Electronic Colloquium on Computational Complexity, Revision 1 of Report No. 108 (2013)

Beating Exhaustive Search for Quantified Boolean Formulas and Connections to Circuit Complexity

Rahul Santhanam* Ryan Williams†

Abstract

We study algorithms for the satisfiability problem for quantified Boolean formulas (QBFs), and consequences of faster algorithms for circuit complexity.

- We show that satisfiability of quantified 3-CNFs with m clauses, n variables, and two quantifier blocks (one existential block and one universal) can be solved deterministically in time $2^{n-\Omega(\sqrt{n})} \cdot \mathrm{poly}(m)$. For the case of multiple quantifier blocks (alternations), we show that satisfiability of quantified CNFs of size $\mathrm{poly}(n)$ on n variables with q quantifier blocks can be solved in $2^{n-n^{1/(q+1)}} \cdot \mathrm{poly}(n)$ time by a zero-error randomized algorithm. These are the first provable improvements over brute force search in the general case, even for quantified polynomial-sized CNFs with two quantifier blocks.

  A second zero-error randomized algorithm solves QBF on circuits of size s in $2^{n-\Omega(q)} \cdot \mathrm{poly}(s)$ time when the number of quantifier blocks is q.
- We complement these algorithms by showing that improvements on them would imply new circuit complexity lower bounds. For example, if satisfiability of quantified CNF formulas with n variables, $\mathrm{poly}(n)$ size and at most q quantifier blocks can be solved in time $2^{n-n^{\omega q(1/q)}}$, then the complexity class NEXP does not have $O(\log n)$ depth circuits of polynomial size. Furthermore, solving satisfiability of quantified CNF formulas with n variables, $\mathrm{poly}(n)$ size and $O(\log n)$ quantifier blocks in time $2^{n-\omega(\log(n))}$ time would imply the same circuit complexity lower bound. The proofs of these results proceed by establishing strong relationships between the time complexity of QBF satisfiability over CNF formulas and the time complexity of QBF satisfiability over arbitrary Boolean formulas.

1 Introduction

The satisfiability (SAT) problem for Boolean formulas is the canonical NP-complete problem. Despite its apparent worst-case intractability, nowadays the ability to solve most SAT instances arising in practice is well-known. Over the past two decades, there have been many substantial advances in SAT solvers that led to the current state of affairs that "SAT is generally easy in practice" [MZ09]. On the theoretical side, there are many known SAT algorithms which provably solve the problem faster than brute force search for formulas in conjunctive normal form (CNF) [MS85, PPZ97, Pud98, PPSZ98, Sch99, Sch05, DW06, CIP09, IMP12].

The quantified Boolean formula (QBF) problem is the analogue of SAT for the larger complexity class PSPACE. Variables can have arbitrary quantification (existential or universal), and the problem is to determine whether a given quantified formula is true or false. QBF would potentially have a much wider range of applications than SAT, if only we understood more about how to solve it. The best known QBF solvers can only tackle a very limited range of the problem space [Zha06, GIB09].

Moreover, in theory, comparatively very little is known concerning general worst-case algorithms for QBF. Williams [Wil02] showed that QBFs over general CNF formulas with m clauses are solvable in $O(1.71^m)$ time, and demonstrated that 3-CNF QBFs with two quantifier blocks can be solved in $O(2^{n-\varepsilon n})$ time (where n is the number of variables) for a constant $\varepsilon > 0$ depending on the clause-variable ratio m/n (however, $\varepsilon \to 0$ as $m/n \to 2$). Santhanam [San10] showed that for every $c \geq 1$, there is a $\delta < 1$ such that Formula-SAT on m clauses and n variables can be solved in $2^{\delta n}$ time, and extended his algorithm to solving QBFs over CNF formulas in $2^{n-\Omega(n/\log n)}$ time when the maximum number of occurrences of any variable is bounded above by a constant. Calabro, Impagliazzo, and Paturi [CIP10] give less-than-2^n

*University of Edinburgh. Supported by the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement no. 615075.

†Stanford University. Supported in part by a David Morgenthaler II Faculty Fellowship, and NSF CCF-1212372. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

ISSN 1433-8092

algorithms for two special cases of QBF: the case where every clause contains at most one existential variable, and the case where every clause contains at least one universal literal.[1]

So although improved algorithms were known for quantified Boolean CNFs with cn clauses and n variables (for small c), slightly more general problems have remained open. Even for CNF formulas with n variables and $\text{poly}(n)$ clauses, it was open whether QBF was solvable in faster than 2^n time on formulas of the form

$$(\forall x_1)\cdots(\forall x_k)(\exists x_{k+1})\cdots(\exists x_n)\phi$$

where ϕ is 3-CNF. Such formulas are said to have *two quantifier blocks*, or *one quantifier alternation*. Calabro, Impagliazzo, and Paturi [CIP10] gave evidence that even this special case is very hard to solve: they showed that general CNF satisfiability on n variables can be reduced in subexponential time to quantified k-CNF on $n+O(n^{1/(k-1)})$ variables with two quantifier blocks. Therefore, a 1.999^n time algorithm for quantified 3CNF would imply similar results for CNF-SAT, resolving a longstanding open question.[2]

In this paper, we present new algorithms for solving QBF on CNF and DNF formulas that run faster than brute force, as well as interesting hardness reductions demonstrating that these problems are surprisingly powerful. Our algorithms exploit known algorithmic techniques for satisfiability together with some new methods of analysis. Our hardness results show that quantified Boolean CNFs are a highly expressive class of logical formulas compared to the usual CNFs, giving a partial explanation for why QBFs in practice are so much more difficult to solve.

Beating Exhaustive Search for QBF Satisfiability. We first consider quantified 3-CNFs with two quantifier blocks. No non-trivial algorithms were known for this case for 3-CNFs of non-linear size. We give an algorithm that beats exhaustive search by a considerable margin:

THEOREM 1.1. *Satisfiability of quantified 3-CNF (or 3-DNF) formulas with two quantifier blocks, n variables, and m clauses (or disjuncts) can be solved deterministically in time* $\text{poly}(m)\cdot 2^{n-\Omega(\sqrt{n})}$.[3]

The algorithm of Theorem 1.1 has a branching/backtracking component, and performs a case analysis based on the number of existential and universal variables in the formula. (More precisely, in one case, a branching strategy occurs; in two other separate cases, we reduce the remaining instance to CNF-SAT and apply existing SAT algorithms.) To analyze the backtracking component, a tricky multivariate recursion arises, which we finesse by taking a combinatorial approach to bound the number of leaves in the recursion tree.

Theorem 1.1 is an interesting complement to the hardness reduction of Calabro, Impagliazzo, and Paturi [CIP10], who reduce arbitrary CNF SAT on n variables and $O(n)$ clauses to an instance of the above problem with only $n+O(\sqrt{n})$ variables. (However, we do not currently know any direct implications of beating the $2^{n-\sqrt{n}}$ time bound of Theorem 1.1.)

Next, we consider satisfiability of quantified CNFs with multiple alternations. We show that for n-variable CNFs with significantly less than $\log n$ quantifier blocks, there are algorithms which can definitively beat exhaustive search.

THEOREM 1.2. *Satisfiability of quantified CNF (or DNF) formulas with q quantifier blocks, n variables, and* $\text{poly}(n)$ *clauses (or disjuncts) can be solved probabilistically with zero error in time* $\text{poly}(n)\cdot 2^{n-\Omega(n^{\delta_q})}$, *where* $\delta_q = 1/(q+1)$.

Theorem 1.2 is shown using the algorithmic paradigm of *truth table generation* [Wil11]. The idea is that interesting circuit analysis algorithms sometimes follow from interesting solutions to the task: *given a circuit C of size s on n variables, output the truth table of the function computed by C.* This can be done trivially in time $2^n\cdot\text{poly}(s)$; when is there a faster algorithm? We show how non-trivial truth table generation for constant-depth circuits can be applied to derive non-trivial algorithms for satisfiability of quantified CNFs, and achieve non-trivial truth table generation for constant-depth circuits by modifying a constant-depth circuit analysis algorithm of Impagliazzo, Matthews, and Paturi [IMP12]. From Theorem 1.2, it follows that quantified CNFs with at most $o(\log n/\log\log n)$ quantifier blocks can be solved in time $2^{n-\omega(\log n)}$.

