Electronic Colloquium on Computational Complexity, Report No. 172 (2022)

Lifting to Parity Decision Trees Via Stifling

Arkadev Chattopadhyay* Nikhil S. Mande† Swagato Sanyal‡ Suhail Sherif§

Abstract

We show that the deterministic decision tree complexity of a (partial) function or relation f lifts to the deterministic parity decision tree (PDT) *size* complexity of the composed function/relation $f \circ g$ as long as the gadget g satisfies a property that we call *stifling*. We observe that several simple gadgets of constant size, like Indexing on 3 input bits, Inner Product on 4 input bits, Majority on 3 input bits and random functions, satisfy this property. It can be shown that existing randomized communication lifting theorems ([Göös, Pitassi, Watson. SICOMP'20], [Chattopadhyay et al. SICOMP'21]) imply PDT-size lifting. However there are two shortcomings of this approach: first they lift *randomized* decision tree complexity of f, which could be exponentially smaller than its deterministic counterpart when either f is a partial function or even a total search problem. Second, the size of the gadgets in such lifting theorems are as large as logarithmic in the size of the input to f. Reducing the gadget size to a constant is an important open problem at the frontier of current research.

Our result shows that even a random constant-size gadget does enable lifting to PDT size. Further, it also yields the first systematic way of turning lower bounds on the *width* of tree-like resolution proofs of the unsatisfiability of constant-width CNF formulas to lower bounds on the *size* of tree-like proofs in the resolution with parity system, i.e., $Res(\oplus)$, of the unsatisfiability of closely related constant-width CNF formulas.

1 Introduction

Theorems that lift the complexity of a function in a weaker model of computation to that of the complexity of a closely related function in a stronger model, have proved to be extremely useful and is a dominant theme of current research (see, for example, [GPW18, CKLM19, GPW20, CFK+21, LMM+22]). One early use of this theme is in the celebrated work of Raz and McKenzie [RM99], that built upon the earlier work of Edmonds et al. [ERIS91] to yield a separation of the hierarchy of monotone circuit complexity classes within NC. The central tool of that work is an argument lifting the query complexity of a (partial) relation f to the two-party communication complexity of the composed relation $f \circ g$, where g is a two-party function, now commonly called a gadget. It required much technical innovation to prove the natural intuition that, if the gadget g is obfuscating enough, the best that the two players can do to evaluate $f \circ g$ is to follow an optimal decision tree algorithm $\mathcal{Q}$ for f, solving the relevant instance of g to resolve each query of $\mathcal{Q}$ encountered along its path of execution. The major utility of this theorem lies in the fact that it reduces the difficult task of proving communication complexity lower bounds to the much simpler task of proving decision tree complexity lower bounds. Indeed, this point was driven home more recently, in the beautiful work of Göös, Pitassi and Watson [GPW18], who established tight quadratic gaps between communication complexity and (rectangular) partition number, resolving a longstanding open problem. This was, arguably, largely possible as the task was reduced to finding an appropriate analog of that separation in the query world. This last

*Tata Institute of Fundamental Research, Mumbai. Partially supported by the MATRICS grant MTR/2019/001633 of the Science & Engineering Research Board of the DST, India arkadev.c@tifr.res.in

†QuSoft and CWI, Amsterdam. Supported by the Dutch Research Council (NWO), as part of the Quantum Software Consortium programme (project number 024.003.037) Nikhil.Mande@cwi.nl

‡Indian Institute of Technology, Kharagpur. Supported by an ISIRD grant by SRIC, IIT Kharagpur swagato@cse.iitkgp.ac.in

§Vector Institute, Toronto suhail.sherif@gmail.com

ISSN 1433-8092

work triggered a whole lot of diverse work on such lifting theorems. Such theorems have been proved for the randomized model [GPW20, CFK+21], models with application to data-structures [CKLM18], proof and circuit complexity (see, for example, [dRNV16, GGKS20, GKMP20]), quantum computing [ABK21], extended formulations [KMR17] etc.

Most of these lifting theorems work with gadgets whose size is at least logarithmic in the arity of the outer function. A current challenge in the area is to break this barrier and prove lifting with sub-logarithmic gadget size (see, for example, [LMM+22]), ultimately culminating in constant-size gadgets. It is not clear whether existing ideas/methods can be pushed to achieve this. Faced with this, we consider lifting decision-tree (DT) complexity to intermediate models that are more complex than DT but simpler than 2-party communication. Parity decision tree (PDT) is a natural choice for such a model. Indeed, in tackling another major open problem in communication complexity, the Log-Rank Conjecture (LRC), researchers have realized the importance and utility of such intermediate models. For instance, the natural analogue of LRC is open for PDTs (see [TWXZ13, TXZ16]) giving rise to a basic conjecture in Fourier analysis. The analogue of LRC was recently proved for AND-decision trees [KLMY21] bringing forth interesting techniques. Further, the randomized analog of LRC, known as the Log-Approximate-Rank Conjecture (LARC) was recently disproved [CMS20] using a surprisingly simple counter-example. The key to discovering this counter-example lay in finding the corresponding counter-example against the analog of the LARC for PDTs.

Taking cue from these developments, we consider lifting DT complexity using small gadgets to PDT and allied complexity in this work. We are able to find an interesting property of gadgets, which we call 'stifling', that we prove is sufficient for enabling such lifting even with constant gadget size. We define a Boolean function $g : \{0,1\}^m \to \{0,1\}$ to be *k-stifled* if for all subsets of k input variables and for all $b \in \{0,1\}$, there exists a setting of the remaining $m-k$ variables that forces the value of g to b (i.e., the values of the k variables are irrelevant under the fixing of these $m-k$ variables). Formally,

Definition 1.1. *Let $g : \{0,1\}^m \to \{0,1\}$ be a Boolean function. We say that g is* k-stifled *if the following holds:*

$$\forall S \subseteq [m] \text{ with } |S| \leq k \text{ and } \forall b \in \{0,1\},$$
$$\exists\ z \in \{0,1\}^{[m]\setminus S} \text{ such that for all } x \in \{0,1\}^m \text{ with } x|_{[m]\setminus S} = z, g(x) = b.$$

We refer the reader to Section 2.1 for formal definitions of complexity measures. We obtain a lifting theorem from decision tree complexity ($\mathsf{DT}(\cdot)$) of a function f to parity decision tree size ($\mathsf{PDTsize}(\cdot)$, denoting the minimum number of nodes in a parity decision tree computing the function) of f composed with a gadget g that is stifled.[1]

Theorem 1.2. *Let $g : \{0,1\}^m \to \{0,1\}$ be a k-stifled function and let $f \subseteq \{0,1\}^n \times \mathcal{R}$ be a relation. Then,*

$$\mathsf{PDTsize}(f \circ g) \geq 2^{\mathsf{DT}(f)\cdot k}.$$

In order to prove this, we start with a PDT for $f \circ g$ of size 2^d, say, and construct a depth-d/k DT for f by 'simulating' the PDT. Using a similar simulation, we also give lower bounds on the *subspace* decision tree complexity of $f \circ g$, where subspace decision trees are decision trees whose nodes can query indicator functions of arbitrary affine subspaces. Let $\mathsf{sDT}(\cdot)$ denote subspace decision tree complexity.

Theorem 1.3. *Let $g : \{0,1\}^m \to \{0,1\}$ be a k-stifled function, and let $f \subseteq \{0,1\}^n \times \mathcal{R}$ be a relation. Then,*

$$\mathsf{sDT}(f \circ g) \geq \mathsf{DT}(f) \cdot k.$$

[1] We remark here that even a lifting theorem from $\mathsf{DT}(f)$ to deterministic communication complexity of $f \circ g$ would not imply Theorem 1.2 as a black box. This is because deterministic communication complexity can be much larger than the logarithm of PDTsize (for instance, for the Equality function). However a lifting theorem from randomized decision tree complexity of f to randomized communication complexity of $f \circ g$ does imply a lifting theorem from *randomized* decision tree complexity of f to PDTsize of $f \circ g$ (see Claim A.3 and the following discussion).

We observe in Claim A.1 that the bounds in the above two theorems are equivalent up to logarithmic factors. More precisely we show that Theorem 1.2 follows (up to a constant factor in the exponent in the RHS) from Theorem 1.3, and that Theorem 1.3 follows (up to a $\log n$ factor in the RHS) from Theorem 1.2. It is easy to see that both of the above theorems individually imply a lifting theorem from $\mathsf{DT}(f)$ to $\mathsf{PDT}(f \circ g)$. We choose to state this as a separate theorem since it is interesting in its own right, and we feel that the proof of Theorem 1.4 gives more intuition.

Theorem 1.4. *Let $g : \{0,1\}^m \to \{0,1\}$ be a k-stifled function, and let $f \subseteq \{0,1\}^n \times \mathcal{R}$ be a relation. Then,*

$$\mathsf{PDT}(f \circ g) \geq \mathsf{DT}(f) \cdot k.$$

Examples of stifled gadgets (functions that are k-stifled for some integer $k \geq 1$) are the well-studied Indexing function and the Inner Product Modulo 2 function, as we show in Claim 1.7 and Claim 1.8. It is worth noting that we obtain tight lifting theorems for *constant-sized* gadgets. Although there is no gadget of arity 1 or 2 that is stifled, both the Indexing gadget on three bits and the Majority gadget on three bits are stifled. Define the Majority function on n input bits by $\mathsf{MAJ}_n(x) = 1$ iff $|\{i \in [n] : x_i = 1\}| \geq n/2$. Define the Indexing function as follows.

