}, \end{aligned}$$

since  $C_2$  is a  $B \cup \{0, 1\}$ -circuit and  $B$  contains only self-dual functions. Therefore  $\#C'_2 = 2^n - \#\text{Sat}(C_2)$ . But  $\#C'_1 = \#C'_2$  implies  $\#\text{Sat}(C_1) = 2^n - \#\text{Sat}(C_2)$ , i.e.,  $\#\text{Sat}(C_1) +$



$\#\text{Sat}(C_2) = 2^n$ , which is not the case by assumption, thus providing a contradiction. Therefore  $\pi(u) = u$  and  $\pi(v) = v$ . With this it is easy to see that  $C'_1 \cong C'_2$  implies that  $C_1 \cong C_2$  through the same permutation  $\pi$ .  $\square$

By a careful inspection of Figure 2 we see that Propositions 4.2, 4.4, 4.5, and 4.6 cover all cases. This leads us to the following classification theorems for the complexity of the equivalence- and isomorphism- problems of  $B$ -circuits:

**Theorem 4.11.** *Let  $B$  be a finite set of Boolean functions.*

- (1) *If  $B \subseteq E$  or  $B \subseteq V$  or  $B \subseteq L$  then  $\text{EQ}_C(B)$  and  $\text{ISO}_C(B)$  are tractable.*
- (2) *In all other cases  $\text{EQ}_C(B)$  is  $\leq_m^{\log}$ -complete for  $\text{coNP}$  and  $\text{ISO}_C(B)$  is  $\leq_m^{\log}$ -hard for  $\text{coNP}$ .*

### 5. RESULTS FOR AUDIT-LIKE PROBLEMS

This section covers our results about the problems related to the audit and frozen variable problems. We start with the following basic facts about complexity upper bounds.

**Proposition 5.1.** *For every finite set  $B$  of Boolean functions, the following upper bounds hold:*

- (1)  $\text{FV}_C(B) \in \text{D}^{\text{P}}$ ,
- (2)  $\exists \text{FV}_C(B) \in \text{D}^{\text{P}}$ ,
- (3)  $\text{Unique SAT}_C(B) \in \text{D}^{\text{P}}$ , and
- (4)  $\text{AUDIT}_C(B) \in \text{coNP}$ .

As an auxiliary problem we will first examine the generalization of the satisfiability problem  $\text{SAT}_C^*(B)$ , which asks whether a  $B$ -circuit has a satisfying assignment different from the all 1's one. This problem was examined in [CH97] in the constraint setting.

**Theorem 5.2.** *Let  $B$  be a finite set of Boolean functions. Then  $\text{SAT}_C^*(B)$  is NP-complete if  $S_{12} \subseteq [B]$ , and solvable in P otherwise.*

*Proof.* First assume  $S_{12} \subseteq [B]$ . In this case by looking at Post's lattice (see Figure 2) we know that  $[S_{12} \cup \{0\}] = S_1$ , hence following Proposition 3.1,  $\text{SAT}_C(S_{12} \cup \{0\})$  is NP-complete. We will now reduce  $\text{SAT}_C(S_{12} \cup \{0\})$  to  $\text{SAT}_C^*(S_{12})$ . Given an  $(S_{12} \cup \{0\})$ -circuit  $C(x_1, \dots, x_n, 0)$  we use a new variable  $x$  as a replacement for the constant 0. Thus we obtain an  $S_{12}$ -circuit  $C'(x_1, \dots, x_n, x)$ . Looking at Table 1 we see that the Boolean function  $g(x, y, z) = x \wedge (y \vee \bar{z})$  belongs to  $S_{12}$ . Hence, let  $\hat{C}$  be the  $S_{12}$ -circuit defined by  $\hat{C} = g(\dots g(g(C', x_1, x), x_2, x), \dots, x_n, x)$ . Observe that  $\hat{C}$  is equivalent to  $C' \wedge (x_1 \vee \bar{x}) \wedge (x_2 \vee \bar{x}) \wedge \dots \wedge (x_n \vee \bar{x})$ . Observe now that  $C$  has a satisfying assignment if and only if there is an assignment different from the all 1's one that satisfies  $\hat{C}$ . We conclude that  $\text{SAT}_C^*(S_{12})$  is NP-hard, thus showing that  $\text{SAT}_C^*(B)$  is NP-complete for all  $B$  such that  $S_{12} \subseteq [B]$ .

If  $B \subseteq M$ , then an  $n$ -ary circuit  $C$  has a satisfying assignment besides the all-1-assignment if and only if it has a satisfying assignment of the form  $1^i 01^{n-i-1}$ . Hence  $\text{SAT}_C^*(B)$  can be solved with  $n$  evaluations of the circuit  $C$ .

If  $B \subseteq L$ , we can use the linear normal form  $0 \oplus \bigoplus_{i=1}^n c_i x_i$ , which is obviously polynomial time computable, to solve  $\text{SAT}_C^*(B)$  efficiently.

