Electronic Colloquium on Computational Complexity, Report No. 186 (2024)

Supercritical Tradeoffs for Monotone Circuits

Mika Göös
EPFL

Gilbert Maystre
EPFL

Kilian Risse
EPFL

Dmitry Sokolov
EPFL

November 21, 2024

Abstract

We exhibit a monotone function computable by a monotone circuit of quasipolynomial size such that any monotone circuit of polynomial depth requires exponential size. This is the first size–depth tradeoff result for monotone circuits in the so-called *supercritical* regime. Our proof is based on an analogous result in proof complexity: We introduce a new family of unsatisfiable 3-CNF formulas (called *bracket formulas*) that admit resolution refutations of quasipolynomial size while any refutation of polynomial depth requires exponential size.

1 Introduction

In monotone circuit complexity, a *size–depth tradeoff* result states that monotone circuits cannot be compressed to shallow depth without blowing up their size. (See Section 1.2 for examples from prior work.) Our main result is such a size–depth tradeoff in the so-called *supercritical* regime of parameters. This type of tradeoff was first envisioned by [BBI16, Raz16] and refers to a blow-up in one parameter S (for us, size) when another parameter D (for us, depth) is restricted to a value above its *critical value*, defined as the largest value that D can take when maximised over all n-bit functions (and when S is unrestricted). Concretely, if we consider monotone circuits of fanin 2, the worst-case upper bound for depth is $D \leq \Theta(n)$: any n-bit monotone function can be computed by a monotone circuit of depth n, and some (e.g., random) functions require depth $\Omega(n)$. We show that restricting the depth to any polynomial bound, $D \leq n^c$ where $c > 1$ is any constant, can still cause a huge blow-up in size.

Theorem 1 (Main result for circuits)**.** *There is a monotone $f\colon \{0,1\}^n \to \{0,1\}$ such that:*

(i) There is a monotone circuit computing f in size $n^{O(\log n)}$.
(ii) Every monotone circuit computing f in depth $n^{O(1)}$ requires size $\exp(n^{\Omega(1)})$.

Supercritical tradeoffs for monotone circuits were conjectured by [GGKS20, FGI+21, FPR22]. Our Theorem 1 does not yet, however, quantitatively reach the parameters stated in those conjectures. They asked for a blow-up to occur already for depth $D \leq S^\varepsilon$ where $\varepsilon > 0$ is a constant and S is the size of the circuit in *(i)*. We leave it for future work to quantitatively sharpen our tradeoff. Nevertheless, our result is the first supercritical tradeoff for monotone circuits (although see Section 1.4 for a discussion of a concurrent work [dRFJ+24]).

Our proof follows the by-now standard *lifting paradigm*: We start by proving an analogous tradeoff result in propositional proof complexity, and then apply a lifting theorem [GGKS20] to import that result to monotone circuit complexity. We state our proof complexity tradeoff for the standard *resolution* proof system; see the textbook [Juk12, §18] for an introduction.

ISSN 1433-8092

Theorem 2 (Main result for proofs)**.** *There is an n-variate 3-CNF formula F such that:*

(i') *There is a resolution refutation of F in size $n^{O(\log n)}$.*
(ii') *Every resolution refutation of F in depth $n^{O(1)}$ requires size $\exp(n^{\Omega(1)})$.*

The previous best size–depth tradeoffs for resolution were due to Fleming, Robere, and Pitassi [FPR22] with a more direct proof given by Buss and Thapen [BT24]. They obtain a tradeoff that is able to match (and go beyond) all our parameters in *(i')–(ii')*, but only for a CNF formula F having quasipolynomially many clauses. That is, their result does not apply when the depth restriction is superlinear in the size of F. As the authors discuss [FPR22, §7], this limitation prevents their results from lifting to a monotone circuit tradeoff. The question of overcoming this limitation has been asked several times previously [Raz16, BN20, FPR22]. Our result now addresses this issue, since our formula is a 3-CNF and hence of size $O(n^3)$. (The same limitation was also overcome in the concurrent work discussed in Section 1.4.)

1.1 New technique: Bracket formulas

To prove Theorem 2 we introduce a family of CNF formulas based on a novel *bracket principle*. Consider a string $s \in \{\texttt{[}, \texttt{[}, \texttt{]}, \texttt{]}\}^n$ of red and blue brackets that begins with a red opening bracket, $s_1 = \texttt{[}$, ends with a blue closing bracket, $s_n = \texttt{]}$, and is *well-parenthesised*: every opening bracket can be paired with a subsequent closing bracket of the same colour, and the bracket pairs are correctly nested (forbidding the pattern $\texttt{[}\cdots\texttt{[}\cdots\texttt{]}\cdots\texttt{]}$). For example,

$$s \;=\; \texttt{[[[]][]][][[[[]][]]]][[[[]]]][[][]]}.$$

We claim that every such string necessarily contains at least one occurrence of the substring $\texttt{][}$. To see this, we can erase all but the top-level brackets to obtain the string

$$\texttt{[}\qquad\texttt{]}\texttt{[}\texttt{]}\texttt{[}\qquad\qquad\texttt{]}\texttt{[}\qquad\texttt{]}\texttt{[}\qquad\texttt{]}.$$

Here it is clear that the region of red brackets has to transition over to the blue region, which is where we find the substring $\texttt{][}$.

CNF encoding with pointers. We can now encode the bracket principle as an unsatisfiable *bracket formula*, denoted Br_n. That is, we express as a CNF the negation of the tautology

$$(s_1 = \texttt{[}) \wedge (s_n = \texttt{]}) \wedge (s \text{ is well-parenthesised}) \quad\Longrightarrow\quad \exists i \in [n] \colon s_i s_{i+1} = \texttt{][}. \tag{1}$$

To help express the property that s is well-parenthesised, we equip each bracket with a pointer to its mate. That is, we actually consider strings over the extended alphabet $\Sigma \coloneqq \{\texttt{[}, \texttt{[}, \texttt{]}, \texttt{]}\} \times [n]$. For a string $s \in \Sigma^n$ to be well-parenthesised, we require that the pointers encode a correct pairing of the brackets. For example, if $s_i = (\texttt{[}, j)$, then we require that $s_j = (\texttt{]}, i)$ and $j > i$. The correct nesting property can be expressed by forbidding patterns for every 4-tuple of indices. Finally, if we encode symbols in the alphabet as $O(\log n)$-bit strings, it is straightforward to write down a $\mathrm{poly}(n)$-size $O(\log n)$-width CNF for $\neg(1)$. The width can be further reduced to 3 by using standard encoding tricks. The full formal definition of Br_n is given in Section 2.

Proof overview. We use the bracket formulas Br_n to prove Theorem 2. It is a recurring theme in proof complexity that the size of a resolution refutation Π is often closely related to its *width*, defined as the maximum width of a clause appearing in Π. Namely, any refutation of width w has size at most $n^{O(w)}$. Conversely, a formula F requiring resolution width w can be lifted (e.g., [GGKS20]) to a related formula F' requiring refutations of size $n^{\Omega(w)}$. (A somewhat

weaker converse holds also without lifting in case w is large enough [BW01].) We start by showing that the bracket formulas exhibit a *width–depth* tradeoff according to the following two theorems. The proofs appear in Sections 3 and 4, respectively.

Theorem 3. *The bracket formula* Br_n *admits a resolution refutation of width* $O(\log n)$.

Theorem 4. *Every width-*w *resolution refutation of* Br_n *has depth* $n^{\Omega\left(\frac{\log n}{\log w}\right)}$.

It is now straightforward to derive our main results (Theorems 1 and 2) by a black-box application of the lifting theorem from [GGKS20], which converts the above width–depth tradeoffs into size–depth tradeoffs. In fact, we get analogous tradeoffs also for the *cutting planes* proof system. The formal proof is given in Section 5.

Genealogy of bracket formulas. Perhaps surprisingly, we did not construct bracket formulas initially for the purpose of proving a tradeoff result. The bracket principle arose serendipitously from our unsuccessful attempts at classifying the *Jordan curve* (JC) theorem within the total search problem theory TFNP (specifically, the *crossing curves* variant defined by [ADD16]). While the complexity of JC remains wide open, we currently know [HMR24] that the bracket principle reduces to JC, and the complexity of the bracket principle falls somewhere between the *unique end-of-potential-line* class UEOPL [FGMS20] and the *end-of-potential-line* class EOPL = PLS ∩ PPAD [GHJ+24a]. For example, Theorem 3 can be interpreted as showing that the principle lies in the class PLS (which is characterised by low-width resolution; see [GHJ+24b] for a discussion). Proving the bracket principle complete for any existing TFNP class is an interesting open problem.

1.2 Related work: Monotone circuit complexity

We have claimed Theorem 1 as the first supercritical size–depth tradeoff for monotone circuits. If we are pedantic, however, and consider the model of *unbounded-fanin* monotone circuits, then the critical value for depth becomes 2: any monotone function can be computed by a monotone DNF, which has depth 2. Under this interpretation, classical results in monotone circuit complexity can indeed be viewed as supercritical tradeoffs for unbounded-fanin circuits. For a prominent example, a long line of work [KW88, RM99, GP18, dRNV16, dRMN+20] has culminated in an extreme separation of the monotone analogues of the classes NC and P. They exhibit an n-bit monotone function f computed by a linear-size monotone circuit such that any unbounded-fanin monotone circuit computing f in depth $n^{1-\varepsilon}$ (which is larger than the critical value 2) requires size $\exp(n^{\Omega(1)})$. Similar blow-ups in size also occur for constant-depth monotone circuits, as is known since the 1980s [KPPY84] with modern results given by Rossman [Ros15, Ros24].

Given that the notion of supercriticality was first conceived in the context of the resolution proof system [BBI16, Raz16] (where the critical value for depth/space is n), in this paper, we have chosen to reserve the term *supercritical* for when the critical value for depth is taken to be $\Theta(n)$. This is also how the term was used in the papers conjecturing a supercritical tradeoff for monotone circuits [GGKS20, FGI+21, FPR22]. (Note that our Theorem 1 holds no matter what fanin we consider: when we convert unbounded-fanin circuits to bounded-fanin circuits, the depth can increase only by a factor of $\log(\text{fanin}) \leq n$.).