Definition 1.5 (Indexing Function)**.** *For a positive integer m, define the Indexing function, denoted $\mathsf{IND}_m : \{0,1\}^{m+2^m} \to \{0,1\}$, by*

$$\mathsf{IND}_m(x, y) = y_{\mathsf{bin}(x)},$$

where $\mathsf{bin}(x)$ denotes the integer in $[2^m]$ represented by the binary expansion x.

In the above definition, we refer to $\{x_i : i \in [m]\}$ as the *addressing variables*, and $\{y_j : j \in [2^m]\}$ as the *target variables*.

Definition 1.6 (Inner Product Modulo 2 Function)**.** *For a positive integer m, define the Inner Product Modulo 2 Function, denoted $\mathsf{IP}_m : \{0,1\}^{2m} \to \{0,1\}$, by*

$$\mathsf{IP}_m(x_1, \ldots, x_n, y_1, \ldots, y_n) = \oplus_{i=1}^{n}(x_i \wedge y_i).$$

We show that IND_m is m-stifled for all integers $m \geq 1$, and IP_m is 1-stifled for all integers $m \geq 2$. We refer the reader to Appendix A for proofs of the three claims below. We also show in Appendix A that the Majority function MAJ_m on m input bits is $(\lceil m/2 \rceil - 1)$-stifled and no Boolean function on m input bits is $\lceil m/2 \rceil$-stifled (hence, 'Majority is stiflest').

Claim 1.7. *For all integers $m \geq 1$, the function IND_m is m-stifled.*

Claim 1.8. *For all integers $m \geq 2$, the function IP_m is 1-stifled.*

We also show that a random Boolean function (the output on each input is chosen independently to be 0 with probability 1/2 and 1 with probability 1/2) on m input variables (where m is sufficiently large) is $(\log m/2)$-stifled with high probability.

Claim 1.9. *Let $g : \{0,1\}^m \to \{0,1\}$ be a random Boolean function with m sufficiently large. Then with probability at least 9/10, g is $((\log m)/2)$-stifled.*

Theorem 1.4, Theorem 1.3 and Theorem 1.2 imply the following results for the Indexing gadget in light of Claim 1.7.

Corollary 1.10. *Let $f \subseteq \{0,1\}^n \times \mathcal{R}$ be a relation. Then for all positive integers m,*

$$\begin{aligned}
\mathsf{PDT}(f \circ \mathsf{IND}_m) &\geq \mathsf{DT}(f) \cdot m \\
\mathsf{sDT}(f \circ \mathsf{IND}_m) &\geq \mathsf{DT}(f) \cdot m, \\
\mathsf{PDTsize}(f \circ \mathsf{IND}_m) &\geq 2^{\mathsf{DT}(f) \cdot m}.
\end{aligned}$$

We now show that the Parity function witnesses tightness of the above corollary. It is easy to see that for all relations f,

$$\mathsf{PDT}(f \circ \mathsf{IND}_m) \leq \mathsf{DT}(f) \cdot (m+1).$$

In fact, our proof can be modified to show $\mathsf{PDT}(f \circ \mathsf{IND}_m) \geq \mathsf{DT}(f) \cdot m + 1$ (see Remark 2.9). Let $\oplus_n$ denote the Parity function on n input variables. It outputs 1 if the number of 1s in the input is odd, and 0 otherwise. It is well known that $\mathsf{DT}(\oplus_n) = n$. We have $\mathsf{PDT}(\oplus_n \circ \mathsf{IND}_m) \leq \mathsf{DT}(\oplus_n) \cdot m + 1 = nm + 1$: query all the m addressing variables in each block to find out the relevant variables using nm queries. The output of the function (which is the parity of all relevant target variables) can now be computed using a single parity query. Thus our result is tight.

Since IP_m is 1-stifled for all integers $m \geq 2$ (Claim 1.8), we analogously obtain the following results for the Inner Product Modulo 2 gadget.

Corollary 1.11. *Let $f \subseteq \{0,1\}^n \times \mathcal{R}$ be a relation. Then for all positive integers $m \geq 2$,*

$$\begin{aligned}
\mathsf{PDT}(f \circ \mathsf{IP}_m) &\geq \mathsf{DT}(f), \\
\mathsf{sDT}(f \circ \mathsf{IP}_m) &\geq \mathsf{DT}(f), \\
\mathsf{PDTsize}(f \circ \mathsf{IP}_m) &\geq 2^{\mathsf{DT}(f)}.
\end{aligned}$$

Since a random gadget on m inputs is $(\Omega(\log m))$-stifled (Claim 1.9), we obtain analogous results for random gadgets as well.

1.1 Consequence for proof complexity

A central question in the area of proof complexity is "Given an unsatisfiable CNF $\mathcal{C}$, what is the size of the smallest polynomial-time verifiable proof that $\mathcal{C}$ is unsatisfiable?" To give lower bounds on the size of such proofs for explicit formulas C, the question has been studied with the restriction that the proof must come from a simple class of proofs. Two of these classes are relevant to us: resolution and linear resolution. Resolution is now well understood and strong lower bounds on the size of resolution based proofs are known for several basic and natural formulas, like the pigeonhole principle, Tseitin formulas and random constant-width CNFs. However, linear resolution introduced by Raz and Tzameret [RT08], where the clauses are conjunctions of affine forms over $\mathbb{F}_2$, are poorly understood. In fact, it remains a tantalizing challenge to prove super-polynomial lower bounds on the size of linear resolution proofs for explicit CNFs. One promising way to make progress on this question is to prove a lifting theorem that lifts ordinary resolution proof-size lower bounds for a CNF to linear resolution proof-size lower bounds for a lifted CNF. Indeed, relatively recently, riding on the success and popularity of lifting theorems on communication protocols, Garg, Göös, Kamath and Sokolov [GGKS20] showed that ordinary resolution proof-size complexity, among other things, can be lifted to cutting plane proof-size complexity. However, lifting to linear resolution proof-size still evades researchers.

A first step towards the above would be to lift *tree-like* resolution proof-size bounds to tree-like linear resolution proof-size bounds. Although lower bounds for tree-like linear resolution proof-size were established by Itsykson and Sokolov [IS20], there was no systematic technique known that would be comparable to the generality of a lifting theorem. Our lifting theorem provides the first such technique as explained below.

The starting point is the well-known fact that the size of the smallest tree-like resolution proof that $\mathcal{C}$ is unsatisfiable is the same as the size of the smallest decision tree that computes the relation $S_{\mathcal{C}} := \{(x, C) : C \text{ is a clause of } \mathcal{C} \text{ and } C(x) = 0\}$. Similarly, the size of the smallest tree-like $Res(\oplus)$ (linear resolution over $\mathbb{F}_2$) proof that $\mathcal{C}$ is unsatisfiable is the same as the size of the smallest parity decision tree that computes $S_{\mathcal{C}}$.

Let $\mathcal{C}$ be a CNF with n variables and t clauses, each of width at most w. For a gadget $g : \{0,1\}^m \to \{0,1\}$, we define the CNF $\mathcal{C} \circ g$ as follows. For each clause $C \in \mathcal{C}$, take the clauses of a canonical CNF computing $\Gamma_C : (y_1, \ldots, y_n) \in (\{0,1\}^m)^n \mapsto C(g(y_1), g(y_2), \ldots, g(y_n))$. Note that Γ_C is a function of at most mw variables, and thus can be expressed as a CNF with at most 2^{mw} clauses, each of width at most mw. Define $\mathcal{C} \circ g$ to be the CNF whose clauses are $\bigcup_{C \in \mathcal{C}} \Gamma_C$.

Our results in this paper yield PDT lower bounds against $S_{\mathcal{C}} \circ g$ given DT lower bounds against $S_{\mathcal{C}}$. To compute $S_{\mathcal{C}} \circ g$ we get as input $y = (y_1, \ldots, y_n) \in (\{0,1\}^m)^n$ and we have to output a clause $C \in \mathcal{C}$ such that $C(x) = 0$ where $x = (g(y_1), \ldots, g(y_n))$. On the other hand to compute $S_{\mathcal{C} \circ g}$ we get as input y and we have to output a clause $D \in \bigcup_{C \in \mathcal{C}} \Gamma_C$ such that $D(y) = 0$. However if $D(y) = 0$ and $D \in \Gamma_C$, then by definition of Γ_C, $C(x) = 0$ where $x = (g(y_1), \ldots, g(y_n))$. Hence computing $S_{\mathcal{C} \circ g}$ is sufficient to compute $S_{\mathcal{C}} \circ g$ and our lower bounds against $S_{\mathcal{C}} \circ g$ translate to lower bounds against $S_{\mathcal{C} \circ g}$, bringing us back to the realm of proof complexity.

Our size lifting (Theorem 1.2) is particularly relevant, when applied with g as the Indexing gadget on 3 bits, for example. Let $\mathcal{C}$ be a 3-CNF $\mathcal{C}$ with n variables and t clauses, with tree-like resolution width at least d. Our theorem implies a tree-like $Res(\oplus)$ size lower bound of 2^d for the CNF $\mathcal{C} \circ \mathsf{IND}_1$, with $3n$ variables and $O(t)$ clauses, each of maximum width at most 9.

1.2 Related work

Independently and concurrently with our work, Beame and Koroth [BK22] obtained closely related results. They show that a particular property of a gadget, conjectured recently by Lovett et al. [LMM+22] to be sufficient for lifting using constant size, is in fact insufficient to yield constant-size lifting. Using properties special to Indexing, they obtain a lifting theorem from decision tree height complexity of f to the complexity of $f \circ \mathsf{IND}$ in a restricted communication model that they define, where IND has constant size. Further, modifying the proof of this result they also lift decision tree complexity of f to PDT size complexity of $f \circ \mathsf{IND}$.