If  $B \subseteq D$  or  $B \subseteq S_0^2$  then we claim that each  $n$ -ary  $B$ -circuit has at least  $2^{n-1}$  satisfying assignments, which obviously makes  $\text{SAT}_C^*(B)$  tractable. If  $B \subseteq D$  the claim holds for any

self-dual circuit has exactly  $2^{n-1}$  solutions. If  $B \subseteq S_0^2$ , note that for every  $B$ -circuit and compatible assignment  $I$ , if  $I$  does not satisfy  $C$ , then  $\text{dual}(I)$  does. Indeed, assume that both  $I$  and  $\text{dual}(I)$  do not satisfy  $C$ . Since  $C$  is a  $B$ -circuit, and  $B \subseteq S_0^2$ , we know that the function described by  $C$  is 0-separating of degree 2. Thus every set  $S$  with  $|S| = 2$  and  $S \subseteq C^{-1}(\{0\})$  is 0-separating. The set  $S$  defined as  $S \{(I(x_1), \dots, I(x_n)), (\text{dual}(I)(x_1), \dots, \text{dual}(I)(x_n))\}$  meets these conditions, and hence is 0-separating. From the definition, it follows that there is some  $i \in \{1, \dots, n\}$  such that  $I(x_i) = \text{dual}(I)(x_i)$ , which is a contradiction to the definition of  $\text{dual}(I)$ . In particular, the number of solutions of such a circuit is at least  $2^{n-1}$ .  $\square$

Next we want to study problems which are related to the concept of frozen variables.

**Lemma 5.3.** *Let  $B$  be a finite set of Boolean functions. If  $D_1 \subseteq [B] \subseteq D$  or  $S_{02} \subseteq [B] \subseteq R_1$ , then  $\text{FV}_C(B)$  and  $\exists \text{FV}_C(B)$  are coNP-complete.*

*Proof.* Observe that for all  $B$  that satisfy the conditions above all  $B$ -circuits are trivially satisfiable, hence  $\text{FV}_C(B)$  and  $\exists \text{FV}_C(B)$  are in coNP.

Let  $S_{02} \subseteq [B] \subseteq R_1$ . We will reduce the coNP-complete problem (see Proposition 3.1)  $\overline{\text{SAT}}_C(R_0)$  to  $\exists \text{FV}_C(B)$  and  $\text{FV}_C(B)$ . Recall that due to the very end of Section 2.1, we know that  $\{x \vee (y \wedge \bar{z}), 0\}$  is a base of  $R_0$ . Hence let  $C$  be a circuit over  $\{x \vee (y \wedge \bar{z}), 0\}$ . We build a new circuit  $C'$  out of  $C$  by taking a fresh variable  $x$  and by replacing every occurrence of 0 in  $C$  with  $x$ . Then  $C'$  is an  $S_{02}$ -circuit. Since  $V_2 \subseteq S_{00} \subseteq S_{02}$  we have  $\vee \in [B]$  (see Figure 2), and hence  $C' \vee x$  can be converted into an equivalent  $B$ -circuit. Note that the only possibly frozen variable in  $C' \vee x$  is  $x$ , since with setting  $x$  to true, every possible assignment to the other variables satisfies the circuit. Finally,  $x$  is a frozen variable in  $C' \vee x$  if and only if  $C$  is not satisfiable.

Now let  $D_1 \subseteq [B] \subseteq D$ . We will reduce the coNP-complete problem  $\text{EQ}_C(D_1)$  (see Theorem 4.11) to  $\exists \text{FV}_C(B)$  and  $\text{FV}_C(B)$ . For that let  $C_1$  and  $C_2$  be two  $n$ -ary  $D_1$ -circuits. Let  $x$  be a fresh variable and let  $C' = x \oplus C_1 \oplus C_2$ . Since  $x \oplus y \oplus z$  is a function in  $D_1$  (see Figure 2),  $C'$  is a  $D_1$ -circuit. Consider now the  $D_1$ -circuit  $C''(x, y, z) = t_2(x, y, z)$  (remind that  $t_2 \in D_1$ ). The reduction  $\Phi$  works as follows:

$$\Phi(C_1, C_2) = \begin{cases} C'' & , \text{ if } C_1(0^n) \neq C_2(0^n) \text{ or } C_1(1^n) \neq C_2(1^n) \\ C' & \text{ otherwise} \end{cases}$$

We claim that  $C_1 \equiv C_2$  holds if and only if  $\Phi(C_1, C_2) \in \exists \text{FV}_C(D_1)$  if and only if  $(\Phi(C_1, C_2), \{x\}) \in \text{FV}_C(B)$ . For that let  $C_1 \equiv C_2$ , then  $\Phi(C_1, C_2) = C'$ . Since  $C_1 \oplus C_2 \equiv 0$  the formula  $C'$  is satisfied if and only if  $x$  is satisfied. Thus  $x$  is a frozen variable and therefore  $(\Phi(C_1, C_2), \{x\}) \in \text{FV}_C(B)$ . On the other hand, if  $C_1 \not\equiv C_2$  then we have two cases. If  $C_1(0^n) \neq C_2(0^n)$  or  $C_1(1^n) \neq C_2(1^n)$  then  $\Phi(C_1, C_2) = C''$ , which is a circuit without frozen variables. If  $C_1(0^n) = C_2(0^n)$  and  $C_1(1^n) = C_2(1^n)$  then  $\Phi(C_1, C_2) = C'$ . Since  $C_1 \not\equiv C_2$ , there is an assignment  $\alpha \in \{0, 1\}^n$  such that  $C_1(\alpha) \neq C_2(\alpha)$ . Therefore  $1 = 0 \oplus C_1(\alpha) \oplus C_2(\alpha) = 1 \oplus C_1(0^n) \oplus C_2(0^n) = 1 \oplus C_1(1^n) \oplus C_2(1^n)$  and there is no frozen variable in  $C'$ .  $\square$