1.3 Related work: Proof complexity

In proof complexity, the first supercritical tradeoffs were proven by Beame, Beck, and Impagliazzo [BBI16] (resolution size–space) and Razborov [Raz16] (tree-like resolution size/depth–width), with another early work by Berkholz [Ber12] (resolution depth–width). Since then, the

phenomenon has been studied extensively for resolution space [Raz17a, BN20, PR23], as well as for other proof systems such as polynomial calculus [BNT13], cutting planes [Raz17b, FPR22], and tree-like resolution over parities [CD24]. As already discussed above, the state-of-the-art for resolution size–depth are given by [FPR22, BT24].

Hardness condensation. In all results cited above (except [Ber12]), the basic technique to obtain supercritical tradeoffs is *hardness condensation*. This technique proceeds as follows.

(1) One starts with a n-variate formula F that exhibits a *sub*critical tradeoff between the two parameters of interest. Here, it suffices to consider standard formulas ubiquitous in proof complexity, such as *Tseitin* (used by [BBI16, Raz17a, BNT13]) or *pebbling* (used by [Raz16, BN20, FPR22, BT24, CD24]).

(2) One then *reduces the number of variables* of F while *preserving the original hardness*. Methods to reduce the number of variable include identifying groups of variables (projections) and composing the formula with small gadgets (e.g., XOR substitutions, pointers).

(3) The resulting formula F' now condenses the original hardness. The hardness parameter of F' is roughly the same as for F, but now the tradeoff has become supercritical: the parameters become larger as a function of the number of variables.

In this work, we do not explicitly employ hardness condensation. Our bracket formulas exhibit robust tradeoffs *naturally* right out of the box. It should be noted, however, that hardness condensation, broadly construed, is a *complete* method for proving size–depth tradeoffs. Any formula witnessing a size–depth tradeoff can be obtained by starting with a pebbling formula in Step (1) and then condensing using operations listed in Step (2).[1]

Resolution width–depth tradeoffs (from hardness condensation) are at the heart of recent breakthroughs in showing lower bounds on the stabilisation time of the Weisfeiler–Lehman algorithm for graph isomorphism [BN23, GLN23, GLNS23]. Can our bracket formulas find applications in this line of work? Finally, we mention that hardness condensation has also been studied in circuit complexity [BS06] (where the technique was first proposed) and query/communication complexity [GNRS24, Hru24].

1.4 Concurrent work by [dRFJ+24]

In concurrent work, de Rezende, Fleming, Janett, Nordström, and Pang [dRFJ+24] have independently obtained a supercritical tradeoff for monotone circuits. They exhibit an f so that:

(i") There is a monotone circuit computing f in size n^c for some constant $c > 1$.
(ii") Every monotone circuit computing f in depth $n^{1.9}$ requires size $n^{1.5c}$.

Comparing this to our result, the size blow-up in *(ii")* is at most polynomial, while our blow-up in *(ii)* is nearly exponential. On the other hand, their blow-up occurs already at depth that is polynomial in the size upper bound *(i")*. In this sense, the two results are incomparable.

Their approach, too, is to start with a new supercritical width–depth tradeoff for resolution (a condensed Tseitin formula, building on [GLNS23]) with parameters analogous to *(i")–(ii")* and then lift that to a monotone circuit tradeoff. Since they start with a small polynomial blow-up in proof complexity, they have to be careful that the lifting construction is not too lossy in its parameters. Consequently, they prove a new more optimised lifting theorem based

[1] This is because pebbling/sink-of-dag formulas are *complete* for resolution: Any formula F with a small resolution proof can be reduced, via decision trees, to a pebbling formula P. In other words, F is obtained from P by identifying variables and local gadget composition. See [GHJ+24b] for an exposition of this perspective.

on [GGKS20, LMM+22]. In comparison, we can afford a black-box application of these theorems. The paper [dRFJ+24] also gives applications to the Weisfeiler–Lehman algorithm, strengthening the breakthrough result of [GLNS23]. By contrast, we leave it as a direction for further research to find out if bracket formulas can yield such applications.

Some preliminary results of [dRFJ+24] were announced already at an Oberwolfach workshop in March 2024. Based on that announcement, Berkholz, Lichter, and Vinall-Smeeth [BLV24] have recently obtained new supercritical size–width tradeoffs for tree-like resolution.

2 Bracket Formulas

In this section, we formally define *bracket formulas* as a particular 3-CNF encoding of the bracket principle (1). For convenience, we start by first expressing bracket principles as a system of constraints over a large alphabet, and then describe how to encode them in binary.

2.1 Large-alphabet encoding

We consider strings $s \in \Sigma^n$ over the alphabet

$$\Sigma \coloneqq \{\,\square, \square, [, [,],]\,\} \times [n].$$

Here we have introduced *trivial brackets* $\square$ and $\square$ for technical convenience (it helps us avoid parity issues, e.g., requiring n to be even). They represent an open/close bracket pair, $[]$ or $[]$, compressed to a single symbol. For a symbol $\sigma \in \Sigma$ we write $\sigma = (b(\sigma), p(\sigma))$ where $b(\sigma)$ is the bracket type and $p(\sigma) \in [n]$ is the pointer. Given a string $s \in \Sigma^n$ we define the following set of constraints, called axioms.

(A1) *Start/end of string:* $b(s_1) \in \{\square, [\}$ and $b(s_n) \in \{\square,]\}$.

(A2) *Pointers define bracket pairs:*

- If $p(s_i) = j$, then $p(s_j) = i$. We say s_i and s_j are *paired* and write $s_i \sim s_j$.
- If $s_i \sim s_i$, then $b(s_i) \in \{\square, \square\}$.
- If $s_i \sim s_j$ where $i < j$, then $b(s_i) \in \{[, [\}$, $b(s_j) \in \{],]\}$, and their colours match.

(A3) *Correct nesting:* If $s_i \sim s_j$ and $s_{i'} \sim s_{j'}$, then we cannot have $i < i' < j < j'$.

(A4) *No red/blue transition:* For every $i \in [n-1]$, we have $b(s_i)b(s_{i+1}) \notin \{][,]\square, \square[, \square\square\}$.

The system (A1)–(A4) is an unsatisfiable set of constraints where each constraint depends on at most 4 indices of the string s. This means there are at most $O(n^4)$ constraints.

2.2 CNF encodings

Wide formula. Translating a system of $\mathrm{poly}(n)$ constraints of $O(1)$-arity over strings $s \in \Sigma^n$ to an $O(\log|\Sigma|)$-CNF formula F of size $\mathrm{poly}(n, |\Sigma|)$ is straightforward: Introduce $m \coloneqq \log|\Sigma|$ variables per index $i \in [n]$. Fix an arbitrary encoding function $\pi\colon \Sigma \to \{0,1\}^m$ that encodes each symbol s_i as an m-bit string. Then encode each $O(1)$-arity constraint separately. It is immaterial here which precise encoding π we use; any choice will work. Note that because every constraint is of constant arity it involves at most $O(\log|\Sigma|)$ variables and is thus translated to at most $2^{O(\log|\Sigma|)} = \mathrm{poly}\,|\Sigma|$ clauses. Hence the size of the resulting formula F is indeed $\mathrm{poly}(n, |\Sigma|)$.

Narrow formula. To be able to lift our tradeoff result from resolution to monotone circuits we need a constant-width CNF. We obtain a constant width encoding of F by introducing extension variables y_D for clauses D of constant *index-width*: the *index-width*[2] of a clause denotes the number of indices mentioned and the *index-width* of a formula is the maximum index-width of any clause in it. Note that F has constant index-width since the original constraints are of constant arity.

More precisely, if k denotes the index-width of F, then we introduce an extension variable y_D for every clause D of index-width $\leq k$ and encode F as a 3-CNF F' of $\mathrm{poly}(n, |\Sigma|)$ size over these variables: F' consists of

- *F-axioms*: if the clause D is in F, then F' contains the unary clause y_D, and
- *extension axioms*:
  - for every variable x of F the formula F' contains the clauses $y_x \vee y_{\neg x}$ and $\neg y_x \vee \neg y_{\neg x}$ ensuring that y_x is the negation of $y_{\neg x}$, and
  - for clauses A, B, D of index-width $\leq k$ satisfying $D = A \vee B$ the formula F' contains clauses enforcing the equivalence $y_D \leftrightarrow (y_A \vee y_B)$.

The index-width required to refute F is closely related to the width needed to refute F'. The proofs of the following propositions are standard and provided in Appendix A.

Proposition 5. *If F admits an index-width-w depth-d resolution refutation, then F' admits a width-$O(w)$ depth-$O(dw)$ resolution refutation.*

Proposition 6. *If F' admits a width-w depth-d resolution refutation, then F admits an index-width-$O(w)$ depth-$O(d \log |\Sigma|)$ resolution refutation.*

Since the constraints (A1)–(A4) are all of constant arity the above translations apply. We denote the resulting $O(\log n)$-CNF formula by WideBr_n and the resulting 3-CNF formula by Br_n. Note that by Propositions 5 and 6 it suffices to prove Theorems 3 and 4 for the more convenient formula WideBr_n.

3 Upper Bound

In this section, we prove the following resolution width upper bound for bracket formulas.

Theorem 3. *The bracket formula Br_n admits a resolution refutation of width $O(\log n)$.*

It suffices to describe a refutation of WideBr_n of small index-width because of Proposition 5. That is, we think of the formula WideBr_n as an unsatisfiable system of axioms over strings in Σ^n. To describe the refutation, we use the standard top-down language of *prover–adversary games* [Pud00, AD08].

3.1 Prover–adversary games

The prover–adversary game for WideBr_n is played between two competing players, prover and adversary, and proceeds in rounds. The state of the game in each round is modeled as a partial assignment $\rho \in (\Sigma \cup \{*\})^n$. At the start of the game, $\rho := *^n$. In each round:

[2]Similar notions of width have previously been studied under the name pigeon-width or block-width.

– *Query a symbol:* Prover specifies an index $i \in [n]$ and the adversary responds with a symbol $\sigma \in \Sigma$. The state ρ is updated by $\rho_i \leftarrow \sigma$.
– *Forget symbols:* Prover specifies a subset $I \subseteq [n]$ and we update $\rho_i \leftarrow *$ for all $i \in I$.

