Effects of symmetry in combinatorial complexity measures of Boolean functions

Chandrima Kayal

Supervisor: Prof. Sourav Chakraborty

Date of Submission: 30th May, 2024

Advanced Computing and Microelectronics Unit
Indian Statistical Institute
203 B. T. Road, Kolkata-700108

A thesis submitted in partial fulfillment of the requirements for the degree of *Doctor of Philosophy* in *Computer Science* at *Indian Statistical Institute*

Dedicated to maa, baba o Pritam

Acknowledgements

When I started my PhD, I imagined many different ways of writing this part. Each time I had a meaningful conversation or met someone new on my academic journey, I kept adding people to my list. Now that I'm finally writing this, I find myself almost out of words and can't locate my list as well. Still, I'm giving it a try. Let me start by thanking my ISI family and my academic family (which includes some people outside of ISI who are close to my heart) for creating such a supportive environment throughout the years. As the saying goes, "It's about the journey, not the destination", with loads of uncertainty and insecurity about where this journey will take me, I can happily conclude that the journey itself has been incredible. I've met so many wonderful people, visited new places, and learned a lot. There's never been a dull moment, as the worry about the future is always there, to fill any quiet space. All I can say, I am grateful to my life and all the people I have met so far. Let me thank some of the people who have played an important role in my journey.

First and foremost, I am incredibly thankful to my supervisor, Sourav, for his kindness, dedication, and the immense amount of time he invested in guiding us. I still remember the day when I approached him if he could be my PhD advisor but also said that I didn't know which area I wanted to work on. Of course, I had no prepared research mind which took some time and Sourav was generous enough to eventually adopt us (me and my friends Soumi, and Avijeet joined Sourav). I'm deeply grateful for his unwavering support throughout this journey. Having a great PhD experience means being in sync with most of your supervisor's academic and non-academic ideas—though I am happy to skip all the food explorations that Sourav is so passionate about and I love not to skip my night sleep!

Next, I want to thank Arijit, without whom ISI wouldn't feel like ISI. He was always there, whether for academic or non-academic conversations. In Soumi's description, he's like the big brother I'm always looking to pick a fight with! I also owe a heartfelt thanks to Rajat Mittal, whose constant encouragement and warm support made me believe I could keep pushing forward. I'm grateful to Nitin Saurabh, who was always just a phone call away. I frequently bothered him with all my instant garbage ideas and he was so patient and considerate to listen to them, especially when I was rushing to finish last-minute proofs and formalizations. A big thanks to Manaswi, for his patience, support, and helpful suggestions. Everyone knows he's my "local dada" during my PhD, though we've talked less recently because of the time gap and my messy schedule. These are my set of go-to people to disturb me with my bad questions, ideas, and anxiety. I also want to express my deep gratitude to all my collaborators for their patience and for allowing me to work at my own pace. Over the years, I've learned from them the importance of prioritizing my peace of mind, and I'm so thankful for this lesson—I believe it will help me in the long run. Also thanks to Sophie Laplante for hosting me twice in IRIF. These visits helped me to build confidence and trust in myself.

I'm grateful to the wonderful seniors, juniors, and friends from ISI who always had my back. I will not start with names as there is no end to my list here. Whether it was solving LaTeX issues or helping me with visa paperwork, I never had to ask for help—they were always there to support me, and of course, I can't forget the tea breaks at 'thokthok Gate.' Love you all!

Lastly, I would like to acknowledge the financial support of the Indian Statistical Institute, without which this research would not have been possible. Thanks to Indian Association for Research in Computing Science (IARCS) for partially sponsoring my travel to Paris to present my first work at ICALP 2022.

Now let me turn to my family. I owe my life in all ways to my parents and no words can be enough to express what they did for me. It was never easy for them, neither for me, but we are doing great. A special thank you to 'Jolly Didi' for making Kolkata feel like home. I spent the entire COVID-19 lockdown in her home, where I began working on my PhD thesis. Thanks to my wonderful cousins Avik da, Boudi vai, Babu dada and Atri, for making my days so warm and fulfilling that I started referring Kolkata as my 'home'.

Finally, I would like to thank my better half, Pritam, who was there through all my bad hours and worst me. With Pritam I have learned how to forgive myself! More importantly how to be happy with small steps, and small achievements.

No space is enough to acknowledge the people who were there for me. Thanks to them, thanks to all the people who made me rich through the year with their affections, I owe you my life.

-Chandrima

Chandrima Kayal
30/10/24

Abstract

Boolean functions are fundamental objects in computer science and mathematics, representing decision problems in a binary format. This thesis investigates Boolean functions through the lens of the query complexity model and its related complexity measures. Query complexity measures the number of input bits examined to determine the function's output. This provides a lower bound on the time complexity. Depending on other available resources like randomness and quantum resources, other query complexity measures have been defined in this area, known as randomized and quantum query complexity, respectively. In addition to deterministic, randomized, and quantum query complexities, there are various other combinatorial complexity measures like sensitivity, block sensitivity, and certificate complexity. There are also complexity measures that analyze Boolean functions represented as polynomials, such as degree and approximate degree.

The analysis of Boolean functions has gone through extensive exploration, resulting in a diverse and rich literature. The problems investigated in this area can be categorized into distinct types, each representing unique directions of interest and applications. This thesis will be developed along three main directions:

- How are two different complexity measures related to each other? Understanding the relationship between various combinatorial measures involves two parts:
  (1) Relation - Proving that one measure is upper bounded by a function of another measure.
  (2) Separation - Constructing functions that demonstrate separation between two measures.
- The upper and lower bound concerning any particular complexity measure.
- How the complexity measure behaves under composition.

This study explores the above three problems in light of symmetry. In particular, we investigate and prove results for the above three questions for classes of functions with partial symmetry, aiming to provide a deeper understanding of the role of symmetry and other structural properties in the behavior of complexity measures. Important classes of functions that capture partial symmetry include transitive functions, junta-symmetric functions, recursive functions, and many more.

The main contribution of this thesis is divided into two parts:

Part I investigates the relationship between complexity measures for transitive functions and lower bounds on non-deterministic degree complexity for some special classes of symmetric functions. It explores by constructing transitive functions by modifying the known example of functions separating various complexity measures for general Boolean functions and discusses the challenges of partial symmetric structures. All the separation results also match the best-known separations for general Boolean functions. Additionally, it addresses the non-deterministic degree of symmetric functions, providing lower bounds for a special class of symmetric functions.

Part II explores composition theorems for randomized query complexity and approximate degree, particularly focusing on the effects of symmetry. It establishes composition theorems for both measures, extending the classes of outer functions for which composition holds.

In the approximate degree case, we prove composition theorems in terms of the block sensitivity of the outer functions. We also prove composition theorems for the classes of (strongly) junta-symmetric functions (when the outer function is junta-symmetric and the inner function can be anything) and for the classes of recursive functions (when the outer function is a recursive function and the inner function can be anything, as well as when the outer function can be anything while the inner function is a recursive function). In the case of randomized query complexity, we prove composition theorems when the outer function has full randomized query complexity and also for the classes of (strongly) junta-symmetric functions (when the outer function is junta-symmetric).

Table of contents

List of figures

List of tables

Chapter 1

Introduction

Boolean functions are one of the central objects in the study of computer science and mathematics. Any decision problem can be expressed as a Boolean function that takes an n-bit string as input and gives a single-bit output. Formally, it has the following structure:

$$f : \{\mathsf{True}, \mathsf{False}\}^n \to \{\mathsf{True}, \mathsf{False}\}.$$

Depending upon context there are various representations of Boolean functions that have been used in literature, for example, $f : \{0,1\}^n \to \{0,1\}$ or $f : \{-1,1\}^n \to \{-1,1\}$ etc. The versatility of Boolean functions in capturing various problems in a simple form makes it crucial to understand their different properties. To understand the various computational aspects of Boolean functions, various models of computation are considered. A model defines the rules of computations and the associated costs. Specifically, how to access the input to the function, what resources are available for computations, what the costs of computations are, and what the correctness requirements are - these are the questions that the model specifies. Given a model of computation, the associated costs of computation define the complexity measures for the Boolean functions.

In the literature, various computational models and the associated complexity measures have been studied. Some of the well-known and classical complexity measures are time complexity, space complexity, circuit complexity, etc ([AB09]). As the name suggests, the time complexity is the amount of time taken by an algorithm to solve a problem (as a function of the input size). On the other hand, space complexity quantifies the amount of memory space required by an algorithm to solve a problem (as a function of the input size). Another important complexity measure is Boolean circuit complexity ([Bei93, Bop97]), which depends on the size, depth, number of fan-ins, and fan-outs of the circuits calculating a Boolean function. The communication complexity ([KN96, NW95, BVdW07]) model is another computational model, where the inputs are distributed among parties and the cost is the number of communications or messages passed to compute the output value of the functions. This model has been foundational in proving lower bounds in computational complexity theory for many years. Another complexity measure that is extensively studied in this area is query complexity. Here, the cost is the number of bits needed to be queried to decide the output value of the function at that input. In this model, the function is known but the input is not known, where one can query the i-th bit of the input upon which the i-th bit will be revealed. Note that for a Boolean function f defined on a n-bit string, if all the n many bits get revealed, the answer can be decided always. Here the challenge is to read as few bits as possible and decide the output value of the function.

Depending on the other aspects of the model of computation various variants of the above complexity measures are studied. For example, if the computation is performed in a deterministic setup, then the associated measures are the deterministic complexity measures. If the computation is performed in a randomized setup with the available random resources, then randomized time complexity, randomized space complexity, etc can be defined depending on the models. While solving the same problem, if the computation is done in a quantum setup, then the associated complexity measures will be quantum time complexity, quantum query complexity, etc. There are complexity measures concerned with the type of computations, such as non-deterministic time complexity and non-deterministic space complexity defined in the non-deterministic model of computations.

In this thesis, we will focus on the query model, different variants of query complexity measures, and the related complexity measures. The query model, also known as the decision tree model, is one of the simplest models of computation for studying Boolean functions. As we have already mentioned, query complexity measures the minimum number of queries or tests required to determine the output of a Boolean function f for all possible inputs. Any deterministic query algorithm gives rise to a binary tree-like structure, which is called a 'deterministic decision tree' computing the Boolean function f ([Bd02]). In particular, the depth of the tree is the worst-case deterministic decision tree complexity (or deterministic query complexity of that Boolean function) which is denoted by $\mathsf{D}(f)$. Note that in this setup we are not concerned with time complexity, but it provides a lower bound on the time complexity, making this model crucial to study.

Randomized query complexity $(\mathsf{R}(f))$ is the randomized version of query complexity that measures the minimum number of queries required by a randomized algorithm to compute a Boolean function correctly with probability at least $\frac{2}{3}$. A randomized query algorithm can be thought of as a probability distribution over all the deterministic decision trees of depth at most $\mathsf{R}(f)$, such that for each input, x, the fraction of trees (weighted according to the probability distribution) that computes $f(x)$ correctly is at least $2/3$. Note that here the algorithm can make errors but the probability of making an error is at most $\frac{1}{3}$. Precisely this is called bounded error randomized query complexity of a Boolean function f. Depending upon the error parameter, other versions of randomized query complexity can be defined, for example, one-sided error randomized query complexity or zero error randomized query complexity. See [Nis89, Bop97] for an overview of such complexity measures.

The quantum setup is quite different compared to the deterministic and randomized decision tree complexities. Here, the unit of computation is a qubit. With a quantum source available, we can define the quantum query complexity ($\mathsf{Q}(f)$) of a Boolean function. The minimum number of quantum queries needed by an optimal quantum algorithm to decide the output value of a Boolean function with probability at least $\frac{2}{3}$ is defined to be (bounded-error) quantum query complexity of that Boolean function f. See [NC10] for more details. Just like in the case of randomized query setup, here also different types of quantum query complexity can be defined depending on the error parameter. For example, exact quantum query complexity, where the quantum algorithm needs to decide the output value of the function with probability 1. We refer to [BdW02] for a formal definition of all these complexity measures.

All these variants of query complexity measures lead us to the fundamental question: do these available random sources or quantum sources offer any advantages over deterministic computations? These are important questions in computational complexity theory that have been investigated for many years. Consequently, it propels investigation and exploration into the possible benefits and constraints of randomness and quantum resources within computational contexts.

Other than the different variants of query complexity, there are various combinatorial complexity measures that are closely related to the query complexity measures, such as sensitivity ($\mathsf{s}(f)$ [CDR86]), block sensitivity ($\mathsf{bs}(f)$ [Nis89]), certificate complexity ($\mathsf{C}(f)$), randomized certificate complexity ($\mathsf{RC}(f)$), unambiguous certificate complexity ($\mathsf{UC}(f)$), and many more. When we talk about the representation of the Boolean function, representing the Boolean function in terms of polynomials is one of the popular and useful forms. From this representation some natural analytic measure comes into the picture like degree ($\deg(f)$), approximate degree ($\widetilde{\deg}(f)$), spectral sensitivity ($\lambda(f)$), etc. For over four decades, understanding these measures has been an active area of research in computational complexity theory. These combinatorial measures have applications in many other areas of theoretical computer science, making the above a question central. Usually one puts these studies under the heading of "analysis of Boolean functions."

The analysis of Boolean functions through various combinatorial complexity measures has received considerable attention over the years, leading to developments in various directions and resulting in interesting new phenomena in diverse fields. Despite this attention and development, many long-standing open problems remain in this area. Some notable examples include the Aaronson-Ambainis conjecture [AA14], the log-rank conjecture, the Sensitivity Conjecture [NS94], the approximate degree composition lower bound, and the randomized query complexity lower bound. In the last couple of years, some celebrated conjectures have been resolved, such as the quadratic relation between sensitivity and the degree of Boolean functions (known as Sensitivity Conjecture) [Hua19]. Many more central open questions continue to drive progress in the community, resulting in a long list of significant works that have served as foundational building blocks throughout computational theory [NS94, NW95, BBC$^+$01a, Aar08, RS12]. Various beautiful surveys in this area can be found in [BdW02, HKP11, ABK$^+$21a], offering insightful introductions to this area.

1.1 Important Meta Questions in this area

Analysis of Boolean function has gone through extensive exploration and study, resulting in a diverse and rich literature. However, most of the problems investigated can be categorized into distinct types, each representing a unique direction of interests and applications, of which we will be describing three main directions towards which this thesis will be developed.

Let us start with an example to see the type of problems that can be addressed by analyzing and understanding the relationship between complexity measures. Suppose we have a deterministic query algorithm for a specific function and we want to design a randomized algorithm for that function. Naturally, we anticipate some improvement in the query complexity, but the question arises: to what extent? Is it reasonable to expect an exponential improvement compared to the deterministic algorithm? Considering the scenario, it has been already established by [Mid04] that for all total Boolean functions, $\mathsf{D}(f)$ is upper bounded by $\mathsf{R}(f)^3$. This says that one can not expect exponential improvement for your randomized algorithm. Hence we can only get polynomial improvement. Similarly, one may be interested to study how much low can one complexity measure be compared to another for any Boolean function.

Question 1.1. *How are two different complexity measures related to each other?*

Understanding the relationship between various combinatorial measures involves two parts:

1. Relationships - proving that one measure is upper bounded by a function of another measure. Precisely, for any two complexity measure M_1 and M_2 if there exists some function p_1 such that $M_1(f) = O(p_1(M_2(f)))$ for all Boolean function f then we say M_2 gives upper bound on the complexity measure M_1.

   For example, for any Boolean function f, $\deg(f) \leq \mathsf{s}(f)^2$ and $\mathsf{D}(f) \leq \mathsf{R}(f)^3$.

2. Separations - constructing functions that demonstrate separation between two measures. Precisely, if there exists a Boolean function g and some polynomial p_2 such that $M_1(g) = \Omega(p_2(M_2(g)))$, then we say there exists some function that achieves a polynomial separation between M_1 and M_2.

   For example, there exists a Boolean function f with $\deg(f) \geq \mathsf{s}(f)^2$. Also, there exists another Boolean function g with $\mathsf{D}(g) \geq \mathsf{R}(g)^2$.

Obtaining tight bounds between pairs of combinatorial measures - that is when the relationship and the separation results match - is the holy grail of this area of research. In other words, the goal is to obtain functions p_1 and p_2 (for the above questions) which are asymptotically equal or close. The current best-known results for different pairs of functions in general have been nicely compiled in [ABK$^+$21b].

Another very natural and well-studied problem is the upper bound and lower bound on a complexity measure for some Boolean functions in terms of the arity of the functions. That is fix some Boolean function on n variables, for example, OR. Now the question to ask what is the deterministic query complexity or quantum query complexity of OR on n bits? So, the second type problem that has been widely studied in different context in this area is:

Question 1.2. *The upper bound and lower bound concerning any particular complexity measure?*

For example, consider the degree of a Boolean function $f : \{0,1\}^n \to \{0,1\}$. How much lower can the degree go if f depends on n variables? That is, what is the minimum possible degree of a non-constant Boolean function that depends on all the variables? [NS94] proved that degree of such Boolean functions is at least $\Omega(\log n)$.

Another crucial step towards understanding a complexity measure is: how does the complexity measure behave when two Boolean functions are combined to obtain a new function (i.e., what is the relationship between the measure of the resulting function and the measures of the two individual functions) [BKT19, BDGKW20, GSS16, Tal13]?

Let $M(\cdot)$ be a complexity measure of Boolean functions. A central question in complexity theory is the following.

Question 1.3 (Composition question for M)**.** *Is the following true for all Boolean functions f and g:*

$$M(f *_1 g) = \tilde{\Theta}(M(f) *_2 M(g))?$$

where $*_1$ is some algebraic operations that are used to combine two Boolean functions the outer function, f, and the inner function, g, and create a new Boolean function $f *_1 g$. The $*_2$ is some other binary operation that acts on $M(g), M(f)$. The notation $\tilde{\Theta}(\cdot)$ hides poly-logarithmic factors of the arity of the outer function f.

One particularly natural combination of functions is called *composition*, and it takes a central role in complexity theory. For any two Boolean functions $f : \{0,1\}^n \to \{0,1\}$ and $g : \{0,1\}^m \to \{0,1\}$, the composed function $f \circ g : \{0,1\}^{nm} \to \{0,1\}$ is defined as the function

$$f \circ g(x_1, \ldots, x_n) = f(g(x_1), \ldots, g(x_n)),$$

where $x_i \in \{0,1\}^m$ for $i \in [n]$. For the function $f \circ g$, the function f is called the outer function, and g is called the inner function. The composition of Boolean functions with respect to different complexity measures is a very important and useful tool in areas like communication complexity, circuit complexity, and many more. To take an example, a popular application of composition is to create new functions demonstrating better separations (refer to [NS94, Tal13, Amb05, GSS16] for some other results of similar flavor). Precisely for *composition* Question 1.3 asks the following:

$$M(f \circ g) = \tilde{\Theta}(M(f) \cdot M(g))?$$

It is known that the answer to the above-mentioned question is in the affirmative for complexity measures like deterministic decision tree complexity [Sav02, Tal13, Mon14], degree [Tal13] and quantum query complexity [Rei11a, LMR$^+$11, Kim13a]. While it is well understood how several complexity measures behave under composition, there are two important measures (even though well studied) for which this problem remains wide open: randomized query complexity (denoted by R) and approximate degree (denoted by $\widetilde{\deg}$) [She12, NS94, Amb05, She13a, BT13, She13b]. (See Definition 2.2 and Definition 2.16 for their respective formal definitions.)

For both R and $\widetilde{\deg}$ the upper bound inequality is known, i.e., $\mathsf{R}(f \circ g) = \tilde{O}(\mathsf{R}(f) \cdot \mathsf{R}(g))$ (folklore) and $\widetilde{\deg}(f \circ g) = O(\widetilde{\deg}(f) \cdot \widetilde{\deg}(g))$ [She13d]. But the corresponding lower bounds for both these measures are still major open problems in this area.

1.2 Special Kind of Functions - Role of Symmetry

All the above-mentioned problems can be studied for various different classes of Boolean functions and the answers to the question may be different for various classes. Exploring special classes of Boolean functions represents another extensively studied area within this field. For example, when a function possesses specific properties, it often leads to variations in both the relationships between measures and the results of separations compared to those observed in general Boolean functions. For example, while it is known that there exists f such that $\mathsf{bs}(f) = \Theta(\mathsf{s}(f)^2)$ [Rub95], for a symmetric function, if we take the function to be 'monotone' (a function is monotone if the output value doesn't increase with setting more variables to 1) then the relation becomes $\mathsf{bs}(f) = \Theta(\mathsf{s}(f))$ for all monotone Boolean function f. On the other hand, the best-known upper bound of $\mathsf{bs}(f)$ in terms of sensitivity for general Boolean functions is $\mathsf{s}(f)^4$ [Hua19]. Similarly, for graph properties, one may obtain tighter bounds compared to general functions.

So depending upon the properties of the function the complexity measure behaves differently. This takes a central space in the area of analysis of Boolean functions due to it's usefulness in understanding the behaviour of Boolean function. Various important classes of function have got attention in the literature. For example linear threshold function, polynomial threshold function, read-once formulas, monotone functions, graph properties, cyclically invariant functions are many more. One crucial aspect of Boolean functions that is often central to the definitions of these classes is "symmetry". One may quantify symmetry in various ways. In this thesis we will concentrate on exploring the classes of functions derived from the presence of symmetry within their structure.

1.2.1 Symmetric functions

Any Boolean function $f : \{0,1\}^n \to \{0,1\}$ is symmetric if it is invariant under any permutation from S_n on the input strings where S_n denotes the set of all permutations on $[n]$. There are other ways of characterizing symmetric functions. Namely, a symmetric function is one whose value on any input depends only on the Hamming weights of the input. Examples of symmetric functions include OR, AND, PARITY, etc. Symmetric functions have gotten significant attention for their applications in cryptography, coding theory, and various other fields, as well as for their inherent structural elegance. Consequently, the classes of symmetric functions have been extensively studied over many years and are almost well-understood. Here is some of the important work that was done for the classes of symmetric functions [Pat92, OWZ11, BBGK18a].

1.2.2 Partially Symmetric functions

Now, we will define classes of Boolean functions that demonstrate partial symmetry but lack comprehensive understanding. Unlike the highly constrained nature of completely symmetric functions, which can be defined in terms of the Hamming weights of the inputs, these partially symmetric functions offer more flexibility due to their relaxed symmetry requirements. Various intriguing classes of functions emerge when we loosen the condition of complete symmetry. Note that partial symmetry can be defined in different ways; for instance, we can demand that the function remains invariant under subsets of S_n, or we can say the function is symmetric on some parts of the input bits while leaving other bits asymmetrical. In the following discussion, we will outline two important classes of partially symmetric functions, which have gained attention and are being studied in various contexts:

- Weakly symmetric functions or Transitive functions,
- Partly symmetric functions or Junta-symmetric functions.

Transitive Functions or weakly symmetric functions

One of the most well-studied classes of Boolean functions that nicely captures the idea of partial symmetry is the classes of transitive functions. A group $G \leq \mathsf{S}_n$ is transitive if for all $i, j \in [n]$ there exists a $\sigma \in G$ such that $\sigma(i) = j$ and we say a function $f : \{0,1\}^n \to \{0,1\}$ is transitive if there is a transitive group $G \leq \mathsf{S}_n$ such that the function value remains invariant even after the indices of the input is acted upon by a permutation from G. Note that, when

$G = \mathsf{S}_n$ then the function is symmetric, hence the classes of transitive functions also capture the classes of Symmetric functions.

Transitive functions are also called "weakly symmetric" functions and have been studied extensively in the context of various complexity measures. It is expected that functions with more symmetry must have less variation among the various combinatorial measures. For example, consider the classes of symmetric functions, where the transitive group is S_n, most of the combinatorial measures become the same up to a constant [1]. Another example of transitive functions is the graph properties. The input is the adjacency matrix, and the transitive group is the graph isomorphism group acting on the bits of the adjacency matrix. A series of works [Tur84, Sun11, LS17, GMSZ13] have tried to obtain tight bounds on the relationship between sensitivity and block sensitivity for graph properties. Those works also investigate how low can sensitivity and block sensitivity go for graph properties.

In papers like [SYZ04, Cha11, Sun07, Dru11] it has been studied how low can the combinatorial measures go for transitive functions. The behavior of transitive functions can be very different from general Boolean functions. For example, it is known that there are Boolean functions for which the sensitivity is as low as $\Theta(\log n)$ where n is the number of effective variables[2], it is known (from [Sun07] and [Hua19]) that if f is a transitive function on n effective variables then its sensitivity $\mathsf{s}(f)$ is at least $\Omega(n^{1/8})$[3]. Similar behavior can be observed in other complexity measures also. For example, it is easy to see that for a transitive function the certificate complexity is $\Omega(\sqrt{n})$, while the certificate complexity for a general Boolean function can be as low as $O(\log n)$. A naturally related question is:

What is the tight relationship between various pairs of combinatorial measures for transitive functions?

By definition, the known 'relationship' results for general functions hold for transitive functions. But tighter 'relationship' results may be obtained for transitive functions. On the other hand, the existing 'separations' don't extend easily since the example used to demonstrate separation between certain pairs of measures may not be transitive. For example, some of the celebrated function constructions like the pointer function [ABB$^+$17], used for demonstrating tight separations between various pairs like $\mathsf{D}(f)$ and $\mathsf{R}_0(f)$ (Check Definition 2.3), are not transitive. Similarly, the functions constructed using the cheat sheet

[1]There are still open problems on the tightness of the constants.

[2]A variable is effective if the function is dependent on it.

[3]Note that, [Nis89] showed that $\text{bs}(f) = \Omega(\sqrt{\mathsf{C}(f)})$, which combined with $\mathsf{C}(f) = \Omega(\sqrt{n})$ ([SYZ04]) implies $\text{bs}(f) = \Omega(n^{1/4})$ for transitive function. It is still an open problem ([Cha11, Dru11]) that what is the best possible $c > 0$ such the sensitivity of a transitive function is $\Omega(n^c)$?

techniques [ABK16] used for separation between quantum query complexity and degree, or approximate degree are not transitive. Constructing transitive functions that demonstrate tight separations between various pairs of combinatorial measures is very challenging. In this thesis, we explore these directions of constructing transitive function that gives separations that match the known separations of general functions.

Junta-Symmetric functions or partly symmetric functions

A function $f : \{0,1\}^n \to \{0,1\}$ is called a k-junta symmetric function if there exists a set J of size k of variables such that the function value depends on assignments to the variables in J as well as on the Hamming weight of the whole input. k-junta symmetric functions can be seen as a mixture of symmetric functions and k-juntas. This class of functions has been considered previously in the literature, particularly in [CFGM12, BWY15] where these functions play a crucial role. [CFGM12] even presents multiple characterisations of k-junta symmetric functions for constant k.

Note that by definition an arbitrary k-junta (i.e., a function that depends on k variables) is also a k-junta symmetric function since we can consider the dependence on Hamming weight to be trivial. Thus, this notion loses out on the symmetry of the function considered. But there are other ways to define Junta symmetric functions in a bit more restricted which we call *strongly* k-junta symmetric functions. A k-junta symmetric function is called strongly k-junta symmetric if every variable is influential. In other words, there exists a setting for the junta variables such that the function value depends on the Hamming weight of the whole input in a non-trivial way. Precisely we can express a k-junta symmetric function on n bit input x as $f(x_1,\ldots,x_k,|x|)$ where strongly k-junta function depends nontrivially on the hamming weights of the input strings $|x|$. That is, there exist Hamming weights such that changing them can change the output of the functions. On the other hand, we can take any k-junta function and it is also a junta symmetric function, $|x|$ being irrelevant.