Next, we observe that for formulas with asymptotically *more* than $\log n$ quantifier blocks, a simple randomized algorithm beats 2^n time:

[1] In slightly related work, Williams [Wil14] recently showed that checking first-order formulas with k quantifiers on arbitrary n-node graphs is possible in $n^{k-1+o(1)}$ time when k is a sufficiently large constant, improving over the $O(n^k)$ of exhaustive search. While that work indeed addresses quantified formulas, the setting is rather different (a *constant* number of variables over a *non-Boolean* domain) and the algorithms apply fast matrix multiplication (which we avoid here).

[2] The Strong Exponential Time Hypothesis essentially posits that k-SAT does not have a 1.999^n time algorithm for all constants k [IP01, CIP09, DW10]. Nevertheless, it is also known that even minor improvements over exhaustive search in SAT algorithms, bounds such as $O(2^n/n^4)$, can lead to proofs of circuit complexity lower bounds [Wil10, Wil11, Wil13].

[3] We use $\text{poly}(n)$ to denote arbitrary polynomial factors in n.

THEOREM 1.3. *Let $q : \mathbb{N} \to \mathbb{N}$ satisfy $q(n) < n$ for all n. Satisfiability of quantified Boolean circuits on n variables, where there are $q(n)$ quantifier blocks and the circuits have size* $\mathrm{poly}(n)$*, can be solved probabilistically with zero-error in* $O(\mathrm{poly}(n)2^{n-\Omega(q(n))})$ *time.*

Theorems 1.2 and 1.3 show how to beat brute force on quantified CNFs with up to $O\left(\frac{\log n}{\log\log n}\right)$ quantifier blocks, and those with $\omega(\log n)$ quantifier blocks, respectively. This raises the question of whether there is a weakness in our ways of reasoning about QBF, or whether the case of $O(\log n)$ quantifier blocks is especially difficult.

The Log-Alternation Barrier. Surprisingly, we show that improving over exhaustive search for the case of $O(\log n)$ blocks/alternations is indeed difficult! We first show that beating brute force on n-variable QBF instances with $O(\log n)$ quantifier blocks, even in CNF form, would imply faster algorithms for satisfiability of arbitrary polynomial-size Boolean formulas over arbitrary gates of fan-in two. More precisely, we show how to efficiently express poly-size Boolean formulas as QBFs in CNF with $O(\log n)$ quantifier blocks and $n + O(\log n)$ variables.

THEOREM 1.4. *There is a polynomial time algorithm that takes any Boolean formula of n inputs and s size and outputs an equivalent quantified CNF instance of $n + O(\log s)$ variables, $O(s^{10})$ clauses, and $O(\log s)$ quantifier blocks.*

It follows that, if quantified CNF formulas of polynomial size are solvable in c^n time, then *all* quantified Boolean formulas (over AND/OR/NOT) of polynomial size are solvable in $c^{n+o(n)}$ time. Combining Theorem 1.4 with a reduction of [CIP09] (cited in this paper as Theorem 4.1), we obtain:

COROLLARY 1.1. *There is a polynomial time algorithm that, for every fixed k, takes any Boolean formula of n inputs and s size and outputs a logically equivalent quantified k-CNF instance of $n + O(s^{10/(k-1)})$ variables,* $\mathrm{poly}(s)$ *clauses, and $O(\log s)$ quantifier blocks.*

Applying work of Williams [Wil11], it follows that faster solution of quantified k-CNFs with $O(\log n)$ quantifier blocks would imply that NEXP does not have $\mathrm{poly}(n)$-size $O(\log n)$-depth circuits, a major open problem in circuit complexity:

COROLLARY 1.2. *If for all k, quantified CNF with n variables, n^k clauses, and $k \log n$ quantifier blocks can be solved in zero-error probabilistic $2^n/n^k$ time, then* $\mathsf{NEXP} \not\subset \mathsf{NC}^1/\mathrm{poly}$.

COROLLARY 1.3. *If there is an $\varepsilon > 0$ such that for all k, quantified k-CNF with n variables, n^k clauses, and $k \log n$ quantifier blocks can be solved in zero-error probabilistic 2^{n-n^ε} time, then* $\mathsf{NEXP} \not\subset \mathsf{NC}^1/\mathrm{poly}$.

That is, such QBF algorithms imply that NEXP does not have $O(\log n)$ depth circuits of bounded fan-in.

For the case where the number of quantifer blocks is constant, it is natural to ask if the $n^{\Omega(1)}$ savings in the exponent of Theorem 1.2 can be improved upon, without proving new lower bounds. Again, we can give a negative answer by showing that even quantified CNFs with a *constant* number of quantifier blocks can be very expressive. Extending Theorem 1.4, we show that every formula of polynomial size can be logically expressed as a quantified CNF with $n + O(n^{1/k})$ variables and only $O(k)$ quantifier blocks.

THEOREM 1.5. *Let $k, r > 0$ be any integers. There is a polynomial-time algorithm that takes any Boolean formula of n inputs and depth $r \log(n)$, and outputs an equivalent quantified CNF instance of $n + O(n^{1/k})$ variables, size* $\mathrm{poly}(n)$*, and $2kr$ quantifier blocks.*

The proof of Theorem 1.5 appears in Section 5. Since any formula of size s can be expressed as a formula of depth $O(\log s)$, the parameter r in the theorem can be made $O(1)$, yielding $O(k)$ quantifier blocks. As a consequence:

COROLLARY 1.4. *If quantified CNF with n variables,* $\mathrm{poly}(n)$ *clauses, and q quantifier blocks can be solved in zero-error probabilistic time $2^{n-n^{\omega_q(1/q)}}$, then* $\mathsf{NEXP} \not\subseteq \mathsf{NC}^1/\mathrm{poly}$. [4]

The proof is in Section 5. We conclude that any substantial improvement in the running time of our algorithms would imply new lower bounds in circuit complexity. We do not wish to suggest that such algorithms do not exist, but rather emphasize that they will have very interesting consequences. Indeed, designing better QBFSAT algorithms might be a route to getting better lower bounds.

[4] Here, $\omega_q(1/q)$ is shorthand for any function $f(q)$ that grows strictly faster than $1/q$.

2 Preliminaries

Given a function $s : \mathbb{N} \to \mathbb{N}$, let $\mathsf{SIZE}(s)$ denote the class of Boolean functions with bounded fan-in circuits of size $O(s)$, and $\mathsf{FormulaSIZE}(s)$ denote the class of Boolean functions with fan-in two formulas of size $O(s)$ over the De Morgan basis (AND/OR/NOT). Given a depth function $d : \mathbb{N} \to \mathbb{N}$, let $\mathsf{DEPTH}(d)$ denote the class of Boolean functions with fan-in 2 Boolean circuits of depth d over the De Morgan basis.

LEMMA 2.1. ([SPI71, BB94]) *Let s be a size function such that $n \leq s(n)$ for all n. Then* $\mathsf{FormulaSIZE}(s) \subseteq \mathsf{DEPTH}(2.1 \log(s))$.

As we will consider several quantified formula problems, it is important to establish notation distinguishing between them. We define QB-kCNF, QB-CNF, QB-FORMULAS to be the quantified Boolean formula problems over k-CNF predicates, arbitrary CNF predicates, and formula predicates, respectively. The case of bounded number of quantifier blocks is often studied in complexity theory and logic:

DEFINITION 2.1. *A quantified Boolean formula ϕ has q* quantifier blocks *or $q-1$* alternations *if it has the form*

$$\phi = (Q_1\ x_1, \ldots, x_{t_1})(Q_2\ x_{t_1+1}, \ldots, x_{t_1+t_2}) \cdots$$

$$(Q_q\ x_{t_1+\cdots+t_{q-1}+1}, \ldots, x_{t_1+\cdots+t_q})F,$$

where each $Q_i \in \{\exists, \forall\}$.

