Electronic Colloquium on Computational Complexity, Report No. 68 (2010)

Breaking local symmetries can dramatically reduce the length of propositional refutations

Shir Ben-Israel Eli Ben-Sasson*
Dept. of Computer Science
Technion
Haifa 32000, Israel
shirbenisrael@gmail.com, eli@cs.technion.ac.il

David Karger
CSAIL
MIT
Cambridge, MA, 02139
karger@mit.edu

April 12, 2010

Abstract

This paper shows that the use of "local symmetry breaking" can dramatically reduce the length of propositional refutations. For each of the three propositional proof systems known as *(i)* treelike resolution, *(ii)* resolution, and *(iii)* k-DNF resolution, we describe families of unsatisfiable formulas in conjunctive normal form (CNF) that are "hard-to-refute", i.e., require exponentially long refutations, but when a polynomial-time procedure for detecting and adding clauses arising from "local" symmetries — that permute only a constant number of variables — is applied to a formula in this family, it is transformed into an "easy-to-refute" CNF whose refutation length is polynomially bounded.

One of the most successful paradigms for solving instances of the satisfiability problem is the branch-and-bound strategy set forth by Davis, Putnam, Loveland and Logemann (DPLL). The three proof systems we discuss in this paper correspond respectively to *(i)* "standard" DPLL, *(ii)* DPLL augmented with clause learning, and *(iii)* multi-valued logic DPLL. Viewed with this correspondence in mind, our results suggest that the running time of SAT solvers from the above mentioned family of algorithms can be dramatically reduced when making use of information about the symmetries of a formula. This is true even when restricting our attention to the set of efficiently detectable symmetries that involve only a constant number of variables.

1 Introduction

The role of symmetries in mathematical proofs Consider how we (humans) prove the pigeonhole principle. Recall that this principle states that whenever $n + 1$ pigeons are placed in n pigeonholes there must exist a hole occupied by more than one pigeon. A typical proof starts by saying

*The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement number 240258. Research of the first two authors was supported grant number 679/06 by the Israeli Science Foundation.

ISSN 1433-8092

"Pigeon number $n+1$ sits in some pigeonhole. *Without loss of generality* let it be the last (n^{th}) one."

We will not complete the inductive argument of this theorem and rather pause to contemplate the phrase *without loss of generality*. It means that all pigeons and all holes are the same, at least as far as the pigeonhole principle is concerned. So if pigeon $n+1$ actually sits in the first pigeonhole we can simply switch the labels of these two holes and nothing would change. Lets state this phenomena again in a formal way using propositional calculus. Define a propositional variable $X_{(i,j)}$ to indicate the truth value of the proposition "pigeon number i sits in pigeonhole number j". This variable takes value 1 if the statement is true and otherwise is 0. Consider the set of constraints over propositional variables $\left\{X_{(i,j)} \;\middle|\; i \in \{1,\ldots,n+1\}, j \in \{1,\ldots,n\}\right\}$ that says *(i)* for every $i \in \{1,\ldots,n+1\}$ pigeon i sits in some pigeonhole (formally, $\bigvee_{j=1}^{n} X_{(i,j)}$), and *(ii)* for every $j \in \{1,\ldots,n\}$ pigeonhole j is occupied by at most 1 pigeon (formally, $\bigwedge_{i\neq i'}(\bar{X}_{(i,j)} \vee \bar{X}_{(i',j)})$). The standard notation in the literature for this unsatisfiable set of constraints capturing the pigeonhole principle is PHP_n. When we say that pigeonhole 1 and n are *the same* we mean that if, for every $i \in \{1,\ldots,n+1\}$, we replace variable $X_{(i,1)}$ by $X_{(i,n)}$ and vice versa replace variable $X_{(i,n)}$ by $X_{(i,1)}$ then the set of constraints obtained from this transposition is identical to the original set of constraints we started with — PHP_n. In other words, the permutation that switches $X_{(i,1)}$ and $X_{(i,n)}$ for all $i \in \{1,\ldots,n+1\}$ simultaneously and leaves all other variable labels unchanged is an *automorphism* of the formula PHP_n, i.e., PHP_n is invariant under the action of this permutation (cf. Definition 3.2).

Once we have identified an automorphism of PHP_n we can go one step further and use this information to add constraints to PHP_n that can potentially simplify the problem of *refuting* PHP_n, i.e., of proving that it is unsatisfiable. For instance, there are four possible assignments to the pair of variables $X_{(i,1)}, X_{(i,n)}$, namely $(0,0),(0,1),(1,0),(1,1)$. But since these two variables are interchangeable we need to consider only three assignments out of the four. Any assignment α that sets $X_{(i,1)}, X_{(i,n)}$ to $(1,0)$ satisfies PHP_n if and only if the assignment α' that sets $X_{(i,1)}, X_{(i,n)}$ to $(0,1)$ and otherwise agrees with α satisfies PHP_n. Consequently, we may safely add the constraint $\bar{X}_{(i,1)} \vee X_{(i,n)}$ to PHP_n — this constraint rules out setting $(0,1)$ to variables $X_{(i,1)}, X_{(i,n)}$ — and possibly make the task of refuting PHP_n easier.

In the case of the pigeonhole principle, humans are capable of seeing the symmetries inherent in the statement and mentally using these symmetries to quickly prove the statement. However, when it comes to automated proofs the situation changes dramatically. Many propositional proof systems used in practice to deal with general constraint satisfaction problems, like *treelike resolution* and *resolution*, are not designed to use symmetries such as those appearing in the pigeonhole principle. For these proof systems, pigeon $n+1$ sitting in the last hole is a completely different situation than that of pigeon $n+1$ sitting in the first hole. Consequently, the pigeonhole principle is known to require a propositional proof of length exponential in n in these ubiquitous systems [Hak85]. This is really a pity because many man-made and industry-based constraint satisfaction problems naturally contain a rich structure of symmetries. For example, think of a statement coming from the field of formal verification which concerns a circuit with many registers and all registers are equivalent.

This unfortunate weakness of propositional proof systems was first addressed by [Kri85] who suggested a way to incorporate symmetry breaking into resolution to provide shorter proofs of statements such as the pigeonhole principle. As pointed out in [CGLR96], there are several formidable challenges needed to be overcome in order to use the symmetries of a formula in a proof. The first is that of finding, via an efficient algorithm, the set of automorphisms of a CNF formula F and a succinct representation of this set (case in point — the pigeonhole principle, cf. [Kri85]). The set of automorphisms forms a group under composition and a succinct representation for this group is given by a set of generators. Finding a set of generators for a formula is as hard as solving the graph isomorphism problem [Cra92]. The second

problem with using the symmetries to simplify refutations is that, even if a set of generators is known for the group of automorphisms, encoding this information in a way that will make proofs shorter is quite difficult [CGLR96]. Assuming F involves n variables, it is known that the group of symmetries partitions the 2^n possible assignments into equivalence classes, and all assignments in a given class evaluate F to the same value. Thus, it is sufficient to examine only one representative from each class. Alas, the structure of the equivalence classes may be too complex, and the number of classes too large, to be useful for getting shorter proofs.

Given the obstacles we face in our attempt to use the symmetries of a formula to obtain shorter proofs for it, one question that naturally arises is whether it is worth the effort? Can information about the group of automorphisms of a formula drastically decrease the length of proofs of a formula F? Maybe the computational effort needed to utilize symmetry breaking predicates is greater than required to solve the formula without using such predicates. To quote from [ARMS02]:

> "In principle, the overhead due to symmetry detection and usage may outweigh the benefits, and it remains to be seen that useful CNF formulae have many symmetries."

The question of the effectiveness of symmetry breaking on decreasing proof length has been addressed in the empirical setting in a number of works [BS94, CGLR96, BFP96, BGS99, ARMS02, ASM06, Zha01, Shl07]. From a theoretical point of view [Urq99] has shown that information about the symmetries in a formula can lead to exponential savings in proof length.

Main results An automorphism of a formula — a permutation of its variables that leaves the formula unchanged — is said to have *support size* k if all but k variables are unchanged by it. (For instance, an automorphism that flips only two variables has support size 2.) Our main results show that symmetry breaking, even when applied to automorphisms of support size 2, can have a dramatic effect on proof length, bringing it down from exponential to polynomial. Our results apply to the three ubiquitous proof systems known as *(i) treelike resolution*, *(ii) resolution* and *(iii) k-DNF resolution*. For each of these systems we show that there exists a family of formulas that are "hard to refute" in the system, meaning any refutation of the formula is of exponential length in the size of the formula, but once all symmetry breaking constraints pertaining to symmetries of small support size are added to the formula (there is a small number of such constraints and they can be found efficiently via exhaustive search) its refutation length drops to polynomial. We now informally present our main theorems (formal statements appear in Section 5). We start with the weakest proof system known as treelike resolution (see Definition 2.2). In what follows we say that a sequence of polynomial-size formulas $\{F_n \mid n \in \mathbb{Z}\}, |F_n| = n^{\theta(1)}$ is *efficiently constructible* if there exists a polynomial time algorithm that on input 1^n outputs F_n.

