

Hardness of Function Composition for Semantic Read once Branching Programs

Jeff Edmonds *
York University
jeff@cse.yorku.ca

Venkatesh Medabalimi *
University of Toronto
venkatm@cs.toronto.edu

Toniann Pitassi *
University of Toronto
toni@cs.toronto.edu

January 24, 2018

Abstract

In this work, we study time/space tradeoffs for function composition. We prove asymptotically optimal lower bounds for function composition in the setting of *nondeterministic read once branching programs*, for the syntactic model as well as the stronger semantic model of read-once nondeterministic computation. We prove that such branching programs for solving the tree evaluation problem over an alphabet of size k requires size roughly $k^{\Omega(h)}$, i.e space $\Omega(h \log k)$. Our lower bound nearly matches the natural upper bound which follows the best strategy for black-white pebbling the underlying tree. While previous superpolynomial lower bounds have been proven for read-once nondeterministic branching programs (for both the syntactic as well as the semantic models), we give the first lower bounds for iterated function composition, and in these models our lower bounds are near optimal.

*Research supported by NSERC.

1 Introduction

One of the most promising approaches to proving major separations in complexity theory is to understand the complexity of function composition. Given two Boolean functions, $f : \{0, 1\}^m \rightarrow \{0, 1\}$ and $g : \{0, 1\}^n \rightarrow \{0, 1\}$, their composition is the function $f \circ g : \{0, 1\}^{mn} \rightarrow \{0, 1\}$ defined by

$$(f \circ g)(x_1, \dots, x_m) = f(g(x_1), \dots, g(x_m)).$$

The complexity of function composition is one of the most tantalizing and basic problems in complexity theory, and has been studied in a variety of models. There is essentially no general setting where function composition can be computed with substantially less resources than first computing each instance of g , followed by computing f on the outputs of the g 's. Indeed, lower bounds for function composition are known to resolve several longstanding open problems in complexity theory.

The most famous conjecture about function composition in complexity theory is the Karchmer-Raz-Wigderson (KRW) conjecture [26], which asserts that the minimum depth of a circuit (of fan-in 2) that computes $f \circ g$ (over AND, OR and NOT gates) is the minimum depth of computing f plus the minimum depth of computing g . That is,

$$D(f \circ g) = \Theta(D(f) + D(g)).$$

Karchmer, Raz and Wigderson show that repeated applications of this conjecture implies super-logarithmic lower bounds on the depth complexity of an explicit function, thus resolving a major open problem in complexity theory (separating P from NC^1). In particular, The *tree evaluation problem* defines iterated function composition with parameters d and h as follows. The input is an ordered d -ary binary tree of depth $h + 1$. Each of the d^h leaf nodes of the tree is labelled with an input bit, and each non-leaf node of the tree is labelled by a 2^d Boolean vector, which is the truth table of a Boolean function from $\{0, 1\}^d \rightarrow \{0, 1\}$. This induces a 0/1 value for each intermediate node in the tree in the natural way: for a node v with corresponding function f_v , we label v with f_v applied to bits that label the children of v . The output is the value of the root node. The basic idea is to apply $h = O(\log n / \log \log n)$ compositions of a random $d = \log n$ -ary function $f : \{0, 1\}^{\log n} \rightarrow \{0, 1\}$ to obtain a new function over $O(n^2)$ bits that is computable in polynomial time but that requires depth $\Omega(\log^2 n)$ (ignoring lower order terms). Raz and McKenzie resolved the function composition in the monotone case, thus separating monotone P from monotone NC^1 .

In communication complexity, lower bounds for function composition have been extremely successful for solving many open problems. *Lifting Theorems* in communication complexity show how to reduce lower bounds in query complexity to lower bounds in communication complexity via function composition. Raz and McKenzie's lower bound for function composition in the monotone setting [34] (in hindsight) is now viewed as a general lifting theorem for deterministic communication complexity; subsequent lifting theorems have been proven for many other models of computation [8, 20–22, 38]. These theorems have had simplified and unified many results, and in addition have led to the resolution of important open problems in areas such as game theory, proof complexity, extension complexity, and communication complexity [7, 11, 19, 27, 29].

The complexity of function composition for space-bounded computation has also been studied since the 1960's. The classical result of Nečiporuk [32] proves $\Omega(n^2 / \log^2 n)$ size lower bounds

for deterministic branching programs for iterated function composition. Subsequently, Pudlak observed that Nečiporuk’s method can be extended to prove $\Omega(n^{3/2}/\log n)$ size lower bounds for *nondeterministic branching programs*.¹ These classical results are still the best unrestricted branching program size lower bounds known, and it is a longstanding open problem to break this barrier.

In this work, we study time/space tradeoffs for function composition. We prove asymptotically optimal lower bounds for function composition in the setting of *nondeterministic read once branching programs*, for the syntactic model as well as the stronger semantic model of read-once nondeterministic computation. We prove that such branching programs for solving the tree evaluation problem over an alphabet of size k requires size roughly $k^{\Omega(h)}$, i.e space $\Omega(h \log k)$. Our lower bound nearly matches the natural upper bound which follows the best strategy for black-white pebbling the underlying tree. While previous superpolynomial lower bounds have been proven for read-once nondeterministic branching programs (for both the syntactic as well as the semantic models), we give the first lower bounds for iterated function composition, and in these models our lower bounds are near optimal.

1.1 History and Related Work

1.1.1 Function Composition and Direct Sum Conjectures

Raz and McKenzie proved the KRW conjecture in the context of monotone circuit depth [34]. In an attempt to prove the KRW conjecture in the non-monotone case, Karchmer, Raz and Wigderson proposed an intermediate conjecture, known as the universal relation composition conjecture. This intermediate conjecture was proven by Edmonds et.al [14] using novel information-theoretic techniques. More recently some important steps have been taken towards replacing the universal relation by a function using information complexity [18] and communication complexity techniques [13]. Dinur and Meir’s lower bound [13] also gives an alternate proof of the cubic formula size lower bounds for the function $f \circ g$ where g is the parity function. (This bound was originally proven by Håstad [35] and more recently by Tal [36].)

1.1.2 Time-Space Tradeoffs

In the uniform setting, time-space tradeoffs for SAT were achieved in a series of papers [15–17, 30]. Fortnow-Lipton-Viglas-Van Melkebeek [17] shows that any algorithm for SAT running in space $n^{o(1)}$ requires time at least $\Omega(n^{\phi-\epsilon})$ where ϕ is the golden ratio $((\sqrt{5}+1)/2)$ and $\epsilon > 0$. Subsequent works [12, 37] improved the time lower bound to greater than $n^{1.759}$.

The state of the art time/space tradeoffs for branching programs were proven in the remarkable papers by Ajtai [1] and Beame-et-al [3]. In the first paper, Ajtai exhibited a polynomial-time computable Boolean function such that any sub-exponential size deterministic branching program requires superlinear length. This result was significantly improved and extended by Beame-et-al who showed that any sub-exponential size randomized branching program requires length $\Omega(n^{\frac{\log n}{\log \log n}})$.

¹While Nečiporuk’s result is not usually stated this way, it can be seen as a lower bound for function composition. We present this alternative proof in section B in the Appendix.

Lower bounds for nondeterministic branching programs have been more difficult to obtain. Length-restricted nondeterministic branching programs come in two flavors: *syntactic* and *semantic*. A length l syntactic model requires that every path in the branching program has length at most l , and similarly a read- c syntactic model requires that every path in the branching program reads every variable at most c times. In the less restricted semantic model, the read- c requirement is only for *consistent* accepting paths from the source to the 1-node; that is, accepting paths along which no two tests $x_i = d_1$ and $x_i = d_2$, $d_1 \neq d_2$ are made. Thus for a nondeterministic read- c semantic branching program, the overall length of the program can be unbounded.

Note that any syntactic read-once branching program is also a semantic read-once branching program, but the opposite direction does not hold. In fact, Jukna [24] proved that semantic read-once branching programs are exponentially more powerful than syntactic read-once branching programs, via the “Exact Perfect Matching” (EPM) problem. The input is a (Boolean) matrix A , and A is accepted if and only if every row and column of A has exactly one 1 and rest of the entries are 0’s i.e if it’s a permutation matrix. Jukna gave a polynomial-size semantic read-once branching program for EPM, while it was known that syntactic read-once branching programs require exponential size [25, 28].

Lower bounds for syntactic read- c (nondeterministic) branching programs have been known for some time [5, 33]. However, for *semantic* nondeterministic branching programs, even for read-once, no lower bounds are known for polynomial time computable functions for the boolean, $k = 2$ case. Nevertheless exponential lower bounds for semantic read- c (nondeterministic) k -way branching programs, where $k \geq 2^{3c+10}$ were shown by Jukna [23]. More recently [9] obtain exponential size lower bounds for semantic read-once nondeterministic branching programs for $k = 3$, leaving only the boolean case open. Liu [31] proved near optimal size lower bounds for *deterministic* read-once branching programs for function composition.

The rest of the paper is organized as follows. In Section 2 we give the formal definitions, present the natural upper bound and state our main result. In Section 3 we give the intuition and proof outline. Sections 4, 5 and 6 are devoted to individual parts of the proof.

2 Definitions and Statement of Results

Definition 2.1. Let $f : [k]^n \rightarrow \{0, 1\}$ be a boolean valued function whose input variables are x_1, \dots, x_n where $x_i \in [k]$. A **k -way nondeterministic branching program** for f is an acyclic directed graph G with a distinguished source node q_{start} and sink node (the accept node) q_{accept} . We refer to the nodes as *states*. Each non-sink state is labeled with some input variable x_i , and each edge directed out of a state is labelled with a value $b \in [k]$ for x_i . For each input $\vec{\xi} \in [k]^n$, the branching program accepts $\vec{\xi}$ if and only if there exists at least one path starting at q_{start} leading to the accepting state q_{accept} , and such that all labels along this path are consistent with $\vec{\xi}$. The *size* of a branching program is the number of states in the graph. A nondeterministic branching program is ***semantic read-once*** if for every path from q_{start} to q_{accept} that is consistent with some input, each variable occurs at most once along the path.

Syntactic read-once branching programs are a more restricted model where no path can read a

variable more than once; in the semantic read-once case, variables may be read more than once, but each accepting path may only query each variable once.

Definition 2.2. The (ternary) height h tree evaluation problem $Tree_{\vec{F}}$, has an underlying 3-ary tree of height h with $n = 3^{h-1}$ leaves. Each leaf is labelled by a corresponding variable in x_1, \dots, x_n . (Note that a tree with a single node has height 1.) Each internal node v is labeled with a function $F : [k]^3 \rightarrow [k]$, where \vec{F} denotes the vector of these functions. The input $\vec{\xi} \in [k]^n$ gives a value in $[k]$ to the leaf variables \vec{x} . This induces a value for each internal node in the natural way, and the output $Tree_{\vec{F}}(\vec{\xi})$ is the labeling of the root. In the boolean version, the input $\vec{\xi}$ is accepted if and only if $Tree_{\vec{F}}(\vec{\xi}) \in [k^{1-\epsilon}]$ where $\epsilon \in (0, 1)$ is a parameter.

The most natural way to solve the tree evaluation problem is to evaluate the vertices of the tree, via a strategy that mimics the optimal black-white pebbling of the underlying tree. In the next section, we review this upper bound, and show that it corresponds to a nondeterministic semantic read-once branching program of size $\Theta(k^{h+1})$. Our main result gives a nearly matching lower bound (when k is sufficiently large compared to h).

