Automating Tree-Like Resolution in Time $n^{o(\log n)}$ Is ETH-Hard

Susanna F. de Rezende
Institute of Mathematics of the Czech Academy of Sciences

March 5, 2021

Abstract

We show that tree-like resolution is not automatable in time $n^{o(\log n)}$ unless ETH is false. This implies that, under ETH, the algorithm given by Beame and Pitassi (FOCS 1996) that automates tree-like resolution in time $n^{O(\log n)}$ is optimal. We also provide a simpler proof of the result of Alekhnovich and Razborov (FOCS 2001) that unless the fixed parameter hierarchy collapses, tree-like resolution is not automatable in polynomial time. The proof of our results builds on a joint work with Göös, Nordström, Pitassi, Robere and Sokolov (STOC 2021), which presents a simplification of the recent breakthrough result of Atserias and Müller (FOCS 2019).

1 Introduction

The P vs NP problem is equivalent to the question of whether there exists a polynomial-time algorithm that given a 3-CNF formula determines if it is satisfiable. If such an algorithm exists, then given an unsatisfiable formula the execution trace of the algorithm can be seen as a polynomial size proof—formalizable in some system—of unsatisfiability. Therefore, the P vs NP problem can be broken into two parts:

(i) Do all unsatisfiable formulas F admit polynomial size proofs of unsatisfiability?

(ii) If an unsatisfiable formula F admits polynomial size proofs of unsatisfiability, can we find one in polynomial time?

Answering any of these questions negatively would prove that $\mathsf{P} \neq \mathsf{NP}$, and answering both of them positively would imply that $\mathsf{P} = \mathsf{NP}$. Proof complexity provides a natural framework for studying these questions and, moreover, studying specific proof systems allows us to perceive the potential and limits of the class of algorithms that are based on these systems. Traditionally, more attention has been given to the first question, but since the late 90's there has been a growing interest in the second [BP96, Iwa97, KP98, BPR00, ABMP01, BDG$^+$04, AB04, AR08, GL10] with a more recent revival [MPW19, AM20, GKMP20, dRGN$^+$21, Gar20, Bel20] mainly due to the breakthrough result of Atserias and Müller [AM20].

To formalize the proof search problem, Bonet, Pitassi and Raz [BPR00] introduced the notion of automatability. A proof system $\mathcal{P}$ is said to be automatable in time $f(n)$ if there is an algorithm that given any unsatisfiable formula F outputs a $\mathcal{P}$-refutation of F in time $f(n)$, where n is the size of the smallest $\mathcal{P}$-refutation of F plus the size of F.

In this paper, we focus on the (non-)automatability of tree-like resolution. Tree-like resolution stands out for being a natural non-trivial proof system that is known to be automatable in quasi-polynomial time. This was observed by Beame and Pitassi in [BP96], where they presented a quite straightforward algorithm that automates tree-like resolution in time $n^{O(\log n)}$. Whether this simple algorithm is optimal was left open.

Alekhnovich and Razborov [AR08] provided the first evidence that tree-like resolution is not-automatable in polynomial time. Their result, together with a derandomization procedure from [EGG08], implies that if tree-like resolution is polynomial-time automatable then $\mathsf{W[P]} = \mathsf{FPT}$. More recently, Mertz, Pitassi and Wei [MPW19] proved that if tree-like resolution is automatable in time $n^{O(\log^{1/7-\epsilon} \log n)}$ then ETH is false. We note that these two results are incomparable since $\mathsf{W[P]} = \mathsf{FPT}$ implies ETH is false but the other direction is not known to hold.

In this work, we strengthen the latter result and, under ETH, settle the question of tree-like resolution's automatability. The proof builds on [dRGN+21] which presents a simplification of the recent breakthrough result of Atserias and Müller [AM20]. By generalizing the approach slightly, we are also able to provide an alternative, arguably much simpler proof of the result of Alekhnovich and Razborov [AR08].

Theorem 1.1. *If tree-like resolution is automatable:*

(i) *in time* $n^{o(\log n)}$ *then* ETH *is false;*
(ii) *in time* $n^{O(\log^{1-\epsilon} n)}$*, for* $0 < \epsilon \leq 1$*, then* 3-SAT $\in \mathsf{DTIME}(2^{O(n^{1-\epsilon/2})})$*;*
(iii) *in time* poly(n) *then* $\mathsf{W[P]} = \mathsf{FPT}$*.*

We note that Theorem 1.1 holds even if the automating algorithm is allowed to produce DAG-like resolution refutations, or even polynomial calculus refutations. In Section 2 we present an overview of the proof and also explain how the rest of the paper is structured.

2 Proof overview

Before we outline the proof of the main theorem, we introduce some notation. We denote by $[n]$ the set of integers $\{0, 1, \ldots, n-1\}$. Given an integer $\sigma \in [2^k]$, we denote by σ_i the $(i+1)$st least significant bit of σ, so that $\sigma = \sum_{i\in[k]} 2^i \cdot \sigma_i$.

Let F be a CNF formula, that is, a conjunction of clauses, over variables x_i for $i \in [n]$. A *literal* is either a variable or its negation. We denote the negation of a variable x_i by $\overline{x}_i$ and also use the notation $x_i^1 = x_i$ and $x_i^0 = \overline{x}_i$. We refer to the clauses of F as *axioms* of F, and sometimes view a clause as a set of literals. The *width of a clause* D, denoted $|D|$, is the number of literals in the clause, and the *width of a CNF formula* F is $\mathrm{w}(F) = \max_{D\in F} |D|$. A *resolution refutation* of F is a sequence of clauses $\mathcal{P} = (D_0, D_1, \ldots, D_{s-1} = \bot)$ such that, for each $i \in [s]$, either D_i is an axiom of F or it is derived from previous clauses by the *resolution rule*

$$D_i = D_j \setminus \{x_k\} \cup D_{j'} \setminus \{\overline{x}_k\} \ , \tag{1}$$

where $x_k \in D_j$, $\overline{x}_k \in D_{j'}$ and $j, j' < i$, or by *weakening* $D_i \supseteq D_j$, where $j < i$. The *size* of the refutation $\mathcal{P}$ is s and the *width* of $\mathcal{P}$ is the maximum width of any clause D_i in $\mathcal{P}$. We denote by $\mathrm{w}(F \vdash \bot)$ the minimum width over all refutations of F. A refutation is said to be *tree-like* if every clause appearing in the refutation is used in at most one derivation step.

The *polynomial calculus* proof system is defined in Section 6. For this overview it is enough to note that it polynomially simulates resolution.

A circuit with OR ($\vee$), AND ($\wedge$), and NOT ($\neg$) gates that receives as input n Boolean variables is referred to as an *n-variate (Boolean) circuit*. An n-variate circuit C is *satisfiable* if there is an assignment to the n input variables such that C outputs 1; otherwise C is *unsatisfiable*. The *size* of the circuit is the number of wires.

The *exponential time hypothesis* (ETH), which is a strengthening of the $\mathsf{P} \neq \mathsf{NP}$ conjecture, states that 3-SAT cannot be solved in subexponential time, i.e., that there is no algorithm that, for any 3-CNF formula F over n variable, determines whether F is satisfiable in time $2^{o(n)}$. The class of all *fixed parameter tractable* problems is denoted FPT and can be viewed as the "parameterized P". There is a whole hierarchy of classes that can be viewed as the "parameterized NP" and $\mathsf{W[P]}$ is placed at the top of this hierarchy. We note that the complexity assumption $\mathsf{W[P]} \neq \mathsf{FPT}$ is in between ETH and $\mathsf{P} \neq \mathsf{NP}$, that is, ETH implies $\mathsf{W[P]} \neq \mathsf{FPT}$ which implies $\mathsf{P} \neq \mathsf{NP}$. We refer the reader to the textbook [FG06] for an excellent survey on parameterized complexity. For our purposes, we only need the following classical result.

Theorem 2.1 (Theorem 6.3 in [ADF95], see also Theorem 3.25 in [FG06])**.** *If there is an algorithm that given any n-variate circuit C of size m decides whether C is satisfiable in time* $\operatorname{poly}(m) \cdot 2^{n/\iota(n)}$*, where $\iota : \mathbb{N} \to \mathbb{N}$ is a nondecreasing, unbounded computable function, then* $\mathsf{W[P]} = \mathsf{FPT}$.

Let $\Gamma = (U \mathbin{\dot\cup} V, E)$ be a bipartite graph. The (maximum) left degree of Γ is the maximum degree of any vertex in U. For $u \in U$, let $N_\Gamma(u) = \{v \in V : (u, v) \in E\}$ denote the set of neighbors of u. We would like to stress that we only use N_Γ for vertices in U and, therefore, we often view N_Γ as a function $N_\Gamma : [|U|] \to 2^{[|V|]}$, where we identify $U = [|U|]$ and $V = [|V|]$. For $U' \subseteq U$, let $\partial_\Gamma(U')$ be the set of unique neighbors of U', that is, the set of vertices $v \in V$ such that $|\{u \in U' : (u, v) \in E\}| = 1$. We sometime denote the set of edges E by $E(\Gamma)$. We consider the following definition of expanders.

Definition 2.2 ([ABRW04])**.** A bipartite graph $\Gamma = (U \mathbin{\dot\cup} V, E)$ is an *(r, Δ, c)-boundary expander* if Γ has left degree at most Δ and, for all $U' \subseteq U$, if $|U'| \leq r$ then $\partial_\Gamma(U') \geq c|U'|$.

There are many known explicit constructions of expanders. For our applications the following special case of the construction in [CRVW02] suffices.