Theorem 1.1 (Treelike resolution — informal)**.** *There exists an efficiently constructible family of 3-CNF formulas $\{F_n \mid n \in \mathbb{Z}\}$ over n variables and $\theta(n)$ clauses satisfying*

1. *The minimal length of a treelike resolution refutation of F_n is $2^{\Omega(n/\log n)}$.*
2. *Letting F'_n denote the formula obtained by adding to F_n all symmetry breaking clauses pertaining to symmetries of support size ≤ 2, then F'_n has a treelike resolution refutation of length $O(n)$.*

Our next result holds for the stronger proof system known as resolution (see Definition 2.1).

Theorem 1.2 (Resolution — informal)**.** *There exists an efficiently constructible family of 6-CNF formulas $\{F_n \mid n \in \mathbb{Z}\}$ over n variables and $\theta(n^{3/2})$ clauses satisfying*

1. *The minimal length of a resolution refutation of F_n is $2^{\Omega(\sqrt{n})}$.*

2. *Letting F'_n denote the formula obtained by adding to F_n all symmetry breaking clauses pertaining to symmetries of support size ≤ 2, then F'_n has a resolution refutation of length $O(n^{3/2})$.*

We end with our strongest proof system, known as k-DNF resolution (see Definition 2.3). For simplicity we state here only the case of constant k (for a more general statement see Theorem 5.6).

Theorem 1.3 (k-DNF resolution — informal)**.** *For any integer $k \geq 1$ there exists an efficiently constructible family of $O(k)$-CNF formulas $\{F_n \mid n \in \mathbb{Z}\}$ over n variables and $\theta(n^{3/2})$ clauses satisfying*

1. *The minimal length of a k-DNF resolution refutation of F_n is $2^{n^{\Omega(1)}}$.*

2. *Letting F'_n denote the formula obtained by adding to F_n all symmetry breaking clauses pertaining to symmetries of support size $\leq k+1$, then F'_n has a resolution refutation of length $O(n^{3/2})$.*

The constants hidden by asymptotic notation may depend on k.

Proof technique The method of proof we use is that of *formula substitution*, which has been employed in several instances in the past to show separations between propositional proof systems (examples include [BS02, BOP03, Ngu07, Sch08, BN08]). In a nutshell, it works like this. Start with a formula H over variables $x_1, \ldots, x_n$ that has a short (think polynomial length) refutation in the proof system of interest. Substitute each variable x_i with a formula over a small number k of variables (think $k = 2$), such as the exclusive-or of k variables. We use k new variables for each "substituted" variable x_i appearing in H and denote these new variables by $x_i(1), \ldots, x_i(k)$. Now we have a new formula F in hand that has $k \cdot n$ variables. Next we argue (relying on previous results) that refuting F requires exponential length. The key point is that if we carefully picked the formula we substitute x_i with to be a "very symmetric" formula (see Definition 4.7) then the symmetry breaking clauses of F of support size k will be sufficient to recover all clauses of H using short derivations. To sum, we find that F is "hard-to-refute" but if we add to F all symmetry breaking clauses of support size at most k then the resulting formula is "easy-to-refute".

Relevance to SAT solving The *satisfiability* problem (SAT) is one of great theoretical and practical importance in computer science. In this problem we are given as input a CNF formula F and asked to find an assignment to the variables satisfying (at least) one literal in each clause, or to output "unsatisfiable" in case no such assignment exists. The theoretical importance of this problem stems from the well-known fact that it is a natural NP-complete problem with a rich combinatorial structure. The practical importance arises from the discovery that reducing constraint satisfaction problems to CNF format and feeding this as input to state-of-the-art *SAT solvers* — algorithms that solve instances of the satisfiability problem — is often the most computationally-efficient way to deal with them. Indeed, in recent years many tasks ranging from motion planning to formal verification have been successfully resolved on real-world instances by use of SAT solvers.

Most SAT solvers used in practice, as well as the vast majority of algorithms competing successfully in recent SAT solving competitions [SAT], are based on the successful branch-and-bound method suggested by Davis, Putnam, Loveland and Logemann (and known as DPLL) [DP60, DLL62]. A recently suggested augmentation of DPLL by *clause learning*, suggested by Bayardo and Schrag and by Silva and Sakallah in [BS97, SS96], has been found to perform particularly well on real-world and industry based instances of the satisfiability problem. The term "DPLL SAT solving" actually refers to a large collection of heuristics

which vary according to the way branches are explored. Similarly, "DPLL augmented with clause learning", which we denote from here on by DPLL+, refers to a family of algorithms, members of which differ based upon the methods used to detect and learn clauses.

Proving lower bounds on the running time of all members of the family of DPLL (or DPLL+) SAT solvers on *satisfiable* instances is beyond our current ability because it implies $\mathbf{P} \neq \mathbf{NP}$[1]. But if instead we consider the running time of these algorithms on *unsatisfiable* formulas then proving lower bounds becomes a feasible task. The reason is that the sequence of branches explored by a DPLL SAT solver on an unsatisfiable CNF input formula F corresponds to *treelike resolution* refutation of F. Similarly, the sequence of clauses learned by a DPLL+ algorithm augmented with clause learning, when added to the above-mentioned branch sequence, corresponds to a *resolution* refutation of F. And like-wise, the sequence of branches explored by a k-valued logic branch-and-bound algorithm on F corresponds to a *k-DNF resolution* refutation of F (cf. [JN02]).

In light of this connection between families of SAT solvers and proof complexity, our results show that applying a symmetry breaking preprocessor, even one that is limited to discovering and breaking only symmetries of small support, can have a great impact on the running time of SAT solvers, potentially reducing their running time from exponential to polynomial, or (in the case of DPLL and treelike resolution) even linear (!).

Organization of the rest of the paper After presenting the necessary definitions in the next section, we start in Section 3 with a formal treatment of the automorphisms of a CNF and its symmetry breaking clauses. Section 4 contains the main technical part of this paper as it shows the effect of substitution on the automorphisms of a CNF formula. Section 5 presents the formal proofs of our main results.

2 Preliminaries

2.1 Notation regarding formulas

Throughout this paper x will denote a Boolean variable over $\{0,1\}$. A *literal* over x is either x (which is called "positive") or $\bar{x}$ ("negative"). We say that x and $\bar{x}$ are *opposite* literals over x. When working with n variables we will number them $x_1, \ldots, x_n$ and to facilitate the definition of permutations over literals we shall denote the positive literal over x_i by x_i^0 and the negative one by x_i^1. In other words, these $2n$ literals are indexed by $\{1,\ldots,n\} \times \{0,1\}$.

A *clause* is a disjunction (OR) of literals. Often we shall view a clause as the set of literals appearing in it. We say that a variable x *appears* in a clause C (denoted $x \in C$) if either x or $\bar{x}$ appears in C. Let $\mathrm{vars}(C)$ denote the set of variables appearing in C. The *width* of a clause C, denoted by $|C|$ is the number of literals in it. A formula in conjunctive normal form (CNF) is a conjunction (AND) of clauses and we often view it also as a set of clauses, i.e., as a set of sets of literals. For example, we view the CNF formula $\{(x_1 \vee x_2), (x_3 \vee \bar{x}_4)\}$ as the following set of sets of literals: $\{\{x_1, x_2\}, \{x_3, \bar{x}_4\}\}$. If all clauses of a CNF formula F have width at most d we say that F is a d-CNF formula. Let $|F|$ denote the number of clauses in F and let $\mathrm{vars}(F) = \bigcup_{C \in F} \mathrm{vars}(C)$. Two CNF formulas F, F' over variable sets X, Y respectively are said to be *equivalent* if there exists a bijective mapping g from the literals of X, denoted $L(X)$ to the literals of Y, denoted $L(Y)$ such that *(i)* each pair of opposite literals in $L(X)$ is mapped to a pair of opposite

[1] To see this, notice that if $\mathbf{P} = \mathbf{NP}$ then there exists a polynomial-time DPLL heuristic that uses the satisfying assignment of a satisfiable formula to decide which way to branch on each variable, in such a way that a satisfying assignment is reached with no backtracking at all.

literals in $L(Y)$, and *(ii)* $F' = g(F)$ where $g(F)$ denotes the formula F with each literal x_i^b replaced by $g(x_i^b)$.

A Formula in disjunctive normal form (DNF) is defined, similarly to a CNF formula, as a disjunction of conjunctions of literals and all notations and definitions appearing in the previous paragraph are extended to DNF in the natural way.