The problems q-QB-kCNF, q-QB-CNF, and q-QB-DNF denote the restriction of the QBF problem over these predicates to formulas with at most q quantifier blocks.

For SAT, the choice of logical predicate can play a major role in the time complexity of solving the problem: for every k there is a $\delta < 1$ such that k-SAT is in $2^{\delta n}$ time (e.g., [MS85]), but no 1.999^n time algorithm is known for general CNF-SAT. Indeed the Strong Exponential Time Hypothesis [IP01, CIP09] posits that none exists. For satisfiability problems over more expressive predicates such as arbitrary formulas, there is no algorithm known to run in time faster than $O(2^n)$ for polynomial-size formulas. In fact if Formula SAT were solvable in $O(2^n/n^{10})$ time, then NEXP would not be in NC^1/poly [Wil11].

In contrast, our hardness results show that quantified k-CNFs are essentially *equivalent* in time complexity to quantifiers over arbitrary formulas. This is a very different picture from the complexity of SAT. It is well-known that 3-SAT can be solved in less than $O(1.4^n)$ time [PPSZ98, DH09, MS11, Her11], and that CNF-SAT on poly(n) clauses can be solved in $2^{n-n/O(\log n)}$ time [Sch05, CIP06]. However it is believed that CNF-SAT cannot be solved in $O(1.9^n)$ (this is implied by the Strong Exponential Time Hypothesis [IP01, CIP09]).

3 Algorithmic Results

3.1 Quantified Formulas with Two Quantifier Blocks Our first result is a non-trivial upper bound for satisfiability of quantified 3-CNFs with two quantifier blocks. No non-trivial upper bounds for this case were known previously. The algorithm is far from straightforward: it is a recursive branching algorithm augmented with a couple of novel "base cases" where a CNF SAT solver takes over and finishes the job. The cases for branching are chosen carefully so that the size of the recursion tree can be non-trivially bounded.

We need the following CNF satisfiability algorithm.

THEOREM 3.1. ([DW06], BUILDING ON [SCH05]) *There is a deterministic algorithm A solving satisfiability of CNFs with m clauses and n variables in time $O(\text{poly}(m)2^{n-n/(1+\log(m))})$.*

REMINDER OF THEOREM 1.1 *Satisfiability of 2-QB-3CNF (or 2-QB-3DNF) with n variables and* poly(n) *clauses can be solved deterministically in time $2^{n-\Omega(n^{1/2})}$.*

Proof. We will run a recursive algorithm R on the given 3CNF formula with n variables and two quantifier blocks. In the following, let n denote the number of variables in the *original* formula fed to the algorithm R. (Although the number of variables in the current formula may decrease over multiple calls to R, the parameter n stays the same throughout the algorithm.) Let $\epsilon > 0$ be a constant which we fix later. Here is the algorithm R:

0. If there is a clause containing only universal literals or an empty clause, return 0. If there are no clauses left to satisfy, return 1. If there are no universal literals in the formula, the instance is a 3SAT formula. If the 3SAT is satisfiable then return 1 else return 0.

1(a). Let u denote the number of universal variables and e be the number of existential variables. If $e > \sqrt{n}$, then try all 2^{u-e} assignments to the universals and solve the remaining 3CNF in 1.4^e time, using (for example) the deterministic 3-SAT algorithm of Moser and Scheder [MS11]. This case takes $O^\star(2^u 1.4^{-e}) = 2^{n-\Omega(\sqrt{n})}$ time.

1(b). *[At this point, the number of existential variables is at most $\sqrt{n}$.]* Suppose there is a clause of length three with two universal literals and one existential. Let it be $(u_i \vee u_j \vee e_k)$, where u_i, u_j are universal and e_k is existential. Perform the three recursive calls:

- Set $(u_i = 1)$ and call the algorithm R on the formula,
- Set $(u_i = 0)$ and $(u_j = 1)$ and call R,
- Set $(u_i = 0)$, $(u_j = 0)$, $(e_k = 1)$ and call R.

If all three calls return 1 then return 1, else return 0.[5]

1(c) *[At this point, all clauses contain at most one universal literal.]* If $u < \epsilon n$, then solve the QBF using brute force search. Otherwise, make a DNF with 2^e conjuncts, consisting of an OR over all variable assignments to the e remaining existential variables, and each conjunct is an AND over the remaining universal literals in each clause, given a fixed existential variable assignment. This DNF has $O(2^e \cdot u)$ clauses, and u variables. The tautology problem for this DNF can be solved in $O^\star(2^{u-\Omega(u/\log(2^e))}) \leq O^\star(2^{u-\Omega(u/e)})$ time using the CNF-SAT algorithm of Theorem 3.1. Return 1 if the DNF is a tautology, else return 0.

Let $T_e(u)$ be the running time of the algorithm R on a formula with n variables initially, and u universal variables and e existential variables. Every run of the algorithm R on a given formula can be expressed as follows. Either $e > \sqrt{n}$ at the beginning of the algorithm, in which case the algorithm R runs in $2^{n-\Omega(\sqrt{n})}$ time and halts, or $e \leq \sqrt{n}$ and we have a recursion tree representing the recursive calls in case 1(b). (Note that if $e \leq \sqrt{n}$ in the very first call to R, then we also have $e \leq \sqrt{n}$ in every subformula, so only cases 1(b) and 1(c) will be applied after that.) At the leaves of the recursion tree (when there is at most one universal literal in each clause), a procedure of $2^{u-\Omega(u/e)}$ time is applied; we associate each leaf with a *cost*, namely the corresponding running time at that leaf. The *running time* for the recursion tree is bounded by the sum of the costs over all leaves of the recursion tree.

We term the three recursive calls in case 1(b) calls of type A, type B and type C respectively. We first bound the size N, i.e., the number of leaves, of the recursion tree. We classify leaves according to how many calls of type C occur on the path from root to leaf. For each i such that $0 \leq i \leq \sqrt{n}$, let $f(i)$ be the number of leaves for which i calls of type C occur on the path from root to leaf. Note that at most $\sqrt{n}$ calls of type C occur on any path from root to leaf, as $e \leq \sqrt{n}$ at the root and e decreases by 1 for each call of type C.

Observe that $f(0) = O(\phi^u) = O(\phi^n)$, where ϕ is the golden ratio, since if no calls of type C occur, there are only calls of type A and type B, the first of which decreases u by 1 and the second of which decreases u by 2. Thus the recurrence for $f(0)$ as a function of u is the same as the Fibonnaci recurrence.

In general, we have that $f(i) \leq \binom{n}{i} O(\phi^u)$, as there are $\binom{n}{i}$ ways of choosing the i levels of the recursion tree at which calls of type C are made, and among these, the size of the recursion tree is largest, namely $O(\phi^u)$, when all calls of type C are made before calls of type A and type B. Thus we have that

$$N \leq \sum_{i=0}^{i=\sqrt{n}} f(i) \leq 2^{o(n)} \phi^u \leq 2^{n-\Omega(n)},$$

since $\binom{n}{i} = 2^{o(n)}$ for $i \leq \sqrt{n}$, and $\phi^u \leq \phi^n \leq 2^{n-\Omega(n)}$. Let $\delta > 0$ be a constant such that $N \leq 2^{n-\delta n}$.

We now partition the set of leaves into *deep* leaves and *shallow* leaves, defined as follows. Let $\gamma = \delta/2$. Deep leaves are those which occur at depth at least $(1-\gamma)n$, and shallow leaves are leaves which are not deep. We will account for the number of deep leaves and shallow leaves separately.