In terms of composition theorems of $\widetilde{\mathrm{deg}}$ and R it was proved in [BBGK18a] and [GJPW18a] that $\widetilde{\mathrm{deg}}$ and R respectively composes when the outer function is symmetric. So it is natural to ask can we extend the class for which composition theorem is true? In particular, can we ask,

Can we prove the composition theorem when the outer function is partly symmetric?

As we have mentioned the main goal of investigating the above-mentioned problems for some classes of functions gives a better understanding of general classes of functions.

That is, what are the results or understandings that are needed and have been produced while exploring the problems for the classes of functions with partial symmetry?

1.2.3 Recursive functions

Here we will define another class of functions that have played an important role from the dawn of Boolean function literature, which is recursive functions. For any Boolean function $f : \{0,1\}^t \to \{0,1\}$ we define recursive function $f^d : \{0,1\}^{t^d} \to \{0,1\}$ by $f^d = \underbrace{f \circ f \circ \ldots \circ f}_{d\ times}$.

Recursive functions are an important class of Boolean functions that are studied in various different contexts in the analysis of Boolean functions, mainly in proving various lower bounds [Amb05, Sni85, SW86a, NS94, NW95, BHT17]. For example, the Kushilevitz's function [NW95] which is the only known non-trivial example of functions with low degree and high sensitivity is a recursive function of a carefully chosen base function. Recursive majority, $\mathsf{MAJORITY}_3^d$, is another recursive function that has been studied extensively in the literature for its different properties [SW86b, JKS03, Leo13, MNS$^+$16]. Boppana (see, e.g., [SW86b]) used it to provide the first evidence that the randomized query is more powerful than the deterministic query [SW86b]. In the same article, they show a similar separation using recursive $\mathsf{AND}_2 \circ \mathsf{OR}_2$ function too. In a different application of recursive $\mathsf{AND}_2 \circ \mathsf{OR}_2$, [JRSW99] shows the separation between deterministic tree-size complexity and the number of monomials in the minimal DNF or CNF.

Note that one of the central motivations of studying composition theorem for various complexity measures is that: composition has been a very useful tool to prove lower bound for various problems. Let us take a simple example. It is known that for all Boolean functions f, sensitivity is less than or equal to the square of the degree of f. Suppose we want to create a Boolean function that archives the separations between degree and sensitivity. Naturally, we need a function for which sensitivity is high and degree is low. One standard approach in this field is to start with a function that works on a small number of bits and has the properties we want. Then, we repeat or "compose" this function recursively to make a bigger one. Ideally, the measures we're interested in, like sensitivity and degree, behave nicely when we compose functions recursively, and this helps us see a polynomial difference between them. One classic example of this is Kushilevitz's function [NS94, Bd02] which demonstrates a polynomial difference between degree and sensitivity through recursive composition. So, it is natural to ask,

Does composition theorem hold for the classes of recursive functions?

1.3 Overview of the Results and Organization

The thesis is divided into two parts where in the first part we have studied Problem 1 and Problem 2. Precisely in the first part we have studied the relationship between complexity measures for Transitive functions and lower bounds on complexity measure for some classes of symmetric functions. In part II we have explored the composition theorem, mainly two big open problem in this area, randomized query complexity composition and approximate degree composition lower bounds. Here again we have explored the problem to see the effects of symmetry on those problems and our result extend the boundary of known results in this direction.

1.3.1 Part I

Our results on Question 1

We try to answer Question 1 regarding the relationship between various pairs of measures for the classes of Transitive functions. In particular, our main contribution is to construct transitive functions that have similar complexity measures as the *pointer functions*.

Our results and the current known relations between various pairs of complexity measures of transitive functions are compiled in Table 5.2. This table is along the lines of the table in [ABK+21b] where the best-known relations between various complexity measures of general Boolean functions were presented.

Chapter 4

Deterministic query complexity and zero-error randomized query complexity are two of the most basic measures and yet the tight relation between these measures was not known until recently. In [Sni85] they showed that for the "balanced NAND-tree" function, $\widetilde{\wedge}$-tree, $\mathsf{D}(\widetilde{\wedge}\text{-tree}) \geq \mathsf{R}_0(\widetilde{\wedge}\text{-tree})^{1.33}$. Although the function $\widetilde{\wedge}$-tree is transitive, the best known 'relationship' was quadratic, that is for all Boolean function f, $\mathsf{D}(f) = O(\mathsf{R}_0(f)^2)$. In [ABB+17] a new function, $\mathsf{A1}$, was constructed for which deterministic query complexity and zero-error randomized query complexity can have a quadratic separation between them, and this matched the known 'relationship' results. The function in [ABB+17] was a variant of the pointer functions - a class of functions introduced by [GPW18] that has found extensive usage in showing separations between various complexity measures of Boolean functions. The function, $\mathsf{A1}$, also gave (the current best known) separations between deterministic query complexity and other measures like quantum query complexity and degree. But the function

$\mathsf{A1}$ is not transitive. Using the $\mathsf{A1}$ function we construct a transitive function that demonstrates a similar gap between deterministic query complexity and zero-error randomized query complexity, quantum query complexity, and degree.

Theorem 1.4 ([CKP21]). *There exists a transitive function $F_{1.4}$ such that*

$$\mathsf{D}(F_{1.4}) = \tilde{\Omega}(\mathsf{Q}(F_{1.4})^4), \qquad \mathsf{D}(F_{1.4}) = \tilde{\Omega}(\mathsf{R}_0(F_{1.4})^2), \qquad \mathsf{D}(F_{1.4}) = \tilde{\Omega}(\deg(F_{1.4})^2).$$

In [ABB$^+$17, BHT17] various variants of the pointer function have been used to show separation between other pairs of measures like R_0 with R, Q_E, deg, and Q, R with $\widetilde{\deg}$, deg, Q_E and sensitivity. Inspired by these functions we construct transitive versions that demonstrate similar separation for transitive functions as that of general functions. The construction of these functions, though more complicated and involved, are similar in flavor to that of $F_{1.4}$. The proof of Theorem 1.4 is presented in Section 4.5 of Chapter 4.

Chapter 5

Our proof techniques also help us make transitive versions of other functions like that used in[ABK16] to demonstrate the gap between Q and certificate complexity. Precisely we prove the following:

Theorem 1.5 ([CKP21]). *There exists a transitive function $F_{1.5} : \{0,1\}^N \to \{0,1\}$ such that*

$$\mathsf{Q}(F_{1.5}) = \widetilde{\Omega}(\mathsf{C}(F_{1.5})^2).$$

The result is presented in Chapter 5. All our results are compiled (and marked in green) in Table 5.2.

One would naturally ask what stops us from constructing transitive functions analogous to the other functions, like cheat sheet-based functions. In fact, one could ask why to use ad-hoc techniques to construct transitive functions (as we have done in most of our proofs) and instead, why not design a unifying technique for converting any function into a transitive function that would display similar properties in terms of combinatorial measures [4]. If one could do so, all the 'separation' results for general functions (in terms of separation between pairs of measures) would translate to separation for transitive functions. We have discussed

[4] In [BCG$^+$20] they have demonstrated a technique that can be used for constructing a transitive partial function that demonstrates gaps (between certain combinatorial measures) similar to a given partial function that need not be transitive. But their construction need not construct a total function even when the given function is total.

why such a task is challenging. We argue the challenges of making transitive versions of the cheat sheet functions.

Our results on Question 2

Towards Problem 2 we consider the complexity measure of the non-deterministic degree (ndeg) of symmetric polynomials and prove a lower bound on the complexity measures for a special kind of symmetric functions. A non-deterministic polynomial representing a Boolean function is nonzero iff $f(x) = 1$. In has been used to characterize non-deterministic version of quantum algorithms. More related detailed explorations can be found in the work of [dW00, Mid04].

Chapter 6

While proving the lower bound on ndeg, we have considered a well-studied problem from discrete geometry known as the ‘Covering problem’. In particular, this problem seeks for a degree lower bound on a polynomial that vanishes on a subset of $\{0,1\}^n$ and non-zero everywhere else on the Boolean cube. In this direction our result gives a lower bound when the polynomial is non-zero on a layer of a Boolean cube, that is when a polynomial is non-zero on all points $\{x : |x| = k\}$ (Hamming weight of x is k) where $k \in [n]$. In particular, our theorem gives a more generalized result that takes zeros of higher order into account. Formally, we say that a polynomial $P \in \mathbb{R}[x_1,\dots,x_n]$ has a zero of *multiplicity* at least t at a point $v \in \mathbb{R}^n$ if all derivatives of P up to order $t-1$ vanish at v and $P(v) = 0$. In this paper, we prove the following generalization of Theorem 6.1.

Theorem 1.6 ([GKN23]). *Given $t \in \mathbb{N}$ and $k \in [n]$, let $P \in \mathbb{R}[x_1,\dots,x_n]$ be a polynomial such that at each point $u \in \{0,1\}^n \setminus \{x : |x| = k\}$, P has a zero of multiplicity at least t and at each point $v \in \{x : |x| = k\}$, P has a zero of multiplicity exactly $(t-1)$. Then*

$$\deg(P) \geq \max\{k, n-k\} + 2t - 2.$$

Putting $t = 1$ we get a lower bound on ndeg of a symmetric polynomial which does not vanish on subsets like $\{x : |x| = k\}$.

This part is based on:

- Title: Separations between Combinatorial Measures for Transitive Functions
  Author: Sourav Chakraborty, Chandrima Kayal, Manaswi Paraashar

Published in ICALP 2022, volume 229, pages 36:1– 36:20, 2022.
https://doi.org/10.4230/LIPIcs.ICALP.2022.36

- *Part of the following paper:*
  Almost covering all the layers of the hypercube with multiplicities
  Author: Arijit Ghosh, Chandrima Kayal, Soumi Nandi
  Published: Discrete Mathematics, volume 346(7), 113397, 2023.
  https://doi.org/10.1016/j.disc.2023.113397

1.3.2 Part II

Our results on Question 3

The initial steps taken towards answering the composition question for R were to show a lower bound by using a weaker measure of outer function than the randomized query complexity. In particular, it was shown that $\mathsf{R}(f \circ g) = \Omega(\mathsf{s}(f) \cdot \mathsf{R}(g))$ [GJPW18a, AKK16], where $\mathsf{s}(f)$ is the sensitivity of f (Definition 7.10). Since $\mathsf{s}(f) = \Theta(\mathsf{R}(f))$ ([BdW02]) for any symmetric function[5] f, these results show that R composes when the outer function is a symmetric function (like OR, AND, Majority, Parity, etc.). The lower bound was later improved to obtain $\mathsf{R}(f \circ g) = \Omega(\mathsf{fbs}(f) \cdot \mathsf{R}(g))$ [BDG$^+$20, BDB20a], where $\mathsf{fbs}(f)$ is the fractional block sensitivity of f (Definition 2.12).

Unfortunately, at this stage, there could even be a cubic gap between R and fbs; the best-known bound is $\mathsf{R}(f) = O(\mathsf{fbs}(f)^3)$ [ABK$^+$21b]. Given that there are already known Boolean functions with quadratic gaps between $\mathsf{fbs}(f)$ and $\mathsf{R}(f)$ (e.g., BKK function [ABK16]), it is natural to consider composition question for randomized query complexity when R is *big* but fbs is *small*. We take a step towards this problem by showing that composition for R holds when the outer function has full randomized query complexity, i.e., $\mathsf{R}(f) = \Theta(n)$, where n is the arity of the outer function f.

For the composition of $\widetilde{\deg}$, Sherstov [She12] already showed that $\widetilde{\deg}(f \circ g)$ composes when the approximate degree of the outer function f is $\Theta(n)$, where n is the arity of the outer function. Thus approximate degree composes for several symmetric functions (having approximate degree $\Theta(n)$, like Majority and Parity). Though, until recently it was not even clear if $\widetilde{\deg}(\mathsf{OR} \circ \mathsf{AND}) = \Omega(\widetilde{\deg}(\mathsf{OR})\,\widetilde{\deg}(\mathsf{AND}))$ (arguably the simplest of composed functions). OR has approximate degree $O(\sqrt{n})$, and thus the result of [She12] does not prove $\widetilde{\deg}$ composition when the outer function is OR (similarly for AND). In a long series of work

[5]Functions that depend only on the Hamming weight of their input.

by [NS94, Amb05, She13a, BT13, She13b], the question was finally resolved; it was later generalized to the case when the outer function is symmetric [BBGK18a].

In contrast to R composition, no lower bound on the approximate degree of composed function is known with a weaker measure on the outer function. It is well known that for any Boolean function f, $\sqrt{\mathsf{s}(f)} \leq \sqrt{\mathrm{bs}(f)} = O(\widetilde{\deg}(f))$ [NS94]. So a natural step towards proving $\widetilde{\deg}$ composition is: prove a lower bound on $\widetilde{\deg}(f \circ g)$ by $\sqrt{\mathrm{bs}(f)} \cdot \widetilde{\deg}(g)$. We show this result by generalizing the approach of [BBGK18a].

While the techniques used for the composition of R and $\widetilde{\deg}$ are quite different, one can still observe similarities between the classes of outer functions for which the composition of R and $\widetilde{\deg}$ is known to hold respectively. We further expand these similarities, by extending the classes of outer functions for which the composition theorem hold.

Chapter 7

Our result related to approximate degree is the following:

Theorem 1.7 ([CKM+23a]). *For all non-constant (possibly partial)*[6] *Boolean functions* $f : \{0,1\}^n \to \{0,1\}$ *and* $g : \{0,1\}^m \to \{0,1\}$, *we have*

$$\widetilde{\deg}(f \circ g) = \widetilde{\Omega}(\sqrt{\mathrm{bs}(f)} \cdot \widetilde{\deg}(g)).$$

We also prove that if the outer function is strongly $\sqrt{n}$-junta symmetric ("strongly" indicating that the dependence on the Hamming weight is non-trivial) then $\widetilde{\deg}$ composes.

Theorem 1.8 ([CKM+23a]). *Let* $k = O(\sqrt{n})$. *For any strongly k-junta symmetric function* $f : \{0,1\}^n \to \{0,1\}$ *and any Boolean function* $g : \{0,1\}^m \to \{0,1\}$, *we have*

- $\widetilde{\deg}(f \circ g) = \widetilde{\Theta}(\widetilde{\deg}(f) \cdot \widetilde{\deg}(g))$.

Chapter 8

In the case of approximate degree composition, while the lower bound is known when the outer function is symmetric, it doesn't yield a nice lower bound for functions that are recursive compositions of symmetric functions on a smaller number of bits. For example, consider the recursive majority $\mathsf{MAJORITY}_3^d$ that we will obtain by composing $\mathsf{MAJORITY}_3$ with itself d times. Clearly, this is a repeated composition of symmetric functions. But

[6]For definitions of block sensitivity and approximate degree in the context of partial functions, please see Definitions 7.10 and 7.11.

known results, like that of [BBGK18b], do not give a composition theorem for such functions, because the $\log n$ factor (hiding in the $\tilde{\Omega}$ notation) adds up after repeated composition. On the other hand, the approximate degree of $\mathsf{MAJORITY}_n$ is full ($\Theta(n)$), but $\mathsf{MAJORITY}_3^d$ doesn't have full approximate degree, and hence the result of [She12] is not useful in proving the lower bound in this case. Our result related to recursive functions are the following:

Theorem 1.9 ([CKM$^+$23b])**.** *Let $f : \{0,1\}^n \to \{0,1\}$ and $g : \{0,1\}^n \to \{0,1\}$. Then*

$$\widetilde{\deg}(f \circ h^d) = \tilde{\Omega}(\widetilde{\deg}(f)\ \widetilde{\deg}(h^d)),$$
$$\widetilde{\deg}(h^d \circ g) = \tilde{\Omega}(\widetilde{\deg}(h^d)\ \widetilde{\deg}(g)),$$

where h is either $\mathsf{MAJORITY}_3 : \{0,1\}^3 \to \{0,1\}$ or $\mathsf{AND}_2 \circ \mathsf{OR}_2 : \{0,1\}^4 \to \{0,1\}\}$, $d \geq C \log\log n$ for a large enough constant C and $\tilde{\Omega}(\cdot)$ hides $\operatorname{polylog}(n)$ factors.

Using the previous theorem we prove a more general result which holds for almost all the recursive functions:

Theorem 1.10 ([CKM$^+$24])**.** *Let $f : \{0,1\}^n \to \{0,1\}$ and $g : \{0,1\}^m \to \{0,1\}$ be two Boolean functions and $d = \Omega(\log\log n)$. Then,*

$$\widetilde{\deg}(f \circ g) = \Omega\left(\frac{\widetilde{\deg}(f)\widetilde{\deg}(g)}{\operatorname{polylog}(n)}\right),$$

if either of the following conditions hold:

1. *$f = h^d$, for any Boolean function h.*
2. *$g = h^d$, for any Boolean function h with constant arity and not equal to AND or OR.*

In light of the above theorem, understanding the composition of approximate degree when inner function is OR is the central case for making progress towards the general composition question.

Chapter 9

We gave the composition theorem for R when the outer function have full symmetry and we gave the composition theorem for $\widetilde{\deg}$ in terms of block sensitivity.

Theorem 1.11 ([CKM$^+$23a])**.** *Let f be a partial Boolean function on n-bits such that $\mathsf{R}(f) = \Theta(n)$. Then for all partial functions g, we have*

$$\mathsf{R}(f \circ g) = \Omega(\mathsf{R}(f) \cdot \mathsf{R}(g)).$$

On the other hand, Theorem 1.11 implies that R composes for any strongly k-junta symmetric functions (as long as $n-k=\Theta(n)$).

Theorem 1.12 ([CKM$^+$23a]). *Let $n-k=\Theta(n)$. For any strongly k-junta symmetric function $f:\{0,1\}^n\to\{0,1\}$ and any Boolean function $g:\{0,1\}^m\to\{0,1\}$, we have*

- $\mathsf{R}(f\circ g)=\widetilde{\Theta}(\mathsf{R}(f)\cdot\mathsf{R}(g))$.

Apart from symmetry there are some other results that we proved using the properties of symmetry that hold for general classes of functions as we have already mentioned. Here is a table representing our result in the scenario of composition theorem.

	In terms of $\mathrm{bs}(f)$	In terms of arity of f
R	$\mathsf{R}(f\circ g)=\tilde{\Omega}(\mathrm{bs}(f)\cdot\mathsf{R}(g))$ [GJPW18a]	$\mathsf{R}(f\circ g)=\tilde{\Omega}(\mathsf{R}(f)\cdot\mathsf{R}(g))$ when $\mathsf{R}(f)=\Theta(n)$ Theorem 1.11
$\widetilde{\deg}$	$\widetilde{\deg}(f\circ g)=\tilde{\Omega}(\sqrt{\mathrm{bs}(f)}\cdot\widetilde{\deg}(g))$ Theorem 1.7	$\widetilde{\deg}(f\circ g)=\tilde{\Omega}(\widetilde{\deg}(f)\cdot\widetilde{\deg}(g))$ when $\widetilde{\deg}(f)=\Theta(n)$ [She12]

Table 1.1 Composition of R and $\widetilde{\deg}$ depending on the complexity of the outer function in terms of block-sensitivity and arity.

This part is based on:

- On the Composition of Randomized Query Complexity and Approximate Degree
  Author: Sourav Chakraborty, Chandrima Kayal, Rajat Mittal, Manaswi Paraashar, Swagato Sanyal, Nitin Saurabh
  Published in RANDOM 2023, volume 275, pages 63:1–63:23, 2023.
  https://doi.org/10.4230/LIPIcs.APPROX/RANDOM.2023.63

- Approximate degree composition for Recursive functions
  Author: Sourav Chakraborty, Chandrima Kayal, Rajat Mittal, Manaswi Paraashar and Nitin Saurabh
  Published in RANDOM 2024, volume 317, pages 71:1–71:17, 2024.
  https://doi.org/10.4230/LIPIcs.APPROX/RANDOM.2024.71

Chapter 2

Preliminaries

Here we will start by noting some useful definitions of complexity measures and some well-known and important classes of Boolean functions that we will be using throughout. We refer the reader to the survey [BdW02] for an introduction to the complexity of Boolean functions and complexity measures. Several additional complexity measures and their relations among each other can also be found in [BHT17] and [ABK$^+$21b]. Similar to the above references, we define several complexity measures of Boolean functions that are relevant to this thesis.

2.1 Combinatorial Complexity measures of Boolean functions

We refer to [BdW02] and [ABB$^+$17] for a more detailed introduction of the deterministic query model, randomized query model, and quantum query model for Boolean functions.

Definition 2.1 (Deterministic query complexity). *The deterministic query complexity of $f : \{0,1\}^n \to \{0,1\}$, denoted by $\mathsf{D}(f)$, is the worst-case cost of the best deterministic query algorithm computing f.*

Definition 2.2 (Randomized query complexity ($\mathbb{R}$)). *Let $f : \{0,1\}^n \to \{0,1,*\}$ be a (possibly partial) Boolean function. A randomized query algorithm A computes f if $\forall x \in \mathsf{Dom}(f), \Pr[A(x) \neq f(x)] \leq 1/3$, where the probability is over the internal randomness of the algorithm. The cost of the algorithm A, cost(A), is the number of queries made in the worst case over any input as well as internal randomness. The randomized query complexity of f, denoted by $\mathbb{R}(f)$, is defined as*

$$\mathbb{R}(f) = \min_{A \text{ computes } f} cost(A).$$

Definition 2.3 (Zero-error randomized query complexity)**.** *The zero-error randomized query complexity of $f : \{0,1\}^n \to \{0,1\}$, denoted by $\mathsf{R}_0(f)$, is the minimum worst-case expected cost of a randomized query algorithm that computes f to zero-error, that is, on every input x the algorithm should give the correct answer $f(x)$ with probability* 1.

Definition 2.4 (Quantum query complexity)**.** *The quantum query complexity of $f : \{0,1\}^n \to \{0,1\}$, denoted by $\mathsf{Q}(f)$, is the worst-case cost of the best quantum query algorithm computing f to error at most* $1/3$.

Definition 2.5 (Exact quantum query complexity)**.** *The exact quantum query complexity of $f : \{0,1\}^n \to \{0,1\}$, denoted by $\mathsf{Q_E}(f)$, is the minimum number of queries made by a quantum algorithm that outputs $f(x)$ on every input $x \in \{0,1\}^n$ with probability* 1.

All the complexity measures we have defined above is different variant of query complexity measures that are defined depending upon different available sources. There are some combinatorial measures that are closely related with the query measures. In this these we will be closely analyzing such combinatorial measures, which we will define next.

Now, we define the notion of partial assignment that will be used to define several other complexity measures.

Definition 2.6 (Partial assignment)**.** *A partial assignment is a function $p : S \to \{0,1\}$ where $S \subseteq [n]$ and the size of p is $|S|$. For $x \in \{0,1\}^n$ we say $p \subseteq x$ if x is an extension of p, that is the restriction of x to S denoted $x|_S = p$.*

Definition 2.7 (Certificate complexity)**.** *A* 1*-certificate is a partial assignment that forces the value of the function to* 1 *and similarly, a* 0*-certificate is a partial assignment that forces the value of the function to* 0*. The certificate complexity of a function f on x, denoted as $\mathsf{C}(x,f)$, is the size of the smallest $f(x)$-certificate that can be extended to x.*

Also, define 0*-certificate of f as $\mathsf{C}^0(f) = \max\{\mathsf{C}(f,x) : x \in \{0,1\}^n, f(x) = 0\}$ and* 1*-certificate of f as $\mathsf{C}^1(f) = \max\{\mathsf{C}(f,x) : x \in \{0,1\}^n, f(x) = 1\}$. Finally, define the certificate complexity of $f : \{0,1\}^n \to \{0,1\}$, denoted by $\mathsf{C}(f)$, to be $\max\{\mathsf{C}^0(f), \mathsf{C}^1(f)\}$.*

The complexity measure we are going to define next can be seen as a generalization of certificate complexity, known as Unambiguous certificate complexity.

Definition 2.8 (Unambiguous certificate complexity)**.** *For any Boolean function $f : \{0,1\}^n \to \{0,1\}$, a set of partial assignments U is said to form an unambiguous collection of* 0*-certificates for f if*

1. *Each partial assignment in U is a* 0*-certificate (with respect to f)*
2. *For each $x \in f^{-1}(0)$, there is some $p \in U$ with $p \subseteq x$*
3. *No two partial assignments in U are consistent.*

We then define $\mathsf{UC}_0(f)$ to be the minimum value of $\max_{p \in U} |p|$ over all choices of such collections U. We define $\mathsf{UC}_1(f)$ analogously, and set $\mathsf{UC}(f) = \max\{\mathsf{UC}_0(f), \mathsf{UC}_1(f)\}$. We also define the one-sided version, $\mathsf{UC}_{\min}(f) = \min\{\mathsf{UC}_0(f), \mathsf{UC}_1(f)\}$.

Unambiguous certificate complexity is also known as *partition complexity* [AKK16].

Next we define randomized certificate complexity (see [Aar08]).

Definition 2.9 (Randomized certificate complexity)**.** *A randomized verifier for input x is a randomized algorithm that, on input y in the domain of f accepts with probability* 1 *if $y = x$, and rejects with probability at least $\frac{1}{2}$ if $f(y) \neq f(x)$. If $y \neq x$ but $f(y) = f(x)$, the acceptance probability can be arbitrary. The Randomized certificate complexity of f on input x is denoted by $\mathsf{RC}(f,x)$, is the minimum expected number of queries used by a randomized verifier for x. The randomized certificate complexity of f, denoted by $\mathsf{RC}(f)$, is defined as* $\max\{\mathsf{RC}(f,x) : x \in \{0,1\}^n\}$.

Here is two very popular combinatorial complexity measure, one is sensitivity which was first introduced in [CDR86], another is block sensitivity, which were first introduced in [Nis89]. Block sensitivity can be thought of generalisation of sensitivity. Over the decades this has been an attraction in theoretical computer science community to understand the connection between sensitivity and block sensitivity, precisely known as 'sensitivity conjecture'.

Definition 2.10 (Sensitivity)**.** *The sensitivity of f on an input x is defined as the number of bits on which the function is sensitive: $\mathsf{s}(f,x) = |\{i : f(x) \neq f(x^i)\}|$. We define the sensitivity of f as $\mathsf{s}(f) = \max\{\mathsf{s}(f,x) : x \in \{0,1\}^n\}$*

We also define 0*-sensitivity of f as $\mathsf{s}^0(f) = \max\{\mathsf{s}(f,x) : x \in \{0,1\}^n, f(x) = 0\}$, and* 1*-sensitivity of f as $\mathsf{s}^1(f) = \max\{\mathsf{s}(f,x) : x \in \{0,1\}^n, f(x) = 1\}$.*

Definition 2.11 (Block sensitivity)**.** *The block sensitivity $\mathsf{bs}(f,x)$ of a function $f : \{0,1\}^n \to \{0,1\}$ on an input x is the maximum number of disjoint subsets $B_1, B_2, \ldots, B_r$ of $[n]$ such that for all j, $f(x) \neq f(x^{B_j})$, where $x^{B_j} \in \{0,1\}^n$ is the input obtained by flipping the bits of x in* *the coordinates in B_j. The block sensitivity of f, denoted by* $\mathsf{bs}(f)$*, is* $\max\{\mathsf{bs}\{(f,x)\} : x \in \{0,1\}^n\}$.

Block sensitivity can be thought of as a generalization of sensitivity where in the sensitivity case block size is restricted to be at most 1.

Here we will now define fractional block sensitivity where the blocks are allowed to have weights.