A *partial assignment* or *restriction* to a formula F is a partial mapping ρ from $\mathrm{vars}(F)$ to $\{0,1\}$. Letting $\mathrm{dom}(\rho)$ denote the subset of variables that ρ maps to $\{0,1\}$, the value of a literal $x_i^b, b \in \{0,1\}$ under ρ is

$$x_i^b|_\rho = \begin{cases} (\rho(x_i))^b & x_i \in \mathrm{dom}(\rho) \\ x_i^b & \text{otherwise} \end{cases}$$

For G a formula constructed of literals and AND and OR gates of unbounded fan-in (in particular, this class of formulas includes clauses and CNF and DNF formulas) we use the standard definition of the restriction of G by ρ, which is defined by induction on the depth of G as follows:

$$G|_\rho = \begin{cases} x_i^b|_\rho & G = x_i^b \\ 1 & G = \bigvee_{j=1}^m G_i \text{ and } G_{j'}|_\rho = 1 \text{ for some } j' \in \{1,\ldots,m\} \\ 0 & G = \bigwedge_{j=1}^m G_i \text{ and } G_{j'}|_\rho = 0 \text{ for some } j' \in \{1,\ldots,m\} \\ \bigodot_{j=1}^m G_j|_\rho & \text{otherwise, where } G = \bigodot_{j=1}^m G_j \text{ and } \bigodot \text{ denotes either } \bigwedge \text{ or } \bigvee \end{cases}$$

An *assignment* α to F is a partial assignment whose domain is all of $\mathrm{vars}(F)$. The *truth value* of an assignment α over a CNF F, denoted $F(\alpha)$, is the value of F according to the Boolean algebra. That is, $F(\alpha) = 1$ iff for each clause of F, at least one literal is evaluated to 1 by α.

2.2 Proof systems

All proof systems considered here are *sequential*, i.e., a *derivation* of a formula G from a CNF formula F is a *sequence* of lines where each line is either a clause of F or is derived from previous lines by one of the allowed *derivation rules* of the proof system, and the very last line of the derivation is G. The *size* of a derivation is defined to be the number of lines in it. Letting P denote the proof system in which the derivation is conducted, we denote by $S_P(F \vdash G)$ the *minimal size* of a derivation of G from F in the proof system P. A *refutation* of F is a derivation of the fixed contradiction, denoted 0, from F. We let $S_P(F \vdash 0)$ denote the minimal size of a P-refutation of F.

The results of this paper pertain to three basic and well-studied proof systems: *(i) resolution* , *(ii)* the *tree-like* version of resolution, and *(iii)* the extension of resolution known as *k-DNF resolution*. The formal definitions of these three proof systems follow (See [Seg07] for a more information on these and other propositional proof systems.)

Definition 2.1 (Resolution)**.** A *resolution* derivation allows use of the following two derivation rules. In what follows E and F are clauses and x is a variable.

1. *Resolution*: Infer $E \vee F$ from $E \vee x$ and $F \vee \bar{x}$
2. *Weakening*: Infer $E \vee F$ from E

For a CNF F and clause C we denote by $S_R(F \vdash C)$ the minimal size of a resolution derivation of C from F and if F is unsatisfiable then $S_R(F \vdash 0)$ denotes the minimal size of a resolution refutation of F.

We point out that the weakening rule is not essential, as even without it the resolution proof system is complete with respect to refutations, but we add this rule for the sake of simplicity.

Definition 2.2 (Tree-like resolution)**.** A *treelike resolution* derivation allows use of the two derivation rules allowed in the resolution system (Definition 2.1), and has the added requirement that every inferred clause is used at most once as an assumption in the derivation of a new clause in the refutation. Axioms, which are noninferred clauses, may be used more than once. For a CNF F and clause C we denote by $S_T(F \vdash C)$ the minimal size of a treelike resolution derivation of C from F and if F is unsatisfiable then $S_T(F \vdash 0)$ denotes the minimal size of a treelike resolution refutation of F.

The following extension of resolution to lines that are k-DNF formulas was introduced by Krajíček in [Kra01] as an interesting intermediate step between resolution and depth-2 Frege systems.

Definition 2.3 (k-DNF resolution)**.** The *k-DNF resolution* system is a proof system in which lines are k-DNF formulas. The derivation rules are as follows, where A, B denote k-DNF formulas and $l, l_1, ..., l_j$ are literals:

1. *Subsumption*: Infer $A \vee l$ from A.
2. *Cut*: Infer $A \vee B$ from $A \vee \bigwedge_{i=1}^{j} l_i$ and $B \vee \bigvee_{i=1}^{j} \bar{l}_i$.
3. *And-elimination*: Infer $A \vee l_i$ from $A \vee \bigwedge_{i=1}^{j} l_i$.
4. *And-introduction*: Infer $A \vee \bigwedge_{i=1}^{j} l_i$ from $A \vee l_1, ..., A \vee l_j$.

For a CNF F and a k-DNF formula D we denote by $S_{\mathrm{R}[k]}(F \vdash D)$ the minimal size of a k-DNF resolution derivation of D from F and if F is unsatisfiable then $S_{\mathrm{R}[k]}(F \vdash 0)$ denotes the minimal size of a k-DNF resolution refutation of F.

Notice that all derivation rules in the definition above are sound. I.e., every assignment that satisfies the assumption, satisfies the conclusion of the rule as well. The resolution derivation rule is a special case of rule number 2 of k-DNF resolution for $k = 1$.

We say that a proof system P *simulates* another proof system Q if for all pair of formulas F and C we have $S_P(F \vdash C) \leq S_Q(F \vdash C)$. Since every treelike resolution derivation is also a resolution derivation and every resolution derivation is also a k-DNF resolution derivation for $k > 1$ we see that k-DNF resolution simulates resolution, which in turn simulates treelike resolution.

The proof of the following statement is folklore, and we will use it later on in our proofs. For F, G two formulas we say that F *implies* G, denoted $F \models G$ if every assignment that sets F to 1 also sets G to 1.

Claim 2.4 (Implicational completeness of treelike resolution)**.** *Let F be a CNF formula over n variables and let C be a clause. If $F \models C$ then there exists a treelike resolution derivation of C from F of size at most 2^{n+1}.*

We conclude with a statement whose proof is obtained by observing that all derivation rules in the proof systems defined above are invariant under renaming of variables and literals.

Claim 2.5. *If F and F' are equivalent unsatisfiable formulas as defined in Section 2.1 and P is one of the proof systems defined earlier in this section, then $S_P(F \vdash 0) = S_P(F' \vdash 0)$.*

3 The Automorphism Group and Symmetry Breaking Clauses

In this section we discuss the group of automorphisms of a CNF formula F and its symmetry breaking clauses. We show how to find "simple" automorphisms, i.e., ones that leave most variables unchanged. We go on to discuss the partitioning of the assignments into equivalence classes based on the group of automorphisms of F. We define the symmetry breaking clauses of F as those clauses that rule out all but the minimal lexicographic assignment in an equivalence class. In the end of this section we prove that adding to F its symmetry breaking clauses is a sound operation — if F was satisfiable then it remains so even when these new clauses are added as constraints.

3.1 The group of automorphisms of a CNF formula

In this paper we shall consider only *legal literal-permutations* as per the following definition. Since all permutations will be of this form, we will abuse notation and refer to them simply as "permutations". Recall from Section 2.1 that a CNF formula F over n variables is viewed as a set of sets of literals and that literals are indexed by $\{1, ..., n\} \times \{0, 1\}$.

Definition 3.1 (Legal literal-permutation)**.** Let F be a CNF over $X = \{x_1, ..., x_n\}$ and let $L(X) = \left\{x_1^0, x_1^1, \ldots, x_n^0, x_n^1\right\}$ be the set of literals over X. A permutation π of $L(X)$ is said to be a *legal literal-permutation* if it sends pairs of opposite literals to pairs of opposite literals. Formally, for each $i \in \{1, ..., n\}$ we have that $\pi(x_i^0)$ and $\pi(x_i^1)$ are opposite literals. The *order* of π, denoted $\mathrm{ord}(\pi)$, is the minimal integer t such that π composed with itself t times (denoted π^t) is the identity permutation. The *support* of π, denoted $\mathrm{supp}(\pi)$, is the set of variables that are not fixed by π. Formally, $\mathrm{supp}(\pi) = \left\{i \;\middle|\; \pi(x_i^0) \neq x_i^0\right\}$.

A permutation π' of $\{1, \ldots, n\}$ will be associated with the legal literal-permutation π defined by $\pi(x_i^0) = x_{\pi'(i)}^0$ and $\pi(x_i^1) = x_{\pi'(i)}^1$, i.e., indices of variables are permuted according to π' without changing the sign of literals.

Definition 3.2 (Formula automorphisms)**.** Let π be a legal literal-permutation of the variables of a CNF formula F. Denote by F^π the CNF where each clause C of F, viewed as a set of literals, is replaced with the set of literals that is the image of C under π.

A legal literal-permutation π is called an *automorphism* of F if it has the property that F^π equals F as sets, i.e., if every clause of F^π is also a clause of F and vice versa.

We denote by $\mathrm{aut}(F)$ the set of automorphisms of F.

For example, for $F = \{(x_1 \vee x_2), (x_3 \vee x_4)\} = \{\{x_1, x_2\}, \{x_3, x_4\}\}$, the permutation $\pi_1 = ((2, 1), (3), (4))$ which swaps x_1 and x_2 and leaves x_3, x_4 unchanged is an automorphism because $F^{\pi_1} = F$. In contrast, the permutation $\pi_2 = ((1)\,(2, 3)\,(4))$ is not an automorphism because $F^{\pi_2} = \{\{x_1, x_3\}, \{x_2, x_4\}\} \neq F$.