The following is our main classification result for the problem that asks if there is any frozen variable:

**Theorem 5.4.** *Let  $B$  be a finite set of Boolean functions.*

- (1) *If  $B \subseteq L$ ,  $B \subseteq M$ , or  $B = S_{12}$  then  $\exists \text{FV}_C(B)$  is tractable.*
- (2) *If  $[B] = S_1$ , then  $\exists \text{FV}_C(B)$  is NP-complete.*

- (3) If  $S_1 \subset [B]$  then  $\exists \text{FV}_C(B)$  is  $D^P$ -complete.  
 (4) In all other cases  $\exists \text{FV}_C(B)$  is  $\text{coNP}$ -complete.

*Proof.*

- (1) If  $B \subseteq L$  then in a  $B$ -circuit there is a frozen variable if and only if exactly one of the variables of its linear normal form has a coefficient of 1. Note that the linear normal form can easily be computed from the circuit using simulation (see proof of Proposition 4.2).

Now let  $B \subseteq M$  and let  $C(x_1, \dots, x_n)$  be a  $B$ -circuit. By monotonicity the variable  $x_i$  is frozen if and only if  $C(1^n) = 1$  and  $C(1^{i-1}01^{n-i}) = 0$ . Moreover the satisfiability of a monotonic  $C$  can be easily tested. If  $B = S_{12} = R_1 \cap S_1$  then every  $B$ -circuit  $C$  is satisfiable and has a frozen variable because  $C$  is 1-separating.

- (2) Note that an  $S_1$ -circuit has a frozen variable by definition if and only if it is satisfiable and it is therefore equivalent to  $\text{SAT}_C(B)$ , which is known to be  $\text{NP}$ -complete by Proposition 3.1.  
 (3) Let  $B$  such that  $S_1 \subset [B]$ . It is obvious, that  $\exists \text{FV}_C(B)$  is in  $D^P$ . Let us now introduce the problem  $\text{SATP}(B)$ :

$$\text{SATP}(B) = \{ (C_1, C_2) : C_1 \text{ and } C_2 \text{ are } B\text{-circuits} \\ \text{and } (C_1 \in \text{SAT}_C(B) \text{ xor } C_2 \in \text{SAT}_C(B)) \}$$

By definition  $\text{SATP}(B)$  is in  $D^P$  and is  $D^P$ -complete as far as  $\text{SAT}_C(B)$  is  $\text{NP}$ -complete (cf. [CGH<sup>+</sup>88, CGH<sup>+</sup>89]). We now reduce  $\text{SATP}(B)$  to  $\exists \text{FV}_C(B)$ , thus completing the proof. Since  $S_1 \subset [B]$ , there is a  $k \geq 2$  such that  $t_k$  is in  $[B]$ . Let  $C_1$  and  $C_2$  be  $B$ -circuits which are  $m$ - and  $n$ -ary respectively. Now define  $C = t_k(C_1, C_2, x_1, \dots, x_{k-1})$ , where  $x_i$  is a fresh variable for  $1 \leq i \leq k-1$ . Clearly  $C$  is a  $B$ -circuit. Next we show that this transformation gives the needed reduction.

If  $C_1, C_2$  are both satisfiable then there are assignments  $\alpha \in \{0, 1\}^m$  and  $\beta \in \{0, 1\}^n$  such that  $C_1(\alpha) = C_2(\beta) = 1$ . Then none of the variables of  $C_1$  is frozen in  $C$ , since  $C(\gamma\beta 1^{k-1}) = 1$  for all  $\gamma \in \{0, 1\}^m$ . The same argumentation holds for all variables in  $C_2$ . Furthermore for all  $1 \leq i \leq k-1$  it holds that  $x_i$  is not frozen in  $C$ , since  $C(\alpha\beta 1^{i-1}01^{k-i-1}) = C(\alpha\beta 1^{k-1}) = 1$ .

If  $C_1, C_2$  are both unsatisfiable, then  $C$  is not satisfiable and therefore has no frozen variables.

Suppose now without loss of generality  $C_1$  is satisfiable and that  $C_2$  is not, then  $C \equiv C_1 \wedge x_1 \wedge \dots \wedge x_{k-1}$  and obviously at least all of the  $x_i$ 's ( $1 \leq i \leq k-1$ ) are frozen.