Definition 2.12 (Fractional Block Sensitivity)**.** *Let $W(f,x) := \{B \subseteq [n] : f(x^B) \neq f(x)\}$ denote the set of all sensitive blocks for the input $x \in \{0,1\}^n$. The fractional block sensitivity of f at x, denoted by* $\mathsf{fbs}(f,x)$ *is the value of the linear program:*

$$\mathsf{fbs}(f,x) := \max \sum_{w \in W(f,x)} b_w$$

s.t.

$$\forall i \in [n], \sum_{w \in W(f,x) : i \in w} b_w \leq 1$$

and

$$\forall w \in W(f,x),\ b_w \in [0,1].$$

The fractional block sensitivity of f is defined as:

$$\mathsf{fbs}(f) := \max_{x \in \{0,1\}^n} \mathsf{fbs}(f,x).$$

Note that restricting the linear program for $\mathsf{fbs}(f,x)$ to only integral values gives $\mathsf{bs}(f,x)$.

The fractional measures fbs and RC were introduced in [Tal13]. There it was observed that for all $x \in \{0,1\}^n$ we have : $\mathsf{fbs}(f,x) = \mathsf{RC}(f,x)$ since the linear program for Fractional Certificate Complexity and Fractional Block Sensitivity are the primal-dual of each other and are also feasible.

Note that the following inequalities follows from the definitions of the above complexity measures:

Lemma 2.13. $\mathsf{s}(f) \leq \mathsf{bs}(f) \leq \mathsf{fbs}(f) = \mathsf{RC}(f) \leq \mathsf{C}(f)$ *for all Boolean function f.*

Here we are defining *Spectral sensitivity* from [ABK$^+$21b]. For more details about *Spectral sensitivity* and it's updated relationship with other complexity measures we refer [ABK$^+$21b].

Definition 2.14 (Spectral sensitivity)**.** *Let $f : \{0,1\}^n \to \{0,1\}$ be a Boolean function. The* sensitivity graph *of f, $G_f = (V,E)$ is a subgraph of the Boolean hypercube, where $V = \{0,1\}^n$, and $E = \{(x, x \oplus e_i) \in V \times V : i \in [n], f(x) \neq f(x \oplus e_i)\}$, where $x \oplus e_i \in V$ is obtained by flipping the ith bit of x. That is, E is the set of edges between neighbors on the hypercube that have different f-value. Let A_f be the adjacency matrix of the graph G_f. We define the spectral sensitivity of f as the largest eigenvalue of A_f.*

Definition 2.15 (Degree)**.** *A polynomial $p : \mathbb{R}^n \to \mathbb{R}$ represents $f : \{0,1\}^n \to \{0,1\}$ if for all $x \in \{0,1\}^n$, $p(x) = f(x)$. The degree of a Boolean function f, denoted by $\mathsf{deg}(f)$, is the degree of the unique multilinear polynomial that represents f.*

Definition 2.16 (Approximate degree)**.** *A polynomial $p : \mathbb{R}^n \to \mathbb{R}$ approximately represents a function $f : \{0,1\}^n \to \{0,1\}$ if for all $x \in \{0,1\}^n$, $|p(x) - f(x)| \leq \frac{1}{3}$. The approximate degree of a Boolean function f, denoted by $\widetilde{\mathsf{deg}}(f)$, is the minimum degree of a polynomial that approximately represents f.*

There are several works exploring the understanding between different complexity measures, one of the most celebrated result is the following:

Theorem 2.17 ([NS94])**.** $\mathrm{bs}(f) = O(\widetilde{\mathrm{deg}}(f)^2)$ *for all Boolean function f.*

As a consequence, $\mathrm{bs}(f) = O(\mathrm{deg}(f)^2)$, since approximate degree is always at most exact degree.

The following is a known relation between degree and one-sided unambiguous certificate complexity measure ([BHT17]). Note that we can express the function f in terms of polynomial by summing up the polynomials corresponding to the minimal unambiguous certificates.

Observation 2.18 ([BHT17])**.** *For any Boolean function f, $\mathsf{UC}_{\mathsf{min}}(f) \geq \mathsf{deg}(f)$.*

One often tries to understand how the complexity measure of a composed function compares with respect to the measures of the individual functions. The following folklore theorem that we will be using multiple times in our paper. Here the notion of the composition is the following:

Definition 2.19 (Composition of functions)**.** *Let $f : \{0,1\}^{nk} \to \{0,1\}$ and $g : \{0,1\}^m \to \{0,1\}^k$ be two functions. The* composition of f and g, *denoted by $f \circ g : \{0,1\}^{nm} \to \{0,1\}$,* *is defined to be a function on nm bits such that on input $x = (x_1, \dots, x_n) \in \{0,1\}^{nm}$, where each $x_i \in \{0,1\}^m$, $f \circ g(x_1, \dots, x_n) = f(g(x_1), \dots, g(x_n))$. We will refer f as* outer function *and g as* inner function.

Note the above definition of composition is not a standard one.

Theorem 2.20. *Let $f : \{0,1\}^{nk} \to \{0,1\}$ and $g : \{0,1\}^m \to \{0,1\}^k$ be two Boolean functions then*

1. $\mathsf{D}(f \circ g) = \Omega(\mathsf{D}(f)/k)$*, assuming g is an onto function.*
2. $\mathsf{Q}(f \circ g) = \Omega(\mathsf{Q}(f)/k)$*, assuming g is onto.*
3. $\mathsf{R_0}(f \circ g) = O(\mathsf{R_0}(f) \cdot m)$ *and if g is onto then* $\mathsf{R_0}(f \circ g) = O(\mathsf{R_0}(f)/m)$
4. $\mathsf{R}(f \circ g) = O(\mathsf{R}(f) \cdot m)$ *and if g is onto then* $\mathsf{R}(f \circ g) = O(\mathsf{R}(f)/m)$.
5. $\deg(f \circ g) = O(\deg(f) \cdot m)$
6. $\widetilde{\deg}(f \circ g) = O(\widetilde{\deg}(f) \cdot m)$.

If the inner function g is Boolean valued then we can obtain some tighter results for the composed functions.

Theorem 2.21. *Let $f : \{0,1\}^n \to \{0,1\}$ and $g : \{0,1\}^m \to \{0,1\}$ be two Boolean functions then*

1. *([Tal13, Mon14])* $\mathsf{D}(f \circ g) = \Theta(\mathsf{D}(f) \cdot \mathsf{D}(g))$.
2. *([Rei11a, LMR+11, Kim13b])* $\mathsf{Q}(f \circ g) = \Theta(\mathsf{Q}(f) \cdot \mathsf{Q}(g))$.
3. *(folklore)* $\deg(f \circ g) = \Theta(\deg(f) \cdot \deg(g))$.

2.2 Some Boolean functions and their properties

In this section we define some standard functions that are either mentioned in the Table 5.2 or used somewhere in the paper. We also state some of their properties that we need for our proofs. We start by defining some basic Boolean functions.

Definition 2.22. *Define* $\mathsf{PARITY} : \{0,1\}^n \to \{0,1\}$ *to be the* $\mathsf{PARITY}(x_1, \dots, x_n) = \sum x_i mod\, 2$. *We use the notation $\oplus$ to denote* PARITY.

Definition 2.23. *Define* $\mathsf{AND} : \{0,1\}^n \to \{0,1\}$ *to be the* $\mathsf{AND}(x_1, \dots, x_n) = 0$ *if and only if there exists an $i \in [n]$ such that $x_i = 0$. We use the notation $\wedge$ to denote* AND.

Definition 2.24. *Define* $\mathsf{OR} : \{0,1\}^n \to \{0,1\}$ *to be the* $\mathsf{OR}(x_1,\ldots,x_n) = 1$ *if and only if there exists an $i \in [n]$ such that $x_i = 1$. We use the notation $\vee$ to denote* OR.

Definition 2.25. *Define* $\mathsf{MAJORITY} : \{0,1\}^n \to \{0,1\}$ *as* $\mathsf{MAJORITY}(x) = 1$ *if and only if* $|x| > \frac{n}{2}$.

We need the following definition of composing iteratively with itself.

Definition 2.26 (Iterative composition of a function). *Let $f : \{0,1\}^n \to \{0,1\}$ a Boolean function. For $d \in \mathbb{N}$ we define the function $f^d : \{0,1\}^{n^d} \to \{0,1\}$ as follows: if $d = 1$ then $f^d = f$, otherwise*

$$f^d(x_1,\ldots,x_{n^d}) = f\left(f^{d-1}(x_1,\ldots,x_{n^{d-1}}),\ldots,f^{d-1}(x_{n^d-n+1},\ldots,x_{n^d})\right).$$

Definition 2.27. *For $d \in \mathbb{N}$ define* NAND*-tree of depth d as* NAND^d *where* $\mathsf{NAND} : \{0,1\}^2 \to \{0,1\}$ *is defined as:* $\mathsf{NAND}(x_1,x_2) = 0$ *if and only if $x_1 \neq x_2$. We use the notation $\widetilde{\wedge}$-tree to denote* NAND*-tree.*

The now define a function that gives a quadratic separation between sensitivity and block sensitivity.

Definition 2.28 (Rubinstein's function ([Rub95])). *Let $g : \{0,1\}^k \to \{0,1\}$ be such that $g(x) = 1$ iff x contains two consecutive ones and the rest of the bits are* 0. *The Rubinstein's function, denoted by* $\mathsf{RUB} : \{0,1\}^{k^2} \to \{0,1\}$ *is defined to be* $\mathsf{RUB} = \mathsf{OR}_k \circ g$.

Theorem 2.29. *[Rub95] For the Rubinstein's function in Definition 2.28* $\mathsf{s}(\mathsf{RUB}) = k$ *and* $\mathsf{bs}(\mathsf{RUB}) = k^2/2$. *Thus* RUB *witnesses a quadratic gap between sensitivity and block sensitivity.*

[NS94] first introduced a function whose deg is significantly smaller than s or bs. This appears in the footnote in [NW95] that E. Kushilevitz also introduced a similar function with 6 variables which gives slightly better gap between s and deg. Later Ambainis computed $\mathsf{Q_E}$ of that function and gave a separation between $\mathsf{Q_E}$ and D [Amb16]. This function is fully sensitive at all zero input, consequently this gives a separation between $\mathsf{Q_E}$ and s.

Definition 2.30 ([NS94]). *Define* NW *as follows:*

$$\mathsf{NW}(x_1,x_2,x_3) = \begin{cases} 1 \text{ iff } x_i \neq x_j \text{ for some } i,j \in \{1,2,3\} \\ 0 \text{ otherwise.} \end{cases}$$

Now define the d-th iteration NW^d *on $(x_1,x_2,\ldots,x_{3^d})$ as Definition 2.26 where $d \in \mathbb{N}$.*

Definition 2.31 (Kushilevitz's function). *Define* K *as follows:*

$$\mathsf{K}(x_1,x_2,x_3,x_4,x_5,x_6) = \Sigma x_i + \Sigma x_i y_i + x_1x_3x_4 + x_1x_4x_5 + x_1x_2x_5 + x_2x_3x_4 + x_2x_3x_5 + x_1x_2x_6 \\ + x_1x_3x_6 + x_2x_4x_6 + x_3x_5x_6 + x_4x_5x_6.$$

Now define the d-th iteration K^d *on* $(x_1,x_2,\ldots,x_{6^d})$ *as Definition 2.26 where* $d \in \mathbb{N}$.

Next we will describe two example that was introduced in [GSS16] and gives the separation between C vs. RC and RC vs. bs respectively . They also have introduces some new complexity measures for iterative version of a function and how to use them to get the critical measure between two complexity measures. For more details we refer to [GSS16].

Definition 2.32 ([GSS16]). *Let n be an even perfect square, let $k = 2\sqrt{n}$ and $d = \sqrt{n}$. Divide the n indices of the input into n/k disjoint blocks. Define* $\mathsf{GSS}_1 : \{0,1\}^n \to \{0,1\}$ *as follows:* $\mathsf{GSS}_1(x) = 1$ *if and only if* $|x| \geq d$ *and all the* 1*'s in x are in a single group. Define* GSS_1^d *with* $\mathsf{GSS}_1^1 = f$.

Definition 2.33 ([GSS16]). *Define* $\mathsf{GSS}_2 : \{0,1\}^n \to \{0,1\}$ *where n is of the form* $\binom{t}{2}$ *for some integer t. Identify the input bits of* GSS_2 *with the edges of the complete graph K_t. An input $x \in \{0,1\}^n$ induces a subgraph of K_t consisting of edges assigned* 1 *by x. The function* $\mathsf{GSS}_2(x)$ *is defined to be* 1 *iff the subgraph induces by x has a star graph.*

Definition 2.34. *For* $\Sigma = [n^k]$ *the function* $\mathsf{k\text{-}sum} : \Sigma^n \to \{0,1\}$ *is defined as follows: on input* $x_1,x_2,\ldots,x_n \in \Sigma$, *if there exists k element* $x_{i_1},\ldots,x_{i_k}$, $i_1,\ldots,i_k \in [n]$, *that sums to* $0 \pmod{|\Sigma|}$ *then output* 1, *otherwise output* 0.

Theorem 2.35 ([ABK16, BS13]). *For the function* $\mathsf{k\text{-}sum} : \Sigma^n \to \{0,1\}$, *if* $|\Sigma| \geq 2\binom{n}{k}$ *then*

$$\mathsf{Q}(\mathsf{k\text{-}sum}) = \Omega(n^{k/(k+1)}/\sqrt{k}).$$

Next, we define the cheat sheet version of a function from [ABK16].

Definition 2.36. *We define cheat sheet version of f as follows: the input to f_{CS} consist of* $\log n$ *inputs to f , each of size n , followed by n blocks of bits of size* $C(f) \times \log n$ *each. Let us denote the input to f_{CS} as* $X = (x_1,x_2,\ldots,x_{\log n},Y_1,Y_2,...,Y_n)$, *where x_i is an input to f , and the Y_i are the aforementioned cells of size* $C(f) \times \log n$. *The first part* $x_1,x_2,\ldots,x_{\log n}$ *of the string, we call them as input section and the rest of the part we call as certificate section of the whole input. Define* $f_{CS} : \{0,1\}^{n\times\log n + n\times(C(f)\log n} \to \{0,1\}$ *to be* 1 *if and only if the following conditions hold:*

- *For all i, x_i is in the domain of f. If this condition is satisfied, let l be the positive integer corresponding to the binary string $(f(x_1), f(x_2), ..., f(x_{\log n}))$.*
- *Y_l certifies that all x_i are in the domain of f and that l equals the binary string formed by their output values, $(f(x_1), f(x_2), ..., f(x_{\log n}))$.*

Finally, we present the pointer functions and its variants introduced in [ABB$^+$17]. They are used for demonstrate separation between several complexity complexity measures like *deterministic query complexity*, *Randomized query complexity*, *Quantum query complexity* etc. These functions was originally motivated from [GPW18] function. The functions that we construct for many of our theorems is a composition function whose outer function is these pointer function, or a slight variant of these. In the next section we present the formal definition of pointer functions.

Definition 2.37 (Pointer matrix over Σ)**.** *For $m, n \in \mathbb{N}$, let M be a $(m \times n)$ matrix with m rows and n columns. We refer to each of the $m \times n$ entries of M as* cells. *Each cell of the matrix is from a alphabet set Σ where $\Sigma = \{0,1\} \times \widetilde{P} \times \widetilde{P} \times \widetilde{P}$ and $\widetilde{P} = \{(i,j) | i \in [m], j \in [n]\} \cup \{\bot\}$. We call $\widetilde{P}$ as set of pointers where, pointers of the form $\{(i,j) | i \in [m], j \in [n]\}$ pointing to the cell (i,j) and $\bot$ is the null pointer. Hence,each entry $x_{(i,j)}$ of the matrix M is a 4-tuple from Σ. The elements of the 4-tuple we refer as* value, left pointer, right pointer *and* back pointer *respectively and denote by Value$(x_{(i,j)})$, LPointer$(x_{(i,j)})$, RPointer$(x_{(i,j)})$ and BPointer$(x_{(i,j)})$ respectively where Value $\in \{0,1\}$, LPointer,RPointer,BPointer $\in \widetilde{P}$. We call this type of matrix as* pointer matrix *and denote by $\Sigma^{n \times n}$.*

A special case of the pointer-matrix, which we call Type$_1$ pointer matrix over Σ, *is when for each cell of M, BPointer $\in \{[n] \cup \bot\}$ that is backpointers are pointing to the columns of the matrix.*

Also, in general when, BPointer $\in \{(i,j) | i \in [m], j \in [n]\} \cup \{\bot\}$, we call it a Type$_2$ pointer matrix over Σ.

Now we will define some additional properties of the domain that we need to define [ABB$^+$17] function.

Definition 2.38 (Pointer matrix with marked column)**.** *Let M be an $m \times n$* pointer-matrix *over Σ. A column $j \in [n]$ of M is defined to be a* marked column *if there exists exactly one cell (i,j), $i \in [m]$, in that column with entry $x_{(i,j)}$ such that $x_{(i,j)} \neq (1, \bot, \bot, \bot)$ and every other* cl *in that column is of the form $(1, \bot, \bot, \bot)$. The* cl *(i,j) is defined to be the* special element *of the* marked column *j.*

Let n be a power of 2. Let T be a rooted, directed and balanced binary tree with n-leaves and $(n-1)$ internal vertices. We will use the following notations that will be used in defining some functions formally.

Notation 2.39. *Let n be a power of 2. Let T be a rooted, directed and balanced binary tree with n-leaves and $(n-1)$ internal vertices. Labels the edges of T as follows: the outgoing edges from each node are labeled by either $left$ or right. The leaves of the tree are labeled by the elements of $[n]$ from left to right, with each label used exactly once. For each leaf $j \in [n]$ of the tree, the path from the root to the leaf j defines a sequence of $left$ and right of length $O(\log n)$, which we denote by $T(j)$.*

When n is not a power of 2, choose the largest $k \in \mathbb{N}$ such that $2^k \leq n$, consider a complete balanced tree with 2^k leaves and add a pair of child node to to each $n-2^k$ leaves starting from left. Define $T(j)$ as before.

More details about *partial assignment* and *certificates* can be found in Definition 2.7. Now we are ready to describe the *Variant 1* of [ABB$^+$17] function.

Definition 2.40 (Variant 1 [ABB$^+$17])**.** *Let $\Sigma^{m\times n}$ be a* Type_1 pointer matrix *where BPointer is a pointer of the form $\{j | j \in [n]\}$ that points to other column and LPointer, RPointer are as usual points to other cell. Define $\mathsf{A1}_{(m,n)} : \Sigma^{m\times n} \to \{0,1\}$ on a* Type_1 pointer matrix *such that for all $x = (x_{i,j}) \in \Sigma^{m\times n}$, the function $\mathsf{A1}_{(m,n)}(x_{i,j})$ evaluates to 1 if and only if it has a* 1- cell certificate *of the following form:*

1. *there exists exactly one* marked column *$j^\star$ in M,*
2. *There is a special cell, say $(i^\star, j^\star)$ which we call the special element in the the* marked column *$j^\star$ and there is a balanced binary tree T rooted at the special cell,*
3. *for each non-marked column $j \in [n] \setminus \{j^\star\}$ there exist a* cell *l_j such that Value$(l_j) = 0$ and BPointer$(l_j) = j^\star$ where l_j is the end of the path that starts at the* special element *and follows the pointers LPointer and RPointer as specified by the sequence $T(j)$. l_j exists for all $j \in [n] \setminus \{j^\star\}$ i.e. no pointer on the path is $\perp$. We refer l_j as the leaves of the tree.*

The above function achieves the separation between D vs. R_0 and D vs. Q for $m = 2n$.

There are other variants of Pointer functions which were used to give optimal separations between various complexity measures [ABB$^+$17]. A more details can be found in Chapter 4.

Part I

Relations and separations between complexity measures for functions with symmetry

Chapter 3

Classes of Boolean functions with Symmetry

3.1 Symmetric Boolean Function

Definition 3.1 (Symmetric Boolean function)**.** *Any Boolean function $f : \{0,1\}^n \to \{0,1\}$ whose output value is invariant under all possible permutations on the input string is a Symmetric Boolean function.*

Examples of Symmetric Functions and their complexity measures:

Symmetric function	**Hamming weight for** (1-input)
$\mathsf{AND} : \{0,1\}^n \to \{0,1\}$	$\lvert x\rvert = n$
$\mathsf{OR} : \{0,1\}^n \to \{0,1\}$	$\lvert x\rvert \geq 1$
$\mathsf{XOR} : \{0,1\}^n \to \{0,1\}$	$\lvert x\rvert$ is even
$\mathsf{MAJORITY} : \{0,1\}^n \to \{0,1\}$	$\lvert x\rvert > \frac{n}{2}$
k-Threshold function	$\lvert x\rvert \geq k$

Table 3.1 Examples of Symmetric functions

Note 3.2. *Symmetric Boolean function can be defined in terms of Hamming weight of the input string. So, an alternative definition of a symmetric Boolean function is the following:*

A function that depends only on the Hamming weight of the input is a Symmetric Boolean function.

3.2 Properties of Symmetric Boolean functions

Since symmetric Boolean function can be visualize as a function $f : \{0,1,\dots,n\} \to \{0,1\}$ on $\{0,1,\dots,n\}$ there are comparatively easy to study compared to general Boolean functions. Here we are presenting some of the important properties of Symmetric Boolean functions.

Proposition 3.3. *If f is non-constant and symmetric Boolean function,then $s(f) \geq \lceil \frac{n+1}{2} \rceil$.*

Proof. Let $f : \{0,1\}^n \to \{0,1\}$ be a non-constant symmetric Boolean function. Since f depends only on the weight of the inputs, let $f(x) = 1$ if $wt(x) = k$ and $f(x) = 0$ if $wt(x) = (k-1)$. Then for inputs with $wt(x) = k$, $\mathsf{s}(f,x) \geq k$ and for inputs with $wt(x) = k-1$, $\mathsf{s}(f,x) \geq (n-(k-1))$. So, $\mathsf{s}(f) \geq max\{k,(n-(k-1))\} = \lceil \frac{n+1}{2} \rceil$. □

Note 3.4. *This proposition is tight as sensitivity of $\lfloor \frac{n}{2} \rfloor$-Threshold function is $\lceil \frac{n+1}{2} \rceil$*

Some well-known symmetric functions and their complexity measures:

Example of functions	D	R	$\widetilde{\deg}$	Q
AND	N	N	$\sqrt{N}$	$\sqrt{N}$
OR	N	N	$\sqrt{N}$	$\sqrt{N}$
MAJORITY	N	N	N	N
PARITY	N	N	N	N

Table 3.2 Example of complexity measures for some well-known Symmetric functions

Observation 3.5. *From Proposition 3.3 it follows that for non-constant symmetric Boolean Function $s(f) = \Theta(n)$. Consequently, $bs(f) = \Theta(n)$ and $C(f) = \Theta(n)$. These bounds are tight because there exists a non-constant symmetric Boolean function reaching these bounds.*

Since the Symmetric function can be viewed as real values polynomial with domain points being integer in $[n]$ there are various exploration in terms of degree and representation of symmetric Boolean functions. Like other complexity measures degree of a Symmetric function is also high.

Proposition 3.6. *If f is a non-constant symmetric Boolean function $deg(f) \geq (\frac{n}{2})$.*

Proof. If f is a non-constant Boolean function then either f or $\neg f$ have at least $\Omega(n)$ number of roots in $[n]$. Since the degree of f and $\neg f$ is same, it follows that $\deg(f) \geq \frac{n}{2}$. □

Precisely, [vZGR97] proved the following:

Complexity Measure	Upper Bound	Lower Bound
D	N PARITY	N PARITY
R_0	N PARITY	N PARITY
R	N PARITY	N PARITY
C	N PARITY	N PARITY
RC	N PARITY	N PARITY
bs	N PARITY	N PARITY
s	N PARITY	N PARITY
λ	N PARITY	$\sqrt{N}$ AND
Q_E	N PARITY	N PARITY
deg	N PARITY	N PARITY
Q	N PARITY	$\sqrt{N}$ AND
$\widetilde{\deg}$	N PARITY	$\sqrt{N}$ AND

Table 3.3 Upper and Lower bound on Complexity measures for Symmetric functions

Theorem 3.7 ([vZGR97]). *If f is non-constant symmetric Boolean function then* $\deg(f) = n - O(n^{0.548})$. *If $n+1$ is a prime, then* $\deg(f) = n$.

Conjecture 3.8 ([vZGR97]). *Gathen and Roche conjecture that* $\deg(f) = n - O(1)$ *for all non-constant symmetric Boolean function.*

The best-known result so far is that deg$(f) = n - \sqrt{n}$.

For the approximate degree of Boolean function there is a fundamental result in this area by [Pat92] is the following:

Theorem 3.9 ([Pat92]). *For any non-constant Symmetric Boolean function,* $\widetilde{\deg}(f) = \Theta(\sqrt{nk})$ *where n is the arity of the Boolean function f and $k \leq \frac{n}{2}$ (otherwise let k be* *defined to be $n-k$) be the closest integer to $\frac{n}{2}$ such that f gives different output value on the string of Hamming weight k and $k+1$.*

Note that the above result is tight and completely characterizes the approximate degree for the classes of Symmetric functions.

Note that since approximate degree gives a lower bound on quantum query complexity of a Boolean function, we have the following lower bound for symmetric functions:

Corollary 3.10. *For any non-constant Symmetric Boolean function f on n bits, $\mathsf{Q}(f) = \Omega(\sqrt{N})$ where n is the arity of the Boolean function f and $k \leq \frac{n}{2}$ (otherwise let k be defined to be $n-k$) be the closest integer to $\frac{n}{2}$ such that f gives different output value on the string of Hamming weight k and $k+1$.*

We also have a matching upper bound for quantum query complexity of symmetric Boolean function by [dW08].

From the above corollary we have the following 'relation' between the degree and quantum query complexity of Boolean functions:

Theorem 3.11. *For non-constant Symmetric Boolean function* $\mathsf{Q}(f) \geq (\deg(f))^2$.

3.3 Separations between Complexity Measures for Symmetric Boolean Function

Table 3.4 best-known separations between combinatorial measures for Symmetric Boolean functions.

$\times$	D	R_0	R	C	RC	bs	s	λ	Q_E	deg	Q	$\widetilde{deg}$
D	$\times$	$1:1$ AND	$1:1$ AND	$1:1$ AND	$1:1$ AND	$1:1$ AND	$1:1$ AND	$2:2$ AND	$1:1$ AND	$1:1$ AND	$2:2$ AND	$2,2$ AND
R_0	$1:1$ AND	$\times$	$1:1$ AND	$1:1$ AND	$1:1$ AND	$1:1$ AND	$1:1$ AND	$2:2$ AND	$1:1$ AND	$1:1$ AND	$2:2$ AND	$2:2$ AND
R	$1:1$ AND	$1:1$ AND	$\times$	$1:1$	$1:1$ AND	$1:1$ AND	$1:1$ AND	$2:2$ AND	$1:1$ AND	$1:1$ AND	$2:2$ AND	$2:2$ AND
C	$1:1$ AND	$1:1$ AND	$1:1$ AND	$\times$	$1:1$ AND	$1:1$ AND	$1:1$ AND	$2:2$ AND	$1:1$ AND	$1:1$ AND	$2:2$ AND	$2,2$ AND
RC	$1:1$ AND	$1:1$ AND	$1:1$ AND	$1:1$ AND	$\times$	$1:1$ AND	$1:1$ AND	$2,2$ AND	$1:1$ AND	$1:1$ AND	$2,2$ AND	$2,2$ AND
bs	$1:1$ AND	$1:1$ AND	$1:1$ AND	$1:1$ AND	$1:1$ AND	$\times$	$1:1$ AND	$2,2$ AND	$1:1$ AND	$1:1$ AND	$2,2$ AND	$2,2$ AND
s	$1:1$ AND	$1:1$ AND	$1:1$ AND	$1:1$ AND	$1:1$ AND	$1:1$ AND	$\times$	$2,2$ AND	$1:1$ AND	$1:1$ AND	$2,2$ AND	$2,2$ AND
λ	$1:1$ AND	$1:1$ AND	$1:1$ AND	$1:1$ AND	$1:1$ AND	$1:1$ AND	$1,1$ AND	$\times$	$1,1$ AND	$1,1$ AND	$1,1$ AND	$2,2$ AND
Q_E	$1:1$ AND	$1:1$ AND	$1:1$ AND	$1:1$ AND	$1:1$ AND	$1:1$ AND	$1:1$ AND	$1:1$ AND	$\times$	$1:1$ AND	$2,2$ AND	$2:2$ AND
deg	$1:1$ AND	$1:1$ AND	$1:1$ AND	$1:1$ AND	$1:1$ AND	$1:1$ AND	$1:1$ AND	$1:1$ AND	$1:1$ AND	$\times$	$2,2$ AND	$2,2$ AND
Q	$1:1$ AND	$1:1$ AND	$1:1$ AND	$1:1$ AND	$1:1$ AND	$1:1$ AND	$1:1$ AND	$1:1$ AND	$1,1$ AND	$1:1$ AND	$\times$	$2:2$ AND
$\widetilde{deg}$	$1:1$ AND	$1:1$ AND	$1:1$ AND	$1:1$ AND	$1:1$ AND	$1:1$ AND	$1:1$ AND	$1:1$ AND	$1,1$ AND	$1,1$ AND	$1,1$ AND	$\times$

(1 Entry $a;b$ in row A and column B represents: for any *transitive* function f, $A(f) = O(B(f))^{b+o(1)}$, and there exists a *Symmetric* Boolean function g such that $A(g) = \Omega(B(g))^a$.