Claim 3.3. *For every formula F, the set of automorphisms* $\mathrm{aut}\,(F)$ *forms a group under composition.*

Proof. First, $\mathrm{aut}(F)$ is closed under composition. Next, the identity permutation is the identity element of the group. Furthermore, composition of automorphisms satisfies associativity and finally, every automorphism has an inverse, namely, its inverse permutation. □

3.2 Symmetry breaking clauses

A legal literal-permutation π over n literals can be viewed as a function over the assignments to variables. Namely, if α is an assignment to $x_1, \ldots, x_n$ then let α^π be defined as the assignment that sets the variable

$\pi(x_i)$ with the value that α assigns to x_i. For example, if $\pi_1 = ((2,1),(3),(4))$ and $\alpha = (1,0,1,0)$ then α^{π_1} is the assignment $(0,1,1,0)$. The following proposition appears in [CGLR96] and we provide its proof for the sake of completeness.

Proposition 3.4. *Let $F = \{C_1, C_2, ..., C_m\}$ be a CNF formula over n variables $X = \{x_1, ..., x_n\}$ and let π be an automorphism of F. Then for any assignment α, $F(\alpha) = F(\alpha^\pi)$.*

Proof. We have $F(\alpha) = F^\pi(\alpha^\pi) = F(\alpha^\pi)$. The first equality follows because we apply the same permutation to literals and to their truth values. The second equality is because $\pi \in \mathrm{aut}(F)$. □

We use Proposition 3.4 to define an equivalence relation on the set of assignments to F. Namely, two assignments α, β are said to be *equivalent* if and only if $\alpha^\pi = \beta$ for some $\pi \in \mathrm{aut}(F)$. Proposition 3.4 implies that for such equivalent α, β we have $F(\alpha) = F(\beta)$. This observation implies that in order to check whether F is satisfiable it is sufficient to check the value of F on a single representative of each equivalence class.

Definition 3.5 (Symmetry breaking clauses)**.** Given a CNF F over variable set X, let $S_1, \ldots, S_t \subseteq \{0,1\}^X$ be the equivalence classes induced by $\mathrm{aut}(F)$. A CNF F' over variable set X is said to be a set of *symmetry breaking clauses* for F if F' has at least one satisfying assignment from S_i for all $i = 1, \ldots, t$.

Notice that Proposition 3.4 implies that F is satisfiable iff $F \bigcup F'$ is satisfiable, where F' is any single set of symmetry breaking clauses F' for F. Notice furthermore that it could so happen that F' and F'' are two distinct sets of symmetry breaking clauses for a satisfiable CNF F, however, $F \bigcup F' \bigcup F''$ is unsatisfiable (see Remark 3.9 for an example).

3.3 Algorithms for adding symmetry breaking clauses of small width

As discussed in the introduction, there are two concerns when attempting to harness the automorphism group of F to speed up a SAT solver's running time on F. The first problem is that of finding elements of $\mathrm{aut}(F)$ and the second problem is that of properly using $\pi \in \mathrm{aut}(F)$ once we have found it. In what follows we offer solutions to both these problems in the limited case of automorphisms of "small" support size. We start by defining an algorithm that runs in time $n^{O(k)}$ and finds all automorphisms of F that have support size at most k.

Algorithm 3.6 (`find-aut`)**.**
Input: *(i)* A CNF formula F and *(ii)* integer k.
Output: The set of automorphisms of F of support size at most k.

`find-aut`(F, k)

1. Initialize $S = \emptyset$.
2. For all legal literal-permutations π over the literals of F
   (a) If $F^\pi = F$ then add π to S.
3. Return S.

Notice that when F has n variables then the running time of `find-aut`(F,k) is bounded by $\binom{n}{k}\cdot k!\cdot 2^k\cdot O(|F|) = O((kn)^k\cdot|F|)$. When k is a small constant (later on we shall mainly work with $k=2$) the running time of this algorithm is polynomial in its input length. Next we define an algorithm that on input F and a set of automorphisms of small support, outputs a set of symmetry breaking clauses. We prove below (in Theorem 3.8) that adding these clauses to F is sound, i.e., will not make a satisfiable formula unsatisfiable. As in [CGLR96], to define the algorithm we require a lexicographic ordering on assignments, which is obtained without loss of generality by ordering the variables as $x_1 < x_2 < \ldots < x_n$. Later on (in Remark 3.9) we explain why such an ordering is critical for maintaining soundness. This ordering of variables also induces a partial order on clauses, where $C < C'$ if both clauses have the same set of variables X and the unique assignment to X that sets C to false is smaller than the unique assignment that sets C' to false. The algorithm will add clauses that will be satisfied only by the minimal assignment of every equivalence class. It can be executed using any legal literal-permutation π but we will use it only with automorphisms.

Algorithm 3.7 (`add-clause`)**.**
Input: *(i)* CNF formula F, *(ii)* legal literal-permutation π.
Output: A set of symmetry breaking clauses to be added to F.

`add-clause`(F,π)

1. `Initialize` $S=\emptyset$.
2. `For each clause` C `with` $\mathrm{vars}(C)=\mathrm{supp}(\pi)$`, in increasing lexicographic order`
   (a) `If` $C\notin S$ `then add` $\left\{C^{\pi^j}\;\middle|\;1\le j<\mathrm{ord}(\pi)\right\}\setminus\{C\}$ `to` S.
3. `Return` S.

Notice that if π has support size k (recall that later on we will consider $k=2$) then the running time of the algorithm is bounded by $O(k\cdot 2^k)$. The following statement is the main result of this section. It says that adding to F the clauses obtained from `add-clause`$(F,\pi_1),\ldots,$`add-clause`(F,π_s) is a sound operation (i.e., it does not turn a satisfiable formula into an unsatisfiable one) as long as $\pi_1,\ldots,\pi_s$ are automorphisms of F.

Theorem 3.8 (Soundness of `add-clause`)**.** *Let F be a CNF and assume $\pi_1,\ldots,\pi_s\in\mathrm{aut}(F)$. Let F'_i be the set of clauses that is returned by* `add-clause`(F,π_i)*. Then $F'=F'_1\bigcup\ldots\bigcup F'_s$ is a set of symmetry breaking clauses for F as per Definition 3.5.*

Consequently, F has a satisfying assignment if and only if $F''=F\bigcup F'$ has a satisfying assignment.

Proof. Let $S_1,\ldots,S_t$ be a partition of the assignments to $\mathrm{vars}(F)$ into equivalence classes induced by $\mathrm{aut}(F)$. We need to show that every equivalence class S_i contains an assignment that satisfies F'. Fix such an equivalence class and let $\alpha_0\in S_i$. Consider the sequence of assignments $\alpha_0,\alpha_1,\ldots$ defined as follows for $t\ge 0$: If α_t satisfies F' terminate the sequence. Otherwise, there exists some $C'\in F'_\ell, 1\le\ell\le s$ such that C' is set to false by α_t. Inspection of `add-clause` reveals that $C'=C^{\pi_\ell^j}$ for some integer ℓ and furthermore by construction we have $C\notin F'_\ell$. In this case set $\alpha_{t+1}=\alpha_t^{\pi_j^{-\ell}}$ and repeat the process with α_{t+1}. That is: α_{t+1} is the assignment that agrees with α_t on all variables except $vars(C)$ and falsifies C.

Notice that all assignments α_t belong to S_i because S_i is invariant under $\mathrm{aut}(F)$. Next, we claim that the sequence must be finite. The key observation is that α_{t+1} is strictly smaller than α_t according to the

lexicographic ordering. To see this, use the notation of the previous paragraph and notice that C' is strictly greater than C according to the lexicographic ordering and α_{t+1} is equivalent to α_t on all variables not in $\mathrm{vars}(C) = \mathrm{vars}(C')$. Furthermore, the assignment of α_t to $\mathrm{vars}(C)$ falsifies C' whereas α_{t+1} falsifies C, i.e., α_{t+1} is strictly smaller than α_t. Now, using the key observation and the fact that the set of assignments is finite we conclude that our sequence is finite, as claimed.

In particular, the very last element in the sequence belongs to S_i and satisfies F'. This shows that F' is symmetry breaking set of clauses.

Consequently, if F is satisfiable and $\alpha_0 \in S_i$ satisfies F, then by Proposition 3.4 the last element in the sequence also satisfies F, and by the previous discussion it also satisfies F'. (If F is unsatisfiable then clearly so is F''). In other words, F is satisfiable iff F'' is satisfiable. This completes the proof. □

Remark 3.9 (The importance of a lexicographic order)**.** The following example shows that if one does not employ a lexicographic order in algorithm `add-clause` then applying the algorithm to two permutations whose support has nonempty intersection can transform a satisfiable formula into an unsatisfiable one. Consider $F = \{\{x_1, x_2, x_3\}, \{\bar{x}_1, \bar{x}_2, \bar{x}_3\}\}$ which is satisfiable and the three automorphism that swap two variables: π_1 which swaps x_1 and x_2, π_2 which swaps x_2 and x_3 and π_3 which swaps x_1 and x_3. If we do not respect a lexicographic order and add arbitrary symmetry breaking clauses, then π_1, π_2, π_3 could respectively introduce the three clauses $\{x_1, \bar{x}_2\}$,$\{x_2, \bar{x}_3\}$,$\{x_3, \bar{x}_1\}$ (the last of which does not respect lexicographic order). One can now verify that adding these clauses to F results in an unsatisfiable formula.