Theorem 2.3. For any h , and k sufficiently large ($k > 2^{42h}$), there exists ϵ and \vec{F} such that any k -ary nondeterministic semantic read-once branching program for $Tree_{\vec{F}}$ requires size $\Omega\left(\frac{k}{\log k}\right)^h$.

We prove the lower bound for the decision version of the tree evaluation problem, with ϵ chosen to be $\frac{9h}{\log k}$. Secondly, we actually show (See appendix C) that the lower bound holds for almost all \vec{F} , whenever each F is independently chosen to be a random 4-invertible function:

Definition 2.4. A function $F : [k]^3 \rightarrow [k]$ is 4-invertible if whenever the output value and two of its inputs from $\{a, b, c\}$ are known, then the third input can be determined up to a set of four values. That is, for each pair of values $(a, b) \in [k]^2$, the mapping $F(a, b, *) : [k] \rightarrow [k]$ is at most 4-to-1, and likewise for pairs (b, c) and (a, c) .

We expect that the lower bound should still hold even if every function in \vec{F} is fixed to be a particular function with nice properties, although we are not able to prove this at present. In particular, we conjecture that the lower bound still holds where for every v , $F_v(a, b, c) = a^3 + b^3 + c^3$ over the field $[k]$. On the other hand, if we take an associative function such as $F_v(a, b, c) = a^3 \cdot b^3 \cdot c^3$ again over the field $[k]$, then there is a very small branching program, since we can compute the root value by reading the elements one at a time and remembering the product so far. One thing that makes proving the lower bound difficult is not being able to properly isolate or take advantage of the differences between these functions over a finite field. For the rest of the paper, we will refer to nondeterministic semantic read-once branching programs as simply branching programs.

2.1 Black/White pebbling, A natural upper bound

In order to get some intuition, we first review the matching upper bound. As mentioned earlier, the upper bound mimics the optimal black/white pebbling strategy for a tree [10]. A black pebble placement on a node v corresponds to remembering the value in $[k]$ labelling that node, and a white

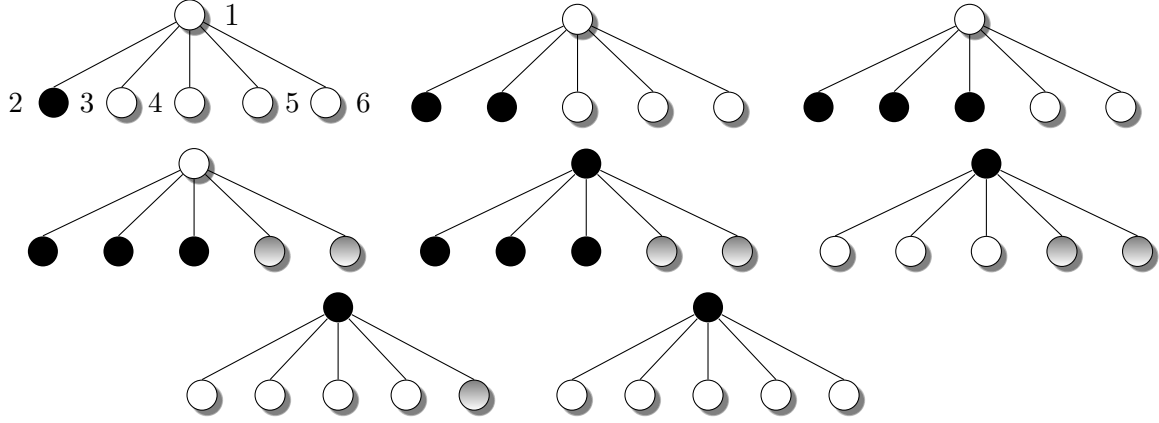


Figure 1: This figure describes a black/white pebbling for a d -ary tree T of height h at $d=5$. We start by pebbling the height $h-1$ subtrees rooted at nodes 2,3 and 4. Then we proceed to the second half of children and guess the value that subtrees at node 5 and 6 would evaluate to. Now we can pebble the root node 1 and remove the black pebbles. The white pebble or guess at node 5 can now be verified and then the same is done subsequently for node 6.

pebble on v corresponds to nondeterministically guessing v 's value (which must later be verified.) The goal is to start with no pebbles on the tree, and end up with one black pebble on the root (and no other pebbles). The legal moves in a black/white pebbling game are:

1. A black pebble can be placed at any leaf.
2. If all children of node v are pebbled (black or white), place a black pebble at v and remove any black pebbles at the children. (When all children are pebbled, a black pebble on a child of v can be slid to v .)
3. Remove a black pebble at any time.
4. A white pebble can be placed at any node at any time.
5. A white pebble can be removed from v if v is a leaf or if all of v 's children are pebbled. (When all children but one are pebbled, the white pebble on v can be slid to the unpebbled child.)

Lemma 2.5. *Black pebbling the root of a d -ary tree of height h can be done with $(d-1)(h-1)+1$ pebbles. With both black and white pebbles, only $\lceil \frac{1}{2}(d-1)h + 1 \rceil$ pebbles are needed.*

Proof. We will assume that d is odd; the case of d even is similar. With only black pebbles, recursively pebble $d-1$ of the d children of the root. Then use $d-1$ pebbles to remember these values as you use $(d-1)(h-2)+1$ more pebbles to pebble its d^{th} child for a total of $(d-1) + (d-1)(h-2)+1 = (d-1)(h-1)+1$ pebbles. Then pebble the root.

Now suppose white pebbles are also allowed (see Figure 1). Recursively pebble $\frac{1}{2}(d-1) + 1$ of the d children of the root. Then use $\frac{1}{2}(d-1)$ pebbles to remember these values as you use $\lceil \frac{1}{2}(d-1)(h-1) + 1 \rceil$ more pebbles to pebble its next child for a total of $\frac{1}{2}(d-1) + \frac{1}{2}(d-1)(h-1) + 1 = \frac{1}{2}(d-1)h + 1$ pebbles. Then use white pebbles to pebble the remaining $\frac{1}{2}(d-1)$ children of the root. Pebble the root and pick up the black pebbles from the children. Replacing the first of these whites requires $\frac{1}{2}(d-1)(h-1) + 1$ in addition to the $\frac{1}{2}(d-1)$ white ones, again for a total of $\frac{1}{2}(d-1)h + 1$. Note as a base case, when $h = 2$ and there is a root with d children, d pebbles are needed, no matter what the color. \square

Lemma 2.6. *A pebbling procedure with p black or white pebbles (and t time) translates to a layered nondeterministic branching program with tk^p states. If only black pebbles are used, the branching program is deterministic.*

Proof. On input $\vec{\xi}$ the branching program moves through a sequence of states $\beta_1, \beta_2, \dots, \beta_t$ where the state $\beta_{t'}$ corresponds to the pebbling configuration at time t' . Each layer of the branching program will have k^p states one for each possible assignment of values in $[k]$ to each of the pebbles. If a black pebble is placed on a leaf during the pebbling procedure, then the branching program queries this leaf. If all of the children of node v are pebbled, then the branching program knows their values v_1, v_2 and v_3 and hence can compute the value $f_v(v_1, v_2, v_3)$ of the node. Remembering this new value corresponds to placing a black pebble at v . Removing a black pebble corresponds to the branching program forgetting this computed value. If a white pebble is placed at v , then the branching program nondeterministically guesses the required value for this node. This white pebble cannot be removed until this value has been verified to be $f_v(v_1, v_2, v_3)$ using the values of its children that were either computed (black pebble) or also guessed (white). \square

3 Proof Overview

The crux of the argument is a compression argument, showing that from a small branching program for $Tree_{\vec{F}}$, we can encode \vec{F} with less bits than what is required information theoretically. We first review the simpler argument where \vec{F} are all invertible functions. (For every F , the value of two of the inputs and the output value completely determines the value of the third input.) The main argument (proven in Section 5) shows that given such a branching program, we can find a *label* $L = \langle P, v_{i*}, \vec{w}, x_{red,i*}, x_{white,i*}, X_{red}, X_{white}, S_{red}, S_{white} \rangle$ such that the following properties are satisfied:²

- (1) $P = v_h, v_h, \dots, v_1$ is a path in the ternary tree (defining the tree evaluation problem), where v_h is the root of the ternary tree, v_1 is a leaf vertex. v_{i*} is a special vertex along this path, where $x_{red,i*} \in X_{red}$ and $x_{white,i*} \in X_{white}$ are leaf variables in the subtree rooted at v_{i*} .
- (2) $X_{red}, X_{white} \subseteq \{x_1, \dots, x_n\}$ are (small) disjoint sets of leaf variables.
- (3) \vec{w} is an assignment to all of the leaf variables other than those in $X_{red} \cup X_{white}$

²We have chosen to use modified descriptions in this overview. We caution the reader that they do not exactly match their names in the proof.

- (4) $S_{red} \subset [k]^{|X_{red}|}$ consists of a set of r partial assignments to the variables of X_{red} such that the projection of these assignments onto the special variable $x_{red,i*}$ gives r distinct assignments. and similarly $S_{white} \subset [k]^{|X_{white}|}$ consists of a set of r partial assignments to the variables of X_{white} such that the projection of these assignments onto the special variable $x_{white,i*}$ gives r distinct assignments.
- (5) The embedded rectangle of r^2 total truth assignments $R = (\vec{w}, \rho, \sigma)$, $\rho \in S_{red}$, $\sigma \in S_{white}$ are all accepting truth assignments.

Given such a label, we prove two things. Let F_{i*} be the function associated with the special vertex v_{i*} along the path P . First, we show that knowing the label, plus knowing the value of all functions in \vec{F} other than F_{i*} , reveals a lot (roughly $O(r^2 \log k)$ bits) of information about F_{i*} , the missing function. The idea is roughly as follows. The r^2 total assignments in the rectangle R described in (5) gives rise to r^2 associated inputs $(a_i, b_j, c_{i,j})$ for F_{i*} where $i, j \in [r]$. These inputs are obtained by starting with an assignment $(\vec{w}, \rho, \sigma) \in R$ to the leaf variables $\{x_1, \dots, x_n\}$ and using the knowledge of \vec{F} , evaluating the assignment bottom-up until we obtain an assignment to the three children of v_{i*} . On the other hand, each of the r^2 assignments in R also gives *constraints* on the allowable *outputs* of F_{i*} on the inputs $(a_i, b_j, c_{i,j})$. The constraints are obtained because we know that each total assignment in R is an accepting input, and thus the value of the root vertex v_h on any of these inputs must lie in $[k^{1-\epsilon}]$. Since the functions in \vec{F} are all invertible, this propagates down the path P , so that for each of the r^2 inputs $(a_i, b_j, c_{i,j})$ to F_{i*} , we have a small set $C(i, j) \subseteq [k]$ of $k^{1-\epsilon}$ values such that $F_{i*}(a_i, b_j, c_{i,j})$ must lie in $C(i, j)$. Thus, overall, the label plus the information about all functions in \vec{F} other than F_{i*} reveals $(\log k - \log k^{1-\epsilon})r^2 = O(r^2 \log k)$ bits of information about F_{i*} .

Secondly, we show that the description length of the above label is less than what it should be – namely, it is much less than the number of bits that it reveals about F_{i*} , giving a contradiction. As is typical in lower bound arguments, the savings is due to the fact that R is an embedded rectangle. Fixing \vec{w} , R is a product of two sets each of size r . Therefore knowing \vec{w} , these inputs can be described with an additional $2r \log k$ bits rather than $r^2 \log k$ bits. (And the other information in the label does not overwhelm this savings that we achieved.)