Entry $a;b$ in row A and column B represents: for any *transitive* function f, $A(f) = O(B(f))^{b+o(1)}$, and there exists a *transitive* function g such that $A(g) = \Omega(B(g))^a$.

Chapter 4

Separation results via Transitive Pointer function

4.1 Introduction

For a Boolean function $f : \{0,1\}^n \to \{0,1\}$ what is the relationship between its various combinatorial measures, like deterministic query complexity ($\mathsf{D}(f)$), bounded-error randomized and quantum query complexity ($\mathsf{R}(f)$ and $\mathsf{Q}(f)$ respectively), zero -randomized query complexity ($\mathsf{R}_0(f)$), exact quantum query complexity ($\mathsf{Q}_\mathsf{E}(f)$), sensitivity ($\mathsf{s}(f)$), block sensitivity ($\mathsf{bs}(f)$), certificate complexity ($\mathsf{C}(f)$), randomized certificate complexity ($\mathsf{RC}(f)$), unambiguous certificate complexity ($\mathsf{UC}(f)$), degree ($\deg(f)$), approximate degree ($\widetilde{\deg}(f)$) and spectral sensitivity ($\lambda(f)$)[1]? For over three decades, understanding the relationships between these measures has been an active area of research in computational complexity theory. These combinatorial measures have applications in many other areas of theoretical computer science, and thus the above question takes a central position.

In the last couple of years, some of the more celebrated conjectures have been answered - like the quadratic relation between sensitivity and degree of Boolean functions [Hua19]. We refer the reader to the survey [BdW02] for an introduction to this area.

Understanding the relationship between various combinatorial measures involves two parts:

- Relationships - proving that one measure is upper bounded by a function of another measure. For example, for any Boolean function f, $\deg(f) \le \mathsf{s}(f)^2$ and $\mathsf{D}(f) \le \mathsf{R}(f)^2$.

[1]We provide formal definitions of the measures used in this Chapter in Section 2.1.

- Separations - constructing functions that demonstrates separation between two measures. For example, there exists a Boolean function f with $\deg(f) \geq \mathsf{s}(f)^2$. Also there exists another Boolean function g with $\mathsf{D}(g) \geq \mathsf{R}(g)^2$.

Obtaining tight bounds between pairs of combinatorial measures - that is, when the relationship and the separation results match - is the holy grail of this area of research. The current best known results for different pairs of functions have been nicely compiled in [ABK$^+$21b].

For special classes of Booleans functions the relationships and the separation results might be different than that of general Boolean functions. For example, while it is known that there exists f such that $\mathsf{bs}(f) = \Theta(\mathsf{s}(f)^2)$ [Rub95], for a symmetric function a more tighter result is known, $\mathsf{bs}(f) = \Theta(\mathsf{s}(f))$. The best known 'relationship' of $\mathsf{bs}(f)$ for a general Boolean functions is $\mathsf{s}(f)^4$ [Hua19]. How the various measures behave for different classes of functions has been studied since the dawn of this area of research.

Transitive Functions: One of the most well-studied classes of Boolean functions is that of the transitive functions. A function $f : \{0,1\}^n \to \{0,1\}$ is transitive if there is a transitive group $G \leq \mathsf{S}_n$ such that the function value remains unchanged even after the indices of the input is acted upon by a permutation from G. Note that, when $G = \mathsf{S}_n$ then the function is symmetric. Transitive functions (also called "weakly symmetric" functions) has been studied extensively in the context of various complexity measure. This is because symmetry is a natural measure of the complexity of a Boolean function. It is expected that functions with more symmetry must have less variation among the various combinatorial measures. A recent work [BCG$^+$20] has studied the functions under various types of symmetry in terms of quantum speedup. So, studying functions in terms of symmetry is important in various aspects.

For example, for symmetric functions, where the transitive group is S_n, most of the combinatorial measures become the same up to a constant [2]. Another example of transitive functions is the graph properties. The input is the adjacency matrix, and the transitive group is the graph isomorphism group acting on the bits of the adjacency matrix. [Tur84, Sun11, LS17, GMSZ13] tried to obtain tight bounds on the relationship between sensitivity and block sensitivity for graph properties. They also tried to answer how low can sensitivity and block sensitivity go for graph properties?

In papers like [SYZ04, Cha11, Sun07, Dru11] it has been studied how low can the combinatorial measures go for transitive functions. The behavior of transitive functions can

[2]There are still open problems on the tightness of the constants.

be very different from general Boolean functions. For example, while it is known that there are Boolean functions for which the sensitivity is as low as $\Theta(\log n)$ where n is the number of effective variables[3], it is known (from [Sun07] and [Hua19]) that if f is a transitive function on n effective variables then its sensitivity $s(f)$ is at least $\Omega(n^{1/12})$[4]. Similar behavior can be observed in other measures too. For example, it is easy to see that for a transitive function the certificate complexity is $\Omega(\sqrt{n})$, while the certificate complexity for a general Boolean function can be as low as $O(\log n)$. In Table 5.1 we summarize the best-known separations of the combinatorial measures for transitive functions. A natural related question is:

What is the tight relationship between various pairs of combinatorial measures for transitive functions?

By definition, the known 'relationship' results for general functions hold for transitive functions. But tighter 'relationship' results may be obtained for transitive functions. On the other hand, the existing 'separations' doesn't extend easily since the example used to demonstrate separation between certain pairs of measures may not be transitive. Some of the most celebrated examples are not transitive. For example some of the celebrated function construction like the pointer function [ABB$^+$17], used for demonstrating tight separations between various pairs like $\mathsf{D}(f)$ and $\mathsf{R}_0(f)$, are not transitive. Similarly, the functions constructed using the cheat sheet techniques [ABK16] used for separation between quantum query complexity and degree, or approximate degree are not transitive. Constructing transitive functions that demonstrate tight separations between various pairs of combinatorial measures is very challenging.

4.2 Our Results

We try to answer the above question for various pairs of measures. More precisely, our main contribution is to construct transitive functions that have similar complexity measures as the *pointer functions*. Hence for those pairs of measures where pointer functions can demonstrate separation for general functions, we prove that similar separation can also be demonstrated by transitive functions.

Our results and the current known relations between various pairs of complexity measures of transitive functions are compiled in Table 5.2. This table is along the lines of the table

[3]A variable is effective if the function is dependent on it.

[4]It is conjectured that the sensitivity of a transitive function is $\Omega(n^{1/3})$.

in [ABK$^+$21b] where the best-known relations between various complexity measures of genral Boolean functions were presented.

Deterministic query complexity and zero-error randomized query complexity are two of the most basic measures and yet the tight relation between these measures was not known until recently. In [Sni85] they showed that for the “balanced NAND-tree” function, $\widetilde{\wedge}$-tree, $\mathsf{D}(\widetilde{\wedge}\text{-tree}) \geq \mathsf{R}_0(\widetilde{\wedge}\text{-tree})^{1.33}$. Although the function $\widetilde{\wedge}$-tree is transitive, the best known ’relationship’ was quadratic, that is for all Boolean function f, $\mathsf{D}(f) = O(\mathsf{R}_0(f)^2)$. In [ABB$^+$17] a new function, $\mathsf{A1}$, was constructed for which deterministic query complexity and zero-error randomized query complexity can have a quadratic separation between them, and this matched the known ’relationship’ results. The function in [ABB$^+$17] was a variant of the pointer functions - a class of functions introduced by [GPW18] that has found extensive usage in showing separations between various complexity measures of Boolean functions. The function, $\mathsf{A1}$, also gave (the current best known) separations between deterministic query complexity and other measures like quantum query complexity and degree. But the function $\mathsf{A1}$ is not transitive. Using the $\mathsf{A1}$ function we construct a transitive function that demonstrates a similar gap between deterministic query complexity and zero-error randomized query complexity, quantum query complexity, and degree.

Theorem 4.1 (Restatement of Theorem 1.4)**.** *There exists a transitive function $F_{1.4}$ such that*

$$\mathsf{D}(F_{1.4}) = \widetilde{\Omega}(\mathsf{Q}(F_{1.4})^4), \qquad \mathsf{D}(F_{1.4}) = \widetilde{\Omega}(\mathsf{R}_0(F_{1.4})^2), \qquad \mathsf{D}(F_{1.4}) = \widetilde{\Omega}(\deg(F_{1.4})^2).$$

[ABB$^+$17] considered two more variants of the pointer function. The first of these variants gives separation between zero-error randomized query complexity and other measures like one-sided randomized query complexity, exact quantum query complexity and degree. The second variant gives separation between randomized query complexity and other measures like approximate degree and degree. Once again both these functions were not transitive. Inspired from these function we construct transitive versions that demonstrate similar separation for transitive functions as that of general functions. The construction of these functions, though more complicated and involved, are similar in flavor to that of $F_{1.4}$.

Consequently, we have the following theorems

Theorem 4.2. *There exists a transitive function $F_{4.2}$ such that*

$$\mathsf{R}_0(F_{4.2}) = \widetilde{\Omega}(\mathsf{R}(F_{4.2})^2), \qquad \mathsf{R}_0(F_{4.2}) = \widetilde{\Omega}(\mathsf{Q}_\mathsf{E}(F_{4.2})^2), \qquad \mathsf{R}_0(F_{4.2}) = \widetilde{\Omega}(\deg(F_{4.2})^2).$$

Theorem 4.3. *There exists a transitive function $F_{4.3}$ such that*

$$\mathsf{R}(F_{4.3}) = \widetilde{\Omega}(\widetilde{\deg}(F_{4.3})^4), \qquad \mathsf{R}(F_{4.3}) = \widetilde{\Omega}(\deg(F_{4.3})^2).$$

Theorem 4.4. *There exists a transitive function $F_{4.4}$ such that $\mathsf{R}_0(F_{4.4}) = \widetilde{\Omega}(\mathsf{Q}(F_{4.4})^3)$.*

Theorem 4.5. *There exists a transitive function $F_{4.5}$ such that $\mathsf{R}(F_{4.5}) = \widetilde{\Omega}(\mathsf{Q_E}(F_{4.5})^{1.5})$.*

4.2.1 Notations and basic definitions

We use $[n]$ to denote the set $\{1,\ldots,n\}$. $\{0,1\}^n$ denotes the set of all n-bit binary strings. For any $X \in \{0,1\}^n$ the Hamming Weight of X (denoted $|X|$) will refer to the number of 1 in X. 0^n and 1^n denotes all 0's string of n-bit and all 1's string of n-bit, respectively.

We denote by S_n the set of all permutations on $[n]$. Given an element $\sigma \in \mathsf{S}_n$ and a n-bit string $x_1,\ldots,x_n \in \{0,1\}^n$ we denote by $\sigma[x_1,\ldots,x_n]$ the string obtained by permuting the indices according to σ. That is $\sigma[x_1,\ldots,x_n] = x_{\sigma(1)},\ldots,x_{\sigma(n)}$. This is also called the action of σ on the $x_1,\ldots,x_n$.

Following are a couple of interesting elements of S_n that will be used in this paper.

Definition 4.6. *For any $n = 2k$ the* flip *swaps $(2i-1)$ and $2i$ for all $1 \le i \le k$. The permutation* $\mathsf{Swap}_{\frac{1}{2}}$ *swaps i with $(k+i)$, for all $1 \le i \le k$. That is,*

$$\mathsf{flip} = (1,2)(3,4)\ldots(n-1,n) \qquad \& \qquad \mathsf{Swap}_{\frac{1}{2}}[x_1,\ldots,x_{2k}] = x_{k+1},\ldots,x_{2k},x_1\ldots,x_k.$$

Every integer $\ell \in [n]$ has the canonical $\log n$ bit string representation. However the number of 1's and 0's in such a representation is not same for all $\ell \in [n]$. The following representation of $\ell \in [n]$ ensures that for all $\ell \in [n]$ the encoding has same Hamming weight.

Definition 4.7 (Balanced binary representation)**.** *For any $\ell \in [n]$, let $\ell_1,\ldots,\ell_{\log n}$ be the binary representation of the number ℓ where $\ell_i \in \{0,1\}$ for all i. Replacing* 1 *by* 10 *and* 0 *by* 01 *in the binary representation of ℓ, we get a $2\log n$-bit unique representation, which we call* Balanced binary representation *of ℓ and denote as $bb(\ell)$.*

In this paper all the functions considered are of form $F : \{0,1\}^n \to \{0,1\}^k$. By Boolean functions we would mean a Boolean valued function that is of the form $f : \{0,1\}^n \to \{0,1\}$.

An input to a function $F : \{0,1\}^n \to \{0,1\}^k$ is a n-bit string but also the input can be thought of as different objects. For example, if the $n = NM$ then the input may be thought of as a $(N \times M)$-matrix with Boolean values. It may also be thought of as a $(M \times N)$-matrix.

If $\Sigma = \{0,1\}^k$ then $\Sigma^{(n\times m)}$ denotes an $(n \times m)$-matrix with an element of Σ (that is, a k-bit string) stored in each cell of the matrix. Note that $\Sigma^{(n\times m)}$ is actually $\{0,1\}^{mnk}$. Thus, a function $F : \Sigma^{(n\times m)} \to \{0,1\}$ is actually a Boolean function from a $\{0,1\}^{nmk}$ to $\{0,1\}$, where we think of the input as an $(n \times m)$-matrix over the alphabet Σ.

One particular nomenclature that we use in this paper is that of 1-cell certificate.

Definition 4.8 (1-cell certificate)**.** *Given a function $f : \Sigma^{(n\times m)} \to \{0,1\}$ (where $\Sigma = \{0,1\}^k$) the* 1*-cell certificate is a partial assignment to the cells which forces the value of the function to* 1*. So a* 1*-cell certificate is of the form $(\Sigma \cup \{*\})^{(n\times m)}$. Note that here we assume that the contents in any cell is either empty or a proper element of Σ (and not a partial k-bit string).*

Another notation that is often used is the following:

Notation 4.9. *If $A \leq \mathsf{S}_n$ and $B \leq \mathsf{S}_m$ are groups on $[n]$ and $[m]$ then the group $A \times B$ act on the cells on the matrix. Thus for any $(\sigma, \sigma') \in A \times B$ and a $M \in \Sigma^{(n\times m)}$ by $(\sigma, \sigma')[M]$ we would mean the permutation on the cell of M according to (σ, σ') and move the contains in the cells accordingly. Note that the relative position of bits within the contents in each cell is not touched.*

Finally the following observation proves that composition of transitive functions is also a transitive function.

Observation 4.10. *Let $f : \{0,1\}^n \to \{0,1\}$ and $g : \{0,1\}^m \to \{0,1\}$ be transitive functions. Then $f \circ g : \{0,1\}^{nm} \to \{0,1\}$ is also transitive.*

Proof. Let $T_f \subseteq \mathsf{S}_n$ and $T_g \subseteq \mathsf{S}_m$ be the transitive groups corresponding to f and g, respectively. On input $x = (X_1, \ldots, X_n)$, $X_i \in \{0,1\}^m$ for $i \in [n]$, the function $f \circ g$ is invariant under the action of the group $T_f \wr T_g$ - the wreath product of the T_f with T_g. The group $T_f \wr T_g$ acts on the input string through the following permutations:

1. any permutation $\pi \in T_f$ acting on indices $\{1, \ldots, n\}$ or
2. any permutations $(\sigma_1, \ldots, \sigma_n) \in (T_g)^n$ acting on $X_1, \ldots, X_n$ i.e. $(\sigma_1, \ldots, \sigma_n)$ sends $X_1, \ldots, X_n$ to $\sigma_1(X_1), \ldots, \sigma_n(X_n)$.

$\square$

Over the years a number of interesting Boolean functions has been constructed to demonstrate differences between various measures of Boolean functions. Some of the functions have been referred to in the Table 5.2. We have described various important Boolean functions in the Preliminaries 2.2.

4.2.2 Transitive Groups and Transitive Functions

Example The central objects in this paper are transitive Boolean functions. We first define transitive groups.

Definition 4.11. *A group $G \leq \mathsf{S}_n$ is transitive if for all $i, j \in [n]$ there exists a $\sigma \in G$ such that $\sigma(i) = j$.*

Definition 4.12. *For $f : A^n \to \{0,1\}$ and $G \leq \mathsf{S}_n$ we say f is invariant under the action of G, if for all $\alpha_1, \ldots, \alpha_n \in A$.*

$$f(\alpha_1, \ldots, \alpha_n) = f(\alpha_{\sigma(1)}, \ldots, \alpha_{\sigma(n)}).$$

Observation 4.13. *If $A \leq \mathsf{S}_n$ and $B \leq \mathsf{S}_m$ are transitive groups on $[n]$ and $[m]$ then the group $A \times B$ is a transitive group acting on the cells on the matrix.*

There are many interesting transitive groups. The symmetric group is indeed transitive. The graph isomorphism group (that acts on the adjacency matrix - minus the diagonal - of a graph by changing the ordering on the vertices) is transitive. The cyclic permutation over all the points in the set is a transitive group. The following is another non-trivial transitive group on $[k]$ that we will use extensively in this paper.

Definition 4.14. *For any k that is a power of 2, the Binary-tree-transitive group Bt_k is a subgroup of S_k. To describe its generating set we think of group Bt_k acting on the elements $\{1, \ldots, k\}$ and the elements are placed in the leaves of a balanced binary tree of depth $\log k$ - one element in each leaf. Each internal node (including the root) corresponds to an element in the generating set of Bt_k. The element corresponding to an internal node in the binary tree swaps the left and right sub-tree of the node. The permutation element corresponding to the root node is called the Root-swap as it swaps the left and right sub-tree to the root of the binary tree.*

Claim 4.15. *The group Bt_k is a transitive group.*

Proof. For any $i, j \in [k]$, we have to show that there exists a permutation $\pi \in \mathsf{Bt}_k$ such that $\pi(i) = j$. Let us form a complete binary tree of height $\log k$ in the following way:

- (Base case:) Start from root node, label the left and right child as 0 and 1 respectively.
- For every node x, label the left and right child as $x0$ and $x1$ respectively.

Note that our complete binary tree has k leaves, where each of the leaf is labeled by a binary string of the form $x_1x_2\dots x_{\log k}$, which is the binary representation of numbers in $[k]$. Similarly any node in the tree can be labeled by a binary string $x_1x_2\dots x_t$, where $0 \le t \le \log k$ and t is the distance of the node from the root.

Now for any $i, j \in [k]$, let the binary representation of i be $(x_1x_2\dots x_{\log k})$ and that of j be $(y_1y_2\dots y_{\log k})$. Now we will construct the permutation $\pi \in \mathsf{Bt}_k$ such that $\pi(i) = j$. Without loss of generality, we can assume $i \neq j$.

Find the least positive integer $\ell \in [\log k]$ such that $x_\ell \neq y_\ell$, then go to the node labeled $x_1x_2\dots x_{\ell-1}$ and swap it's left and right child. Let $\pi_{x_1\dots x_{\ell-1}} \in S_k$ be the corresponding permutation of the leaves of the tree, in other words on the set $[k]$. Note that, by definition, the permutation $\pi_{x_1\dots x_{\ell-1}} \in S_k$ is in Bt_k. Also note that the permutation $\pi_{x_1\dots x_{\ell-1}}$ acts of the set $[k]$ as follows:

- $\pi_{x_1\dots x_{\ell-1}}(z_1\dots z_{\log k}) = z_1\dots z_{\log k}$ if $z_1\dots z_{\ell-1} \neq x_1\dots x_{\ell-1}$
- $\pi_{x_1\dots x_{\ell-1}}(x_1\dots x_{\ell-1}0z_{\ell+1}\dots z_{\log k}) = (x_1\dots x_{\ell-1}1z_{\ell+1}\dots z_{\log k})$
- $\pi_{x_1\dots x_{\ell-1}}(x_1\dots x_{\ell-1}1z_{\ell+1}\dots z_{\log k}) = (x_1\dots x_{\ell-1}0z_{\ell+1}\dots z_{\log k})$

Since $i = x_1\dots x_{\ell-1}x_\ell\dots x_{\log k}$ and $j = y_1\dots y_{\ell-1}y_\ell\dots y_{\log k}$ with $x_1\dots x_{\ell-1} = y_1\dots y_{\ell-1}$ and $x_\ell \neq y_\ell$, so

$$\pi_{x_1\dots x_{\ell-1}}(i) = y_1\dots y_{\ell-1}y_\ell x_{\ell+1}\dots y_{\log k}$$

So the binary representation of $\pi_{x_1\dots x_{\ell-1}}(i)$ and j matching in the first ℓ positions which is one more that the number of positions where the binary representation of i and j matched. By doing this trick repeatedly, that is by applying different permutations from Bt_k one after another we can map i to j. $\square$

The following claim describes how the group Bt_k acts on various encoding of integers. Recall the balance-binary representation (Definition 4.7).

Claim 4.16. *For all $\widehat{\gamma} \in \mathsf{Bt}_{2\log n}$ there is a $\gamma \in \mathsf{S}_n$ such that for all $i, j \in [n]$, $\widehat{\gamma}[bb(i)] = bb(j)$ iff $\gamma(i) = j$.*

Proof. Recall the group $\mathsf{Bt}_{2\log n}$: assuming that the elements of $[2\log n]$ are placed on the leaves of the binary tree of depth $\log(2\log n)$, the group $\mathsf{Bt}_{2\log n}$ is generated by the permutations of the form "pick a node in the binary tree of and swap the left and right sub-tree of the node". So it is enough to prove that for any elementary permutation $\widehat{\gamma}$ of the form "pick a node in the binary tree and swap the left and right sub-tree of the node" there is a $\gamma \in \mathsf{S}_n$ such that for all $i, j \in [n]$, $\widehat{\gamma}[bb(i)] = bb(j)$ iff $\gamma(i) = j$.

Any node in the binary tree of depth $\log(2\log n)$ can be labeled by a $0/1$-string of length t, where $0 \leq t \leq \log(2\log n)$ is the distance of the node from the root. We split the proof of the claim into two cases depending on the value of the t - the distance from the root.

If $t = \log(2\log n)$: This is the case when the node is at the last level - just above the leaf level. Let the node be u and let s be the number of whose binary representation is the label of the node u. Let the numbers in the leaves of the tree corresponds to the $bb(i)$ - the balanced binary representation of $i \in [n]$. Note that because of the balanced binary representation the children of u are

- 0 (left-child) and 1 (right-child) if the s-th bit in the binary representation of i is 0
- 1 (left-child) and 0 (right-child) if the s-th bit in the binary representation of i is 1

So the permutation (corresponding to swapping the left and right sub-trees of u) only change the order of 0 and 1 - which corresponds to flipping the s-th bit of the binary representation of i. And so in this case the γ acting on the set $[n]$ is just collection transpositions swapping i and j iff the the binary representation of i and j are same except for the s-th bit.

So in this case for all $i, j \in [n]$, $\widehat{\gamma}[bb(i)] = bb(j)$ iff $\gamma(i) = j$.

If $t < \log(2\log n)$: Let the node be v. Note that in this case since the node keeps the order of the $2r-1$ and $2r$ bits unchanged (for any $1 \leq r \leq \log n$), so it is enough we can visualise the action by an action of swapping the left and right sub-trees of the node v on the binary representation of i (instead of the balance binary representation of i). And so we can see that the action of the permutation (corresponding to swapping the left and right sub-trees of v) automatically gives a permutation of the binary representations of numbers between 1 and n, as was discussed in the proof of Claim 4.15. And hence we have for all $i, j \in [n]$, $\widehat{\gamma}[bb(i)] = bb(j)$ iff $\gamma(i) = j$.

□

Now let us consider another encoding that we will using for the set of rows and columns of a matrix.

Definition 4.17. *Given a set R of n rows $r_1,\dots,r_n$ and a set C of n columns $c_1,\dots,c_n$ we define the balanced-pointer-encoding function $\mathscr{E} : (R\times\{0\})\cup(\{0\}\times C)\to\{0,1\}^{4\log n}$, as follows:*

$$\mathscr{E}(r_i,0) = bb(i)\cdot 0^{2\log n}, \text{ and, } \mathscr{E}(0,c_j) = 0^{2\log n}\cdot bb(j).$$

The following is a claim is easy to verify.

Claim 4.18. *Let R be a set of n rows $r_1,\dots,r_n$ and C be a set of n columns $c_1,\dots,c_n$ and consider the balanced-pointer-encoding function $\mathscr{E} : (R\times\{0\})\cup(\{0\}\times C)\to\{0,1\}^{4\log n}$. For any elementary permutation $\widehat{\sigma}$ in $\mathsf{Bt}_{4\log n}$ (other than the Root-swap) there is a $\sigma\in\mathsf{S}_n$ such that for any $(r_i,c_j)\in(R\times\{0\})\cup(\{0\}\times C)$*

$$\widehat{\sigma}[\mathscr{E}(r_i,c_j)] = \mathscr{E}(r_{\sigma(i)},c_{\sigma(j)}),$$

where we assume $r_0=c_0=0$ and any permutation of in S_n sends 0 to 0.

If $\widehat{\sigma}$ is the root-swap then for any $(r_i,c_j)\in(R\times\{0\})\cup(\{0\}\times C)$

$$\widehat{\sigma}[\mathscr{E}(r_i,c_j)] = \mathsf{Swap}_{\frac{1}{2}}(\mathscr{E}(r_i,c_j)) = \mathscr{E}(c_j,r_i).$$

4.3 Pointer functions and their properties

For the sake of completeness first we will describe the function introduced in [ABB$^+$17] that achieves separation between several complexity complexity measures like *Deterministic query complexity*, *Randomized query complexity*, *Quantum query complexity* etc. This function was originally motivated from [GPW18] function. There are three three variants of [ABB$^+$17] function that have some special kind of non-Boolean domain, which we call *Pointer matrix*. Our function is a special "encoding" of that non-Boolean domain such that the resulting function becomes transitive and achieves the separation between complexity measures that matches the known separation between the general functions. Here we will define only the first variant of [ABB$^+$17] function. Rest of the variants are defined in Section 4.3.1 of Chapter 2.