[5] A reference note: this branching step originally appears in Williams [Wil02], who used it in a (weaker) QB-3CNF algorithm. Its correctness follows because in the third branch, the two universals have forced the existential to a value. Note that in (for example) a 4-CNF clause with two universals and two existentials, we could not necessarily perform such a branching.

First we analyze the deep leaves. Consider any deep leaf at depth $d \geq (1-\gamma)n$. The cost of any deep leaf is at most $2^{\gamma n}$. Since there are at most $2^{n-\delta n}$ total leaves in the recursion tree, there are at most $2^{n-\delta n}$ deep leaves, and the total cost of all such leaves is at most $2^{n-\delta n+\gamma n} = 2^{n-\Omega(n)}$.

Now set $\epsilon = \delta/4$. We partition the shallow leaves into two classes – *heavy* and *light* leaves: heavy leaves are shallow leaves with $u > \epsilon n$, and light leaves are shallow leaves which are not heavy.

The cost of any light leaf is at most $2^{\epsilon n}$. Because light leaves are shallow, there are at most $2^{(1-\gamma)n}$ light leaves, and hence the cumulative cost of such leaves is at most $2^{n-\gamma n+\epsilon n} = 2^{n-\delta n/4} = 2^{n-\Omega(n)}$.

Finally, we consider the heavy leaves. The cost of a heavy leaf at depth d is at most

$$2^{u-\Omega(u/e)} \leq 2^{n-d-\Omega(\sqrt{n})},$$

since for such leaves we have $u \leq n-d$, $u \geq \epsilon n$ and $e \leq \sqrt{n}$. Thus the cumulative cost of all heavy leaves is at most $2^{n-\Omega(\sqrt{n})}$.

Summing up all contributions, the total cost of all leaves of the recursion tree is at most $O(2^{n-\Omega(\sqrt{n})})$, establishing the running time of the algorithm. □

3.2 Quantified Formulas with a Bounded Number of Quantifier Blocks The algorithm and analysis of Theorem 1.1 do not seem to extend naturally to CNFs of larger width and to more quantifier blocks. For example, in a 4-CNF, one may have a clause with two universally quantified variables and two existentially quantified variables. The branching step 1(b) doesn't naturally extend to that case: we need to quantify over all the other universal variables, before setting the two existentials – the existentials are not yet forced to certain values.

Instead, our algorithm for q-QB-CNF (quantified CNFs with a bounded number of quantifier blocks q), when $q > 2$, applies the function enumeration paradigm based on a recent result on satisfiability of constant-depth circuits [IMP12].

THEOREM 3.2. ([IMP12]) *There is a probabilistic zero-error algorithm A solving satisfiability of constant-depth AND/OR circuits of n variables, s size, and d depth, in time $O(s \cdot 2^{n-\Omega(n/(\log s)^{d-1})})$.*

A key component is a method for computing the truth table of a constant-depth circuit more efficiently. Computing the truth table of a constant-depth circuit of size s over n variables can be done trivially in $O(2^n \cdot s)$ time; we can do better using ideas of [IMP12].

We require some definitions. A *restriction over n bits* is a string $y \in \{0,1,\star\}^n$. A bit string x is said to *complete a restriction y* if x agrees with y on all non-$\star$ coordinates. Given a set S of restrictions over n bits, we say that S *partitions the n-bit cube* if for each bit string $x \in \{0,1\}^n$, there is precisely one restriction $y \in S$ such that x completes y.

LEMMA 3.1. *There is a probabilistic zero-error algorithm E which, given an unbounded fan-in circuit of size s and depth d on n variables, outputs the truth table corresponding to the circuit in time $O((2^n + s2^{n-\Omega(n/(\log(s)^{d-1})})\text{poly}(n))$.*

Proof. Theorem 3.2 is proved in [IMP12] by showing that there exists a probabilistic zero-error algorithm B which, given an unbounded fan-in circuit C of size s and depth d on n variables, outputs a set S of restrictions over n bits partitioning the n-bit cube, as well as a Boolean function f on S, such that for each input $x \in \{0,1\}^n$, $C(x)$ is equal to $f(y)$, where y is the unique restriction in S such that x completes y. Furthermore, the algorithm B runs in $\text{poly}(n) \cdot s \cdot 2^{n-\Omega(n/(\log(s)^{d-1})}$ time, hence $|S|$ is upper bounded by the same quantity.

We define algorithm E as follows. Suppose E is given a circuit C over n bits, as input. E has random access to a table T of length 2^n bits, which are all initially zero. At the end of E's computation, T will hold the truth table of C.

E first runs B on C to obtain S and f. Then for each restriction y in S, E computes the set $X(y) \subseteq \{0,1\}^n$ of all n-bit strings that complete y. For a restriction y with k stars, this can be done in time $2^k \cdot \text{poly}(n)$ time. E then evaluates f on y, and updates the values of all bits in T corresponding to strings in $X(y)$ to $f(y)$. This update takes time $|X(y)| \cdot \text{poly}(n) \leq 2^k \cdot \text{poly}(n)$. After all elements y of S have been processed, E halts and outputs T.

To analyze the running time, notice that

$$\sum_{y \in S} |X(y)| = 2^n,$$

since S partitions the n-bit Boolean hypercube. Therefore, the running time of E is bounded by the running time of B, plus $\sum_{y \in S} |X(y)| \cdot \text{poly}(n) \leq 2^n \cdot \text{poly}(n)$. Therefore the running time of E is at most $(2^n + |S|) \cdot \text{poly}(n) \leq (2^n + s \cdot 2^{n-\Omega(n/(\log(s)^{d-1})}) \cdot \text{poly}(n)$. □

We now show how to use Lemma 3.1 and Theorem 3.2 to solve Satisfiability for quantified CNFs and DNFs with q quantifier blocks.

REMINDER OF THEOREM 1.2 *Satisfiability of q-QB-CNF (resp. q-QB-DNF) with n variables and $m = \text{poly}(n)$ clauses (resp. disjuncts) can be solved probabilistically in time $\text{poly}(n) \cdot 2^{n-\Omega(n^{1/(q+1)})}$ with zero error.*

Proof. First, observe that the most difficult cases of q-QB-CNF occur when the innermost quantifier block is existential; otherwise, it is easy to tell whether universally quantified variables over a CNF evaluate to true or false (such a QBF is true iff the CNF is a tautology over the universal variables, i.e., the empty formula with no clauses). Similarly, the difficult case of q-QB-DNF occurs when the innermost quantifier is universal.

Therefore, without loss of generality, we may assume the input QBF ϕ has the form

$$\exists \vec{x_1} \forall \vec{x_2} \dots \vec{x_q} \psi(\vec{x_1} \dots \vec{x_q}),$$

where ψ is a DNF with m clauses if q even, a CNF with m clauses if q odd. (If this is not the case, we can solve the satisfiability question for $\neg\phi$ instead, and flip the answer.)

The number of variables n equals $\sum_{i=1}^{q} |\vec{x_i}|$. We assume q is even for simplicity (so $\vec{x_q}$ is universally quantified); the case of odd q is analogous. For each i such that $1 \leq i \leq q$, let $n_i = |\vec{x_i}|$.

We consider two cases, depending on the size of n_1. If $n_1 \geq n - n^{1/(q+1)}$, let $n' = \Sigma_{i=2}^{q} n_i$. We have that $n_1 + n' = n$. The algorithm constructs an AC^0 circuit C of depth at most $q+1$ and size at most $s = 2^{n'} \text{poly}(n)$ which has as input $\vec{x_1}$, as follows: C is an AND over all possible assignments to $\vec{x_2}$, of an OR over all possible assignments to $\vec{x_3}$, ..., of an AND over all possible assignments to $\vec{x_q}$ of the DNF predicate $\psi(\vec{x_1}, \vec{x_2}, \dots, \vec{x_q})$. It then runs the satisfiability algorithm A of Theorem 3.2 on C. Note that C is satisfiable if and only if the QBF ϕ is satisfiable. By Theorem 3.2, the running time in this case is at most $O(\text{poly}(n) 2^{n'} 2^{n_1 - \Omega(n_1/(\log(s))^q)})$, which since $n' \leq n^{1/(q+1)}$, is at most $O(\text{poly}(m) 2^{n-\Omega(n/n^{q/(q+1)})})$, i.e., at most $O(\text{poly}(m) 2^{n-\Omega(n^{1/(q+1)})})$, as desired.

