
Article:
Sanyal, S. (2025) Randomized query composition and sabotage complexity. ACM Transactions on Computation Theory, 17 (4). pp. 1-19. ISSN: 1942-3454

https://doi.org/10.1145/3747851

Published: 28 November 2025
Online AM: 15 July 2025
Accepted: 20 June 2025
Revised: 16 May 2025
Received: 05 November 2024

RESEARCH-ARTICLE

Randomized query composition and sabotage complexity

SWAGATO SANYAL, The University of Sheffield, Sheffield, South Yorkshire, U.K.

Randomized query composition and sabotage complexity

SWAGATO SANYAL, Department of Computer Science and Engineering, IIT Kharagpur, Kharagpur, India and School of Computer Science, The University of Sheffield, Sheffield, United Kingdom of Great Britain and Northern Ireland

Let R_ϵ denote the randomized query complexity for error probability ϵ, and $R := R_{1/3}$. In this work, we investigate whether a perfect composition theorem $R(f \circ g^n) = \Omega(R(f) \cdot R(g))$ holds for a relation $f \subseteq \{0,1\}^n \times \mathcal{S}$ and a total inner function $g : \{0,1\}^m \to \{0,1\}$.

Ben-David and Kothari (ICALP 2016, TOC 2018) showed that $R(f \circ g^n) = \Omega(R(f) \cdot RS(g))$, where RS is the sabotage complexity, a measure that they defined. Their result holds even when f is any relation and g is any partial function. Ben-David and Kothari asked in their article whether Sabotage complexity RS is asymptotically equivalent to R for total Boolean functions. In this article, we show that RS is asymptotically at least the maximum distributional query complexity for any product distribution (i.e., probability distribution where the individual bits are independently distributed). In light of the minimax theorem, which states that R is the maximum distributional query complexity for any distribution, our result makes progress toward answering the composition question. We prove our result by introducing a novel complexity measure called R^{prod}, and proving it to be equivalent to RS up to a logarithmic factor.

How tight is our bound? We show that our bound is polynomially loose for the complete binary NAND tree function, for which RS is polynomially larger than the distributional query complexity for any product distribution. Our result also confirms that RS and R are asymptotically equivalent for this function. We thus also show a polynomial separation between R and the maximum distributional query complexity for any product distribution which, to our knowledge, was not known prior to our work.

CCS Concepts: • **Theory of computation → Probabilistic computation**; • **Mathematics of computing** → *Discrete mathematics;*

Additional Key Words and Phrases: Randomized decision tree, query complexity, analysis of boolean functions

ACM Reference Format:
Swagato Sanyal. 2025. Randomized query composition and sabotage complexity. *ACM Trans. Comput. Theory* 17, 4, Article 21 (November 2025), 19 pages. https://doi.org/10.1145/3747851

1 Introduction

A deterministic decision tree or a deterministic query algorithm for a relation $f \subseteq \{0,1\}^n \times S$ can query various bits of an input bit string $x = (x_1, \ldots, x_n)$ in an adaptive fashion, with the goal of outputting an $s \in S$ such that $(x, s) \in f$. The complexity of a deterministic decision tree is the

Swagato Sanyal is currently affiliated with the University of Sheffield, UK. This work was carried out when the author was affiliated with IIT Kharagpur, and was partially supported by an ISIRD grant by Sponsored Research and Industrial Consultancy, IIT Kharagpur.
Author's Contact Information: Swagato Sanyal, Department of Computer Science and Engineering, IIT Kharagpur, Kharagpur, WB, India and School of Computer Science, The University of Sheffield, Sheffield, England, United Kingdom of Great Britain and Northern Ireland; e-mail: swagato.sanyal@sheffield.ac.uk.

ACM 1942-3454/2025/11-ART21
https://doi.org/10.1145/3747851

number of variables that it queries in the worst case. A randomized decision tree or a randomized query algorithm is a probability distribution over deterministic decision trees. The complexity of a randomized decision tree is the maximum complexity of any deterministic decision tree in its support. A deterministic decision tree that, for every x, outputs an s such that $(x, s) \in f$ is called a deterministic decision tree computing f. The deterministic query complexity or the deterministic decision tree complexity of f, denoted by $\mathsf{D}(f)$, is the least complexity of any deterministic decision tree computing f. The randomized query complexity or the randomized decision tree complexity of f for error ϵ, denoted by $\mathsf{R}_\epsilon(f)$, is the least complexity of any randomized decision tree $\mathcal{R}$ with the property that, for every input x, a deterministic decision tree sampled from $\mathcal{R}$ outputs s such that $(x, s) \in f$ with probability (over the randomness of $\mathcal{R}$) at least $1 - \epsilon$. For a probability distribution μ over the domain of f, the distributional query complexity of f with respect to μ and for error ϵ, denoted by $\mathsf{D}^\mu_\epsilon(f)$, is the least complexity of any deterministic decision tree that, for a random input x sampled from μ, fails to output an s such that $(x, s) \in f$ with probability at most ϵ. Define $\mathsf{R}(f) := \mathsf{R}_{1/3}(f)$ and $\mathsf{D}^\mu(f) := \mathsf{D}^\mu_{1/3}(f)$. μ is called a "product distribution" if the bits of the input string x sampled from μ are independent. We refer the reader to the excellent survey by Buhrman and de Wolf [6] for an introduction to the various decision tree models.

For a total Boolean function $g : \{0,1\}^m \to \{0,1\}$, the composition $f \circ g^n$ is the relation comprising all pairs $((x_1, \ldots, x_n), s) \in (\{0,1\}^m)^n$ such that $((g(x_1), \ldots, g(x_n)), s) \in f$.

It is easy to see that $\mathsf{D}(f \circ g^n) \leq \mathsf{D}(f) \cdot \mathsf{D}(g)$; a decision tree for $f \circ g^n$ may be constructed simply by simulating an optimal decision tree of f, and serving each query that it makes by solving the corresponding copy of g using an optimal tree of g. For randomized query algorithms, a similar idea works out, albeit with some additional work to handle errors, to show that $\mathsf{R}(f \circ g^n) = O(\mathsf{R}(f) \cdot \mathsf{R}(g) \cdot \log \mathsf{R}(f))$.

Composition questions ask whether the aforementioned upper bounds on the complexity of $f \circ g^n$ are asymptotically optimal. These are fundamental questions about the structure of optimal algorithms for $f \circ g^n$, and have received considerable attention in research.

It is known from the works of Montenaro [13] and Tal [20] that $\mathsf{D}(f \circ g^n) = \mathsf{D}(f) \cdot \mathsf{D}(g)$. Thus, the composition question for deterministic query complexity has been completely answered. On the contrary, in spite of extensive research, a complete answer to the composition question for randomized query complexity is still lacking.

1.1 Past Works on Randomized Query Composition

Past works dealt with a more general composition question for randomized query complexity, where the inner function g is allowed to be partial. The definition of $f \circ g^n$ and the aforementioned upper bounds on $\mathsf{D}(f \circ g^n)$ and $\mathsf{R}(f \circ g^n)$ can be accordingly generalized; we are omitting the details in this article. In 2015, Ben-David and Kothari [5] defined the sabotage complexity measure $\mathsf{RS}(g)$ of a partial Boolean function g. They showed that $\mathsf{R}(f \circ g^n) = \Omega(\mathsf{R}(f) \cdot \mathsf{RS}(g))$. They further showed that for total g, $\mathsf{RS}(g) = \widetilde{\Omega}(\sqrt{\mathsf{R}(g)})$, implying $\mathsf{R}(f \circ g^n) = \widetilde{\Omega}(\mathsf{R}(f) \cdot \sqrt{\mathsf{R}(g)})$. In 2019, Gavinsky, Lee, Santha and Sanyal [8] introduced the max-conflict complexity $\overline{\chi}(g)$ and showed that $\mathsf{R}(f \circ g^n) = \Omega(\mathsf{R}(f) \cdot \overline{\chi}(g))$. They further showed that even for partial functions g, $\overline{\chi}(g) = \Omega(\sqrt{\mathsf{R}(g)})$, implying $\mathsf{R}(f \circ g^n) = \Omega(\mathsf{R}(f) \cdot \sqrt{\mathsf{R}(g)})$. Moreover, they showed that for all partial functions g, $\overline{\chi(g)} = \Omega(\mathsf{RS}(g))$. They also demonstrated unbounded separation between $\overline{\chi}(g)$ and $\mathsf{RS}(g)$ for a partial g. In 2022, Ben-David, Blais, Göös and Maystre [4] introduced the linearized complexity measure $\mathsf{LR}(g)$. They showed that for any partial g, $\mathsf{R}(f \circ g^n) = \Omega(\mathsf{R}(f) \cdot \mathsf{LR}(g))$, and that LR is an asymptotically largest measure M for which the statement $\mathsf{R}(f \circ g^n) = \Omega(\mathsf{R}(f) \cdot \mathsf{M}(g))$ holds. They also demonstrated polynomial separation between $\mathsf{LR}(g)$ and $\overline{\chi}(g)$ for a partial g.

A different line of work has focused on proving bounds on $\mathsf{R}(f \circ g^n)$ of the form $\Omega(\mathsf{M}(f) \cdot \mathsf{R}(g))$ for some complexity measure M [1, 2, 9]. In 2020 Ben-David and Blais [3] defined the noisy query complexity noisyR and showed that noisyR is an asymptotically largest measure M for which the statement $\mathsf{R}(f \circ g^n) = \Omega(\mathsf{M}(f) \cdot \mathsf{R}(g))$ holds. In 2023, Chakraborty et al. [7] showed that for the special case when $\mathsf{R}(f) = \Theta(n)$, a near-perfect randomized query composition theorem $\mathsf{R}(f \circ g^n) = \widetilde{\Omega}(\mathsf{R}(f) \cdot \mathsf{R}(g))$ holds.

1.2 Our Results

This work investigates the possibility of a perfect randomized query composition theorem $\mathsf{R}(f \circ g^n) = \Omega(\mathsf{R}(f) \cdot \mathsf{R}(g))$ when g is a total function. As discussed in the preceding section, past works have introduced measures RS, $\overline{\chi}$ and LR that are asymptotically non-decreasing in the above order. As discussed before, we also know that any two of them can be asymptotically separated. However, the Boolean functions that witness these separations are all partial, and to the best of our knowledge, no separation between these measures is known for total functions. Do any or all of these measures coincide with R for total functions?

Ben-David and Kothari asked in their article whether $\mathsf{RS}(g) = \Theta(\mathsf{R}(g))$ for total g.