Theorem 2.3 ([CRVW02])**.** *For any large enough $n \in \mathbb{N}$ and any $m \in \mathbb{N}$ such that $n \leq m \leq n^{O(1)}$, there is a polynomial-time constructible bipartite graph $\Gamma_n^m = (U \mathbin{\dot\cup} V, E)$, with $|U| = m$ and $|V| = n$ such that Γ_n^m is an $(r, \Delta, \Delta/2)$-boundary expander, where $r = \Theta(n)$ and $\Delta = \Theta(1)$.*

2.1 Main theorem

Our main theorem, from which Theorem 1.1 follows almost immediately, says that given any circuit C, it is possible to construct a CNF formula that has moderately small tree-like resolution refutations when C is satisfiable but requires much larger refutations, even in the stronger polynomial calculus proof system, when C is unsatisfiable.

Theorem 2.4 (Main theorem)**.** *There is a* $\operatorname{poly}(m) \cdot 2^{O(\sqrt{n})}$*-time algorithm $\mathcal{A}$ that given as input an n-variate Boolean circuit C of size m, where $n \leq m \leq 2^n$, outputs a CNF formula $\mathcal{A}(C)$ such that:*

(i) *If C is* satisfiable*, then $\mathcal{A}(C)$ admits a tree-like resolution refutation of size* $\operatorname{poly}(m) \cdot 2^{O(\sqrt{n})}$.

(ii) *If C is* unsatisfiable*, then $\mathcal{A}(C)$ requires polynomial calculus refutations of size at least $2^{\Omega(n)}$.*

This theorem is in the same spirit as its counterparts in [AM20] and [dRGN⁺21], which state that given a CNF formula F it is possible to construct in polynomial time a CNF formula $\mathcal{A}(F)$ that has polynomial size resolution refutations when F is satisfiable and requires resolution refutations of size $2^{\Omega(n)}$ when F is unsatisfiable. This implies that resolution is NP-hard to automate. The reason the previous versions of $\mathcal{A}(F)$ do not give any non-automatability result for tree-like resolution is that the upper bound strategy, when F is satisfiable, essentially produces a DAG-like refutation. Moreover, we do not expect there to be a polynomial, or even $\exp(o(n/\log n))$, size tree-like resolution refutation of the previous versions of $\mathcal{A}(F)$ since—by analyzing the algorithm from [BP96] that automates tree-like resolution in quasi-polynomial time—the former would imply that NP is in quasi-polynomial time and the latter that ETH is false.

We would like, therefore, to define some variant of the $\mathcal{A}(F)$ formula of [AM20, dRGN$^+$21] that has a small(-ish) tree-like refutation whenever F is satisfiable. We show that this is possible by blowing up the size of the formula. There is, in fact, a certain trade-off between the size of the formula $\mathcal{A}(F)$ and the tree-like upper bound we can prove for $\mathcal{A}(F)$, and the best results are obtained when these two values are approximately the same. We explain this in more detail in Section 4.

Another difference between our result and that of [AM20, dRGN$^+$21] is that we generalize the algorithm $\mathcal{A}$ to receive as input any circuit and not only CNF formulas. This generalization is not necessary for obtaining items (i) and (ii) of Theorem 1.1 but is important for item (iii).

2.2 Proof of Theorem 1.1 from Theorem 2.4

Let us see how Theorem 1.1 follows from the main theorem (Theorem 2.4).

Proof of Theorem 1.1. Suppose that tree-like resolution is automatable in time $f(N)$ (even if allowed to output a polynomial calculus refutation), that is, that there exist an algorithm $\mathbb{A}$ that given an unsatisfiable formula F finds a polynomial calculus refutation of F in time $f(N)$, where N is the size of the smallest tree-like resolution refutation of F plus the size of F. We will use $\mathbb{A}$ to decide whether a circuit (for (i) and (ii) specifically a 3-CNF formula) is satisfiable.

Let C be an n-variate circuit of size m. Let p be the largest of the two $\mathrm{poly}(m)$ factors that appear in the statement of Theorem 2.4. By Theorem 2.4 we have that the size of the formula $\mathcal{A}(C)$, which we denote by R, is at most $p \cdot 2^{O(\sqrt{n})}$ and that:

- if C is satisfiable, there is a tree-like resolution refutation of $\mathcal{A}(C)$ of size $S \leq p \cdot 2^{O(\sqrt{n})}$;
- if C is unsatisfiable, there is no polynomial calculus refutation of $\mathcal{A}(C)$ of size $L = 2^{\Omega(n)}$.

For items (i) and (ii) we consider the case where C is a 3-CNF formula viewed as a depth-2 circuit of size $m = O(n^3)$ (hence p is polynomial in n). For item (i) our assumption is that $f(N) = N^{o(\log N)}$. In order to decide whether C is satisfiable we simply run the algorithm $\mathbb{A}$ on $\mathcal{A}(C)$ for $f(R+S)$ steps. If C is satisfiable we are guaranteed $\mathbb{A}$ will return a refutation, and if C is unsatisfiable $\mathbb{A}$ cannot return a refutation since $f(R+S) = 2^{o(n)} < L$ for any large enough n. Therefore, we are solving 3-SAT in time $f(R+S) = 2^{o(n)}$, which implies that ETH is false. Item (ii) follows analogously by considering $f(N) = N^{O(\log^{1-\epsilon} N)}$.

It remains to prove item (iii), where we start with the assumption that $f(N) = O(N^k)$ for some constant k. We show that, for n large enough, there is an algorithm that decides whether C is satisfiable in time $\mathrm{poly}(m) \cdot \exp(O(\sqrt{n}))$. By Theorem 2.1 this implies that $\mathsf{W[P]} = \mathsf{FPT}$.

We first note that if $p \geq L^{1/2k}$, then $m = 2^{\Omega(n)}$ and the trivial algorithm that evaluates C on each of the 2^n assignments runs in time $\mathrm{poly}(m) \cdot 2^n = \mathrm{poly}(m)$. We therefore assume that $p < L^{1/2k}$. We run algorithm $\mathbb{A}$ on the formula $\mathcal{A}(C)$ for $f(R+S)$ steps. If C is satisfiable, $\mathbb{A}$ is guaranteed to return a refutation. Moreover, since $f(R+S) \leq O((2p)^k) \cdot 2^{O(\sqrt{n})}$, which (assuming n large enough) is less than $L = 2^{\Omega(n)}$ by our assumption $p < L^{1/2k}$, it follows that if $\mathbb{A}$ did return a refutation it cannot be the case that C is unsatisfiable. Therefore, we conclude that in time $f(R+S) = \mathrm{poly}(m) \cdot 2^{O(\sqrt{n})}$ we can decide C's satisfiability. □

The rest of the paper is dedicated to the proof of Theorem 2.4. Let C be an n-variate circuit with m gates. The formula output by the algorithm $\mathcal{A}$ of Theorem 2.4 when receiving C as input is the CNF formula $\mathrm{ShallowRef}(C)$ that is defined in Section 3. This definition is, perhaps, the most important technical contribution of this paper. The construction is based on the formula $\mathrm{Ref}(F)$ of [AM20, dRGN$^+$21] but requires some non-trivial modifications. Once the correct definition is in place, proving Theorem 2.4 is a rather straightforward adaptation of previous works [AM20, dRGN$^+$21].

In Section 4, we exhibit a tree-like resolution refutation of $\mathrm{ShallowRef}(C)$, when C is satisfiable, in size $\mathrm{poly}(m) \cdot 2^{O(\sqrt{n})}$ proving the upper bound in Theorem 2.4. Then, in Section 5, we prove the lower bound of Theorem 2.4 for DAG-like resolution by showing that, when C is unsatisfiable, $\mathrm{ShallowRef}(C)$ requires DAG-like resolution refutations of size at least $2^{\Omega(n)}$. This is done by first proving a width lower bound via a reduction to the pigeonhole principle (as was done in [dRGN$^+$21], although here we use a slightly different encoding based on expander graphs in order to get optimal parameters) and then applying the well-known size-width relation of Ben-Sasson and Wigderson [BW01]. Finally, in Section 6 we show that the lower bound holds even for the polynomial calculus proof system. We wrap-up in Section 7 with some open questions.

3 The formula $\mathrm{ShallowRef}(C)$

Given a CNF formula F over n variables, the formula $\mathrm{Ref}(F)$ of [AM20, dRGN$^+$21] encodes that there is a resolution refutation of F of some fixed polynomial size. It is convenient to view this purported refutation as being layered in such a way that every clause at a given layer is obtained from two clauses in the layer precisely below it and that there are a total of n layers, each with $\mathrm{poly}(n)$ clauses.

As mentioned earlier, it seems that the formula $\mathrm{Ref}(F)$ cannot give non-automatability results for tree-like resolution. Indeed, it might well be the case that tree-like resolution requires size at least $2^{\Omega(n)}$ to refute $\mathrm{Ref}(F)$, regardless of F's satisfiability. The problem is that the resolution refutation upper bound in [AM20, dRGN$^+$21], when F is satisfiable, is crucially DAG-like. It consists of following a root-to-leaf path in the purported refutation in order to reach a leaf that contains a clause falsified by a satisfying assignment of F (leading to a contradiction). If we transform this refutation into a tree-like refutation the size blows up to $\exp(\Omega(n \log n))$ since it must contain a clause for each of the $\exp(\Omega(n \log n))$ different root-to-leaf paths.

Thus, in order to obtain a shorter tree-like refutation, we adapt $\mathrm{Ref}(F)$ to reduce the number of root-to-leaf paths. This can be done by considering a shallow variant, which we denote $\mathrm{ShallowRef}(F)$, where instead of n layers we have $h \ll n$ layers. If each layer has $\mathrm{poly}(n)$ clauses, there can be at most $\exp(O(h \log n))$ root-to-leaf paths, hence there are tree-like resolution refutations of approximately this size (which for h small enough is much smaller than the lower bound of $2^{\Omega(n)}$).