- (4) If  $D_1 \subseteq [B] \subseteq D$  or  $S_{02} \subseteq [B] \subseteq R_1$ , then  $\exists \text{FV}_C(B)$  is  $\text{coNP}$ -complete because of Lemma 5.3. The only remaining case is  $S_{12} \subset [B] \subseteq R_2$ . Then  $B$ -circuits are trivially satisfiable and therefore  $\exists \text{FV}_C(B) \in \text{coNP}$ . The proof of the lower bound is similar to Case 3, but this time the reduction starts with  $\text{SAT}_C^*(B)$ , which is  $\text{coNP}$ -complete (see Theorem 5.2). Since  $S_{12} \subset [B] \subseteq R_1$  there is a  $k \geq 2$  such that  $t_k$  is in  $[B]$ . Let  $C(x_1, \dots, x_n)$  be a  $B$ -circuit with the variables  $x_1, \dots, x_n$ . Let  $C' = t_k(C, y_1, \dots, y_k)$ ,  $C'' = \left( (\bigwedge_{i=1}^k y_i) \vee \neg(\bigwedge_{j=1}^n x_j) \right)$  and  $G(x_1, \dots, x_n, y_1, \dots, y_k) = C' \wedge C''$ . Observe that  $C''$  and therefore  $G$  can be converted into equivalent  $B$ -circuits, because  $\wedge \in [B]$  and  $\{x \wedge (y \vee \bar{z})\} = S_{12}$ .

If  $C$  is unsatisfiable, then because of  $C'$  then  $G$  is satisfiable only by setting  $y_i$  to 1 for all  $1 \leq i \leq k$ . The same holds if  $C$  has the all-1 assignment as only satisfying assignment because of  $C''$ . Hence in both cases all  $y_i$  are frozen.

On the other hand, if there is an  $\alpha \in \{0, 1\}^n$  such that  $\alpha \neq (1, \dots, 1)$  and  $C(\alpha) = 1$  then none of the  $x_i$ 's is frozen ( $1 \leq i \leq n$ ), since  $G$  can be satisfied by just setting all the  $y_i$ 's to 1. Furthermore, for each  $j \in \{1, \dots, k\}$  holds  $G(\alpha 1^{j-1} 0 1^{k-j}) = 1 = G(\alpha 1^k)$ , hence none of the  $y_j$ 's is frozen.  $\square$

Concerning the variant of the above problem, where the frozen variable is part of the input, we obtain the following classification:

**Theorem 5.5.** *Let  $B$  be a finite set of Boolean functions.*

- (1) *If  $B \subseteq M$  or  $B \subseteq L$ , then  $FV_C(B)$  is tractable,*
- (2) *else if  $S_1 \subseteq [B]$ , then  $FV_C(B)$  is  $D^P$ -complete,*
- (3) *else  $FV_C(B)$  is coNP-complete.*

*Proof.* If  $B \subseteq M$  or  $B \subseteq L$ , then the argumentations from Theorem 5.4.1 hold.

We have seen in Lemma 5.3 that if  $D_1 \subseteq [B] \subseteq D$  or  $S_{02} \subseteq [B] \subseteq R_1$ , then  $FV_C(B)$  is coNP-complete.

This leaves the coNP-hardness of  $FV_C(B)$  for  $B$  such that  $S_{12} \subseteq [B]$  and the  $D^P$ -hardness of  $FV_C(B)$  for all  $B$  such that  $S_1 \subseteq [B]$  to show. We reduce  $FV_C(R_1)$  to  $FV_C(S_{12})$  ( $FV_C(BF)$  to  $FV_C(S_1)$ , resp.). For that, let  $C$  be a circuit over the  $R_1$ -base  $\{x \wedge (y \vee \bar{z}), 1\}$  (over the BF-base  $\{x \wedge \bar{y}, 1\}$ , resp.) and let  $V$  be the set of variables used in  $C$ . Build a circuit  $C'$  by taking a variable  $x$  that is not contained in  $C$  and replace every occurrence of 1 in  $C$  by  $x$ . Then  $C' \wedge x$  is an  $S_{12}$ -circuit (an  $S_1$ -circuit, resp.), which can only be satisfied by assignments that set  $x$  to 1. For all these assignments,  $C' \wedge x$  is satisfied if and only if  $C$  is satisfied. Therefore  $(C, V) \in FV_C(R_1)$  ( $(C, V) \in FV_C(BF)$ , resp.) if and only if  $(C' \wedge x, V) \in FV_C(S_{12})$  ( $(C' \wedge x, V) \in FV_C(S_1)$ , resp.).  $\square$

To obtain a classification for the audit problem, we first note the following link between the complexity of the problem  $\exists FV_C(B)$  and the audit problem:

**Proposition 5.6.** *Let  $B$  be an arbitrary set of Boolean functions, then*

- (1)  *$\exists FV_C(B) = SAT_C(B) \cap AUDIT_C(B)$ , and*
- (2) *If  $SAT_C(B) \in P$  then  $\exists FV_C(B) \equiv_m^P AUDIT_C(B)$ .*

The classification now is as follows:

**Theorem 5.7.** *Let  $B$  be a finite set of Boolean functions.*

- (1) *If  $B \subseteq M$  or  $B \subseteq L$  or  $B \subseteq S_1$ , then  $AUDIT_C(B)$  is tractable,*
- (2) *else  $AUDIT_C(B)$  is coNP-complete.*