An important detail is that the adversary is allowed to choose $\sigma \in \Sigma$ afresh even if the i-th symbol was queried and subsequently forgotten during past play. The game ends when ρ falsifies some axiom of F. The prover's goal is to end the game while keeping the memory size of ρ (number of non-$*$ entries) at most w. The least memory size w so that prover has a winning strategy characterizes the index-width of WideBr_n [Pud00, AD08].

3.2 Prover strategy

During our prover strategy, we maintain the invariant that our state ρ always contains whole bracket pairs. That is, whenever we query a symbol, we follow its pointer and query its mate. Similarly, if we forget a symbol, we also forget its mate. We tacitly assume that the adversary responds each query without falsifying the first three axioms (A1)–(A3) (otherwise we win immediately). Our goal as the prover then becomes to find the substring `][`.

At the heart of our prover strategy is a recursive procedure that takes as input a state ρ that contains a red bracket pair R and a blue bracket pair B that lies to the right of R. The procedure outputs a state that has found either:

– a blue bracket pair enclosing R;
– a red bracket pair enclosing B; or
– the procedure outright discovers the substring `][` that falsifies (A4).

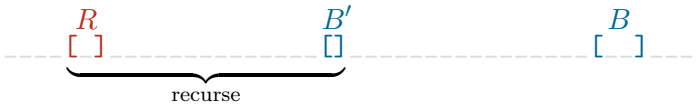
That is, in pictures:

In particular, if we invoke the procedure on the initial red/blue pair of brackets guaranteed by axiom (A1), the procedure must find the substring `][`. The procedure is:

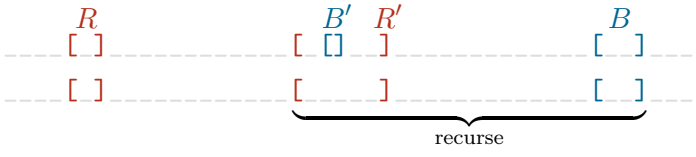
1. If the pairs R and B are touching (`[`$\cdots$`][`$\cdots$`]`), then we have found our substring `][`.

2. Otherwise we query the middle index in the interval between R and B. Say the adversary responds with a blue bracket pair B'. If B' encloses R, we are done.

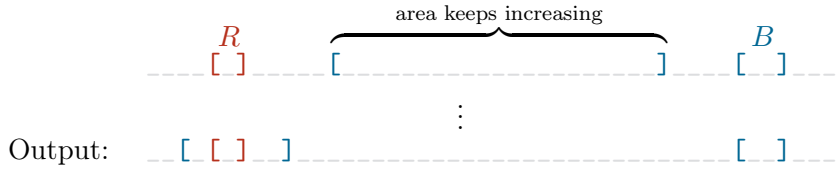
3. Otherwise we recurse on R and B'.



4. If this recursive call returns a blue bracket pair enclosing R, then we are done. Otherwise we have found a red middle pair R' whose *area* (number of indices enclosed by the pair) is larger than the area of B'. We now forget B' and recurse on R' and B.



5. By continuing this way, we find a sequence of middle pairs whose area keeps increasing every step. Eventually, a recursive call must return a bracket enclosing R or B that is of the opposite colour. (It is possible that the middle pair, say R', encloses the same-coloured input pair R at some point. This only means that the next recursive call on R' and B will return a desired oppositely-coloured output pair.)



The recursive depth of this strategy is $O(\log n)$ and we keep at most 4 bracket pairs in memory in each level of recursion. It follows that the total memory usage is $O(\log n)$ symbols. This shows that WideBr_n has a refutation of index-width $O(\log n)$, which concludes the proof of Theorem 3.

4 Lower Bound

In this section we prove the following depth lower bound for the bracket formulas.

Theorem 4. *Every width-w resolution refutation of Br_n has depth $n^{\Omega\left(\frac{\log n}{\log w}\right)}$.*

Note that by Proposition 6 it suffices to prove that any index-width-w resolution refutation of WideBr_n has depth $n^{\Omega(\log n/\log w)}$. We prove this by exhibiting an adversary strategy for the prover–adversary game such that if the game state $\rho \in (\Sigma \cup \{*\})^n$ is limited to memory size w, then the prover cannot win the prover–adversary game in less than $n^{\Omega(\log n/\log w)}$ rounds. Since the minimum number of rounds before the prover can win corresponds to the minimum depth of a resolution refutation of width w [Pud00, AD08] we may conclude Theorem 4.

The high-level objective of the adversary is to maintain the property that differently coloured top-level brackets of ρ are far apart. This is achieved by a strategy that forces the prover to essentially follow the upper bound strategy of Theorem 3. Throughout the game the adversary maintains a "cleaned-up" view of the game state ρ. We formalize this notion as a *container* of ρ in Section 4.1. This section is followed by Section 4.2 devoted to the adversary strategy. Finally in Sections 4.3 and 4.4 we prove that our adversary strategy indeed guarantees that the prover cannot win the game in the first $n^{\Omega(\log n/\log w)}$ rounds.

4.1 Containers

The game state $\rho \in (\Sigma \cup \{*\})^n$ is a somewhat unruly partial assignment. It may contain brackets that are not paired or bracket pairs that are redundant: if ρ assigns consecutive indices to, say, [], then the blue trivial bracket pair [] is of no use to the prover. In order to concisely state the adversary strategy we want a "cleaned up" view of ρ.

To this end we introduce the notion of a *container* of ρ. Informally a container of ρ is a partial assignment that (1) consists of bracket pairs and (2) is minimal while guaranteeing that ρ does not falsify an axiom of the bracket formula. Thus, since a container ensures that ρ does not falsify an axiom, the task of the adversary is reduced to maintaining a container of ρ for the initial $n^{\Omega(\log n/\log w)}$ rounds. In Section 4.2 we exhibit such a strategy. In the remainder of this section we formalize the notion of a container.

We say that a partial assignment is consistent if it falsifies no axiom of the system (A1)–(A4). Recall that for a symbol $\sigma \in \Sigma$ we write $\sigma = (b(\sigma), p(\sigma))$ where $b(\sigma) \in \{$ [] , [] , [, [,] ,] $\}$ is the bracket type and $p(\sigma) \in [n]$ is the pointer.

Definition 7 (Configuration)**.** A *configuration* $\mathcal{C} \in (\Sigma \cup \{*\})^n$ is a consistent partial assignment consisting of bracket pairs, that is, if $\mathcal{C}$ assigns index i and $p(\mathcal{C}_i) = j$, then $\mathcal{C}$ assigns index j.

For example the partial assignment `[[]  _ []` is *not* a configuration: it is consistent (assuming the pointers are correctly chosen) but the bracket `[` is unpaired.

Note that a configuration $\mathcal{C}$ can be thought of as a set of bracket pairs. In the following we thus interchangeably identify $\mathcal{C}$ as a set of bracket pairs $\mathcal{C} = \{P_i\}_i$ and as a consistent partial assignment $\mathcal{C} \in (\Sigma \cup \{*\})^n$.

The *area* of a bracket pair P, denoted by $\mathrm{area}(P)$, is the set of indices enclosed by P and we say that an index $i \in [n]$ is *covered* by P if $i \in \mathrm{area}(P)$. These notions are extended in the natural way to a configuration $\mathcal{C}$: the *area of* $\mathcal{C}$ consists of all indices covered by its bracket pairs $\mathrm{area}(\mathcal{C}) = \bigcup_{P \in \mathcal{C}} \mathrm{area}(P)$ and an index i is *covered* by $\mathcal{C}$ if $i \in \mathrm{area}(\mathcal{C})$.

A configuration $\mathcal{C}$ is said to be *closed* if it assigns all covered indices and it is *locally consistent* if there is a closed configuration $\mathcal{C}'$ that extends $\mathcal{C} \subseteq \mathcal{C}'$. For example, the configuration

is locally consistent as witnessed by the closed configuration

. (2)

By contrast, the following configuration is *not* locally consistent:

.

(There is no consistent assignment for the two indices to the right of the left-most `]`.)

The configuration $\mathcal{C}$ maintained by the adversary is a locally consistent configuration that contains *top-level bracket pairs* only: a bracket pair $P \in \mathcal{C}$ is top-level if for all other pairs $P' \in \mathcal{C}$ it holds that $\mathrm{area}(P) \not\subseteq \mathrm{area}(P')$. For example, the configuration

contains the top-level bracket pairs

.

A configuration $\mathcal{C}$ is *monotone* if the red top-level bracket pairs of $\mathcal{C}$ are before some index i while the blue top-level bracket pairs come after i. The *separating interval* of such a monotone configuration $\mathcal{C}$ is the (unique) maximal interval $I \subseteq [n]$ that contains $i \in I$ and satisfies that no index in I is covered by $\mathcal{C}$. The above example configuration is monotone and has a separation interval of size 3.

The locally consistent configuration $\mathcal{C}$ maintained by the adversary is furthermore monotone and related to the game state $\rho \in (\Sigma \cup \{*\})^n$ as follows. Let the *support* of a partial assignment τ, denoted by $\mathrm{supp}(\tau)$, be the set of indices assigned by τ, that is, $\mathrm{supp}(\tau) = \{i \in [n] : \tau_i \neq *\}$.

Definition 8 (Domination)**.** A configuration $\mathcal{C}$ *dominates* a partial assignment τ if the support of τ is covered by $\mathcal{C}$ (i.e., $\mathrm{supp}(\tau) \subseteq \mathrm{area}(\mathcal{C})$) and there is a closed configuration $\mathcal{C}'$ that extends $\mathcal{C} \subseteq \mathcal{C}'$ as well as $\tau \subseteq \mathcal{C}'$.

For example the partial assignment

is dominated, assuming the pointers are appropriately chosen, by

as witnessed by (2).

To summarize the above discussion the adversary maintains a monotone and locally consistent configuration $\mathcal{C}$ that dominates the game state ρ. Furthermore, the configuration $\mathcal{C}$ consists of top-level bracket pairs only. We impose in fact a slightly stronger condition than only containing top-level pairs as summarized in the next definition.

Definition 9 (Container)**.** A *container* of a partial assignment τ is a locally consistent and monotone configuration $\mathcal{C}$ such that

(C1) $\mathcal{C}$ dominates τ, and
(C2) $\mathcal{C}$ is minimal while dominating τ: removing any bracket pair from $\mathcal{C}$ results in a configuration that does not dominate τ.