The catch is that, given a satisfying assignment of the formula F, we must be able to trace one of these root-to-leaf paths and this requires that each clause in the purported refutation be derived from $2^{n/h}$ clauses in the layer below. This is the reason we have the trade-off, mentioned earlier, between the size of the formula $\mathrm{ShallowRef}(F)$ and the tree-like resolution upper bound we can obtain. In short, for every $h \in \{1, \ldots, n\}$ we can define a version of the formula $\mathrm{ShallowRef}(F)$ of size $\mathrm{poly}(n) \cdot 2^{O(n/h)}$ that, when F is satisfiable, has a tree-like resolution refutation of size $\mathrm{poly}(n) \cdot 2^{O(h \log n)}$.

To get the tight non-automatability results, however, we need to reduce further the number root-to-leaf paths to obtain an upper bound of $\mathrm{poly}(n) \cdot 2^{O(h)}$. To this end, for any given clause in the purported refutation, we restrict the clauses from the layer below that can be used to obtain it.

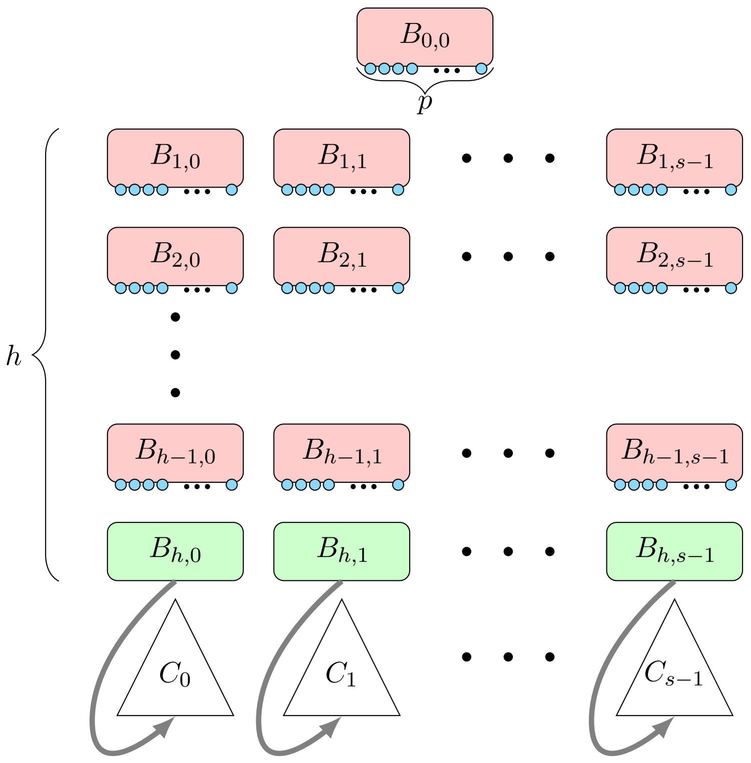

Figure 1: Structure of the blocks of $\mathrm{ShallowRef}(C)$, where $h = \sqrt{n}$ and $s = m2^{2n/h}$. Red blocks are internal blocks and each one has $p = 2^{n/h}$ pointers (represented by blue circles). Green blocks are leaf blocks and each one is associated to a copy of the circuit C.

This is done by considering a constant degree expander between layers and, for such an expander to exists, increasing the number of clauses per layer to $2^{\Theta(n/h)}$. Finally, we choose $h = \sqrt{n}$ so that both the size of the formula and that of the tree-like refutation are approximately the same.

Another difference from $\mathrm{Ref}(F)$ is that $\mathrm{ShallowRef}(C)$ is defined for any circuit C while $\mathrm{Ref}(F)$ is defined only for CNF formulas F. As noted before, this generalization is necessary in order to obtain item (iii) in Theorem 1.1. We point out that this requires that we encode $2^{\Theta(n/h)}$ copies of the circuit C in $\mathrm{ShallowRef}(C)$ (one for every leaf of the purported refutation).

We now proceed to define the formula $\mathrm{ShallowRef}(C)$. Let C be a Boolean circuit—i.e., with OR ($\vee$), AND ($\wedge$), and NOT ($\neg$) gates and Boolean inputs—of size m and let $x_0, \ldots, x_{n-1}$ be its input variables. For simplicity, we assume C has fan-in 2 (otherwise we can construct in time $O(m)$ a circuit C' of fan-in 2 that computes the same function as C and that has size at most $2m$). For the rest of this section, we fix parameters $h = \sqrt{n}$, $p = 2^{n/h}$ and $s = mp^2$. Moreover, let $\Gamma = \Gamma_s^{sp} = (U \dot\cup V, E)$ be the polynomial-time constructible bipartite $(r, \Delta, \Delta/2)$-boundary expander, given by Theorem 2.3, with $|U| = sp$, $|V| = s$, $r = \Theta(s)$ and $\Delta = \Theta(1)$. Recall that $N_\Gamma : [sp] \to 2^{[s]}$ is the neighborhood function from $U = [sp]$ to subsets of $V = [s]$.

3.1 Variables of $\mathrm{ShallowRef}(C)$

The variables of $\mathrm{ShallowRef}(C)$ are structured in $sh + 1$ blocks set up in $h + 1$ layers (see Figure 1): for $\alpha \in \{1, \ldots, h\}$, layer α contains blocks $B_{\alpha,0}, \ldots, B_{\alpha,s-1}$. The first block $B_{0,0}$ is alone on layer 0 and is called the root. Blocks on layer h are called leaf blocks and all non-leaf blocks (including the root) are internal blocks.

The idea is for each block to contain a partial assignment,[1] each internal block to have p pointers, one for each of the different ways of extending the current assignment to the next $\log p$ variables, and each leaf block to have a copy of the circuit C. One caveat is that each pointer only has a constant number of blocks in the layer below that it can point to, and these options are given by the fixed expander graph Γ. Formally, for each block B we have the following Boolean variables:

- *Enabled variable.* There is one variable enbl^B encoding whether the block B is *enabled* or not.
- *Literal set variables.* There are $2n$ many indicator variables lit_ℓ^B, one for each of the literals $\ell \in \{x_0, \overline{x}_0, \ldots, x_{n-1}, \overline{x}_{n-1}\}$. This defines the set of literals corresponding to a partial assignment at B. The intended interpretation is that $\mathsf{lit}_\ell^B = 1$ if and only if the partial assignment in block B assigns ℓ to true. (Note that assignments are defined for literals and can be inconsistent if both $\mathsf{lit}_x^B = 1$ and $\mathsf{lit}_{\overline{x}}^B = 1$. Moreover, $\mathsf{lit}_\ell^B = 0$ simply means that the partial assignment does not assign ℓ to true, and not that it assigns ℓ to false.)

Moreover, if $B = B_{\alpha,i}$ is an internal block we have:

- *Pointer variables.* There are p pointers with at most $\Delta = O(1)$ many Boolean variables each, namely, for each $\sigma \in [p]$, there are variables $\mathsf{pntr}_j^{B,\sigma}$ for all $j \in N_\Gamma(ip + \sigma)$. The intention is that $\mathsf{pntr}_j^{B,\sigma}$ encodes whether the σth child of the block B is the block j on the layer below, and that each of the p children of B corresponds to one of all the possible extension of the partial assignment in B to the next $\log p$ variables.

And if B is a leaf block we have:

- *Circuit variables.* There is one copy of the circuit C per leaf block. This means that there are $m = |C|$ variables gate_g^B for $g \in [m]$ that encode the value of each gate (of this copy) of the circuit C. The intention is that these variables encode the evaluation of each gate on the assignment at the leaf block B.

We note that in total, the formula $\mathrm{ShallowRef}(C)$ has less than $(hs + 1)(1 + 2n + \Delta p + m) = O(nmsp) = O(nm^2 \cdot 2^{3\sqrt{n}})$ many variables.

3.2 Axioms of ShallowRef(C)

The axioms of $\mathrm{ShallowRef}(C)$ can be partitioned into three groups: root axioms, internal block axioms and circuit axioms.

Root axioms. Let $B = B_{0,0}$ be the root. This first group of axioms consists of the unit clause enbl^B and the $2n$ unit clauses

$$\neg \mathsf{lit}_\ell^B \qquad \forall \ell \in \{x_0, \overline{x}_0, \ldots, x_{n-1}, \overline{x}_{n-1}\} \tag{2}$$

enforcing the root to be enabled and to contain the empty assignment.

Internal block axioms. For each of the $s(h-1)+1$ internal blocks, we have a set of axioms encoding that each of its children extends the partial assignment in the block to the next $\log p$ variables in a different way.

[1]This may seem like a further distinction from the $\mathrm{Ref}(F)$ formula of [AM20, dRGN+21], where blocks contained clauses not partial assignments, but it is a purely cosmetic one since there is a one-to-one correspondence between both objects. In our setting it is more natural to consider partial assignments.

Let $B = B_{\alpha,i}$ be an internal block. For $\sigma \in [p]$, we have the clause

$$\bigvee_{j \in N_\Gamma(ip+\sigma)} \mathsf{pntr}_j^{B,\sigma} \tag{3}$$

encoding that the σth child should exist. Now let $\mathcal{L}_\sigma = \{x_{\alpha \log p}^{\sigma_0}, x_{\alpha \log p+1}^{\sigma_1}, \ldots, x_{\alpha \log p + \log p - 1}^{\sigma_{\log p - 1}}\}$ be the set of literals that are not in B and that we expect the σth child of B to contain, and let $\overline{\mathcal{L}}_\sigma = \{x_0, \overline{x}_0, \ldots, x_{n-1}, \overline{x}_{n-1}\} \setminus \mathcal{L}_\sigma$. For succinctness, we denote by B_j the block $B_{\alpha+1,j}$.