In the other case, $n_1 < n - n^{1/(q+1)}$. Let i be the largest number such that $\Sigma_{j=1}^{i} n_i < n - n^{1/(q+1)}$. Note that $i \geq 1$. Let $y \in \vec{x}_{i+1}$ be the variable such that the number of variables quantified after y is $n^{1/(q+1)} - 1$. Let us assume WLOG that i is odd; a similar argument works when i is even. The algorithm constructs an AC^0 circuit C' of depth at most $q+1$ and size at most $s = 2^{n^{1/q+1}} \text{poly}(n)$ as follows: C' is the AND over all assignments to variables in $\vec{x}_{i+1}$ which are either y or *after* y in order of quantification, of the OR over all possible assignments to variables in $\vec{x}_{i+2}$, ..., of the AND over all possible assignments to $\vec{x_q}$ of the DNF predicate $\psi(\vec{x_1}, \vec{x_2}, \dots, \vec{x_q})$. Note that circuit C' has as input variables all variables in $\vec{x_1} \dots \vec{x_i}$, as well as all variables in $\vec{x}_{i+1}$ before y. The satisfiability of ϕ is equivalent to the satisfiability of the quantified Boolean circuit C', where variables in $\vec{x_1} \dots \vec{x}_{i+1}$ are quantified as in ϕ.

The algorithm then does the following. It uses algorithm E from Lemma 3.1 to compute the truth table of C', which it stores in random access memory. The time taken for this computation is $\text{poly}(n) \cdot (2^{n-n^{1/(q+1)}} + 2^{n^{1/(q+1)}} 2^{n-n^{1/(q+1)} - \Omega((n-n^{1/(q+1)})/n^{q/q+1})})$, which is $O(\text{poly}(n) \cdot 2^{n-\Omega(n^{1/(q+1)})})$.

Next, the algorithm evaluates the quantified satisfiability of C' exhaustively by constructing the AND-OR tree of assignments corresponding to variables in $\vec{x_1} \dots \vec{x}_{i+1}$ being quantified as in ϕ, and searching the tree exhaustively. However, for each leaf of the tree, rather than evaluating C' to determine this value, the algorithm simply looks up the corresponding value in its stored truth table, in $\text{poly}(n)$ time per lookup. Thus the total time for evaluation is $O(2^{n-n^{1/(q+1)}} \text{poly}(n))$, since the tree has $O(2^{n-n^{1/(q+1)}})$ nodes.

Hence the total running time of the algorithm in this case is $O(\text{poly}(n) 2^{n-n^{1/(q+1)}})$, as desired. □

Note that Theorem 1.2 gives a non-trivial improvement over brute force search even when the number of quantifier blocks is $q = o(\log(n)/\log\log(n))$. For $q = 2$, the savings given by Theorem 1.2 is $\Omega(n^{1/3})$, which is not as good as Theorem 1.1 when the quantified formula is in 3CNF. However, as shown in Section 4, the dependence on q for large q is optimal modulo showing new circuit lower bounds.

3.3 Quantified Formulas with Many Quantifier Blocks Theorem 1.2 gives improvements over brute force search for QBF satisfiability on formulas with a small number of quantifiers. Somewhat counterintuitively, we observe that it is also possible to get savings when the number of quantifiers is relatively large, by a simple application of the idea of Snir [Sni85] and Saks-Wigderson [SW86] for reducing the decision tree complexity of game-tree evaluation.

REMINDER OF THEOREM 1.3 *Let $q : \mathbb{N} \to \mathbb{N}$ satisfy $q(n) < n$ for all n. Satisfiability of $q(n)$-QBF over m-size circuits with n input variables can be solved probabilistically in $\text{poly}(m) \cdot 2^{n-\Omega(q(n))})$ time, with zero-error.*

Proof. The idea is to explore the tree of variable assignments randomly. In particular, we branch on each variable in turn according to the quantifier order of the QBF, picking a random 0-1 assignment to the variable and recursing on the resulting instance. If the current variable is quantified existentially and the call returns SAT, then we return SAT, else we flip the variable to its opposite value and recurse, returning the answer resulting from the call. Dually, if the current variable is universally quantified and a recursive call returns UNSAT, then we return UNSAT, else we flip the variable and recurse on it.

We claim that random search runs in $2^{n-\Omega(q(n))}$ (worst-case) expected time on QBFs with $q(n)$ quantifier blocks. In particular, we expect that the exponent of the running time is reduced by $\Omega(1)$ per quantifier block. First note that, if the QBF instance is SAT, then at most $1/2$ fraction of the possible assignments to variables in each existential quantifier block are explored on average, before finding a certificate that the instance is satisfiable (i.e., a recursive call that returns SAT). Second, if the QBF is UNSAT, then at most $1/2$ fraction of the assignments in each universal quantifier block are explored on average, before finding a certificate that the instance is unsatisfiable (i.e., a call that returns UNSAT). No matter whether the given QBF instance is SAT or UNSAT, there is a $\Omega(1)$ savings in the exponent of the running time for every two consecutive quantifier blocks. When there are $q(n)$ quantifier blocks, we achieve an $\Omega(q(n))$ savings in the exponent. □

Note that the above algorithm only improves over exhaustive search when $q(n) = \omega(\log(n))$: otherwise, the $\text{poly}(n)$ factor dominates the the $2^{-\Omega(q(n))}$ savings. This is an interesting phenomenon where the presence of many quantifiers creates so many additional constraints on the QBF problem that the problem actually becomes easier to solve.

Theorem 1.2 and Theorem 1.3 together show that the "hard" cases of satisfiability of quantified Boolean formulas are where the number of quantifier blocks is between $\Theta(\log(n)/\log\log(n))$ and $\Theta(\log(n))$ – in all other cases, we are able to obtain non-trivial savings.

4 From Algorithms for QBF Satisfiability to Circuit Lower Bounds

In the second part of the paper, we investigate the consequences of finding faster QBF satisfiability algorithms. We find that Boolean formulas can be encoded rather well as QBF satisfiability instances over CNF, even bounded-width CNFs. This is extremely different from the case of typical satisfiability over 3-CNF (i.e., 3-SAT), as there are many known faster-than-2^n time algorithms for 3-SAT, but no general reductions from formula SAT to 3-SAT which yield faster SAT algorithms for general Boolean formulas. Relating these encoding results to known connections between SAT algorithms and circuit lower bounds, we prove that faster algorithms for QBF satisfiability over CNF would imply new lower bounds in circuit complexity such as NEXP is not contained in non-uniform NC^1.

4.1 Quantified Boolean Formulas with Arbitrary Number of Quantifier Blocks

We establish a rather tight relationship between the time complexity of solving quantified formulas and that of solving quantified 3-CNF. We will show that the choice of predicate in a quantified problem essentially does not matter: if quantified 3-CNF is solvable in c^n time then all quantified Boolean formulas are solvable in about c^n time.

REMINDER OF THEOREM 1.4 *There is a polynomial time algorithm that takes any Boolean formula of n inputs and s size and outputs an equivalent quantified CNF instance of $n + O(\log s)$ variables, $O(s^{10})$ clauses, and $O(\log n)$ quantifier blocks.*

Proof. We first do some general massaging of a given formula F, construed as a tree with interior nodes labeled by AND/OR gates, and leaves labelled by literals. We can assume WLOG that F has depth $c \log s$ where $c < 5$. (If this is not true, we can make it the case via known reductions [BCE91, BB94], at the cost of squaring the size of the formula.) Moreover, we may assume that the depth is even, and odd depths of F contain no AND gates (only OR gates, along with possibly $0-1$ constants and literals), while even depths contain no OR gates. (Enforcing this only increases the depth by a factor of two, and at most squares the size.) Finally, we can make the length of every path in F from the output gate to a literal *exactly* $d = 2c \log s$. (Suppose a path ends early at a literal ℓ; since $b \wedge b = b$ and $b \vee b = b$ for $b \in \{0,1\}$, we can duplicate occurrences of ℓ to match the desired alternations of ANDs and ORs and the desired length of each path.) Notice that F now has exactly $2^d \leq s^{10}$ leaves.