Observe that (C2) implies that containers consist of top-level bracket pairs only. It further allows us to relate the size of a container to the size of the support of τ as follows.

Proposition 10. *Consider a partial assignment τ and a configuration $\mathcal{C}$. If $\mathcal{C}$ is a container of τ, then τ is consistent and, furthermore, $\mathcal{C}$ contains at most $|\mathrm{supp}(\tau)|$ many bracket pairs.*

Proof. By (C1) we have that $\mathcal{C}$ dominates τ and hence by Definition 8 there is a configuration $\mathcal{C}'$ that extends $\tau \subseteq \mathcal{C}'$. Since configurations are consistent this implies that τ is consistent. The claim about the number of bracket pairs in $\mathcal{C}$ follows from the minimality property (C2). □

By Proposition 10 it holds that while the adversary maintains a container of the game state ρ the prover cannot win the prover–adversary game. The following section exhibits an adversary that maintains a container of ρ for the initial $n^{\Omega(\log n/\log w)}$ rounds.

4.2 The Adversary Strategy

This section is devoted to the adversary strategy. Throughout we assume that the memory size of the game state ρ is bounded by w.

The main objective of the adversary is to maintain a container $\mathcal{C}$ of ρ for the initial $n^{\Omega(\log n/\log w)}$ rounds of the prover–adversary game. By Proposition 10 it thus follows that the prover cannot win the game in the first $n^{\Omega(\log n/\log w)}$ rounds. Theorem 4 follows from the fact that the minimum number of rounds required for the prover to win the game corresponds to the minimum depth of a width-w resolution refutation.

In addition to the container $\mathcal{C}$ of ρ the adversary also maintains an interval $I \subseteq [n]$. Initially $I = [n]$ and we should think of I as the separation interval of $\mathcal{C}$. Let us stress, though, that for technical convenience the actual strategy only guarantees that I is a sub-interval of the separation interval.

Let us describe the adversary strategy. There are two integer parameters $\ell_0 = \lfloor \varepsilon \frac{\log n}{\log w} \rfloor$ and $d = \lfloor n^{\varepsilon} \rfloor$. The adversary repeats the following d times.

1. Identify a large interval I with no index covered by the container $\mathcal{C}$.
2. Move the separation interval to I by modifying $\mathcal{C}$.
3. Recursively invoke the adversary on the interval I.

Once the adversary has reached recursion level ℓ_0 they are ready to play a round of the prover–adversary game. Suppose that the prover queries some index i. If index i is covered by the container $\mathcal{C}$, then the adversary answers according to the complete configuration $\mathcal{C}' \supseteq \mathcal{C} \cup \rho$ as guaranteed to exist by (C1). Otherwise, if i is not covered by $\mathcal{C}$, for $[a, b] = I$ the adversary answers with a trivial bracket coloured red if $i < \frac{a+b}{2}$ and coloured blue otherwise.

This essentially completes the description of the adversary strategy modulo the "move" operation performed in Step 2. This is best explained by a picture. Say the current container is

$$\mathcal{C} = \quad \underbrace{}_{I} \quad ,$$

where I is the interval as identified in Step 1. The move operation replaces any red (blue) bracket pair to the right (to the left) of I by a blue (red) bracket pair that minimally covers it. If any of these newly added pairs overlap, then they are merged into a single bracket pair.

For the above example container $\mathcal{C}$ the move operation results in the new container

$$\mathcal{C}' = \quad .$$

Observe that if $\mathcal{C}$ is a container of ρ, then by construction $\mathcal{C}'$ is also a container of ρ. Further the area of $\mathcal{C}'$ is quite closely related to the area of $\mathcal{C}$: according Proposition 10 any container of ρ contains at most w bracket pairs and hence $\mathcal{C}'$ may only cover an additional $2w$ indices. Finally note that the MOVE operation is the only source of non-trivial bracket pairs in the container.

This completes the description of the adversary strategy. For a more thorough treatment we refer to Algorithm 1. The strategy is initially invoked by ADVERSARY$(\emptyset, [n], 0)$.

Let us remark that the game–state ρ is treated rather implicitly; we do not keep explicit track of ρ. It is understood, however, that we are playing a prover–adversary game and in each round played ρ changes. This implies in particular that the game state when an adversary returns is (most likely) distinct from the game state when the adversary was instantiated.

Finally observe that the adversary changes the maintained $\mathcal{C}$ in two places only: in PLAYROUND when playing a round of the game and in MOVE when moving the separation interval. Hence $\mathcal{C}$ may fail to be a container of ρ in only these two places. The following claim shows that if the adversary does not fail, then $\mathcal{C}$ is indeed a container of ρ.

Lemma 11. *Unless the adversary fails it holds that the maintained configuration $\mathcal{C}$ is a container of the game state.*

Proof. Consider an adversary invoked by ADVERSARY$(\mathcal{C}, I, \ell)$. By construction it should be evident that the interval I is a sub-interval of the separation interval of $\mathcal{C}$. Since the adversary does not fail we may assume that $|I| \geq n/(4w)^{\ell_0} \geq 10$ by our choice of $\ell_0 = \lfloor \varepsilon \frac{\log n}{\log w} \rfloor$.

By induction on the recursion-depth $\ell = \ell_0, \ldots, 0$ we prove that if $\mathcal{C}' = \text{ADVERSARY}(\mathcal{C}, I, \ell)$ is instantiated with a container $\mathcal{C}$ of the current game state, then the adversary returns a container $\mathcal{C}'$ of the resulting game state with a separation interval of size at least 5. Note that once the induction is established the statement follows for the initial call ADVERSARY$(\emptyset, [n], 0)$ since the empty configuration is a container for the initial game state $\rho = *^n$.

The base case, that is for an adversary of recursion-depth ℓ_0, the inductive hypothesis holds by inspection of the procedure PLAYROUND and the assumption that the interval I is of size at least 10. Note that the separation interval of the container returned is of size at least 5.

We may thus assume the inductive hypothesis for depth $\ell + 1$. Consider an adversary of depth ℓ. We need to argue that MOVE does not cause the container to become inconsistent. This is readily verified: note that the bracket pairs $\mathcal{P}$ being replaced in MOVE$(\tilde{\mathcal{C}}_i, \tilde{I}_i)$ lie in-between the intervals $\tilde{I}_i$ and the separation interval of $\tilde{\mathcal{C}}_i$. By induction we may assume that

Algorithm 1 The adversary strategy.

procedure ADVERSARY($\mathcal{C}, I, \ell$)
  if $|I| < n/(4w)^\ell$ **then**
    return failure
  if $\ell = \ell_0$ **then**
    $\tilde{\mathcal{C}}_d \leftarrow$ PLAYROUND($\mathcal{C}, I$)
  else
    $\tilde{\mathcal{C}}_0 \leftarrow \mathcal{C}$
    for $i = 0, 1, \ldots, d-1$ **do**
      $\tilde{I}_i \leftarrow$ a largest interval in I with no index covered by $\tilde{\mathcal{C}}_i$
      $\mathcal{C}_i \leftarrow$MOVE($\tilde{\mathcal{C}}_i, \tilde{I}_i$)
      $I_i \leftarrow$ maximum interval in $\tilde{I}_i$ with no index covered by $\mathcal{C}_i$
      $\tilde{\mathcal{C}}_{i+1} \leftarrow$ADVERSARY($\mathcal{C}_i, I_i, \ell + 1$)
  return $\tilde{\mathcal{C}}_d$

Algorithm 2 Plays a round of the prover–adversary game.

procedure PLAYROUND($\mathcal{C}, I$)
  if prover forgets **then**
    $\mathcal{C} \leftarrow$ minimum sub-configuration of $\mathcal{C}$ that dominates ρ
  else
    $i \leftarrow$ index queried by the prover
    if $i \in \text{area}(\mathcal{C})$ **then**
      $\mathcal{C}' \leftarrow$ closed configuration $\mathcal{C}' \supseteq \mathcal{C} \cup \rho$ as guaranteed by Definition 8
      $\rho_i \leftarrow \mathcal{C}'_i$
    else
      $[a, b] \leftarrow I$
      if $i < \frac{a+b}{2}$ **then**
        $\rho_i \leftarrow$ ▯; $\mathcal{C}_i \leftarrow$ ▯
      else
        $\rho_i \leftarrow$ ▯; $\mathcal{C}_i \leftarrow$ ▯
  return $\mathcal{C}$

Algorithm 3 Returns a configuration $\mathcal{C}'$ such that the separation interval intersects I.

procedure MOVE($\mathcal{C}, I$)
  $\mathcal{P} \leftarrow$ red (blue) bracket pairs to the right (left) of I
  $\mathcal{P}' \leftarrow \emptyset$
  for $P \in \mathcal{P}$ **do**
    $P' \leftarrow$ minimum bracket pair that strictly contains P of opposite colour
    $\mathcal{P}' \leftarrow \mathcal{P}' \cup \{P'\}$
  while $\exists P, P' \in \mathcal{P}'$ such that $\text{area}(P) \cap \text{area}(P') \neq \emptyset$ **do**
    $P'' \leftarrow$ bracket pair such that $\text{area}(P'') = \text{area}(P) \cup \text{area}(P')$ of equal colour
    $\mathcal{P}' \leftarrow \mathcal{P}'$ with P, P' replaced by P''
  return $(\mathcal{C} \setminus \mathcal{P}) \cup \mathcal{P}'$

the separation interval of $\tilde{\mathcal{C}}_i$ is of size at least 5 and by assumption it holds that $|\tilde{I}_i| \geq 10$ as otherwise the adversary would fail in the coming recursion. This implies in particular that the bracket pairs $P' \in \mathcal{P}'$ do not intersect the area of the configuration $\mathcal{C} \setminus \mathcal{P}$. This establishes the induction. The statement follows. □

4.3 Buffers

It remains to establish that every adversary $\textsc{Adversary}(\mathcal{C}, I, \ell)$ is invoked with an interval I of size $|I| \geq n/(4w)^\ell$. Let $A_\ell = w(A_{\ell+1} + 3d)$ with $A_{\ell_0} = 3d$ for $\ell \in \{0, 1, \ldots, \ell_0\}$.