Question 1.1. *Does the relation* $\mathsf{RS}(g) = \Theta(\mathsf{R}(g))$ *hold for every total Boolean function* g*?*

In light of their composition theorem $\mathsf{R}(f \circ g^n) = \Omega(\mathsf{R}(f) \cdot \mathsf{RS}(g))$ mentioned before, an affirmative answer to Question 1.1 implies a perfect composition theorem for randomized query complexity.

Informally, the sabotage complexity captures the minimum number of randomized queries required to distinguish any pair of input strings on which the function values differ (see Section 2.4 for a formal definition). In a little more concrete terms, the sabotage complexity of a Boolean function is the least complexity of a randomized decision tree which, for every input bit strings $x \in f^{-1}(0), y \in f^{-1}(1)$, when run on x (equivalently, on y), queries some index where x and y differ with high probability. Question 1.1 asks whether this task is at least as hard as classifying an input instance, i.e., deciding the value of the function on it. Besides its importance in relation to the composition question discussed before, this is a natural and fundamental structural question in its own right.

In this article, we study Question 1.1, make progress toward answering it and uncover interesting structural insights that we believe will be useful to answer the composition question. Our first result is that for any total g, $\mathsf{RS}(g)$ is, up to a logarithmic factor, at least the maximum distributional query complexity of g for any product distribution. Let PROD be the set of all product distributions over the domain $\{0,1\}^m$ of g. Define $\mathsf{D}^{\mathsf{prod}}(g) := \max_{\mu \in \mathsf{PROD}} \{\mathsf{D}^{\mu}(g)\}$.

Theorem 1.2. *For any total function* $g : \{0,1\}^m \to \{0,1\}$,

$$\mathsf{RS}(g) = \widetilde{\Omega}\left(\mathsf{D}^{\mathsf{prod}}(g)\right).$$

The minimax theorem (Fact 2.2) states that $\mathsf{R}(g) = \max_{\mu} D^{\mu}(g)$, where the maximum is over all probability distributions over the domain of g. In this light Theorem 1.2 makes progress toward answering Question 1.1. An additional motivation for our first result is that product distributions comprise a natural class of distributions that has received significant attention in Boolean function complexity research [10–12, 18]. Theorem 1.2 shows that distinguishing pairs of inputs with differing function values is at least as hard as deciding the function on every possible product distribution (potentially with a different query algorithm for each distribution).

We prove Theorem 1.2 by introducing a new complexity measure $\mathsf{R}^{\mathsf{prod}}_{\epsilon}$. Informally speaking, $\mathsf{R}^{\mathsf{prod}}_{\epsilon}(g)$ is the minimum complexity of any randomized decision tree with unlabeled leaves with the property that, for every product distribution μ over the inputs, the average bias of its leaves is at

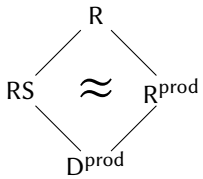
least $((1-\epsilon)-\epsilon)/2 = 1/2-\epsilon$. Define $\mathsf{R}^{\mathrm{prod}}(g) := \mathsf{R}^{\mathrm{prod}}_{1/3}(g)$. See Section 2.3 for formal definitions. It follows by standard arguments that $\mathsf{R}^{\mathrm{prod}}(g) = \Omega(\mathsf{D}^{\mathrm{prod}}(g))$ (see Claim 2.4). We show that RS is characterized by $\mathsf{R}^{\mathrm{prod}}$ up to a logarithmic factor.

Theorem 1.3. *For all total functions* $g : \{0,1\}^m \to \{0,1\}$,

(1) $\mathsf{RS}(g) = O(\mathsf{R}^{\mathrm{prod}}(g))$, *and*
(2) $\mathsf{RS}(g) = \Omega(\mathsf{R}^{\mathrm{prod}}(g)/\log \mathsf{R}^{\mathrm{prod}}(g))$.

Theorem 1.2 follows immediately from Theorem 1.3(2) and the fact that $\mathsf{R}^{\mathrm{prod}}(g) = \Omega(\mathsf{D}^{\mathrm{prod}}(g))$ (Claim 2.4).

The following figure depicts a diagrammatic comparison among the measures R, RS, $\mathsf{R}^{\mathrm{prod}}$, and $\mathsf{D}^{\mathrm{prod}}$, all for constant errors.



Since any non-trivial product distribution (i.e., where no variable is fixed to a constant) is supported on all of $\{0,1\}^m$, $\mathsf{R}^{\mathrm{prod}}(g)$ and $\mathsf{D}^{\mathrm{prod}}(g)$ are well-defined only for total functions g. The proof of Theorem 1.3 (that goes via Lemma 1.6 discussed later) makes important use of the totality of g. We hope that the measure $\mathsf{R}^{\mathrm{prod}}$, the characterization of RS presented in Theorem 1.3, and the insights acquired in our proof techniques, specially pertaining to ways of exploiting totality, will be useful in future research.

In light of Theorem 1.3 the question whether $\mathsf{R}(g) = \Theta(\mathsf{RS}(g))$ for total functions g translates to the question whether $\mathsf{R}(g) = \widetilde{O}(\mathsf{R}^{\mathrm{prod}}(g))$. We generalize this question for every error ϵ.

Question 1.4. *Is it true that for every total function* $g : \{0,1\}^m \to \{0,1\}$ *and* $\epsilon : \mathbb{N} \to (0,1/2)$, $\mathsf{R}_{\epsilon(m)}(g) = \widetilde{O}(\mathsf{R}^{\mathrm{prod}}_{\epsilon(m)}(g))$?

From the work of Ben-David, Blais, Göös and Maystre [4] it follows that for any error parameter ϵ, the linearized complexity measure $\mathsf{LR}(g)$ of g is bounded above by $O\left(\frac{1}{\epsilon}\cdot \mathsf{R}_{\frac{1}{2}-\epsilon}(g)\right)$. As discussed before, they also show that LR is the largest measure M for which the statement $\mathsf{R}(f\circ g) = \Omega(\mathsf{R}(f)\mathsf{M}(g))$ holds for all relations f and partial functions g. We thus have that for a perfect composition theorem $\mathsf{R}(f\circ g) = \Omega(\mathsf{R}(f)\cdot\mathsf{R}(g))$ to hold for any relation f and any total Boolean function g, a necessary condition is that $\mathsf{R}(g) = O(\frac{1}{\epsilon}\cdot \mathsf{R}_{\frac{1}{2}-\epsilon}(g))$ holds for any total Boolean function g. In light of Question 1.4, we may ask if $\mathsf{R}^{\mathrm{prod}}_{\epsilon}$ admits a similar error reduction. Our next result answers this question in the affirmative (up to a logarithmic factor).

Theorem 1.5. *For every total function* $g : \{0,1\}^m \in \{0,1\}$ *and* $\epsilon : \mathbb{N} \to (0,1/2)$,

$$\mathsf{R}^{\mathrm{prod}}(g) = \widetilde{O}\left(\frac{1}{\epsilon(m)}\cdot \mathsf{R}^{\mathrm{prod}}_{\frac{1}{2}-\epsilon(m)}(g)\right).$$

We remark here that from the definition of $\mathsf{R}^{\mathrm{prod}}_{\epsilon}$ it is not immediately clear whether it admits any error-reduction at all.

To prove Theorems 1.3 and 1.5, we define a version of sabotage complexity with errors, which we denote by RS_{ϵ}. Informally, $\mathsf{RS}_{\epsilon}(g)$ is the minimum number of randomized queries required

to distinguish each pair of inputs with different function values with probability at least $1-\epsilon$ (see Section 2.4 for a formal definition). Let $s(g)$ denote the sensitivity of g (see Section 2.2 for a formal definition). The following lemma constitutes the technical core of our proofs of Theorems 1.3 and 1.5.

Lemma 1.6. *For all total Boolean functions $g : \{0,1\}^m \to \{0,1\}$, and $\epsilon : \mathbb{N} \to (0,1/2)$,*
(1) $\mathrm{R}^{\mathrm{prod}}(g) = O(\frac{1}{\epsilon(m)} \cdot \mathrm{RS}_{1-\epsilon(m)}(g) \log s(g))$, *and*
(2) $\mathrm{RS}_{1-2\epsilon(m)}(g) \leq \mathrm{R}^{\mathrm{prod}}_{\frac{1}{2}-\epsilon(m)}(g)$.

Is the bound in Theorem 1.2 tight? Our next result gives a negative answer to this question. We show that for the complete binary NAND tree function (to be defined shortly), RS and $\mathrm{D}^{\mathrm{prod}}$ are polynomially separated. Consider a complete binary tree of depth d. Each leaf is labeled by a distinct Boolean variable. Each internal node is a binary NAND gate. For each input, the evaluation of this Boolean formula is the output of the complete binary NAND tree function, which we denote by g_d.

Theorem 1.7. $\mathrm{D}^{\mathrm{prod}}(g_d) = O(\mathrm{RS}(g_d)^{1-\Omega(1)})$.

Saks and Wigderson [16] showed that the zero-error randomized query complexity of g_d is $\Theta(\alpha^d)$ for $\alpha = \frac{1+\sqrt{33}}{4}$. Later Santha [17] showed that $\mathrm{R}(g_d) = \Theta(\alpha^d)$. We prove Theorem 1.7 in two parts. First, we show an upper bound of $O((\alpha - \Omega(1))^d)$ on $\mathrm{D}^{\mathrm{prod}}(g_d)$.

Lemma 1.8. *There exists a constant $\delta > 0$ such that $\mathrm{D}^{\mathrm{prod}}(g_d) = O((\alpha-\delta)^d)$.*

Works of Pearls [15] and Tarsi [21] showed that there exists a constant $\eta > 0$ such that for all distributions μ where each variable is set to 1 independently with some probability p, $\mathrm{D}^{\mu}(g_d) = O((\alpha-\eta)^d)$. In Lemma 1.8, we bound $\mathrm{D}^{\mu}(g_d)$ for any product distribution μ. Our bound is quantitatively weaker than those of Pearls and Tarsi, and we do not comment on its tightness.

Lemma 1.8 also gives an explicit polynomial separation between R and $\mathrm{D}^{\mathrm{prod}}$ which, to our knowledge, was not known prior to our work.

Next, we prove a tight lower bound on $\mathrm{RS}(g_d)$. As a by-product, our proof of the following lemma yields an alternative proof of the bound $\mathrm{R}(g_d) = \Omega(\alpha^d)$, originally proven by Santha [17].

Lemma 1.9. $\mathrm{RS}(g_d) = \Omega(\alpha^d)$.