We now give a high level overview of how we obtain a label L from a small branching program. What want to find a path P down the ternary tree and a state q in the branching program such that the subtrees of the path are sorted, with P taking the middle road. The subtrees hanging off the left of the path will be called “red” and those hanging off the right will be called “white”. The property that we require is that for a large set of accepting inputs (we only consider \vec{F} for which the set of accepted inputs is large) a reasonable number of the leaves in the red subtrees are read before state q , and a reasonable number of the leaves in the white subtrees are read after state q . We choose this path greedily, by taking the most popular path (with respect to the set of accepted inputs) that meets our requirements. From this property, we can obtain an embedded rectangle which is a large set of accepting inputs where the assignment to all variables outside of X_{red} and X_{white} is fixed to \vec{w} , and the assignments over the rest of the variables form a large product set $S_{red} \times S_{white}$ over X_{red} times X_{white} .

We then clean up the embedded rectangle so that it is square, and so that it is associated with one specific function F_{i*} that labels a particular vertex v_{i*} on the path P . More specifically, we show that there is one vertex v_{i*} along the tree path P that has a red variable $x_{red,i*} \in X_{red}$ and a white variable $x_{white,i*} \in X_{white}$ such that as we run over the r assignments in S_{red} , $x_{red,i*}$ takes on r distinct values, and similarly for $x_{white,i*}$. (See Figure 2).

Some complications arise when trying to carry out the above proof outline, making the actual proof more intricate. First, the compression argument requires that each \vec{F} has a lot of accepting instances, so we need to show that most random \vec{F} have this property. The more serious complication is the fact that we cannot easily count over random invertible functions, so instead we use functions that are almost invertible. More specifically \vec{F} is a vector of 4-invertible functions which means that for each $F \in \vec{F}$, knowing two of the inputs to F and the output value, there are at most four consistent values for the third input. We use a novel argument that allows us to count over 4-invertible functions (Section 6). Our compression argument sketched above is then adapted to handle the case of 4-invertible functions with a small quantitative loss. Namely when going down the path P to determine the constraints on the output of F_{i*} on an input $(a_i, b_j, c_{i,j}) \in R$, the number of allowable values for $F_{i*}(a_i, b_j, c_{i,j})$ will be $k^{1-\epsilon}$ at the root vertex, and by 4-invertibility, we will gain a factor of four for each subsequent function along the path. Since the path height is very small relative to r this will still give us adequate compression.

4 Most \vec{F} have a lot of accepting instances

Let $S_{yes} = \{\vec{\xi} \mid Tree_{\vec{F}}(\vec{\xi}) \in [k^{1-\epsilon}]\}$. That is, S_{yes} is the set of accepting inputs to $Tree_{\vec{F}}$. Let $Bad(\vec{F})$ be the event that the size of S_{yes} is significantly smaller than expected – in particular $|S_{yes}| \leq \frac{1}{6k^\epsilon} \cdot k^n$. Let \mathcal{F} be the uniform distribution over 4-invertible functions, and let $\vec{\mathcal{F}}$ be the uniform distribution over vectors of 4-invertible functions (one for each non-leaf vertex in the tree). Lemma 4.1 proves that $\Pr_{\vec{F}}[Bad(\vec{F})]$ is exponentially small, where \vec{F} is sampled from $\vec{\mathcal{F}}$.

Lemma 4.1. For $k > 2^{42h}$ and $\epsilon = \frac{9h}{\log k}$, $\Pr_{\vec{F}}[Bad(\vec{F})] \leq \frac{1}{10}$.

See section A in the Appendix for the proof. The above probability is in fact much smaller but the above bound suffices for our purpose.

5 Finding an Embedded Rectangle

This section proves that the accepted instances of $Tree_{\vec{F}}$ solvable by a small branching program contain a large embedded rectangle whenever $Bad(\vec{F})$ does not occur.

Parameters. The number of variables is $n = 3^{h-1}$ and each variable is from $[k]$. In what follows we will fix $r = \frac{2^{6h}}{\epsilon}$ and $\epsilon = \frac{9h}{\log k}$. The lower bound will hold for $s \leq \left(\frac{k}{n^{26} \log k}\right)^h$. For k sufficiently large ($k > 2^{42h}$), the lower bound is $\Omega(k/\log k)^h$.

Definition 5.1. For $\pi \subset \{1, \dots, n\}$, let x_π denote the set of variables $\{x_i \mid i \in \pi\}$. An *embedded rectangle* [2, 23] is defined by a 5-tuple $(\pi_{red}, \pi_{white}, A, B, \vec{w})$, where:

- (i) π_{red}, π_{white} are disjoint subsets of $\{1, \dots, n\}$,
- (ii) $A \subseteq [k]^{|\pi_{red}|}$ is a set of assignments to $x_{\pi_{red}}$ and $B \subseteq [k]^{|\pi_{white}|}$ is a set of assignments to $x_{\pi_{white}}$;
- (iii) $\vec{w} \in [k]^{n-|\pi_{red}|-|\pi_{white}|}$ is a fixed assignment to the remaining variables.

The assignments defined by the rectangle are all assignments $(\vec{\alpha}, \vec{\beta}, \vec{w})$ where $x_{\pi_{red}} = \vec{\alpha}$, $x_{\pi_{white}} = \vec{\beta}$ and the rest of the variables are assigned \vec{w} , where $\vec{\alpha} \in A$ and $\vec{\beta} \in B$.

5.1 Finding a rectangle over the leaves

In this section, we prove the following lemma, that shows the existence of a large embedded rectangle of accepting instances if the branching program solving $Tree_{\vec{F}}$ is small.

Lemma 5.2. *Let \mathcal{B} be a size s nondeterministic, semantic read-once BP over $\{x_1, \dots, x_n\}$ solving $Tree_{\vec{F}}$ for some \vec{F} such that $\neg \text{Bad}(\vec{F})$ holds. Let s, r be chosen as above. Then there exists an embedded rectangle $(\pi_{red}, \pi_{white}, A, B, \vec{w})$ such that:*

- 1. $|\pi_{red}| = |\pi_{white}| = h$,
- 2. $|A| \times |B| \geq \frac{k^{2h-\epsilon}}{s^{23h^2}}$,
- 3. \mathcal{B} accepts all inputs in the embedded rectangle.

In order to prove the above Lemma, we will need the following definitions.

Definition 5.3. Let $\vec{\xi}$ be an accepting input, and let $Comp_{\vec{\xi}}$ be an accepting computation path for $\vec{\xi}$. Since every variable is read exactly once, $Comp_{\vec{\xi}}$ defines a permutation Π of $\{1, \dots, n\}$. If q is a state that $Comp_{\vec{\xi}}$ passes through at time $t \in [n]$, the pair (Π, q) partitions the variables x_1, \dots, x_n into two sets, $Red(\Pi, q) = \{x_i \mid \Pi(i) \leq t\}$ and $White(\Pi, q) = \{x_j \mid \Pi(j) > t\}$. Intuitively, since the branching program reads the variables in the order given by Π (on input $\vec{\xi}$), then $Red(\Pi, q)$ are the variables that are read at or before reaching state q , and $White(\Pi, q)$ are the variables that are read after reaching state q .

Definition 5.4. A labelled path P down the ternary tree is a sequence of vertices v_h, \dots, v_1 that forms a path from the root to a leaf of the ternary input tree. For each vertex v_j of height j along the path, its three subtrees are labelled as follows: one of its subtrees is labelled *red* and is referred to as $Redtree(v_j)$, another is labelled *white* and is referred to as $Whitetree(v_j)$ and lastly, $Thirddtree(v_j)$ refers to the subtree with root v_{j-1} that continues along the path P . The root of $Redtree(v_j)$ will be called $redchild(v_j)$, the root of $Whitetree(v_j)$ will be called $whitechild(v_j)$, and the root of $Thirddtree(v_j)$ will be called $thirdchild(v_j)$.

Lemma 5.5. *Let $\vec{\xi}$ be an accepting input with computation path $\text{Comp}_{\vec{\xi}}$, where the ordering of variables read along $\text{Comp}_{\vec{\xi}}$ is given by permutation Π of $\{1, \dots, n\}$. Then there exists a state q and a labelled path $P = v_h, \dots, v_1$ in the ternary tree such that for all v_j in the path, $2 \leq j \leq h$ $\text{Redtree}(v_j)$ contains greater than 2^{j-2} variables in $\text{Red}(\Pi, q)$ and $\text{Whitetree}(v_j)$ contains greater than 2^{j-2} variables in $\text{White}(\Pi, q)$.*

Proof. We will prove the above lemma by (downwards) induction on the path length. At step j , $2 \leq j \leq h$, we will have constructed a labelled partial path v_h, v_{h-1}, \dots, v_j , an interval $[t_0(j), t_1(j)]$, and a partial coloring of the variables such that the following properties hold:

1. All variables x_i such that $\Pi(x_i) \leq t_0(j)$ will be Red and all variables x_i such that $\Pi(x_i) \geq t_1(j)$ will be White. (The remaining variables that are read between time step $t_0(j)$ and $t_1(j)$ are still uncolored.)
2. For each $v_{j'}$, $j \leq j' \leq h$, $\text{Redtree}(v_{j'})$ contains greater than $2^{j'-2}$ red variables, and $\text{Whitetree}(v_{j'})$ contains greater than $2^{j'-2}$ white variables.
3. The subtree of v_j that continues the path, $\text{Thirddtree}(v_j)$, has at most 2^{j-2} red variables and at most 2^{j-2} white variables.

Initially $j = h$, the path is empty, $t_0[h] = 1$ and $t_1[h] = n$. Thus the size of the interval is $n = 3^{h-1}$ and since no variables have been assigned to be red or white, the above properties trivially hold. For the inductive step, assume that we have constructed the partial path v_h, \dots, v_{j+1} . By the inductive hypothesis, the tree rooted at v_{j+1} contains at most 2^{j-1} red variables and at most 2^{j-1} white variables. Thus at most one subtree of v_{j+1} can contain greater than 2^{j-2} red variables. If one subtree of v_{j+1} does contain greater than 2^{j-2} red variables, then let this be $\text{Redtree}(v_{j+1})$. Otherwise, increase $t_0[j+1]$ until one of v_{j+1} 's three subtrees contains (for the first time) more than 2^{j-2} red variables and let this subtree be $\text{Redtree}(v_{j+1})$. Since each of v_{j+1} 's three subtrees has 3^{j-1} leaves and at most 2^{j-1} white variables, there are at least $3^{j-1} - 2^{j-1} \geq 2^{j-2}$ variables remaining in each subtree that are either uncolored or colored red, and thus the process is well-defined.

Next we work with the remaining two subtrees of v_{j+1} in order to define $\text{Whitetree}(v_{j+1})$. Again by the inductive hypothesis, the tree rooted at v_{j+1} contains at most 2^{j-1} white variables, and thus at most one subtree of the remaining two can contain greater than 2^{j-2} white variables. If one is found, then designate it as $\text{Whitetree}(v_{j+1})$, and otherwise, decrease $t_1[j+1]$ until one of v_{j+1} 's remaining two subtrees contains (for the first time) 2^{j-2} white variables and designate it as $\text{Whitetree}(v_{j+1})$. Again since each subtree has 3^{j-1} leaves and at most 2^{j-1} red variables, there are at least $3^{j-1} - 2^{j-1} \geq 2^{j-2}$ variables remaining in each of the two subtrees that are uncolored or colored white and thus the process is well-defined.