Theorem 3.8 shows that running `add-clause` produces a set of symmetry breaking clauses. Later on we will examine what effect does `add-clause` have on proof length and to this end make the following definition.

Definition 3.10 (Symmetry breaking clauses derived from `add-clause`)**.** For F a CNF formula let $\mathrm{aut}^k(F)$ denote the set of automorphisms of F of support size at most k. (Notice this set is not necessarily a group.) Let $\mathrm{SBC}(F, k)$ denote the set of symmetry breaking clauses output by running `add-clause` on all $\pi \in \mathrm{aut}^k(F)$.

Claim 3.11. *Let F be a CNF formula over variable set X. Let $S_1, \ldots, S_t \subseteq \{0,1\}^X$ be the equivalence classes of assignments satisfying F that is induced by $\mathrm{aut}(F)$. Then the only assignments satisfying $F \cup \mathrm{SBC}(F, n)$ are the lexicographically minimal assignments of $S_1, \ldots, S_t$.*

Proof. Let β be an assignment satisfying $F \cup \mathrm{SBC}(F, n)$. To satisfy F, the assignment β must belong to some S_i. Let α be the lexicographically minimal assignment in S_i. Assume by way of contradiction that $\beta \neq \alpha$. There exists some (nonidentity) permutation $\pi \in \mathrm{aut}(F)$ such that $\alpha^\pi = \beta$. Invoking `add-clause`(F, π) will add the clause set to false by β to the set $\mathrm{SBC}(F, k)$, contradicting our assumption that β satisfies $F \cup \mathrm{SBC}(F, n)$ and completing our proof. □

4 Substitution and local symmetries

4.1 Substitution

All our proofs follow the same outline which is as follows. Suppose P is a proof system. We will start with an "easy" formula F that has n variables and $n^{O(1)}$ clauses and has short P-refutations, i.e., $S_P(F \vdash 0) = n^{O(1)}$. We will transform F using "substitution", to be described below, into a formula F' such that $\mathcal{S}_P(F' \vdash 0) > \exp(n^{\Omega(1)})$. Finally, we will show, using the main technical theorem (Theorem 4.11) that

$S_P((\mathcal{F}' \cup \mathrm{SBC}(\mathcal{F}', k)) \vdash \mathcal{F}) \leq n^{O(1)}$. Combining these two inequalities we obtain our desired separation and show that adding symmetry breaking clauses, even when they pertain to symmetries of constant support size, can lead to an exponential reduction in proof length and associated SAT solving running time.

The transformation of "easy" formulas into "hard" ones goes by *formula substitution* as defined in [BN09]. See also the discussion on "hardness escalation" in [BHP09] and references within. The definition we use is slightly different than that appearing in [BN09], the difference being that here we associate a *canonical* CNF formula with a boolean function f whereas there any pair of CNF formulas that compute f and $\bar{f}$ were sufficient.

Definition 4.1 (Canonical CNF)**.** Let $f : \{0,1\}^k \to \{0,1\}$ be a boolean function. The *canonical CNF* of f over variables $y_1, \ldots, y_k$, denoted $H_f(y_1, \ldots, y_k)$, is defined to be the unique CNF formula over $y_1, \ldots, y_k$ that computes f and consists of clauses of width precisely k. Formally, for $\alpha \in \{0,1\}^k$, letting C_α denote the unique clause of width k that is set to false by α, we have

$$H_f(y_1, \ldots, y_k) = \left\{ C_\alpha \;\middle|\; \alpha \in \{0,1\}^k, f(\alpha) = 0 \right\}.$$

In what follows, for $\mathcal{F}', \mathcal{F}''$ a pair of CNF formulas, each viewed as a set of sets of literals, let $\mathcal{F}' \times \mathcal{F}''$ be the CNF formula defined by

$$\mathcal{F}' \times \mathcal{F}'' = \left\{ C' \vee C'' \;\middle|\; C' \in \mathcal{F}', C'' \in \mathcal{F}'' \right\}.$$

For more explanations and examples of substitution, see [BN09].

Definition 4.2 (Substitution)**.** Let $\mathcal{F}$ be a CNF with m clauses over variables $X = \{x_1, ..., x_n\}$. Let $f : \{0,1\}^k \to \{0,1\}$ be a boolean function and let $\bar{f}$ denote the negation of f, i.e., the boolean function defined by $\bar{f}(\alpha) = 1 - f(\alpha)$ for all $\alpha \in \{0,1\}^k$.

To define the substitution of $\mathcal{F}$ with f we start by making k copies of each variable $x_i \in X$ and denote them by $x_i(1), \ldots, x_i(k)$. The *substitution* of $\mathcal{F}$ with f is the CNF formula $\mathcal{F}[f] = \{C[f] \mid C \in \mathcal{F}\}$ where $C[f]$ is defined as follows. Assuming $x_{i_1}, \ldots, x_{i_s}$ are the variables appearing as positive literals in C and $x_{j_1}, \ldots, x_{j_t}$ are the variables appearing negatively in C, define

$$\begin{aligned} C[f] \;=\; & H_f\left(x_{i_1}(1), \ldots, x_{i_1}(k)\right) \times \ldots \times H_f\left(x_{i_s}(1), \ldots, x_{i_s}(k)\right) \times \\ & H_{\bar{f}}\left(x_{j_1}(1), \ldots, x_{j_1}(k)\right) \times \ldots \times H_{\bar{f}}\left(x_{j_t}(1), \ldots, x_{j_t}(k)\right). \end{aligned}$$

An alternative and equivalent definition is to set $\mathcal{F}[f]$ to the formula obtained as the output of the following two-stage process. First, substitute every occurrence of a positive literal x_i in $\mathcal{F}$ with $H_f\left(x_i(1), ..., x_i(k)\right)$ and replace every occurrence of a negative literal $\bar{x}_i$ in $\mathcal{F}$ with $H_{\bar{f}}\left(x_i(1), ..., x_i(k)\right)$. The resulting formula is of the form $\wedge \vee \wedge \vee$. Use the distributivity of disjunction over conjunction to collapse the two middle levels and transform it into a CNF formula.

Example 4.3. Let $\mathcal{F} = (x_1 \vee \bar{x}_2)$ and $f : \{0,1\}^2 \to \{0,1\}$ be the exclusive or (or parity) of its two inputs. In this case

$$H_f(y(1), y(2)) = ((y(1) \vee y(2)) \wedge (\bar{y}(1) \vee \bar{y}(2)))$$

and

$$H_{\bar{f}}(y(1), y(2)) = ((\bar{y}(1) \vee y(2)) \wedge (y(1) \vee \bar{y}(2))).$$

Substituting $H_f(x_1(1), x_1(2))$ for x_1 and substituting $H_{\bar{f}}(x_2(1), x_2(2))$ for x_2 in $\mathcal{F}$ we obtain

$$((x_1(1) \vee x_1(2)) \wedge (\bar{x}_1(1) \vee \bar{x}_1(2))) \vee ((\bar{x}_2(1) \vee x_2(2)) \wedge (x_2(1) \vee \bar{x}_2(2))).$$

Using distributivity we open the expression above to obtain the following CNF:

$$\begin{aligned} F[f] &= (x_1(1) \vee x_1(2) \vee x_2(1) \vee \bar{x}_2(2)) \wedge (x_1(1) \vee x_1(2) \vee \bar{x}_2(1) \vee x_2(2)) \wedge \\ &\quad (\bar{x}_1(1) \vee \bar{x}_1(2) \vee x_2(1) \vee \bar{x}_2(2)) \wedge (\bar{x}_1(1) \vee \bar{x}_1(2) \vee \bar{x}_2(1) \vee x_2(2)) \,. \end{aligned}$$

We use the following claim from [BN09], the proof of which follows by inspection.

Claim 4.4. *If F is a CNF over n variables consisting of m clauses of width at most w and $f : \{0,1\}^k \to \{0,1\}$ is any boolean function, then $F[f]$ is a CNF with $k \cdot n$ variables that has at most $m \cdot 2^{w \cdot k}$ clauses and each clause is of width at most $w \cdot k$.*

Remark 4.5. Notice that if F is not satisfiable then so is $F[f]$. This is because the first step of the substitution plugs a formula into each input of F and this leaves the resulting formula unsatisfiable. The second step obtains $F[f]$ by applying the distributive law which does not change the function being computed.

4.2 Symmetries of substituted formula

We now begin to explore the effect of substitution on the set of automorphisms of the substituted formula. The following lemma will be crucial later on. It says that automorphisms of H_f will also show up as automorphisms in $F[f]$.