Lemma 1.9, together with the upper bound by Saks and Wigderson, shows that $\mathrm{RS}(g_d) = \Theta(\mathrm{R}(g_d))$, which confirms an affirmative answer to Question 1.1 for the complete binary NAND tree function. From the composition theorem proven by Ben-David and Kothari, we thus have that

Corollary 1.10. *For all relations f, $\mathrm{R}(f \circ g_d^n) = \Theta(\mathrm{R}(f) \cdot \mathrm{R}(g_d))$.*

Lemmas 1.8 and 1.9 immediately imply Theorem 1.7.

1.3 Proof Ideas

In this section, we sketch the ideas and techniques that have gone into the proofs of our results. We begin with Lemma 1.6, from which Theorems 1.3 and 1.5 follow. We then discuss Lemmas 1.8 and 1.9, from which Theorem 1.7 as follows:

Lemma 1.6. Recall that RS captures the complexity of probabilistically distinguishing each pair of inputs with differing function values. We show that this task is equivalent to probabilistically distinguishing each pair of inputs *differing in exactly one position* with differing function values. Put another way, this task is equivalent to probabilistically reading each sensitive bit of each input (Lemma 3.1). This is a simple but key ingredient of the proofs of both parts of Lemma 1.6.

We first discuss part 2. Let a randomized algorithm $\mathcal{R}$ achieve $\mathsf{R}^{\mathsf{prod}}$ (we ignore the error parameter in this outline). We will show that $\mathcal{R}$ distinguishes any pair of inputs $x \in f^{-1}(0)$ and $y \in f^{-1}(1)$ differing in exactly one location, and hence by the above discussion achieves RS. Note that $\mu :=$ uniform distribution on $\{x, y\}$ is a product distribution. If a deterministic decision tree T sampled from $\mathcal{R}$, when run on x, does not query the lone index at which x differs from y, then x and y both land up in the same leaf of T. In that case, the leaves of T are going to be balanced with respect to μ, and hence the biases of the leaves will be 0. However, by our assumption about $\mathcal{R}$ and the definition of $\mathsf{R}^{\mathsf{prod}}$, the biases of the leaves of $\mathcal{R}$ with respect to μ are 'high' on an average. This implies that $\mathcal{R}$ distinguishes x and y with high probability. The details depend on the technical definition of $\mathsf{R}^{\mathsf{prod}}$ and may be found in the formal proof presented in Section 3.

Now we turn to part 1. This step needs arguments involving sensitivity and influence of Boolean functions, which are defined and discussed in Section 2.2. Using Lemma 3.1 (informally discussed in the first paragraph of this outline), by a sequence of arguments involving standard error-reduction, we infer that there is a randomized tree $\mathcal{R}$ of complexity $O(\frac{1}{\epsilon(n)} \cdot \mathsf{RS}_{1-\epsilon(n)}(g) \log s(g))$ that, for every input, with probability $1 - \frac{1}{s(g)}$, queries all its sensitive bits. This translates to the claim that the average influence of the restrictions of g to the leaves of a deterministic decision tree T sampled from $\mathcal{R}$ is low. Poincaré inequality (Lemma 2.3) now lets us conclude that that the average bias of those restrictions is large. This implies that $\mathcal{R}$ achieves $\mathsf{R}^{\mathsf{prod}}$, yielding the lemma.

Lemma 1.8. Saks and Wigderson gave a zero-error recursive algorithm for g_d. Their algorithm recursively evaluates a randomly chosen child of the root. If that child evaluates to 0, the algorithm outputs 1 and terminates. Else, the algorithm recursively evaluates the other child and outputs the complement.

If the output of the function is 0, then the algorithm will be forced to evaluate both children. However, if the output is 1, then the algorithm avoids evaluating one of the children with probability $1/2$.

We observe that if the inputs are sampled from a product distributions, then firstly, the output will not always be 0; so we may always hope to be able to avoid evaluating one child. Secondly, we will also have both children evaluating to 0 with positive probability, in which case we are guaranteed to save evaluating one child.

We modify the algorithm by Saks an Wigderson to tap these opportunities; in each step, we query the child which is more likely to evaluate to 0. Note that this requires knowledge of the distribution. We look at two successive levels of the tree and show that the above considerations bring us significant advantage over the algorithm by Saks and Wigderson.

Lemma 1.9. As mentioned before, here we work with the original definition of sabotage complexity. Our proof splits into the following steps.

(1) We recursively define a "hard" distribution $\mathcal{P}_d$ over pairs in $g_d^{-1}(0) \times g_d^{-1}(1)$.
(2) We consider an arbitrary zero-error randomized algorithm $\mathcal{R}$ for g_d. We now wish to give a lower bound on the number of queries it makes on expectation to distinguish a random pair sampled from $\mathcal{P}_d$.
(3) Using $\mathcal{R}$, we recursively define a sequence of algorithms $\mathcal{A}_d, \mathcal{A}_{d-1}, \ldots, \mathcal{A}_0$ such that for each i, $\mathcal{A}_i$ is a zero-error algorithm for g_i.
(4) We establish a recursive relation amongst the expected number of queries that g_i makes to distinguish a pair sampled from $\mathcal{P}_i$, for various i. We make a distinction between queries based on their answers (0 or 1). This step involves a small case analysis involving all deterministic trees with two variables.
(5) We finish by solving the recursion established in the previous step.

2 Preliminaries

The notation $[n]$ denotes the set $\{1, \ldots, n\}$. Throughout the article, $g : \{0,1\}^m \to \{0,1\}$ will stand for a total Boolean function and $x = (x_1, \ldots, x_m)$ will stand for a generic input to g. For $b \in \{0,1\}$, $g^{-1}(b) = \{x \in \{0,1\}^n \mid g(x) = b\}$. For a subset S of $\{0,1\}^m$, let $g|_S$ denote the restriction of g to S. A probability distribution μ over $\{0,1\}^m$ is a function $\mu : \{0,1\}^m \to [0,1]$ such that $\sum_{x \in \{0,1\}^m} \mu(x) = 1$. For a subset S of $\{0,1\}^m$, define $\mu(S) := \sum_{x \in S} \mu(x)$. For a subset S of $\{0,1\}^m$ such that $\mu(S) > 0$, $\mu|_S$ is the distribution obtained by conditioning μ on the event that the sample belongs to S. In other words:

$$\mu|_S(x) = \begin{cases} 0 & \text{if} x \notin S, \\ \frac{\mu(x)}{\mu(S)} & \text{if} x \in S. \end{cases}$$

μ is said to be a product distribution if there exist $p_1, \ldots, p_m \in [0,1]$ such that for each $x \in \{0,1\}^n$, $\mu(x) = \prod_{i=1}^m (x_i p_i + (1 - x_i)(1 - p_i))$. In other words, each x_i is independently equal to 1 with probability p_i and 0 with probability $1 - p_i$. Let PROD be the set of all product distributions over $\{0,1\}^m$.

For a subset $I \subseteq [m]$ of indices, $x^{\oplus I}$ denotes the string obtained from x by flipping the variables x_i for each $i \in I$. If $I = \{i\}$, we abuse notation and write $x^{\oplus i}$.

Definition 2.1 (Subcube). A subset C of $\{0,1\}^m$ is called a subcube if there exists a set $S \subseteq [m]$ of indices and bits $\{b_i \mid i \in S\}$ such that $C = \{x \in \{0,1\}^m \mid \forall i \in S, x_i = b_i\}$. The co-dimension of C is defined to be $|S|$.

2.1 Decision Trees for Boolean Functions

A deterministic decision tree for m variables is a binary tree T. Each internal node of T is labeled by a variable x_i for $i \in [m]$, and has two children that corresponds to $x_i = 0$ and $x_i = 1$. Each leaf is labeled by 0 or 1. A decision tree is evaluated on a given input $x = (x_1, \ldots, x_m)$, as follows: Start at the root. In each step, if the current node is an internal node, then query its label x_i. Then navigate to that child of the current node that corresponds to the value of x_i. The computation stops when it reaches a leaf, and outputs the label of the leaf. Let $T(x)$ denote the output of the tree when evaluated on x.

The inputs x that take the tree T to leaf ℓ is exactly the ones which agree with the path from the root to ℓ for every variable queried on the path. Thus, the set of such inputs is a subcube of $\{0,1\}^m$ of co-dimension equal to the depth of ℓ (depth of a leaf is the number of edges on the unique path from the root to the leaf). The notation ℓ will also refer to the subcube corresponding to the leaf ℓ.

T is said to compute $g : \{0,1\}^m \to \{0,1\}$ if

$$\forall x \in \{0,1\}^m, T(x) = g(x).$$

The *deterministic decision tree complexity of* g, denoted by $\mathrm{D}(g)$ is the minimum depth of a decision tree that computes f.

Let μ be a distribution over $\{0,1\}^m$. For a given error parameter $\epsilon \in [0, 1/2]$, T computes g with error probability ϵ over μ if

$$\Pr_{x \sim \mu}[g(x) \neq T(x)] \leq \epsilon.$$

The *distributional query complexity of* g *for error* ϵ *with respect to* μ, denoted by $\mathrm{D}^{\mu}_{\epsilon}(f)$, is the minimum depth of a decision tree that computes f with error probability ϵ over μ. $\mathrm{D}^{\mu}(g)$ is defined to be $\mathrm{D}^{\mu}_{1/3}(g)$.

A randomized decision tree is a probability distribution $\mathcal{R}$ over deterministic decision trees. $\mathcal{R}$ is said to compute g with error probability ϵ if

$$\forall x \in \{0,1\}^m, \Pr_{T\sim\mathcal{R}}[T(x) \neq g(x)] \leq \epsilon.$$

The query complexity of $\mathcal{R}$ is the maximum depth of any decision tree in its support. The *randomized query complexity of g for error ϵ*, denoted by $\mathsf{R}_\epsilon(g)$, is the minimum query complexity of any randomized decision tree $\mathcal{R}$ that computes g with error ϵ. Define $\mathsf{R}(g) := \mathsf{R}_{1/3}(g)$. The following fact is well-known (see, for example [8] for a proof).

Fact 2.2 (Minimax theorem). $\forall \epsilon \in (0,1), \mathsf{R}_\epsilon(g) = \max_\mu \mathsf{D}^\mu_\epsilon(g)$.