Let d be the depth of F, and let $L = \{x_1, \ldots, x_n, \neg x_1, \ldots, \neg x_n\}$ be the set of literals of F. Define a mapping $\phi_F : \{0,1\}^d \to L$ as follows. Given a d-bit string b, label every edge in F to a left child with a 0, and every edge to a right child with 1. Follow the path from the output gate (the root of the tree) by reading the bits of b from left to right, then output the literal at the leaf found.

The variables of our QB-CNF instance will be $x_1, \ldots, x_n$ along with new variables $y_1, \ldots, y_d$. For convenience, we use the notation $y_i^1 := y_i$ and $y_i^0 := \neg y_i$. The instance of QB-CNF includes all 2^d possible clauses of the form

$$(y_1^{1-b_1} \vee \cdots \vee y_d^{1-b_d} \vee \phi_F(b_1, \ldots, b_d)),$$

over all possible vectors $(b_1, \ldots, b_d) \in \{0,1\}^d$. Noticing that

$$\begin{aligned}&(y_1^{1-b_1} \vee \cdots \vee y_d^{1-b_d} \vee \phi_F(b_1, \ldots, b_d))\\ \equiv\ &((y_1^{b_1} \wedge \cdots \wedge y_d^{b_d}) \to \phi_F(b_1, \ldots, b_d)),\end{aligned}$$

we have that (a) for every assignment to $(y_1, \ldots, y_d)$, at most one of the above clauses is not trivially satisfied, and (b) this remaining clause is satisfied if and only if the literal at the leaf defined by the root-to-leaf path $b_1 \cdots b_d$ is true. The final formula is then

$$(Q_1 x_1, \ldots, Q_n x_n)(\forall y_1)(\exists y_2) \cdots (\forall y_d)[C]$$

where C is the above collection of clauses and $Q_i \in \{\exists, \forall\}$ is the quantifier on variable x_i from the original quantified Boolean formula. □

The following two reductions to QB-CNF are immediate corollaries:

COROLLARY 4.1. *There is a polynomial time reduction from* QB-FORMULA *instances of n inputs and s size to* QB-CNF *instances of $n + O(\log s)$ variables and $O(s^{10})$ size.*

COROLLARY 4.2. *There is a polynomial time reduction from* FORMULA-SAT *instances of n inputs and s size to* QB-CNF *instances of $n + O(\log s)$ variables, $O(s^{10})$ size, and $O(\log n)$ quantifier blocks.*

These corollaries have several nice consequences. First, the QBF problem over arbitrary formulas and the QBF problem over CNFs are essentially identical with respect to time complexity. Second, Boolean formula satisfiability can be efficiently reduced to QBF over CNF formulas with only $O(\log n)$ quantifier blocks. Applying results of Williams [Wil11], it follows that a nontrivial algorithm for QB-CNF satisfiability (even with zero-error) for $O(\log n)$ quantifier blocks would imply new circuit lower bounds:

REMINDER OF COROLLARY 1.2 *If for all k, quantified CNF with n variables, n^k clauses, and $k \log n$ quantifier blocks can be solved in zero-error probabilistic $2^n/n^k$ time, then* $\mathsf{NEXP} \not\subset \mathsf{NC}^1/\text{poly}$.

Proof. Williams [Wil11] shows that if there is a co-nondeterministic algorithm for FORMULA-SAT on instances with n inputs and n^k size running in $O(2^n/n^{10})$ time for all k, then $\mathsf{NEXP} \not\subset \mathsf{NC}^1/\text{poly}$. Hence the proof follows from Corollary 4.2. □

We can reduce from quantified CNF formulas to quantified k-CNF formulas, by applying a reduction of Calabro, Impagliazzo, and Paturi [CIP10].

THEOREM 4.1. ([CIP10]) *For all k, there is a polynomial-time reduction converting any* CNF *with n variables and m clauses into a logically equivalent* 2-QB-kCNF *with $n + O(m^{1/(k-1)})$ variables and* poly(m, n) *clauses.*

Proof. (Sketch) Calabro, Impagliazzo, and Paturi use $O(m^{1/(k-1)})$ additional variables to encode CNF evaluation in such a way that the given CNF F on n variables evaluates to true on an assignment if and only if the new QBF with $n+O(m^{1/(k-1)})$ variables is unsatisfiable. In particular, define a $(k-1)$-CNF formula G with m clauses and $O(m^{1/(k-1)})$ variables which is *minimally unsatisfiable*, in the sense that if any clause is removed then the formula is satisfiable. To build the clauses of the QBF, iterate over all clauses c_i of the given F and literal ℓ in it, putting in the QBF the clause $(c_i \vee \neg\ell)$. The final QBF existentially quantifies over the n variables of F, then universally quantifies over the $O(m^{1/(k-1)})$ variables of G. Then, a given assignment A to the n variables of F is satisfying if and only if every clause in G is present in the QBF after variable assignment A (which is true if and only if the remaining clause set is *unsatisfiable*). □

The above proof easily extends to a reduction from QB-CNF with q quantifier blocks to QB-kCNF with $q+1$ quantifier blocks, with the same increase in the number of variables.

Combining the reduction of Theorem 1.4 with the reduction of Theorem 4.1, we obtain:

REMINDER OF COROLLARY 1.1 *There is a polynomial time algorithm that, for every fixed k, takes any Boolean formula of n inputs and s size and outputs an equivalent quantified k-CNF instance of $n + O(s^{10/(k-1)})$ variables,* poly(s) *clauses, and $O(\log s)$ quantifier blocks.*

Proof. Theorem 1.4 says that we can take any Boolean formula F with n inputs and s size and generate (in polynomial time) an equivalent quantified CNF F' with $n + O(\log s)$ variables, $O(s^{10})$ clauses, and $O(\log s)$ quantifier blocks. Applying Theorem 4.1 to the CNF part of F', we can reduce F' to an equivalent 2-QB-kCNF F'' with $n + O(\log s) + O(s^{10/(k-1)})$ variables and poly(s) clauses. That is, by adding a quantifier to F' we obtain an equivalent k-CNF F'' with a few more variables. This completes the proof. □

Corollary 1.1 implies that faster algorithms for solving quantified k-CNF would imply new circuit lower bounds:

REMINDER OF COROLLARY 1.3 *If there is an $\varepsilon > 0$ such that for all k, quantified k-CNF with n variables, n^k clauses, and $k \log n$ alternations can be solved in zero-error probabilistic 2^{n-n^ε} time, then* NEXP $\not\subset$ NC^1/poly.

Proof. The hypothesis implies that Formula-SAT on n^c size formulas can be solved in zero-error $O(2^n/n^c)$ time for all constants c. To see this, let F be an n^c size formula. By the reduction of Corollary 1.1, we can convert F into a quantified k-CNF F' with $n + O(n^{10c/(k-1)})$ variables, n^{cd} clauses, and $dc \log n$ quantifier blocks. The hypothesis of the theorem says that, for every k, such formulas on m variables can be solved in 2^{m-m^ε} time for a universal $\varepsilon > 0$. Let k be greater than $(10c)/\varepsilon^3 + 1$. Then the running time of the algorithm on the formula F' is at most $2^{n+O(n^{\varepsilon^2})-(n+O(n^{10c/(k-1)}))^\varepsilon} \leq 2^{n-\Omega(n^\varepsilon)}$.