For every $\sigma \in [p]$, and every $j \in N_\Gamma(ip+\sigma)$ we have the following set of axioms

$$\left(\mathsf{enbl}^B \wedge \mathsf{pntr}_j^{B,\sigma}\right) \to \mathsf{enbl}^{B_j} \tag{4}$$

$$\left(\mathsf{enbl}^B \wedge \mathsf{pntr}_j^{B,\sigma}\right) \to \mathsf{lit}_\ell^{B_j} \qquad \forall \ell \in \mathcal{L}_\sigma \tag{5}$$

$$\left(\mathsf{enbl}^B \wedge \mathsf{pntr}_j^{B,\sigma} \wedge \mathsf{lit}_\ell^B\right) \to \mathsf{lit}_\ell^{B_j} \qquad \forall \ell \in \overline{\mathcal{L}}_\sigma \tag{6}$$

$$\left(\mathsf{enbl}^B \wedge \mathsf{pntr}_j^{B,\sigma} \wedge \neg\mathsf{lit}_\ell^B\right) \to \neg\mathsf{lit}_\ell^{B_j} \qquad \forall \ell \in \overline{\mathcal{L}}_\sigma \tag{7}$$

encoding that if B is enabled and B_j is the σth child of B, then B_j must be enabled and contain exactly the literals in $\mathcal{L}_\sigma$ union the literals contained in B. Note that, to make the semantics clearer, we use the notation $C \to D$, for C being a conjunct and D a clause, to denote the clause $\neg C \vee D$. This yields a total of $(s(h-1)+1)p$ clauses of width at most $\Delta = O(1)$ and at most $(s(h-1)+1)(p\Delta)(1+4n-\log p) = O(nhsp)$ clauses of width at most 4.

Circuit axioms. For every $i \in [s]$ we have a set of axioms encoding that the ith copy of C computes correctly on the assignment on the leaf block $B_{h,i}$. Let $B = B_{h,i}$. For the sake of brevity, we refer to the gth gate of C simply as g.

For every $g \in [m]$, we have one of the following constraints depending on what kind of gate g is.

If g is an input gate	$\mathsf{gate}_g^B = \mathsf{lit}_\ell^B$,	where ℓ is the input variable of g.	(8)
If g is a NOT gate	$\mathsf{gate}_g^B = \neg\mathsf{gate}_{g'}^B$,	where g' is the input gate of g.	(9)
If g is an AND gate	$\mathsf{gate}_g^B = \mathsf{gate}_{g'}^B \wedge \mathsf{gate}_{g''}^B$,	where g' and g'' are the input gates of g.	(10)
If g is an OR gate	$\mathsf{gate}_g^B = \mathsf{gate}_{g'}^B \vee \mathsf{gate}_{g''}^B$,	where g' and g'' are the input gates of g.	(11)

Finally, we have one extra constraint $\mathsf{gate}_g^B = 0$, where g is the sink gate of C, enforcing that the circuit output 0. Note that each of these $m+1$ constraints can be encoded by at most 3 clauses, all of width at most 3.

Summing up, since $h = \sqrt{n}$, $p = 2^{n/h} = 2^{\sqrt{n}}$, $s = mp^2 = m2^{2\sqrt{n}}$ and $m \geq n$, we can conclude that ShallowRef(C) has $\mathrm{poly}(m) \cdot 2^{O(\sqrt{n})}$ clauses, all of width $O(1)$.

4 Upper bound for ShallowRef(C)

In this section we prove the upper bound in Theorem 2.4, namely we show that, if C is a satisfiable n-variate circuit of size m, there is a tree-like resolution refutation of ShallowRef(C) of size at most $\mathrm{poly}(m) \cdot 2^{O(\sqrt{n})}$. We again consider the parameters $h = \sqrt{n}$, $p = 2^{n/h}$ and $s = mp^2$, and fix the $(r, \Delta, \Delta/2)$-boundary expander $\Gamma = \Gamma_s^{sp}$ given by Theorem 2.3, where we recall that $r = \Theta(s)$ and $\Delta = \Theta(1)$.

We prove the upper bound by considering the *falsified clause search problem* on the formula, that is, the problem of, given a total assignment to the variables of the formula, finding a falsified clause. A decision tree that solves the falsified clause search problem on an unsatisfiable formula F is a decision tree that queries the variables of F and, at each leaf, outputs a clause of F that is falsified by the partial assignment given by the queries that lead to this leaf.

We proceed to describe a decision tree solving the falsified clause search problem on $\mathrm{ShallowRef}(C)$ that is of size $\mathrm{poly}(m) \cdot 2^{O(h)}$, and, by the classical equivalence between decision trees solving the falsified clause search problem on F and tree-like resolution refutations of F (see, e.g., [BIW04, Lemma 7]), item (i) of Theorem 2.4 follows directly.

Intuition. The main idea is to fix a satisfying assignment x^* of the circuit C and then walk down the blocks of $\mathrm{ShallowRef}(C)$ from the root block to a leaf block, keeping the invariant that the current block contains a partial assignment of x^*. At each step, since the axioms of $\mathrm{ShallowRef}(C)$ guarantee that it is possible to extend the current partial assignment to the next $\log p$ variables in every possible way, we can either move to a child-block that contains a larger partial assignment of x^* or we already find a falsified axiom of $\mathrm{ShallowRef}(C)$. If we reach a leaf block of $\mathrm{ShallowRef}(C)$, the partial assignment at this block is all of x^* and we can therefore detect a falsified axiom of $\mathrm{ShallowRef}(C)$ with a decision tree of size $O(m)$. Indeed, since x^* satisfies C, we can evaluate each gate of C and either find a gate that computes its output incorrectly, or find that the circuit C outputs 1. Either way, we have found a falsified axiom of $\mathrm{ShallowRef}(C)$. The size of this tree is approximately the number of different root-to-leaf paths, that is, $\Delta^h = 2^{O(h)}$, multiplied by $O(m)$.

Formal proof. Let x^* be a satisfying assignment of C, and let $\mathcal{L}$ be the set of n literals satisfied by x^* ordered according to the index (that is, the ith literal of $\mathcal{L}$ is either x_i or $\overline{x}_i$). We say a node of the decision tree is open if the partial assignment given by the queries leading to this node does not falsify any axiom of $\mathrm{ShallowRef}(C)$. Otherwise, we say the node is closed or is a leaf.

Let $B = B_{0,0}$ be the the root of $\mathrm{ShallowRef}(C)$. The decision tree starts by querying the enabled-variable enbl^B (one query and leaves one open node since there is an axiom of $\mathrm{ShallowRef}(C)$ that says the root is enabled). Then it queries the indicator variables lit_ℓ^B for all $2n$ literals ℓ. The branches in which any of these queried literals are present falsifies the axiom of $\mathrm{ShallowRef}(C)$ that says that the root does not contain this literal. Therefore, so far we have a decision tree with $2n+1$ leaves and one open node. We label this open node with B.

The decision tree then repeats the following iterative process, keeping the invariant that at the beginning of each iteration the current open node is labeled with some internal block $B = B_{\alpha,i}$ and the partial assignment leading to this node determines that B is enabled and that the literals present in B are exactly the $\alpha \log p$ first literals of $\mathcal{L}$.

Let $\sigma \in [p]$ be such that x^* assigns to the variable $x_{\alpha \log p + j}$ the value σ_j for all $j \in [\log p]$. The decision tree queries one by one the $\leq \Delta$ variables of the σth pointer of B, that is, variables $\mathsf{pntr}_j^{B,\sigma}$ for $j \in N_\Gamma(ip + \sigma)$, until it finds one that is 1 or finds that all are 0. The node corresponding to all pointer variables being 0 contradicts the axiom that states that the σth child should exist. For each $j \in N_\Gamma(ip + \sigma)$ let v_j be the node reached if $\mathsf{pntr}_j^{B,\sigma} = 1$. At each of these nodes, we repeat the queries done for the root. To be precise, fix an open node v_j and let $B' = B_{\alpha+1,j}$. The decision tree queries the enabled variable $\mathsf{enbl}^{B'}$ and the indicator variables $\mathsf{lit}_\ell^{B'}$ for the $2n$ literals. Observe that if B' is not enabled or does not contain exactly the first $(\alpha+1)\log p$ first literals of $\mathcal{L}$, then we immediately falsify one of the clauses of $\mathrm{ShallowRef}(C)$. Therefore, we are left with one open node, say v'_j, which we label with B'. If B' is not a leaf block, v'_j satisfies the invariant and we repeat the process. We note that the subtree constructed in each iteration has $\leq \Delta(2n+1)+1$ leaves and $\leq \Delta$ open nodes.

After h steps, we reach a leaf block $B_{h,i}$. Here the decision tree queries all circuit variables c_g^B, starting from the input gates and only querying gates for which its input gate(s) have already been queried. In this way, we obtain a subtree with $m+1$ leaves: m leaves correspond to a gate that did not correctly compute its output given its input(s) and one leaf corresponds to all gates computing correctly and, since x^* satisfies C, the circuit outputting 1. For each of these leaves there is some clauses of ShallowRef(C) that is falsified.

This completes the decision tree since there are no more open nodes. The number of leaves of the tree is at most

$$\left(\sum_{i=0}^{h-1} \Delta^i(2n+1)+1\right) + \Delta^h(m+1) = O(m+n)\cdot\Delta^h \quad , \tag{12}$$

and, therefore, the size of the tree is at most $O(m+n)\cdot\Delta^h = \mathrm{poly}(m)\cdot 2^{O(\sqrt{n})}$. This concludes the proof of the upper bound in Theorem 2.4.

5 Lower bound for ShallowRef(C)

We now prove the lower bound in Theorem 2.4, which states that, if C is an unsatisfiable n-variate circuit, then any DAG-like resolution refutation of ShallowRef(C) requires size $2^{\Omega(n)}$. The main step in this proof is to show a resolution width lower bound and the size lower bound then follows by the size-width relation of Ben-Sasson and Wigderson.

Proposition 5.1 ([BW01])**.** *Any DAG-like resolution refutation of a CNF formula F on n variables has size at least* $\exp\left(\Omega\left(\frac{(\mathrm{w}(F\vdash\bot)-\mathrm{w}(F))^2}{n}\right)\right)$.