Lemma 12 (Cover Lemma)**.** *Consider the adversary $\textsc{Adversary}(\mathcal{C}, I, \ell)$ of recursion depth ℓ, suppose that $|I| \geq n/(4w)^\ell$, and let $\tilde{\mathcal{C}}_{i+1} = \textsc{Adversary}(\mathcal{C}_i, I_i, \ell + 1)$ as in Algorithm 1. It holds that $|\text{area}(\tilde{\mathcal{C}}_i) \cap I| \leq w(A_{\ell+1} + 3i)$.*

Let us first argue that Lemma 12 is indeed sufficient, that is, if Lemma 12 holds, then the intervals with which the adversaries are invoked are large.

Lemma 13 (Size Lemma)**.** *Consider the adversary $\textsc{Adversary}(\mathcal{C}, I, \ell)$ of recursion depth ℓ, suppose that $|I| \geq n/(4w)^\ell$, and let $\tilde{\mathcal{C}}_{i+1} = \textsc{Adversary}(\mathcal{C}_i, I_i, \ell + 1)$ as in Algorithm 1. If it holds that $|\text{area}(\tilde{\mathcal{C}}_i) \cap I| \leq A_\ell$, then $|I_i| \geq n/(4w)^{\ell+1}$.*

Proof. We need to argue that there is a large interval $\tilde{I}_i \subseteq I$ such that $\tilde{\mathcal{C}}_i$ covers no index in $\tilde{I}_i$. By minimality of the configuration $\mathcal{C}$ (formally this can be established by a straightforward induction) it holds that $\mathcal{C}$ contains at most w top-level bracket pairs. Hence there is such an interval $\tilde{I}_i$ of size

$$|\tilde{I}_i| \geq \frac{|I \setminus \text{area}(\tilde{\mathcal{C}}_i)|}{w+1} \ . \tag{3}$$

By assumption and our choice of parameters it holds that $|I| \geq n/(4w)^\ell \geq \Omega(n^{1-2\varepsilon})$ while

$$|I \cap \text{area}(\tilde{\mathcal{C}}_i)| \leq A_\ell \leq O(d\ell_0 w^{\ell_0}) = O(n^{3\varepsilon}) \ll |I|/2 \ . \tag{4}$$

Substituting the above estimates into Equation (3) reveals that

$$|\tilde{I}_i| \geq \frac{n}{(4w)^\ell 2(w+1)} \ . \tag{5}$$

This implies that any invocation of $\textsc{Adversary}(\mathcal{C}_i, I_i, \ell + 1)$ at recursion-depth $\ell + 1$ ends up with an interval I_i of size at least $|I_i| \geq n/((4w)^\ell 2(w+1)) - 1 > n/(4w)^{\ell+1}$. Note the minus one due to the replacement of bracket pairs in Move. This establishes the statement. □

With the Size Lemma at hand it remains to prove the Cover Lemma in order to conclude Theorem 4.

We prove the Cover Lemma by induction on the recursion–depth $\ell = \ell_0, \ldots, 0$ and $i = 0, 1, \ldots, d$. Let us give a naïve proof sketch to then explain how to improve the bound. The formal proof of the Cover Lemma is provided in Section 4.4.

Suppose by induction that $\tilde{\mathcal{C}}_{i+1} = \textsc{Adversary}(\mathcal{C}_i, I_i, \ell+1)$ covers at most $A_{\ell+1}$ indices in I_i. If we apply the inductive hypothesis naïvely we obtain that $\tilde{\mathcal{C}}_{i+1}$ covers at most $(i+1)(A_{\ell+1}+3w)$ indices in I. This is clearly insufficient since $i + 1 \in [d]$ and $d = \lfloor n^\varepsilon \rfloor \gg w$.

The above argument has not used the fact that the game state ρ is limited to memory size w. We intend to argue that the game state may contain at most w bracket pairs originating from distinct recursive calls. This allows us to then conclude that $\tilde{\mathcal{C}}_d$ covers at most $w(A_{\ell+1}+3d) = A_\ell$ indices in I as claimed.

To this end we show that non-trivial bracket pairs obtained from different recursive calls are far apart. In a bit more detail, for $i \neq j$, we intend to argue that two non-trivial bracket pairs $P_i \in \tilde{\mathcal{C}}_i \setminus \mathcal{C}_{i-1}$ and $P_j \in \tilde{\mathcal{C}}_j \setminus \mathcal{C}_{j-1}$ are at distance at least $4dw$. Since the MOVE operation increases the cover by at most $2w$ the large gap between P_i and P_j cannot be bridged. Hence if in $\tilde{\mathcal{C}}_d$ both bracket pairs are remembered, then these are distinct bracket pairs.

Making the above intuition into a formal proof requires some care. To argue that non-trivial bracket pairs from distinct recursive calls are far apart we introduce the notion of a *buffer*.

Definition 14 (Buffer)**.** For integer $s \in \mathbb{N}^+$ and an interval $I = [a, b] \subseteq [n]$ we say that a configuration $\mathcal{C}$ has an *s-buffer in I* if every bracket pair $P \in \mathcal{C}$ satisfies that if P is contained in the interval $[1, a + s]$, then it is a trivial red pair $P = \square$ and if P is contained in the interval $[b - s, n]$, then it is a trivial blue pair $P = \square$.

The following lemma establishes that bracket pairs originating from distinct recursive calls are far apart. In terms of buffers we establish that bracket pairs in $\tilde{\mathcal{C}}_{i+1} \setminus \mathcal{C}_i$ have a buffer in I_i. The lemma is stated in a way so that it can be readily applied in the inductive proof of the Cover Lemma.

Lemma 15 (Buffer Lemma)**.** *Consider the adversary* ADVERSARY$(\mathcal{C}, I, \ell)$ *of recursion depth ℓ, fix $c_\ell = 4dw - (\ell_0 - \ell)$, and let $\tilde{\mathcal{C}}_{i+1} =$* ADVERSARY$(\mathcal{C}_i, I_i, \ell + 1)$ *as in Algorithm 1. If for some $i \in [d-1]$ it holds that the configuration $\tilde{\mathcal{C}}_i \setminus \mathcal{C}$ has a c_ℓ-buffer in I, the configuration $\tilde{\mathcal{C}}_i \setminus \mathcal{C}_{i-1}$ has a $c_{\ell+1}$-buffer in I_{i-1}, and the configuration $\tilde{\mathcal{C}}_{i+1} \setminus \mathcal{C}_i$ has a $c_{\ell+1}$-buffer in I_i, then $\tilde{\mathcal{C}}_{i+1} \setminus \mathcal{C}$ has a c_ℓ-buffer in I.*

Proof. Let $[a, b] = I$ and consider a bracket pair $P \in \tilde{\mathcal{C}}_{i+1} \setminus \mathcal{C}$. If $P \in \tilde{\mathcal{C}}_{i+1} \setminus \mathcal{C}_i$ a $c_{\ell+1}$-buffer in I_i, then there is nothing to show since $I_i \subseteq I$ and hence $\tilde{\mathcal{C}}_{i+1} \setminus \mathcal{C}_i$ is also a c_ℓ-buffer in I.

Otherwise $P \in \mathcal{C}_i \setminus \mathcal{C}$. If $P \in \tilde{\mathcal{C}}_i \setminus \mathcal{C}$, then there is nothing to show since $\tilde{\mathcal{C}}_i \setminus \mathcal{C}$ is a c_ℓ-buffer in I. We are left to show the statement for bracket pairs $P \in \mathcal{C}_i \setminus \tilde{\mathcal{C}}_i$. Note that these bracket pairs P were added by MOVE and are thus non-trivial. We need to establish that all such pairs P are contained in the interval $[a + c_\ell + 1, b - (c_\ell + 1)]$.

Observe that all bracket pairs in $\mathcal{C}_i \setminus \tilde{\mathcal{C}}_i$ are of equal colour since the maintained configurations are monotone. Without loss of generality we may assume that the replaced bracket pairs $\tilde{\mathcal{C}}_i \setminus \mathcal{C}_i$ are blue and that the newly added pairs $\mathcal{C}_i \setminus \tilde{\mathcal{C}}_i$ are red.

Denote by a_{i-1} the left endpoint of the interval I_{i-1}. Since the configuration $\tilde{\mathcal{C}}_i \setminus \mathcal{C}_{i-1}$ has a $c_{\ell+1}$-buffer in I_{i-1} it holds that the $c_{\ell+1}$ left-most indices of I_{i-1} do *not* contain a blue bracket pair. Further using the fact that $\mathcal{C}_{i-1}$ covers no positions in I_{i-1} we obtain that all bracket pairs $\tilde{\mathcal{C}}_i \setminus \mathcal{C}_i$ being replaced are contained in the interval $[a_{i-1} + c_{\ell+1} + 1, b]$ and the new bracket pairs are thus contained in the interval $[a_{i-1} + c_\ell + 1, b]$. Furthermore, since I_i is large $|I_i| \gg c_{\ell+1}$ for our choice of parameters it holds that the newly added bracket pairs $\mathcal{C}_i \setminus \tilde{\mathcal{C}}_i$ cannot cover the c_ℓ right-most indices of I. Hence all these bracket pairs are contained in the interval $[a + c_\ell + 1, b - (c_\ell + 1)]$ as required. This establishes the statement. □

As previously explained the notion of a buffer lets us argue that bracket pairs from different recursive calls are far apart. The next definition formalizes what it means for a bracket pair to originate from a recursive call.

Definition 16 (Legacy, ancestors)**.** Consider ADVERSARY$(\mathcal{C}, I, \ell)$ of recursion depth ℓ and let $\tilde{\mathcal{C}}_{i+1} =$ ADVERSARY$(\mathcal{C}_i, I_i, \ell+1)$ as in Algorithm 1. Consider a non-trivial bracket pair $P \in \tilde{\mathcal{C}}_i$. If P is not present in $\mathcal{C}_{i-1}$, then P is a *legacy-i bracket pair*. If $\tilde{\mathcal{C}}_{i-1}|_P$ denotes the set of non-trivial bracket pairs $P' \in \tilde{\mathcal{C}}_{i-1}$ satisfying $\text{area}(P') \subseteq \text{area}(P)$, then the *legacy ancestors* of a

bracket pair P are

$$\text{ancestors}(P) = \begin{cases} \{P\} & \text{if } P \text{ is a legacy-}i \text{ bracket pair, and} \\ \bigcup_{P' \in \tilde{\mathcal{C}}_{i-1}|_P} \text{ancestors}(P') & \text{otherwise.} \end{cases}$$

The area of a bracket pair is contained in the area of its legacy ancestors except for a few indices as stated next.