We define the *product distributional query complexity of g* for error ϵ, $\mathsf{D}^{\text{prod}}_\epsilon(g)$, as follows:

$$\mathsf{D}^{\text{prod}}_\epsilon(g) := \max_{\mu\in\text{PROD}} \mathsf{D}^\mu_\epsilon(g).$$

2.2 Sensitivity and Influence

For a total Boolean function g, a variable x_i is said to be sensitive for an input x if $g(x) \neq g(x^{\oplus i})$. The sensitivity of x with respect to g, denoted by $s(g,x)$, is the number of sensitive bits of x, i.e., $|\{i \in [n] \mid g(x) \neq g(x^{\oplus i})\}|$. The sensitivity of g, denoted by $s(g)$, is the maximum sensitivity of any input x with respect to g, i.e.,

$$s(g) = \max_{x\in\{0,1\}^m} s(g,x).$$

For a product distribution $\mu \in \text{PROD}$ given by parameters $p_1, \ldots, p_m$, the *influence* of x_i with respect to g and μ is defined as

$$\mathsf{Inf}_i(g) := 4p_i(1-p_i) \Pr_{x\sim\mu}\left[g(x) \neq g(x^{\oplus i})\right],$$

and the influence of g with respect to μ is defined as

$$\mathsf{Inf}(g) = \sum_{i=1}^m \mathsf{Inf}_i(g).$$

The following inequality follows easily from the above definitions, linearity of expectation, and the observation that $4p_i(1-p_i) \leq 1$ for all $p_i \in [0,1]$.

$$\mathsf{Inf}(g) \leq \mathbf{E}_{x\sim\mu} s(g,x). \tag{1}$$

Let $\mathsf{Var}(g)$ denote the variance of the random variable $g(x)$ when x is drawn from μ. The Poincaré inequality bounds $\mathsf{Var}(g)$ in terms of $\mathsf{Inf}(g)$.

Lemma 2.3 (Poincaré inequality). *For every product distribution μ,* $4\mathsf{Var}(g) \leq \mathsf{Inf}(g)$.

A proof of the Poincaré inequality may be found in [14].
In the notations Inf_i, Inf and Var, the dependence on μ is suppressed. μ will be clear from the context.

2.3 Randomized Query Complexity for Product Distributions

Let μ be a product distribution, and T be a deterministic decision tree. For each leaf ℓ of T, let p^μ_ℓ be the probability that the computation of T on an input drawn from μ reaches ℓ. Let $\mathbf{p}^\mu$ denote the probability distribution $(p^\mu_\ell)_\ell$ over the leaves of T. We say that a randomized decision tree $\mathcal{R}$ computes g with error ϵ for product distributions if for every product distribution $\mu \in \text{PROD}$,

$$\mathbf{E}_{T\sim\mathcal{R}}\mathbf{E}_{\ell\sim\mathbf{p}^\mu}\left[\min\left\{\Pr_{x\in\mu|_\ell}[g(x)=0], \Pr_{x\in\mu|_\ell}[g(x)=1]\right\}\right] \leq \epsilon,$$

where the inner expectation is over the leaves of T. We define $\min\{\Pr_{x\in\mu|_\ell}[f(x)=0], \Pr_{x\in\mu|_\ell}[f(x)=1]\}$ to be 0 if $p_\ell^\mu = 0$; the conditional distribution $\mu|_\ell$ is not defined in this case. The *randomized query complexity* of g for product distribution for error ϵ, denoted by $\mathrm{R}_\epsilon^{\mathrm{prod}}(f)$, is the minimum query complexity of a randomized decision tree $\mathcal{R}$ that satisfies the above condition. Define $\mathrm{R}^{\mathrm{prod}}(f) := \mathrm{R}_{1/3}^{\mathrm{prod}}(f)$.

Note that in the above definition, no reference has been made to the labels of the leaves of T. For the purpose of this definition, $\mathcal{R}$ can be thought of as a probability distribution over trees whose leaves are unlabeled.

The following claim shows that $\mathrm{D}_\epsilon^{\mathrm{prod}}(g)$ is bounded above by $\mathrm{R}_\epsilon^{\mathrm{prod}}(g)$.

Claim 2.4. *For every Boolean function g and parameter $\epsilon \in [0, 1/2]$, $\mathrm{D}_\epsilon^{\mathrm{prod}}(g) \le \mathrm{R}_\epsilon^{\mathrm{prod}}(g)$.*

Proof. Let $\mathcal{R}$ be a randomized decision tree that achieves $\mathrm{R}_\epsilon^{\mathrm{prod}}(g)$. Fix a product distribution μ. From $\mathcal{R}$, we will construct a deterministic decision tree (with labeled leaves) T' that errs with probability at most ϵ with respect to μ. This will complete the proof.

To this end, consider any deterministic decision tree T (with unlabeled leaves) in the support of $\mathcal{R}$. We label each leaf ℓ of T as follows: Condition μ on ℓ (assume that the conditional probability is defined; otherwise label ℓ arbitrarily). If the probability of the event "$g(x) = 1$" with respect to this conditional distribution is at least $1/2$, we label ℓ as 1. Else, we label ℓ as 0.

In this way we label each leaf of each deterministic decision tree in the support of $\mathcal{R}$. By the guarantee of $\mathcal{R}$, the resulting randomized decision tree (with labeled leaves) computes g on inputs from μ with error at most ϵ.

Finally, by averaging, it follows that there exists a deterministic tree T' in the support of $\mathcal{R}$ which computes g on a random $x \sim \mu$ with error probability at most ϵ. □

2.4 Sabotage Complexity

The *sabotage complexity* of a Boolean function g for error ϵ, denoted by $\mathrm{RS}_\epsilon(g)$, is defined to be the minimum query complexity of any randomized decision tree $\mathcal{R}$ for which the following is true: For every $x = (x_1, \ldots, x_m) \in g^{-1}(0)$, $y = (y_1, \ldots, y_m) \in g^{-1}(1)$, with probability at least $1 - \epsilon$, a decision tree T drawn from $\mathcal{R}$ when run on x queries an index i such that $x_i \neq y_i$.[1] Define $\mathrm{RS}(g) := \mathrm{RS}_{1/3}(g)$.

Sabotage complexity was defined by Ben-David and Kothari [5]. They defined the measure as the minimum expected query complexity of any randomized decision tree to distinguish each pair of inputs $x \in g^{-1}(0), y \in g^{-1}(1)$. However, as the authors observed, the definition stated above is within a constant factor of the original definition in [5]. See more discussion on this in Section 5 where we work with the original definition.

The following fact can be proven by standard BPP amplification.

Fact 2.5. *$\forall \epsilon, \epsilon' \in (0,1)$ and $\epsilon < \epsilon'$, $\mathrm{RS}_\epsilon(g) = O(\mathrm{RS}_{\epsilon'}(g) \cdot \frac{\log(1/\epsilon)}{\log(1/\epsilon')})$.*

Ben-David and Kothari proved that the sabotage complexity is asymptotically bounded below by many complexity measures that are studied in the context of decision trees. In particular, $\mathrm{RS}(g)$ is asymptotically bounded below by $s(g)$.

Fact 2.6 ([5]). *For all Boolean function $g : \{0,1\}^m \to \{0,1\}$, $\mathrm{RS}(g) = \Omega(s(g))$.*

[1] Note that T queries an index i such that $x_i \neq y_i$ when run on x if and only if T queries an index j such that $x_j \neq y_j$ when run on y.

3 Sabotage Complexity and Product Distributions

In this section, we first prove Lemma 1.6. We then use Lemma 1.6 to prove Theorems 1.3 and 1.5. Lemma 1.6 is restated below.

Lemma 1.6. *For all total Boolean functions* $g : \{0,1\}^m \to \{0,1\}$, *and* $\epsilon : \mathbb{N} \to (0, 1/2)$,

(1) $\mathrm{R}^{\mathrm{prod}}(g) = O(\frac{1}{\epsilon(m)} \cdot \mathrm{RS}_{1-\epsilon(m)}(g) \log s(g))$, *and*

(2) $\mathrm{RS}_{1-2\epsilon(m)}(g) \leq \mathrm{R}^{\mathrm{prod}}_{\frac{1}{2}-\epsilon(m)}(g)$.

The following lemma says that to distinguish each pair of inputs on which the function values differ with high probability, it is necessary and sufficient to query each sensitive bit of each input with high probability.

Lemma 3.1. *Let* $g : \{0,1\}^n \to \{0,1\}$ *be a total Boolean function. Then,* $\mathrm{RS}_\epsilon(g) \leq r$ *if and only if there is a randomized decision tree* $\mathcal{R}$ *of query complexity at most* r *such that for each input* x *and each variable* x_i *sensitive for* x, $\Pr_{T\sim\mathcal{R}}[T \text{ does not query } x_i \text{ when run on } x] \leq \epsilon$.

Proof. **(If)** Let $\mathcal{R}$ be a randomized decision tree of complexity at most r such that for every input x and every variable x_i sensitive for x, $\Pr_{T\sim\mathcal{R}}[T \text{ does not query } x_i \text{ when run on } x] \leq \epsilon$. We will show that $\mathcal{R}$ fails to distinguish any pair $w \in g^{-1}(0), y \in g^{-1}(1)$ with probability at most ϵ. Fix such a pair w, y. let $B = \{i_1, \ldots, i_k\}$ be the positions where w and y differ. Define $B_0 := \emptyset$ and for $1 \leq j \leq k$, define $B_j := \{i_1, \ldots, i_j\}$. Let m be the smallest index such that $g(w^{\oplus B_m}) = 1$. Thus, variable w_m is sensitive for $w^{\oplus B_{m-1}}$ and $w^{\oplus B_m}$. Now, observe that if T does not query any variable w_{i_j} with $i_j \in B$ when run on w, then T does not query w_m when run on $w^{\oplus B_{m-1}}$. By our assumption about $\mathcal{R}$, the probability of this happening when T is sampled from $\mathcal{R}$ is at most ϵ.

(Only if) Let $\mathrm{RS}_\epsilon(g) \leq r$. Thus there exists a randomized decision tree $\mathcal{R}$ of query complexity r that fails to distinguish each pair $w \in g^{-1}(0), y \in g^{-1}(1)$ with probability at most ϵ. Without loss of generality, assume that $x \in g^{-1}(0)$. Then $x^{\oplus i} \in g^{-1}(1)$. Since distinguishing x and $x^{\oplus i}$ is equivalent to querying x_i when run on x, the proof as follows: □

Now, we proceed to proving Lemma 1.6. For convenience, we use ϵ for $\epsilon(n)$ throughout the following proof.