Lemma 4.6. *Let F be a CNF formula over variables $x_1, \ldots, x_n$ and $f : \{0,1\}^k \to \{0,1\}$ be a boolean function. If $\pi \in \mathrm{aut}(H_f(x_i(1), \ldots, x_i(k)))$, letting π' be the permutation on $\mathrm{vars}(H_f)$ whose support is $x_i(1), \ldots, x_i(k)$ and that agrees with π on this support, then $\pi' \in \mathrm{aut}(F[f])$.*

Proof. Let

$$\begin{aligned} F_0 &= \{C \mid (x_i \notin C) \in F\} \\ F_{x_i} &= \{C \mid (x_i \vee C) \in F\} \\ F_{\bar{x}_i} &= \{C \mid (\bar{x}_i \vee C) \in F\}. \end{aligned}$$

Notice

$$F = F_0 \bigcup (F_{x_i} \times \{\{x_i\}\}) \bigcup (F_{\bar{x}_i} \times \{\{\bar{x}_i\}\}) \,.$$

Thus, by definition of the substitution operation we get

$$F[f] = F_0[f] \bigcup (F_{x_i}[f] \times H_f(x_i(1), \ldots, x_i(k))) \bigcup \left(F_{\bar{x}_i}[f] \times H_{\bar{f}}(x_i(1), \ldots, x_i(k))\right) . \tag{1}$$

Consider $(F[f])^{\pi'}$ by examining the effect of π' on each of the three sets appearing in the right hand side of (1). We start with $F_0[f]$. Since $\mathrm{supp}(\pi') \subseteq \{x_i(1), \ldots, x_i(k)\}$ and $F_0[f] \cap \{x_i(1), \ldots, x_i(k)\} = \emptyset$ we conclude $(F_0[f])^{\pi'} = F_0[f]$. We move on to the second term in the right hand side of (1). Since $\mathrm{vars}(F_{x_i}[f]) \cap \{x_i(1), \ldots, x_i(k)\} = \emptyset$ we deduce

$$(F_{x_i}[f] \times H_f(x_i(1), \ldots, x_i(k)))^{\pi'} = (F_{x_i}[f]) \times (H_f(x_i(1), \ldots, x_i(k)))^{\pi'} \,.$$

But since π' acts like π on $x_i(1), \ldots, x_i(k)$ and $\pi \in \mathrm{aut}(H_f(x_i(1), \ldots, x_i(k)))$ we conclude that the right hand side of the previous equation equals $F_{x_i}[f] \times H_f(x_i(1), \ldots, x_i(k))$. The third term of Equation (1) is argued similarly, using the observation that $\pi \in \mathrm{aut}(H_{\bar{f}}(x_i(1), \ldots, x_i(k)))$. This latter fact follows immediately from Proposition 3.4 and Definition 4.1.

We have shown that all three terms of the left hand side of Equation (1) remain unchanged under the application of the permutation π' and this completes our proof. □

4.3 Recovering a formula from its substituted version

We have reached the main part of our paper, where we show how in certain cases a formula can be derived via a short derivation from its substituted version. For this to be possible, we need the substitution function f to be *simple*.

Definition 4.7 (Simple function)**.** Let f be a boolean function $f : \{0,1\}^k \to \{0,1\}$ and let $S_1, \ldots, S_t$ be the equivalence classes of assignments induced by $\mathrm{aut}(H_f(x(1), \ldots, x(k)))$ and similarly let $S'_1, \ldots, S'_{t'}$ be the equivalence classes induced by $\mathrm{aut}(H_{\bar{f}}(x(1), \ldots, x(k)))$. We say that f is *simple* if there exists $r \in \{1, \ldots, k\}$ and $b \in \{0,1\}$ such that

1. For every $i \in \{1, \ldots, t\}$, the lexicographically minimal assignment satisfying S_i also satisfies $(x(r))^b$.
2. For every $i' \in \{1, \ldots, t'\}$, the lexicographically minimal assignment satisfying $S'_{i'}$ also satisfies $(x(r))^{1-b}$.

Example 4.8. If both $\mathrm{aut}(H_f)$ and $\mathrm{aut}(H_{\bar{f}})$ each induce a single equivalence class then clearly f is simple, because there must be some r on which the lexicographically minimal assignment setting f to 1 differs from the lexicographically minimal assignment setting f to 0. In particular, the exclusive-or function over k inputs is simple, because $\mathrm{aut}(H_f)$ includes all literal-permutations that permute variable-indices arbitrarily and flip an even number of literal signs. Consequently, $\mathrm{aut}(H_f)$ induces a single equivalence class that includes all assignments with an odd number of ones.

Example 4.9. Any threshold function on k inputs with threshold $0 < \ell \leq k$, i.e., a function that outputs 1 iff at least ℓ of its inputs are 1, is simple. As special cases we get that the OR and AND functions are simple. First observe that the equivalence classes induced by $\mathrm{aut}(H_f)$ are the sets of assignments of weight exactly ℓ' for $\ell \leq \ell' \leq k$. And the lexicographically minimal assignment in each of these classes must set $x(\ell)$ to 1. A similar argument shows that the lexicographically minimal assignment in an equivalence class induced by $\mathrm{aut}(H_{\bar{f}})$ sets $x(\ell)$ to 0.

Although the following lemma is not needed to prove our main technical theorem, it explains the role played by simple functions in the statement of the main theorem, and its proof may help understand the proof of the main technical theorem. Roughly speaking, the lemma says that a simple function $f : \{0,1\}^k \to \{0,1\}$ has the desirable property that the symmetry breaking clauses $\mathrm{SBC}(H_f, k)$ can be used to derive the lexicographically minimal satisfying assignment of f.

Lemma 4.10. *Let $f : \{0,1\}^k \to \{0,1\}$ be a simple nonconstant function as in Definition 4.7 and let $r \in \{1, \ldots, k\}, b \in \{0,1\}$ be as defined there. Let $\alpha = (\alpha_1, \ldots, \alpha_k) \in \{0,1\}^k$ be the lexicographically minimal assignment setting f to 1. Then the clause $(x(r))^b$ can be derived from*

$$H_f(x(1), \ldots, x(k)) \bigcup \mathrm{SBC}(H_f(x(1), \ldots, x(k)))$$

via a treelike resolution derivation of size at most 2^{k+1}.

Proof. By the implicational completeness of treelike resolution (Claim 2.4) it suffices to prove that every assignment β that satisfies $H_f(x(1), \ldots, x(k)) \bigcup \mathrm{SBC}(H_f(x(1), \ldots, x(k)))$ also satisfies $x(r)^b$. To satisfy $H_f(x(1), \ldots, x(k))$ we must have $f(\beta) = 1$. Let S be the equivalence class of assignments setting f to 1 induced by $\mathrm{aut}(H_f(x(1), \ldots, x(k)))$ to which β belongs. Claim 3.11 implies that $\mathrm{SBC}(H_f(x(1), \ldots, x(k)))$ can only be satisfied by the lexicographically minimal assignment in S. By the simplicity of f this assignment also satisfies $x(r)^b$. This completes the proof.

□

The following theorem is the main technical contribution of this section. Roughly speaking, it says that if F has a short refutation and f is a simple boolean function on a small (e.g., constant) number of inputs, then adding the symmetry breaking clauses of support size $\leq k$ to $F[f]$ results in a formula with short proofs (even when $F[f]$, taken by itself, has no short refutations).

Theorem 4.11 (Main Technical)**.** *Let F be an unsatisfiable d-CNF formula and $f : \{0,1\}^k \to \{0,1\}$ be a nonconstant simple boolean function as in Definition 4.7 and let $r \in \{1,\ldots,k\}, b \in \{0,1\}$ be as defined there. Then for any proof system P that simulates treelike resolution we have*

$$S_P((F[f] \bigcup \mathrm{SBC}(F[f],k)) \vdash 0) \leq S_P(F \vdash 0) \cdot 2^{dk+1}.$$

Proof. The overall idea is to refute $F[f]$ using a P-refutation of F of minimal length, and whenever an axiom of F is needed we shall use Lemma 4.12 below to derive it from $F[f] \bigcup \mathrm{SBC}(F[f],k)$.

The mapping that sends each positive literal x_j (belonging to $\mathrm{vars}(F)$) to the literal $(x_j(r))^b$ (belonging to $\mathrm{vars}(F[f])$) and each negative literal $\bar{x}_j$ to the literal $(x_j(r))^{1-b}$ maps F to a formula F' that is equivalent to F as defined in Section 2.1. This implies by Claim 2.5 that $S_P(F \vdash 0) = S_P(F' \vdash 0)$.

To complete the proof notice that, by Lemma 4.12 below, for every clause $C \in F$ the corresponding clause in F', denoted C', can be derived from $C[f] \bigcup \mathrm{SBC}(C[f],k)$ via a P-derivation of size at most 2^{dk+1}, because P simulates treelike resolution. This completes the proof of the theorem. □

Lemma 4.12. *Let $C = \{x_1,\ldots,x_s,\bar{y}_1,\ldots,\bar{y}_t\}$ be a clause with s positive literals and t negative literals (s and t may equal zero). Let $f : \{0,1\}^k \to \{0,1\}$ be a nonconstant simple boolean function as in Definition 4.7 and let $r \in \{1,\ldots,k\}, b \in \{0,1\}$ be as defined there. Then the clause*

$$C' = \left\{(x_1(r))^b,\ldots,(x_s(r))^b,(y_1(r))^{1-b},\ldots,(y_t(r))^{1-b}\right\}$$

can be derived from $C[f] \bigcup \mathrm{SBC}(C[f],k)$ via a treelike resolution derivation of size at most $2^{k(s+t)+1}$.

Proof. Similar to the proof of Lemma 4.10 above. By the implicational completeness of resolution (Claim 2.4) it suffices to prove that every assignment satisfying $C[f] \bigcup \mathrm{SBC}(C[f],k)$ also satisfies C'. Let β be any assignment to the variables of $C[f]$ that satisfies $C[f] \bigcup \mathrm{SBC}(C[f],k)$. To satisfy $C[f]$ at least one of the following must hold.