Lemma 17. *Consider the adversary* $\textsc{Adversary}(\mathcal{C}, I, \ell)$ *of recursion depth* ℓ *and let* $\tilde{\mathcal{C}}_{i+1} = \textsc{Adversary}(\mathcal{C}_i, I_i, \ell+1)$ *as in Algorithm 1. For all* $i \in \{0, 1, \ldots, d\}$ *there is a subset* $U \subseteq [n]$ *of size* $|U| \leq 3iw$ *such that for any bracket pair* $P \in \tilde{\mathcal{C}}_i$ *contained in the interval* $\text{area}(P) \subseteq I$ *it holds that*

$$\text{area}(P) \subseteq U \cup \bigcup_{P' \in \text{ancestors}(P)} \text{area}(P') \ .$$

Proof. By induction on $i = 0, 1, \ldots, d$. For $i = 0$ there is nothing to show since $\tilde{\mathcal{C}}_0 = \mathcal{C}$ and $\mathcal{C}$ covers no index in I by construction.

We may thus assume the statement for some i to prove the statement for $i+1$. Recall that

$$|\text{area}(\mathcal{C}_i) \setminus \text{area}(\tilde{\mathcal{C}}_i)| \leq 2w \tag{6}$$

since the Move operation increases the area of the resulting configuration by at most $2w$. Add all the indices in the difference to U_{i+1} along with all trivial bracket pairs in $\tilde{\mathcal{C}}_i$. Observe that $|U_{i+1}| \leq 3w$ and that it holds that

$$\text{area}(P) \subseteq U_{i+1} \cup \bigcup_{P' \in \tilde{\mathcal{C}}_i|_P} \text{area}(P') \ . \tag{7}$$

The statement follows by appealing to the inductive hypothesis for each $P' \in \tilde{\mathcal{C}}_i|_P$ separately. □

Finally we are in a position to formally argue that a bracket pair originates from a single legacy, that is, from a single recursive call.

Lemma 18. *Consider the adversary* $\textsc{Adversary}(\mathcal{C}, I, \ell)$ *of recursion depth* ℓ *and let* $\tilde{\mathcal{C}}_{i+1} = \textsc{Adversary}(\mathcal{C}_i, I_i, \ell+1)$ *as in Algorithm 1. If for all* $j \leq i$ *it holds that* $\tilde{\mathcal{C}}_j \setminus \mathcal{C}_{j-1}$ *has a* $c_{\ell+1}$*-buffer in* I_{j-1}*, then for every bracket pair* $P \in \tilde{\mathcal{C}}_i$ *there is a* $j \leq i$ *such that all legacy-ancestors of* P *are legacy-*j *bracket pairs.*

Proof. By contradiction. Suppose there are two bracket pairs $P_1, P_2 \in \text{ancestors}(P)$ from distinct legacies $j_1 < j_2$. Without loss of generality we may assume that there is no bracket pair $P^\star \in \text{ancestors}(P)$ that lies in-between P_1 and P_2.

By definition of an ancestor there is a non-trivial bracket pair $P_1' \in \tilde{\mathcal{C}}_{j_2}$ such that $\text{area}(P_1') \supseteq \text{area}(P_1)$. By assumption $\tilde{\mathcal{C}}_{j_2} \setminus \mathcal{C}_{j_2-1}$ has a $c_{\ell+1}$-buffer in I_{j-1}. Hence the distance between the bracket pairs P_1' and P_2 is strictly larger than $3dw$.

Since there are no other legacy-ancestors in-between P_1' and P_2 by Lemma 17 all these indices between P_1' and P_2 need to be covered by the set U of size $|U| \leq 3dw$. This cannot be; the statement follows. □

4.4 Proof of the Cover Lemma

As previously explained we prove the Cover Lemma by induction on the recursion–depth $\ell = \ell_0, \dots, 0$ and on $i = 0, 1, \dots, d$. It is most convenient to think of the recursion as an outer recursion on ℓ and an inner recursion on i.

The outer induction on the recursion–depth $\ell = \ell_0, \dots, 0$ establishes the following statement. For $\mathcal{C}' = \textsc{Adversary}(\mathcal{C}, I, \ell)$ of recursion depth ℓ it holds that if $|I| \geq n/(4w)^\ell$, then

1. $\mathcal{C}' \setminus \mathcal{C}$ has a c_ℓ-buffer in I and
2. $\mathcal{C}'$ covers at most A_ℓ indices in I.

For the base case we need to consider $\mathcal{C}' = \textsc{Adversary}(\mathcal{C}, I, \ell_0)$. Note that by construction $\mathcal{C}$ does not cover any index in I. Hence if the resulting $\mathcal{C}'$ covers any index in I, then this is due to a trivial bracket pair P either coloured red or blue. Since for our choice of parameters $|I| \geq n/(4w)^\ell \gg 8dw = 2c_{\ell_0}$ the bracket pair P is appropriately coloured. Hence $\mathcal{C}' \setminus \mathcal{C}$ has a c_{ℓ_0}-buffer in I. Since $\mathcal{C}'$ covers at most $1 \leq 3d = A_{\ell_0}$ indices in I the base case is established.

We may thus assume the above statement for adversaries of recursion–depth $\ell+1$. It remains to show the statement for $\textsc{Adversary}(\mathcal{C}, I, \ell)$ of recursion–depth ℓ. By an inner induction on $i = 0, 1, \dots, d$ we intend to establish that if $|I| \geq n/(4w)^\ell$, then

1. $\tilde{\mathcal{C}}_i \setminus \mathcal{C}$ has a c_ℓ-buffer in I and
2. $\tilde{\mathcal{C}}_i$ covers at most $w\big(A_{\ell+1} + 3i\big)$ indices in I.

Observe that in order to establish Cover Lemma it suffices to complete the above (inner and outer) induction.

There is nothing to show for the (inner) base case $i = 0$ since $\tilde{\mathcal{C}}_0 = \mathcal{C}$ and $\mathcal{C}$ covers no index in I by construction.

We may thus assume the statement for i to show it for $i + 1$. First we appeal to the Size Lemma to conclude that $|I_i| \geq n/(4w)^{\ell+1}$. This allows us to appeal to the outer induction hypothesis to conclude that $\tilde{\mathcal{C}}_{i+1} \setminus \mathcal{C}_i$ has a $c_{\ell+1}$ buffer in I_i.

To establish Property 1 we may appeal to the Buffer Lemma with the inner and outer induction hypothesis.

It remains to establish Property 2. According to Proposition 10 there are at most w bracket pairs in the configuration $\tilde{\mathcal{C}}_{i+1}$. Consider one such bracket pair $P \in \tilde{\mathcal{C}}_{i+1}$ contained in $\mathrm{area}(P) \subseteq I$. By Lemma 18 we may assume that there is a $j \leq i + 1$ such that all ancestors are legacy-j bracket pairs. By induction we know that $|I_j| \geq n/(4w)^{\ell+1}$ and we may thus appeal to the outer inductive hypothesis to conclude that the legacy ancestors of P cover at most $A_{\ell+1}$ indices. Combining this estimate with Lemma 17 we obtain that $|\mathrm{area}(P)| \leq A_{\ell+1} + 3(i+1)w$. This implies that $|\mathrm{area}(\tilde{\mathcal{C}}_{i+1}) \cap I| \leq w\big(A_{\ell+1} + 3(i+1)\big)$ since the set U (as in Lemma 17) is independent of the choice of the bracket pair. This establishes Property 2 of the inner induction for $i + 1$.

This establishes the induction and thereby the Cover Lemma. Theorem 4 follows from combining Lemmas 11 to 13 and the fact that $\textsc{Adversary}(\emptyset, [n], 0)$ plays $d^{\ell_0} = n^{\Omega(\frac{\log n}{\log w})}$ rounds of the prover–adversary game before terminating.

5 Proofs of Theorems 1 and 2

Before proving Theorems 1 and 2 we need to recall what it means to compose a CNF formula F with indexing gadget $\textsc{Ind}_t \colon [t] \times \{0,1\}^t \to \{0,1\}$ that maps an input (x, y) to y_x. If $z_1, z_2, \dots, z_n$ denote the variables of F, then $F \circ \textsc{Ind}_t^n$ denotes the CNF formula obtained from F by substituting each variable $z_i \mapsto \textsc{Ind}_t(x_i, y_i)$, where x_i, y_i are new sets of variables. Note that if F is

an unsatisfiable k-CNF formula with m clauses, then $F \circ \textsc{Ind}_t^n$ is an unsatisfiable CNF formula over $O(nt)$ variables and $O(tm^k + n)$ clauses.

Let us start with the proof of Theorem 2. The theorem follows from Theorems 3 and 4 along with an application of dag-like lifting [GGKS20]. The following theorem is taken from [FPR22].

Theorem 19 ([GGKS20, LMM+22, FPR22])**.** *Let $\varepsilon > 0$ be any constant and let F be an unsatisfiable CNF formula on n variables. For any semantic CP proof Π of $F \circ \textsc{Ind}_{n^{1+\varepsilon}}^n$ there is a resolution refutation Π^* of F satisfying:*

- $\text{width}(\Pi^*) = O\left(\frac{\log |\Pi|}{\log n}\right)$;
- $\text{depth}(\Pi^*) = O\left(\text{depth}(\Pi)\frac{\log |\Pi|}{\log n}\right)$.

The width and size of refuting a CNF formula F are closely related to the size of refuting the composed formula $F \circ \textsc{Ind}_{n^{1+\varepsilon}}^n$. The following claim is folklore – the proof is straightforward and may be found in, e.g., [SBI04].

Claim 20 ([SBI04], [GGKS20])**.** *Let F be an unsatisfiable CNF formula. If F admits a resolution refutation of width w and size s, then $F \circ \textsc{Ind}_t^n$ admits a resolution refutation of size $st^{O(w)}$.*

With Theorem 19 and Claim 20 at hand we are ready to prove Theorem 2. Let us restate it here for convenience.

Theorem 2 (Main result for proofs)**.** *There is an n-variate 3-CNF formula F such that:*

(i') There is a resolution refutation of F in size $n^{O(\log n)}$.
(ii') Every resolution refutation of F in depth $n^{O(1)}$ requires size $\exp(n^{\Omega(1)})$.