We, on the other hand, prove directly a lifting theorem for PDT size, from deterministic decision tree height, that works for any constant size gadget that satisfies the abstract property of stifling. Examples of such gadgets are IND, IP, Majority and random functions. While there are some similarities between our and their arguments, especially in the use of row-reduction, our proof is entirely linear algebraic with no recourse to information theory.

2 The simulation

In Section 2.1 we introduce necessary notation for proving our theorems. In Section 2.2 we give an overview of the simulation for Theorem 1.4 (i.e., we show an algorithm that takes a low-depth PDT computing $f \circ g$ and constructs a low-depth DT computing f). We do this for the sake of simplicity; the simulations used to prove Theorem 1.2 and Theorem 1.3 are small modifications of the simulation used to prove Theorem 1.4. In Section 2.3 we show the simulation in full detail, with a key subroutine explained in Section 2.4. We then analyze correctness of the simulation in Section 2.5. We conclude the proofs of all of our theorems in Section 2.6.

2.1 Notation

Throughout this paper, we identify $\{0,1\}$ with $\mathbb{F}_2$. We identify vectors in $\mathbb{F}_2^n$ with their characteristic set, i.e., a vector $x \in \mathbb{F}_2^n$ is identified with the set $\{i \in [n] : x_i = 1\}$. For vectors $a, b \in \mathbb{F}_2^n$, let $\langle a, b \rangle$ denote $\sum_{i=1}^n a_i b_i$ modulo 2. We say a vector in $\mathbb{F}_2^n$ is non-zero if it does not equal 0^n. For a positive integer n we use the notation $[n]$ to denote the set $\{1, 2, \ldots, n\}$ and the notation $[n]_0$ to denote the set $\{0, 1, \ldots, n-1\}$. Throughout this paper $\mathcal{R}$ denotes an arbitrary finite set. When clear from context, '+' denotes addition modulo 2.

2.1.1 Decision trees and complexity measures

A *parity decision tree* (PDT) is a binary tree whose leaf nodes are labeled in $\mathcal{R}$, each internal node is labeled by a parity $P \in \mathbb{F}_2^n$ and has two outgoing edges, labeled 0 and 1. On an input $x \in \{0,1\}^n$, the tree's computation proceeds from the root down as follows: compute $\mathsf{val}(P, x) := \langle P, x \rangle$ as indicated by the node's

label and follow the edge indicated by the computed value. Continue in a similar fashion until reaching a leaf, at which point the value of the leaf is output. When the computation reaches a particular internal node, the PDT is said to *query* the parity label of that node. A decision tree (subspace decision tree, respectively), denoted DT (sDT, respectively), is defined as above, except that queries are to individual bits (indicator functions of affine subspaces of $\mathbb{F}_2^n$, respectively) instead of parities as in PDTs.

A DT/PDT/sDT is said to compute a relation $f \subseteq \{0,1\}^n \times \mathcal{R}$ if the output r of the DT/PDT/sDT on input x satisfies $(x,r) \in f$ for all $x \in \{0,1\}^n$. The DT/PDT/sDT complexity of f, denoted $\mathsf{DT}(f)/\mathsf{PDT}(f)/\mathsf{sDT}(f)$, is defined as

$$\mathsf{DT}(f)/\mathsf{PDT}(f)/\mathsf{sDT}(f) := \min_{T:T \text{ is a DT/PDT/sDT computing } f} \text{depth}(T).$$

The decision tree size (parity decision tree size, respectively) of f, denoted $\mathsf{DTsize}(f)$ ($\mathsf{PDTsize}(f)$, respectively), is defined as

$$\mathsf{DTsize}(f)/\mathsf{PDTsize}(f) := \min_{T:T \text{ is a DT/PDT computing } f} \text{size}(T),$$

where the size of a tree is the number of leaves in it.

2.1.2 Composed relations and simulation terminology

For positive integers n, m, a relation $f \subseteq \{0,1\}^n \times \mathcal{R}$, a function $g : \{0,1\}^m \to \{0,1\}$ and strings $\{y_i \in \{0,1\}^m : i \in [n]\}$, we use the notation $g^n(y_1, \ldots, y_n)$ to denote the n-bit string $(g(y_1), \ldots, g(y_n))$. The relation $f \circ g \subseteq (\{0,1\}^m)^n \times \mathcal{R}$ is defined as $(y_1, \ldots, y_n, r) \in (f \circ g) \iff (g(y_1), \ldots, g(y_n), r) \in f$.

We view the input to $f \circ g$ as an mn-bit string, with the bits naturally split into n *blocks*. A parity query P to an input of $f \circ g$ can be specified as an element of $(\mathbb{F}_2^m)^n$. For $i \in [n]$ and a parity query P, $P|_i \in \mathbb{F}_2^m$ refers to the restriction of P to the ith block. For a parity query P and a set of indices $S \subseteq [m] \times [n]$, $P|_S$ refers to the restriction of P to the set of indices S. We say parity P *touches block* i or P *touches a set* S if $P|_i$ is non-zero or $P|_S$ is non-zero, respectively.

A string $y' \in (\{0,1\}^m)^n$ is said to be a *completion* of a partial assignment $y \in (\{0,1,*\}^m)^n$ if $y'_i = y_i$ for all $i \in [m] \times [n]$ with $y_i \neq *$.

For a node N in a PDT and bit $a \in \{0,1\}$, we use $\text{child}(N,a)$ to denote the node reached on answering the parity at the node N as a. For a parity P and a partial assignment y such that $y_i \neq *$ for all $i \in P$, $\mathsf{val}(P,y)$ denotes $\langle P, y \rangle$.

2.2 Overview of the simulation

In this subsection we sketch and describe our simulation algorithm with the goal of proving Theorem 1.4. The proofs of Theorem 1.2 and Theorem 1.3 follow along similar lines, and are formally proved in Section 2.6. We prove Theorem 1.4 via a simulation argument: given a PDT of depth d for $f \circ g$, we construct a DT of depth at most d/k for f.

On input $x \in \{0,1\}^n$, our decision tree algorithm simulates the PDT traversing a path from its root to a leaf, and outputs the value that the PDT outputs at that leaf. During this traversal the decision tree algorithm queries bits of x. It also keeps track of a partial assignment $y \in (\{0,1,*\}^m)^n$ satisfying the following property that guarantees correctness: For all $w \in \{0,1\}^n$ consistent with the queried bits of x there is a completion y' of y such that

- $g^n(y') = w$, and
- the path traversed by the PDT on input y' is the same as that traversed in the simulation.

This ensures that once the simulation reaches a leaf of the PDT, every input w that is consistent with the queried bits of x has the same output as the output at the leaf. That is, the simulation is a valid query algorithm for f.

To carry out the simulation we assign to every parity query along the path a block that the parity query touches, unless its output value is already determined by y. We say that the parity query has that block *marked*. At a high level, if a block has been marked by only a few parity queries, then we have enough freedom in the block to do the following:

- set its output (i.e., the value of g on the variables in the block) to any bit we wish by setting some of the variables in the block (this uses the stifling property of g), and
- retroactively complete this partial assignment so that each of the parity queries marking the block evaluate to any bit of our choosing on the completed input.

We use these to prove the property that guarantees correctness. The first point helps us in getting a partial assignment y such that $g^n(z) = w$ for all completions z of y, and the second helps us complete it to a y' that reaches the leaf we want it to reach. A formal statement of the required properties is in Claim 2.6.

During the simulation once a block, say block i, has been marked by k parity queries we carefully choose k bits of the block that we will keep unset in our partial assignment throughout the simulation. These k bits are what allow us to perform the completion as in the second bullet above. We then query the value of x_i and set the other $m - k$ bits in block i to ensure that the output of block i is set to x_i. The fact that we can do this crucially uses the assumption that the gadget g is k-stifled. In the process above, every parity in the PDT simulation marks at most one block, and a variable x_i is not queried until the ith block is marked k times. Thus the depth of the resultant DT is at most d/k.

A situation that may arise is when a parity P marks block i but a previous parity query P' that marked block i has $P|_i = P'|_i$, for example. However, at some point we may be required to set $P|_i$ to 0 and $P'|_i$ to 1 in order to follow the path in the simulation, which is impossible. To avoid such issues we preprocess each parity query P to ensure that when it marks block i, $P|_i$ is not in the linear span of $\{P'|_i : P' \text{ is an earlier parity that marks block } i\}$. The simulation algorithm is given in Algorithm 1 with the preprocessing subroutine given in Algorithm 2. Algorithm 1 uses two other functions, described below.

Definition 2.1. *Let $g : \{0,1\}^m \to \{0,1\}$ be a k-stifled Boolean function. We define the function STIFLE_g to be a canonical function that takes as input a set $S \subseteq [m]$ with $|S| \leq k$ and a bit b and outputs a partial assignment $y \in \{0,1,*\}^m$ such that*

- *for all $i \in [m]$, $i \in S \iff y_i = *$, and*
- *for all completions y' of y, $g(y') = b$.*

The existence of such a function in the definition above is guaranteed since g is k-stifled.