1. There exists $\ell \in [s]$ such that $f(\beta(x_\ell(1)),\ldots,\beta(x_\ell(k))) = 1$.
2. There exists $q \in [s]$ such that $f(\beta(y_q(1)),\ldots,\beta(y_q(k))) = 0$.

Assume the former case occurs (the proof in the other case is identical). Claim 3.11 implies that in order to satisfy $\mathrm{SBC}(x_\ell[f],k)$, which is a subset of $\mathrm{SBC}(C[f],k)$, the assignment $(\beta(x_\ell(1)),\ldots,\beta(x_\ell(k)))$ must in fact be the minimal lexicographic assignment in its equivalence class. Since f is simple we conclude β satisfies $(x_\ell(r))^b$ as claimed. □

5 Proofs of main results

With our main technical theorem (Theorem 4.11) in hand we can use existing proof-system separation results to obtain exponential separations between proof-length with and without symmetry breaking clauses for all three proof systems of interest. We start by stating the overall proof strategy formally and then give the detailed proofs for each proof system.

5.1 Proof outline

Let P be a proof system that simulates treelike resolution (all our systems are such). Suppose $\{F_n \mid n \in \mathbb{Z}\}, |F_n| = n^{\Theta(1)}$ is a family of d-CNF formulas (for constant d) and $f : \{0,1\}^k \to \{0,1\}$ is a simple nonconstant boolean function such that:

1. F_n has short P-refutations: $S_P(F \vdash 0) \leq n^{O(1)}$.
2. The formula obtained from substituting F_n with f has no short P-refutations: $S_P(F[f] \vdash 0) \geq 2^{n^{\Omega(1)}}$.

Then Theorem 4.11 implies that $F[f] \bigcup \mathrm{SBC}(F[f], k)$ has short (polynomial length) proofs as long as $k = O(\log n)$, whereas the second condition above says that $F[f]$ requires exponential length proofs.

Remark 5.1. We point out that for certain formulas and proof systems, substitution does not increase proof length. For instance, the family of "pebbling contradictions" used in the proof of Theorem 5.2 below has short (linear size) resolution refutations, and applying substitution to this family with any nonconstant boolean function results in a family of functions that still has linear size resolution refutations (cf. [Ben09]).

5.2 Treelike resolution

Recall that a sequence of polynomial-size formulas $\{F_n \mid n \in \mathbb{Z}\}, |F_n| = n^{\theta(1)}$ is *efficiently constructible* if there exists a polynomial time algorithm that on input 1^n outputs F_n.

Theorem 5.2. *For all integers n there exist efficiently constructible 3-CNF formulas F_n over n variables and $O(n)$ clauses satisfying the following properties.*

1. $S_T(F_n \vdash 0) = 2^{\Omega(n/\log n)}$.
2. $S_T\left(F_n \bigcup \mathrm{SBC}(F_n, 2) \vdash 0\right) = O(n)$.

Proof. Ben-Sasson et al. showed in [BIW04, Ben09] a polynomial time construction of 3-CNF formulas F'_n with n variables and $O(n)$ clauses (called "pebbling contradictions" there, see [Ben09, Definition 3.2]) satisfying the following pair of properties. In what follows let OR_k denote the OR (disjunction) function with k inputs. Notice that f is a nonconstant simple boolean function (see Example 4.9).

1. $\mathcal{S}_T(F'_n \vdash 0) = O(n)$.
2. For all $k > 1$ it holds that $\mathcal{S}_T(F'_n[\mathrm{OR}_k] \vdash 0) \geq 2^{\Omega(n/\log n)}$.

Part 1 is shown in [Ben09, Lemma 3.3] and Part 2 is proved in [BIW04, Theorem 1]. Take, say, $k = 2$ and set $F_n = F'_n[\mathrm{OR}_2]$. Claim 4.4 implies that F_n is a 6-CNF formula with $2n$ variables and $O(n)$ clauses. Theorem 4.11 completes the proof. □

5.3 Resolution

Theorem 5.3. *For all integers n there exist efficiently constructible 6-CNF formulas F_n over n variables and $O(n^{3/2})$ clauses satisfying the following properties.*

1. $S_R(F_n \vdash 0) = 2^{\Omega(\sqrt{n})}$.

2. $S_R(F_n \bigcup \mathrm{SBC}(F_n, 2) \vdash 0) = O(n^{3/2})$.

To prove the theorem we need to work a bit harder by proving the following lemma. In what follows let $\oplus_k$ denote the exclusive-or function on k inputs.

Lemma 5.4. *For all integers n there exist efficiently constructible 3-CNF formulas F'_n over n variables and $O(n^{3/2})$ clauses satisfying the following properties.*

1. $S_R(F'_n \vdash 0) = n^{O(1)}$.
2. *For all $k > 1$ it holds that $S_R(F'_n[\oplus_k] \vdash 0) \geq 2^{\Omega(\sqrt{n})}$.*

Proof of Theorem 5.3. Take $k = 2$ and set $F_n = F'_n[\oplus_2]$ where F'_n is the formula from Lemma 5.4. Recall (from Example 4.8) that $\oplus_2$ is a simple nonconstant boolean function. Claim 4.4 implies that F_n is a 6-CNF formula with $2n$ variables and $O(n^{3/2})$ clauses. Lemma 5.4 and Theorem 4.11 complete the proof. □

Proof of Lemma 5.4. Bonet and Galesi showed in [BG01, Section 3] a polynomial time construction of 3-CNF formulas F'_n (called "modified graph transitivity" and denoted by MGT there) with n variables and $O(n^{3/2})$ clauses satisfying the following properties.

1. $S_R(F'_n \vdash 0) \leq n^{O(1)}$.
2. $W(F'_n \vdash 0) = \Omega(\sqrt{n})$, where $W(F'_n \vdash 0)$ denotes the minimal width of a refutation of F'.

Part 1 is proved in [BG01, Theorem 3.2] and Part 2 is shown in Theorem 3.7 there. Next, we "lift" the width lower bound on F'_n to an exponential size lower bound on $F'_n[\oplus_k]$ using the method described in [Ben09, Section 4] and due to the late Mikhail Alekhnovich [Personal Communication].

Lemma 5.5 (Lifting width lower bounds to size lower bounds)**.** *For every unsatisfiable CNF formula F and integer $k > 1$ we have*

$$S_R(F[\oplus_k] \vdash 0) \geq \left(\frac{4}{3}\right)^{W(F \vdash 0)}.$$

This lemma, combined with the lower bound on $W(F'_n \vdash 0)$ stated in Part 2 above completes the proof. □

Proof of Lemma 5.5. Let π be a resolution refutation of $F[\oplus_k]$ of size S. For ρ a restriction to the variables of $F[\oplus_k]$, let $\pi|_\rho$ denote the sequence of clauses obtained by applying the restriction to each clauses in π and removing all clauses that are set to 1 by ρ. It is well-known that π_ρ is a resolution refutation of $F[\oplus_k]|_\rho$ (see, e.g., [Ben09, Section 4]). By definition the size of $\pi|_\rho$ is at most the size of ρ.

Consider a restriction ρ obtained in the following manner. For each variable x_i appearing in F, pick $j_i \in [k]$ uniformly at random. Then fix random values to $\{x_i(\ell) \mid \ell \neq j_i\}$. In other words, ρ fixes random values to all but one randomly chosen variable in $x_i(1), \ldots, x_i(k)$ and this is done independently for each variable $x_1, \ldots, x_n$ appearing in F.

Inspection reveals that $(F[\oplus_k])|\rho$ is equivalent to F. Since $\pi|_\rho$ is a refutation of F we conclude $W(\pi|_\rho) \geq W(F \vdash 0)$. By assumption there are at most S clauses in π that have width $\geq W(F \vdash 0)$. We claim that the probability that one such clause of width w is not set to 1 by ρ is at most

$$\left(\frac{3}{4}\right)^w. \tag{2}$$

To see this, let k_i denote the number of variables among $x_i(1), \ldots, x_i(k)$ that appear in C (notice k_i may equal zero). Assume without loss of generality that these variables are $x_i(1), \ldots, x_i(k_i)$. Let $E(i)$ be the event "$x_i(1)|_\rho \neq 1$ and $x_i(2)|_\rho \neq 1$ and $\ldots x_i(k')|_\rho \neq 1$". We claim that

$$\Pr[E(i)] \leq \left(\frac{3}{4}\right)^{k_i} \tag{3}$$

which immediately implies (2). To see (3) there are two cases to consider.