Let us remark that the lower bound in Theorem 2 even holds for the semantic Cutting Planes proof system.

Proof. Let $\varepsilon > 0$ and choose $F \coloneqq \text{Br}_n \circ \textsc{Ind}_{n^{1+\varepsilon}}^n$. Recall that the formula Br_n is a 3-CNF formula with $\text{poly}(n)$ clauses and $\text{poly}(n)$ variables. This implies in particular that F contains $\text{poly}(n)$ clauses and is defined over $\text{poly}(n)$ variables.

Note that *(i')* follows immediately from Theorem 3 along with Claim 20. We show *(ii')* by contradiction: suppose there is a resolution refutation of F of depth n^γ and size 2^{n^δ}. By Theorem 19 there exists a resolution proof Π of Br_n of width $n^\delta / \log n$ and depth $n^{\gamma+\delta}/\log n$. This is in contradiction to Theorem 4 for small δ. □

Let us turn our attention to Theorem 1, that is, our main result regarding monotone circuits. The proof follows by similar arguments as the proof of Theorem 2. We again apply a dag-like lifting theorem [GGKS20]. The below statement appeared in this form in [FPR22].

Theorem 21 ([GGKS20, LMM+22, FPR22])**.** *Let F be an unsatisfiable k-CNF formula on n variables and m clauses and let $\varepsilon > 0$ be constant. There is a monotone Boolean function f_F on $mn^{k(1+\varepsilon)}2^k$ variables such that any monotone circuit C computing f_F implies a resolution refutation Π of F satisfying:*

- $\text{size}(\Pi) = O(\text{size}(C))$;
- $\text{depth}(\Pi) = O\left(\text{depth}(C)\frac{\log \text{size}(C)}{\log n}\right)$.

Any resolution refutation Π^* of F implies monotone circuit C for f_F of size $n^{O(\text{width}(\Pi^*))}$.

The monotone function f_F can be extracted in several ways [Raz90, Gál98, GP18, HP17] from a CNF formula F. We refer the interested reader to [Rob18] for a detailed discussion. Let us restate our main result regarding monotone circuits.

Theorem 1 (Main result for circuits)**.** *There is a monotone $f\colon \{0,1\}^n \to \{0,1\}$ such that:*

(i) There is a monotone circuit computing f in size $n^{O(\log n)}$.
(ii) Every monotone circuit computing f in depth $n^{O(1)}$ requires size $\exp(n^{\Omega(1)})$.

Proof. We use the formula $F \coloneqq \mathrm{Br}_n$ and by Theorem 21 we create a monotone function f_F. The formula Br_n is a 3-CNF formula with $\mathrm{poly}(n)$ variables and clauses. Hence f_F is defined over $\mathrm{poly}(n)$ variables

According to Theorem 3 there is a width $O(\log n)$ resolution refutation of Br_n. Hence *(i)* readily follows by Theorem 21. We show *(ii)* by contradiction: suppose there is a monotone circuit computing f_F in depth n^γ and size 2^{n^δ}. By Theorem 21 there exists a resolution proof Π of Br_n of size 2^{n^δ} and depth $n^{\gamma+\delta}/\log n$. For small δ this contradicts Theorem 2. □

A Omitted Proofs

In the following we prove Propositions 5 and 6. Before diving into the respective proofs we need to record a simple fact.

Claim 22. *Denote by k the index-width of F and consider clauses A, B such that $A \vee B$ has index-width $\leq k$. For any clause D there is a constant size resolution derivation of*

- *$D \vee y_{A\vee B}$ from $D \vee y_A \vee y_B$ and F',*
- *$D \vee y_A \vee y_B$ from $D \vee y_{A\vee B}$ and F'.*

Proof. Consider the first statement. Recall that F' contains clauses that ensure that $y_{A\vee B} = y_A \vee y_B$. Hence $y_{A\vee B}$ semantically follows from these clauses of F' and the clause $y_A \vee y_B$. As resolution is implicationally complete there is a resolution derivation of $y_{A\vee B}$ from $y_A \vee y_B$ and the relevant clauses of F'. Since this derivation is over a constant number of variables it is of constant size. We obtain the desired statement by replacing the clause $y_A \vee y_B$ by $D \vee y_A \vee y_B$ and propagating this replacement through the resolution derivation. The second statement follows by a similar consideration. □

Proposition 5. *If F admits an index-width-w depth-d resolution refutation, then F' admits a width-$O(w)$ depth-$O(dw)$ resolution refutation.*

Proof. Let $\pi = (D_1, D_2, \ldots, D_s)$ be a resolution refutation of the formula F in index-width w and depth d. Note that each clause $D_i \in \pi$ can be written as $D_i = C_{i,1} \vee C_{i,2} \vee \cdots \vee C_{i,n}$ where $C_{i,j}$ is a clause that consists of variables related to index j only.

Denote by τ the function mapping each $D_i \in \pi$ to the clause $y_{C_{i,1}} \vee y_{C_{i,2}} \vee \cdots \vee y_{C_{i,n}}$. Let $\pi' \coloneqq (\tau(D_1), \tau(D_2), \ldots, \tau(D_s))$ and note that every clause in π' has width at most w. The sequence π' is *not* a resolution refutation yet. In the following we show that any clause $\tau(D_i)$ can be derived from previous clauses in width $O(w)$ and depth $O(dw)$ thereby establishing the claim. Let us consider two cases.

1. The clause D_i is an axiom of the index-width-k formula F. Hence D_i contains variables related to at most k distinct indices and we can thus write $D_i = C_{i,j_1} \vee \cdots \vee C_{i,j_k}$. Using Claim 22 k times we obtain a derivation of $y_{C_{i,j_1}} \vee \cdots \vee y_{C_{i,j_k}} = \tau(D_i)$ from the axioms of F' where we rely on the fact that $y_{D_i} \in F'$. This establishes the first case.

2. The clause D_i is obtained by an application of the resolution rule from clauses $A \vee x$ and $B \vee \neg x$. In this case, we have already derived $\tau(A \vee x)$ and $\tau(B \vee \neg x)$. According to Claim 22 there is a constant sized resolution derivation of $\tau(A) \vee y_x$ and $\tau(B) \vee y_{\neg x}$ from $\tau(A \vee x)$, $\tau(B \vee \neg x)$ and F'. From $\tau(A) \vee y_x$ and $\tau(B) \vee y_{\neg x}$ and the axiom $\neg y_x \vee \neg y_{\neg x}$ we can derive $\tau(A) \vee \tau(B)$ by applying the resolution rule twice.

   We can write $\tau(A) \vee \tau(B) = y_{A_1} \vee y_{A_2} \vee \cdots \vee y_{A_n} \vee y_{B_1} \vee y_{B_2} \vee \cdots \vee y_{B_n}$ where $A_i \subseteq A$ ($B_i \subseteq B$) is the part of A (of B) related to index i. Since the clauses A, B are of index-width at most $w - 1$ there are at most $2w - 2$ indices in the above expansion. Hence by appealing at most w times to Claim 22 we obtain $y_{A_1 \vee B_1} \vee y_{A_2 \vee B_2} \vee \cdots \vee y_{A_n \vee B_n} = \tau(D_i)$ as required. □

Proposition 6. *If F' admits a width-w depth-d resolution refutation, then F admits an index-width-$O(w)$ depth-$O(d \log |\Sigma|)$ resolution refutation.*

Proof. Consider a width-w depth-d resolution refutation $\pi = (D_1, D_2, \ldots, D_s)$ of the formula F'. To show the existence of a small depth and index-width resolution proof of F we exhibit a strategy for the prover to win the prover–adversary game (see Section 3.1) in memory-size $5w$.

A partial assignment $\rho \in (\Sigma \cup \{*\})^n$ induces a partial assignment μ_ρ to the variables of F' by $\mu_\rho(y_D) = D|_\rho$, where $\mu_\rho(y_D) = *$ if and only if ρ maps all the variables of D into $\{0, *\}$ and at least one variable is mapped to $*$.

Throughout the strategy the prover maintains a clause D and a partial assignment $\rho \in (\Sigma \cup \{*\})^n$ such that

- μ_ρ falsifies D, and
- ρ is minimal with the above property.

The prover attempts to trace a path in the graph of the resolution proof π starting from contradiction $D_s = \bot$. In each round the prover updates D to one of the premises of D to eventually reach an axiom of F'. Since by definition μ_ρ cannot violate an extension axiom of F' the prover must reach a clause y_D for $D \in F$. Since the assignment μ_ρ falsifies y_D if and only if ρ falsifies D the prover wins the game.

Initially $\rho := *^n$. Suppose that the prover remembers the clause D_i and that ρ satisfies the aforementioned properties. If D_i is derived from $D_j = A \vee y_C$ and $D_k = B \vee \neg y_C$ by an application of the resolution rule the prover queries all symbols mentioned by C (of constant index-width) and updates ρ accordingly. Either D_j or D_k is falsified by μ_ρ. Update D_j to point to the falsified clause and shrink ρ. □

Acknowledgements

We thank Alexandros Hollender (who declined a coauthorship) for discussions that inspired the bracket principle. M.G., G.M., and D.S. are supported by the Swiss State Secretariat for Education, Research and Innovation (SERI) under contract number MB22.00026. K.R. is supported by Swiss National Science Foundation project 200021-184656 "Randomness in Problem Instances and Randomized Algorithms".

References

[AD08] Albert Atserias and Víctor Dalmau. A combinatorial characterization of resolution width. *Journal of Computer and System Sciences*, 74(3):323–334, 2008. doi:10.1016/j.jcss.2007.06.025.

[ADD16] Aviv Adler, Constantinos Daskalakis, and Erik Demaine. The complexity of hex and the Jordan curve theorem. In *Proceedings of the 43rd International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 55 of *LIPIcs*, pages 24:1–24:14. Schloss Dagstuhl, 2016. doi:10.4230/LIPIcs.ICALP.2016.24.

[BBI16] Paul Beame, Christopher Beck, and Russell Impagliazzo. Time-space trade-offs in resolution: Superpolynomial lower bounds for superlinear space. *SIAM Journal on Computing*, 45(4):1612–1645, 2016. doi:10.1137/130914085.

[Ber12] Christoph Berkholz. On the complexity of finding narrow proofs. In *Proceedings of the 53rd Symposium on Foundations of Computer Science (FOCS)*, volume 40, pages 351–360. IEEE, 2012. doi:10.1109/FOCS.2012.48.