Proof of Lemma 1.6. **(Part 1)** Let $\mathrm{RS}_{1-\epsilon}(g) = r$. By Lemma 3.1, there exists a randomized query algorithm $\mathcal{R}$ of complexity at most r such that for each input x and each variable x_i sensitive for x, $\Pr_{T\sim\mathcal{R}}[T \text{ does not query } x_i \text{ when run on } x] \leq 1-\epsilon$. Let $\mathcal{R}'$ be the algorithm obtained by repeating $\mathcal{R}$ $\frac{2}{\epsilon} \ln s(g)$ times with independent randomness. Thus for each input x and each variable x_i sensitive for x, we have that $\Pr_{T\sim\mathcal{R}'}[T \text{ does not query } x_i \text{ when run on x}] \leq (1-\epsilon)^{(\frac{1}{\epsilon}\cdot 2 \ln s(g))} \leq \frac{1}{s(g)^2}$, where we have used the inequality $1 - x \leq e^{-x}$ for all $x \in (-\infty, \infty)$. Again for each input x, by a union bound over all variables x_i sensitive for x, we have that the probability that a deterministic tree sampled from $\mathcal{R}'$ does not query all variables sensitive for x when run on x, is at most $\frac{s(g,x)}{s(g)^2} \leq \frac{1}{s(g)}$. The query complexity of $\mathcal{R}'$ is $O(\frac{1}{\epsilon}\cdot\mathrm{RS}_{1-\epsilon}(g) \log s(g))$. We will show that $\mathcal{R}'$ computes g with error $1/3$ for product distributions. This will complete the proof of this part.

To this end, fix a product distribution μ. For any deterministic decision tree T and input x of f, define

$$\mathrm{Q}(T, x) = \begin{cases} 1 & \text{if } T \text{ does not query all sensitive variables of } x \text{ when run on } x, \\ 0 & \text{otherwise.} \end{cases}$$

By the property of $\mathcal{R}'$, for every input x, we have that

$$\mathbf{E}_{T\sim\mathcal{R}'}[\mathrm{Q}(T,x)] = \Pr_{T\sim\mathcal{R}'}[\mathrm{Q}(T,x)=1] \le \frac{1}{s(g)}.$$

Since the above is true for each x, we have the following for a random input x sampled from μ.

$$\mathbf{E}_{T\sim\mathcal{R}'}\mathbf{E}_{x\sim\mu}[\mathrm{Q}(T,x)] \le \frac{1}{s(g)}. \tag{2}$$

For each leaf ℓ of T, let p_ℓ^μ be the probability that the computation of T on an input drawn from μ reaches ℓ and $\mathbf{p}^\mu$ denote the probability distribution $(p_\ell^\mu)_\ell$ over the leaves of T. We rewrite Equation (2) as follows:

$$\mathbf{E}_{T\sim\mathcal{R}'}\mathbf{E}_{\ell\sim\mathbf{p}^\mu}\mathbf{E}_{x\sim\mu|_\ell}[\mathrm{Q}(T,x)] \le \frac{1}{s(g)}, \tag{3}$$

treating $\mathbf{E}_{x\sim\mu|_\ell}[\mathrm{Q}(T,x)]$ as 0 if $p_\ell^\mu = 0$. Now, fix an arbitrary leaf ℓ of T such that $p_\ell^\mu > 0$, and consider the Boolean function $g|_\ell$. Note that for any $x \in \ell$, if $\mathrm{Q}(T,x) = 0$, then $s(g|_\ell, x) = 0$. We thus have that

$$\mathbf{E}_{x\sim\mu|_\ell}[s(g|_\ell,x)] \le \Pr_{x\sim\mu|_\ell}[\mathrm{Q}(T,x)=1]\cdot s(g|_\ell) \le \mathbf{E}_{x\sim\mu|_\ell}[\mathrm{Q}(T,x)]\cdot s(g). \tag{4}$$

Since μ is a product distribution and ℓ is a subcube, $\mu\ |_\ell$ is also a product distribution. Equations (1) and (4) thus imply that

$$\mathsf{Inf}(g|_\ell) \le \mathbf{E}_{x\sim\mu|_\ell}[\mathrm{Q}(T,x)]\cdot s(g). \tag{5}$$

Together with Poincaré inequality (Lemma 2.3), Equation (5) implies that

$$\mathrm{Var}(f|_\ell) \le \frac{1}{4}\cdot\mathbf{E}_{x\sim\mu|_\ell}[\mathrm{Q}(T,x)]\cdot s(g). \tag{6}$$

Now, for a random variable X taking value in $\{0,1\}$, $\mathrm{Var}(X) = 4\Pr[X=0]\Pr[X=1] \ge 2\min\{\Pr[X=0],\Pr[X=1]\}$ (since $\max\{\Pr[X=0],\Pr[X=1]\} \ge \frac{1}{2}$). Since ℓ is an arbitrary leaf, we have by Equations (6) and (3) that

$$\begin{aligned}
&\mathbf{E}_{T\sim\mathcal{R}'}\mathbf{E}_{\ell\sim\mathbf{p}^\mu}\left[\min\left\{\Pr_{x\sim\mu|_\ell}[g(x)=0],\ \Pr_{x\sim\mu|_\ell}[g(x)=1]\right\}\right]\\
\le&\frac{1}{2}\cdot\mathbf{E}_{T\sim\mathcal{R}'}\mathbf{E}_{\mathbf{p}^\mu}[\mathrm{Var}(g|_\ell)] && \text{by the above discussion}\\
\le&\frac{1}{8}\cdot\mathbf{E}_{T\sim\mathcal{R}'}\mathbf{E}_{\ell\sim\mathbf{p}^\mu}\mathbf{E}_{x\sim\mu|_\ell}[\mathrm{Q}(T,x)]\cdot s(g) && \text{by Equation (6)}\\
\le&\frac{1}{8} < \frac{1}{3}. && \text{by Equation (3)}
\end{aligned}$$

Since μ is an arbitrary product distribution, we have that $\mathcal{R}'$ computes g with error $1/3$ for product distributions.

(Part 2) Fix a randomized query algorithm $\mathcal{R}$ that attains $\mathrm{R}^{\mathrm{prod}}_{\frac{1}{2}-\epsilon}(g)$. We will show that $\mathcal{R}$ also attains $\mathrm{RS}_{1-2\epsilon}(g)$. By Lemma 3.1 it is sufficient to show that for each input x and each variable x_i sensitive for x, $\Pr_{T\sim\mathcal{R}}[T \text{ does not query } x_i \text{ when run on } x] \le 1-2\epsilon$. To this end, fix an input x and a variable x_i sensitive for x. Now consider the distribution μ that places a probability

mass of $1/2$ on x and places the remaining mass of $1/2$ on $x^{\oplus i}$. Note that μ is a product distribution. Thus from the property of $\mathcal{R}$ we have that

$$\mathbf{E}_{T\sim\mathcal{R}}\mathbf{E}_{\ell\sim\mathbf{p}^\mu}\left[\min\left\{\Pr_{x\in\mu|_\ell}[g(x)=0],\ \Pr_{x\in\mu|_\ell}[g(x)=1]\right\}\right]\leq\frac{1}{2}-\epsilon. \tag{7}$$

Now, if T does not query x_i when run on x, then T has a leaf ℓ that contains both x and $x^{\oplus i}$, $p^\mu_\ell = 1$, and for all other leaves ℓ' of T, $p^\mu_{\ell'} = 0$. Furthermore, $\min\{\Pr_{x\in\mu|_\ell}[g(x) = 0], \Pr_{x\in\mu|_\ell}[g(x)=1]\} = 1/2$. Thus, $\mathbf{E}_{\ell\sim\mathbf{p}^\mu}[\min\{\Pr_{x\in\mu|_\ell}[g(x)=0], \Pr_{x\in\mu|_\ell}[g(x)=1]\}] = 1/2$. We thus have that,

$$\begin{aligned}&\mathbf{E}_{T\sim\mathcal{R}}\mathbf{E}_{\ell\sim\mathbf{p}^\mu}\left[\min\left\{\Pr_{x\in\mu|_\ell}[g(x)=0],\ \Pr_{x\in\mu|_\ell}[g(x)=1]\right\}\right]\\ &\geq\Pr_{T\sim\mathcal{R}}[T\text{ does not query }x_i\text{ when run on }x]\cdot\frac{1}{2}.\end{aligned} \tag{8}$$

Equations (7) and (8) imply that

$$\begin{aligned}&\Pr_{T\sim\mathcal{R}}[T\text{ does not query }x_i\text{ when run on }x]\cdot\frac{1}{2}\leq\frac{1}{2}-\epsilon\\ \Longrightarrow\ &\Pr_{T\sim\mathcal{R}}[T\text{ does not query }x_i\text{ when run on }x]\leq 1-2\epsilon.\end{aligned}$$

This completes the proof. □

Now, we prove Theorems 1.3 and 1.5.

Proof of Theorem 1.3 Part 1. Substituting $\epsilon(n) = 1/6$ in part 2 of Lemma 1.6 we have that

$$\mathrm{R}^{\mathrm{prod}}(g)\geq\mathrm{RS}_{2/3}=\Omega(\mathrm{RS}(g)),$$

where the second relation follows from Fact 2.5. □

Theorem 1.3 (1) and Fact 2.6 imply that

$$\mathrm{R}^{\mathrm{prod}}(g)=\Omega(s(g)). \tag{9}$$

Proof of Theorem 1.3 Part 2.

$$\begin{aligned}\mathrm{RS}(g)=\mathrm{RS}_{1/3}(g)&\geq\mathrm{RS}_{2/3}(g)\\ &=\Omega(\mathrm{R}^{\mathrm{prod}}(g)/\log s(g))&&\text{by Lemma 1.6 part 1 with }\epsilon(n)=1/3\\ &=\Omega(\mathrm{R}^{\mathrm{prod}}(g)/\log\mathrm{R}^{\mathrm{prod}}(g))&&\text{by Equation (9)}\end{aligned}$$ □

Proof of Theorem 1.5. We have

$$\begin{aligned}\mathrm{R}^{\mathrm{prod}}(g)&=O\left(\frac{1}{\epsilon(n)}\cdot\mathrm{RS}_{1-\epsilon(n)}(g)\log s(g)\right)&&\text{Part (1) of Lemma 1.6}\\ &=O\left(\frac{1}{\epsilon(n)}\cdot\mathrm{RS}_{1-2\epsilon(n)}(g)\log s(g)\right)&&\text{since }1-\epsilon(n)\geq 1-2\epsilon(n)\\ &=O\left(\frac{1}{\epsilon(n)}\cdot\mathrm{R}^{\mathrm{prod}}_{\frac{1}{2}-\epsilon(n)}(g)\log s(g)\right)&&\text{by part (2) of Lemma 1.6}\\ &=O\left(\frac{1}{\epsilon(n)}\cdot\mathrm{R}^{\mathrm{prod}}_{\frac{1}{2}-\epsilon(n)}(g))\log\mathrm{R}^{\mathrm{prod}}(g)\right).&&\text{by Equation (9)}\end{aligned}$$ □

ALGORITHM 1: $\mathcal{A}_\mu(x)$

Input: Query access to $x = (x_1, \ldots, x_{2^d})$.
$g \leftarrow g_d$.
if *g is a variable* **then**
  Query g. Return the outcome of the query.
end
else
  Let g_ℓ and g_r respectively be the left and right subtrees of g.
  Let μ_ℓ and μ_r respectively be the product distributions induced on the input spaces of g_ℓ and g_r by μ.
  $t \leftarrow \arg\max_{i \in \{\ell, r\}} \Pr_{y \sim \mu_i}[g_i(y) = 0]$.
  $s \leftarrow \{\ell, r\} \setminus \{t\}$.
  For $i \in \{\ell, r\}$, let $x^{(i)}$ be the input to g_i.
  if $\mathcal{A}_{\mu_t}(x^{(t)}) = 0$ **then**
    return 1.
  end
  else
    return $\overline{\mathcal{A}_{\mu_s}(x^{(s)})}$.
  end
end

4 Separation between $\mathsf{D}^{\mathrm{prod}}$ and R

Recall that g_d denotes the complete binary NAND tree function of depth d. Snir [19] and Saks and Wigderson [16] were the first to study g_d in the context of randomized query complexity. As mentioned in Section 1, it is known from the works of Saks and Wigderson [16] and Santha [17] that $\mathsf{R}(g_d) = \Theta(\alpha^d))$ where $\alpha = \frac{1+\sqrt{33}}{4}$. In this section, we prove Lemma 1.8 (restated below).