Let C be an unsatisfiable n-variate circuit of size m. Fix the parameters $p = 2^{\sqrt{n}}$ and $s = mp^2$. For the rest of this section we focus on proving that resolution requires width

$$\mathrm{w}(\mathrm{ShallowRef}(C)\vdash\bot) = \Omega(s/(\sqrt{n}\log s)) \tag{13}$$

to refute ShallowRef(C). Since ShallowRef(C) has width $O(1)$ and has $O(nmsp) = O(ns^2/p)$ many variables, by Proposition 5.1 this implies a resolution size lower bound of

$$\exp\left(\Omega\left(\frac{p}{n^2\log^2 s}\right)\right) \geq \exp\left(\Omega\left(\frac{2^{\sqrt{n}}}{n^4}\right)\right) \geq \exp\left(\Omega\left(n\right)\right) \quad , \tag{14}$$

where for the first inequality we use the fact that $m \leq 2^n$ and thus $\log s = O(n)$.

We prove the width lower bound (13) via a reduction to a convenient version of the pigeonhole principle. The formula we use is a graph version of the *retraction pigeonhole principle*, rPHP [Jeř07, PT19]. The retraction pigeonhole principle was also used in [dRGN$^+$21] to prove a similar reduction. We note, however, that we require a weaker version (with more pigeons) and that we define it over a graph in order to match the definition of ShallowRef(C).

We assume the reader is familiar with standard notions of Boolean functions (see, e.g., [Juk12, §14]), in particular, that a depth d decision tree computing a Boolean function f naturally gives both a d-CNF and a d-DNF formula computing f.

5.1 Weak graph rPHP formula

In this variant of the pigeonhole principle, the pigeon-mapping is restricted to the edges of a given bipartite graph and, moreover, there is an efficient way to *invert* the mapping. Formally, given a bipartite graph $\Gamma = (U \dot\cup V, E)$ with $U = [m]$ and $V = [n]$, the variables of the formula rPHP(Γ) are:

- *edge variables:* for every edge $(i, j) \in E$ there is a variable $x_{i,j}$; and
- *hole mapping:* for every hole $j \in [n]$ there are variables $g_{j,\ell}$, $\ell \in [\log m]$.

The variables $x_{i,j}$ have the intended meaning that pigeon i is mapped to hole j, and the variables $g_{j,\ell}$ for $\ell \in [\log m]$ encode the pigeon that is mapped to hole j and can thus be seen as describing a function $g\colon [n] \to [m]$. The formula $\mathrm{rPHP}(\Gamma)$ states that for each pigeon $i \in U$

$$\bigvee_{j \in N_\Gamma(i)} x_{i,j} \ , \tag{15}$$

that is, it must be mapped to (at least) one of its neighbors, and that for every $(i, j) \in E$,

$$x_{i,j} \implies g(j) = i \tag{16}$$

or, equivalently,

$$\overline{x}_{i,j} \ \vee \bigwedge_{\ell \in [\log m]} (g_{j,\ell} = i_\ell) \ . \tag{17}$$

Note that (17) corresponds to $\log m$ clauses each of width 2. Therefore, if the left degree of Γ is at most Δ, $\mathrm{rPHP}(\Gamma)$ can be written as an $O(\Delta)$-width CNF formula with $O(n\Delta \log m)$ clauses.

It was shown in [BW01] that if Γ is a good enough expander, then the pigeonhole principle formula over Γ requires large width to be refuted in resolution. The proof presented in [BW01] is for the classical encoding of the graph pigeonhole principle, but it is straightforward to see that it also holds for $\mathrm{rPHP}(\Gamma)$.

Lemma 5.2 ([BW01]). *If Γ is an (r, Δ, c)-bipartite boundary expander, resolution requires width $rc/2$ to refute* $\mathrm{rPHP}(\Gamma)$*.*

5.2 Decision tree reductions

Our goal is to show, via a moderately small-depth reduction that the width required for refuting $\mathrm{ShallowRef}(C)$ is almost as large as the width required for refuting $\mathrm{rPHP}(\Gamma_s^{sp})$. For the definition of reduction, we view a clause as a function, that is, given a CNF formula G over m variables, we view a clause $D \in G$ as a function from $\{0,1\}^m$ to $\{0,1\}$, which can be computed by a depth-$|D|$ decision tree. In this way, given a function $r : \{0,1\}^n \to \{0,1\}^m$, $D \circ r$ is a function from $\{0,1\}^n$ to $\{0,1\}$. Moreover, if r_i is computed by a depth-d decision tree, then $D \circ r$ can be naturally written as a $(d \cdot |D|)$-CNF formula over n variables.

Definition 5.3 ([dRGN+21]). Given two CNF formulas F and G over n and m variables respectively, there is a *depth-d reduction* from F to G, denoted $F \leq_d G$, if there is a function $r : \{0,1\}^n \to \{0,1\}^m$ such that

- each output bit of $r_i : \{0,1\}^n \to \{0,1\}$ for $i \in [m]$ is computed by a depth-d decision tree; and
- for every axiom $D \in G$, each clause in the representation of $D \circ r$ as a $(d \cdot |D|)$-CNF formula (as discussed above) is a weakening of some axiom of F.

The important property of reductions that we use is that width bounds carry over.

Lemma 5.4 ([dRGN+21]). *If $F \leq_d G$, then $\mathrm{w}(F \vdash \bot) \leq d \cdot \mathrm{w}(G \vdash \bot)$.*

For the rest of this section, let $\Gamma = \Gamma_s^{sp}$ be the boundary expander given by Theorem 2.3. Combining Lemmas 5.2 and 5.4 implies that, in order to obtain (13), it is enough to show that

$$\mathrm{rPHP}(\Gamma) \ \leq_{O(\sqrt{n}\log s)} \ \mathrm{ShallowRef}(C) \ . \tag{18}$$

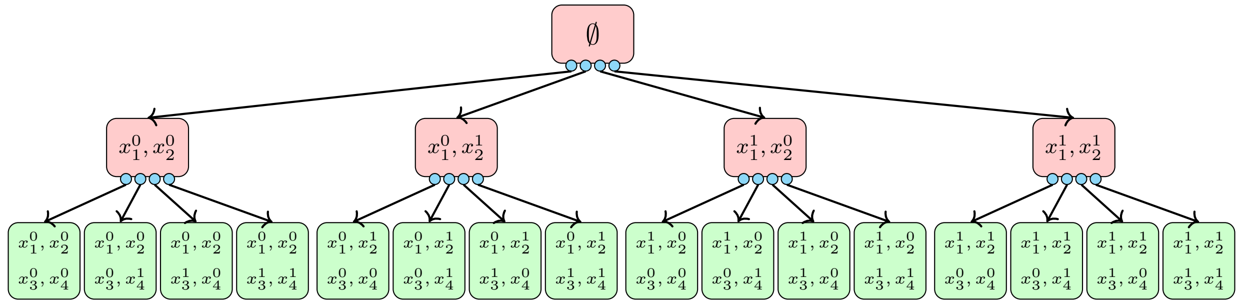
$\emptyset$
x_1^0, x_2^0
x_1^0, x_2^1
x_1^1, x_2^0
x_1^1, x_2^1
x_1^0, x_2^0 x_3^0, x_4^0
x_1^0, x_2^0 x_3^0, x_4^1
x_1^0, x_2^0 x_3^1, x_4^0
x_1^0, x_2^0 x_3^1, x_4^1
x_1^0, x_2^1 x_3^0, x_4^0
x_1^0, x_2^1 x_3^0, x_4^1
x_1^0, x_2^1 x_3^1, x_4^0
x_1^0, x_2^1 x_3^1, x_4^1
x_1^1, x_2^0 x_3^0, x_4^0
x_1^1, x_2^0 x_3^0, x_4^1
x_1^1, x_2^0 x_3^1, x_4^0
x_1^1, x_2^0 x_3^1, x_4^1
x_1^1, x_2^1 x_3^0, x_4^0
x_1^1, x_2^1 x_3^0, x_4^1
x_1^1, x_2^1 x_3^1, x_4^0
x_1^1, x_2^1 x_3^1, x_4^1

Figure 2: Full tree $\mathcal{T}$, defined over 4 variables, of height 2 and arity 4.

5.3 Overview of the reduction

The proof of the reduction is similar to that of [dRGN+21]. The main difference is that in [dRGN+21] the reduction is guided by a full binary tree of height n and our reduction is guided by a full tree $\mathcal{T}$ of height $h = \sqrt{n}$ and arity $p = 2^{n/h}$. Recall that we consider a fixed ordering $x_0, \ldots, x_{n-1}$ of the variables of C. Each node of $\mathcal{T}$ contains a partial assignment, which, as before, we identify with the set of literals that are satisfied by the assignment. The root contains the empty assignment and each node at distance α from the root contains an assignment to the first $\alpha \log p$ variables defined recursively as follows. If a non-leaf node at distance α from the root contains the partial assignment ρ to the first $\alpha \log p$ variables, each of its p children contains one of the possible extensions of ρ to the next $\log p$ variables, that is, for $\sigma \in [p]$, the σth child contains the assignment $\rho \cup \{x_{\alpha \log p}^{\sigma_0}, x_{\alpha \log p+1}^{\sigma_1}, \ldots, x_{\alpha \log p + \log p - 1}^{\sigma_{\log p - 1}}\}$. Thus, $\mathcal{T}$ has 2^n leaves each containing a different full assignments to the n variables of C (see Figure 2 for an example).

The formula ShallowRef(C) is, intuitively, saying that this tree $\mathcal{T}$ "fits" in a tree with s blocks per layer, that is, that there is a mapping from nodes of $\mathcal{T}$ to blocks of ShallowRef(C). At a high-level, the reduction defines an rPHP(Γ) instance between every two consecutive layers of ShallowRef(C) so that refuting ShallowRef(C) amounts to refuting one of the instances of rPHP(Γ).