Let the remaining subtree of v_{j+1} be $\text{Thirddtree}(v_{j+1})$ and let the next vertex v_j in our path be $\text{thirdchild}(v_{j+1})$. By construction $\text{Thirddtree}(v_{j+1})$ contains at most 2^{j-2} red variables and at most this same number of white variables.

For the base case $j = 2$, by induction we will have reached a vertex v_2 with 3 child vertices, where at most one is colored red and at most one is colored white and thus the size of the interval

$[t_0[2], t_1[2]]$ is between one and three. Increase t_0 and then decrease t_1 so that v_2 has exactly one red vertex and two white vertices and let q be the state that $Comp_{\vec{\xi}}$ passes through as it reads the red child. \square

Proof. (of Lemma 5.2) Consider a nondeterministic semantic read-once branching program \mathcal{B} for $Tree_{\vec{F}}$. For each accepting input $\vec{\xi}$, fix one accepting path $Comp_{\vec{\xi}}$ in the branching program. Each of the n variables must be read in this path exactly once, and thus it defines a permutation $\Pi_{\vec{\xi}}$ of the n variables. Apply Lemma 5.5 for $\vec{\xi}$ (and corresponding permutation $\Pi_{\vec{\xi}}$) to obtain an associated labelled path $P_{\vec{\xi}}$ and state $q_{\vec{\xi}}$. Do this for all accepting inputs, and pick the pair P, q that occurs the most frequently. There are at most s possible values for q and at most 6^{h-1} possible labelled paths: $n = 3^{h-1}$ ending leaves of the path and then for each of the h vertices $v_{h'}$ along this path, we specify which of its subtrees are Red and White, for another 2^{h-1} choices. Let S be those accepting inputs that give rise to the popular pair P, q . Since there are at least $|S_{Yes}| > \frac{1}{6k^\epsilon} \cdot k^n$ accepting inputs, S is of size at least $\left(\frac{1}{6^h s k^\epsilon}\right) k^n$.

Next we will select one common red variable in each of the h Red subtrees, and one common white variable in each of the h White subtrees. Denoting the vertices of P by v_h, v_{h-1}, \dots, v_1 , we will select the Red and White variables iteratively for $j = h, h-1, \dots, 1$ as follows. Starting at $Redtree(v_j)$: for each $\vec{\xi} \in S$, by Lemma 5.5 at least 2^{j-2} of its 3^{j-1} variables are red, and thus there is one variable that is red in at least a $\frac{2^{j-2}}{3^{j-1}}$ fraction of S . Choose this variable, and update S to contain only those inputs in S where this variable is red. (That is, $\vec{\xi} \in S$ will stay in S if and only if the variable is read by $Comp_{\vec{\xi}}$ before reaching state q .) Do the same thing for $Whitetree(v_j)$. At the end, we will have selected for each j one variable that is red in $Redtree(v_j)$, and one variable that is white in $Whitetree(v_j)$, and a set of inputs S such that all h of the selected red variables (one per subtree) are read before reaching q and all h of the selected white variables are read after reaching q . Let π_{red} be vector of h indices corresponding to these h red variables, where $\pi_{red,j}$ is the index of the common red variable in $Redtree(v_j)$. and let π_{white} be the vector of h indices corresponding to these h white variables, where $\pi_{white,j}$ is the index of the common white variable in $Whitetree(v_j)$. The size of S after this process will be reduced by a factor of

$$\prod_{j \in [2, \dots, h]} \left(\frac{2^{j-2}}{3^{j-1}} \right)^2 \geq 2^{-2h} \cdot 1.5^{-h^2}.$$

Our final pruning of S is to fix a partial assignment, \vec{w} , to the remaining $n - 2h$ variables that have not been identified as red or white. There are k^{n-2h} choices here. Once again choose the most popular one. Overall, for $h \geq 2$ this gives

$$|S| \geq \frac{1}{k^\epsilon 6^h 2^{2h} 1.5^{h^2} s k^{n-2h}} k^n \geq \frac{k^{2h-\epsilon}}{s 1.5^{h^2+8h}} \geq \frac{k^{2h-\epsilon}}{s 2^{3h^2}}.$$

Let $S_{red} \subseteq [k]^{\pi_{red}}$ be the projection of S onto the coordinates of π_{red} , the red variables and let $S_{white} \subseteq [k]^{\pi_{white}}$ be the projection of S onto the coordinates of π_{white} , the white variables. Let all the other variables be set according to the vector \vec{w} . It is clear that this gives an embedded rectangle, $(\pi_{red}, \pi_{white}, S_{red}, S_{white}, \vec{w})$. We want to show that all assignments in the rectangle are accepted

by \mathcal{B} . To see this, consider an assignment $\vec{\alpha}\vec{\beta}\vec{w}$ in the embedded rectangle, where $\vec{\alpha} \in S_{red}$ is an assignment to $x_{\pi_{red}}$, and $\vec{\beta} \in S_{white}$ is an assignment to $x_{\pi_{white}}$, and \vec{w} is an assignment to the remaining variables. By definition $\vec{\alpha}$ is in the projection of S onto π_{red} , and thus there must be an assignment $\vec{\alpha}'\vec{\beta}'\vec{w} \in S$. Similarly, there must be an assignment $\vec{\alpha}'\vec{\beta}\vec{w} \in S$. Since these assignments are in S , the computation paths on each of them goes through q , and all variables $x_{\pi_{red}}$ are read before reaching q , and all variables $x_{\pi_{white}}$ are read after q . We want to show that $\vec{\alpha}\vec{\beta}\vec{w}$ is also an accepting input (in S). To see this, we follow the first half of the computation path of $\vec{\alpha}'\vec{\beta}'\vec{w}$ until we reach q , and then we follow the second half of the computation path of $\vec{\alpha}'\vec{\beta}\vec{w}$ after q . In this new spliced computation path, the variables $x_{\pi_{red}}$ are all read (and have value $\vec{\alpha}$) prior to reaching q , and the variables $x_{\pi_{white}}$ are all read after reaching q (and have value $\vec{\beta}$), and since all other variables have the same values on all paths, the new spliced computation path must be consistent and must be accepting. Therefore the input $\vec{\alpha}\vec{\beta}\vec{w}$ is in S and is an accepting input. \square

5.2 Refining the Rectangle

In this section, we refine the embedded rectangle given above, so that it will be a *square* r -by- r rectangle.

Definition 5.6. Let \mathcal{B} be a branching program for $Tree_{\vec{F}}$ for some \vec{F} such that $\neg Bad(\vec{F})$ holds, and let $(\pi_{red}, \pi_{white}, S_{red}, S_{white}, \vec{w})$ be the embedded rectangle guaranteed by Lemma 5.2. We recall the notation/concepts from the proof of Lemma 5.2:

1. Let $P = v_h, \dots, v_1$ be the common labelled path in the ternary tree, where $Redtree(v_i)$, $Whitetree(v_i)$ denotes the Red and White subtrees of v_i .
2. Let q be the common state in the branching program;
3. Let π_{red}, π_{white} be the indices of the red/white variables (h red variables altogether, one per Red subtree, and h white variables altogether, one per White subtree);
4. For all (accepting) inputs in the rectangle, all of the variables $x_{\pi_{red}}$ are read before q , and all variables $x_{\pi_{white}}$ are read after q .

We will now define a special kind of embedded rectangle that isolates a particular vertex v along the path P (which corresponds to a particular function F_v).

Definition 5.7. Let $P = v_h, \dots, v_1$ be the labelled path in the ternary tree, and let $r = 2^{6h}/\epsilon$. Let v_{i*} be a special vertex in the path P , where $\pi_{red,i*}$ is the index of the red variable in $Redtree(v_{i*})$, and $\pi_{white,i*}$ is the index of the white variable in $Whitetree(v_{i*})$. An embedded rectangle $(\pi_{red}, \pi_{white}, A, B, \vec{w})$ is *special* for v_{i*} if:

1. $|A| = |B| = r$;
2. The projection of A onto $x_{\pi_{red,i*}}$ has size r , and the projection of B onto $x_{\pi_{white,i*}}$ has size r .

Lemma 5.8. *Let \mathcal{B} be a size s branching program for $Tree_{\vec{F}}$ for some \vec{F} such that $\neg \text{Bad}(\vec{F})$ holds. Then (for our choice of parameters) there is an $i^* \in [h]$ and an embedded rectangle that is special for v_{i^*} .*

Proof. Let \mathcal{B} be a size s branching program for $Tree_{\vec{F}}$ and let $(\pi_{\text{red}}, \pi_{\text{white}}, S_{\text{red}}, S_{\text{white}}, \vec{w})$ be the embedded rectangle guaranteed by Lemma 5.2. For each $j \in [h]$, call v_j *red-good* if $|Proj(S_{\text{red}}, \pi_{\text{red},j})| \geq r$. That is, v_j is red-good if S_{red} projected to the red variable in $\text{Redtree}(v_j)$ has size at least r . Similarly, j is *white-good* if $|Proj(S_{\text{white}}, \pi_{\text{white},j})| \geq r$.

If there are l_{red} vertices that are red-good, then it is not hard to see that $|S_{\text{red}}| \leq (r-1)^{h-l_{\text{red}}} k^{l_{\text{red}}}$. To see this, every v_j that is not red-good can take on at most $r-1$ values, and the red-good ones could take on at most k values. If we similarly define l_{white} to be the number of vertices that are white-good, then similarly we have, $|S_{\text{white}}| \leq (r-1)^{h-l_{\text{white}}} k^{l_{\text{white}}}$.

We want to show that there must exist an i^* such that v_{i^*} is both red-good and white-good. If not, then $l_{\text{red}} + l_{\text{white}} \leq h$, and therefore $|S_{\text{red}} \times S_{\text{white}}| \leq (r-1)^h k^h < r^h k^h$. But on the other hand, Lemma 5.2 dictates that $|S_{\text{red}} \times S_{\text{white}}| \geq \frac{k^{2h-\epsilon}}{s^{2^{3h^2}}}$. This is a contradiction since by our choice of parameters ($r = 2^{6h}/\epsilon$, $\epsilon = 9h/\log k$, $s \leq \left(\frac{k}{n^{2^6 \log k}}\right)^h$, $n = 3^{h-1}$) we have:

$$\begin{aligned} \frac{k^{2h-\epsilon}}{s^{2^{3h^2}}} &\geq \frac{k^{2h-\epsilon}}{2^{3h^2}} \cdot \left(\frac{3^{26(h-1)} \log k}{k}\right)^h \geq k^{h-\epsilon} 2^{10h^2} (\log k)^h \\ &= k^h 2^{10h^2} \left(\frac{\log k}{2^9}\right)^h \quad \text{since } \epsilon = \frac{9h}{\log k}, \\ &= k^h \frac{2^{10h^2}}{2^{9h}} \left(\frac{2^h \log 9h}{\epsilon^h}\right) \geq \frac{k^h 2^{6h^2}}{\epsilon^h} = r^h k^h \quad \text{since } 4h + \log(9h) - 9 > 0, \forall h \geq 2 \end{aligned}$$

Let $i^* \in [h]$ denote the index such that vertex v_{i^*} along the path P is both red-good and white-good. Thus $\text{Redtree}(v_{i^*})$ contains the red variable indexed by π_{red,i^*} , and the projection of S_{red} to $x_{\pi_{\text{red},i^*}}$ has size at least r . Prune S_{red} to contain r assignments to $x_{\pi_{\text{red},i^*}}$, where we have exactly one assignment for each of the r distinct values for $x_{\pi_{\text{red},i^*}}$. Similarly, $\text{Whitetree}(v_{i^*})$ contains the white variable indexed by π_{white,i^*} , and the projection of S_{white} to $x_{\pi_{\text{white},i^*}}$ has size at least r . Prune S_{white} to contain r assignments to $x_{\pi_{\text{white},i^*}}$, where we have exactly one assignment for each of the r distinct values for $x_{\pi_{\text{white},i^*}}$. Because the pruned sets S_{red} and S_{white} will be important for our encoding, the following definition describes these sets more explicitly.