Lemma 1.8. *There exists a constant $\delta > 0$ such that $\mathsf{D}^{\mathrm{prod}}(g_d) = O((\alpha - \delta)^d)$.*

For a distribution μ, the zero-error distributional complexity of a Boolean function g, which we denote by $\overline{\mathsf{D}_0^{\mu}}(g)$, is the least expected number of queries made by any deterministic decision tree T on a random input sampled from μ. Define $\overline{\mathsf{D}_0^{\mathrm{prod}}}(g) := \max_{\mu \in \mathrm{PROD}} \overline{\mathsf{D}_0^{\mu}}(g)$. By Markov's inequality, it follows that $\mathsf{D}^{\mathrm{prod}}(g) = O(\overline{\mathsf{D}_0^{\mathrm{prod}}}(g))$.

Proof of Lemma 1.8. We will prove an upper bound on $\overline{\mathsf{D}_0^{\mathrm{prod}}}(g)$. By the preceding discussion, that will prove the lemma.

Let μ be any product distribution over $\{0,1\}^n$. Consider the query algorithm $\mathcal{A}_\mu$[2] given in Algorithm 1.

$\mathcal{A}_\mu$ works as follows: if $d = 1$, i.e., if g_d is a single variable, then $\mathcal{A}_\mu$ queries and returns the value of the variable. Else, $\mathcal{A}_\mu$ recursively evaluates a subtree of the root of g_d whose probability of evaluating to 0 is at least that of the other subtree.[3] If the recursive call returns 0, $\mathcal{A}_\mu$ returns 1. Else, $\mathcal{A}_\mu$ recursively evaluates the other subtree of the root of g_d and returns the complement of the value returned by that recursive call. It is clear that on every input, $\mathcal{A}_\mu$ returns the correct answer with probability 1.

[2]Note that $\mathcal{A}_\mu$ needs the knowledge of μ.
[3]By 'recursively evaluates' we mean that $\mathcal{A}_\mu$ invokes $\mathcal{A}_{\mu'}$ for the distribution μ' induced by μ on the domain of the subfunction under consideration.

Now, we analyze the query complexity of $\mathcal{A}$. Define $T(d,\mu)$ to to be the expected number of queries made by $\mathcal{A}$ on a random input sampled from μ and $T(d) := \max_{\mu\in\mathrm{PROD}} T(d,\mu)$. Clearly, $\overline{\mathrm{D}_0^{\mu}}(g) \le T(d,\mu)$. So, to prove the lemma it suffices to bound $T(d)$.

For $i \in \{\ell, r\}$, define $p_i := \Pr_{x^{(i)}\sim\mu_i}[g_d(x^{(i)}) = 0]$. WLOG assume that $p_\ell \ge p_r$. $\mathcal{A}_\mu$ on input x will recursively evaluate g_ℓ by invoking $\mathcal{A}_{\mu_\ell}$ on input $x^{(\ell)}$. If the recursive call returns 1, then $\mathcal{A}$ will recursively evaluate g_r by invoking $\mathcal{A}_{\mu_r}$ on input $x^{(r)}$. We thus have that,

$$T(d,\mu) = T(d-1,\mu_\ell) + (1-p_\ell)T(d-1,\mu_r). \tag{10}$$

Note that Equation (10) uses the fact that the distribution $\mu_r \mid \{g_\ell(x^{(\ell)}) = 1\}$ is the same as μ_r, which holds as μ is a product distribution.

Let α_ℓ, α_r respectively be the probabilities that the left and right children of g_ℓ evaluate to 0. Similarly, let β_ℓ, β_r respectively be the probabilities that the left and right children of g_r evaluate to 0. Without loss of generality assume that $\alpha_\ell \ge \alpha_r, \beta_\ell \ge \beta_r$ (other cases are similar). Define $m := \min\{\alpha_\ell, \beta_\ell\}$. By using a similar analysis as above and since $T(d',\mu') \le T(d')$ for any depth d' and product distribution μ', we have that

$$\begin{aligned} T(d-1,\mu_\ell) &\le T(d-2) + (1-\alpha_\ell)T(d-2) \\ &\le T(d-2) + (1-m)T(d-2), \text{ and} \qquad (11)\\ T(d-1,\mu_r) &\le T(d-2) + (1-\beta_\ell)T(d-2) \\ &\le T(d-2) + (1-m)T(d-2). \qquad (12)\end{aligned}$$

Substituting Equations (11) and (12) in (10) we have that

$$\begin{aligned} T(d) &\le T(d,\mu) \\ &\le (2-m)(2-p_\ell)T(d-2). \end{aligned} \tag{13}$$

Now, $p_\ell = (1-\alpha_r)(1-\alpha_\ell) \ge (1-\alpha_\ell)^2$.[4] Also, by our assumption $p_\ell \ge p_r = (1-\beta_r)(1-\beta_\ell) \ge (1-\beta_\ell)^2$. We thus have that $p_\ell \ge (1-\min\{\alpha_\ell,\beta_\ell\})^2 = (1-m)^2$. Substituting in Equation (13) we have that

$$\begin{aligned} T(d) &\le (2-m)(2-(1-m)^2)T(d-2) \\ &= (2-m)(1+2m-m^2)T(d-2). \end{aligned} \tag{14}$$

The maximum value of the function $f(x) := (2-x)(1+2x-x^2)$ in the domain $[0,1]$ is $\frac{2}{27}\cdot(17+7\sqrt{7})$. From Equation (14) we have that

$$T(d) = O\left(\sqrt{\frac{2}{27}\cdot(17+7\sqrt{7}))}\right)^d = O(\alpha-\delta)^d \text{ for some constant } \delta > 0. \qquad \square$$

5 Sabotage Complexity of Complete Binary NAND Tree

Recall that g_d stands for the complete binary NAND tree function of depth d. Define $g_0(b) = b$ for $b \in \{0,1\}$. Saks and Wigderson [16] and Santha [17] showed that $\mathrm{R}(g_d) = \Theta(\alpha^d))$ where $\alpha = \frac{1+\sqrt{33}}{4}$. In this section, we prove Lemma 1.9 (restated below).

Lemma 1.9. $\mathrm{RS}(g_d) = \Omega(\alpha^d)$.

Proof. For a randomized query algorithm $\mathcal{R}$ that decides $g : \{0,1\}^m \to \{0,1\}$ with error probability 0, and for inputs x, y such that $g(x) = 0, g(y) = 1$, define the expected sabotage complexity of $\mathcal{R}$ on the pair x, y, denoted by $\mathrm{RS}_E(\mathcal{R}, x, y)$, to be the expected number of queries that

[4]Here we use that μ is a product distribution.

$\mathcal{R}$ makes until (and including) it queries an index i such that $x_i \neq y_i$ when run on x (or y). Define the expected sabotage complexity $\mathrm{RS}_E(\mathcal{R})$ to be $\max_{x\in g^{-1}(0),y\in g^{-1}(1)} \mathrm{RS}_E(\mathcal{R},x,y)$, and the expected sabotage complexity $\mathrm{RS}_E(g)$ to be the minimum $\mathrm{RS}_E(\mathcal{R})$ for any randomized query algorithm $\mathcal{R}$ that decides g with error probability 0. As observed by Ben-David and Kothari, $\mathrm{RS}_E(g) = \Theta(\mathrm{RS}(g))$. In this section, we will work with $\mathrm{RS}_E(g)$ in place of $\mathrm{RS}(g)$.

It follows by standard arguments that for every distribution $\mathcal{D}$ on $g^{-1}(0)\times g^{-1}(1)$ there exists a zero-error randomized (even deterministic) decision tree $\mathcal{R}$ of g such that $\mathbf{E}_{(x,y)\sim\mathcal{D}}[\mathrm{RS}_E(R,x,y)] \leq \mathrm{RS}_E(g)$. To prove Lemma 1.9 it thus suffices to exhibit a hard distribution $\mathcal{D}$ on $g^{-1}(0)\times g^{-1}(1)$ such that for every zero-error randomized tree $\mathcal{R}$ of g, $\mathbf{E}_{(x,y)\sim\mathcal{D}}[\mathrm{RS}_E(\mathcal{R},x,y)]$ is large. The first step in our proof is to define a hard distribution.

5.1 A Hard Distribution

We define a probability distribution $\mathcal{P}_d$ on ${g_d}^{-1}(0)\times {g_d}^{-1}(1)$ as follows: Define $\mathcal{P}_0$ to be the point distribution $\{(0,1)\}$. For $d\geq 1$, $\mathcal{P}_d$ is defined recursively by the following sampling procedure. Let $n := 2^{d-1}$.

(1) Sample $(x,y)\sim\mathcal{P}_{d-1}$. Let $x=(x_1,\dots,x_n)$ and $y=(y_1,\dots,y_n)$.
(2) Sample $\overline{b} := (b_1,\dots,b_n)$ uniformly at random from $\{0,1\}^n$.
(3) For each $i=1,\dots,n$, let $u_i=(u_i^{(0)},u_i^{(1)}), v_i=(v_i^{(0)},v_i^{(1)})\in\{0,1\}^2$ be defined as follows:
  (a) If $(x_i,y_i)=(0,0)$, set $u_i,v_i\leftarrow(1,1)$.
  (b) If $(x_i,y_i)=(0,1)$, set $u_i\leftarrow(1,1)$ and set $v_i\leftarrow(b_i,1-b_i)$.
  (c) If $(x_i,y_i)=(1,0)$, set $u_i\leftarrow(b_i,1-b_i)$ and set $v_i\leftarrow(1,1)$.
  (d) If $(x_i,y_i)=(1,1)$, set $u_i,v_i\leftarrow(b_i,1-b_i)$.
(4) Let x' be the string obtained from x by replacing each x_i by u_i. Similarly let y' be the string obtained from y by replacing each y_i by v_i.
(5) Return (x',y').