We now prove the reduction formally. For each variable of ShallowRef(C), we describe a $O(n)$-depth decision tree that computes its value by querying variables of rPHP(Γ). This defines the reduction function r and shows it satisfies the first condition of Definition 5.3. We then argue that r also satisfies the second condition.

5.4 The reduction function

We start by describing the decision trees that compute the variables of ShallowRef(C). For simplicity, we sometimes describe one decision tree that computes several variables at once. Recall that the variables of rPHP(Γ) are $x_{i,j}$ for $(i,j) \in E(\Gamma)$ and $g_{j,\ell}$ for $j \in [s]$ and $\ell \in [\log sp]$, and that the latter set of variables defines a function $g\colon [s] \to [sp]$.

Pointer variables. The pointer variables are defined in terms of the variables of rPHP(Γ). Since each block has p pointers, each layer $\alpha \in \{1, \ldots, h-1\}$ has a total of sp pointers (layer 0 has only one block and thus only p pointers) that should be mapped to the s blocks at level $\alpha + 1$. The idea is to define this mapping by "copying" the rPHP(Γ) mapping from the sp pigeons to the s holes. Formally, for $\alpha \in \{0, 1, \ldots, h-1\}$, let $B = B_{\alpha,i}$ be an internal block, let $\sigma \in [p]$ and let $k = ip + \sigma$. For all $j \in N_\Gamma(k)$, we define the pointer variable $\mathsf{pntr}_j^{B,\sigma} = x_{k,j}$. This can clearly be done by a decision tree of depth 1.

Enabled and literal set variables. We have now reached the most complicated part of the reduction. Consider the function $g' \colon [s] \to [sp] \cup \{\star\}$ defined as

$$g'(j) = \begin{cases} g(j) & \text{if } j \in N_\Gamma(g(j)) \text{ and } x_{g(j),j} = 1 \\ \star & \text{otherwise.} \end{cases} \tag{19}$$

In order to determine whether a block $B = B_{\alpha,j}$ is enabled or not and what literal set it contains, we recursively define a path of blocks and pointers starting from B, denoted $\mathsf{path}(B)$, in the following way. If B is the root or if $g'(j) = \star$, then the path ends at B, that is, $\mathsf{path}(B) = B$. Otherwise, let

$$j' = \left\lfloor \frac{g'(j)}{s} \right\rfloor \qquad\qquad \text{and} \qquad\qquad \sigma = g'(j) \mod s \ . \tag{20}$$

Note that j' and σ are defined so that the σth pointer of the block $B' = B_{\alpha-1,j'}$ is j, that is, $\mathsf{pntr}_j^{B',\sigma} = x_{g(j),j} = 1$. We define $\mathsf{path}(B)$ to be (B, σ) concatenated with $\mathsf{path}(B')$.

We are now ready to define a decision tree that computes the enabled variable and the literal set variables of the block B. In order to do so, the decision tree must compute $\mathsf{path}(B)$, which can be done in depth $O(h \log s) = O(\sqrt{n} \log s)$. Indeed, the length of $\mathsf{path}(B)$ is at most h and to determine the next block in a path it is enough to compute g', which can be done with $O(\log sp) = O(\log s)$ queries to the variables of rPHP(Γ).

Once $\mathsf{path}(B)$ is determined, the decision tree outputs the value of the variables as follows. If $\mathsf{path}(B)$ does not end at the root, the enabled variable enbl^B and the literal variables lit_ℓ^B for $\ell \in \{x_0, \overline{x}_0, \ldots, x_{n-1}, \overline{x}_{n-1}\}$ are all set to 0. If $\mathsf{path}(B)$ ends at the root, let π be the reverse of $\mathsf{path}(B)$. Note that π defines a unique path in the tree $\mathcal{T}$ starting from the root and following the pointers in π. This path in $\mathcal{T}$ ends in some block B' at level α. The decision tree sets the enabled variable enbl^B to 1 and sets the literal variables lit_ℓ^B for $\ell \in \{x_0, \overline{x}_0, \ldots, x_{n-1}, \overline{x}_{n-1}\}$ to match the assignment of the block B'.

Circuit variables. The circuit variables are computed "honestly" given the assignments contained at the leaf blocks. That is, in order to determine the circuit variables for the leaf block $B = B_{h,i}$, the decision tree computes $\mathsf{path}(B)$ (in depth $O(\sqrt{n} \log s)$), in this way determining the literal variables of B, and then it outputs the value of each circuit variable gate_g^B to be consistent with the literal variables. We note that, if $\mathsf{path}(B)$ does not end at the root, then the literal variables of B are all set to 0 and the circuit variable gate_g^B are simply the value of the gate on the all zero input.

Therefore, we conclude that the reduction r satisfies the required condition that for each i there is a decision of depth $O(n)$ that computes r_i.

5.5 Correctness of the reduction

We now show that the reduction satisfies the second condition of Definition 5.3. Recall that r is the function defined by the decision trees described above. Let D be an axiom of ShallowRef(C). It is enough to show that for every partial assignment ρ to the variables of rPHP(Γ), if ρ falsifies $D \circ r$ then ρ falsifies an axiom of rPHP(Γ).

We first note that, by definition of r, if D is a root axiom or a circuit axiom, then $D \circ r$ is a tautology, that is, for any full assignment ρ to the variables of rPHP(Γ), the formula $D \circ r$ is satisfied. It remains to consider the case when D is an internal block axiom. If D is one of the axioms (3) of ShallowRef(C), that is, the axiom $\bigvee_{j \in N_\Gamma(ip+\sigma)} \mathsf{pntr}_j^{B,\sigma}$ for some block $B = B_{\alpha,i}$ and some $\sigma \in [p]$, then we have found that ρ does not map pigeon $ip + \sigma$ to any of its neighbors (falsifying axiom (15) of rPHP(Γ)). If D is one of the axioms (4)–(7) of ShallowRef(C) involving some block $B = B_{\alpha,i}$, some $\sigma \in [p]$ and some block $B_j = B_{\alpha+1,j}$, it must be the case that, under ρ, block B is enabled (i.e., $\mathsf{enbl}^B = 1$ and thus $\mathsf{path}(B)$ reaches the root), its σth child is block B_j (i.e. $\mathsf{pntr}_j^{B,\sigma} = 1$), and either B_j is not enabled (falsifying axiom (4)) or it does not contain the right set of literals (falsifying one of the axioms (5)–(7)). Let $k = ip + \sigma$. Since $\mathsf{pntr}_j^{B,\sigma} = 1$, by the definition of the reduction r, it follows that $x_{k,j} = 1$. Now if $g(j) = k$, then $\mathsf{path}(B_j)$ would have been (B_j, σ) concatenated with $\mathsf{path}(B)$. But by the definition of r, this would imply that B_j reaches the root (hence is enabled) and that it contains the set of literals compatible with being the σth child of B. This contradicts the fact that one of the axioms (4)–(7) is falsified, so it must be the case that $g(j) \neq k$ and thus one of the axioms (17) of rPHP(Γ) that encode that $x_{k,j} \Longrightarrow g(j) = k$ must be falsified. This completes the proof of the reduction.

6 Generalization for polynomial calculus

We now prove that the lower bound of Theorem 2.4 holds for the polynomial calculus proof system [CEI96, ABRW02]. In this section, all polynomials are over the polynomial ring $\mathbb{F}[X]$, where $\mathbb{F}$ is a fixed field and X is a set of formal variables. We define the *size* of a polynomial p, denoted by $\|p\|$, to be the number of non-zero monomials of p when expanded as a linear combination of monomials.

Polynomial Calculus. A polynomial calculus (PC) *derivation* (over $\mathbb{F}$) of a polynomial equation $p = 0$ from a set of polynomial equations $F = \{f_0 = 0, \ldots, f_{m-1} = 0\}$ is a sequence of polynomials $\mathcal{P} = (p_0, \ldots, p_{t-1})$ such that $p_{t-1} = p$ and for each $i \in [t]$ either:

- *Axiom:* $p_i = 0 \in F$; or
- *Linear combination:* $p_i = \alpha p_j + \beta p_{j'}$ for some $\alpha, \beta \in \mathbb{F}$ and $j, j' < i$; or
- *Multiplication:* $p_i = x p_j$ for some $x \in X$ and $j < i$.

The *size* of the derivation is $\|\mathcal{P}\| \coloneqq \sum_{i\in[t]} \|p_i\|$ and its *degree* is $\deg(\mathcal{P}) \coloneqq \max_{i\in[t]} \deg(p_i)$. A PC *refutation* of F is a PC derivation of $1 = 0$ from F. We denote by $\deg(F \vdash \bot)$ the minimum degree of any PC refutation of F. We sometimes refer to the equation in F as axioms.

Given a CNF formula F over variables $\mathrm{var}(F)$ we consider the standard translation of F into a set of polynomial equations F^* that has the same set of satisfying assignments. For every variable $x \in \mathrm{var}(F)$, we include in F^* *Boolean axioms* $x^2 - x = 0$ (enforcing $x \in \{0, 1\}$). For every clause

$$\bigvee_{i\in I} x_i \vee \bigvee_{j\in J} \overline{x}_j \tag{21}$$

in F, the set F^* contains the equation

$$\prod_{i\in I}(1 - x_i) \cdot \prod_{j\in J} x_j = 0 \ , \tag{22}$$

where we interpret 1 as true and 0 as false.

We also include *twin variables*, that is, formal variables $\overline{x}$ for every variable $x \in \mathrm{var}(F)$, with the corresponding Boolean axioms $\overline{x}^2 - \overline{x} = 0$ and *complementary axioms* $x + \overline{x} - 1 = 0$ (enforcing x and $\overline{x}$ take complementary values). Henceforth, we often do not distinguish between F and its translation F^*. We note that using twin variable we could have the equation $\prod_{i\in I} \overline{x}_i \cdot \prod_{j\in J} x_j = 0$ instead of (22), but for our purposes the precise encoding is not relevant since the CNF formulas we consider have constant width and thus it is possible to derive one from the other in constant degree and size. It was shown in [ABRW02] that PC with twin variables—sometimes referred to as PCR—polynomially simulates resolution with respect to size.