Definition 5.9. The (pruned) assignments in S_{red} consist of r partial assignment to $x_{\pi_{\text{red},i^*}}$. Each such assignment gives a distinct value for $x_{\pi_{\text{red},i^*}}$, with the values for the rest of the variables in $x_{\pi_{\text{red},i^*}}$ being completely determined by these. Let $\vec{\alpha}_i, i \in [r]$ denote the partial assignments in S_{red} . That is, for each $i \in [r]$, $\vec{\alpha}_i = \alpha_i^1, \dots, \alpha_i^h$ is a vector of h values given to $\text{redchild}(v_i)$ for all $i \in [h]$. Viewing the vectors $\vec{\alpha}_i, i \in [r]$ as an r -by- h matrix, the entries in column i^* ($\alpha_i^{i^*}$) run over the r distinct values given to $x_{\pi_{\text{red},i^*}}$. Similarly, S_{white} consists of r partial assignments to $x_{\pi_{\text{white},i^*}}$. Let $\vec{\beta}_i, i \in [r]$ denote the partial assignments in S_{white} . That is, for each $i \in [r]$, $\vec{\beta}_i = \beta_i^1, \dots, \beta_i^h$ is a vector of h values given to $\text{whitchild}(v_i)$ for all $i \in [h]$. Viewed as an r -by- h matrix, the entries in column i^* ($\beta_i^{i^*}$) run over the r distinct values given to $x_{\pi_{\text{white},i^*}}$.

It is clear from our construction that $(\pi_{red}, \pi_{white}, S_{red}, S_{white}, \vec{w})$ is an embedded rectangle that is accepted by \mathcal{B} and that is special for v_{i*} . □

6 The Encoding

In this section, \vec{F} is a vector of functions, one function each for each non-leaf vertex of the ternary tree, where each F in \vec{F} is a 4-invertible function from $[k]^3$ to $[k]$. Let \mathcal{F} denote the uniform distribution on 4-invertible functions. Let $H(\mathcal{F})$ refer to the entropy of \mathcal{F} . Assume that for each \vec{F} where every constituent function is 4-invertible, we have a size s branching program, $\mathcal{B}_{\vec{F}}$ for $Tree_{\vec{F}}$.

Our goal is to communicate a random \vec{F} using less bits than is information-theoretically possible (under the assumption of a small branching program for $Tree_{\vec{F}}$). If $Bad(\vec{F})$ is true, then we simply communicate \vec{F} using the full $H(\mathcal{F})$ bits that describe a uniformly random 4-invertible function at all the internal nodes of the tree. This requires $H(\vec{\mathcal{F}}) = (\text{number of internal nodes}) \times H(\mathcal{F})$ bits. If $Bad(\vec{F})$ is false, using Lemma 5.8, from $\mathcal{B}_{\vec{F}}$, we will define a vector of information, $L_{\vec{F}}$, which we call a *label* that will allow us to encode \vec{F} with fewer bits than is possible on average to get a contradiction. The following lemma describes how one can come up with $L_{\vec{F}}$.

Lemma 6.1. *Let \vec{F} be such that $Bad(\vec{F})$ is false, and assume that $Tree_{\vec{F}}$ has a small branching program $\mathcal{B}_{\vec{F}}$. Then there exists a vector $L_{\vec{F}}$ that can be specified with at most $4hr \log k = O(hr \log k)$ bits such that given \vec{F}_{-*} : the knowledge of all functions in \vec{F} except for F_* at one special node, $L_{\vec{F}}$ can be used to infer r'^2 inputs $(a_i, b_j, c_{i,j}) \in [k]^3$, $i, j \in [r']$ in the domain of function F_* , where $r' = \frac{r}{4^{i*}}$ and i^* is the height of node of F_* and corresponding to these inputs one can infer r'^2 sets of outputs $C(i, j) \subset [k]$, $i, j \in [r']$, specifying a small set of values such that $F_*(a_i, b_j, c_{i,j}) \in C(i, j)$. Moreover,*

$$Pr_{F \sim \mathcal{F}}[\forall i, j \in [r'] F(a_i, b_j, c_{i,j}) \in C(i, j)] \leq k^{-\frac{7}{9 \cdot 2^{4h}} \epsilon r^2}.$$

Proof. By Lemmas 5.2 and 5.8, there is a path P , a vertex $v_{i*} \in P$ and an embedded rectangle $(\pi_{red}, \pi_{white}, S_{red}, S_{white}, \vec{w})$ that is special for v_{i*} .

The vector $L_{\vec{F}}$ will consist of:

- (0) a description of \vec{w} ;
- (1) a description of the labelled path P ;
- (2) the index i^* of the special vertex along the path;
- (3) a vector $\langle \vec{\alpha}_1, \dots, \vec{\alpha}_r \rangle$ of r assignments as described in Definition 5.9.
- (4) the vector $\langle \vec{\beta}_1, \dots, \vec{\beta}_r \rangle$ of r assignments as described in Definition 5.9.

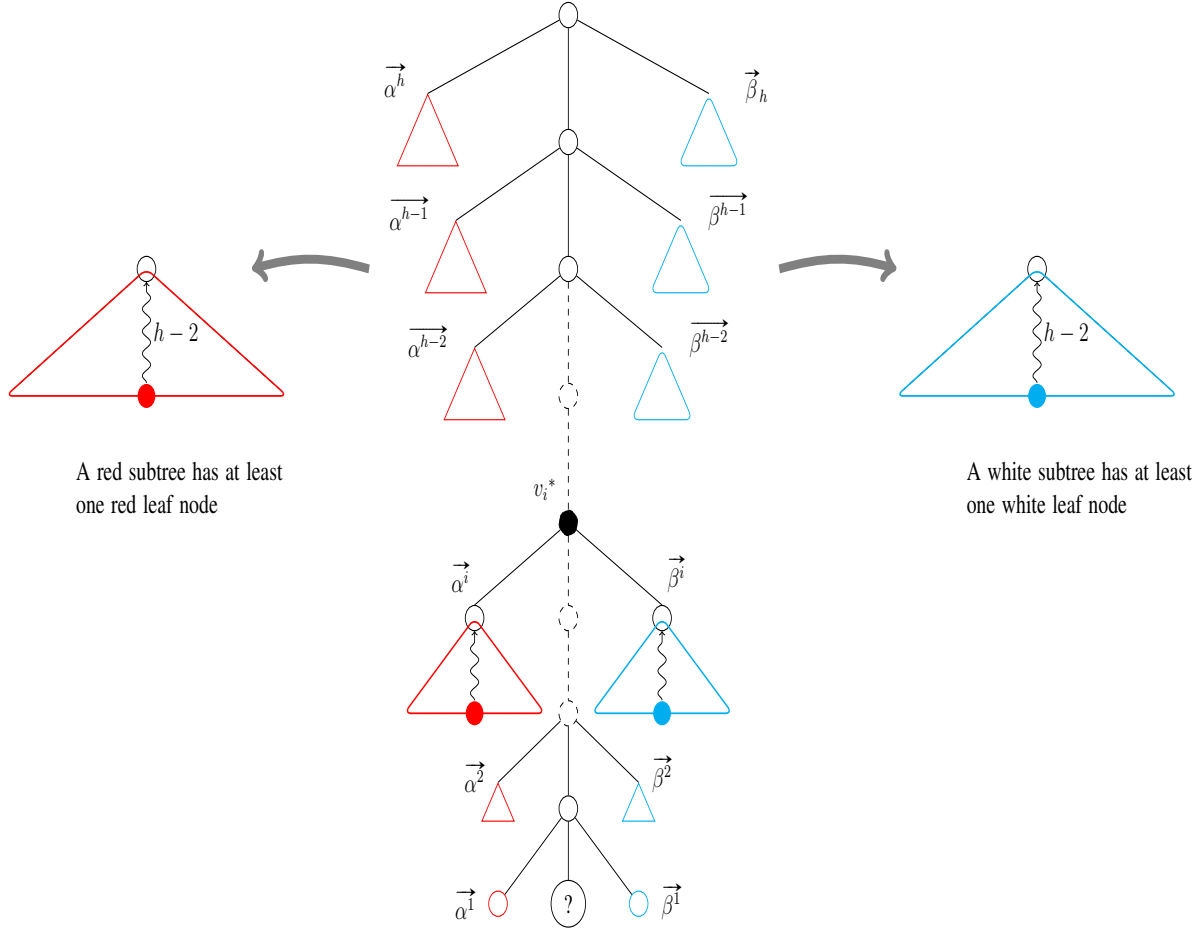
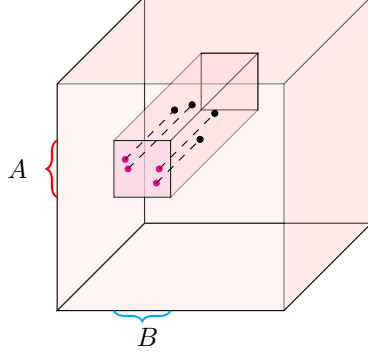


Figure 2: This figure depicts a label $L_{\vec{F}}$ associated with a problem instance $Tree_{\vec{F}}$ obtained as a consequence of having a *small* branching program $\mathcal{B}_{\vec{F}}$. A label as guaranteed by lemma 6.1 consists of a labelled path P reaching a leaf node, a special vertex v_{i*} along the path and a vector of r values each: $\vec{\alpha}$ and $\vec{\beta}$ respectively for the red and white sub trees at each node along the path. (We use blue for white here).

Figure 2 depicts a labelling that is induced by a small branching program. We first check that the length of $L_{\vec{F}}$ is $O(hr \log k)$. The length of (0) is $n \log k = 3^{h-1} \log k$. The length of (1) is $h \log 6$, since there are 6^h labelled paths (3^{h-1} different paths, and 2^h choices for the labels). The length of (2) is $\log h$. The length of (3) is $hr \log k$, and similarly the length of (4) is $hr \log k$. Thus the total length is at most $4hr \log k$.

Given the vector $L_{\vec{F}}$, the special function F_* will be the function associated with the vertex v_{i*} . For each $i, j \in [r]$, the corresponding input values $(a_i, b_j, c_{i,j})$ for F_* are obtained by a bottom-up evaluation of the subtree rooted at v_{i*} as follows. First, using $L_{\vec{F}}$ parts (3) and (4) we extract values for all red and white children of vertices in the path below v_{i*} . Secondly, using $L_{\vec{F}}$ part (0) we extract from \vec{w} values for all other leaf vertices of the subtree rooted at v_{i*} . Now using the knowledge of all internal functions corresponding to nodes below v_{i*} (given in \vec{F}_{-*}), we can



$$A, B \subset [k] \quad |A| = |B| = r' \quad |\{(a_i, b_j, c_{i,j}) | a_i \in A, b_j \in B\}| = r'^2$$

Figure 3: A subset in the input domain of F_{v^*} with product structure in two coordinates and over which the possible values taken by F_{v^*} has low entropy.

evaluate the subtree rooted at v_{i^*} in a bottom-up fashion in order to determine the values $(a_i, b_j, c_{i,j})$ for $redchild(v_{i^*})$, $whitechild(v_{i^*})$ and $thirdchild(v_{i^*})$.