*Proof.*

- (1) Since  $SAT_C(B) \in P$  if  $[B] \subseteq L$  or  $[B] \subseteq M$  (Proposition 3.1), the claim for such  $B$  follows from Proposition 5.6 and Theorem 5.4. If  $B \subseteq S_1$ , a  $B$ -circuit is either not satisfiable or has always a frozen variable, hence the problem is tractable.
- (2) If  $B \subseteq R_1$  or  $B \subseteq D$ , every  $B$ -circuit is trivially satisfiable. Therefore we can use Proposition 5.6 and Theorem 5.4 again. It remains to show that  $AUDIT_C(B)$  is coNP-hard if  $S_1 \subseteq [B]$ . We will reduce the coNP-complete problem  $\overline{SAT_C(B)}$  (see Proposition 3.1) to  $AUDIT_C(B)$ . The proof runs along the same lines as in Theorem 5.4. Take an  $B$ -circuit  $C$ . Since  $S_1 \subseteq [B]$  there is a  $k$  with  $t_k \in [B]$ . Define  $C' = t_k(C, x_1, \dots, x_k)$ , where  $x_i$  is a variable not occurring in  $C$  for  $1 \leq i \leq k$ . If  $C$  is not satisfiable then  $x_i$  is frozen for  $1 \leq i \leq k$ . If  $C$  is satisfiable by an assignment  $\alpha$  none of the variables from

$C$  is frozen in  $C'$ , since  $C'$  can be satisfied by setting all the  $x_i$ 's to 1. Furthermore  $x_i$  is not frozen for  $1 \leq i \leq k$ , because  $C'(\alpha 1^{i-1} 0 1^{k-i}) = C'(\alpha 1^k) = 1$ .  $\square$

We finish this section with a classification of the unique satisfiability problem.

**Theorem 5.8.** *Let  $B$  be a finite set of Boolean functions.*

- (1) *If  $S_1 \subseteq [B]$ , then  $\text{Unique SAT}_C(B) \equiv_m^{\log} \text{Unique SAT}_C(\text{BF})$*
- (2) *else if  $S_{12} \subseteq [B] \subseteq R_1$ , then  $\text{Unique SAT}_C(B)$  is coNP-complete*
- (3) *In all other cases  $\text{Unique SAT}_C(B)$  is tractable.*

*Proof.*

- (1) Trivially  $\text{Unique SAT}_C(B) \leq_m^{\log} \text{Unique SAT}_C(\text{BF})$  for an arbitrary set  $B$  of Boolean functions. According to Proposition 3.5,

$$\text{Unique SAT}_C(\text{BF}) \leq_m^{\log} \text{Unique SAT}_C(S_1 \cup \{1\}).$$

We show that  $\text{Unique SAT}_C(S_1 \cup \{1\}) \leq_m^{\log} \text{Unique SAT}_C(S_1)$ , which in turn will prove that  $\text{Unique SAT}_C(\text{BF}) \leq_m^{\log} \text{Unique SAT}_C(B)$  for all  $B$  such that  $S_1 \subseteq [B]$ . Let  $C$  be a  $S_1 \cup \{1\}$ -circuit and let  $x$  be a variable not occurring in  $C$ . Let  $C''$  be the circuit obtained from  $C$  in replacing every occurrence of 1 by  $x$ . Finally consider  $C' = C'' \wedge x$ . Observe, that since  $\wedge \in S_1$  the circuit  $C'$  is an  $S_1$ -circuit and  $\#\text{Sat}(C) = \#\text{Sat}(C')$ .

- (2) If  $B \subseteq R_1$  we have  $\text{Unique SAT}_C(B) \equiv_m^{\log} \overline{\text{SAT}_C^*(B)}$ . Hence for all  $B \subseteq R_1$  holds  $\text{Unique SAT}_C(B) \in \text{coNP}$  and therefore the coNP-completeness for all  $B$  with  $S_{12} \subseteq [B] \subseteq R_1$  follows by Theorem 5.2.
- (3) For all  $B \subseteq S_0^2$  or  $B \subseteq D$  the claim holds because as we have seen before any such circuit has at least  $2^{n-1}$  satisfying assignments. If  $B \subseteq M$ , then an  $n$ -ary  $B$ -circuit  $C$  has more than one satisfying assignment if and only if there is an  $i \in \{1, \dots, n\}$  such that  $C(1^{i-1} 0 1^{n-i}) = 1$ . If  $B \subseteq L$ , then the number of satisfying assignments for every  $B$ -circuit can easily be determined using its linear normal form.  $\square$

## 6. ENUMERATION PROBLEMS

We now present our results for the enumeration problem. The analogous problem has been studied in the constraint context by Nadia Creignou, Jean-Jacques Hébrard, Henning Schnoor, and Ilka Schnoor in [CH97, SS07]. The counting problem (i.e., determine the number of solutions of a given circuit has been studied in [RW05]).

**Theorem 6.1.** *Let  $B$  be a finite set of Boolean functions. Then the following holds:*

- (1) *If  $B \subseteq M$ , or  $B \subseteq L$ , or  $B \subseteq D$ , or  $B \subseteq S_0^2$ , then  $\text{Enum SAT}_C(B)$  has a polynomial-delay enumeration algorithm.*
- (2) *Else  $\text{Enum SAT}_C(B)$  has no polynomial-total-time enumeration algorithm unless  $P = \text{NP}$ .*

Note that since every polynomial-delay algorithm is also a polynomial-total-time algorithm, the above theorem implies that in the context of enumerating the solutions for  $B$ -formulas, the two notions coincide. In particular, the theorem completely classifies the “efficient” cases with respect to either of these notions.