Definition 2.2. *Define the function $\mathsf{FINDVARS}$ to be a canonical function that takes as input a set $\{P_1, \ldots, P_\ell\}$ of ℓ linearly independent vectors in $\mathbb{F}_2^m$ and outputs a set S of ℓ indices in $[m]$ such that $\{P_i|_S\}_{i\in[\ell]}$ are also linearly independent vectors.*

The existence of such a function in the definition above is guaranteed by the fact that an $\ell \times m$ matrix of rank ℓ has ℓ linearly independent columns, and the $\ell \times \ell$ matrix defined by restricting the original matrix to these columns also has rank ℓ.

2.3 The simulation algorithm

In Algorithm 1, we start with a PDT T of size 2^d for $f \circ g$, and an unknown input $x \in \{0,1\}^n$. Our simulation constructs a decision tree for f of depth at most d/k.[2] This would prove Theorem 1.2. We use the following notation in Algorithm 1:

- num is the number of parity queries simulated in T so far.

[2]Our algorithm is designed to work with $k \geq 1$. For instance, Line 16 is intended to catch the kth time a block is marked, which does not make sense for $k = 0$.

- M is an array storing parity queries made so far (after processing).
- A is an array storing the answers to the parities in M. These are such that answering the queries in M with the answers in A is equivalent to answering the queries in the PDT to reach the node reached by the simulation algorithm.
- For all $i \in [n]$, $\mathsf{MARK}[i]$ is the set of indices in M of parity queries that have the ith block marked. That is, $\mathsf{MARK}[i] = \{j \in [\mathsf{num}]_0 : M[j] \text{ marks block } i\}$.
- y is the partial assignment stored by the simulation algorithm. It is initialized to $(*^m)^n$.
- For all $i \in [n]$, $\mathsf{FREE}[i]$ is the set of indices in block i of y that have not yet been set. That is, $\mathsf{FREE}[i] = \{(i,j) : j \in [m], y_{i,j} = *\}$. It is initialized to $\{i\} \times [m]$ for each $i \in [n]$.
- Q is the set of indices of x queried during the simulation.

The algorithm starts at the root node of the PDT and does the following for each parity query P it comes across in its traversal down the PDT (Line 9).

- In Line 10 it processes the parity query using the ROW-REDUCE subroutine. The subroutine outputs a parity P', a 'correction' bit b and an index $j \in [n]$ denoting which block was marked by the parity (or $\perp$ if no block was marked). As stated in Claim 2.3, these outputs satisfy the following: for all $y \in \{0,1\}^{mn}$,

$$\Big[\mathsf{val}(M[i], y) = A[i]\, \forall i \in [\mathsf{num}]_0\Big] \implies \Big[\mathsf{val}(P, y) = 0 \iff \mathsf{val}(P', y) = b\Big].$$

  In other words, for all inputs consistent with the answers to the previous parity queries made along the path, P evaluates to 0 if and only if P' evaluates to b.
- It adds P' to the list M of processed parities in Line 11. There are now two possibilities:
  - If P' does not mark a block (Line 12), then all variables appearing in P' are already set in y (see Claim 2.5). Hence $A[\mathsf{num}]$ is set to $\mathsf{val}(P', y)$ (Line 13) and the simulation algorithm answers $\mathsf{val}(P', y) + b$ to the original parity query at the current node (Line 26).
  - If P' marks block j:
    - If this is the kth parity to mark block j (Line 16), the simulation queries the value of x_j (Line 17). It then uses the FINDVARS function to select k bits in the jth block to keep unset (Line 19). It sets the remaining $m-k$ bits of the jth block of y using the STIFLE_g function to ensure that the output of the jth block is x_j (Line 20). The stifling property of g is crucially used in this step.
    - The algorithm sets the value of $A[\mathsf{num}]$ (in Line 24) such that the simulation algorithm answers the parity query at the current node (Line 26) so as to go to the smaller subtree.

When it reaches a leaf it outputs the value at that leaf.

2.4 The ROW-REDUCE subroutine

We now discuss the ROW-REDUCE subroutine, which describes how to process an incoming parity and choose which block it should mark. Algorithm 2 takes as inputs P,M,A,MARK,FREE, each of which are described below.

- P is the new parity query to be processed.
- The previously processed parity queries are stored in M.
- The answers given in the simulation to the parities in M are stored in A.

Algorithm 1 Simulation of a PDT T for $f \circ g$ to obtain a DT for f

Input: $x \in \{0,1\}^n$

1: $P_0 \leftarrow \text{root}(T)$ ▷ Starting the simulation of the PDT
2: $\mathsf{num} \leftarrow 0$
3: $M \leftarrow []$
4: $A \leftarrow []$
5: $\mathsf{MARK}[i] \leftarrow \emptyset \; \forall i \in [n]$
6: $y \leftarrow (*^m)^n$
7: $\mathsf{FREE}[i] \leftarrow \{i\} \times [m] \; \forall i \in [n]$
8: $Q \leftarrow \emptyset$
9: **while** P_{num} is not a leaf of T **do**
10: $P', b, j \leftarrow \mathsf{ROW\text{-}REDUCE}(P_{\mathsf{num}}, M, A, \mathsf{MARK}, \mathsf{FREE})$ ▷ Process the current parity query and return the processed query, a correction bit and the index of the marked block.
11: $M[\mathsf{num}] \leftarrow P'$
12: **if** $j = \perp$ **then** ▷ If no block has been marked
13: $A[\mathsf{num}] \leftarrow \mathsf{val}(M[\mathsf{num}], y)$ ▷ By Claim 2.5, all variables appearing in $M[\mathsf{num}]$ have been set in y.
14: **else**
15: $\mathsf{MARK}[j] \leftarrow \mathsf{MARK}[j] \cup \{\mathsf{num}\}$
16: **if** $|\mathsf{MARK}[j]| = k$ **then** ▷ If this is the kth time the jth block is marked
17: Query x_j
18: $Q \leftarrow Q \cup \{j\}$ ▷ Update set of queried variables.
19: $S \leftarrow \mathsf{FINDVARS}(\{M[i]|_j : i \in \mathsf{MARK}[j]\})$ ▷ Select k bits of block j that will remain unset in y.
20: $y|_j \leftarrow \mathsf{STIFLE}_g(S, x_j)$ ▷ Set other bits in block j to force the jth block to output value x_j. This is possible since g is k-stifled.
21: $\mathsf{FREE}[j] \leftarrow \{j\} \times S$
22: **end if**
23: $b' \leftarrow \arg\min_a \text{size}(\text{child}(P_{\mathsf{num}}, a))$
24: $A[\mathsf{num}] \leftarrow b' + b$
25: **end if**
26: $P_{\mathsf{num}+1} \leftarrow \text{child}(P_{\mathsf{num}}, A[\mathsf{num}] + b)$ ▷ If no block was marked (Line 12-13), go to the child as dictated by the set variables. Else, go to the child with the smaller subtree, as ensured by Lines 23 and 24.
27: $\mathsf{num} \leftarrow \mathsf{num} + 1$
28: **end while**
29: Output the value output at P_{num}

- $\mathsf{MARK}[i]$ is the set of indices of parity queries that have the ith block marked.
- $\mathsf{FREE}[i]$ is the set of unset variables in block i in the partial assignment (referred to as y in Section 2.2) of the input to $f \circ g$ that the simulation keeps.

Algorithm 2 The ROW-REDUCE routine

Input: P,M,A,MARK,FREE

1: markedindex $\leftarrow \perp$
2: $P' \leftarrow P$
3: $b \leftarrow 0$
4: **for** j from 1 to n **do**
5: **if** $P'|_{\mathsf{FREE}[j]}$ is in the linear span of $\{M[i]|_{\mathsf{FREE}[j]} : i \in \mathsf{MARK}[j]\}$ **then**
6: Let $S \subseteq \mathsf{MARK}[j]$ be such that $P'|_{\mathsf{FREE}[j]} = \sum_{i \in S} M[i]|_{\mathsf{FREE}[j]}$.
7: $P' \leftarrow P' + \sum_{i \in S} M[i]$ ▷ Observe that $P'|_{\mathsf{FREE}[j]} = 0^{\mathsf{FREE}[j]}$ after this step.
8: $b \leftarrow b + \sum_{i \in S} A[i]$
9: **else**
10: markedindex $\leftarrow j$
11: **break** ▷ Ensure that no more than 1 block is marked.
12: **end if**
13: **end for**
14: **return** $P', b,$ markedindex

The processed parity P' is initialized to P, and a correction bit b is initialized to 0. In Line 4, we go through the blocks of variables in order looking for a block to assign the incoming parity to. The check made on Line 5 is to see whether this parity can mark the current block j. It cannot mark block j if its restriction to $\mathsf{FREE}[j]$ is in the linear span of the restrictions of earlier parities that marked block j. We detail what the algorithm does based on this check below.

- In Lines 5 to 8, we consider the case where the parity being processed, restricted to the jth block, is a linear combination of the restrictions of earlier parities that marked the jth block. That is, there is a set $S \subseteq \mathsf{MARK}[j]$ such that $P'|_{\mathsf{FREE}[j]} = \sum_{i \in S} M[i]|_{\mathsf{FREE}[j]}$. In Line 7 we update the parity by adding $\sum_{i \in S} M[i]$ to P'. This ensures that $P'|_{\mathsf{FREE}[j]} = 0$. Then in Line 8 we update the correction bit b by adding $\sum_{i \in S} A[i]$ to it. This is so that the following holds for all y.

$$\Big[\mathsf{val}(M[i], y) = A[i]\, \forall i \in [\mathsf{num}]_0\Big] \implies \Big[\mathsf{val}(P, y) = 0 \iff \mathsf{val}(P', y) = b\Big].$$

- In Lines 9 to 11, we consider the case when the current parity restricted to $\mathsf{FREE}[j]$ is not a linear combination of the restrictions of the earlier parities that marked the jth block. In this case we have the current parity mark the jth block (Line 10) and then return P', b, j.