[BLV24] Christoph Berkholz, Moritz Lichter, and Harry Vinall-Smeeth. Supercritical size-width tree-like resolution trade-offs for graph isomorphism. Technical report, arXiv, 2024. doi:10.48550/ARXIV.2407.17947.

[BN20] Christoph Berkholz and Jakob Nordström. Supercritical space-width trade-offs for resolution. *SIAM Journal on Computing*, 49(1):98–118, 2020. doi:10.1137/16M1109072.

[BN23] Christoph Berkholz and Jakob Nordström. Near-optimal lower bounds on quantifier depth and Weisfeiler–Leman refinement steps. *Journal of the ACM*, 70(5):32:1–32:32, 2023. doi:10.1145/3195257.

[BNT13] Chris Beck, Jakob Nordström, and Bangsheng Tang. Some trade-off results for polynomial calculus: extended abstract. In *Proceedings of the 45th Symposium on Theory of Computing (STOC)*, pages 813–822. ACM, 2013. doi:10.1145/2488608.2488711.

[BS06] Joshua Buresh-Oppenheim and Rahul Santhanam. Making hard problems harder. In *Proceedings of the 21st Conference on Computational Complexity (CCC)*, pages 73–87. IEEE, 2006. doi:10.1109/ccc.2006.26.

[BT24] Sam Buss and Neil Thapen. A simple supercritical tradeoff between size and height in resolution. Technical Report TR24-001, Electronic Colloquium on Computational Complexity (ECCC), 2024. URL: https://eccc.weizmann.ac.il/report/2024/001.

[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.

[CD24] Arkadev Chattopadhyay and Pavel Dvořák. Super-critical trade-offs in resolution over parities via lifting. Technical Report TR24-132, Electronic Colloquium on Computational Complexity (ECCC), 2024. URL: https://eccc.weizmann.ac.il/report/2024/132/.

[dRFJ+24] Susanna de Rezende, Noah Fleming, Duri Janett, Jakob Nordström, and Shuo Pang. Truly supercritical trade-offs for resolution, cutting planes, monotone circuits, and Weisfeiler–Leman. Technical report, 2024.

[dRMN+20] Susanna de Rezende, Or Meir, Jakob Nordström, Toniann Pitassi, Robert Robere, and Marc Vinyals. Lifting with simple gadgets and applications to circuit and proof complexity. In *Proceedings of the 61st Symposium on Foundations of Computer Science (FOCS)*, volume 169, pages 24–30. IEEE, 2020. doi:10.1109/FOCS46700.2020.00011.

[dRNV16] Susanna de Rezende, Jakob Nordström, and Marc Vinyals. How limited interaction hinders real communication (and what it means for proof and circuit complexity). In *Proceedings of the 57th Symposium on Foundations of Computer Science (FOCS)*, pages 295–304, 2016. doi:10.1109/FOCS.2016.40.

[FGI+21] Noah Fleming, Mika Göös, Russell Impagliazzo, Toniann Pitassi, Robert Robere, Li-Yang Tan, and Avi Wigderson. On the power and limitations of branch and cut. In *Proceedings of the 36th Conference on Computational Complexity (CCC)*. Schloss Dagstuhl, 2021. doi:10.4230/LIPIcs.CCC.2021.6.

[FGMS20] John Fearnley, Spencer Gordon, Ruta Mehta, and Rahul Savani. Unique end of potential line. *Journal of Computer and System Sciences*, 114:1–35, 2020. doi:10.1016/j.jcss.2020.05.007.

[FPR22] Noah Fleming, Toniann Pitassi, and Robert Robere. Extremely deep proofs. In *Proceedings of the 13th Innovations in Theoretical Computer Science Conference (ITCS)*, volume 215 of *LIPIcs*, pages 70:1–70:23. Schloss Dagstuhl, 2022. doi:10.4230/LIPICS.ITCS.2022.70.

[Gál98] Anna Gál. A characterization of span program size and improved lower bounds for monotone span programs. In *Proceedings of the 30th Symposium on Theory of Computing (STOC)*, pages 429–437. ACM, 1998. doi:10.1145/276698.276855.

[GGKS20] Ankit Garg, Mika Göös, Pritish Kamath, and Dmitry Sokolov. Monotone circuit lower bounds from resolution. *Theory of Computing*, 16(1):1–30, 2020. doi:10.4086/toc.2020.v016a013.

[GHJ+24a] Mika Göös, Alexandros Hollender, Siddhartha Jain, Gilbert Maystre, William Pires, Robert Robere, and Ran Tao. Further collapses in TFNP. *SIAM Journal on Computing*, 53(3):573–587, 2024. doi:10.1137/22M1498346.

[GHJ+24b] Mika Göös, Alexandros Hollender, Siddhartha Jain, Gilbert Maystre, William Pires, Robert Robere, and Ran Tao. Separations in proof complexity and TFNP. *Journal of the ACM*, 71(4), 2024. doi:10.1145/3663758.

[GLN23] Martin Grohe, Moritz Lichter, and Daniel Neuen. The iteration number of the Weisfeiler–Leman algorithm. In *Proceedings of the 38th Symposium on Logic in Computer Science (LICS)*, pages 1–13, 2023. doi:10.1109/LICS56636.2023.10175741.

[GLNS23] Martin Grohe, Moritz Lichter, Daniel Neuen, and Pascal Schweitzer. Compressing CFI graphs and lower bounds for the Weisfeiler–Leman refinements. In *Proceedings of the 64th Symposium on Foundations of Computer Science (FOCS)*, pages 798–809. IEEE, 2023. doi:10.1109/FOCS57990.2023.00052.

[GNRS24] Mika Göös, Ilan Newman, Artur Riazanov, and Dmitry Sokolov. Hardness condensation by restriction. In *Proceedings of the 56th Symposium on Theory of Computing (STOC)*, pages 2016–2027. ACM, 2024. doi:10.1145/3618260.3649711.

[GP18] Mika Göös and Toniann Pitassi. Communication lower bounds via critical block sensitivity. *SIAM Journal on Computing*, 47(5):1778–1806, 2018. doi:10.1137/16M1082007.

[HMR24] Alexandros Hollender, Gilbert Maystre, and Kilian Risse. Unpublished manuscript, 2024.

[HP17] Pavel Hrubeš and Pavel Pudlák. Random formulas, monotone circuits, and interpolation. In *Proceedings of the 58th Symposium on Foundations of Computer Science (FOCS)*, pages 121–131. IEEE, 2017. doi:10.1109/FOCS.2017.20.

[Hru24] Pavel Hrubeš. Hard submatrices for non-negative rank and communication complexity. In *Proceedings of the 39th Conference on Computational Complexity (CCC)*, volume 300 of *LIPIcs*, pages 13:1–13:12. Schloss Dagstuhl, 2024. doi:10.4230/LIPIcs.CCC.2024.13.

[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.

[KPPY84] Maria Klawe, Wolfgang Paul, Nicholas Pippenger, and Mihalis Yannakakis. On monotone formulae with restricted depth. In *Proceedings of the 16th Symposium on Theory of Computing (STOC)*, pages 480–487. ACM, 1984. doi:10.1145/800057.808717.

[KW88] Mauricio Karchmer and Avi Wigderson. Monotone circuits for connectivity require super-logarithmic depth. In *Proceedings of the 20th Symposium on Theory of Computing (STOC)*, pages 539–550. ACM, 1988. doi:10.1145/62212.62265.

[LMM+22] Shachar Lovett, Raghu Meka, Ian Mertz, Toniann Pitassi, and Jiapeng Zhang. Lifting with sunflowers. In *Proceedings of the 13th Innovations in Theoretical Computer Science Conference (ITCS)*, volume 215 of *LIPIcs*, pages 104:1–104:24. Schloss Dagstuhl, 2022. doi:10.4230/LIPICS.ITCS.2022.104.

[PR23] Theodoros Papamakarios and Alexander Razborov. Space characterizations of complexity measures and size-space trade-offs in propositional proof systems. *Journal of Computer and System Sciences*, 137:20–36, 2023. doi:10.1016/j.jcss.2023.04.006.

[Pud00] Pavel Pudlák. Proofs as games. *American Mathematical Monthly*, pages 541–550, 2000. doi:10.2307/2589349.

[Raz90] Alexander Razborov. Applications of matrix methods to the theory of lower bounds in computational complexity. *Combinatorica*, 10(1):81–93, 1990. doi:10.1007/BF02122698.

[Raz16] Alexander Razborov. A new kind of tradeoffs in propositional proof complexity. *Journal of the ACM*, 63(2):16:1–16:14, 2016. doi:10.1145/2858790.

[Raz17a] Alexander Razborov. On space and depth in resolution. *Computational Complexity*, 27(3):511–559, 2017. doi:10.1007/s00037-017-0163-1.

[Raz17b] Alexander Razborov. On the width of semialgebraic proofs and algorithms. *Mathematics of Operations Research*, 42(4):1106–1134, 2017. doi:10.1287/moor.2016.0840.

[RM99] Ran Raz and Pierre McKenzie. Separation of the monotone NC hierarchy. *Combinatorica*, 19(3):403–435, 1999. doi:10.1007/s004930050062.

[Rob18] Robert Robere. *Unified Lower Bounds for Monotone Computation*. PhD thesis, University of Toronto, Canada, 2018. URL: http://hdl.handle.net/1807/92007.

[Ros15] Benjamin Rossman. Correlation bounds against monotone NC^1. In *Proceedings of the 30th Conference on Computational Complexity (CCC)*, volume 33 of *LIPIcs*, pages 392–411. Schloss Dagstuhl, 2015. doi:10.4230/LIPIcs.CCC.2015.392.

[Ros24] Benjamin Rossman. Formula size-depth tradeoffs for iterated sub-permutation matrix multiplication. In *Proceedings of the 56th Symposium on Theory of Computing (STOC)*, pages 1386–1395. ACM, 2024. doi:10.1145/3618260.3649628.

[SBI04] Nathan Segerlind, Sam Buss, and Russell Impagliazzo. A switching lemma for small restrictions and lower bounds for k-DNF resolution. *SIAM Journal on Computing*, 33(5):1171–1200, 2004. doi:10.1137/S0097539703428555.

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il