*Proof.*

- (1) Let us first examine the case where  $B \subseteq M$  or  $B \subseteq L$ . In this case it follows from Proposition 3.1 and Figure 2 that the satisfiability problem for  $B \cup \{0, 1\}$ -circuits can be solved in polynomial time. Thus it is easy to see that the following algorithm has polynomial delay: Let  $C(x_1, \dots, x_n)$  be a  $B$ -circuit. We first check if  $C[x_1/0]$  (that is, the circuit resulting from  $C$  when replacing all gates labeled  $x_1$  with a gate computing the constant 0-function) is satisfiable, if yes, we recursively print the satisfying solutions of this circuit with the additional assignment  $x_1 = 0$ . We do the same for the analogously defined  $C[x_1/1]$ . For a circuit without variables, we print the empty assignment.

Let us now consider the case where  $B \subseteq D$ , or  $B \subseteq S_0^2$ . In this case, as we have seen in the proof of Theorem 5.2, we know that for any  $B$ -circuit  $C$  and any assignment  $I$  to the variables of  $C$ , if  $I$  is no solution for  $C$ , then  $\text{dual}(I)$  is. This gives a polynomial-delay enumeration algorithm for the solutions of  $C$ , by testing the set of all assignments in an appropriate order: let the variables of  $C$  be  $x_1, \dots, x_n$ , then use an arbitrary order, for example the lexicographical order, on the assignments  $I$  with  $I(x_1) = 0$ , and for each of the assignments considered, test if  $I$  or  $\text{dual}(I)$  satisfies the circuit. In the cases where the answer is “yes,” print the corresponding assignment. Due to the above mentioned property, this gives at least one solution for each  $I$  considered, since if  $I$  is not a solution, then  $\text{dual}(I)$  is. Therefore, since it can be verified in polynomial time if a given assignment is a solution for the circuit, this clearly gives a polynomial delay algorithm.

- (2) According to Figure 2 in order to complete the proof of the theorem it remains to show that if  $B$  is such that  $S_{12} \subseteq [B]$ , then  $\text{Enum SAT}_C(B)$  has no polynomial-total-time enumeration algorithm unless  $P = NP$ . We show that the existence of such an algorithm for  $B$ -circuits implies that  $\text{SAT}_C^*(B)$  can be decided in polynomial time. The theorem then follows from the proof of Theorem 5.2, since there it was proven that  $\text{SAT}_C^*(B)$  is NP-hard.

Let  $C$  be a  $B$ -circuit. First check if the constant 1-assignment is a solution of  $C$ , this can be done in polynomial time. Let  $i$  be 1 if this is the case, and let  $i$  be 0 otherwise (i.e., if the constant 1-assignment does not satisfy  $C$ ). Clearly,  $C$  has a solution different from the all-1-solution if and only if it has at least  $i+1$  many solutions. Using a polynomial-total-time enumeration algorithm for  $B$ -circuits, this question can be decided as follows:

Since  $i + 1$  can be at most 2, the time that a polynomial-total-time enumeration algorithm can spend for enumerating all of  $C$ 's solution is bounded by a polynomial in  $C$ . Therefore, we can simply start the algorithm, and wait if it finishes in this time. If it does, then its output is the full list of solutions for  $C$ , and we obviously can decide if there is solution different from the constant-1-solution present in this list. If it does not finish in this time, then there are more than  $i + 1$  solutions, and thus there is one which is not the constant-1-solution. Note that we deduce this fact solely from the observation that the algorithm runs longer than allowed for  $i + 1$  solutions, independent of any output the algorithm may have printed up to that time.  $\square$

In the case of the existence of a polynomial-delay enumeration algorithm it is of interest to further examine the complexity of the enumeration when requiring the solutions to be output in lexicographic order. As observed in [JPY88] this further requirement can dramatically increase the complexity. We prove that this is indeed the case for some sets  $B$ .

**Proposition 6.2.** *Let  $B$  be a finite set of Boolean functions such that  $B \subseteq M$ , or  $B \subseteq L$ , or  $B \subseteq D$ , or  $B \subseteq S_0^2$ .*

- (1) *If  $B \subseteq M$  or  $B \subseteq L$ , then there exists a polynomial-delay enumeration algorithm that produces all the solutions of a  $B$ -circuit in lexicographic order.*
- (2) *Else such an algorithm does not exist unless  $P = NP$ .*

*Proof.* Observe that the enumeration algorithm described in the proof of Theorem 6.1 when  $B \subseteq M$  or  $B \subseteq L$  produces the solutions in lexicographic order. According to Figure 2 it remains to consider the case where  $S_{02} \subseteq [B]$  or  $D_1 \subseteq [B]$ . We prove that for any set  $B$ , if one of these algorithms exists, then the satisfiability problem for  $B \cup \{0\}$ -circuits can be solved in polynomial time. The result then follows with Proposition 3.1, since due to Figure 2,  $[B \cup \{0\}] = BF$ , and therefore this problem is NP-complete.