Definition 4.19 (Pointer matrix over Σ). *For $m,n\in\mathbb{N}$, let M be a $(m\times n)$ matrix with m rows and n columns. We refer to each of the $m\times n$ entries of M as* cells. *Each cell of the matrix is from a alphabet set Σ where $\Sigma=\{0,1\}\times\widetilde{P}\times\widetilde{P}\times\widetilde{P}$ and $\widetilde{P}=\{(i,j)|i\in[m],j\in[n]\}\cup\{\perp\}$. We call $\widetilde{P}$ as set of pointers where, pointers of the form $\{(i,j)|i\in[m],j\in[n]\}$ pointing to the cell (i,j) and $\perp$ is the null pointer. Hence,each entry $x_{(i,j)}$ of the matrix M is a*

4-*tuple from* Σ. *The elements of the* 4-*tuple we refer as* value, left pointer, right pointer *and* back pointer *respectively and denote by Value*$(x_{(i,j)})$, *LPointer*$(x_{(i,j)})$, *RPointer*$(x_{(i,j)})$ *and BPointer*$(x_{(i,j)})$ *respectively where Value* $\in \{0,1\}$, *LPointer*, *RPointer*, *BPointer* $\in \widetilde{P}$. *We call this type of matrix as* pointer matrix *and denote by* $\Sigma^{n\times n}$.

A special case of the pointer-matrix, which we call Type_1 pointer matrix over Σ, *is when for each cell of M, BPointer* $\in \{[n] \cup \perp\}$ *that is backpointers are pointing to the columns of the matrix.*

Also, in general when, BPointer $\in \{(i,j) | i \in [m], j \in [n]\} \cup \{\perp\}$, *we call it a* Type_2 pointer matrix over Σ.

Now we will define some additional properties of the domain that we need to define [ABB$^+$17] function.

Definition 4.20 (Pointer matrix with marked column). *Let M be an* $m \times n$ pointer-matrix *over* Σ. *A column* $j \in [n]$ *of M is defined to be a* marked column *if there exists exactly one cell* (i,j), $i \in [m]$, *in that column with entry* $x_{(i,j)}$ *such that* $x_{(i,j)} \neq (1,\perp,\perp,\perp)$ *and every other* cl *in that column is of the form* $(1,\perp,\perp,\perp)$. *The* cl (i,j) *is defined to be the* special element *of the* marked column *j*.

Let n be a power of 2. Let T be a rooted, directed and balanced binary tree with n-leaves and $(n-1)$ internal vertices. We will use the following notations that will be used in defining some functions formally.

Notation 4.21. *Let n be a power of* 2. *Let T be a rooted, directed and balanced binary tree with n-leaves and* $(n-1)$ *internal vertices. Labels the edges of T as follows: the outgoing edges from each node are labeled by either* $left$ *or right. The leaves of the tree are labeled by the elements of* $[n]$ *from left to right, with each label used exactly once. For each leaf* $j \in [n]$ *of the tree, the path from the root to the leaf j defines a sequence of* $left$ *and right of length* $O(\log n)$, *which we denote by* $T(j)$.

When n is not a power of 2, *choose the largest* $k \in \mathbb{N}$ *such that* $2^k \leq n$, *consider a complete balanced tree with* 2^k *leaves and add a pair of child node to to each* $n - 2^k$ *leaves starting from left. Define* $T(j)$ *as before.*

More details about *partial assignment* and *certificates* can be found in Definition 2.7. Now we are ready to describe the *Variant 1* of [ABB$^+$17] function.

Definition 4.22 (Variant 1 [ABB$^+$17]). *Let* $\Sigma^{m\times n}$ *be a* Type_1 pointer matrix *where BPointer is a pointer of the form* $\{j | j \in [n]\}$ *that points to other column and LPointer, RPointer are* *as usual points to other cell. Define* $\mathsf{A1}_{(m,n)} : \Sigma^{m\times n} \to \{0,1\}$ *on a* Type_1 pointer matrix *such that for all* $x = (x_{i,j}) \in \Sigma^{m\times n}$*, the function* $\mathsf{A1}_{(m,n)}(x_{i,j})$ *evaluates to* 1 *if and only if it has a* 1- cell certificate *of the following form:*

1. *there exists exactly one* marked column $j^\star$ *in* M*,*
2. *There is a special cell, say* $(i^\star, j^\star)$ *which we call the special element in the the* marked column $j^\star$ *and there is a balanced binary tree* T *rooted at the special cell,*
3. *for each non-marked column* $j \in [n] \setminus \{j^\star\}$ *there exist a* cell l_j *such that Value*$(l_j) = 0$ *and BPointer*$(l_j) = j^\star$ *where* l_j *is the end of the path that starts at the* special element *and follows the pointers LPointer and RPointer as specified by the sequence* $T(j)$*.* l_j *exists for all* $j \in [n] \setminus \{j^\star\}$ *i.e. no pointer on the path is* $\perp$*. We refer* l_j *as the leaves of the tree.*

The above function achieves the separation between D vs. $\mathsf{R_0}$ and D vs. Q for $m = 2n$. Here we will restate some of the results from [ABB+17] which we will use to prove the results for our function:

Theorem 4.23 ([ABB+17])**.** *The function* $\mathsf{A1}_{(m.n)}$ *in Definition 2.40 satisfies*

$$\begin{aligned} \mathsf{D} &= \Omega(n^2) \textit{ for } m = 2n \textit{ where } m,n \in \mathbb{N}, \\ \mathsf{R_0} &= \widetilde{O}(m+n) \textit{ for any } m,n \in \mathbb{N}, \\ \mathsf{Q} &= \widetilde{O}(\sqrt{m}+\sqrt{n}) \textit{ for any } m,n \in \mathbb{N}. \end{aligned}$$

Though [ABB+17] gives the deterministic lower bound for the function $\mathsf{A1}$ precisely for $2m \times m$ matrices following the same line of argument it can be proved that $\mathsf{D}(\Omega(n^2))$ holds for $n \times n$ matrices also. For sake of completeness we give a proof for $n \times n$ matrices.

Theorem 4.24. $\mathsf{D}(\mathsf{A1}_{(n,n)}) = \Omega(n^2)$.

Adversary Strategy for $\mathsf{A1}_{(n,n)}$: We describe an adversary strategy that ensures that the value of the function is undetermined after $\Omega(n^2)$ queries. Assume that deterministic query algorithm queries a cell (i,j). Let k be the number of queried cell in the column j. If $k \le \frac{n}{2}$ adversary will return $(1,\perp,\perp,\perp)$. Otherwise adversary will return $(0,\perp,\perp,n-k)$.

Claim 4.25. *The value of the function* $\mathsf{A1}_{(n,n)}$ *will be undetermined if there is a column with at most* $n/2$ *queried cells in the first* $\frac{n}{2}$ *columns* $\{1,2,\ldots,\frac{n}{2}\}$ *and at least* $3n$ *unqueried cells in total.*

Proof. Adversary can always set the value of function to 0 if the conditions of the claim are satisfied.

Adversary can also set the value of the function to 1: If $s \in [\frac{n}{2}]$ be the column with at most $\frac{n}{2}$ queried cell, then all the queried cells of the column are of the form $(1,\perp,\perp,\perp)$. Assign $(1,\perp,\perp,\perp)$ to the other cell and leave one cell for the *special element* $a_{p,s}$(say).

For each non-marked column $j \in [n]$ s define l_j as follows: If column j has one unqueried cell then assign $(0,\perp,\perp,s)$ to that cell. If all the cells of the column j were already queried then the column contains a cell with $(0,\perp,\perp,s)$ by the adversary strategy. So, in either case we are able to form a *leave* l_j in each of the non-marked column.

Now using the cell of *special element* $a_{p,s}$ construct a rooted tree of pointers isomorphic to tree T as defined in Definition 2.40 such that the internal nodes we will use the other unqueried cells and assign pointers such that $l(j)$'s are the leaves of the tree and the *special element* $a_{p,s}$ is the *root* of the tree. Finally assign anything to the other cell. Now the function will evaluates to 1.

To carry out this construction we need at most $3n$ number of unqueried cells. Outside of the *marked column* total $n-2$ cells for the internal nodes of the tree, atmost $n-1$ unqueried cell for the *leaves* and the *all* -1 *unique marked column* contains total n cell, so total $3n$ unqueried cell will be sufficient for our purpose.

Now there are total n number of columns and to ensure that each of the column in $\{1,2,\ldots,\frac{n}{2}\}$ contains at least $\frac{n}{2}$ queried cell we need at least $\frac{n^2}{4}$ number of queries. Since $n^2-3n \geq \frac{n^2}{4}$ for all $n \geq 6$. Hence $D(\mathsf{A1}_{(n,n)}) = \Omega(n^2)$. □

Hence Theorem 4.24 follows.

Also [GPW18]'s function realises quadratic separation between D and deg and the proof goes via UC_{min} upper bound. But $\mathsf{A1}_{(n,n)}$ exhibits the same properties corresponding to UC_{min}. So, from the following observation it follows that $\mathsf{A1}_{(n,n)}$ also achieves quadratic separation between D and deg.

Observation 4.26. *It is easy to observe that for each positive input x to the function* $\mathsf{A1}_{(n,n)}$, *the* marked column *together with the rooted tree of pointers with* leaves *in every other column gives a unique minimal* 1*-certificate of x. Thus,* $\mathsf{UC}_1(\mathsf{A1}_{(n,n)}) = \widetilde{O}(n)$. *Now, from the definition of* UC_{min} *it follows that* $\mathsf{UC}_{min}(\mathsf{A1}_{(n,n)}) \leq \mathsf{UC}_1(\mathsf{A1}_{(n,n)})$. *Hence,* $\mathsf{UC}_{min}(\mathsf{A1}_{(n,n)}) = \widetilde{O}(n)$. $\mathsf{UC}_{min}(\mathsf{A1}_{(n,n)}) = \widetilde{O}(n)$. *From the fact* $\mathsf{UC}_{min} \geq \deg$ *it follows that* $\deg(\mathsf{A1}_{(n,n)}) = O(n)$.

Observation 4.27 ([ABB$^+$17]). *For any input $\Sigma^{n\times n}$ to the function $\mathsf{A1}_{(n,n)}$ (in Definition 2.40) if we permute the rows of the matrix using a permutation σ_r and permute the columns of the matrix using a permutation σ_c and we update the pointers in each of the cells of the matrix accordingly then the function value does not change.*

4.3.1 Variants of Pointer function

Now we describe *Variant 2* of [ABB$^+$17] function where the domain is slightly different from that of *Variant 1*.

Definition 4.28 (Variant 2 [ABB$^+$17]). *Let $\Sigma^{m\times n}$ be a* Type_2 pointer matrix *such that BPointer $\in \{(i,j)|i \in [m], j \in [n]\}$. Here the BPointer points to a cell of M, not a column. Let n be even. Define $\mathsf{A2}_{(m,n)} : \Sigma^{m\times n} \to \{0,1\}$ on* Type_2 pointer matrix *such that for all $x = (x_{i,j}) \in \Sigma^{m\times n}$, the function $\mathsf{A2}_{(m,n)}(x_{i,j})$ evaluates to 1 if and only if it has a* 1-cell certificate *of the following form:*

1. *there exists exactly one* Marked column *$j^\star$ in M. Let $(i^\star, j^\star)$ be the* special element.
2. *for each non-marked column $j \in [n] \setminus \{b\}$ there exist an element l_j such that $Value(l_j) = 0$ where l_j be the end of the path that starts at the* special element *and follows the pointers LPointer and RPointer as specified by the sequence $T(j)$. l_j exists in all $j \in [n] \setminus \{b\}$. We call that l_j the Leaves of the tree.*
3. *The size of the set $\{j \in [n] \setminus \{b\}|BPointer(l_j) = (i^\star, j^\star)\}$ is exactly $\frac{n}{2}$.*

Variant 2 achieves the separation between $\mathsf{R_0}$ vs. R and $\mathsf{R_0}$ vs $\mathsf{Q_E}$. The following results are some of the properties of *Variant 2* function:

Theorem 4.29. *[ABB$^+$17] For any $m,n \in \mathbb{N}$, the function $\mathsf{A2}_{(m,n)}$ in Definition 4.28 satisfies*

$$\mathsf{R_0} = \Omega(mn),$$
$$\mathsf{R} = \widetilde{O}(m+n) \text{ and}$$
$$\mathsf{Q_E} = \widetilde{O}(m+n).$$

Finally we will describe *Variant 3* of [ABB$^+$17] function, where there is one extra parameter, the number of *marked column*. So, we will address the function as $\mathsf{A3}_{(k,m,n)}$ where m,n are number of rows and columns respectively and k is the number of *marked column*.

Definition 4.30 (Variant 3 [ABB$^+$17]). *Define* $\mathsf{A3}_{(k,m,n)} : \Sigma^{m\times n} \to \{0,1\}$ *on* Type_2 pointer-matrix Σ *such that for all* $x = (x_{i,j}) \in \Sigma^{m\times n}$, *the function* $\mathsf{A3}_{(k,m,n)}(x_{i,j})$ *evaluates to* 1 *if and only if it has a* 1-cell certificate *of the following form:*

1. *there exists k* Marked column *in M. Let* $b_1, b_2, \ldots, b_k \in [n]$ *denotes the* Marked column *in M and* (a_k, b_k) *be the* special element *in each* Marked column. *Let us denote the cell of* special element *by a*
2. $BPointer(x_{a_j,b_j}) = (a_{j+1}, b_{j+1})$ *for all* $j \in [k-1]$ *and* $BPointer(x_{a_k,b_k}) = (a_1, b_1)$. *Also,* $LPointer(x_{a_s,b_s}) = LPointer(x_{a_t,b_t})$ *and* $RPointer(x_{a_s,b_s}) = RPointer(x_{a_t,b_t})$ *for all* $s, t \in [k]$.
3. *for each non-marked column* $j \in [n] \setminus \{b_1, b_2, \ldots, b_k\}$ *there exist an element* ℓ_j *such that* $Value(l_j) = 0$ *and* $BPointer(l_j) = b$ *where* l_j *be the end of the path that starts at the* special element *and follows the pointers LPointer and RPointer as specified by the sequence* $T(j)$. *We need that* l_j *exists in each* $j \in [n] \setminus \{b_1, b_2, \ldots, b_k\}$ *i.e. no pointer on the path is* $\perp$. *We call that* l_j *the Leaves of the tree.*

Variant 3 realises the separation between R_0 vs. Q, R vs. Q_E, R vs. deg and $\widetilde{\deg}$ for different m, n and k. The followings are some important result for *variant 3* from [ABB$^+$17]:

Theorem 4.31 ([ABB$^+$17]). *For sufficiently large* $m, n\mathbb{N}$ *and a natural number* $k < n$, *the function* $\mathsf{A3}_{(k,m,n)}$ *in Definition 4.30 satisfies*

$$\begin{aligned}
\mathsf{R}_0 &= \Omega(mn) \textit{ for } k < \frac{n}{2}, \\
\mathsf{R}_0 &= \Omega(\frac{mn}{\log n}) \textit{ for } k = 1, \\
\mathsf{Q} &= \widetilde{O}(\sqrt{mn/k} + \sqrt{km} + k + \sqrt{n}), \\
\mathsf{Q}_\mathsf{E} &= \widetilde{O}(m\sqrt{n/k} + km + n) \textit{ and} \\
\widetilde{\deg} &= \widetilde{O}(\sqrt{m} + \sqrt{n}) \textit{ for } k = 1.
\end{aligned}$$

Observation 4.32. *It is easy to observe that for each positive input x to the function* $\mathsf{A3}_{(1,n,n)}$, *the* marked column *together with the rooted tree of pointers with* leaves *in every other column gives a unique minimal* 1*-certificate of x. Thus,* $\mathsf{UC}_1(\mathsf{A3}_{(1,n,n)}) = \widetilde{O}(n)$. *Now, from the definition of* UC_{min} *it follows that* $\mathsf{UC}_{min}(\mathsf{A3}_{(n,n)}) = \widetilde{O}(n)$. *Also it follows from the fact* $\mathsf{UC}_{min} \geq$ deg *that* $\deg(\mathsf{A3}_{(n,n)}) = O(n)$.

4.4 Separation results via Transitive pointer functions

4.4.1 High level description of our techniques

Pointer functions are defined over a special domain called *pointer matrix*, which is a $m \times n$ grid matrix. Each cell of the matrix contains some labels and some pointers that point either to some other cell or to a row or column [5]. For more details, refer to Section 4.3 of Chapter 2. As described in [GPW18], the high level idea of pointer functions is the usage of pointers to make certificates unambiguous without increasing the input size significantly. This technique turns out to be very useful to give separations between various complexity measures as we see in [MS15], [GJPW18b] and [ABB$^+$17].

Now we want to produce a new function that possesses all the properties of pointer functions, along with the additional property of being transitive. To do so, first, we will encode the labels so that we can permute the bits (by a suitable transitive group) while keeping the structure of unambiguous certificates intact so that the function value remains invariant. One such natural technique would be to encode the contents of each cell in such a way that allows us to permute the bits of the contents of each cell using a transitive group and permute the cells among each other using another transitive group, and doing all of these while ensuring the unambiguous certificates remains intact [6]. This approach has a significant challenge: namely how to encode the pointers.

The information stored in each cell (other than the pointers) can be encoded using fixed logarithmic length strings of different Hamming weights - so that even if the strings are permuted and/or the bits in each string are permuted, the content can be “decoded". Unfortunately, this can only be done when the cell's contents have a constant amount of information - which is the case for pointer functions (except for the pointers). Since the pointers in the cell are strings of size $O(\log n)$ (as they are pointers to other columns or rows), if we want to use the similar Hamming weight trick, the size of the encoding string would need to be polynomial in $O(n)$. That would increase the size of the input compared to the unambiguous certificate. This would not give us tight separation results.

Also, there are three more issues concerning the encodings of pointers:

[5]We naturally think of a pointer pointing to a cell as two pointers - one pointing to the row and the other to the column.

[6]Here, we use the word “encode" since we can view the function defined only over codewords, and when the input is not a codeword, then it evaluates to 0. In our setting, since we are trying to preserve the one-certificates, the codewords are those strings where the unambiguous certificate is encoded correctly. At the same time, we must point out that the encoding of an unambiguous certificate is not necessarily unique.

- As we permute the cells of the matrix according to some transitive group, the pointers within each cell need to be appropriately changed. In other words, when we move some cell's content to some other cell, the pointers pointing to the previous cell should point to the current cell now.
- If a pointer is encoded using a certain t-bit string, different permutations of bits of the encoded pointer can only generate a subset of all t-bit strings.

  For example: if we encode a pointer using a string of Hamming weight 10 then however we permute the bits of the string, the pointer can at most be modified to point to cells (or rows or columns) the encoding of whose pointers also have Hamming weight 10. (The main issue is that permuting the bits of a string cannot change the Hamming weight of a string).

  The encoding of all the pointers should have the same Hamming weight.
- The encoding of the pointers has to be transitive. That is, we should be able to permute the bits of the encodings of the pointer using a transitive group in such a way that either the pointer value does not change or as soon as the pointer values changes, the cells gets permuted accordingly - kind of like an "entanglement".

The above three problems are somewhat connected. Our first innovative idea is to use *binary balance representation* (Definition 4.7) to represent the pointers. This way, we take care of the second issue. For the first and third issues, we define the transitive group - both the group acting on the contents of the cells (and hence on the encoding of the pointers) and the group acting on the cells itself - in a "entangled" manner. For this we induce a group action acting on the nodes of a *balanced binary tree* and generate a transitive subgroup in S_n and $S_{2\log n}$ with the same action which will serve our purpose (Definition 4.14, Claim 4.16). This helps us to permute the rows (or columns) using a permutation while updating the encoding of the pointers accordingly.

By Claim 4.16, for every allowed permutation σ acting on the rows (or columns), there is a unique $\widehat{\sigma}$ acting on the encodings of the pointers in each of the cells such that the pointers are updated according to σ. This still has a delicate problem. Namely, each pointer is either pointing to a row or column. But the permutation $\widehat{\sigma}$ has no way to understand whether the encoding on which it is being applied points to a row or column. To tackle this problem, we think of the set of rows and columns as a single set. All of them are encoded by a string of size (say) $2t$, where for the rows, the second half of the encoding is all 0 while the columns have the first t bits all 0. This is the encoding described in Definition 4.17 using binary balanced representation. However, this adds another delicate issue about permuting between the first t bits of the encoding and the second t bits.

To tackle this problem, we modify the original function appropriately. We define a slightly modified version of existing pointer functions called $\mathsf{ModA1}$. This finally helps us obtain our “transitive pointer function," which has almost the same complexities as the original pointer function.

We have so far only described the high-level technique to make the 1st variation of pointer functions (Definition 2.40) transitive where there is the same number of rows and columns. The further variations need more delicate handling of the encoding and the transitive groups - though the central idea is similar.

4.5 Separations between deterministic query complexity and other complexity measures

4.5.1 Transitive Pointer Function $F_{1.4}$ for Theorem 1.4

Our function $F_{1.4} : \Gamma^{n\times n} \to \{0,1\}$ is a composition of two functions - an outer function $\mathsf{ModA1}_{(n,n)} : \bar{\Sigma}^{n\times n} \to \{0,1\}$ and an inner function $\mathscr{D} : \Gamma \to \bar{\Sigma}$. We will set Γ to be $\{0,1\}^{96\log n}$.

The outer function is a modified version of the $\mathsf{A1}_{(n,n)}$ - pointer function described in [ABB+17] (see Definition 2.40 for a description). The function $\mathsf{A1}_{(n,n)}$ takes as input a $(n\times n)$-matrix whose entries are from a set Σ and the function evaluates to 1 if a certain kind of 1-cell-certificate exists. Let us define a slightly modified function $\mathsf{ModA1}_{(n,n)} : \bar{\Sigma}^{n\times n} \to \{0,1\}$ where $\bar{\Sigma} = \Sigma \times \{\vdash, \dashv\}$. We can think of an input $A \in \bar{\Sigma}^{n\times n}$ as a pair of matrices $B \in \Sigma^{n\times n}$ and $C \in \{\vdash, \dashv\}^{n\times n}$. The function $\mathsf{ModA1}_{(n,n)}$ is defined as

$$\mathsf{ModA1}_{(n,n)}(A) = 1 \text{ iff } \begin{cases} \text{Either, (i)} & \mathsf{A1}_{(n,n)}(B) = 1, \text{ and, all the cells in the} \\ & \text{1-cell-certificate have } \vdash \text{ in the corresponding cells in } C \\ \text{Or, (ii)} & \mathsf{A1}_{(n,n)}(B^T) = 1, \text{ and, all the cells in the} \\ & \text{1-cell-certificate have } \dashv \text{ in the corresponding cells in } C^T \end{cases}$$

Note that both the two conditions (i) and (ii) cannot be satisfied simultaneously. From this it is easy to verify that the function $\mathsf{ModA1}_{(n,n)}$ has all the properties as $\mathsf{A1}_{(n,n)}$ as described in Theorem 4.23.

The inner function $\mathscr{D}$ (we call it a decoding function) is function from Γ to $\bar{\Sigma}$, where $\Gamma = 96\log n$. Thus our final function is

$$F_{1.4} := \left(\mathsf{ModA1}_{(n,n)} \circ \mathscr{D}\right) : \Gamma^{n\times n} \to \{0,1\}.$$

Inner Function $\mathscr{D}$

The input to $\mathsf{A1}_{(n,n)}$ is a Type_1 pointer matrix $\Sigma^{n\times n}$. Each cell of a Type_1 pointer matrix contains a 4-tuple of the form (Value, LPointer, RPointer, BPointer) where Value is either 0 or 1 and LPointer, RPointer are pointers to the other cells of the matrix and BPointer is a pointer to a column of the matrix (or can be a null pointer also). Hence, $\Sigma = \{0,1\} \times [n]^2 \times [n]^2 \times [n]$. For the function $\mathsf{A1}_{(n,n)}$ it was assumed (in [ABB$^+$17]) that the elements of Σ is encoded as a k-length[7] binary string in a canonical way.

The main insight for our function $F_{1.4} := \left(\mathsf{ModA1}_{(n,n)} \circ \mathscr{D}\right)$ is that we want to maintain the basic structure of the function $\mathsf{A1}_{(n,n)}$ (or rather of $\mathsf{ModA1}_{(n,n)}$) but at the same time we want to encode the $\bar{\Sigma} = \Sigma \times \{\vdash, \dashv\}$ in such a way that the resulting function becomes transitive. To achieve this, instead of having a unique way of encoding an element in $\bar{\Sigma}$ we produce a number of possible encodings[8] for any element in $\bar{\Sigma}$. The inner function $\mathscr{D}$ is therefore a decoding algorithm that given any proper encoding of an element in $\bar{\Sigma}$ will be able to decode it back.

For the ease of understanding we start by describing the possible "encodings" of $\bar{\Sigma}$, that is by describing the pre-images of any element of $\bar{\Sigma}$ in the function $\mathscr{D}$.

"Encodings" of the content of a cell in $\bar{\Sigma}^{n\times n}$:

We will encode any element of $\bar{\Sigma}$ using a string of size $96\log n$ bits. Recall that, an element in $\bar{\Sigma}$ is of the form $(V,(r_L,c_L),(r_R,c_R),(c_B),T)$, where V is the Boolean value, (r_L,c_L), (r_R,c_R) and c_B are the left pointer, right pointers and bottom pointer respectively and T take the value $\vdash$ or $\dashv$. The overall summary of the encoding is as follows:

- **Parts:** We will think of the tuple as 7 objects, namely V, r_L, c_L, r_R, c_R, c_B and T. We will use $16\log n$ bits to encode each of the first 6 objects. The value of T will be encoded in a clever way. So the encoding of any element of $\bar{\Sigma}$ contains 6 *parts* - each a binary string of length $16\log n$.

[7]For the canonical encoding $k = (1+5\log n)$ was sufficient

[8]We use the term "encoding" a bit loosely in this context as technically an encoding means a unique encoding. What we actually mean is the pre-images of the function $\mathscr{D}$.

- **Blocks:** Each of 6 *parts* will be further broken into 4 *blocks* of equal length of $4\log n$. One of the blocks will be a special block called the "encoding block".

Now we explain, the contents of a tuple $(V,(r_L,c_L),(r_R,c_R),(c_B),T)$ precisely we will describe what are the 4 blocks in each part. We will start by describing a "standard-form" encoding of a tuple $(V,(r_L,c_L),(r_R,c_R),(c_B),T)$ where $T=\vdash$. Then we will extend it to describe the standard for encoding of $(V,(r_L,c_L),(r_R,c_R),(c_B),T)$ where $T=\dashv$. And finally we will explain all other valid encoding of a tuple $(V,(r_L,c_L),(r_R,c_R),(c_B),T)$ by describing all the allowed permutations on the bits of the encoding.

Standard-form encoding of $(V,(r_L,c_L),(r_R,c_R),(c_B),T)$ **where** $T=\vdash$**:** For the standard-form encoding we will assume that the information of V,r_L,c_L,r_R,c_R,c_B are stored in parts $P1,P2,P3,P4,P5$ and $P6$ respectively. For all $i\in[6]$, the part P_i with have blocks B_1,B_2,B_3 and B_4, of which the block B_1 will be the encoding-block. The encoding will ensure that every parts within a cell will have distinct Hamming weight. The description is also compiled in the Table 4.1.

- For part $P1$ (that is the encoding of V) the encoding block B_1 will store $\ell_1\cdot\ell_2$ where ℓ_1 be the $2\log n$ bit binary string with Hamming weight $2\log n$ and ℓ_2 is any $2\log n$ bit binary string with Hamming weight $2\log n-1-V$. The blocks B_2, B_3 and B_4 will store a $4\log n$ bit string that has Hamming weight $4\log n, 2\log n+1$ and $2\log n+2$ respectively. Any fixed string with the correct Hamming weight will do. We are not fixing any particular string for the blocks B_2, B_3 and B_4 to emphasise the fact that we will be only interested in the Hamming weights of these strings.