Notice that for each $i=1,\dots,n$, $x_i=\mathrm{NAND}(u_i^{(1)},u_i^{(2)})$ and $y_i=\mathrm{NAND}(v_i^{(1)},v_i^{(2)})$. Hence, $g_d(x')=g_{d-1}(x)$ and $g_d(y')=g_{d-1}(y)$. Thus we inductively establish that $\mathcal{P}_d$ is supported on $g_d^{-1}(0)\times g_d^{-1}(1)$. The following observation can be verified to be true by a simple case analysis.

Observation 5.1. *For each $i=1,\dots,n$, $x_i=\overline{u_i^{(b_i)}}$ and $y_i=\overline{v_i^{(b_i)}}$. Furthermore, $u_i^{(1-b_i)}=v_i^{(1-b_i)}=1$.*

In light of Observation 5.1, the sampling process above can be intuitively described as follows: We first sample (x,y) from $\mathcal{P}_{d-1}$. Then, for each i, we sample two two-bit strings u_i and v_i that are jointly distributed in a certain way. If $x_i=y_i$, then $u_i=v_i$. If $x_i\neq y_i$, then the values of x_i and y_i are embedded (as complements) in the b_i-th bits of u_i and v_i respectively. The $(1-b_i)$-th bit of u_i and v_i are set to 1. The marginals of $\mathcal{P}_d$ can be seen to be obtained by conditioning uniform distribution on the "reluctant inputs" considered by Saks and Wigderson [16] to the events $g(x)=0$ and $g(y)=1$. We couple these two conditional distributions in a specific way to obtain $\mathcal{P}_d$.

5.2 A Sequence of Algorithms

Now, we proceed to prove a lower bound on $\mathbf{E}_{(x,y)\sim\mathcal{P}_d}[\mathrm{RS}_E(\mathcal{R},x,y)]$ for any zero-error algorithm $\mathcal{R}$ of g_d. Toward this goal, let $\mathcal{R}$ be a zero-error randomized query algorithm for g_d. Now, using $\mathcal{R}$, we will define a sequence of randomized query algorithms $\mathcal{A}_d,\mathcal{A}_{d-1},\dots,\mathcal{A}_1,\mathcal{A}_0$, where for each $t=d,d-1,\dots,0$, $\mathcal{A}_t$ is a zero-error randomized query algorithm for g_t. Define $\mathcal{A}_d := \mathcal{R}$. Now for $t\leq d-1$, define $\mathcal{A}_t$ recursively as follows: Let $x=(x_1,\dots,x_{2^t})$ be the input to $\mathcal{A}_t$.

(1) Sample $\overline{b}=(b_1,\dots,b_{2^t})$ uniformly at random from $\{0,1\}^{2^t}$.

(2) For each $i = 1, \ldots, 2^t$, define $u_i \in \{0,1\}^2$ as in the definition of $\mathcal{P}_d$ above. Let $x' \in \{0,1\}^{2^{t+1}}$ be the string obtained from x by replacing each x_i by u_i.
(3) Simulate $\mathcal{A}_{t+1}$ on x'. If $\mathcal{A}_{t+1}$ queries $u_i^{(1-b_i)}$ for some i, answer 1. If $\mathcal{A}_{t+1}$ queries $u_i^{(b_i)}$ for some i, make a query to x_i and answer $\overline{x_i}$.
(4) When $\mathcal{A}_{t+1}$ terminates, terminate and return what $\mathcal{A}_{t+1}$ returns.[5]

The correctness of the simulation in step 3 follows from Observation 5.1 and the observation that $g_t(x) = g_{t+1}(x')$. Thus we may inductively establish that for every $t = 1, \ldots, d$, $\mathcal{A}_t$ is a zero-error randomized decision tree of g_t. Moreover, observe that sampling (x, y) from $\mathcal{P}_t$ and running A_t on x (or y) amounts to sampling (x', y') from $\mathcal{P}_{t+1}$ and running A_{t+1} on x' (or y'). Furthermore, $\mathcal{A}_t$ queries the first index i such that $x_i \neq y_i$ exactly when the simulation of $\mathcal{A}_{t+1}$ inside it queries the first index j such that $x'_j \neq y'_j$. We will index the bits of x' as tuples (i, b) where $i \in \{1, \ldots, 2^t\}$ and $b \in \{0,1\}$. Thus $x'_{(i,b)} = u_i^{(b)}$.

5.3 The Lower Bound

For each $b \in \{0,1\}$, each $t = 0, \ldots, d$ and each (x, y) in the support of $\mathcal{P}_t$, define $Q(t, b, x, y)$ to be the number of variables with value b queried by $\mathcal{A}_t$ when run on x until (and including) $\mathcal{A}_t$ queries an index i such that $x_i \neq y_i$. Define $Q(t, b)$ to be the expected value of $Q(t, b, x, y)$ for a random sample (x, y) from $\mathcal{P}_t$, where the expectation is over both the internal randomness of $\mathcal{A}_t$, and the randomness of $\mathcal{P}_t$. Our goal is to derive a recursive relationship amongst the quantities $Q(t, b)$, and then obtain a lower bound on $Q(d, b)$. Since $\mathbf{E}[\mathrm{RS}_E(\mathcal{R}, x, y)] = Q(d, 0) + Q(d, 1)$, the lemma will follow.

Let $0 \leq t \leq d$. For (x, y) in the support of $\mathcal{P}_t$ and $i \in \{1, \ldots, 2^t\}$ define $\mathsf{I}(t, b, i, x, y) := 1$ if $x_i = b$ and $\mathcal{A}_t$ queries x_i when run on x not later than it queries an index on which x and y differ, and define $\mathsf{I}(t, b, i, x, y) := 0$ otherwise. We thus have that

$$Q(t, b, x, y) = \sum_{i=1}^{2^t} \mathsf{I}(t, b, i, x, y). \tag{15}$$

Consider $0 \leq t \leq d-1$, an (x, y) in the support of $\mathcal{P}_t$, bits $b, b' \in \{0,1\}$ and $i \in \{1, \ldots, 2^t\}$ such that $x_i = b$. We are interested in a lower bound on the quantity

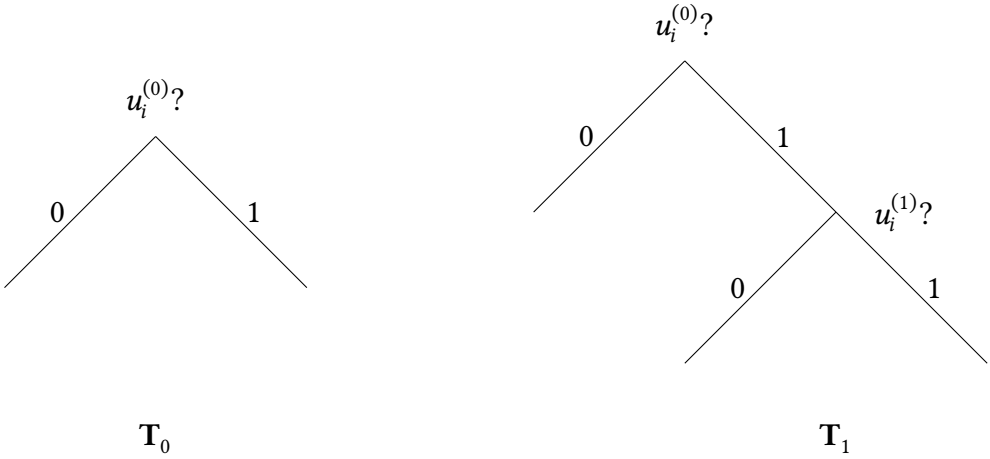
$$F(t, b, b', i, x, y) := \frac{\mathbf{E}[\mathsf{I}(t+1, b', (i,0), x', y') + \mathsf{I}(t+1, b', (i,1), x', y')]}{\mathbf{E}[\mathsf{I}(t, b, i, x, y)]}, \tag{16}$$

whenever the denominator is not 0. We now describe F in words. t, b, b', i, x and y are fixed. x_i is assumed to be b. The denominator is the probability that $\mathcal{A}_t$ queries x_i not later than it queries an index where x and y differ. The numerator is the expected number of b'-valued variables in $\{u_i^{(0)}, u_i^{(1)}\}$ that is queried by the simulation of $\mathcal{A}_{t+1}$ inside $\mathcal{A}_t$, not later than the simulation of $\mathcal{A}_{t+1}$ queries an index where x' and y' differ (which, as discussed before, is exactly when $\mathcal{A}_t$ queries an index where x and y differ). Both expectations are over the randomness of $\mathcal{A}_t$, which includes the sampling of $\overline{b} = (b_1, \ldots, b_{2^t})$ and the randomness in $\mathcal{A}_{t+1}$ that is simulated inside $\mathcal{A}_t$. Note that x' and y' are random strings, as they depend on $b_1, \ldots, b_{2^t}$.

Lower bounding F. We wish to show a lower bound on $F(t, b, b', i, x, y)$. Toward this, let us fix a deterministic decision tree T in the support of $\mathcal{A}_{t+1}$. Furthermore, fix the values of all b_j for $j \neq i$. This fixes all the bits of the string x' except $u_i^{(0)}$ and $u_i^{(1)}$. Now, consider the expression for F where the expectations are conditioned on the above fixings, and are only over the randomness

[5]The return value is not important here. We are bothered only about distinguishin x and y. The algorithms may be thought to have unlabeled leaves.

of b_i (notice that b_i determines $u_i^{(0)}, u_i^{(1)}$ and whether $\mathcal{A}_t$ queries x_i). Under the above fixing, the action of T on the variables $u_i^{(0)}$ and $u_i^{(1)}$ before it queries an index where x' and y' differ is a deterministic decision tree on these two variables. We assume that the tree is not the empty tree (which in particular implies that T does not query an index where x' and y' differ before it queries any of $u_i^{(0)}$ and $u_i^{(1)}$). Assume further that if one of the two variables is queried and found to be 0, the other one is not queried (as their NAND is already fixed to 1, and so the value of g_d is insensitive to the value of the other variable). Under these assumptions there are only two structurally different trees on two variables. The two trees T_0 and T_1 are given below. Two other trees can be obtained by interchanging the roles of $u_i^{(0)}$ and $u_i^{(1)}$ in T_0 and T_1. However, from the symmetry of the NAND function and our distributions, considering T_0 and T_1 suffices.