The strategy for proving the PC size lower bound is the same as for resolution. We first prove a degree lower bound and then, by the size-degree relation of [IPS99], we obtain a size lower bound.

Proposition 6.1 ([IPS99]). *Any polynomial calculus refutation of a CNF formula F on n variables has size at least* $\exp\left(\Omega\left(\frac{(\deg(F\vdash\bot)-\deg(F))^2}{n}\right)\right)$.

Therefore, it is enough to prove a degree lower bound of

$$\deg(\mathrm{ShallowRef}(C)\vdash\bot)=\Omega(s/(\sqrt{n}\log s))\ , \tag{23}$$

where $s=mp^2$ and $p=2^{\sqrt{n}}$ are defined as before. To do so, we can use the same reduction as defined in Section 5 since it was shown in [dRGN+21] that decision tree reductions—and even the more general *algebraic reductions*—transfer polynomial calculus degree hardness.

Lemma 6.2 ([dRGN+21]). *If $F\le_d G$, then* $\deg(F\vdash\bot)\le d\cdot\deg(G\vdash\bot)$.

However, in order to apply this lemma to ShallowRef(C), and complete the proof of the size lower bound, we need a lower bound on the degree of any PC refutation of rPHP(Γ_s^{sp}). So for the rest of this section, we focus on showing that if Γ is an (r,Δ,c)-bipartite boundary expander, $\deg(\mathrm{rPHP}(\Gamma)\vdash\bot)\ge rc/2$.

We note that in terms of the degree required to refute a formula, it does not matter whether we include twin variables or not, since given a refutation with twin variables we can substitute every occurrence of a variable $\overline{x}$ by $(1-x)$ and this yields a valid refutation in the same degree. Therefore, when proving degree lower bounds we consider the encoding without twin variables.

First we consider the more classical encoding of the graph pigeonhole principle. Let $\Gamma=(U\dot\cup V,E)$ be a bipartite graph. We define aPHP(Γ) to be the following system of polynomial equations over variables $x_{i,j}$ for $(i,j)\in E(\Gamma)$:

$$\begin{array}{llllll}
Q_i & := \sum_{j\in N_\Gamma(i)} x_{i,j}-1 & = & 0 & \forall i\in U & \text{“\textit{each pigeon occupies a hole,}”}\\
Q_{i,i';j} & := x_{i,j}x_{i',j} & = & 0 & \forall j\in V;\forall i\neq i'\in N_\Gamma(j) & \text{“\textit{no hole houses two pigeons,}”}\\
Q_{i,j} & := x_{i,j}^2-x_{i,j} & = & 0 & \forall (i,j)\in E(\Gamma) & \text{“\textit{Boolean axioms.}”}
\end{array}$$

We can define rPHP(Γ) in a similar way, where we write $g^{(1)}:=g$ and $g^{(0)}:=1-g$ for short:

$$\begin{array}{llllll}
Q_i & := \sum_{j\in N_\Gamma(i)} x_{i,j}-1 & = & 0 & \forall i\in U & \text{“\textit{each pigeon occupies a hole,}”}\\
R_{i,j,\ell} & := x_{i,j}g_{j,\ell}^{(1-i_\ell)} & = & 0 & \forall (i,j)\in E(\Gamma);\forall \ell\in[\log|U|] & \text{“\textit{no hole houses two pigeons,}”}\\
Q_{i,j} & := x_{i,j}^2-x_{i,j} & = & 0 & \forall (i,j)\in E(\Gamma) & \text{“\textit{x Boolean axioms,}”}\\
R_{j,\ell} & := g_{j,\ell}^2-g_{j,\ell} & = & 0 & \forall j\in V;\forall \ell\in[\log|U|] & \text{“\textit{g Boolean axioms.}”}
\end{array}$$

We refer to the variables $g_{j,\ell}$ as type-g variables and to the variable $x_{i,j}$ as type-x variables.

Note that instead of translating the clauses $\bigvee_{j\in N_\Gamma(i)} x_{i,j}$ of rPHP(Γ) into $\prod_{j\in N_\Gamma(i)}(1-x_{i,j})=0$, we included the equation $\sum_{j\in N_\Gamma(i)} x_{i,j}=1$. The latter equation is a stronger constraint: it is enforcing that each pigeon be mapped to exactly one hole. Since the graphs we are interested in have constant left degree—hence these equations are over a constant number of variables—by derivational completeness of PC, it is possible to derive the former from the latter in constant degree.

Alekhnovich and Razborov [AR03] (see also [MN15]) proved a PC degree lower bound for refuting aPHP(Γ) when Γ is a good enough expander.

Lemma 6.3 ([AR03])**.** *If Γ is an (r, Δ, c)-bipartite boundary expander,* $\deg(\text{aPHP}(\Gamma) \vdash \perp) \geq rc/2$.

In [dRGN$^+$21] it was shown that (the standard, non-graph version of) aPHP can be reduced, via an *algebraic reduction*, to (the standard version of) rPHP. The same reduction also works for the graph versions and, therefore, by Lemma 6.3 PC requires large degree to refute rPHP(Γ) when Γ is a good enough expander. We include a self-contained proof that is essentially that of [dRGN$^+$21], except that in our setting it can be made even simpler so we avoid defining algebraic reductions and resorting to the generic "reduction preserves degree" lemma of [dRGN$^+$21].

Lemma 6.4. *If Γ is an (r, Δ, c)-bipartite boundary expander,* $\deg(\text{rPHP}(\Gamma) \vdash \perp) \geq rc/2$.

Proof. Let $\Gamma = (U \dot\cup V, E)$ be any bipartite graph (not necessarily an expander). We show that, given a PC refutation $\mathcal{P} = \{p_0, \ldots, p_{t-1}\}$ of rPHP(Γ) in degree d, we can construct a PC refutation $\mathcal{P}'$ of aPHP(Γ) also in degree d. The lemma then follows from Lemma 6.3.

For each polynomial $p_i \in \mathcal{P}$, let p_i' be the polynomial obtained by substituting the g-variables by $g_{j,\ell} \coloneqq \sum_{i \in I_\ell} x_{i,j}$, where $I_\ell \coloneqq \{i : (i,j) \in E(\Gamma), i_\ell = 1\}$ are the pigeons that can be mapped to j and that have the ℓth bit equal to 1. Each polynomial p_i' is now only on type-x variables and the degree has not increased, that is, $\deg(p_i') \leq \deg(p_i)$. We argue by induction over i that if p_i is derived from rPHP in degree d, then p_i' can be derived from aPHP in degree d.

- *Linear Combination.* If p_i is derived by linear combination from p_k and $p_{k'}$ for $k, k' < i$, then p_i' can be derived by the same linear combination from p_k' and $p_{k'}'$.
- *Multiplication.* If p_i is derived by multiplying p_k by a variable v for $k < i$ then, if v is a type-x variable, p_i' can be derived by multiplying p_k' by the same variable v; if $v = g_{j,\ell}$ is a type-g variable, p_i' can be derived by several applications of the multiplication rule to p_k', each by one of the variables $\{x_{i',j} : i' \in I_\ell\}$, and then adding these by repeatedly applying the linear combination rule.

In both cases degree of the derivation is preserved. It remains to argue that if p_i is an axiom of rPHP(Γ), then p_i' can be derived in low degree from aPHP(Γ). We only need to consider the axioms that involve type-g variables. We first consider the Boolean axioms of rPHP(Γ) for variables $g_{j,\ell}$, which can be derived as follows:

$$R_{j,\ell} = \; g_{j,\ell}^2 - g_{j,\ell} \; = \; \Big(\sum_{i \in I_\ell} x_{i,j}\Big)^2 - \sum_{i \in I_\ell} x_{i,j} \; = \; \sum_{i \in I_\ell} (x_{i,j}^2 - x_{i,j}) + \sum_{\substack{i,i' \in I_\ell \\ i \neq i'}} x_{i,j} x_{i',j} \; = \; \sum_{i \in I_\ell} Q_{i,j} + \sum_{\substack{i,i' \in I_\ell \\ i \neq i'}} Q_{i,i';j} \, .$$

In order to derive $x_{i,j} \cdot g_{j,\ell}^{(1-i_\ell)}$ (recall we write $g^{(1)} \coloneqq g$ and $g^{(0)} \coloneqq 1 - g$) we consider two cases.

- Case $i_\ell = 0$ (where $i \notin I_\ell$):

$$R_{i,j,\ell} \; = \; x_{i,j} \cdot g_{j,\ell}^{(1-i_\ell)} \; = \; x_{i,j} \cdot \sum_{i' \in I_\ell} x_{i',j} \; = \; \sum_{i' \in I_\ell} Q_{i,i';j} \; .$$

- Case $i_\ell = 1$ (where $i \in I_\ell$):

$$R_{i,j,\ell} \; = \; x_{i,j} \cdot g_{j,\ell}^{(1-i_\ell)} \; = \; x_{i,j} \Big(1 - \sum_{i' \in I_\ell} x_{i',j}\Big) \; = \; x_{i,j} - x_{i,j}^2 - \sum_{i' \in I_\ell \setminus \{i\}} x_{i',j} x_{i,j} \; = \; - \, Q_{i,j} - \sum_{i' \in I_\ell \setminus \{i\}} Q_{i,i';j} \; .$$

Since all derivations are in degree 2, the lemma follows. ☐

7 Concluding remarks

In this paper, we showed that automating tree-like resolution in time $n^{o(\log n)}$ is ETH-hard, even if the automating algorithm is allowed to produce polynomial calculus proofs. The lower bound required for this result was obtained via a width/degree lower bound and then an application of the size-width [BW01] or size-degree [IPS99] relation. It would also have been possible to derive the lower bound by *block-lifting/relativization*, as was done in [AM20, dRGN$^+$21]. In this setting, we think of the variables as partitioned into blocks and define complexity measures (width, degree, depth) with respect to these blocks. While this would yield a somewhat more elaborate proof, it would slightly improve some parameters.