Notice that a block is marked only in the latter case. If this case does not occur for any of the $[n]$ blocks, then no block is marked and the procedure returns $P', b, \perp$.

2.5 Correctness

We first address the fact that we primarily deal with the values of the processed parities rather than the original parity queries from the PDT. The following claim says that answering the processed parity is in fact equivalent to answering the original parity from the PDT.

Claim 2.3. *Let P', b, j be the output of ROW-REDUCE when run with its first three inputs being P, M and A. Let* num *be the number of entries in M. Then for all $y \in \{0,1\}^{mn}$,*

$$\Big[\mathsf{val}(M[i], y) = A[i]\, \forall i \in [\mathsf{num}]_0\Big] \implies \Big[\mathsf{val}(P, y) = 0 \iff \mathsf{val}(P', y) = b\Big].$$

Proof. The output P' is computed from the input P only through modifications by Line 7 of Algorithm 2. That is, there is some subset $S \subseteq [\mathsf{num}]_0$ such that $P' = P + \sum_{i \in S} M[i]$. The output b is also modified alongside the modifications of P' so that $b = \sum_{i \in S} A[i]$.

Hence for any input y, $\mathsf{val}(P', y) = \mathsf{val}(P, y) + \sum_{i \in S} \mathsf{val}(M[i], y)$. The claim immediately follows. □

As a corollary, we get that analyzing inputs that answer all the processed parities according to the answer array A is the same as analyzing inputs that follow the path taken by the PDT simulation.

Corollary 2.4. *The following statement is a loop invariant for the while loop in Line 9 of Algorithm 1. For all $y \in \{0,1\}^{mn}$,*

$$\Big[\mathsf{val}(M[i], y) = A[i]\ \forall i \in [\mathsf{num}]_0\Big] \iff \Big[y \text{ reaches } P_{\mathsf{num}}\Big].$$

Proof. The loop is initially entered with the array M being empty and P_{num} being the root of the PDT. Clearly the statement holds at this point. We now prove by induction that it is a loop invariant.

Suppose the statement holds at the beginning of an execution of the loop. $M[\mathsf{num}]$ is populated in this execution by the parity output by the ROW-REDUCE subroutine in Line 10. By Claim 2.3 and the fact that the statement held at the beginning of the execution, every input y that reaches the node P_{num} satisfies $\mathsf{val}(M[\mathsf{num}], y) = \mathsf{val}(P_{\mathsf{num}}, y) + b$. In Line 26 we go to the node $P_{\mathsf{num}+1} = \mathrm{child}(P_{\mathsf{num}}, A[\mathsf{num}] + b)$. Hence the inputs that reach $P_{\mathsf{num}+1}$ are exactly those that reach P_{num} and satisfy $M[\mathsf{num}] = A[\mathsf{num}]$. We then increment num and the loop ends, so the statement holds at the end of an execution of the loop. □

We now show that the processed parities are very structured.

Claim 2.5. *Let* num *be the number of queries made by the simulated PDT at a certain point in the simulation. Recall that the processed parities are stored in $M[0]$ to $M[\mathsf{num} - 1]$.*

1. *Let $i \in [\mathsf{num}]_0$. If the parity $M[i]$ marks block j (i.e., $i \in \mathsf{MARK}[j]$), then for every block $j_1 < j$,*

$$M[i] \cap \mathsf{FREE}[j_1] = \emptyset$$

   where $M[i]$ is viewed as the set of indices $\{\alpha \in [m] \times [n] : M[i]_\alpha = 1\}$.
   If $M[i]$ does not mark any block, then the above holds for all $j_1 \in [n]$.

2. *For every block $j \in [n]$, the parities $\left\{M[i]|_{\mathsf{FREE}[j]} : i \in \mathsf{MARK}[j]\right\}$ are linearly independent.*

Proof. We prove the two properties in order.

1. We prove the first property by induction on i.

   - Let's assume that the property holds for all $i' < i$. Note that $M[i]$ is the processed parity output by ROW-REDUCE when it was called to process P_i. Similarly j is the block marked by ROW-REDUCE during the same call. During the processing, the algorithm ran through all blocks $j_1 < j$ in order (or $j_1 \le n$ if no block was marked) and took the branch of Line 5 of Algorithm 2 in each. To prove that $M[i] \cap \mathsf{FREE}[j_1] = \emptyset$ for each $j_1 < j$, we use a second induction on j_1.
     - When the algorithm looks at block j_1 (i.e., in the j_1th iteration of Line 4 in Algorithm 2), we assume that the partially processed parity P' has, for each $j_2 < j_1$, $P' \cap \mathsf{FREE}[j_2] = \emptyset$. When looking at block j_1, the algorithm may process P' further by adding to it some previously processed parities that had marked block j_1 (Line 7). But any previously processed parity would be stored in $M[i']$ for some $i' < i$. By our first induction hypothesis we know that any such $M[i']$ that had marked block j_1 has, for each $j_2 < j_1$, $M[i'] \cap \mathsf{FREE}[j_2] = \emptyset$. Hence even after adding such parities to P' it still holds that for each $j_2 < j_1$, $P' \cap \mathsf{FREE}[j_2] = \emptyset$. The added parities explicitly ensure that $P' \cap \mathsf{FREE}[j_1] = \emptyset$ and so for each $j_2 \le j_1$, $P' \cap \mathsf{FREE}[j_2] = \emptyset$. This completes the second induction.

   Since this second induction proves the required statement for all $j_1 < j$, this completes the first induction as well.

2. For the second property we note that if a processed parity P' marks a block j (Line 10 of Algorithm 2), then $P'|_{\mathsf{FREE}[j]}$ is not in the linear span of $P''|_{\mathsf{FREE}[j]}$ for previous parities P'' that have block j marked (since the **if** condition Line 5 was not satisfied). Hence the set of parities that mark a block are necessarily linearly independent. Moreover, when $\mathsf{FREE}[j]$ is modified (see Line 19 of Algorithm 1), it is chosen to be the output of the function $\mathsf{FINDVARS}$ on input $\{M[i]|_j : i \in \mathsf{MARK}[j]\}$. By the definition of $\mathsf{FINDVARS}$, this ensures that $\left\{M[i]|_{\mathsf{FREE}[j]} : i \in \mathsf{MARK}[j]\right\}$ remains linearly independent.

□

We finally prove the correctness of the simulation.

Claim 2.6. *For any string $w \in \{0,1\}^n$ consistent with the queried bits of x there is an input y to $f \circ g$ that reaches the leaf P_{num} and satisfies $g^n(y) = w$.*

Proof. Fix a string $w \in \{0,1\}^n$ consistent with the queried bits of x. We start with the partial assignment y maintained by the PDT simulation and complete it.

- **Fixing variables in y to ensure $g^n(y) = w$:** The bits of y that are already fixed ensure that for any queried bit x_j, $g(y|_j) = w_j$ (these are set in Line 20 of Algorithm 1).

  Now for each unqueried bit x_j, first note that $\ell := |\mathsf{MARK}[j]| < k$ (otherwise Line 17 of Algorithm 1 ensures that x_i is queried). We do the following:

  - Get a set $S = \mathsf{FINDVARS}(\{M[i]|_j\}_{i \in \mathsf{MARK}[j]})$ of ℓ indices such that $\{M[i]|_S\}_{i \in \mathsf{MARK}[j]}$ are still linearly independent.
  - Assign $y|_j$ to be $\mathsf{STIFLE}_g(S, w_j)$ to ensure that all the bits of $y|_j$ indexed in S are unset, those outside are set, and all completions of $y|_j$ evaluate to w_j when g is applied to them. This is possible since g is k-stifled and $\ell < k$.[3] We will continue to use $\mathsf{FREE}[j]$ to refer to the set S.

  Now we are guaranteed that any extension of y with bits set as above will yield the string w when g^n is applied to it.

  Before we move on to ensuring that y reaches the leaf P_{num}, let us make the useful observation that at this point $|\mathsf{MARK}[j]| = |\mathsf{FREE}[j]|$ for all $j \in [n]$. We have ensured this for all j that were unqueried by the simulation. For those that were queried, note that the simulation only queries x_j when $|\mathsf{MARK}[j]| = k$, and on querying x_j, $\mathsf{FREE}[j]$ is modified to have size k (Lines 16 and 21 of Algorithm 1). By Claim 2.5, we know that $\left\{M[i]|_{\mathsf{FREE}[j]} : i \in \mathsf{MARK}[j]\right\}$ are linearly independent parities. Hence in Algorithm 2, when processing block j, Line 5 will always fire and block j is never marked again. This ensures that $\mathsf{MARK}[j]$ and $\mathsf{FREE}[j]$ do not get modified again.

- **Extending y to guarantee that it reaches the leaf P_{num}:**

  By Corollary 2.4 the inputs y that reach the leaf P_{num} are exactly those that answer $A[i]$ to each processed parity query $M[i]$.

  We analyze each parity query in M. There are two cases: one where the current parity query under consideration marks a block, and the other where the parity query under consideration does not mark a block.

  - **When $i \notin \bigcup_j \mathsf{MARK}[j]$:** By Claim 2.5 the only non-zero entries of $M[i]$ are variables that have previously been set in y (in Line 20 of Algorithm 1). Hence the parity query $M[i]$ has to evaluate to $\mathsf{val}(M[i], y)$ when queried on any extension of y, and indeed $A[i]$ is set to $\mathsf{val}(M[i], y)$ in Line 13 of Algorithm 1.