Finally, Williams [Wil10] showed that such a Formula-SAT algorithm would imply NEXP $\not\subset$ NC^1/poly. □

5 Lower Bound Implications of Solving Quantified Boolean Formulas with Constant Number of Quantifier Blocks

First we show how to convert an arbitrary Boolean formula into a quantified Boolean CNF with a small number of extra variables:

REMINDER OF THEOREM 1.5 *Let $k, r > 0$ be any integers. There is a polynomial-time algorithm that takes any Boolean formula of n inputs and depth $r \log(n)$, and outputs an equivalent QB-CNF instance of $n + O(n^{1/k})$ variables, size* poly(n)*, and $2kr$ quantifier blocks.*

Proof. We use a similar divide-and-conquer approach to the proof of Nepomnjascii's theorem [Nep70] that NL is in alternating linear time. Let F be the given formula of depth $r \log(n)$ on variables $x_1 \dots x_n$. We assume WLOG that all leaves (i.e., literals or constants) of F are at depth $r \log(n)$; if there is a literal or constant v at lower depth d, we replace it by a full binary tree of ANDs of depth $(r \log(n) - d)$ with v at each leaf.

We imagine F as divided into kr layers, where layer i consists of all nodes between depth $\lfloor (i \log(n))/k \rfloor$ and depth $\lfloor (i \log(n) + \log(n))/k \rfloor$. Sub-formulas at layer i are those sub-trees of the formula which have as root a node at depth $\lfloor (i \log(n))/k \rfloor$ and as leaves the descendants of the root at depth $\lfloor (i \log(n) + \log(n))/k \rfloor$. We call such a sub-formula for any $i, 0 \leq i \leq kr - 1$, a *local* sub-formula. Every local sub-formula has at most $4n^{1/k}$ nodes, since F is a formula where all nodes have fan-in at most two.

We construct a QB-CNF instance ϕ as follows. ϕ begins with $2kr - 1$ quantifier blocks $Q_1, Q_2 \dots Q_{2kr}$, where Q_i is existential if i is odd, and universal if i is even. The quantifier blocks Q_i for odd i quantify over variable blocks $y^1, y^2 \dots y^{kr}$, where for each $j, 1 \leq j \leq kr$, each variable block y^j consists of $\lceil 4n^{1/k} \rceil$ variables $y^j_l, l = 1 \dots \lceil 4n^{1/k} \rceil$. The quantifier blocks Q_i for even i quantify over variable blocks $z^1, z^2 \dots z^{kr-1}$, where for each $j, 1 \leq j \leq kr - 1$, each variable block z^j consists of $m(n) = \lceil \log(2n^{1/k}) \rceil$ variables $z^j_l, l = 1 \dots \lceil \log(2n^{1/k}) \rceil$. Note that the total number of quantified variables is $O(n^{1/k})$. Following the quantifier blocks is a CNF formula ψ in variables $x_1 \dots x_n, y^1 \dots y^{kr}, z^1 \dots z^{kr-1}$. The variables $x_1 \dots x_n$ will be called x-variables, $y^1 \dots y^{kr}$ will be y-variables, and $z^1 \dots z^{kr}$ will be z-variables.

Before describing ψ, we give some intuition for the construction of the QBF instance. The idea is to partition the formula into several local sub-formulas, each of small (i.e., $O(n^{1/k})$) size, and to verify the computation of the formula by verifying each local sub-formula independently. To verify the computation of a local sub-formula, values for all the nodes in the sub-formula are guessed existentially and checked for local consistency. Of course, to guess values for all nodes of the formula independently would require too many extra variables. Instead, universal quantifiers are used to take advantage of the layered structure of the formula and re-use variables between different sub-formula verifications. It turns out that a *constant* number of alternations suffices for this purpose.

We define a way to inductively index local sub-formulas. We identify each local sub-formula with its root. The root of the formula F is indexed by $()$. Now, given an index $(t_1, t_2 \dots t_i)$ for a local sub-formula T at layer i, let $T_1, T_2 \dots T_\ell$ be the local sub-formulas at layer $i + 1$ which are leaves of T, where $t_1 \dots t_i \in \mathbb{N}$, and $\ell \leq 2n^{1/k}$. We say that $(t_1, t_2 \dots t_i, t_{i+1})$ indexes T_j for $t_{i+1} \in \mathbb{N}, 1 \leq j \leq \ell$, if $t_{i+1} \equiv j \mod \ell$.

Now, let $t_1, t_2 \dots t_{kr-1} \in \{0,1\}^{m(n)}$ be arbitrary. For each such sequence, and for each initial segment $t_1, t_2 \dots t_i$, $0 \leq i \leq kr-1$ of the sequence, we define a CNF formula $\psi_{(t_1,t_2\dots t_i)}$ of size poly(n). The CNF formula ψ is the conjunction of $\psi_{(t_1,t_2\dots t_i)}$ over all sequences and all $i, 0 \leq i \leq kr-1$. It is not hard to see that ψ is of size poly(n).

Fix i. Interpreting $t_1, t_2 \dots t_{kr-1}$ as non-negative integers in the natural way, let T be the local sub-formula indexed by $(t_1, t_2 \dots t_i)$. We define a CNF $cons_T$ which has as variables $y^1, y^2 \dots y^{i+1}$ and $x_1 \dots x_n$. We then define $\psi_{(t_1,t_2\dots t_i)}$ using $cons_T$ and additional variables $z^1, s^2 \dots z^i$. In fact, $cons_T$ will only involve y-variables in y^i and y^{i+1}, and will only involve x-variables when $i = kr-1$. $\psi_{(t_1,t_2\dots t_i)}$ will only involve additional z-variables in z^i.

We distinguish two cases: $i < kr-1$ and $i = kr-1$. When $i < kr-1$, we define $cons_T$ as follows. Identify the variables in y^{i+1} with distinct nodes of T in some canonical way. There are at least as many variables in y^{i+1} as nodes in T, so some variables might be left over - these just won't be used in $cons_T$. Let a, b, c be arbitrary nodes in T such that b and c are children of a. Assume wlog that y_1^{i+1}, y_2^{i+1} and Y_3^{i+1} are the y-variables identified with a,b and c respectively. If a is an AND gate, $cons_T$ contains a set of clauses encoding that $y_2^{i+1} \wedge y_3^{i+1} = y_1^{i+1}$, else a is an OR-gate and $cons_T$ contains a set of clauses encoding that $y_2^{i+1} \vee y_3^{i+1} = y_1^{i+1}$. In addition, suppose v is the root of T, y_s^{i+1} is associated with v in T and y_q^i is associated with v in the local sub-formula of which v is a leaf. Then there is a pair of clauses in $cons_T$ encoding that $y_s^{i+1} = y_q^i$.

When $i = kr-1$, $cons_T$ contains all clauses as before, but in addition "leaf clauses" as follows. In this case, the leaves of T are all literals, i.e., x-variables or their complements, or else constants. Leaf clauses encode, for each leaf v' of T, that the y-variable associated with v' is equal to the corresponding literal/constant in formula F.

The CNF $cons_T$ encodes local consistency of the local sub-formula T in terms of the guessed information represented by the y-variables, but since we would like to re-use the y-variables for all local sub-formulas at a given layer, we would like this consistency check to kick in only for a specific setting of the universal variables. This is ensured by incorporating the z-variables as follows.

For each $j, 1 \leq j \leq m(n)$, let $w_j = z_j^i$ if the j'th bit of t_i is 0, and let w_j be the complement of z_j^i otherwise. When $i \geq 1$, we define $\psi_{t_1\dots t_i}$ to be $\vee_{j=1}^{m(n)} w_j \vee cons_T$. When $i = 0$, $\psi_{t_1\dots t_i}$ is the same as $cons_T$.

This completes the description of the QB-CNF formula ϕ. It is clear that ϕ can be constructed from F in polynomial time. What remains to be shown is that ϕ is equivalent to F.