The recent break-through result of Atserias and Müller [AM20] proved that resolution is NP-hard to automate. This result was extended to cutting planes [GKMP20], $\mathsf{Res}(k)$ (resolution over k-DNFs) for constant k [Gar20], Nullstellensatz, polynomial calculus and Sherali-Adams [dRGN$^+$21], and ordered resolution [Bel20]. There are many intriguing open problems regarding automatability: here we highlight only some of the most related to this work. Are there any other tree-like proof systems that are automatable in subexponential time? A generalization of the [BP96] algorithm that automates tree-like resolution in time $n^{O(\log n)}$ implies that tree-like $\mathsf{Res}(k)$ can be automated in time $n^{O(k \log n)}$ (see, e.g., [AB02]). Is this optimal? What can be said about tree-like $\mathsf{Res}(\oplus)$ (resolution over linear equations mod 2) or tree-like cutting planes? (For tree-like cutting planes some partial results were obtained in [GKMP20].) Are they automatable in, even randomized, subexponential time? Are they NP-hard to automate?

Acknowledgments. The author would like to thank Mika Göös, Jakob Nordström, Toni Pitassi, Robert Robere, Dmitry Sokolov and Navid Talebanfard for discussions on automatability, and Erfan Khaniki and Lukáš Folwarczný for reading a preliminary version of this paper. The author was supported by the Knut and Alice Wallenberg grant KAW 2018.0371.

References

[AB02] Albert Atserias and María Luisa Bonet. On the automatizability of resolution and related propositional proof systems. Technical Report TR02-010, Electronic Colloquium on Computational Complexity (ECCC), 2002. URL: https://eccc.weizmann.ac.il/report/2002/010/.

[AB04] Albert Atserias and María Luisa Bonet. On the automatizability of resolution and related propositional proof systems. *Information and Computation*, 189(2):182–201, March 2004. Preliminary version in *CSL '02*. doi:10.1016/j.ic.2003.10.004.

[ABMP01] Michael Alekhnovich, Sam Buss, Shlomo Moran, and Toniann Pitassi. Minimum propositional proof length is NP-hard to linearly approximate. *Journal of Symbolic Logic*, 66(1):171–191, 2001. doi:10.2307/2694916.

[ABRW02] Michael Alekhnovich, Eli Ben-Sasson, Alexander A. Razborov, and Avi Wigderson. Space complexity in propositional calculus. *SIAM Journal on Computing*, 31(4):1184–1211, 2002. doi:10.1137/S0097539700366735.

[ABRW04] Michael Alekhnovich, Eli Ben-Sasson, Alexander A. Razborov, and Avi Wigderson. Pseudorandom generators in propositional proof complexity. *SIAM Journal on Computing*, 34(1):67–88, 2004. Preliminary version in *FOCS '00*. doi:10.1137/S0097539701389944.

[ADF95] Karl A. Abrahamson, Rodney G. Downey, and Michael R. Fellows. Fixed-parameter tractability and completeness IV: On completeness for W[P] and PSPACE analogues. *Annals of Pure and Applied Logic*, 73(3):235–276, June 1995. doi:10.1016/0168-0072(94)00034-z.

[AM20] Albert Atserias and Moritz Müller. Automating resolution is NP-hard. *Journal of the ACM*, 67(5), 2020. Preliminary version in *FOCS '19*. doi:10.1145/3409472.

[AR03] Michael Alekhnovich and Alexander A. Razborov. Lower bounds for polynomial calculus: Non-binomial case. *Proceedings of the Steklov Institute of Mathematics*, 242:18–35, 2003. Preliminary version in *FOCS '01*. URL: http://mi.mathnet.ru/eng/tm403.

[AR08] Michael Alekhnovich and Alexander A. Razborov. Resolution is not automatizable unless W[P] is tractable. *SIAM Journal on Computing*, 38(4):1347–1363, 2008. Preliminary version in *FOCS '01*. doi:10.1137/06066850X.

[BDG$^+$04] María Luisa Bonet, Carlos Domingo, Ricard Gavaldà, Alexis Maciel, and Toniann Pitassi. Non-automatizability of bounded-depth Frege proofs. *Computational Complexity*, 13(1-2):47–68, December 2004. Preliminary version in *CCC '99*. doi:10.1007/s00037-004-0183-5.

[Bel20] Zoë Bell. Automating regular or ordered resolution is NP-hard. Technical Report TR20-105, Electronic Colloquium on Computational Complexity (ECCC), 2020. URL: https://eccc.weizmann.ac.il/report/2020/105/.

[BIW04] Eli Ben-Sasson, Russell Impagliazzo, and Avi Wigderson. Near optimal separation of tree-like and general resolution. *Combinatorica*, 24(4):585–603, September 2004. doi:10.1007/s00493-004-0036-5.

[BP96] Paul Beame and Toniann Pitassi. Simplified and improved resolution lower bounds. In *Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science (FOCS '96)*, pages 274–282, October 1996. doi:10.1109/SFCS.1996.548486.

[BPR00] María Luisa Bonet, Toniann Pitassi, and Ran Raz. On interpolation and automatization for Frege systems. *SIAM Journal on Computing*, 29(6):1939–1967, 2000. Preliminary version in *FOCS '97*. doi:10.1137/S0097539798353230.

[BW01] Eli Ben-Sasson and Avi Wigderson. Short proofs are narrow—resolution made simple. *Journal of the ACM*, 48(2):149–169, 2001. doi:10.1145/375827.375835.

[CEI96] Matthew Clegg, Jeff Edmonds, and Russell Impagliazzo. Using the Groebner basis algorithm to find proofs of unsatisfiability. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing (STOC 96)*, pages 174–183, 1996. doi:10.1145/237814.237860.

[CRVW02] Michael Capalbo, Omer Reingold, Salil Vadhan, and Avi Wigderson. Randomness conductors and constant-degree lossless expanders. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC '02)*, pages 659–668, May 2002. doi:10.1145/509907.510003.

[dRGN$^+$21] Susanna F. de Rezende, Mika Göös, Jakob Nordström, Toniann Pitassi, Robert Robere, and Dmitry Sokolov. Automating algebraic proof systems is NP-hard. In *Proceedings of the 53rd Annual ACM Symposium on Theory of Computing (STOC '21)*, June 2021. To appear. URL: https://eccc.weizmann.ac.il/report/2020/064/.

[EGG08] Kord Eickmeyer, Martin Grohe, and Magdalena Grüber. Approximation of natural W[P]-complete minimisation problems is hard. In *Proceedings of the 23rd Annual IEEE Conference on Computational Complexity (CCC '08)*, pages 8–18, June 2008. doi:10.1109/CCC.2008.24.

[FG06] Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Springer Berlin Heidelberg, 2006. doi:10.1007/3-540-29953-x.

[Gar20] Michal Garlík. Failure of feasible disjunction property for k-DNF resolution and NP-hardness of automating it. Technical report, Electronic Colloquium on Computational Complexity (ECCC), 2020. URL: https://eccc.weizmann.ac.il/report/2020/037/.

[GKMP20] Mika Göös, Sajin Koroth, Ian Mertz, and Toniann Pitassi. Automating cutting planes is NP-hard. In *Proceedings of the 52nd Annual ACM Symposium on Theory of Computing (STOC '20)*, pages 68–77, June 2020. doi:10.1145/3357713.3384248.

[GL10] Nicola Galesi and Massimo Lauria. On the automatizability of polynomial calculus. *Theory of Computing Systems*, 47(2):491–506, 2010. doi:10.1007/s00224-009-9195-5.

[IPS99] Russell Impagliazzo, Pavel Pudlák, and Jiří Sgall. Lower bounds for the polynomial calculus and the Gröbner basis algorithm. *Computational Complexity*, 8(2):127–144, 1999. doi:10.1007/s000370050024.

[Iwa97] Kazuo Iwama. Complexity of finding short resolution proofs. In *Proceedings of the 22nd International Symposium on Mathematical Foundations of Computer Science (MFCS '97)*, pages 309–318, 1997. doi:10.1007/BFb0029974.

[Jeř07] Emil Jeřábek. On independence of variants of the weak pigeonhole principle. *Journal of Logic and Computation*, 17(3):587–604, 2007. doi:10.1093/logcom/exm017.

[Juk12] Stasys Jukna. *Boolean Function Complexity: Advances and Frontiers*, volume 27 of *Algorithms and Combinatorics*. Springer, 2012. doi:10.1007/978-3-642-24508-4.

[KP98] Jan Krajícek and Pavel Pudlák. Some consequences of cryptographical conjectures for S_2^1 and EF. *Information and Computation*, 140(1):82–94, 1998. doi:10.1006/inco.1997.2674.

[MN15] Mladen Mikša and Jakob Nordström. A generalized method for proving polynomial calculus degree lower bounds. In *Proceedings of the 30th Annual Computational Complexity Conference (CCC '15)*, volume 33 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 467–487, June 2015. doi:10.4230/LIPIcs.CCC.2015.467.

[MPW19] Ian Mertz, Toniann Pitassi, and Yuanhao Wei. Short proofs are hard to find. In *Proceedings of the 46th International Colloquium on Automata, Languages, and Programming (ICALP '19)*, pages 84:1–84:16, 2019. doi:10.4230/LIPIcs.ICALP.2019.84.

[PT19] Pavel Pudlák and Neil Thapen. Random resolution refutations. *Computational Complexity*, 28(2):185–239, 2019. doi:10.1007/s00037-019-00182-7.