[3] It suffices to assume $\ell \leq k$ here. See Remark 2.9 for a discussion on how we can modify our simulation to exploit this.

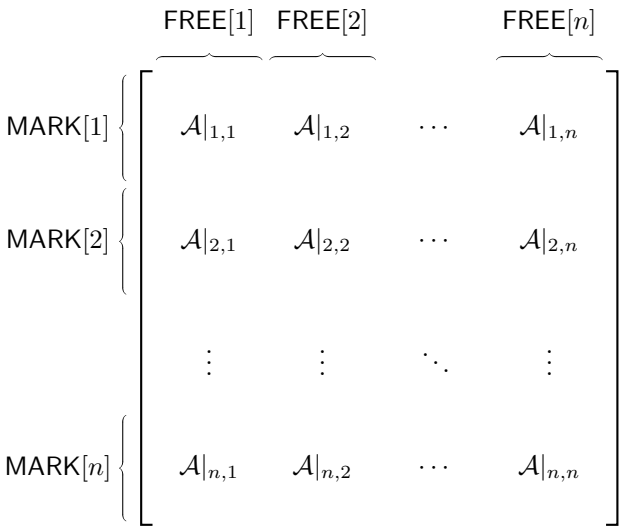

Figure 1: The structure of the matrix $\mathcal{A}$. $\mathcal{A}|_{i,i}$ is a linearly independent submatrix for each $i \in [n]$, and $\mathcal{A}|_{i,j}$ is the all-0 matrix when $i > j$.

– **When $i \in \bigcup_j \mathsf{MARK}[j]$:** Let $I = \bigcup_j \mathsf{MARK}[j]$. Note that $|I| = \sum_j |\mathsf{MARK}[j]|$ since at most one block gets marked for each parity. Since $|\mathsf{FREE}[j]| = |\mathsf{MARK}[j]|$ for all $j \in [n]$, there are still $|I|$ unset bits of y. Let us refer to these variables by $\mathsf{unset}(y)$ and their complement by $\mathsf{set}(y)$.

Let us now construct a matrix $\mathcal{M} \in \mathbb{F}_2^{I \times [mn]}$ where the rows are indexed by I and row i is the vector $M[i]$. We also consider the column vector $b \in \mathbb{F}_2^I$ where the rows are indexed by I and the entry $b_i = A[i]$.

Our goal is to find a column vector $v \in \mathbb{F}_2^{[mn]}$ such that $v_{\mathsf{set}(y)} = y_{\mathsf{set}(y)}$ and $\mathcal{M}v = b$. Setting the unset bits of y according to v would give us what we want, a completion of y that answers $A[i]$ to each parity query $M[i]$. To this end let us write the matrix $\mathcal{M}$ as follows (after an appropriate permutation of columns).

$$\mathcal{M} = \begin{bmatrix} \mathcal{A} & | & \mathcal{A}' \end{bmatrix},$$

where the first set of columns corresponds to indices in $\mathsf{unset}(y)$, and the second set of columns corresponds to indices in $\mathsf{set}(y)$.

Let $b' = \mathcal{A}' v_{\mathsf{set}(y)}$. If we find a setting of $v_{\mathsf{unset}(y)}$ such that $\mathcal{A} v_{\mathsf{unset}(y)} = b + b'$, then $\mathcal{M}v = b$ and this would conclude the proof. We prove the existence of such a setting by noting below that the rows of $\mathcal{A}$ are independent and hence its image is $\mathbb{F}_2^I$. To prove the independence of the rows of $\mathcal{A}$, view $\mathcal{A}$ as a block matrix made up of submatrices $\{\mathcal{A}|_{i,j}\}_{i,j \in [n]}$ with $\mathcal{A}|_{i,j}$ containing the rows $\mathsf{MARK}[i]$ and columns $\mathsf{FREE}[j]$ (see Figure 1). Note that submatrices of the form $\mathcal{A}|_{i,i}$ are square matrices since $|\mathsf{MARK}[i]| = |\mathsf{FREE}[i]|$ for all $i \in [n]$.

By Claim 2.5 and the way we defined $\mathsf{FREE}[i]$ using the $\mathsf{FINDVARS}$ function (ensuring that the restriction of parities in $\mathsf{MARK}[i]$ to $\mathsf{FREE}[i]$ preserves independence),

* $\mathcal{A}|_{i,i}$ is a linearly independent submatrix for each $i \in [n]$, and
* $\mathcal{A}|_{i,j}$ is the all-0 matrix when $i > j$.

The independence of the rows of A easily follows from this 'upper triangular'-like structure.

□

We require the following observation for the proof of Theorem 1.3 since we work with a slightly modified simulation algorithm there.

Observation 2.7. *The correctness analysis of Algorithm 1 described in this section still holds if we set* $A[\mathsf{num}]$ *to* $b_{\mathsf{num}} \in \{0,1\}$ *in Line 24, where* b_{num} *is an arbitrary function of the history of the simulation.*

Finally we observe that the number of queries made by the simulation is at most d/k.

Observation 2.8. *Algorithm 1 makes at most* d/k *queries to* x.

Proof. First observe that every parity in the simulation marks at most one block. The query x_j is made by the simulation (Line 17) only when $\mathsf{MARK}[j]$ gets k elements in it (Line 16). So the number of queries made is at most $\left|\bigcup_{j\in[n]} \mathsf{MARK}[j]\right|/k \leq d/k$, since the number of iterations in the simulation (captured by num) is at most the depth of the PDT, which is d by assumption. □

It follows from the correctness of Algorithm 1 and the observation above that $\mathsf{PDT}(f \circ g) \geq \mathsf{DT}(f) \cdot k$ (proving Theorem 1.4). We observe below that our simulation can be modified so as to give a stronger bound of $\mathsf{PDT}(f \circ g) \geq \mathsf{DT}(f) \cdot k + 1$. Note that this bound is tight since $\mathsf{PDT}(\oplus_n \circ \mathsf{IND}_1) \leq n + 1$, for instance.

Remark 2.9. *Note that we query* x_j *in Algorithm 1 when the* j*th block is marked for the* k*th time. The reason behind this is that we can handle a block being marked* k *times while still having the freedom to set its output by the stifling property of* g*. Moreover this might not be true after the block is marked for the* $(k+1)$*th time. However querying* x_j *in* the same iteration *where the* j*th block is marked for the* k*th time (Lines 16 and 17) is premature. We can modify our algorithm so as to query* x_j *after block* j *is marked* k *times* and *a new parity* attempts to *mark it for the* $(k+1)$*th time. With this modification, once we query* x_j *and set variables so that its output is fixed, the parity can no longer mark block* j *and needs to be reprocessed using Algorithm 2 with the updated value of* $\mathsf{FREE}[j]$ *so that it either triggers a query in a later block (in which case we repeat this process), marks a later block, or marks no block at all. If the set of queries to* x *is denoted by* Q*, then the number of parity queries being made by the simulation is at least* $|Q|k+1$*: When the last of the queries in* Q *is made, say* x_j*, then the* j*th block and all blocks corresponding to previously queried variables had already been marked* k *times* before *the parity query appeared that triggered the query of* x_j*. Hence there were at least* $|Q|k$ *parity queries processed* before *the parity query that resulted in the simulation querying* x_j*. As a result we get*

$$\mathsf{PDT}(f \circ g) \geq \mathsf{DT}(f) \cdot k + 1.$$

2.6 Simulation theorems

In this section we conclude the proofs of Theorem 1.4, Theorem 1.3, Theorem 1.2, Corollary 1.10 and Corollary 1.11.

Proof of Theorem 1.4. It follows from the correctness of Algorithm 1 described in the previous section, and Observation 2.8. □

Proof of Theorem 1.2. Consider a PDT T_P for $f \circ g$ of size s. Suppose the DT T we get for f from the simulation makes d queries on a worst-case input, say on $x \in \{0,1\}^n$. Note that whenever a parity marks a block in the simulation algorithm (i.e., Line 14 fires), the algorithm then chooses to go to the child with the smaller subtree (Lines 24 and 26). So in Line 26, the size of the subtree rooted at $P_{\mathsf{num}+1}$ is at most half the size of the subtree rooted at P_{num}. When the simulation algorithm is run on input x, it reaches a leaf of T_P, and at least dk parities are such that they mark some block in the process. This is because the simulation queries a variable only after its corresponding block has been marked k times. Hence the size of the subtree rooted at the reached leaf is at most a 2^{-dk} fraction of s. Thus, $s \geq 2^{dk}$. The theorem now follows from the correctness of Algorithm 1 described in the previous section. □

Towards the proof of Theorem 1.3, we work with a modified query model instead of considering subspace decision trees. This query model is described below.

For each affine subspace choose an arbitrary ordering of the constraints. We define a 'parity constraint' query to be a query of the form $\langle v, x\rangle \stackrel{?}{=} a$ where $v \in \mathbb{F}_2^{mn}, a \in \mathbb{F}_2$. Define a 'parity constraint' query protocol

as a protocol that makes parity constraint queries. The answer to a query $\langle v, x\rangle \stackrel{?}{=} a$ is either 'Right' if $\langle v, x\rangle = a$ or 'Wrong' if $\langle v, x\rangle \neq a$. The cost of the protocol is defined to be the worst-case number of queries that were answered 'Wrong' during a run of the protocol.