- The encoding block $B1$ for parts $P2,P3,P4,P5$ and $P6$ will store the string $\mathscr{E}(r_L,0)$, $\mathscr{E}(0,c_L)$, $\mathscr{E}(r_R,0)$, $\mathscr{E}(0,c_r)$ and $\mathscr{E}(0,C_B)$ respectively, where $\mathscr{E}$ is the Balanced-pointer-encoding function (Definition 4.17). For part P_i (with $2\le i\le 6$) block B_2,B_3 and B_4 will store any $4\log n$ bit string with Hamming weight $2\log n+1+i$, $2\log n+1$ and $2\log n+2$ respectively.

Standard form encoding of $(V,(r_L,c_L),(r_R,c_R),(c_B),T)$ **where** $T=\dashv$**:**
For obtaining a standard-form encoding of $(V,(r_L,c_L),(r_R,c_R),(c_B),T)$ where $T=\dashv$, first we encode $(V,(r_L,c_L),(r_R,c_R),(c_B),T)$ where $T=\vdash$ using the standard-form encoding. Let $(P1,P2,\dots,P6)$ be the standard-form encoding of $(V,(r_L,c_L),(r_R,c_R),(c_B),T)$ where $T=\vdash$. Now for each of the block apply the $\mathsf{Swap}_{\frac{1}{2}}$ operator.

Table 4.1 Standard form of encoding of a cell of a pointer matrix: Variant 1 (Type I element)

$\ldots$	B_1 "encoding"-block	B_2	B_3	B_4	Hamming weight
$P1$	$\ell_1\ell_2$, where $\lvert\ell_1\rvert = 2\log n$, and $\lvert\ell_2\rvert = 2\log n - 1 - V$	$4\log n$	$2\log n+1$	$2\log n+2$	$12\log n+2-V$
$P2$	$\mathscr{E}(r_L,0)$	$2\log n+3$	$2\log n+1$	$2\log n+2$	$7\log n+6$
$P3$	$\mathscr{E}(0,c_L)$	$2\log n+4$	$2\log n+1$	$2\log n+2$	$7\log n+7$
$P4$	$\mathscr{E}(r_R,0)$	$2\log n+5$	$2\log n+1$	$2\log n+2$	$7\log n+8$
$P5$	$\mathscr{E}(0,c_R)$	$2\log n+6$	$2\log n+1$	$2\log n+2$	$7\log n+9$
$P6$	$\mathscr{E}(0,c_B)$	$2\log n+7$	$2\log n+1$	$2\log n+2$	$7\log n+10$

Standard form of encoding of element $(V,(r_L,c_L),(r_R,c_R),c_B,\vdash)$ by a $96\log n$ bit string that is broken into 6 parts $P_1,\ldots,P_6$ of equal size and each Part is further broken into 4 Blocks B_1,B_2,B_3 and B_4. So all total there are 24 blocks each containing a $4\log n$-bit string. For the standard form of encoding of element $(V,(r_L,c_L),(r_R,c_R),c_B,\dashv)$ we encode $(V,(r_L,c_L),(r_R,c_R),c_B,\vdash)$ in the standard form as described in the table and then apply the $\mathsf{Swap}_{\frac{1}{2}}$ on each block. The last column of the table indicates the Hamming weight of each Part.

Valid permutation of the standard form: Now we will give a set of valid permutations to the bits of the encoding of any element of $\bar{\Sigma}$. The set of valid permutations are classified into into 3 categories:

1. Part-permutation: The 6 parts can be permuted using any permutation from S_6
2. Block-permutation: In each of the part, the 4 blocks (say B_1,B_2,B_3,B_4) can be permuted is two ways. (B_1,B_2,B_3,B_4) can be send to one of the following

   (a) Simple Block Swap: (B_3,B_4,B_1,B_2) (b) Block Flip: $(B_2,B_1,\mathsf{flip}(B_3),\mathsf{flip}(B_4))$

The "decoding" function $\mathscr{D}: \{0,1\}^{96\log n} \to \bar{\Sigma}$:

- Identify the parts containing the encoding of V, r_L, c_L, r_R, c_R and c_B. This is possible because every part has a unique Hamming weight.
- For each part identify the blocks. This is also possible as in any part all the blocks have distinct Hamming weight. Recall, the valid Block-permutations, namely Simple Block Swap and Block Flip. By seeing the positions of the blocks one can understand if flip was applied and to what and using that one can revert the blocks back to the standard-form (recall Definition 4.7).
- In the part containing the encoding of V consider the encoding-block. If the block is of the form $\{(\ell_1\ell_2)\,\text{such that}\,\lvert\ell_1\rvert = 2\log n, \lvert\ell_2\rvert \le 2\log n - 1\}$ then $T = \{\vdash\}$. If the block is of the form $\{(\ell_2\ell_1)\,\text{such that}\,\lvert\ell_1\rvert = 2\log n, \lvert\ell_2\rvert \le 2\log n - 1\}$ then $T = \{\dashv\}$.

- By seeing the encoding block we can decipher the original values and the pointers.
- If the $96 \log n$ bit string doesn't have the form of a valid encoding, then decode it as $(0, \perp, \perp, \perp)$.

4.5.2 Proof of Transitivity of the function

We start with describing the transitive group for which $F_{1.4}$ is transitive.

The Transitive Group: We start with describing a transitive group $\mathscr{T}$ acting on the cells of the matrix A. The matrix has rows $r_1, \dots, r_n$ and columns $c_1, \dots, c_n$. And we use the encoding function $\mathscr{E}$ to encode the rows and columns. So the index of the rows and columns are encoded using a $4 \log n$ bit string. A permutation from $\mathsf{Bt}_{4 \log n}$ (see Definition 4.14) on the indices of a $4 \log n$ bit string will therefore induce a permutation on the set of rows and columns which will give us a permutation on the cells of the matrix. We will now describe the group $\mathscr{T}$ acting on the cells of the matrix by describing the permutation group $\widehat{\mathscr{T}}$ acting on the indices of a $4 \log n$ bit string. The group $\widehat{\mathscr{T}}$ will be the group $\mathsf{Bt}_{4 \log n}$ acting on the set $[4 \log n]$. We will assume that $\log n$ is a power of 2. The group $\mathscr{T}$ with be the resulting group of permutations on the cells of the matrix induced by the group $\widehat{\mathscr{T}}$ acting on the indices on the balanced-pointer-encoding. Note that $\mathscr{T}$ is acting on the domain of $\mathscr{E}$ and $\widehat{\mathscr{T}}$ is acting on the image of $\mathscr{E}$. Also $\widehat{\mathscr{T}}$ is a transitive subgroup of $\mathsf{S}_{4 \log n}$ from Claim 4.15.

Observation 4.33. *For any $1 \leq i \leq 2 \log n$ consider the permutation "ith-bit-flip" in $\widehat{\mathscr{T}}$ that applies the transposition $(2i-1, 2i)$ to the indices of the balanced-pointer-encoding. Since the $\mathscr{E}$-encoding of the row $(r_k, 0)$ uses the balanced binary representation of k in the first half and all zero sting in the second half, the jth bit in the binary representation of k is stored in the $2j-1$ and $2j$-th bit in the $\mathscr{E}$-encoding of r_i. So the j-th-bit-flip acts on the sets of rows by swapping all the rows with 1 in the j-th bit of their index with the corresponding rows with 0 in the j-th bit of their index. Also, if $i > \log n$ then there is no effect of the i-th-bit-flip operation on the set of rows. Similarly for the columns.*

Using Observation 4.33 we have the following claim.

Claim 4.34. *The group $\mathscr{T}$ acting on the cells of of the matrix is a transitive group. That is, for all $1 \leq i_1, j_1, i_2, j_2 \leq n$ there is a permutation $\widehat{\sigma} \in \widehat{\mathscr{T}}$ such that $\widehat{\sigma}[\mathscr{E}(i_1, 0)] = \mathscr{E}(i_2, 0)$ and $\widehat{\sigma}[\mathscr{E}(0, j_1)] = (0, j_2)$. Or in other words, there is a $\sigma \in \mathscr{T}$ acting on the cell of the matrix that would take the cell corresponding to row r_{i_1} and column c_{j_1} to the cell corresponding to row r_{i_2} and column c_{j_2}.*

From the Claim 4.34 we see the group $\mathscr{T}$ acting on the cells of of the matrix is a transitive. But it does not touch the contents within the cells of the matrix. But the input to the function $F_{1.4}$ contains element of $\Gamma = \{0,1\}^{96\log n}$ in each cell. So we now need to extend the group $\mathscr{T}$ to a group G that acts on all the indices of all the bits of the input to the function $F_{1.4}$.

Recall that the input to the function $F_{1.4}$ is a $(n \times n)$-matrix with each cell of matrix containing a binary string of length $96\log n$ which has 6 parts of size $16\log n$ each and each part has 4 blocks of size $4\log n$ each. We classify the generating elements of the group G into 4 categories:

1. Part-permutation: In each of the cells the 6 parts can be permuted using any permutation from S_6.
2. Block-permutation: In each of the Parts the 4 blocks can be permuted in the following ways. (B_1, B_2, B_3, B_4) can be send to one of the following
   (a) Simple Block Swap: (B_3, B_4, B_1, B_2),
   (b) Block Flip (#1): $(B_2, B_1, \mathsf{flip}(B_3), \mathsf{flip}(B_4))$,
   (c) Block Flip (#2)[9]: $(\mathsf{flip}(B_1), \mathsf{flip}(B_2), B_4, B_3)$.
3. Cell-permutation: for any $\sigma \in \mathscr{T}$ the following two action has to be done simultaneously:
   (a) (Matrix-update) Permute the cells in the matrix according to the permutation σ. This keeps the contents within each cells untouched - it just changes the location of the cells.
   (b) (Pointer-update) For each of blocks in each of the parts in each of the cells permute the indices of the $4\log n$-bit strings according to σ, that is apply $\widehat{\sigma} \in \widehat{\mathscr{T}}$ corresponding to σ.

We now have the following theorems that would prove that the function $F_{1.4}$ is transitive.

Theorem 4.35. G *is a transitive group and the function $F_{1.4}$ is invariant under the action of the* G.

Proof of Theorem 4.35. To prove that the group G is transitive we show that for any indices $p, q \in [96n^2 \log n]$ there is a permutation $\sigma \in \mathsf{G}$ that would take p to q. Recall that the string $\{0,1\}^{96n^2 \log n}$ is a matrix $\Gamma^{(n\times n)}$ with $\Gamma = \{0,1\}^{96\log n}$ and every element in Γ is broken into 6 parts and each part being broken into 4 block of size $4\log n$ each. So we can think of

[9]Actually this Block flip can be generated by a combination of Simple Block Swap and Block Flip (#1)

the index p as sitting in k_pth position $(1 \leq k_p \leq 4\log n)$ in the block B_p of the part P_p in the (r_p, c_p)-th cell of the matrix. Similarly, we can think of q as sitting in k_qth position $(1 \leq k_q \leq 4\log n)$ in the block B_q of the part P_q in the (r_q, c_q)-th cell of the matrix.

We will give a step by step technique in which permutations from G can be applied to move p to q.

Step 1 **Get the positions in the block correct:** If $k_p \neq k_q$ then take a permutation $\widehat{\sigma}$ from $\widehat{\mathscr{T}}$ that takes k_p to k_q. Since $\widehat{\mathscr{T}}$ is a transitive so such a permutation exists. Apply the cell-permutation $\sigma \in \mathscr{T}$ corresponding to $\widehat{\sigma}$. As a result the index p can be moved to a different cell in the matrix but, by the choice of $\widehat{\sigma}$ its position in the block in which it is will be k_q. Without loss of generality, we assume the the cell location does not change.

Step 2 **Get the cell correct:** Using a cell-permutation that corresponds to a series of "bit-flip" operations change r_p to r_q and c_p to c_q. Since one bit-flip operations basically changes one bit in the binary representation of the index of the row or column such a series of operations can be made.

Since each bit-flip operation is executed by applying the bit-flips in each of the blocks so this might have once again changed the position of the index p in the block. But, even if the position in the block changes it must be a flip operation away. Or in other word, since in the beginning of this step $k_p = k_q$, so if k_q is even (or odd) then after the series bit-flip operations the position of p in the block is either k_q or $(k_q - 1)$ (or $(k_q + 1)$).

Step 3 **Align the Part:** Apply a suitable permutation to ensure that the part P_p moves to part P_q. Note this does not change the cell or the block within the part or the position in the block.

Step 4 **Align the Block:** Using a suitable combination of Simple Block Swap and Block Flip ensures the Block number gets matched, that is B_p goes to B_q. In this case the cell or the Part does not change. But depending on whether the Block Flip operation is applied the position in the block can again change. But, the current position in the block k_p is at most one flip away from k_q.

Step 5 **Apply the final flip:** It might so happen that already we a done after the last step. If not we know that the current position in the block k_p is at most one flip away from k_q. So we apply the suitable Block-flip operation. Thus will not change the cell position, Part number, Block number and the position in the block will match.

Hence we have proved that the group G is transitive. Now we show that the the function $F_{1.4}$ is invariant under the action of G, i.e., for any elementary operations π from the group G and for any input $\Gamma^{(n\times n)}$ the function value does not change even if after the input is acted upon by the permutation π.

Case 1: π is a Part-permutation: It is easy to see that the decoding algorithm $\mathscr{D}$ is invariant under Part-permutation. This was observed in description of the decoding algorithm $\mathscr{D}$ in Section 4.5.1. So clearly that the function $F_{1.4}$ is invariant under any Part-permutation.

Case 2: π is a Block-permutation: Here also it is easy to see that the decoding algorithm $\mathscr{D}$ is invariant under Block-permutation. This was observed in description of the decoding algorithm $\mathscr{D}$ in Section 4.5.1. Thus $F_{1.4}$ is also invariant under any Block-permutation.

Case 3: π is a Cell-permutation From Observation 4.27 it is enough to prove that when we permute the cells of the matrix we update the points in the cells accordingly.

Let $\pi \in \mathscr{T}$ be a permutation that permutes only the rows of the matrix. By Claim 4.18, we see that the contents of the cells will be updated accordingly. Similarly if π only permute the columns of the matrix we will be fine.

Finally, if π swaps the row set and the column set (that is if π makes a transpose of the matrix) then for all i row i is swapped with column i and it is easy to see that $\widehat{\pi}[\mathscr{E}(i,0)] = \mathscr{E}(0,i)$. In that case the encoding block of the value part in a cell also gets swapped. This will thus be encoding the T value as $\dashv$. And so the function value will not be affected as the $T = \dashv$ will ensure that one should apply the π that swaps the row set and the column set to the input before evaluating the function. $\square$

Properties of the Function

Claim 4.36. *Deterministic query complexity of $F_{1.4}$ is $\Omega(n^2)$.*

Proof. The function $\mathsf{ModA1}_{(n,n)}$ is a "harder" function than $\mathsf{A1}_{(n,n)}$. So $\mathsf{D}(\mathsf{ModA1}_{(n,n)})$ is at least that of $\mathsf{D}(\mathsf{A1}_{(n,n)})$. Now since, $F_{1.4}$ is $\left(\mathsf{ModA1}_{(n,n)} \circ \mathscr{D}\right)$ so clearly the $\mathsf{D}(F_{1.4})$ is at least $\mathsf{D}(\mathsf{A1}_{(n,n)})$. Theorem 4.24 proves that $\mathsf{D}(\mathsf{A1}_{(n,n)})$ is $\Omega(n^2)$. Hence $\mathsf{D}(F_{1.4}) = \Omega(n^2)$. $\square$

The following Claim 4.37 follows from the definition of the function $\mathsf{ModA1}_{(n,n)}$.

Claim 4.37. *The following are some properties of the function* $\mathsf{ModA1}_{(n,n)}$

1. $\mathsf{R}_0(\mathsf{ModA1}_{(n,n)}) \leq 2\mathsf{R}_0(\mathsf{A1}_{(n,n)}) + O(n\log n)$
2. $\mathsf{Q}(\mathsf{ModA1}_{(n,n)}) \leq 2\mathsf{Q}(\mathsf{A1}_{(n,n)}) + O(n\log n)$

3. $\mathsf{deg}(\mathsf{ModA1}_{(n,n)}) \leq 2\mathsf{deg}(\mathsf{A1}_{(n,n)}) + O(n\log n)$

Finally, from Theorem 2.20 we see that the $\mathsf{R_0}(F_{1.4})$, $\mathsf{Q}(F_{1.4})$ and $\mathsf{deg}(F_{1.4})$ are at most $O(\mathsf{R_0}(\mathsf{ModA1}_{(n,n)} \cdot \log n)$, $O(\mathsf{Q}(\mathsf{ModA1}_{(n,n)} \cdot \log n)$ and $O(\mathsf{deg}(\mathsf{ModA1}_{(n,n)} \cdot \log n)$, respectively. So combining this fact with Claim 4.36, Claim 4.37 and Theorem 4.23 (from [ABB+17]) we have Theorem 1.4.

4.6 Separation between zero-error randomized query complexity and some of other complexity measures

[ABK16] used the a variant of the pointer function to give separation between zero-error randomized query complexity($\mathsf{R_0}$) and other measures like one-sided randomized query complexity, exact quantum query complexity and degree. In this section we construct a transitive version of the functions that was used in [ABK16] to show such separations and we prove the Theorem 4.2 where we show that there exists transitive function achieving the following separations:

$$\mathsf{R_0}(F_{4.2}) = \widetilde{\Omega}(\mathsf{R}(F_{4.2})^2), \qquad \mathsf{R_0}(F_{4.2}) = \widetilde{\Omega}(\mathsf{Q_E}(F_{4.2})^2), \qquad \mathsf{R_0}(F_{4.2}) = \widetilde{\Omega}(\mathsf{deg}(F_{4.2})^2).$$

4.6.1 Transitive Pointer function $F_{4.2}$ for Theorem 4.2

Recall from Definition 2.37, the input to the function $\mathsf{A2}$ is a $\mathsf{Type_2}$ *pointer matrix* $\Sigma^{n\times n}$. Each cell of such matrix is of the form $(\text{Value}, \text{LPointer}, \text{RPointer}, \text{BPointer})$ where Value is either 0 or 1 and LPointer, RPointer and BPointer are pointers to the other cells of the matrix (or can be a null pointer also). Here the BPointer are the pointers to the cells, which is different from $\mathsf{A1}_{(n,n)}$. That's why we need an encoding which is slightly different from the encoding scheme of $\mathsf{A1}_{(n,n)}$. Our function $F_{4.2} : \Gamma^{n\times n} \to \{0,1\}$ is again a composition of two functions - an outer function $\mathsf{ModA2}_{(n,n)} : \bar{\Sigma}^{n\times n} \to \{0,1\}$ and an inner function $\mathscr{D}_1 : \Gamma \to \bar{\Sigma}$ where Γ is $\{0,1\}^{112\log n}$. Here the function $\mathscr{D}_1$ has different domain from the previous one described in Section 4.5.1.

First we define the outer function $\mathsf{ModA2}_{(n,n)} : \bar{\Sigma}^{n\times n} \to \{0,1\}$ where $\bar{\Sigma} = \Sigma \times \{\vdash, \dashv\}$ to be the modified version of the $\mathsf{A2}_{(n,n)}$ as we have defined $\mathsf{ModA1}_{(n,n)}$.

Think of an input $A \in \bar{\Sigma}^{n\times n}$ as a pair of matrices $B \in \Sigma^{n\times n}$ and $C \in \{\vdash, \dashv\}^{n\times n}$. The function $\mathsf{ModA2}_{(n,n)}$ is defined as

$$\mathsf{ModA2}_{(n,n)}(A) = 1 \text{ iff } \begin{cases} \text{Either, (i)} & \mathsf{A2}_{(n,n)}(B) = 1, \text{ and, all the cells in the} \\ & \text{1-cell-certificate have } \vdash \text{ in the corresponding cell in } C \\ \text{Or, (ii)} & \mathsf{A2}_{(n,n)}(B^T) = 1, \text{ and, all the cells in the} \\ & \text{1-cell-certificate has } \dashv \text{ in the corresponding cell in } C^T \end{cases}$$

Consequently the function $\mathsf{ModA2}_{(n,n)}$ has all the properties as $\mathsf{A2}_{(n,n)}$ as described in Theorem 4.29.

The inner function $\mathscr{D}_1$ (we call it a decoding function) is function from Γ to $\bar{\Sigma}$, where $\Gamma = 112\log n$. Thus our final function is

$$F_{4.2} := \left(\mathsf{ModA2}_{(n,n)} \circ \mathscr{D}_1\right) : \Gamma^{n\times n} \to \{0,1\}.$$

Inner Decoding Function

In the similar fashion of Section 4.5.1 we start by describing the standard "encodings" of a single element of $\bar{\Sigma}$, then we will describe a multiple possible ways of encoding a single element of the $\bar{\Sigma}$ and it's decoding scheme.

"Encodings" of the content of a cell in $\bar{\Sigma}^{n\times n}$ where $\Sigma^{n\times n}$ is a Type_2 *Pointer matrix*

We will encode any element of $\bar{\Sigma}$ using a string of size $112\log n$ bits. Recall from Definition 2.37 that in case of Type_2 *pointer matrix*, an element from $\bar{\Sigma}$ is of the following form $(V,(r_L,c_L),(r_R,c_R),(r_B,c_B),T)$, where V is a binary value and $T \in \{\vdash,\dashv\}$. The overall summary of the encoding is compiled in the Table 4.2:

- **Parts:** We will think of the tuple as 8 objects, namely V, r_L, c_L, r_R, c_R, r_B, c_B and T. We will use $16\log n$ bits to encode each of the first 7 objects. So the encoding of any element of $\bar{\Sigma}$ contains 7 *parts*-each a binary string of length $16\log n$. The value of T will be encoded in a kind-of hidden way.
- **Blocks:** Each of 7 parts will be further broken into 4 blocks of equal length of $4\log n$. One of the block will be a special block called the "encoding block".

Now we will start by describing a "standard-form" encoding of a tuple of the following form $(V,(r_L,c_L),(r_R,c_R),(r_B,c_B),T)$ where $T = \vdash$. Then we will extend it to describe the standard for the encoding of $(V,(r_L,c_L),(r_R,c_R),(r_B,c_B),T)$ where $T = \dashv$. And finally, we will explain all other valid encoding of a tuple $(V,(r_L,c_L),(r_R,c_R),(r_B,c_B),T)$ by describing all the allowed operations on the bits of the encoding.

Table 4.2 Standard form of encoding of a cell of a pointer matrix: Variant 1 (Type II elements)

$\ldots$	B_1 "encoding"-block	B_2	B_3	B_4	Hamming weight
$P1$	$\ell_1\ell_2$, where $\lvert\ell_1\rvert = 2\log n$, and $\lvert\ell_2\rvert = 2\log n - 1 - V$	$4\log n$	$2\log n+1$	$2\log n+2$	$12\log n+2-V$
$P2$	$\mathscr{E}(r_L,0)$	$2\log n+3$	$2\log n+1$	$2\log n+2$	$7\log n+6$
$P3$	$\mathscr{E}(0,c_L)$	$2\log n+4$	$2\log n+1$	$2\log n+2$	$7\log n+7$
$P4$	$\mathscr{E}(r_R,0)$	$2\log n+5$	$2\log n+1$	$2\log n+2$	$7\log n+8$
$P5$	$\mathscr{E}(0,c_R)$	$2\log n+6$	$2\log n+1$	$2\log n+2$	$7\log n+9$
$P6$	$\mathscr{E}(r_B,0)$	$2\log n+7$	$2\log n+1$	$2\log n+2$	$7\log n+10$
$P7$	$\mathscr{E}(0,c_B)$	$2\log n+8$	$2\log n+1$	$2\log n+2$	$7\log n+11$

Standard form of encoding of element $(V,(r_L,c_L),(r_R,c_R),(r_B,c_B),\vdash)$ by a $112\log n$ bit string that is broken into 7 parts $P1,\ldots,P7$ of equal size and each Part is further broken into 4 Blocks B_1,B_2,B_3 and B_4. So all total there are 24 blocks each containing a $4\log n$-bit string. For the standard form of encoding of element $(V,(r_L,c_L),(r_R,c_R),(r_B,c_B),\dashv)$ we encode $(V,(r_L,c_L),(r_R,c_R),(r_B,c_B),\vdash)$ in the standard form as described in the table and then apply the $\mathsf{Swap}_{\frac{1}{2}}$ on each block. The last column of the table indicates the Hamming weight of each Part.

Standard form encoding of $(V,(r_L,c_L),(r_R,c_R),(r_B,c_B),T)$ **where** $T=\vdash$

For the standard-form encoding we will assume that the information of $V,r_L,c_L,r_R,c_R,r_B,c_B$ are stored in parts $P1,P2,P3,P4,P5$, $P6$ and $P7$ respectively. For all $i\in[7]$, the part P_i with have blocks B_1,B_2,B_3 and B_4, of which the block B_1 will be the encoding-block. The description of the standard-form encoding is also compiled in the Table 4.2.

Standard form encoding of $(V,(r_L,c_L),(r_R,c_R),(c_B),T)$ **where** $T=\dashv$

For obtaining a standard-form encoding of $(V,(r_L,c_L),(r_R,c_R),(r_B,c_B),T)$ where $T=\dashv$, first we encode $(V,(r_L,c_L),(r_R,c_R),(r_B,c_B),T)$ where $T=\vdash$ using the standard-form encoding in Table 4.2. Let $(P1,P2,\ldots,P7)$ be the standard-form encoding of $(V,(r_L,c_L),(r_R,c_R),(r_B,c_B),T)$ where $T=\vdash$. Now for each of the block apply the following $\mathsf{Swap}_{\frac{1}{2}}$ operator.

Valid permutation of the standard form

Now we will give a set of valid permutations to the bits of the encoding of any element of $\bar{\Sigma}$. The set of valid permutations are classified into into 3 categories:

1. Part-permutation: The 7 parts can be permuted using any permutation from S_7
2. Block-permutation: In each of the part, the 4 blocks (say B_1, B_2, B_3, B_4) can be permuted is two ways. (B_1, B_2, B_3, B_4) can be send to one of the following
   (a) Simple Block Swap: (B_3, B_4, B_1, B_2)
   (b) Block Flip: $(B_2, B_1, \mathsf{flip}(B_3), \mathsf{flip}(B_4))$

We are now ready to describe the decoding function $\mathscr{D}_1$ from $\Gamma = \{0,1\}^{112\log n}$ to $\bar{\Sigma}$.

- Identify the parts containing the encoding of V, r_L, c_L, r_R, c_R, r_B and c_B. This is possible because every part has a unique Hamming weight.
- For each part identify the blocks. This is also possible as in any part all the blocks have distinct Hamming weight. By seeing the positions of the blocks one can understand if flip was applied and to what and using that one case revert the blocks back to the standard-form.
- In the part containing the encoding of V consider the encoding-block. If the block is of the form $\{(\ell_1\ell_2)\, \textit{such that} |\ell_1| = 2\log n, |\ell_2| \leq 2\log n - 1\}$ then $T = \{\vdash\}$. If the block is of the form $\{(\ell_2\ell_1)\, \textit{such that} |\ell_1| = 2\log n, |\ell_2| \leq 2\log n - 1\}$ then $T = \{\dashv\}$.
- By seeing the encoding block we can decipher the original values and the pointers.
- If the $112\log n$ bit string doesn't have the form of a valid encoding, then decode it as $(0, \perp, \perp, \perp)$.

4.6.2 Proof of transitivity of the function

Theorem 4.38. *$F_{4.2}$ is transitive under the action of the transitive group* G_1.

The group G_1 is formed with same permutation sets as G except S_6 that was acting on the *Part*. Instead of S_6 we will take S_7 as Part-permutation. Proof of Theorem 4.38 follows from similar argument of Theorem 4.35. Here everything else is similar as function $F_{1.4}$ except *Part*.