Suppose F evaluates to 1 on a specific input $x_1, x_2 \dots x_n$. Then by assigning y-variables to the values of corresponding interior nodes in the formula for this assignment, it is easy to see that ϕ is satisfied. Conversely, if ϕ is satisfied, then using the fact that local consistency as well as consistency between a local sub-formula and its parent is encoded into ϕ, F evaluates to 1. Indeed, values of interior nodes for F can be recovered from any witness tree for ϕ. □

COROLLARY 5.1. *There is a polynomial-time reduction from* FORMULA-SAT *instances of n inputs and size* poly(n) *to* QB-CNF *instances with $O(k)$ quantifier blocks, $n + O(n^{1/k})$ variables and size* poly(n).

Corollary 5.1 follows from Theorem 1.5 using Lemma 2.1, which gives a simulation of poly(n) size formulas by $O(\log(n))$ depth formulas. Note that any $O(\log(n))$ depth circuit can be simulated by an $O(\log(n))$ depth formula by creating separate copies of sub-formulas to replace gates with fan-out greater than 1.

REMINDER OF COROLLARY 1.4 *If satisfiability of quantified CNF with n variables,* poly(n) *clauses and q quantifier blocks can be solved in zero-error probabilistic time $2^{n-n^{\omega_q(1/q)}}$, then* NEXP $\not\subseteq$ NC^1/poly.

Proof. Suppose the assumption holds. Then, using Corollary 5.1, we have that Formula-SAT instances of n inputs and size poly(n) can be solved in zero-error probabilistic time $2^{n-\omega(\log(n))}$. Using the algorithms-to-lower-bounds connection of Williams [Wil10], this implies that NEXP $\not\subseteq$ NC^1/poly. □

By combining Corollary 5.1 with Theorem 4.1, we can derive an even stronger connection:

COROLLARY 5.2. *If satisfiability of quantified q-CNF with n variables,* poly(n) *clauses and q quantifier blocks can be solved in zero-error probabilistic time $2^{n-n^{\omega_q(1/q)}}$, then* NEXP $\not\subseteq$ NC^1/poly.

Acknowledgements. We thank the SODA reviewers for their helpful comments.

References

[BB94] Maria Bonet and Samuel Buss. Size-depth tradeoffs for boolean formulae. *Information Processing Letters*, 49(3):151–155, 1994.

[BCE91] Nader H. Bshouty, Richard Cleve, and Wayne Eberly. Size-depth tradeoffs for algebraic formulae. In *FOCS*, pages 334–341, 1991.

[CIP06] Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. A duality between clause width and clause density for SAT. In *Proceedings of IEEE Conference on Computational Complexity*, pages 252–260, 2006.

[CIP09] Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. The complexity of satisfiability of small depth circuits. In *Proc. International Workshop on Parameterized and Exact Computation*, 2009.

[CIP10] Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. On the exact complexity of evaluating quantified k-CNF. *Algorithmica*, 65(4):817–827, 2013. See also IPEC 2010.

[DH09] Evgeny Dantsin and Edward Hirsch. Worst-case upper bounds. In *Handbook of Satisfiability*, chapter 12, pages 403–424. IOS Press, 2009.

[DW06] Evgeny Dantsin and Alexander Wolpert. A faster clause-shortening algorithm for sat with no restriction on clause length. *JSAT*, 1(1):49–60, 2006.

[DW10] Evgeny Dantsin and Alexander Wolpert. On moderately exponential time for SAT. In *Proceedings of 13th International Conference on Satisfiability Testing*, pages 313–325, 2010.

[GIB09] Alexandra Goultiaeva, Vicki Iverson, and Fahiem Bacchus. Beyond CNF: A circuit-based QBF solver. In *Theory and Applications of Satisfiability Testing (SAT 2009)*, pages 412–426, 2009.

[Her11] Timon Hertli. 3-sat faster and simpler - unique-sat bounds for ppsz hold in general. In *FOCS*, pages 277–284, 2011.

[IMP12] Russell Impagliazzo, William Matthews, and Ramamohan Paturi. A satisfiability algorithm for AC0. In *Proceedings of 23rd Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 961–972, 2012.

[IP01] Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001.

[MS85] Burkhard Monien and Ewald Speckenmeyer. Solving satisfiability in less than 2^n steps. *Discrete Applied Mathematics*, 10:287–295, 1985.

[MS11] Robin A. Moser and Dominik Scheder. A full derandomization of Schöning's k-SAT algorithm. In *STOC*, pages 245–252, 2011.

[MZ09] Sharad Malik and Lintao Zhang. Boolean satisfiability from theoretical hardness to practical success. *Communications of the ACM*, 52(8):76–82, 2009.

[Nep70] V. Nepomnjascii. Rudimentary predicates and turing calculations. *Soviet Mathematics - Doklady*, 11(6):1462–1465, 1970.

[PPSZ98] Ramamohan Paturi, Pavel Pudlak, Mike Saks, and Francis Zane. An improved exponential-time algorithm for k-sat. In *Proceedings of 39th International Symposium on Foundations of Computer Sciece (FOCS)*, pages 628–637, 1998.

[PPSZ98] Ramamohan Paturi, Pavel Pudlák, Michael E. Saks, and Francis Zane. An improved exponential-time algorithm for *k*-sat. *J. ACM*, 52(3):337–364, 2005. (See also FOCS'98).

[PPZ97] Ramamohan Paturi, Pavel Pudlak, and Francis Zane. Satisfiability coding lemma. In *Proceedings of 38th International Symposium on Foundations of Computer Science (FOCS)*, pages 566–574, 1997.

[Pud98] Pavel Pudlak. Satisfiability – algorithms and logic. In *Mathematical Foundations of Computer Science, Springer LNCS Volume 1450*, pages 129–141, 1998.

[San10] Rahul Santhanam. Fighting perebor: New and improved algorithms for formula and QBF satisfiability. In *Proceedings of 51st Annual IEEE Symposium on Foundations of Computer Science*, pages 183–192, 2010.

[Sch99] Uwe Schoning. A probabilistic algorithm for k-sat and constraint satisfaction problems. In *Proceedings of 40th Annual Symposium on Foundations of Computer Science*, pages 410–414, 1999.

[Sch05] Rainer Schuler. An algorithm for the satisfiability problem of formulas in conjunctive normal form. *J. Algorithms*, 54(1):40–44, 2005.

[Sni85] Marc Snir. Lower bounds on probabilistic decision trees. *Theoretical Computer Science*, 38:69–82, 1985.

[Spi71] Philip Spira. On time-hardware complexity tradeoffs for boolean functions. In *Proceedings of the Fourth Hawaii International Symposium on System Sciences*, pages 525–527, 1971.

[SW86] Michael Saks and Avi Wigderson. Probabilistic boolean decision trees and the complexity of evaluating game trees. In *Proceedings of 27th Annual IEEE Symposium on Foundations of Computer Science*, pages 29–38, 1986.

[Wil02] Ryan Williams. Algorithms for quantified boolean formulas. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 299–307, 2002.

[Wil10] Ryan Williams. Improving exhaustive search implies superpolynomial lower bounds. In *Proceedings of the 42nd Annual ACM Symposium on Theory of Computing*, pages 231–240, 2010.

[Wil11] Ryan Williams. Non-uniform ACC circuit lower bounds. In *Proceedings of 26th Annual IEEE Conference on Computational Complexity*, pages 115–125, 2011.

[Wil13] Ryan Williams. Natural proofs versus derandomization. In *STOC*, pages 21–30, 2013.

[Wil14] Ryan Williams. Faster decision of first-order graph properties. In *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria*, page 80, 2014.

[Zha06] Lintao Zhang. Solving QBF by combining conjunctive and disjunctive normal forms. In *Proceedings of 21st National*

Conference on Artificial Intelligence and 18th Innovative Applications of Artificial Intelligence Conference (AAAI 2006), 2006.

ECCC ISSN 1433-8092

http://eccc.hpi-web.de