We now show how to bound F for $b = 1$ and $b' = 0$. Bounds for other combinations can be derived similarly; we list them in Table 1.

First consider tree T_0. Assume that $x_i = b = 1$. Consider the denominator of F. A_t queries x_i if and only if T_0 queries $u_i^{(b_i)}$. T_0 queries only $u_i^{(0)}$. Thus, T_0 queries $u_i^{(b_i)}$ if and only if $b_i = 0$, which happens with probability $1/2$. Thus the denominator is $1/2$.

Now consider the numerator. Number of variables with value $b' = 0$ queried by T is 1 if $u_i^{(0)} = 0$ and 0 otherwise. $u_i^{(0)} = 0$ if and only if $b_i = 0$, which happens with probability $1/2$. Thus the denominator is $\frac{1}{2} \cdot 1 = 1/2$. Hence, in this case, $F = (1/2)/(1/2) = 1$.

Next, consider tree T_1. In this case, x_i is guaranteed to be queried, as the tree always queries the variable whose value is 0. Thus, the denominator is 1. The numerator is also 1; exactly one of the two variables is $b' = 0$ and T_1 stops when it queries a 0. Thus, in this case too, $F = 1/1 = 1$.

We conclude that when $b = 1$ and $b' = 0$, a lower bound on F is $\min\{1, 1\} = 1$. The above analysis holds for a fixed T, as long as its restriction to $\{u_i^{(1)}, u_i^{(2)}\}$ until it queries an index where x' and y' differ, is not an empty tree. By averaging, the lower bounds in Table 1 hold for $\mathcal{A}_{t+1}$ and a random $b_1, \ldots, b_{2^t}$ as long as with positive probability the aforementioned restricted tree is not empty.

A recursive relation for $Q(t, b)$. Fix $0 \le t \le d - 1$, inputs $x, y \in \{0,1\}^{2^t}$ such that (x, y) is in the support of $\mathcal{P}_t$, and bit $b' \in \{0, 1\}$. Now, consider $Q(t + 1, b', x', y')$. Note that x' and y' are random strings, and are determined by x, y (fixed) and $b_1, \ldots, b_{2^t}$ (random). We have that

$$Q(t+1, b', x', y') = \sum_{i=1}^{2^t} (\mathsf{I}(t+1, b', (i,0), x', y') + \mathsf{I}(t+1, b', (i,1), x', y')). \tag{17}$$

Table 1. Lower Bounds on F

b	b'	F
0	0	≥ 0
0	1	≥ 2
1	0	≥ 1
1	1	$\geq 1/2$

We split the above sum into two parts depending on x_i.

$$Q(t+1,b',x',y') = \sum_{1\leq i\leq 2^t, x_i=0} (\mathsf{I}(t+1,b',(i,0),x',y') + \mathsf{I}(t+1,b',(i,1),x',y')) + \sum_{1\leq i\leq 2^t, x_i=1} (\mathsf{I}(t+1,b',(i,0),x',y') + \mathsf{I}(t+1,b',(i,1),x',y')). \quad (18)$$

Now, we take an expectation on both sides over $b_1,\ldots,b_{2^t}$ and the randomness of $\mathcal{A}_{t+1}$, and apply linearity of expectation.

$$\mathbf{E}[Q(t+1,b',x',y')] = \sum_{1\leq i\leq 2^t, x_i=0} \mathbf{E}[\mathsf{I}(t+1,b',(i,0),x',y') + \mathsf{I}(t+1,b',(i,1),x',y')] + \sum_{1\leq i\leq 2^t, x_i=1} \mathbf{E}[\mathsf{I}(t+1,b',(i,0),x',y') + \mathsf{I}(t+1,b',(i,1),x',y')]. \quad (19)$$

Note that the summands of the first sum for which $\mathbf{E}[\mathsf{I}(t,0,i,x,y)]$ are not 0, are $F(t,0,b',i,x,y)\cdot\mathbf{E}[\mathsf{I}(t,0,i,x,y)]$. A similar statement holds for the second sum. We thus have,

$$\mathbf{E}[Q(t+1,b',x',y')] \geq \sum_{1\leq i\leq 2^t, x_i=0, \mathbf{E}[\mathsf{I}(t,0,i,x,y)]\neq 0} F(t,0,b',i,x,y)\cdot\mathbf{E}[\mathsf{I}(t,0,i,x,y)] + \sum_{1\leq i\leq 2^t, x_i=1, \mathbf{E}[\mathsf{I}(t,1,i,x,y)]\neq 0} F(t,1,b',i,x,y)\cdot\mathbf{E}[\mathsf{I}(t,1,i,x,y)]. \quad (20)$$

We would now like to consider $b'=0$ and 1 separately, and plug the bounds of Table 1 into Equation (20). If $\mathbf{E}[\mathsf{I}(t,b,i,x,y)]$ is non-zero, then with positive probability, the restriction of the tree T considered earlier to variables $u_i^{(0)}, u_i^{(1)}$ is not the empty tree; thus the lower bounds of Table 1 are applicable. We thus have

$$\mathbf{E}[Q(t+1,0,x',y')] \geq \mathbf{E}[Q(t,1,x,y)], \text{ and} \quad (21)$$

$$\mathbf{E}[Q(t+1,1,x',y')] \geq 2\mathbf{E}[Q(t,0,x,y)] + \frac{1}{2}\mathbf{E}[Q(t,1,x,y)]. \quad (22)$$

Finally, we take expectations over $(x,y)\sim\mathcal{P}_t$. As discussed before, this has the effect of inducing the distribution $\mathcal{P}_{t+1}$ on (x',y'). We thus have

$$Q(t+1,0) \geq Q(t,1), \text{ and} \quad (23)$$

$$Q(t+1,1) \geq 2Q(t,0) + \frac{1}{2}Q(t,1). \quad (24)$$

One can directly check by enumerating all deterministic zero-error trees for $t=0,1$ that $Q(0,0), Q(0,1), Q(1,0)$, and $Q(1,1)$ are all $\Omega(1)$. It thus follows from Equations (23) and (24) that $Q(t,b)=\Omega(\alpha^t)$ for $b\in\{0,1\}$. In particular, $Q(d,0), Q(d,1)=\Omega(\alpha^d)$. This completes the proof of Lemma 1.9. □

References

[1] Anurag Anshu, Dmitry Gavinsky, Rahul Jain, Srijita Kundu, Troy Lee, Priyanka Mukhopadhyay, Miklos Santha, and Swagato Sanyal. 2018. A composition theorem for randomized query complexity. In *Proceedings of the 37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*. 1.

[2] Andrew Bassilakis, Andrew Drucker, Mika Göös, Lunjia Hu, Weiyun Ma, and Li-Yang Tan. 2020. The power of many samples in query complexity. In *Proceedings of the 47th International Colloquium on Automata, Languages, and Programming*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

[3] Shalev Ben-David and Eric Blais. 2020. A tight composition theorem for the randomized query complexity of partial functions. In *Proceedings of the 2020 IEEE 61st Annual Symposium on Foundations of Computer Science*. IEEE, 240–246.

[4] Shalev Ben-David, Eric Blais, Mika Göös, and Gilbert Maystre. 2022. Randomised composition and small-bias minimax. In *Proceedings of the 2022 IEEE 63rd Annual Symposium on Foundations of Computer Science*. IEEE, 624–635.

[5] Shalev Ben-David and Robin Kothari. 2016. Randomized query complexity of sabotaged and composed functions. In *Proceedings of the 43rd International Colloquium on Automata, Languages, and Programming*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[6] Harry Buhrman and Ronald De Wolf. 2002. Complexity measures and decision tree complexity: A survey. *Theoretical Computer Science* 288, 1 (2002), 21–43.

[7] Sourav Chakraborty, Chandrima Kayal, Rajat Mittal, Manaswi Paraashar, Swagato Sanyal, and Nitin Saurabh. 2023. On the composition of randomized query complexity and approximate degree. In *Proceedings of the Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, September 11-13, 2023, Atlanta, Georgia, USA*. Nicole Megow and Adam D. Smith (Eds.), LIPIcs, Vol. 275, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 63:1–63:23.

[8] Dmitry Gavinsky, Troy Lee, Miklos Santha, and Swagato Sanyal. 2019. A composition theorem for randomized query complexity via max-conflict complexity. In *Proceedings of the 46th International Colloquium on Automata, Languages, and Programming*. LIPIcs, Vol. 132, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 64:1–64:13.

[9] Mika Göös, TS Jayram, Toniann Pitassi, and Thomas Watson. 2018. Randomized communication versus partition number. *ACM Transactions on Computation Theory* 10, 1 (2018), 1–20.

[10] Prahladh Harsha, Rahul Jain, and Jaikumar Radhakrishnan. 2016. Partition bound is quadratically tight for product distributions. In *Proceedings of the 43rd International Colloquium on Automata, Languages, and Programming*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[11] Rahul Jain, Hartmut Klauck, Srijita Kundu, Troy Lee, Miklos Santha, Swagato Sanyal, and Jevgēnijs Vihrovs. 2020. Quadratically tight relations for randomized query complexity. *Theory of Computing Systems* 64, 1 (2020), 101–119.

[12] Gillat Kol. 2016. Interactive compression for product distributions. In *Proceedings of the 48th annual ACM symposium on Theory of Computing*. 987–998.

[13] Ashley Montanaro. 2014. A composition theorem for decision tree complexity. *Chicago Journal Of Theoretical Computer Science* 6 (2014), 1–8.

[14] Ryan O'Donnell. 2014. *Analysis of Boolean Functions*. Cambridge University Press.

[15] Judea Pearl. 1982. The solution for the branching factor of the alpha-beta pruning algorithm and its optimality. *Communications of the ACM* 25, 8 (1982), 559–564.

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

[17] M. Santha. 1991. On the monte carlo boolean decision tree complexity of read-once formulae. In *1991 Proceedings of the 6th Annual Structure in Complexity Theory Conference*. IEEE Computer Society, 180–181.

[18] Clifford Smyth. 2002. Reimer's inequality and tardos' conjecture. In *Proceedings of the 34th annual ACM Symposium on Theory of Computing*. 218–221.

[19] Marc Snir. 1985. Lower bounds on probabilistic linear decision trees. *Theoretical Computer Science* 38 (1985), 69–82.

[20] Avishay Tal. 2013. Properties and applications of boolean function composition. In *Proceedings of the 4th conference on Innovations in Theoretical Computer Science*. 441–454.

[21] Michael Tarsi. 1983. Optimal search on some game trees. *Journal of the ACM* 30, 3 (1983), 389–396.

Received 5 November 2024; revised 16 May 2025; accepted 20 June 2025