Note that when we evaluate $redchild(v_{i^*})$, $whitechild(v_{i^*})$ and $thirdchild(v_{i^*})$ for each pair of $i, j \in [r]$ since all of the functions in \vec{F} are 4-invertible, we are guaranteed that there will be at least $r' = \frac{r}{4^{i^*}}$ distinct values taken by $redchild(v_{i^*})$ and similarly $r' = \frac{r}{4^{i^*}}$ distinct values taken by $whitechild(v_{i^*})$ resulting in at least r'^2 distinct inputs $(a_i, b_j, c_{i,j})$ with $i, j \in [r']$ in the domain of F_* .

We will now describe how to obtain the sets $C(i, j) \subset [k]$, $i, j \in [r']$, using $L_{\vec{F}}$ and the functions \vec{F}_{-*} . Fix an input $(a_i, b_j, c_{i,j})$. We want to determine the set $C(i, j)$ of possible values for $F_*(a_i, b_j, c_{i,j})$. Recall that for each $i, j \in [r']$, we know the value given to all inputs of the ternary tree. We want to work our way down the path P , starting at the root vertex v_h in order to determine $C(i, j)$. If the functions in \vec{F} were all invertible, then knowing that $(a_i, b_j, c_{i,j})$ is a yes input, this limits the number of possible values of the root vertex to the set $C(i, j)^h = [k^{1-\epsilon}]$. Working down the path, since we know the values of the red child and white child of v_h , this in turn gives us another set of at most $k^{1-\epsilon}$ values, $C(i, j)^{h-1}$ that v_{h-1} can have. We continue in this way down the path until we arrive at a set of at most $k^{1-\epsilon}$ values, $C(i, j)$ that v_{i^*} can take on.

However we are not working with invertible functions, but instead with 4-invertible functions. This can be handled by a simple modification of the above argument. Again we start at the root of the path v_h . As before, we know the values associated with the root is the set $C(i, j)^h = [k^{1-\epsilon}]$. At vertex $v_{h'}$, we define the set $C(i, j)^{h'}$ based on the previous set $C(i, j)^{h'+1}$. For a particular value $z \in C(i, j)^{h'+1}$, we know the value of $redchild(v_{h'})$, and $whitechild(v_{h'})$. This gives us values z, a, c . By the definition of $F_{v_{h'}}$ being 4-invertible, there are at most 4 values of b such that $z = F_{v_{h'}}(a, b, c)$. Thus we know the four possible values of b that can lead to z . Running over all z 's in $C(i, j)^{h'+1}$ defines the set $C(i, j)^{h'}$ which has size at most four times the size of $C(i, j)^{h'+1}$. Thus, the size of $C(i, j)^{i^*}$ is at most $4^{h-i^*} k^{1-\epsilon}$. We set $C(i, j)$ equal to $C(i, j)^{i^*}$.

Let \mathcal{F} be the uniform distribution over all 4-invertible functions from $[k]^3$ to $[k]$. Let E denote the event that for every (i, j) , $F(a_i, b_j, c_{i,j}) \in C(i, j)$. It is left to show that $Pr_{F \sim \mathcal{F}}[E] \leq$

$k^{-\frac{7}{9}\epsilon r^2 2^{-4h}}$. Let \mathcal{F}' be the uniform distribution over *all* functions from $[k]^3$ to $[k/4]$. Lemma 6.2 below shows that $Pr_{F \sim \mathcal{F}}[E] \leq Pr_{F' \sim \mathcal{F}'}[E]$. Thus we have:

$$Pr_{F \sim \mathcal{F}}[E] \leq Pr_{F' \sim \mathcal{F}'}[E] = \left(\frac{|C(i, j)|}{k/4} \right)^{(r')^2} \leq (4 \cdot 4^{h-i*} \cdot k^{-\epsilon})^{(r/4^{i*})^2} \leq k^{-\frac{7}{9 \cdot 2^{4h}} \epsilon r^2}.$$

□

Proof. (of Theorem 2.3) We are now ready to complete the proof of our main theorem. Let $\vec{\mathcal{F}}$ be the uniform distribution over vectors \vec{F} of all 4-invertible functions from $[k]^3$ to $[k]$. We prove the theorem by showing that if for every \vec{F} , if $Tree_{\vec{F}}$ has a size s branching program where $s \leq \left(\frac{k}{n^{26} \log k} \right)^h$, then the expected number of bits required for encoding an \vec{F} sampled from the distribution $\vec{\mathcal{F}}$ is less than the minimum number of bits required, which is $3^{h-1} H(\mathcal{F})$, giving us the contradiction. Given \vec{F} , the encoding is as follows.

- (1) If $\vec{F} \in Bad(\vec{F})$, encode each function using $H(\mathcal{F})$ bits, thus using $3^{h-1} H(\mathcal{F})$ bits over all the internal functions.
- (2) If $\vec{F} \notin Bad(\vec{F})$, encode as follows.
 - (2a) The first part is the description of $L_{\vec{F}}$.
 - (2b) The second part is an optimal encoding of all of \vec{F} except for F_* .
 - (2c) The third part is an optimal encoding of F_* . Recall that F_* is an element from the (uniform) distribution $(\mathcal{F} \mid E)$ where E denotes the event that for every (i, j) , $F(a_i, b_j, c_{i,j}) \in C(i, j)$.

Using this encoding, the decoding procedure is as follows. Whenever $Bad(\vec{F})$ holds, we use the information in (1) in order to recover $\vec{\mathcal{F}}$. Otherwise, if $\neg Bad(\vec{F})$ holds³, we proceed as follows. First we use the label $L_{\vec{F}}$ from (2a) in order to determine v_{i*} . Then we use label $L_{\vec{F}}$ from (2a) along with information about the rest of the functions from (2b) to find the special $(r')^2$ inputs $(a_i, b_j, c_{i,j})$, $i, j \in [r']$ to the function F_* . We also use the label $L_{\vec{F}}$ from (2a) and information from (2b) to determine the sets $C(i, j) \subset [k]$ such that $F_*(a_i, b_j, c_{i,j}) \in C(i, j)$ for all $i, j \in [r']$. We can then determine using the information from (2c) the values $F_*(a_i, b_j, c_{i,j})$ for all $i, j \in [r']$ (and also the remaining inputs in $[k]^3$).

We want to compare the savings of this encoding over the optimal one that uses $H(\vec{\mathcal{F}})$ bits. Let $p = Pr_{F \sim \mathcal{F}}[E]$. Then $1/p$ is equal to the number of 4-invertible functions divided by the number of 4-invertible functions satisfying E . Thus, when $\neg Bad(\vec{F})$ holds, the savings of our encoding in bits is $\log(1/p) - |L_{\vec{F}}|$, and therefore the overall savings in bits is

$$(1 - p_{Bad})[\log(1/p) - |L_{\vec{F}}|] \geq (1 - p_{Bad}) \left[\frac{7}{9 \cdot 2^{4h}} \epsilon r^2 \log k - 4hr \log k \right] = \left[\frac{7}{9 \cdot 2^{4h}} \epsilon r^2 - 4hr \right] (1 - p_{Bad}) \log k$$

³Astute reader might have observed that in order to recognize if $Bad(\vec{F})$ holds or not one needs to convey information, albeit just 1 bit. We end up saving a lot more so we ignore it.

since by Lemma 6.1, $|L_{\bar{F}}| \leq 4hr \log k$ and $p \leq k^{-\frac{7}{9}\epsilon r^2 2^{-4h}}$.

In the expression $[\frac{7}{9 \cdot 2^{4h}}\epsilon r^2 - 4hr]$, the quadratic dependence on r in the first term whereas only a linear dependence in the second allows us to choose $r = \frac{2^{6h}}{\epsilon}$, large enough so that we make savings. At $r = \frac{2^{6h}}{\epsilon}$, $[\frac{7}{9 \cdot 2^{4h}}\epsilon r^2 - 4hr] = r [\frac{7}{9 \cdot 2^{4h}}2^{6h} - 4h] > r \quad \forall h \geq 1$. Also, by Lemma 4.1 we know $p_{Bad} \leq \frac{1}{10}$ and since $k \geq 2^{42h}$ this implies $(1 - p_{Bad}) \log k > 1$. Thus our savings is greater than r bits, giving a contradiction. \square

Lemma 6.2. *Let \mathcal{F} be the uniform distribution over all 4-invertible functions from $[k]^3$ to $[k]$ and let \mathcal{F}' be the uniform distribution over all functions from $[k]^3$ to $[k/4]$. Fix r^2 inputs τ_i , $i \in [r^2]$, and let C_i be a corresponding subset of $[k]$, such that $|\cup_i C_i| \leq k/4$. E be the event that for all i , $F(\tau_i) \in C_i$. Then $\Pr_{F \sim \mathcal{F}}[E] \leq \Pr_{F' \sim \mathcal{F}'}[E]$.*

Proof. Since $|\cup_i C_i| \leq k/4$, we can assume without loss of generality that all of these values are in $[k/4]$. Let E_i denote the event that $F(\tau_i) \in C_i$, and let $E_{<i}$ denote the event that for all $j < i$, $F(\tau_j) \in C_j$. Then $\Pr_{F \sim \mathcal{F}}[E] = \prod_i \Pr_{F \sim \mathcal{F}}[E_i \mid E_{<i}]$. We will show that for any i , $\Pr_{F \sim \mathcal{F}}[E_i \mid E_{<i}] \leq \Pr_{F' \sim \mathcal{F}'}[E_i]$. Let σ specify the values of F for all tuples except for τ_i . Then $\Pr_{F \sim \mathcal{F}}[E_i \mid E_{<i}] \leq \max_{\sigma} \Pr_{F \sim \mathcal{F}}[E_i \mid \sigma]$. That is, the true probability is at most the probability where we fix all values except for the value of F on τ_i to the worst possible scenario.

We want to show that this probability only increases when the distribution switches from \mathcal{F} to \mathcal{F}' . But then note that under the distribution \mathcal{F}' , the values σ do not change the probability. Thus we want to show: $\Pr_{F \sim \mathcal{F}}[E_i \mid \sigma] \leq \Pr_{F' \sim \mathcal{F}'}[E_i \mid \sigma] \leq \Pr_{F' \sim \mathcal{F}'}[E_i]$.

To prove the first inequality, note that σ specifies all but one of the $[k]^3$ inputs to F . We visualize this as a k -by- k -by- k cube, where all entries (x, y, z) are filled in with a value in $[k]$ except for the one entry corresponding to τ_i . We want to get an upper bound on how many values we can choose for this last entry and still have a 4-invertible function. When choosing this last value, in order for F to be 4-invertible, we cannot choose one of the at most $k/4$ values that already appears four times along the “x” dimension, or one of the at most $k/4$ values that already appears four times in the “y” dimension, or $k/4$ times in the “z” dimension. This rules out at most $3k/4$ values, leaving at least $k/4$ possible values. Thus there is a set of at least $k/4$ values that can legally be filled in for $F(\tau_i)$ (even under the worst possible σ), and because \mathcal{F} is uniform on such functions, these completions all have the same probability. The event E_i is when $F(\tau_i)$ is chosen to be in C_i . This probability is at most that for the distribution \mathcal{F}' on all functions from $[k]^3$ to $[k/4]$. \square

7 Conclusion

It is open to prove lower bounds for function composition for the case of Boolean nondeterministic semantic read-once branching programs. In fact, it is open to prove lower bounds for the Boolean case for any explicit function. Another longstanding open problem is to break the Nečiporuk barrier of $n^2/\log^2 n$ for deterministic branching programs, and $n^{3/2}/\log n$ for nondeterministic branching programs. When g is the parity function, this bound is optimal. Lower bounds for $f \circ g$ for g equal to the element distinctness function (or even for the majority function) would be a significant breakthrough.