Now given a subspace decision tree T, we construct a parity constraint query protocol as follows:

- When an affine subspace query is made in T, instead query the parity constraints making up the affine subspace query in a canonical order. Stop when any of the parity constraints is answered 'Wrong'.
- If a constraint has been answered 'Wrong', traverse T as if the affine subspace query had failed. If no constraint has been answered 'Wrong', traverse T as if the affine subspace query had succeeded.

Clearly the number of wrong answers in any execution of this protocol is upper bounded by the worst case number of affine subspace queries made by T, and the parity constraint protocol computes the same relation as the original subspace decision tree.

Proof of Theorem 1.3. From a subspace decision tree computing $f \circ g$, construct a parity constraint query protocol T computing $f \circ g$ without increasing the cost, as described above. Note that a parity constraint query protocol tree is just a parity decision tree except that for every internal node its outgoing edges have an extra label of 'Right' and 'Wrong'.

We modify Lines 23 and 24 of Algorithm 1 as follows. We look at the current query node in the PDT to see what answer $b' \in \mathbb{F}_2$ corresponds to a 'Wrong' edge. We set $A[\mathsf{num}]$ to $b' + b$ so that in Line 26 the simulation does take the 'Wrong' edge.

Hence for every $i \in \bigcup_{j\in[n]} \mathsf{MARK}[j]$, the parity query $M[i]$ corresponds to a 'Wrong' answer. Thus note that $\mathrm{cost}(T) \geq \left|\bigcup_{j\in[n]} \mathsf{MARK}[j]\right|$. On the other hand, the query x_j is made by the simulation only when $\mathsf{MARK}[j]$ gets k elements in it. So the number of queries made is at most $\left|\bigcup_{j\in[n]} \mathsf{MARK}[j]\right|/k \leq \mathrm{cost}(T)/k$. The correctness of this modified algorithm follows from the correctness of Algorithm 1 described in the previous section, along with Observation 2.7. □

Corollary 1.10 and Corollary 1.11 follow from Theorem 1.4, Theorem 1.3, Theorem 1.2, the fact that IND_m is m-stifled (Claim 1.7) and the fact that IP_m is 1-stifled (Claim 1.8).

2.7 Non-adaptive and round-respecting simulations

When run on a non-adaptive parity decision tree for $f \circ g$ of cost d, our simulation algorithm actually results in a non-adaptive decision tree for f of cost d/k. This observation relies on the fact that in the course of every iteration (of Line 9 of Algorithm 1), the updates done to M, MARK, Q and FREE depend solely on the sequence of parities P that are processed. More formally,

- The outputs P' and j from the ROW-REDUCE routine are functions of P, M, MARK, k and FREE. They *do not* depend on the input A. M and MARK are then updated depending solely on these outputs.
- The decision to query a variable x_j (and hence to append j to Q, in Lines 17 and 18 of Algorithm 1) depends only on MARK, and the resulting update to $\mathsf{FREE}[j]$ (Line 21 and the two preceding lines of Algorithm 1) only depends on M and MARK.

Now consider a run of the simulation algorithm on a non-adaptive PDT. By Observation 2.8 this results in a DT that computes f with cost at most d/k. By the observations above and since the parity decision tree is non-adaptive, the simulation algorithm processes the same sequence of parity queries regardless of what path it takes down the non-adaptive PDT. Hence the set Q (and hence the queries made) does not depend on what path the simulation algorithm takes, and they can be made non-adaptively. Thus, we obtain the following theorem where $\mathsf{NADT}(\cdot)$ denotes non-adaptive decision tree complexity and $\mathsf{NAPDT}(\cdot)$ denotes non-adaptive parity decision tree complexity.

Theorem 2.10. *Let $g : \{0,1\}^m \to \{0,1\}$ be a k-stifled function, and let $f \subseteq \{0,1\}^n \times \mathcal{R}$ be a relation. Then,*

$$\mathsf{NAPDT}(f \circ g) \geq \mathsf{NADT}(f) \cdot k.$$

This argument can easily be adapted to say that when run on a 't-round' PDT (that is, parity queries are done simultaneously in each round), the simulation algorithm results in a t-round DT with the guarantee that if the parity decision tree has made at most d_i queries during its first i rounds, the decision tree makes at most d_i/k queries during its first i rounds.

References

[ABK21] Anurag Anshu, Shalev Ben-David, and Srijita Kundu. On query-to-communication lifting for adversary bounds. In *36th Computational Complexity Conference (CCC)*, volume 200 of *LIPIcs*, pages 30:1–30:39. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[BK22] Paul Beame and Sajin Koroth. On disperser/lifting properties of the Index and Inner-Product functions. 2022. To appear at ITCS 2023.

[CFK+21] Arkadev Chattopadhyay, Yuval Filmus, Sajin Koroth, Or Meir, and Toniann Pitassi. Query-to-communication lifting using low-discrepancy gadgets. *SIAM J. Comput.*, 50(1):171–210, 2021. Earlier version in ICALP'19.

[CKLM18] Arkadev Chattopadhyay, Michal Koucký, Bruno Loff, and Sagnik Mukhopadhyay. Simulation beats richness: new data-structure lower bounds. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 1013–1020. ACM, 2018.

[CKLM19] Arkadev Chattopadhyay, Michal Koucký, Bruno Loff, and Sagnik Mukhopadhyay. Simulation theorems via pseudo-random properties. *Comput. Complex.*, 28(4):617–659, 2019.

[CMS20] Arkadev Chattopadhyay, Nikhil S. Mande, and Suhail Sherif. The log-approximate-rank conjecture is false. *J. ACM*, 67(4):23:1–23:28, 2020. Earlier version in STOC'19.

[dRNV16] Susanna F. 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 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 295–304. IEEE Computer Society, 2016.

[ERIS91] Jeff Edmonds, Steven Rudich, Russell Impagliazzo, and Jirí Sgall. Communication complexity towards lower bounds on circuit depth. In *32nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 249–257. IEEE Computer Society, 1991.

[GGKS20] Ankit Garg, Mika Göös, Pritish Kamath, and Dmitry Sokolov. Monotone circuit lower bounds from resolution. *Theory Comput.*, 16:1–30, 2020. Earlier version in STOC'18.

[GKMP20] Mika Göös, Sajin Koroth, Ian Mertz, and Toniann Pitassi. Automating cutting planes is NP-hard. In *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 68–77. ACM, 2020.

[GPW18] Mika Göös, Toniann Pitassi, and Thomas Watson. Deterministic communication vs. partition number. *SIAM J. Comput.*, 47(6):2435–2450, 2018. Earlier version in FOCS'15.

[GPW20] Mika Göös, Toniann Pitassi, and Thomas Watson. Query-to-communication lifting for BPP. *SIAM J. Comput.*, 49(4), 2020. Earlier version in FOCS'17.

[IS20] Dmitry Itsykson and Dmitry Sokolov. Resolution over linear equations modulo two. *Ann. Pure Appl. Log.*, 171(1), 2020.

[KLMY21] Alexander Knop, Shachar Lovett, Sam McGuire, and Weiqiang Yuan. Log-rank and lifting for AND-functions. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 197–208. ACM, 2021.

[KMR17] Pravesh K. Kothari, Raghu Meka, and Prasad Raghavendra. Approximating rectangles by juntas and weakly-exponential lower bounds for LP relaxations of csps. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 590–603. ACM, 2017.

[LMM+22] Shachar Lovett, Raghu Meka, Ian Mertz, Toniann Pitassi, and Jiapeng Zhang. Lifting with sunflowers. In Mark Braverman, editor, *13th Innovations in Theoretical Computer Science Conference (ITCS)*, volume 215 of *LIPIcs*, pages 104:1–104:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[RM99] Ran Raz and Pierre McKenzie. Separation of the monotone NC hierarchy. *Combinatorica*, 19(3):403–435, 1999. Earlier version in FOCS'97.

[RT08] Ran Raz and Iddo Tzameret. Resolution over linear equations and multilinear proofs. *Ann. Pure Appl. Log.*, 155(3):194–224, 2008.

[TWXZ13] Hing Yin Tsang, Chung Hoi Wong, Ning Xie, and Shengyu Zhang. Fourier sparsity, spectral norm, and the log-rank conjecture. In *Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 658–667. IEEE Computer Society, 2013.

[TXZ16] Hing Yin Tsang, Ning Xie, and Shengyu Zhang. Fourier sparsity of GF(2) polynomials. In *Proceedings of Computer Science - Theory and Applications - 11th International Computer Science Symposium in Russia (CSR)*, volume 9691 of *Lecture Notes in Computer Science*, pages 409–424. Springer, 2016.

A Deferred proofs

Claim A.1. *Theorem 1.2 follows (up to a constant factor in the exponent in the RHS) from Theorem 1.3. Theorem 1.3 follows (up to a* $\log n$ *factor in the RHS) from Theorem 1.2.*

This claim immediately follows from the following bounds relating $\mathsf{PDTsize}$ and sDT.

Claim A.2. *Let* $F \subseteq \{0,1\}^n \times \mathcal{R}$ *be a relation. Then,*

$$\frac{\log \mathsf{PDTsize}(F)}{\log n} \leq \mathsf{sDT}(F) \leq 2 \log \mathsf{PDTsize}(F).$$

We show the first inequality by a naive simulation of a sDT by a PDT, and the second one via a standard tree/protocol-balancing trick.

Proof. We prove the inequalities in order.