Properties of the function

Claim 4.39. *Zero error randomized query complexity of $F_{4.2}$ is $\Omega(n^2)$.*

Proof. The function $\mathsf{ModA2}_{(n,n)}$ is a "harder" function than $\mathsf{A2}_{(n,n)}$. So $\mathsf{R}_0(\mathsf{ModA2}_{(n,n)})$ is at least that of $\mathsf{R}_0(\mathsf{A2}_{(n,n)})$. Now since, $F_{4.2}$ is $\left(\mathsf{ModA2}_{(n,n)} \circ \mathscr{D}_1\right)$ so clearly the $\mathsf{R}_0(F_{4.2})$ is at least $(\mathsf{R}_0(\mathsf{A2}_{(n,n)})/\log n)$ by Theorem 2.20. From Theorem 4.29 we have $\mathsf{R}_0(F_{4.2})$ at least $\Omega(n^2)$. □

The following Claim 4.40 that follows from the definition of the function $\mathsf{ModA2}_{(n\times n)}$.

Claim 4.40. *The following are some properties of the function* $\mathsf{ModA2}_{(n,n)}$

1. $\mathsf{R}(\mathsf{ModA2}_{(n,n)}) \leq 2\mathsf{R}(\mathsf{A2}_{(n,n)}) + O(n\log n)$
2. $\mathsf{Q_E}(\mathsf{ModA2}_{(n,n)}) \leq 2\mathsf{Q_E}(\mathsf{A2}_{(n,n)}) + O(n\log n)$
3. $\deg(\mathsf{ModA2}_{(n,n)}) \leq 2\deg(\mathsf{A2}_{(n,n)}) + O(n\log n)$

Finally, from Theorem 2.20 we see that $\mathsf{R}_0(F_{4.2})$, $\mathsf{Q_E}(F_{4.2})$ and $\deg(F_{4.2})$ are at most $O(\mathsf{R}_0(\mathsf{ModA2}_{(n\times n)} \cdot \log n)$, $O(\mathsf{Q_E}(\mathsf{ModA2}_{(n\times n)} \cdot \log n)$ and $O(\deg(\mathsf{ModA2}_{(n\times n)} \cdot \log n)$, respectively. So combining this fact with Claim 4.39, Claim 4.40 and Theorem 4.29 (from [ABB+17]) we have Theorem 4.2.

4.7 Separation between other complexity measures

Another variant of the pointer function was constructed in [ABK16] which demonstrates the gap between randomized query complexity(R) and complexity measures like approximate degree and degree. In this section we show how to obtain a transitive version of this function, thus proving the following separations: There exists a transitive function $F_{4.3}$ such that

$$\mathsf{R}(F_{4.3}) = \widetilde{\Omega}(\widetilde{\deg}(F_{4.3})^4), \qquad \mathsf{R}(F_{4.3}) = \widetilde{\Omega}(\deg(F_{4.3})^2).$$

4.7.1 Transitive pointer function $F_{4.3}$ for Theorem 4.3

In [ABB+17] the function $\mathsf{A3}_{(1,n,n)}$ achieves the separation between R and $\widetilde{\deg}$. Before that [GJPW18b] introduced a function that achieves quadratic separation between R and deg. The function $\mathsf{A3}$ is motivated from [GJPW18b] function and generates the same separation between R and deg. We will give a transitive function that is motivated from [ABB+17] and achieves the separation between R vs. deg and R vs. $\widetilde{\deg}$ which is same as general function.

First let us define the outer function $\mathsf{ModA3}_{(1,n,n)} : \bar{\Sigma}^{n\times n} \to \{0,1\}$ where $\bar{\Sigma} = \Sigma \times \{\vdash, \dashv\}$ to be the modified version of the $\mathsf{A3}_{(1,n,n)}$ as we have defined $\mathsf{ModA1}_{(1,n,n)}$.

Think of an input $A \in \bar{\Sigma}^{n\times n}$ as a pair of matrices $B \in \Sigma^{n\times n}$ and $C \in \{\vdash, \dashv\}^{n\times n}$. The function $\mathsf{ModA3}_{(1,n,n)}$ is defined as

$$\mathsf{ModA3}_{(1,n,n)}(A) = 1 \text{ iff } \begin{cases} \text{Either, (i)} & \mathsf{A3}_{(1,n,n)}(B) = 1, \text{ and, all the cells in the} \\ & \text{1-cell-certificate have } \vdash \text{ in the corresponding cell in } C \\ \text{Or, (ii)} & \mathsf{A3}_{(1,n,n)}(B^T) = 1, \text{ and, all the cells in the} \\ & \text{1-cell-certificate has } \dashv \text{ in the corresponding cell in } C^T \end{cases}$$

Consequently the function $\mathsf{ModA3}_{(1,n,n)}$ has all the properties of $\mathsf{A3}_{(1,n,n)}$ as described in Theorem 4.31. Here the input matrix to the function $\mathsf{A3}_{(1,n,n)}$ is a Type_2 pointer matrix which is same as $\mathsf{A2}_{(n,n)}$. Hence we can encode the elements in $\bar{\Sigma}$ in a similar fashion that we have done in Section 4.6.1. That's why we can choose the same decoding function $\mathscr{D}_1$ function as of $F_{4.2}$. Thus our final function is

$$F_{4.3} := \left(\mathsf{ModA3}_{(1,n,n)} \circ \mathscr{D}_1\right) : \Gamma^{n\times n} \to \{0,1\}$$

where $\mathscr{D}_1 : \Gamma \to \bar{\Sigma}$ and $\Gamma = 112\log n$.

From the proof of Theorem 4.38 we have the following Theorem:

Theorem 4.41. *$F_{4.3}$ is transitive under the action of the transitive group G_1.*

Claim 4.42. *Randomized query complexity of $F_{4.3}$ is $\Omega(n^2)$.*

Proof. From the construction of $\mathsf{ModA3}_{(1,n,n)}$ one can note that $\mathsf{ModA3}_{(1,n,n)}$ is a "harder" function than $\mathsf{A3}_{(1,n,n)}$. So $\mathsf{R}(\mathsf{ModA3}_{(1,n,n)})$ is at least that of $\mathsf{R}(\mathsf{A3}_{(1,n,n)})$.

Now since, $F_{4.3}$ is $\left(\mathsf{ModA3}_{(1,n,n)} \circ \mathscr{D}_1\right)$ from Theorem 2.20 it follows that $\mathsf{R}(F_{4.3})$ is at least $\mathsf{R}(\mathsf{A3}_{(1,n,n)})$. In [ABB+17] they proved that $\mathsf{R}(\mathsf{A3}_{(1,n,n)})$ is $\Omega(n^2)$. So $\mathsf{R}(F_{4.3})$ is at least $\Omega(\frac{n^2}{\log n})$. □

Claim 4.43. *The following are some properties of the function $\mathsf{ModA3}_{(1,n,n)}$*

1. $\widetilde{\deg}(\mathsf{ModA3}_{(1,n,n)}) \leq 2\widetilde{\deg}(\mathsf{A3}_{(1,n,n)}) + O(n\log n)$
2. $\deg(\mathsf{ModA3}_{(1,n,n)}) \leq 2\deg(\mathsf{A3}_{(1,n,n)}) + O(n\log n)$

Finally from Theorem 2.20, Theorem 4.31 and Claim 4.43 it follows that $\widetilde{\deg}(F_{4.3}) = O(\sqrt{n})$. Also from Theorem 2.20, Observation 4.32 and Claim 4.43 it follows that $\deg(F_{4.3}) = (n)$. From Theorem 4.41 and Claim 4.42 we have Theorem 4.3.

4.7.2 Transitive pointer function $F_{4.4}$ for Theorem 4.4

In [ABB^{+}17] they proved that $\mathsf{A3}_{(n,n,n^2)}$ achieves the separation between R_0 and Q where the input is a $n \times n^2$ Type_2 *pointer matrix*. Till now all of our encoding was precisely for $n \times n$ matrix. Those encoding schemes directly cannot be generalized for any non-square matrix. But using some modifications to our encoding scheme in the previous sections, we give an encoding for $n \times n^2$ matrix also. Before going into the encoding scheme we will define some useful terms and representation of a matrix that we are going to use in the encoding scheme.

Definition 4.44 (*Brick*). *For any $n \times n^2$ matrix M we divide the matrix into total n number of square matrices each of size $n \times n$, which we refer to as* Brick. *Thus there is total n number of* Bricks, *which we denote as $\{b_1, b_2, \ldots, b_n\}$ and each b_i is a sub-matrix of M. Now let us denote the n rows and n columns of an $n \times n$ matrix as $\{r_1, \ldots, r_n\}$ and $\{c_1, \ldots, c_n\}$ respectively. So, each entries of M can be uniquely defines as $\{(r,c,b) | r,c,b \in [n]\}$ where r,c and b is the row number, column number and brick number respectively. In this set-up we will write an $n \times n^2$ matrix A as $A_{(n \times n \times n)}$.*

In a general setup when we swap between row and column we get the transpose of that matrix. In our row, column and brick setup we define two new concept similar to transpose matrix.

Definition 4.45 ($A^{\top}, A^{\dashv}$). *Denote each entry of a $n \times n^2$ matrix $A_{(n \times n \times n)}$ by $(row, column, brick)$ then, define $A^{\top}_{(n \times n \times n)}$ to be the matrix where (r,c,b)-th element of $A_{(n \times n \times n)}$ is the (b,r,c)-th element of $A^{\top}_{(n \times n \times n)}$. Similarly define $A^{\dashv}_{(n \times n \times n)}$ to be the matrix where (r,c,b)-th element of $A_{(n \times n \times n)}$ is the (c,b,r)-th element of $A^{\dashv}_{(n \times n \times n)}$.*

Now we are ready to define our outer function.

The outer function

The outer function is a modified version of the $\mathsf{A3}_{(n,n,n^2)}$. Recall, from Definition 4.30, the function $\mathsf{A3}_{(n,n,n^2)}$ takes as input a $(n \times n^2)$ Type_2 *pointer matrix* over Σ.

Let us define $\mathsf{ModA3*}_{(n,n,n^2)} : \tilde{\Sigma}^{n \times n^2} \rightarrow \{0,1\}$ where the alphabet set is $\tilde{\Sigma} = \Sigma \times \{\vdash, \top, \dashv\}$. We can think of an input $M \in \tilde{\Sigma}^{n \times n^2}$ as a pair of matrices $B \in \Sigma^{n \times n^2}$ and $C \in \{\vdash, \top, \dashv\}^{n \times n^2}$. The function $\mathsf{ModA3*}_{(n,n,n^2)}$ is defined as

$$\mathsf{ModA3*}_{(n,n,n^2)}(M) = 1 \text{ iff } \begin{cases} \text{Either,} & \textbf{(i) } \mathsf{A3}_{(n,n,n^2)}(B) = 1, \text{ and, all the cells in the} \\ & \text{1-cell-certificate have } \vdash \text{ in the corresponding cell in } C, \\ \text{Or,} & \textbf{(ii) } \mathsf{A3}_{(n,n,n^2)}(B^{\top}) = 1, \text{ and, all the cells in the} \\ & \text{1-cell-certificate has } \top \text{ in the corresponding cell in } C^{\top}, \\ \text{Or,} & \textbf{(iii) } \mathsf{A3}_{(n,n,n^2)}(B^{\dashv}) = 1, \text{ and, all the cells in the,} \\ & \text{1-cell-certificate has } \dashv \text{ in the corresponding cell in } C^{\dashv}. \end{cases}$$

At most one of conditions (i), (ii) and (iii) can be true for an input. From this it is easy to verify that the function $\mathsf{ModA3*}_{(n,n,n^2)}$ has all the properties as $\mathsf{A3}_{(n,n,n^2)}$ as described in Theorem 4.31.

Thus our final function is

$$F_{4.4} := \left(\mathsf{ModA3*}_{(n,n,n^2)} \circ \mathscr{D}_2\right) : \Gamma^{n^3} \to \{0,1\},$$

where the inner function $\mathscr{D}_2$ (we call it a decoding function) is a function from Γ to $\bar{\Sigma}$ where $\Gamma = 240\log n$.

Inner Decoding Function

Any element of Type_2 pointer matrix is of the form $(V, \text{LPointer}, \text{RPointer}, \text{BPointer})$ where $V \in \{0,1\}$ and LPointer, RPointer and BPointer are the pointers to the other cell of the matrix. Here pointers are of the form (i, j) where i is the row number and j is the column number. Now in our setup (Definition 4.44) we have represented the cells of an $n \times n^2$ matrix as (r, c, b) (where $r, c, b \in [n]$), where r, c and b specify the row, column and brick of that cell, respectively.

"Encodings" of the content of a cell in $\bar{\Sigma}^{n\times n^2}$ where $\Sigma^{n\times n^2}$ is a Type_2 *Pointer matrix*

We will encode any tuple of the form $(V, (r_L, c_L, b_L), (r_R, c_R, b_R), (r_B, c_B, b_B), T)$ which is an element of $\bar{\Sigma}$ using a string of size $240\log n$ bits. The overall summary of the encoding is as follows:

- **Parts:** We will think of the tuple as 11 objects, namely V, r_L, c_L, b_L r_R, c_R, b_R, r_B, c_B, b_B and T. We will use a $24\log n$ bit string to encode each of the first 10 objects and the enoding of T will be implicit. So the encoding of any element of $\bar{\Sigma}$ contains 10 *parts*-each a binary string of length $24\log n$.

- **Blocks:** Each of the 10 parts will be further broken into 4 blocks of equal length of $6\log n$. One of the block will be a special block called the "encoding block".

Let us start by describing a "standard-form" encoding of a tuple of the following form $(V,(r_L,c_L,b_L),(r_R,c_R,b_R),(r_B,c_B,b_B),T)$ where $T = \vdash$. Then extend it for $T = \top$ and $T = \dashv$. Finally, we will explain all other valid encodings of a tuple $(V,(r_L,c_L,b_L),(r_R,c_R,b_R),(r_B,c_B,b_B),T)$ by describing all the allowed operations on the bits of the encoding.

Using the Balanced-Binary Encoding we can define an hybrid encoding for the set of rows and columns.

Definition 4.46. *Given a set R of n rows $r_1,\dots,r_n$, a set C of n columns $c_1,\dots,c_n$ and a set B of n* bricks *$b_1,\dots,b_n$ we define the balanced-pointer-encoding function*

$$\mathcal{E}' : (R\times\{0\}\times\{0\})\cup(\{0\}\times C\times\{0\})\cup(\{0\}\times\{0\}\times B)\to\{0,1\}^{6\log n}$$

as follows:

$$\begin{aligned}\mathcal{E}'(r_i,0,0) &= bb(i)\cdot 0^{2\log n}\cdot 0^{2\log n},\\ \mathcal{E}'(0,c_i,0) &= 0^{2\log n}\cdot bb(i)\cdot 0^{2\log n} \text{ and}\\ \mathcal{E}'(0,0,b_i) &= 0^{2\log n}\cdot 0^{2\log n}\cdot bb(i).\end{aligned}$$

Definition 4.47. *Given a binary string $x_1,\dots,x_{3k}\in\{0,1\}^{3k}$, the operation* $\mathsf{Rotation}_1$ *and* $\mathsf{Rotation}_2$ *act on the string by*

$$\mathsf{Rotation}_1(x_1,\dots,x_{3k}) = x_{2k+1},\dots,x_{3k},x_1\dots,x_k,x_{k+1},\dots,x_{2k}$$

and

$$\mathsf{Rotation}_2(x_1,\dots,x_{3k}) = x_{k+1},\dots,x_{2k},x_{2k+1}\dots,x_{3k},x_1,\dots,x_k$$

Now we are set to describe the standard form of encoding of the tuple of the following form $(V,(r_L,c_L,b_L),(r_R,c_R,b_R),(r_B,c_B,b_B),T)$ where $T = \vdash$.

Standard form encoding of $(V,(r_L,c_L,b_L),(r_R,c_R,b_R),(r_B,c_B,b_B),T)$ where $T = \vdash$

For the standard-form encoding we will assume that the information of $V,r_L,c_L,b_L,r_R,c_R,b_R,r_B,c_B,b_B$ are stored in parts $P1,P2,P3,P4,P5$, $P6$, $P7$, $P8$, $P9$ and $P10$ respectively. For all $i\in[10]$, the part P_i with have blocks B_1,B_2,B_3 and B_4, of which the block B_1 will be

...	B_1	B_2	B_3	B_4	Hamming weight
$P1$	$\ell_1\ell_3\ell_2$ where $\lvert\ell_1\rvert = 2\log n$ $\lvert\ell_2\rvert = 2\log n - 1$, and $\lvert\ell_3\rvert = 2\log n - 2 - V$	$6\log n$	$2\log n + 1$	$2\log n + 2$	$16\log n - V$
$P2$	$\mathscr{E}'(i)$	$2\log n + 3$	$2\log n + 1$	$2\log n + 2$	$7\log n + 6$
$P3$	$\mathscr{E}'(j)$	$2\log n + 4$	$2\log n + 1$	$2\log n + 2$	$7\log n + 7$
$P4$	$\mathscr{E}'(k)$	$2\log n + 5$	$2\log n + 1$	$2\log n + 2$	$7\log n + 8$
$P5$	$\mathscr{E}'(t)$	$2\log n + 6$	$2\log n + 1$	$2\log n + 2$	$7\log n + 9$
$P6$	$\mathscr{E}'(m)$	$2\log n + 7$	$2\log n + 1$	$2\log n + 2$	$7\log n + 10$
$P7$	$\mathscr{E}'(n)$	$2\log n + 8$	$2\log n + 1$	$2\log n + 2$	$7\log n + 11$
$P8$	$\mathscr{E}'(p)$	$2\log n + 9$	$2\log n + 1$	$2\log n + 2$	$7\log n + 12$
$P9$	$\mathscr{E}'(q)$	$2\log n + 10$	$2\log n + 1$	$2\log n + 2$	$7\log n + 13$
$P10$	$\mathscr{E}'(r)$	$2\log n + 11$	$2\log n + 1$	$2\log n + 2$	$7\log n + 14$

Table 4.3 Standard form of a cell of a pointer matrix: Variant 3

Standard form of encoding of $(V,(r_L,c_L,b_L),(r_R,c_R,b_R),(r_B,c_B,b_B),\vdash)$ by a $240\log n$ bit string that is broken into 10 parts $P1,\ldots,P10$ of equal size and each Part is further broken into 4 Blocks B_1,B_2,B_3 and B_4. So all total there are 40 blocks each containing a $6\log n$-bit string. [2.] For the standard form of encoding of $(V,(r_L,c_L,b_L),(r_R,c_R,b_R),(r_B,c_B,b_B),\top)$ we encode $(V,(r_L,c_L,b_L),(r_R,c_R,b_R),(r_B,c_B,b_B),\vdash)$ in the standard form as described in the table and then apply the $\mathsf{Rotation}_1$ on each block. [3.] For the standard form of encoding of $(V,(r_L,c_L,b_L),(r_R,c_R,b_R),(r_B,c_B,b_B),\dashv)$ we encode $(V,(r_L,c_L,b_L),(r_R,c_R,b_R),(r_B,c_B,b_B),\vdash)$ in the standard form as described in the table and then apply the $\mathsf{Rotation}_2$ on each block. The last column of the table indicates the unique Hamming weight of each Part.

the encoding-block. The description of the standard-form encoding is also compiled in the Table 4.3.

- For *part* $P1$ (that is the encoding of V) the encoding block B_1 will store $\ell_1 \cdot \ell_2 \cdot \ell_3$ where ℓ_1 is any $2\log n$ bit binary string with Hamming weight $2\log n$, ℓ_2 is any $2\log n$ bit binary string with Hamming weight $2\log n - 1$ and ℓ_3 is any $2\log n$ bit binary string with Hamming weight $2\log n - 2 - V$. Block B_2, B_3 and B_4 will store any $6\log n$ bit string with Hamming weight $6\log n$, $2\log n + 1$ and $2\log n + 2$ respectively.

- For *part* $P2$ (this is the encoding of r_L) the encoding block $B1$ will store the string $\mathscr{E}'(r_L, 0, 0)$. Block B_2, B_3 and B_4 will store any $6\log n$ bit string with Hamming weight $2\log n + 3$, $2\log n + 1$ and $2\log n + 2$ respectively.

- For *part* $P3$ (this is the encoding of c_L) the encoding block $B1$ will store the string $\mathscr{E}'(0, c_L, 0)$. Block B_2, B_3 and B_4 will store any $6\log n$ bit string with Hamming weight $2\log n + 4$, $2\log n + 1$ and $2\log n + 2$ respectively.

- For *part* $P4$ (this is the encoding of b_L) the encoding block $B1$ will store the string $\mathscr{E}'(0, 0, b_L)$. Block B_2, B_3 and B_4 will store any $6\log n$ bit string with Hamming weight $2\log n + 5$, $2\log n + 1$ and $2\log n + 2$ respectively.

- For *part* $P5$ (this is the encoding of r_R) the encoding block $B1$ will store the string $\mathscr{E}'(r_R, 0, 0)$. Block B_2, B_3 and B_4 will store any $6\log n$ bit string with Hamming weight $2\log n + 6$, $2\log n + 1$ and $2\log n + 2$ respectively.

- For *part* $P6$ (this is the encoding of c_R) the encoding block $B1$ will store the string $\mathscr{E}'(0, c_R, 0)$. Block B_2, B_3 and B_4 will store any $6\log n$ bit string with Hamming weight $2\log n + 7$, $2\log n + 1$ and $2\log n + 2$ respectively.

- For *part* $P7$ (this is the encoding of b_R) the encoding block $B1$ will store the string $\mathscr{E}'(0, 0, b_R)$. Block B_2, B_3 and B_4 will store any $6\log n$ bit string with Hamming weight $2\log n + 8$, $2\log n + 1$ and $2\log n + 2$ respectively.

- For *part* $P8$ (this is the encoding of r_B) the encoding block $B1$ will store the string $\mathscr{E}'(r_B, 0, 0)$. Block B_2, B_3 and B_4 will store any $6\log n$ bit string with Hamming weight $2\log n + 9$, $2\log n + 1$ and $2\log n + 2$ respectively.

- For *part* $P9$ (this is the encoding of c_B) the encoding block $B1$ will store the string $\mathscr{E}'(0, c_B, 0)$. Block B_2, B_3 and B_4 will store any $6\log n$ bit string with Hamming weight $2\log n + 10$, $2\log n + 1$ and $2\log n + 2$ respectively.

- For *part* $P10$ (this is the encoding of b_B) the encoding block $B1$ will store the string $\mathscr{E}'(0,0,b_B)$. Block B_2, B_3 and B_4 will store any $6\log n$ bit string with Hamming weight $2\log n+11$, $2\log n+1$ and $2\log n+2$ respectively.

Standard form encoding of $(V,(r_L,c_L,b_L),(r_R,c_R,b_R),(r_B,c_B,b_B),T)$ **where** $T=\top$ **and** $T=\dashv$

For obtaining a standard-form encoding of $(V,(r_L,c_L,b_L),(r_R,c_R,b_R),(r_B,c_B,b_B),T)$ where $T=\top$, first we encode $(V,(r_L,c_L,b_L),(r_R,c_R,b_R),(r_B,c_B,b_B),T)$ where $T=\vdash$ using the standard-form encoding. Let $(P1,P2,\ldots,P10)$ be the standard-form encoding of the following tuple $(V,(r_L,c_L,b_L),(r_R,c_R,b_R),(r_B,c_B,b_B),T)$ where $T=\vdash$. Now for each of the block apply the following $\mathsf{Rotation}_1$ operator. To get the encoding for $T=\dashv$ apply $\mathsf{Rotation}_2$ on each block of standard encoding of the tuple $(V,(r_L,c_L,b_L),(r_R,c_R,b_R),(r_B,c_B,b_B),\vdash)$.

Valid permutation of the standard form

Now we will give a set of valid permutations to the bits of the encoding of any element of $\bar{\Sigma}$. The set of valid permutations are classified into into 3 categories:

1. Part-permutation: The 10 parts can be permuted using any permutation from S_{10}
2. Block-permutation: In each of the part, the 4 blocks (say B_1,B_2,B_3,B_4) can be permuted is two ways. (B_1,B_2,B_3,B_4) can be send to one of the following
   (a) Simple Block Swap: (B_3,B_4,B_1,B_2)
   (b) Block Flip: $(B_2,B_1,\mathsf{flip}(B_3),\mathsf{flip}(B_4))$

The decoding function $\mathscr{D}_2:\{0,1\}^{240\log n}\to\bar{\Sigma}$

We are now ready to describe the decoding function from $\Gamma=\{0,1\}^{240\log n}$ to $\bar{\Sigma}$.

- Identify the parts containing the encoding of V, r_L, c_L,b_L, r_R, c_R, b_R,r_B, c_B and b_B. This is possible because every part has a unique Hamming weight.
- For each part identify the blocks. This is also possible as in any part all the blocks have distinct Hamming weight. By seeing the positions of the blocks one can understand if flip was applied and to what and using that one case revert the blocks back to the standard-form.

- In the part containing the encoding of V consider the encoding-block. If the block is of the form $\{(\ell_1\ell_2\ell_3)\, \textit{such that}\, |\ell_1| = 2\log n, |\ell_2| = 2\log n - 1 \textit{ and } |\ell_3| \leq 2\log n - 2\}$ then $T = \{\vdash\}$. If the block is of the form $\{(\ell_3\ell_1\ell_2)\, \textit{such that}\, |\ell_1| = 2\log n, |\ell_2| = 2\log n - 1$ *and* $|\ell_3| \leq 2\log n - 2\}$ then $T = \{\top\}$. If the block is of the form $\{(\ell_2\ell_3\ell_1)\, \textit{such that}\, |\ell_1| = 2\log n, |\ell_2| = 2\log n - 1 \textit{ and } |\ell_3| \leq 2\log n - 2\}$ then $T = \{\dashv\}$.
- By seeing the encoding block we can decipher the original values and the pointers.
- If the $240\log n$ bit string doesn't have the form of a valid encoding, then decode it as $(0, \perp, \perp, \perp)$.

Proof of Transitivity of the function

To prove the function $F_{4.4}$ is transitive we start with describing the transitive group for which the $F_{4.4}$ is transitive.

The Transitive Group

We will now describe the group $\mathscr{T}$ acting on the cells of the matrix and a group $\widehat{\mathscr{T}}$ acting on the indices of a $6\log n$ bit string. First consider the group $\mathsf{Bt}_{2\log n}$ acting on the set $[2\log n]$. We will assume that $\log n$ is a power of 2. The matrix has rows $r_1, \dots, r_n$, columns $c_1, \dots, c_n$ and bricks $b_1, \dots, b_n$. We have used the encoding function $\mathscr{E}'$ to encode the rows, columns and bricks. So the index of the rows, columns and bricks are encoded using a $6\log n$ bit string. From the Definition of $\mathscr{E}'$ and the group $\mathsf{Bt}_{2\log n}$ the following claim from Definition 4.46 and Claim 4.16.

Claim 4.48. *For any elementary permutation $\widehat{\sigma}$ in $\mathsf{Bt}_{2\log n} \times \mathsf{Bt}_{2\log n} \times \mathsf{Bt}_{2\log n}$ that takes $\mathscr{E}'(x)$ to $\mathscr{E}'(y)$ there exists a permutation $\sigma \in \mathsf{S}_n \times \mathsf{S}_n \times \mathsf{S}_n$ that takes x to y where $x, y \in (R \times \{0\} \times \{0\}) \cup (\{0\} \times C \times \{0\}) \cup (\{0\} \times \{0\} \times B)$. That is*

$$\widehat{\sigma}[\mathscr{E}'(x)] = \mathscr{E}'(\sigma(x)).$$

Note that $\mathsf{Bt}_{2\log n} \times \mathsf{Bt}_{2\log n} \times \mathsf{Bt}_{2\log n}$ is not transitive on $6\log n$ bit string. Let us define the group $\widehat{\mathscr{T}} \subset \mathsf{S}_{6\log n}$ containing the following permutations:

1. all permutations in $\mathsf{Bt}_{2\log n} \times \mathsf{Bt}_{2\log n} \times \mathsf{Bt}_{2\log n}$,
2. $\mathsf{Rotation}_1$ to be applied on Block ($6\log n$-bit),
3. $\mathsf{Rotation}_2$ to be applied on Block ($6\log n$-bit).