- $k_i < k$: The variable $x_i(1)$ is set to 1 by ρ with probability $\frac{k-1}{k} \cdot \frac{1}{2}$. The first term accounts for the probability of ρ setting a value to $x_i(1)$ and the second term is the probability that this value is 1. Thus, $\Pr[x_i(1)|_\rho \neq 1] = 1 - \frac{k-1}{k} \cdot \frac{1}{2}$. Assuming $x_i(1)|_\rho \neq 1$, the variable $x_i(2)$ is set to 1 with probability at most $\frac{k-2}{k-1} \cdot \frac{1}{2}$, where the first term comes from assuming ρ sets a value to $x_i(1)$ (if ρ does not fix a value to $x_i(1)$ then ρ must fix a value to $x_i(2)$ and the probability of $x_i(2)|_\rho \neq 1$ is even smaller). Continuing in this manner we see that

  $$\Pr[E(i)] \leq \prod_{j=1}^{k_i} \left(1 - \frac{k-j}{k-(j-1)} \cdot \frac{1}{2}\right)$$

  and the right hand side is at most $(3/4)^{k_i}$ for all integers $k > k_i \geq 1$.

- $k_i = k$: In this case ρ assigns random values to all but one variable, thus $\Pr[E(i)] \leq 2^{-(k-1)}$ which is at most $(3/4)^k$ for all $k > 1$.

□

5.4 k-DNF resolution

Theorem 5.6. *There exist constants $\beta, \gamma > 0$ and integer $d > 0$ such that the following holds. For integers $k \geq 1$ and n satisfying $k \leq \beta \log n$ there exist efficiently constructible $d(k+1)$-CNF formulas $F_n^{(k)}$ over n variables and $O(2^{dk} \cdot n^{3/2})$ clauses satisfying the following properties.*

1. $S_{\mathrm{R}[k]}(F_n^{(k)} \vdash 0) \geq 2^{\Omega(n^\gamma)}$.
2. $S_{\mathrm{R}[k]}\left(F_n^{(k)} \bigcup \mathrm{SBC}(F_n^{(k)}, k+1) \vdash 0\right) = O(2^{dk} \cdot n^{3/2})$.

Remark 5.7. Setting $k = 1$ in the previous theorem gives another proof of Theorem 5.3, but with a smaller exponential lower bound on proof size of the form 2^{n^γ} (where $\gamma < 1/2$). We include Theorem 5.3 because *(i)* it has a somewhat better exponential lower bound and *(ii)* a different (and arguably simpler) method of proof than what is needed to prove lower bounds on $\mathrm{R}[k]$ for $k > 1$.

Proof. Segerlind showed in [Seg05, Definition 3.1] a polynomial time construction of d-CNF formulas F'_n with n variables and $O(n^{3/2})$ clauses (called "graph ordering principle" and denoted GOP there) satisfying the following pair of properties.

1. $S_R(F'_n \vdash 0) = O(n^{3/2})$.
2. For all $1 \leq k \leq \beta \log n$ it holds that $S_{\mathrm{R}[k]}(F'_n[\oplus_{k+1}] \vdash 0) \geq 2^{n^\gamma}$.

Part 1 above is shown in [Seg05, Lemma 2] and Part 2 appears in Theorem 12 there. Setting $F_n^{(k)} = F'_n[\oplus_{k+1}]$, Claim 4.4 implies that $F_n^{(k)}$ is a $d(k+1)$-CNF with $(k+1)n$ variables and $O(2^{dk} \cdot n^{3/2})$ clauses and $\oplus_k$ is a simple nonconstant boolean function (see Example 4.8). Theorem 4.11 completes the proof. □

Acknowledgements

We thank Daniel Jackson and Ilya Shlyakhter for introducing us to the problem and for many helpful discussions in early stages of this work. We thank Orna Grumberg and Nachum Dershowitz for helpful comments on an earlier draft.

References

[ARMS02] Fadi A. Aloul, Arathi Ramani, Igor L. Markov, and Karem A. Sakallah. Solving difficult sat instances in the presence of symmetry. In *DAC '02: Proceedings of the 39th annual Design Automation Conference*, pages 731–736, New York, NY, USA, 2002. ACM.

[ASM06] F.A. Aloul, K.A. Sakallah, and I.L. Markov. Efficient symmetry breaking for boolean satisfiability. *Computers, IEEE Transactions on*, 55(5):549–558, May 2006.

[Ben09] Eli Ben-Sasson. Size-space tradeoffs for resolution. *SIAM Journal on Computing*, 38(6):2511–2525, 2009.

[BFP96] Cynthia A. Brown, Larry Finkelstein, and Paul Walton Purdom, Jr. Backtrack searching in the presence of symmetry. *Nordic J. of Computing*, 3(3):203–219, 1996.

[BG01] Maria Luisa Bonet and Nicola Galesi. Optimality of size-width tradeoffs for resolution. *Computational Complexity*, 10(4):261–276, December 2001.

[BGS99] Laure BRISOUX, Eric GREGOIRE, and Lakhdar SAIS. Improving backtrack search for sat by means of redundancy. In Ras Z.W. et Skowron A. (eds.), editor, *Proceedings of the 11th International Symposium on Methodologies for Intelligent Systems - ISMIS'99(ISMIS'99)*, pages 301–309, Varsovie, jun 1999. LNCS 1609, Springer Verlag.

[BHP09] Paul Beame, Trinh Huynh, and Toniann Pitassi. Hardness amplification in proof complexity. *CoRR*, abs/0912.0568, 2009.

[BIW04] Eli Ben-Sasson, Russell Impagliazzo, and Avi Wigderson. Near optimal separation of treelike and general resolution. *Combinatorica*, 24(4):585–603, September 2004.

[BN08] Eli Ben-Sasson and Jakob Nordström. Short proofs may be spacious: An optimal separation of space and length in resolution. In *Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science (FOCS '08)*, pages 709–718, October 2008.

[BN09] Eli Ben-Sasson and Jakob Nordström. Understanding space in resolution: Optimal lower bounds and exponential trade-offs. 2009.

[BOP03] Joshua Buresh-Oppenheim and Toniann Pitassi. The complexity of resolution refinements. *Logic in Computer Science, Symposium on*, 0:138, 2003.

[BS94] B. Benhamou and Lakhdar SAIS. Tractability through symmetries in propositional calculus. *Journal of Automated Reasoning(JAR)*, 12:89–102, 1994.

[BS97] Roberto J. Jr. Bayardo and Robert C. Schrag. Using CSP look-back techniques to solve real-world SAT instances. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI'97)*, pages 203–208, Providence, Rhode Island, 1997.

[BS02] Eli Ben-Sasson. Size space tradeoffs for resolution. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC '02)*, pages 457–464, May 2002.

[CGLR96] James M. Crawford, Matthew L. Ginsberg, Eugene Luck, and Amitabha Roy. Symmetry-breaking predicates for search problems. In Luigia Carlucci Aiello, Jon Doyle, and Stuart Shapiro, editors, *KR'96: Principles of Knowledge Representation and Reasoning*, pages 148–159. Morgan Kaufmann, San Francisco, California, 1996.

[Cra92] James M. Crawford. A theoretical analysis of reasoning by symmetry in first-order logic (extended abstract). In *In AAAI Workshop on Tractable Reasoning*, pages 17–22, 1992.

[DLL62] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem proving. *Communications of the ACM*, 5(7):394–397, July 1962.

[DP60] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, 1960.

[Hak85] Armin Haken. The intractability of resolution. *Theoretical Computer Science*, 39(2-3):297–308, August 1985.

[JN02] Jan Johannsen and N. S. Narayanaswamy. An optimal lower bound for resolution with 2-conjunctions. In *Proceedings of the 27th International Symposium on Mathematical Foundations of Computer Science (MFCS '02)*, volume 2420 of *Lecture Notes in Computer Science*, pages 387–398. Springer, August 2002.

[Kra01] Jan Krajíček. On the weak pigeonhole principle. *Fundamenta Mathematicae*, 170(1-3):123–140, 2001.

[Kri85] Balakrishnan Krishnamurthy. Short proofs for tricky formulas. *Acta Informatica*, 22(3):253–275, August 1985.

[Ngu07] Phuong Nguyen. Separating dag-like and tree-like proof systems. In *LICS '07: Proceedings of the 22nd Annual IEEE Symposium on Logic in Computer Science*, pages 235–244, Washington, DC, USA, 2007. IEEE Computer Society.

[SAT] The international SAT Competitions web page. `http://www.satcompetition.org`.

[Sch08] Grant Schoenebeck. Linear level lasserre lower bounds for certain k-csps. In *FOCS '08: Proceedings of the 2008 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 593–602, Washington, DC, USA, 2008. IEEE Computer Society.

[Seg05] Nathan Segerlind. Exponential separation between res(k) and res($k+1$) for $k \leq \epsilon \log n$. *Information Processing Letters*, 93(4):185–190, 2005.

[Seg07] Nathan Segerlind. The complexity of propositional proofs. *Bulletin of Symbolic Logic*, 13(4):482–537, December 2007.

[Shl07] Ilya Shlyakhter. Generating effective symmetry-breaking predicates for search problems. *Discrete Applied Mathematics*, 155(12):1539–1548, 2007.

[SS96] João P. Marques Silva and Karem A. Sakallah. GRASP - a new search algorithm for satisfiability. In *ICCAD*, pages 220–227, 1996.

[Urq99] Alasdair Urquhart. The symmetry rule in propositional logic. *Discrete Applied Mathematics*, 96-97:177–193, 1999.

[Zha01] Jian Zhang. Automatic symmetry breaking method combined with sat. In *SAC*, pages 17–21. ACM, 2001.

ECCC ISSN 1433-8092

http://eccc.hpi-web.de