8 Acknowledgements

We would like to thank Stephen A. Cook for many helpful discussions.

A Proofs

Proof of lemma 4.1: For $k > 2^{42h}$ and $\epsilon = \frac{9h}{\log k}$, $\Pr_{\vec{F}}[Bad(\vec{F})] \leq \frac{1}{10}$.

Proof. We will choose a random \vec{F} somewhat indirectly as follows. First, we sample a random vector $\vec{F} \in \vec{\mathcal{F}}$. Secondly, we choose a random permutation Π of the values $[k]$, and let $\Pi(\vec{F})$ be the same as \vec{F} except that the root values have been permuted by Π . (This requires only changing the outputs of the function at the root.) Note that this distribution on \vec{F} is identical to the uniform distribution over $\vec{\mathcal{F}}$. It follows that $\Pr_{\vec{F}}[Bad(\vec{F})] = \Pr_{\langle \vec{F}, \Pi \rangle}[Bad(\Pi(\vec{F}))]$. We will consider the worst case value of \vec{F} in order to bound the above probability. Observe that

$$\Pr_{\langle \vec{F}, \Pi \rangle}[Bad(\Pi(\vec{F}))] \leq \text{Max}_{\vec{F}} \Pr_{\Pi}[Bad(\Pi(\vec{F})) \mid \vec{F}].$$

Fix such a worst case \vec{F} . For this \vec{F} , for each value $v \in [k]$ let q_v denote the fraction of leaf values $\vec{\xi}$ that give value v at the root. Note $\sum_v q_v = 1$ and $\text{Avg}_v q_v = \frac{1}{k}$.

Because the permutation Π is randomly chosen, $\Pi^{-1}([k^{1-\epsilon}])$ is a random subset of $[k]$ of size $k^{1-\epsilon}$. Therefore via linearity of expectation,

$$\text{Exp} \left(\frac{|S_{yes}|}{|\{\vec{\xi}\}|} \right) = \text{Exp} \left(\sum_{v \in \Pi^{-1}([k^{1-\epsilon}])} q_v \right) = \frac{k^{1-\epsilon}}{k} = k^{-\epsilon}.$$

We want to bound the probability that the size of S_{yes} is significantly smaller than its expected value of $k^{1-\epsilon}$. But first, the lemma below proves that $0 \leq q_v \leq \frac{4^{h-1}}{k}$.

Lemma A.1. $\forall v \in [k], q_v \leq \frac{4^{h-1}}{k}$.

Proof. Fix \vec{F} . Fix all of the leaf values as in $\vec{\xi}$, except for the left most leaf. Working down from the root, for any value v at the root one can see that there are at most 4^{h-1} values in $[k]$ for this left most leaf that can lead to value v at the root of \vec{F} . This is because each internal function is 4-invertible and for any fixed value of an internal node, given the value of two of its children(subtree evaluations) there are at most 4 possible values the other child can take. \square

We select a uniformly random set of size $k^{1-\epsilon}$ to be mapped to $[k^{1-\epsilon}]$ as follows. Flip a biased coin for each point 'v' in $[k]$ to be selected with probability $k^{-\epsilon}$. Given a vector of q_v describing the fraction of inputs that map to v , let Q_v be a vector of random variables associated with corresponding coin flips with each of them taking value q_v with probability $k^{-\epsilon}$ and 0 with the remaining $1 - k^{-\epsilon}$. The expected number of points selected is $k^{1-\epsilon}$. The experiment repeats until the number

of points selected is within some standard deviations say $c.k^{\frac{1-\epsilon}{2}}$ of the mean $k^{1-\epsilon}$. Let's first analyze the number of inputs selected corresponding to the points selected in the process without the size requirement on number of points.

We are interested in the fraction of inputs that get to be Yes inputs as a result of being selected during the coin flipping process. Let $Q_{Yes} = \sum_v Q_v$. So

$$E[Q_{Yes}] = \sum_v E[Q_v] = \sum_v q_v k^{-\epsilon} = k^{-\epsilon}. \quad (1)$$

In this experiment Q_v are independent (but not necessarily identically distributed) non-negative random variables. Consequently Q_{Yes} obeys the following concentration bound [6] around its mean

$$Prob \left[(E[Q_{Yes}] - \sum_v Q_v) \geq t \right] \leq e^{\left(\frac{-t^2}{2 \sum_v E[Q_v^2]} \right)} \quad (2)$$

Since by the regularity property from Lemma A.1 we have $q_v \leq \frac{4^{h-1}}{k}$ for all $v \in [k]$

$$\begin{aligned} \sum_v E[Q_v^2] &= \sum_v q_v^2 k^{-\epsilon} = k^{-\epsilon} \sum_v q_v^2 \leq k^{-\epsilon} \sum_v \left(\frac{4^{h-1}}{k} \right)^2 = k^{-\epsilon} k \cdot \left(\frac{4^{h-1}}{k} \right)^2 = \frac{4^{2h-2}}{k^{1+\epsilon}} \\ \implies Prob \left[(E[Q_{Yes}] - Q_{Yes}) \geq t \right] &\leq e^{\left(\frac{-t^2}{2 \sum_v E[Q_v^2]} \right)} \leq e^{\left(\frac{-t^2}{2 \left(\frac{4^{2h-2}}{k^{1+\epsilon}} \right)} \right)} = e^{\frac{-t^2 k^{1+\epsilon}}{2 \cdot 4^{2h-2}}} \end{aligned}$$

Consequently,

$$Prob [Q_{Yes} \leq E[Q_{Yes}] - t] \leq e^{\frac{-t^2 k^{1+\epsilon}}{2 \cdot 4^{2h-2}}} \quad (3)$$

Set $t = \frac{1}{2k^\epsilon}$ for the event $Bad' = [Q_{Yes} \leq E[Q_{Yes}] - t] = [Q_{Yes} \leq \frac{1}{2k^\epsilon}]$.

$$p_{Bad'} = Prob \left[Q_{Yes} \leq \frac{1}{2k^\epsilon} \right] \leq e^{\frac{-k^{1-\epsilon}}{8 \cdot 4^{2h-2}}} \quad (4)$$

Now consider the following transformed process in which the experiment repeats until number of points selected is within some fixed deviation g from the mean. Let the set of points be A . Depending on the count of number of points in A selected, if the count falls below $k^{1-\epsilon}$ a few more points are uniformly randomly selected from $[k] \setminus A$ to obtain a set of size $k^{1-\epsilon}$ and likewise if the number is larger than $k^{1-\epsilon}$ the required number of points are uniformly randomly discarded from the set. Clearly, this process doesn't discriminate against any point in $[k]$ and so generates a uniformly random subset of size exactly $k^{1-\epsilon}$ from $[k]$. Let call this set A'' , it shall be our final set of size $k^{1-\epsilon}$. Let p_{Bad} be the probability that that the fraction of inputs associated with the set of points in A'' is less than $\frac{1}{6k^\epsilon}$. For the intermediate set A let U be the event $[k^{1-\epsilon} - g \leq |A| \leq k^{1-\epsilon} + g]$. Then,

$$Prob [Bad' | U] = \frac{Prob(Bad' \cap U)}{Prob(U)} \leq \frac{Prob(Bad')}{Prob(U)} \quad (5)$$

Since $|A|$ is binomially distributed with $(n, p) = (k, k^{-\epsilon})$, seen as a sum of independent non-negative random variables, for a deviation $g \approx 2k^{\frac{1-\epsilon}{2}}$ we have the following concentration guaranteed by (2)

$$\text{Prob}(U) = \text{Prob}\left[k^{1-\epsilon} - 2k^{\frac{1-\epsilon}{2}} \leq |A| \leq k^{1-\epsilon} + 2k^{\frac{1-\epsilon}{2}}\right] \geq 0.8 \quad (6)$$

By (4) it follows that $\text{Prob}(\text{Bad}') \leq e^{\frac{-k^{1-\epsilon}}{8.4^{2h-2}}}$ and together with (6) and (5) this implies

$$\text{Prob}[\text{Bad}' \mid U] \leq \frac{5}{4} e^{\frac{-k^{1-\epsilon}}{8.4^{2h-2}}} \quad (7)$$

the chance that $S_{Y_{es}}^A$ is small is exponentially small. Now consider the transformation of A to A'' . Note that whenever new points are added to A or some points in A are discarded so as to obtain A'' i.e a uniformly random choice of a set of exact size $k^{1-\epsilon}$ the change from $S_{Y_{es}}^A$ to $S_{Y_{es}}^{A''}$ is at most $g \cdot \max_v q_v$. But by regularity property given by Lemma A.1, $q_v \leq \frac{4^h}{k}$. So $||S_{Y_{es}}^{A''} - S_{Y_{es}}^A|| \leq g \cdot \frac{4^h}{k} \approx k^{\frac{1-\epsilon}{2}} \frac{4^h}{k} = \frac{4^h}{k^{\frac{1-\epsilon}{2} + \epsilon}} = \left(\frac{4^h}{k^{\frac{1-\epsilon}{2}}}\right) \frac{1}{k^\epsilon} \leq \frac{1}{3k^\epsilon}$ for $k > 2^{42h}$ at $\epsilon = \frac{9h}{\log k}$. The resulting set A'' will then always have size at least $\frac{1}{2k^\epsilon} - \frac{1}{3k^\epsilon} = \frac{1}{6k^\epsilon}$ whenever $Q_{Y_{es}}^A > \frac{1}{2k^\epsilon}$. This implies $p_{\text{Bad}} = \text{Prob}\left[Q_{Y_{es}}^{A''} \leq \frac{1}{6k^\epsilon}\right] \leq \text{Prob}\left[Q_{Y_{es}}^A \leq \frac{1}{2k^\epsilon}\right] = \text{Prob}[\text{Bad}' \mid U]$ and hence $\leq \frac{5}{4} e^{\frac{-k^{1-\epsilon}}{8.4^{2h-2}}}$.

For $k > 2^{42h}$ and $\epsilon = \frac{9h}{\log k}$ it can be seen that $p_{\text{Bad}} \leq \frac{5}{4} e^{\frac{-k^{1-\epsilon}}{8.4^{2h-2}}} \leq \frac{5}{4} \left(\frac{1}{e}\right)^{\frac{2^{42h-9h}}{2^{4h-1}}} \leq \frac{1}{2^{28h}} \leq \frac{1}{10}$, $\forall h \geq 1$. \square

B Nečiporuk via Function Composition

Consider the composition of two boolean functions $f : \{0, 1\}^a \rightarrow \{0, 1\}$ and $g : \{0, 1\}^b \rightarrow \{0, 1\}$. Let f be a hard function in the sense that any non-deterministic branching program computing f requires size at least $2^{a/2}$. Such functions are guaranteed to exist by a simple counting argument. Fix g to be any function such that it does not take a constant value when all but any one of its b input bits are set.