- Consider a sDT T of depth d computing F. Consider the following PDT computing F: Start from the root of T and traverse down until a leaf in the following way. For a subspace query encountered at a node, query its parities in order until encountering a parity that violates the subspace constraint. At this point go to the 0-child of the current node in T and recurse. If no such parity is encountered then go to the 1-child of the current node in T and recurse. Note that the 0-child of a node can be reached in at most n different ways (one for each constraint of the subspace, and since the co-dimension of a subspace is at most n) and the 1-child can be reached in exactly one way. Iterating until we reach a leaf, we obtain a PDT of size at most n^d. Thus,

$$\log \mathsf{PDTsize}(F) \leq \mathsf{sDT}(F) \log n.$$

- Consider a PDT T of size $s = \mathsf{PDTsize}(F)$ computing F. We first claim that there is an internal node v in T such that the number of leaves ℓ in the subtree rooted at it is at least $s/3$ and at most $2s/3$. To see this, first identify a node v' in T with the smallest number of leaves under it among all nodes with at least $2s/3$ leaves under it. The child of this node with the larger subtree has at least $s/3$ leaves and at most $2s/3$ leaves (by the minimality of v'). A subspace DT protocol for F is as follows: Determine whether the input reaches v or not. This can be done with one subspace query since it involves checking whether a set of affine parity constraints are all satisfied (and this is precisely the indicator of an affine subspace). If the input reaches v, then continue in the tree rooted at v. In the other case, we can eliminate the tree rooted at v from T. In either case, the number of leaves in the resultant PDT is at most $2s/3$. Recursively using the same protocol until we reach a leaf, this gives a subspace DT computing F of cost $\log_{3/2} s \leq 2 \log s$.

□

We next state a claim that shows that the randomized communication complexity of a function is bounded from above by its PDT size. Let $\mathsf{R}^{cc}_{\varepsilon}(\cdot)$ denote ε-error randomized communication complexity. It is well known that the public-coin ε-error randomized communication complexity of the Equality function on $2n$ input bits is $O(\log(1/\varepsilon))$. It is easy to see that the same protocol and same upper bound also holds for the communication complexity of checking whether a joint input lies in an arbitrary (but known to both players) affine subspace.

Claim A.3. *Let $F \subseteq \{0,1\}^n \times \{0,1\}^n \times \mathcal{R}$ be a relation. Then,*

$$\mathsf{R}^{cc}(F) \leq O(\log(\mathsf{PDTsize}(F)) \times \log\log(\mathsf{PDTsize}(F))).$$

Proof. Consider a PDT T of size s computing F. Just as in the previous proof, Alice and Bob jointly identify a node v' in T that has at least $s/3$ and at most $2s/3$ leaves under it. They use $O(\log\log s)$ bits of communication to find out with error probability at most $1/(6\log s)$ whether the computation of T on their joint input reaches v'. This can be done since the set of inputs reaching v' is an affine subspace. Irrespective of the outcome, the size of the resultant PDT reduces to at most $2s/3$. The players then recurse at most $2\log s$ times to reach a leaf and output the value at that leaf. By a union bound, the error of this protocol is at most $1/3$. The total cost of this protocol is at most $O(\log s \log\log s)$. □

In particular, the claim above implies that a lifting theorem from randomized decision tree complexity of f to $\mathsf{R}^{cc}(f \circ g)$ (see, for example, [GPW20, CFK+21], for such lifting theorems, albeit for gadgets of super-constant size) implies a lifting theorem from randomized decision tree complexity of f to $\mathsf{PDTsize}(f \circ g)$.

Proof of Claim 1.7. Let S be an arbitrary subset of $[m] \sqcup [2^m]$ of size m, and let $b \in \{0,1\}$. We need to show that there exists a setting of variables in $[m] \sqcup [2^m] \setminus S$ that fixes the value of IND_m to b. Suppose there are ℓ addressing variables and $m - \ell$ target variables in S. Fix the remaining $2^m - m + \ell$ target variables to value b.

We now claim that there exists a setting of the free (outside of S) $m - \ell$ addressing variables such that the 'relevant' target variable is always one of the free (outside of S) target variables, regardless of the setting of the ℓ 'constrained' addressing variables in S. This would prove the claim since we have set all of the free target variables to value b, and hence IND_m would always output b regardless of the assignments to variables in S. Towards a contradiction, suppose not. Then, for each setting of the free $m - \ell$ addressing variables there exists a setting of the constrained addressing variables such that the relevant target variable is one of the $m - \ell$ constrained target variables. Thus,

$$2^{m-\ell} \leq m - \ell,$$

which is easily seen to be false for all $\ell \in \{0, 1, \ldots, m-1\}$ and all integers $m \geq 1$. □

Proof of Claim 1.8. Let the inputs to IP_m be denoted by $x_1, x_2, \ldots, x_m, y_1, y_2, \ldots, y_m$. Pick an arbitrary set of size 1. Without loss of generality we may assume that this set is $\{x_1\}$. Given $b \in \{0,1\}$ we need to show that there exists a setting of the variables $x_2, \ldots, x_m, y_1, \ldots, y_m$ that fixes the value of IP_m to b. To this end, fix $y_1 = 0$. This implies $x_1 \wedge y_1 = 0$. Hence, regardless of the value of x_1, the function now evaluates to $\mathsf{IP}_{m-1}(x_2, \ldots, x_m, y_2, \ldots, y_m)$. Since $m \geq 2$, IP_{m-1} is a non-constant function, and we can set the values of $x_2, \ldots, x_m, y_2, \ldots, y_m$ such that $\mathsf{IP}_{m-1}(x_2, \ldots, x_m, y_2, \ldots, y_m)$ (and thus $\mathsf{IP}_m(x_1, x_2, \ldots, x_m, y_1, y_2, \ldots, y_m)$) evaluates to b. This concludes the proof. □

Remark A.4. *The parameters in Claim 1.7 and Claim 1.8 cannot be improved. That is,*

- IND_m *is not* $(m+1)$*-stifled: Consider the set* S *of all addressing variables and a single target variable. We have* $|S| = m + 1$*. Regardless of how we set the remaining variables, the restricted function is never fixed because the addressing variables could be assigned values to point to the target variable in* S *(which could have value either 0 or 1).*
- IP_m *is not 2-stifled: Consider the set* x_1, y_1*. Regardless of how we set the remaining variables, the function value is different when* $x_1 \wedge y_1 = 1$ *as compared to when* $x_1 \wedge y_1 = 0$*.*

We observe below that 'Majority is stiflest'. More precisely, we show that $\mathsf{MAJ}_n : \{0,1\}^n \to \{0,1\}$ is $(\lceil n/2 \rceil - 1)$-stifled and no Boolean function on n inputs is $\lceil n/2 \rceil$-stifled.

Claim A.5 (Majority is stiflest)**.** *Let* $f : \{0,1\}^n \to \{0,1\}$ *be a Boolean function. Then,* f *cannot be* $\lceil n/2 \rceil$*-stifled. Moreover* MAJ_n *is* $(\lceil n/2 \rceil - 1)$*-stifled.*

Proof. We prove the two claims in order.

- Towards a contradiction, assume f is $\lceil n/2 \rceil$-stifled. Pick an arbitrary set S of size $\lceil n/2 \rceil$. By definition, there exists a setting of variables in $\bar{S}$ that forces the value of f to 0. Since $|\bar{S}| = \lfloor n/2 \rfloor \leq \lceil n/2 \rceil$, the assumption also implies the existence of a setting of variables in S that forces the value of f to 1. Since these settings are on disjoint sets of variables, this implies a setting of variables that forces the value of f to both 0 and 1, which yields a contradiction.
- Pick an arbitrary set of variables S of size $\lceil n/2 \rceil - 1$ and an arbitrary $b \in \{0,1\}$. By definition of the Majority function, fixing all variables in $\bar{S}$ to value b forces the function value to be b, concluding the proof.

□

Proof of Claim 1.9. Fix $S \subseteq [m]$ with $|S| = k = \log m/2$. For $z \in \{0,1\}^{[m]\setminus S}$ and $z' \in \{0,1\}^S$, let the input $(z, z') \in \{0,1\}^m$ be defined by $(z, z')|_S = z'$ and $(z, z')|_{[m]\setminus S} = z$. We have for every fixed $z \in \{0,1\}^{[m]\setminus S}$,

$$\Pr[\forall z' \in \{0,1\}^S, g(z, z') = 0] = \frac{1}{2^{2^k}}.$$

Since these events are independent for each $z \in \{0,1\}^{[m]\setminus S}$,

$$\Pr[\nexists z \in \{0,1\}^{[m]\setminus S} \text{ such that } \forall z' \in \{0,1\}^S, g(z, z') = 0] = \left(1 - \frac{1}{2^{2^k}}\right)^{2^{m-k}}.$$

Thus, the probability that 'S isn't 0-stifled' is at most $\left(1 - \frac{1}{2^{2^k}}\right)^{2^{m-k}}$. The same bound holds for the probability that S isn't 1-stifled. By a union bound over all $S \subseteq [m]$ with $|S| = k$, we get

$$\Pr[g \text{ is not } k\text{-stifled}] \leq \binom{m}{k} \cdot 2 \left(1 - \frac{1}{2^{2^k}}\right)^{2^{m-k}}$$

$$\begin{aligned}
&\leq 2^{k\log m+1}\cdot\exp\left(-2^{m-k-2^k}\right) && \text{using standard inequalities}\\
&\leq 2^{(\log^2 m)/2+1-2^{(m-\log m-\sqrt{m})\log e}}\\
&\leq 1/10,
\end{aligned}$$

where the last inequality holds for sufficiently large m. □

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il