We show how a polynomial-time decision algorithm for this problem can be obtained from a polynomial-delay enumeration algorithm for  $B$ -circuits that produces solutions in lexicographic order. To this end, let  $C$  be a  $B \cup \{0\}$ -circuit. Introduce a new variable  $x_0$ , and construct the circuit  $C'$ , which is obtained from  $C$  by replacing every occurrence of 0 by  $x_0$ . Then  $C'$  is a  $B$ -circuit. It is clear that  $C$  has a solution if and only if  $C'$  is satisfiable and the lexicographically first solution of  $C'$  maps  $x_0$  to 0, which clearly finishes the proof, since the lexicographic order enumeration algorithm has to produce the first solution in polynomial time, or determine that none exists.  $\square$

In addition to the cases where the satisfiability problem for  $B$ -circuits is NP-complete, and therefore efficient enumeration algorithms obviously cannot be hoped for unless  $P = NP$ , we also showed that in the cases where tractability of the satisfiability problem follows from a simple “trick,” like the knowledge that the all-1-assignment is a solution to the circuits, efficient enumeration algorithms do not exist. An interesting special case here is the case of self-dual circuits. The satisfiability problem again is easy, simply because any such circuit is always satisfiable. But the property of self-duality does not only give one solution, it guarantees that half of the possible assignments are solutions. Therefore it is not surprising that these solutions also can be enumerated in an efficient way. However, since the property of self-duality does not say anything about the set of solutions where a given variable is set to 0, this does not help us to construct a lexicographical order enumeration algorithm.

Given the above results and those on counting given in [RW05], one can see that counting is “harder” than enumeration in the following sense: For all cases in which [RW05] gives a polynomial-time algorithm for the counting problem, we also obtain an efficient (polynomial-delay) algorithm for enumeration. The converse is not true: For monotone functions, efficient enumeration is possible, but counting cannot be done in polynomial time, unless  $\#P \subseteq FP$ . When considering lexicographic enumeration algorithm, the picture is similar, with the notable exception of the clone of the self-dual functions: As already discussed above, enumeration is trivial for these functions. For a similar reason, the counting problem is trivial here as well (a self-dual function is satisfied by exactly half of its possible arguments). However, the self-dual property does not help in obtaining an algorithm for enumeration in lexicographic order.

## 7. CONCLUSION

We have obtained complete classifications for the equivalence and isomorphism problems, the frozen variables problems, the unique satisfiability problem, the audit problem and the

enumeration problem for Boolean circuits. The classification into “hard” and “easy” classes can be refined such that the internal structure of the tractable cases becomes visible. For this, one has to use stricter reductions (e.g., logspace reductions or logtime projections), and one obtains problems complete for subclasses of P. For some of our problems, this has been done in [Rei01].

We think it is interesting to observe that, e.g., equivalence of OBDDs is decidable in polynomial time, while we identify here intractability for many clones in the lattice. As a consequence, this shows that, if  $P \neq NP$ , in all these cases OBDDs provide a provably less succinct representation than Boolean circuits. Analogous remarks hold for cases of the other algorithmic tasks that we consider, where a difference in complexity between OBDD representation and circuit representation appears.

In general, given a Boolean function  $f$ , it is coNP-hard to determine if it is in a clone  $B$  (if  $f$  is given by a general circuit; the problem becomes very easy if  $f$  is given by truth-table, see [Vol09]). This might seem to destroy all relevance of our just discussed results. However, we would like to mention that in practice, circuits computing functions  $f$  are synthesized in one way or the other, hence we know the minimal clone it belongs to, and thus, our tractability results are relevant.

In this paper we studied the complexity of problems related to circuits. So it is natural to ask what can be said about the formula case. For this we define  $B$ -formulas as “tree-like”  $B$ -circuits or analogously as  $B$ -circuits, where all gates have a fan-out of at most 1. Interestingly the study of  $B$ -formulas leads to different dichotomy-theorems. The main reason for this phenomenon is, that circuits can be regarded as a succinct representation of formulas. Partial results in this direction have been obtained in [Rei01, Sch10].

Finally we would like to mention that another possible syntactic restriction of formula related problems is to consider generalized Boolean CNF formulas, also known as CSPs, see [CKS00, CV08]. Many results about the problems considered here have been obtained in the CSP framework, see the survey [CV08].

*Acknowledgement.* We are grateful to the reviewers for many comments that helped to improve the presentation considerably.