Lemma B.1. *Any non-deterministic branching program solving $f \circ g$ has size at least $b2^{a/2}$.*

Proof. Let there be a non-deterministic branching program solving $f \circ g$ of size s . For each of the a copies of g in the composition $f \circ g$ pick the least queried input bit from amongst each group of b input bits that correspond to a single copy of g , then set all remaining $b - 1$ variables in this input group to any value and reconnect the outgoing edges amongst the remaining states appropriately. The resulting collapsed branching program has size at most $\frac{s}{b}$. But recall that g has the property that fixing $b - 1$ of its input bits doesn't make the function a constant. Thus the resulting collapsed branching program has to have size at least that required for computing f , that is $2^{a/2}$. Therefore the original non-deterministic branching program must have size at least $s \geq b2^{a/2}$. \square

Let $g = \oplus$ be the parity function on b bits. The input to $f \circ \oplus$ is the description of f , plus a vector of ab bits (the input to $f \circ \oplus$). The input length is $2^a + ab$. Setting $a = \log n$ and $b = \frac{n}{\log n}$, the input length is $2n$. By the above lemma, the size of a branching program required to solve the composition $f \circ \oplus$ is at least $b2^{a/2} = \left(\frac{n}{\log n}\right) \left(2^{\frac{\log n}{2}}\right) = \frac{n^{3/2}}{\log n}$. This lower bound is also known to be the best achievable by Nečiporuk as shown by Beame and McKenzie in [4].

By essentially similar means an $\Omega\left(\frac{n^2}{\log^2 n}\right)$ lower bound can be shown for deterministic branching programs. Consider deterministic branching programs solving $f \circ g$ where f is now a hard function in the sense that any deterministic branching program computing f requires size at least $\frac{2^a}{a}$ (once again such functions are guaranteed to exist by counting argument). Just as before, fix $g = \oplus$ to be the parity function (or any function that is not constant when all but any one of its input bits are set.) A similar argument shows that any branching program solving $f \circ \oplus$ requires size at least $b2^{a/2}$. Set $a = \log n$ and $b = \frac{n}{\log n}$ to obtain an $\Omega\left(\frac{n^2}{\log^2 n}\right)$ lower bound.

C The lower bound holds for most \vec{F}

We now argue that for most vectors of 4-invertible functions \vec{F} , $Tree_{\vec{F}}$ does not have a small branching program. We show that the probability that a uniformly randomly chosen \vec{F} has a small branching program is at most $p_{Bad} + \frac{1}{2^r} \leq \frac{1}{2^{27h}}$. First, let $\#L = 2^{|L_{\vec{F}}|}$ be the total number of labels. Recall that $|L_{\vec{F}}|$ is the number of bits needed to encode a label and that the number of bits saved in our alternate encoding from the proof of Theorem 2.3 is $(1 - p_{Bad})[\log(1/p) - |L_{\vec{F}}|] = (1 - p_{Bad}) \log\left(\frac{1}{p \cdot \#L}\right)$.

Note that for a uniformly randomly chosen \vec{F} the probability that it has a small branching program is at most the chance that $Bad(\vec{F})$ holds plus the chance that $Bad(\vec{F})$ doesn't hold and there exists a label L that is consistent with \vec{F} (in other words a label obtained via lemma 6.1 as a guaranteed consequence of \vec{F} having a small branching program).

$$\begin{aligned} \Pr_{\vec{F}}[\exists \text{ a small BP solving } Tree_{\vec{F}}] &\leq \Pr_{\vec{F}}[Bad(\vec{F}) \cup [\neg Bad(\vec{F}) \cap \exists \text{ a label } L \text{ consistent with } \vec{F}]] \\ &\leq p_{Bad} + \Pr_{\vec{F}}[\neg Bad(\vec{F}) \cap [\exists \text{ a label } L \text{ that is consistent with } \vec{F}]] \quad (\text{by Union bound}) \\ &\leq p_{Bad} + \Pr_{\vec{F}}[\exists \text{ a label } L \text{ that is consistent with } \vec{F}] \quad (\text{since } P(A \cap B) \leq \min\{P(A), P(B)\}) \\ &\leq p_{Bad} + \#L \cdot \max_L \Pr_{\vec{F}}[\text{label } L \text{ is consistent with } \vec{F}] \quad (\text{by Union bound}) \\ &\leq p_{Bad} + p \cdot \#L \end{aligned}$$

We have shown in the proof of theorem 2.3 that the number of bits saved in our alternate encoding is at least r . So, $(1 - p_{Bad}) \log\left(\frac{1}{p \cdot \#L}\right) \geq r \implies \frac{1}{p \cdot \#L} \geq 2^{r/(1-p_{Bad})} \geq 2^r \implies p \cdot \#L \leq \frac{1}{2^r}$. Consequently it follows that:

$$\Pr_{\vec{F}}[\exists \text{ a small BP solving } Tree_{\vec{F}}] \leq p_{Bad} + \frac{1}{2^r}$$

Now note that the proof of Lemma 4.1 (see Appendix A) actually shows that $p_{Bad} \leq 2^{-28h}$. As a result, $\Pr_{\vec{F}}[\exists \text{ a small BP solving } Tree_{\vec{F}}] \leq \frac{1}{2^{28h}} + \frac{1}{2^r} \leq \frac{1}{2^{27h}}$. (the last inequality follows since $r = \frac{2^{6h}}{\epsilon} = \frac{2^{6h} \log k}{9h} \geq 2^{6h+2}$). Thus we can conclude that most vectors of 4-invertible functions in fact do not have small branching programs.

References

- [1] M. Ajtai. A non-linear time lower bound for boolean branching programs. In *Proceedings 40th FOCS*, pages 60–70, 1999.
- [2] P. Beame, T.S. Jayram, and M. Saks. Time-space tradeoffs for branching programs. *J. Comput. Syst. Sci.*, 63(4):542–572, 2001.
- [3] P. Beame, M. Saks, X. Sun, and E. Vee. Time-space trade-off lower bounds for randomized computation of decision problems. *Journal of the ACM*, 50(2):154–195, 2003.
- [4] Paul Beame and Pierre McKenzie. A note on neciporuks method for nondeterministic branching programs. *Manuscript*, August, 2011.
- [5] Allan Borodin, A Razborov, and Roman Smolensky. On lower bounds for read-k-times branching programs. *Computational Complexity*, 3(1):1–18, 1993.
- [6] Stéphane Boucheron, Gábor Lugosi, and Pascal Massart. *Concentration inequalities: A nonasymptotic theory of independence*. Oxford university press, 2013.
- [7] Siu On Chan, James R. Lee, Prasad Raghavendra, and David Steurer. Approximate constraint satisfaction requires large LP relaxations. *J. ACM*, 63(4):34:1–34:22, 2016.
- [8] Arkadev Chattopadhyay, Michal Koucký, Bruno Loff, and Sagnik Mukhopadhyay. Simulation theorems via pseudorandom properties. *CoRR*, abs/1704.06807, 2017.
- [9] Stephen Cook, Jeff Edmonds, Venkatesh Medabalimi, and Toniann Pitassi. Lower Bounds for Nondeterministic Semantic Read-Once Branching Programs. In *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*, volume 55 of *Leibniz International Proceedings in Informatics (LIPIcs)*, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.
- [10] Stephen Cook, Pierre McKenzie, Dustin Wehr, Mark Braverman, and Rahul Santhanam. Pebbles and branching programs for tree evaluation. *ACM Transactions on Computation Theory (TOCT)*, 3(2):4, 2012.
- [11] 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 *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 295–304, 2016.
- [12] Scott Diehl and Dieter Van Melkebeek. Time-space lower bounds for the polynomial-time hierarchy on randomized machines. *SIAM Journal on Computing*, 36(3):563–594, 2006.
- [13] Irit Dinur and Or Meir. Toward the krw composition conjecture: Cubic formula lower bounds via communication complexity. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 50. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016.

- [14] Jeff Edmonds, Russell Impagliazzo, Steven Rudich, and Jiri Sgall. Communication complexity towards lower bounds on circuit depth. *Computational Complexity*, 10(3):210–246, 2001.
- [15] L. Fortnow. Nondeterministic polynomial time versus nondeterministic logarithmic space: Time space tradeoffs for satisfiability. In *Proceedings 12th Conference on Computational Complexity*, pages 52–60, 1997.
- [16] L. Fortnow and D. Van Melkebeek. Time-space tradeoffs for nondeterministic computation. In *Proceedings 15th Conference on Computational Complexity*, pages 2–13, 2000.
- [17] Lance Fortnow, Richard Lipton, Dieter Van Melkebeek, and Anastasios Viglas. Time-space lower bounds for satisfiability. *Journal of the ACM (JACM)*, 52(6):835–865, 2005.
- [18] Dmitry Gavinsky, Or Meir, Omri Weinstein, and Avi Wigderson. Toward better formula lower bounds: an information complexity approach to the krw composition conjecture. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, pages 213–222. ACM, 2014.
- [19] Mika Göös. Lower bounds for clique vs. independent set. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 1066–1076, 2015.
- [20] Mika Göös, Shachar Lovett, Raghu Meka, Thomas Watson, and David Zuckerman. Rectangles are nonnegative juntas. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 257–266, 2015.
- [21] Mika Göös, Toniann Pitassi, and Thomas Watson. Deterministic communication vs. partition number. In Venkatesan Guruswami, editor, *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 1077–1088. IEEE Computer Society, 2015.
- [22] Mika Göös, Toniann Pitassi, and Thomas Watson. Query-to-communication lifting for BPP. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA*, pages 132–143. IEEE Computer Society, 2017.
- [23] S. Jukna. A nondeterministic space-time tradeoff for linear codes. *Information Processing Letters*, 109(5):286–289, 2009.
- [24] Stasys Jukna. A note on read- k times branching programs. *Informatique théorique et applications*, 29(1):75–83, 1995.
- [25] Stasys P Jukna. The effect of null-chains on the complexity of contact schemes. In *Fundamentals of Computation Theory*, pages 246–256. Springer, 1989.

- [26] Mauricio Karchmer, Ran Raz, and Avi Wigderson. Super-logarithmic depth lower bounds via the direct sum in communication complexity. *computational complexity*, 5(3):191–204, 1995.
- [27] 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 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 590–603, 2017.
- [28] Matthias Krause, Christoph Meinel, and Stephan Waack. Separating the eraser turing machine classes le, nle, co-nle and pe. In *Mathematical Foundations of Computer Science 1988*, pages 405–413. Springer, 1988.
- [29] James R. Lee, Prasad Raghavendra, and David Steurer. Lower bounds on the size of semidefinite programming relaxations. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 567–576, 2015.
- [30] R. Lipton and A. Viglas. Time-space tradeoffs for sat. In *Proceedings 40th FOCS*, pages 459–464, 1999.
- [31] David Liu. Pebbling arguments for tree evaluation. *CoRR*, abs/1311.0293, 2013.
- [32] Edward I Nechiporuk. On a boolean function. *Doklady Akademii Nauk SSSR*, 169(4):765–+, 1966.
- [33] EA Okolnishnikova. On lower bounds for branching programs. *Siberian Advances in Mathematics*, 3(1):152–166, 1993.
- [34] Ran Raz and Pierre McKenzie. Separation of the monotone NC hierarchy. *Combinatorica*, 19(3):403–435, 1999.
- [35] Johan Håstad. The shrinkage exponent of de morgan formulas is 2. *SIAM Journal on Computing*, 27(1):48–64, 1998.
- [36] Avishay Tal. Shrinkage of de morgan formulae by spectral techniques. In *Foundations of Computer Science (FOCS), 2014 IEEE 55th Annual Symposium on*, pages 551–560. IEEE, 2014.
- [37] Ryan Williams. Better time-space lower bounds for sat and related problems. In *Computational Complexity, 2005. Proceedings. Twentieth Annual IEEE Conference on*, pages 40–49. IEEE, 2005.
- [38] Xiaodi Wu, Penghui Yao, and Henry Yuen. Raz-mckenzie simulation with the inner product gadget. *Electronic Colloquium on Computational Complexity (ECCC)*, 2017.