## REFERENCES

- [AT00] M. Agrawal and T. Thierauf. The formula isomorphism problem. *SIAM Journal on Computing*, 30(3):990–1009, 2000.
- [BCRV03] E. Böhler, N. Creignou, S. Reith, and H. Vollmer. Playing with Boolean blocks, part I: Post’s lattice with applications to complexity theory. *ACM SIGACT-Newsletter*, 35(4):38–52, 2003.
- [BG82] A. Blass and Y. Gurevich. On the unique satisfiability problem. *Information and Control*, 82:80–88, 1982.
- [CGH<sup>+</sup>88] J. Cai, T. Gundermann, J. Hartmanis, L. Hemachandra, V. Sewelson, K. Wagner, and G. Wechsung. The Boolean hierarchy I: Structural properties. *SIAM Journal on Computing*, 17(6):1232 – 1252, December 1988.
- [CGH<sup>+</sup>89] J. Cai, T. Gundermann, J. Hartmanis, L. Hemachandra, V. Sewelson, K. Wagner, and G. Wechsung. The Boolean hierarchy II: Applications. *SIAM Journal on Computing*, 18(1):95 – 111, February 1989.
- [CH97] N. Creignou and J.-J. Hébrard. On generating all solutions of generalized satisfiability problems. *Informatique Théorique et Applications/Theoretical Informatics and Applications*, 31(6):499–511, 1997.



- [CKS00] N. Creignou, S. Khanna, and M. Sudan. *Complexity Classifications of Boolean Constraint Satisfaction Problems*. Monographs on Discrete Applied Mathematics. SIAM, 2000.
- [CV08] N. Creignou and H. Vollmer. Boolean constraint satisfaction problems: When does Post’s lattice help? In Nadia Creignou, Phokion G. Kolaitis, and Heribert Vollmer, editors, *Complexity of Constraints*, volume 5250 of *Lecture Notes in Computer Science*, pages 3–37. Springer, 2008.
- [JPY88] D. Johnson, C. Papadimitriou, and M. Yannakakis. On generating all maximal independent sets. *Inf. Process. Lett.*, 27(3):119–123, 1988.
- [JYP88] D.S. Johnson, M. Yannakakis, and C.H. Papadimitriou. On generating all maximal independent sets. *Information Processing Letters*, 27:119–123, 1988.
- [KPR03] J. M. Kleinberg, C. H. Papadimitriou, and P. Raghavan. Auditing Boolean attributes. *J. Comput. Syst. Sci.*, 66(1):244–253, 2003.
- [Lad75] R. E. Ladner. The circuit value problem is log space complete for P. *SIGACT News*, 7(1):12–20, 1975.
- [Lau06] D. Lau. *Function Algebras on Finite Sets*. Springer Monographs in Mathematics. Springer, 2006.
- [Lew79] H. R. Lewis. Satisfiability problems for propositional calculi. *Mathematical Systems Theory*, 13:45–53, 1979.
- [Lup58] O. B. Lupanov. A method of circuit synthesis. *Izvestia V.U.Z. Radiofizika*, 1:120–140, 1958.
- [MT98] Christoph Meinel and Thorsten Theobald. *Algorithms and Data Structures in VLSI Design: OBDD - Foundations and Applications*. Springer, 1998.
- [Pip97] N. Pippenger. *Theories of Computability*. Cambridge University Press, Cambridge, 1997.
- [PK79] R. Pöschel and L.A. Kalužnin. *Funktionen- und Relationenalgebren*. DVW, Berlin, 1979.
- [Pos41] E. L. Post. The two-valued iterative systems of mathematical logic. *Annals of Mathematical Studies*, 5:1–122, 1941.
- [Rei01] S. Reith. *Generalized Satisfiability Problems*. PhD thesis, Fachbereich Mathematik und Informatik, Universität Würzburg, 2001.
- [RS42] J. Riordan and C. Shannon. The number of two-terminal series-parallel networks. *Journal of Mathematics and Physics*, 21:83–93, 1942.
- [RW05] S. Reith and K. W. Wagner. The complexity of problems defined by Boolean circuits. In *Proceedings Mathematical Foundation of Informatics (MFI99)*, pages 141–156. World Scientific Publishing, 2005.
- [Sav76] J. E. Savage. *The Complexity of Computing*. John Wiley, New York, 1976.
- [Sch10] Henning Schnoor. The complexity of model checking for Boolean formulas. *Int. J. Found. Comput. Sci.*, 21(3):289–309, 2010.
- [Sha38] C. Shannon. A symbolic analysis of relay and switching circuits. *Transactions AIEE*, 57:59–98, 1938.
- [SS07] H. Schnoor and I. Schnoor. Enumerating all solutions for constraint satisfaction problems. In Wolfgang Thomas and Pascal Weil, editors, *Proceedings of the 24th International Symposium on Theoretical Aspects of Computer Science*, pages 694–705, 2007.
- [Sze86] Á. Szendrei. *Clones In Universal Algebra*. Les Presses De L’Université de Montréal, 1986.
- [Vol99] H. Vollmer. *Introduction to Circuit Complexity – A Uniform Approach*. Texts in Theoretical Computer Science. Springer Verlag, Berlin Heidelberg, 1999.
- [Vol09] H. Vollmer. The complexity of deciding if a Boolean function can be computed by circuits over a restricted basis. *Theory Comput. Syst.*, 44(1):82–90, 2009.
- [Weg87] I. Wegener. *The Complexity of Boolean Functions*. Wiley-Teubner series in computer science. B. G. Teubner & John Wiley, Stuttgart, 1987.
- [Weg00] I. Wegener. *Branching Programs and Binary Decision Diagrams*. Monographs on Discrete Mathematics and Applications. SIAM, 2000.