$\widehat{\mathscr{T}}$ is the group generated by the above set of permutations and clearly $\widehat{\mathscr{T}}$ is transitive.

The group $\mathscr{T}$ will be the resulting group of permutations on the cells of the matrix induced by the group $\widehat{\mathscr{T}}$ acting on the $6\log n$ bit. The group $\mathscr{T}$ is generated by the following permutations:

1. all permutations induced by $\mathsf{Bt}_{2\log n} \times \mathsf{Bt}_{2\log n} \times \mathsf{Bt}_{2\log n}$ that acts on the cell,
2. for all $i \in [n]$ row i is goes to column i, column i goes to brick i and brick i goes to row i. That is $\mathsf{Rotation}_1[\mathscr{E}'(r_i, c_i, b_i)] = \mathscr{E}'(\sigma(b_i, r_i, c_i))$ for all $i \in [n]$ where σ acts on the cell.
3. for all $i \in [n]$ row i is goes to brick i, column i goes to *row i* and brick i goes to column i. That is $\mathsf{Rotation}_2[\mathscr{E}'(r_i, c_i, b_i)] = \mathscr{E}'(\sigma(c_i, b_i, r_i))$ for all $i \in [n]$ where σ acts on the cell.

From the construction of $\widehat{\mathscr{T}}$ and $\mathscr{T}$ we have the following claim:

Claim 4.49. *For any permutation $\widehat{\sigma}$ in $\widehat{\mathscr{T}} \subset \mathsf{S}_{6\log n}$ acting on the $6\log n$ bit string there exists a permutation σ in $\mathscr{T} \subset \mathsf{S}_{3n}$ acting on the cell such that $\widehat{\sigma}[(\mathscr{E}'(x)] = \mathscr{E}'(\sigma(x))$ for all x in the domain of $\mathscr{E}'$.*

Observation 4.50. *One special set of permutation in $\widehat{\mathscr{T}}$ is called the "ith-bit-flips" or simply "bit-flips". For all $1 \le i \le 3\log n$ the ith-bit-flip applies the transposition $(2i-1, 2i)$ to the indices of the balanced-pointer-encoding.*

Since the $\mathscr{E}'$-encoding of the row $(r_k, 0, 0)$ uses the balanced binary representation of k in the first half and all zero sting in the second and third half, the j-th bit in the binary representation of k is stored in the $2j-1$ and $2j$-th bit in the $\mathscr{E}'$-encoding of r_i. So the j-th-bit-flip acts on the sets of rows by swapping all the rows with 1 in the j-th bit of their index with the corresponding rows with 0 in the j-th bit of their index. Also, if $i > \log n$ then there is no effect of the i-th-bit-flip operation on the set of rows.

One can make a similar observation for the columns and Bricks *also.*

From Definition 4.46, Claim 4.49 and Observation 4.50 we claim the following:

Claim 4.51. *The group $\mathscr{T}$ acting on the cells of of the matrix is a transitive group. That is, for all $1 \le i_1, j_1, i_2, j_2, k_1, k_2 \le n$ there is a permutation $\widehat{\sigma} \in \widehat{\mathscr{T}}$ such that $\widehat{\sigma}[\mathscr{E}'(i_1, 0, 0)] = \mathscr{E}'(i_2, 0, 0)$, $\widehat{\sigma}[\mathscr{E}'(0, j_1, 0)] = \mathscr{E}'(0, j_2, 0)$ and $\widehat{\sigma}[\mathscr{E}'(0, 0, k_1)] = \mathscr{E}'(0, 0, k_2)$. Or in other words, there is a $\sigma \in \mathscr{T}$ acting on the cell of the matrix that would take the cell corresponding to row r_{i_1}, column c_{j_1} and brick b_{k_1} to the cell corresponding to row r_{i_2}, column c_{j_2} and brick b_{k_2}.*

From the Claim 4.51 we see the group $\mathscr{T}$ acting on the cells of of the matrix is a transitive. But it does not touch the contains within the cells of the matrix. But the input to the function $F_{4.4}$ contains element of $\Gamma = \{0,1\}^{240\log n}$ in each cell. So we now need to extend the group $\mathscr{T}$ to a group G_2 that acts on all the indices of all the bits of the input to the function $F_{4.4}$.

Recall that the input to the function $F_{4.4}$ is a $(n \times n^2)$-matrix with each cell of matrix containing a binary string of length $240\log n$ and for each of the cells the binary string contained in it has 10 *parts* of size $24\log n$ each and each part has 4 blocks of size $6\log n$ each. We classify the generating elements of the group G_2 into 4 categories:

1. Part-permutation: In each of the cells the 10 parts can be permuted using any permutation from S_{10}
2. Block-permutation: In block the 4 blocks (say B_1, B_2, B_3, B_4) can be permuted is two ways. (B_1, B_2, B_3, B_4) can be send to one of the following
   (a) Simple Block Swap: (B_3, B_4, B_1, B_2)
   (b) Block Flip (#1): $(B_2, B_1, \mathsf{flip}(B_3), \mathsf{flip}(B_4))$
   (c) Block Flip (#2): $(\mathsf{flip}(B_1), \mathsf{flip}(B_2), B_4, B_3)$
3. Cell-permutation: for any $\sigma \in \mathscr{T}$ one has the do the following two steps simultaneously:
   (a) (Matrix-update) Permute the cells in the matrix according to the permutation σ. This does keeps each of the contains in the cells untouched - it just changes the location of the cells.
   (b) (Pointer-update) For each of blocks in each of the parts in each of the cells permute the indices of the $6\log n$-bit strings according to σ, that is apply $\widehat{\sigma} \in \widehat{\mathscr{T}}$ corresponding to σ(existence of such permutation follows from Claim 4.49).

We now have the following theorem that would prove that the function $F_{4.4}$ is transitive.

Theorem 4.52. *The group* G_2 *is a transitive group.*

Theorem 4.53. *The function* $F_{4.4}$ *is a transitive function under the action of the transitive group* G_2.

Proof of Theorem 4.52. To prove that the group G_2 is transitive we have to show that for any indices $p, q \in [240n^2\log n]$ there is a permutation $\sigma \in \mathsf{G}_2$ that would take p to q. So we can think of the index p as sitting in k_pth position ($1 \leq k_p \leq 6\log n$) in the block B_p of the part P_p

in the (r_p, c_p, b_p)-th cell of the matrix. Similarly, we can think of q as sitting in k_qth position ($1 \leq k_q \leq 6\log n$) in the block B_q of the part P_q in the (r_q, c_q, b_q)-th cell of the matrix.

We will give a step by step technique in which permutations from G_2 can be applied to move p to q.

Step 1 **Get the positions in the block correct:** If $k_p \neq k_q$ then take a permutation $\widehat{\sigma}$ from $\widehat{\mathscr{T}}$ that takes k_p to k_q. Since $\widehat{\mathscr{T}}$ is transitive, such a permutation exists. Apply the cell-permutation from $\mathscr{T}$ corresponding to $\widehat{\sigma}$ on the cells. As a result the index p can be moved to a different cell in the matrix but, by the choice of $\widehat{\sigma}$ its position in the block in which it is will be k_q.

Without loss of generality, we assume the the cell location does not change.

Step 2 **Get the cell correct:** Using a series of "bit-flip" operations change r_p to r_q,c_p to c_q and b_p to b_q. Since one bit-flip operations basically changes one bit in the binary representation of the index of the row or column or brick such a series of operations can be made. This is another Cell-permutation. In this case we always do Matrix-update and Pointer-update simultaneously.

Since each bit-flip operation is executed by applying the bit-flips in each of the blocks so this might have once again changed the position of the index p in the block. But, even if the position in the block changes it must be a flip operation away. Or in other word, since in the beginning of this step $k_p = k_q$, so if k_q is even (or odd) then after the series bit-flip operations the position of p in the block is either k_q or $(k_q - 1)$ (or $(k_q + 1)$).

Step 3 **Align the Part:** Apply a suitable Part-permutation from S_{10} to ensure that the part P_p moves to part P_q.

Step 4 **Align the Block:** Using a suitable combination of Simple Block Swap and Block Flip ensure the Block number gets matched, that is B_p goes to B_q. In this case the cell or the Part does not change. But depending on whether the Block Flip operation is applied the position in the block can again change. But once again, the current position in the block k_p is at most one flip away from k_q.

Step 5 **Apply the final flip:** It might so happen that already we a done after the last step. If not we know that the current position in the block k_p is at most one flip away from k_q. So we apply the suitable Block-flip operation. Thus will not change the cell position, Part number, Block number and the position in the block will match. Hence we are done.

□

Proof of Theorem 4.53. To prove that the function $F_{4.4}$ is transitive we prove that for any elementary operations σ from the group G_2 and for any input Γ^{n^3} the function value does not change even if after the input is acted upon by the permutation σ.

Case 1: σ is a Part-permutation and **Case 2: σ is a Block-permutation** directly follows from our construction of the encoding so that $\mathscr{D}_2$ is invariant under Part permutation and Block-permutation.

Case 3: σ is a Cell-permutation From Observation 4.27 it is enough to prove that when we permute the cells of the matrix we update the points in the cells accordingly.

Let $\sigma \in \mathscr{T}$ be a permutation that permutes only the rows of the matrix. By Claim 4.49, there exist a permutation $\widehat{\sigma} \in \widehat{\mathscr{T}}$ to be applied on every block and so that the contents of the cells will be updated accordingly. Similarly if σ only permute the columns or *Bricks* of the matrix we will be fine.

Finally, if $\mathsf{Rotation}_1$ was applied then for all i row i is swapped with column i, column i is swapped with brick i and brick i is swapped with row i then it is easy to see that $\widehat{\sigma}[\mathscr{E}'(i,0,0)] = \mathscr{E}'(0,i,0)$, $\widehat{\sigma}[\mathscr{E}'(0,i,0)] = \mathscr{E}'(0,0,i)$ and $\widehat{\sigma}[\mathscr{E}'(0,0,i)] = \mathscr{E}'(i,0,0)$. In that case the encoding block of the value part in a cell also gets swapped. This will thus be encoding the T value as $\top$. And so the function value will not be affected as the $T = \top$ will ensure that one should apply the $\mathsf{Rotation}_1$ on the cells of the input before evaluating the function. For $\mathsf{Rotation}_2$ we can argue similarly. Hence, $F_{4.4}$ is transitive. □

Properties of the Function

Claim 4.54. *Zero error randomized query complexity of $F_{4.4}$ is $\Omega(n^3)$.*

Proof. The function $\mathsf{ModA3}*_{(n,n,n^2)}$ is a "harder" function than $\mathsf{A3}_{(n,n,n^2)}$. So $\mathsf{R}_0(\mathsf{ModA3}*_{(n,n,n^2)})$ is at least that of $\mathsf{R}_0(\mathsf{A3}_{(n,n,n^2)})$.

Now since, $F_{4.4}$ is $\left(\mathsf{ModA3}*_{(n,n,n^2)} \circ \mathscr{D}_2\right)$ so clearly the $\mathsf{R}_0(F_{4.4})$ is at least $\mathsf{R}_0(\mathsf{A3}_{(n,n,n^2)})$. From [ABB+17] we have $\mathsf{D}(\mathsf{A3}_{(n,n,n^2)})$ is $\Omega(n^3)$. Hence $\mathsf{R}_0(F_{4.4})$ at least $\Omega(n^3)$. □

The following Claim4.55 that follows from the definition of the function $\mathsf{ModA3}*_{(n,n,n^2)}$.

Claim 4.55. *The following are some properties of the function $\mathsf{ModA3}*_{(n,n,n^2)}$:*

$\mathsf{Q}(\mathsf{ModA3}*_{(n,n,n^2)}) \leq 3\mathsf{Q}(\mathsf{A3}_{(n,n,n^2)}) + O(n^2 \log n)$

Finally, from Theorem 2.20 we see that $\mathsf{Q}(F_{4.4})$ is at most $O(\mathsf{Q}(\mathsf{ModA3}*_{(n,n,n^2)} \cdot \log n)$. So combining this fact with Claim 4.54, Claim 4.55 and Theorem 4.31 (from [ABB$^+$17]) we have Theorem 4.4.

4.7.3 Transitive pointer function $F_{4.5}$for Theorem 4.5

In [ABB$^+$17] they have proved that $\mathsf{A3}_{(1,n,n^2)}$ is such that $\mathsf{R}(\mathsf{A3}_{(1,n,n^2)}) = \Omega(\mathsf{Q_E}(\mathsf{A3}_{(1,n,n^2)})^{1.5}$. In this function $\mathsf{A3}_{(1,n,n^2)}$ number of *marked column* is 1, which is different from $\mathsf{A3}_{(n,n,n^2)}$. But for both of these function the domain is same. So, proof follows directly.

For transitive function first we will define $\mathsf{ModA3}*_{(1,n,n^2)}$ on $\bar{\Sigma}^{n\times n^2}$ similar as $\mathsf{ModA3}*_{(1,n,n^2)}$ in Section 4.7.2. Then define a composition function with outer function $\mathsf{ModA3}*_{(1,n,n^2)}$ and inner function $\mathscr{D}_2$ as defined in Section 4.7.2. Our final function is $F_{4.5} : \Gamma^{n^3} \to \{0,1\}$ such that $F_{4.5} = \mathsf{ModA3}*_{(1,n,n^2)} \circ \mathscr{D}_2$ and $\mathscr{D}_2$ is a function from Γ to $\bar{\Sigma}$ where $\Gamma = 240 \log n$.

From the proof of Theorem 4.53 it directly follows that $F_{4.5}$ is transitive under the action of the group G_2.

Since $F_{4.5}$ is transitive, considering the fact that $\mathsf{ModA3}*_{(1,n,n^2)}$ is harder than $\mathsf{A3}_{(1,n,n^2)}$ and combining [ABB$^+$17] result with Theorem 2.20 we have Theorem 4.5.

Chapter 5

Separation results for Transitive functions using other functions

One would naturally ask what stops us from constructing transitive functions analogous to the other functions, like cheat sheet-based functions. In fact, one could ask why to use ad-hoc techniques to construct transitive functions (as we have done in most of our proofs) and instead why not design a unifying technique for converting any function into a transitive function that would display similar properties in terms of combinatorial measures [1]. If one could do so, all the 'separation' results for general functions (in terms of separation between pairs of measures) would translate to separation for transitive functions. In Section 5.2 we have discussed why such a task is challenging. We argue the challenges of making transitive versions of the cheat-sheet functions.

Our results:

Function constructed in [ABK16] demonstrates a quadratic separation between quantum query complexity and certificate complexity. Their function was not transitive. We modify their function to obtain a transitive function that gives a similar separation.

Theorem 5.1 (Restatement of Theorem 1.5)**.** *There exists a transitive function $F_{1.5} : \{0,1\}^N \to \{0,1\}$ such that*

$$\mathsf{Q}(F_{1.5}) = \widetilde{\Omega}(\mathsf{C}(F_{1.5})^2).$$

[1] In [BCG$^+$20] they have demonstrated a technique that can be used for constructing a transitive partial function that demonstrates gaps (between certain combinatorial measures) similar to a given partial function that need not be transitive. But their construction need not construct a total function even when the given function is total.

It was proved in [BHT17] that the functions witness a gap between deterministic query complexity (or randomized query complexity) and $\mathsf{UC}_{\mathsf{min}}$ can be transformed to give functions that witness separation between deterministic query complexity (or randomized query complexity) and sensitivity. We observe that transformation the [BHT17] described preserves transitivity. Our transitive functions from Theorem 4.3 along with the transformation from [BHT17] gives the following theorem.

Theorem 5.2. *There exists transitive functions $F_{5.2}$ such that* $\mathsf{R}(F_{5.2}) = \widetilde{\Omega}(\mathsf{s}(F_{5.2})^3)$.

We also discuss the difficulties related to the important classes of functions called cheat sheets introduced in [ABK16]. All our results are compiled (and marked in green) in Table 5.2 which also serves as a survey of the study of Transitive Boolean functions.

5.1 Other transitive functions and related separation

5.1.1 Separation between quantum query complexity and certificate complexity

We start this section by recalling the definition of $\mathsf{k\text{-}sum}$. For $\Sigma = [n^k]$ the function $\mathsf{k\text{-}sum}$: $\Sigma^n \to \{0,1\}$ is defined as follows: on input $x_1, x_2, \dots, x_n \in \Sigma$, if there exists k element $x_{i_1}, \dots, x_{i_k}$, $i_1, \dots, i_k \in [n]$, that sums to 0 (mod $|\Sigma|$) then output 1, otherwise output 0.

First, we will describe an encoding scheme for the inputs to $\mathsf{k\text{-}sum}$ function such that the resulting function $\mathsf{ENC\text{-}k\text{-}Sum}$ is transitive. We then, similar to [ABK16], define $\mathsf{ENCBlock\text{-}k\text{-}Sum}$ function. Composing $\mathsf{ENC\text{-}k\text{-}Sum}$ with $\mathsf{ENCBlock\text{-}k\text{-}Sum}$ as outer function gives us $F_{1.5}$.

Function definition

Recall that, from Definition 2.34, for $\Sigma = [n^k]$, the function $\mathsf{k\text{-}sum} : \Sigma^n \to \{0,1\}$ is defined as follows: on input $x_1, x_2, \dots, x_n \in \Sigma$, if there exists k element $x_{i_1}, \dots, x_{i_k}$, $i_1, \dots, i_k \in [n]$, that sums to 0 (mod $|\Sigma|$) then output 1, otherwise output 0. We first define an encoding scheme for Σ.

Encoding scheme

Similar to Section 4.5.1 we first define the standard form of the encoding of $x \in \Sigma$ and then extend it by action of suitable group action to define all encodings that represent $x \in \Sigma$ where Σ is of size n^k for some $k \in \mathbb{N}$.

Fix some $x \in \Sigma$ and let $x = x_1 x_2 \ldots x_{k\log n}$ be the binary representation of x. The standard form of encoding of x is defined as follows: For all $i \in [k\log n]$ we encode x_i with with $4(k\log n + 2)$ bit Boolean string satisfying the following three conditions:

1. $x_i = x_{i1}x_{i2}x_{i3}x_{i4}$ where each x_{ij}, for $j \in [4]$, is a $(k\log n + 2)$ bit string,
2. if $x_i = 1$ then $|x_{i1}| = 1, |x_{i2}| = 0, |x_{i3}| = 2, |x_{i4}| = i + 2$, and
3. if $x_i = 0$ then $|x_{i1}| = 0, |x_{i2}| = 1, |x_{i3}| = 2, |x_{i4}| = i + 2$.

Having defined the standard form, other valid encodings of $x_i = (x_{i1}x_{i2}x_{i3}x_{i4})$ are obtained by the action of permutations $(12)(34), (13)(24) \in \mathsf{S}_4$ on the indices $\{i1, i2, i3, i4\}$. Finally if $x = \{(x_{ij} | i \in [k\log n], j \in [4]\}$, then $\{(x_{\sigma(i)\gamma(j)}) | \sigma \in \mathsf{S}_{k\log n}, \gamma \in T \subset \mathsf{S}_4\}$ is the set of all valid encoding for $x \in \Sigma$. The decoding scheme follows directly from the encoding scheme.

Given $y \in \{0,1\}^{k\log n(4k\log n + 8)}$, first break y into $k\log n$ blocks each of size $4k\log n + 8$ bits. If each block is a valid encoding then output the decoded string else output that y is not a valid encoding for any element from Σ.

Definition of the encoded function

ENC-k-Sum is a Boolean function that defined on n-bit as follows: Split the n-bit input into block of size $4k\log n(k\log n + 2)$. We say such block is a valid block iff it follows the encoding scheme in Section 5.1.1 i.e. represents a number from the alphabet Σ. The output value of the function is 1 iff there exists k valid block such that the number represented by the block in Σ sums to 0 (mod $|\Sigma|$).

ENCBlock-k-Sum is a special case of the ENC-k-Sum function. We define it next. ENCBlock-k-Sum is a Boolean function that defined on n-bit as follows: The input string is splits into block of size $4k\log n(k\log n + 2)$ and we say such block is a valid block iff it follows the encoding scheme of Section 5.1.1 i.e. represents a number from the alphabet Σ. The output value of the function is 1 iff there exists k valid block such that the number represented by the block in Σ sums to 0(mod $|\Sigma|$) and the number of 1 in the other block is at least $6 \times (k\log n)$. Finally, similar to [ABK16] define $F_{1.5} : \{0,1\}^{n^2} \to \{0,1\}$ to be ENCBlock-k-Sum $\circ$ ENC-k-Sum, with $k = \log n$.

The proof of Theorem 1.5 is same as that of [ABK16]. We give the proof here for completeness.

Proof of Theorem 1.5. We first show that the certificate complexity of $F_{1.5}$ is $\mathsf{O}(4nk^2\log n(k\log n + 2)$. For this we show that every input to ENCBlock-k-Sum, the outer function of $F_{1.5}$, has a certificate with $\widetilde{O}(k \times (4k\log n(k\log n+2)))$ many 0's and $O(n)$ many 1's. Also the inner function of $F_{1.5}$, i.e. $\mathsf{ENC\text{-}k\text{-}Sum}$, has 1-certificate of size $\mathsf{O}(4k^2\log n(k\log n+2)$ and 0-certificate of size $\mathsf{O}(n)$. Hence, the $F_{1.5}$ function has certificate of size $\mathsf{O}(4nk^2\log n(k\log n+2))$.

Every 1-input of $\mathsf{ENCBlock\text{-}k\text{-}Sum}$ has k valid encoded blocks such that the number represented by them sums to (0 mod $|\Sigma|$). This can be certified using at most $\widetilde{\mathsf{O}}(k \times (4k\log n(k\log n+2)))$ number of 0's and all the 1's from every other block.

There are two types of 0-inputs of $\mathsf{ENCBlock\text{-}k\text{-}Sum}$. First type of 0-input has at least one block in which number of 1 is less than $6 \times (k\log n)$ and the zeros of that block is a 0-certificate of size $\widetilde{O}(4k\log n(k\log n+2))$. The other type of 0-input is such that every block contains at least $6 \times (k\log n)$ number of 1's. This type of 0-input can be certified by providing all the 1's in every block, which is at most $\mathsf{O}(n)$. This is because using all the 1's we can certify that even if the blocks were valid, no k-blocks of them is such that the number represented by them sums to 0 (mod $|\Sigma|$).

Next, we prove $\Omega(n^2)$ lower bound on quantum query complexity of $F_{1.5}$. From Theorem 2.21, $\mathsf{Q}(F_{1.5}) = \Omega(\mathsf{Q}(\mathsf{ENCBlock\text{-}k\text{-}Sum})\mathsf{Q}(\mathsf{ENC\text{-}k\text{-}Sum}))$. Since $\mathsf{ENC\text{-}k\text{-}Sum}$ reduces to $\mathsf{ENC-Block\text{-}k\text{-}Sum}$, from Theorem 2.35 the quantum query complexity of the $\mathsf{ENC-Block\text{-}k\text{-}Sum}$ function is $\Omega\left(\frac{n^{\frac{k}{k+1}}}{k^{\frac{3}{2}}\log n(k\log n+2)}\right)$. Thus

$$\mathsf{Q}(F_{1.5}) = \Omega\left(\frac{n^{\frac{2k}{k+1}}}{k^3\log n(k\log n+2)}\right).$$

Hence, $\mathsf{Q}(F_{1.5}) = \widetilde{\Omega}(n^2)$, taking $k = \log n$. □

5.1.2 Separation between sensitivity and randomized query complexity

We start by defining *desensitisation transform* of Boolean functions as defined in [BHT17].

Definition 5.3 (Desensitized Transformation). *Let $f : \{0,1\}^n \to \{0,1\}$. Let U be a collection of unambiguous 1-certificates for f, each of size at most $\mathsf{UC}_1(f)$. For each $x \in f^{-1}(1)$, let $p_x \in U$ be the unique certificate in U consistent with x. The desensitized version of f is the function $f_{\mathsf{DT}} : \{0,1\}^{3n} \to \{0,1\}$ defined by $f_{\mathsf{DT}}(x_1x_2x_3) = 1$ if and only if $f(x_1) = f(x_2) = f(x_3) = 1$ and $p_{x_1} = p_{x_2} = p_{x_3}$.*

Observation 5.4. *If $f : \{0,1\}^n \to \{0,1\}$ is transitive, then $f_{\mathsf{DT}} : \{0,1\}^{3n} \to \{0,1\}$ is also transitive.*

Proof. Let $T_f \subseteq \mathsf{S}_n$ be the transitive group corresponding to f and let $x_1x_2x_3 \in \{0,1\}^{3n}$ be the input to f_{DT}. Consider the following permutations acting on the input $x_1x_2x_3$ to f_{DT}:

1. S_3 acting on the indices $\{1,2,3\}$ and
2. $\{(\sigma,\sigma,\sigma) \in \mathsf{S}_{3n} | \sigma \in T_f\}$ acting on (x_1,x_2,x_3).

Observe that the above permutations act transitively on the inputs to f_{DT}. Also from the definition of f_{DT} the value of the function f_{DT} is invariant under these permutations. □

Next, we need the following theorem from [BHT17]. The theorem is true for more general complexity measures. We refer the reader to [BHT17] for a more general statement.

Theorem 5.5 ([BHT17])**.** *For any $k \in \mathbb{R}^+$, if there is a family of function with $\mathsf{D}(f) = \widetilde{\Omega}(\mathsf{UC}_{min}(f)^{1+k})$, then for the family of functions defined by $\widetilde{f} = \mathsf{OR}_{3\mathsf{UC}_{\min}(f)} \circ f_{\mathsf{DT}}$ satisfies $\mathsf{D}(\widetilde{f}) = \widetilde{\Omega}(s(\widetilde{f})^{2+k})$. Also, if we replace $\mathsf{D}(f)$ by $\mathsf{R}(f)$, $\mathsf{Q}(f)$ or $\mathsf{C}(f)$, we will get the same result.*

Proof of Theorem 5.2. Let us begin with the transitive functions $F_{4.3}$ from Section 4.7.1 which will desensitize to get the desired claim. From Claim 4.42 and Observation 4.32 we have $\mathsf{R}(F_{4.3}) \geq \widetilde{\Omega}(\mathsf{UC}_{min}(F_{4.3})^2)$.

Let $F_{5.2}$ be the desensitize $F_{4.3}$. From Theorem 5.5, Observation 5.4 we have the theorem. □

5.2 Challenges in transitive versions of “cheat sheet” based functions

In this section we show that it is not possible to give a quadratic separation between degree and quantum query complexity for transitive functions by modifying the cheat sheet function using the techniques in [ABK16] which go via unambiguous certificate complexity.

Let us start by recalling the cheat sheet framework from [ABK16]. Let $f : \{0,1\}^n \to \{0,1\}$ be a total Boolean function. Let $C(f)$ be its certificate complexity and $\mathsf{Q}(f)$ be its bounded-error quantum query complexity. We consider the following cheat sheet function, which we denote by $f_{CS,t} : \{0,1\}^{n\times\log t + t\times\log t\times C(f)\times\log n} \to \{0,1\}$:

- There are $\log t$ copies of f on disjoint sets on inputs denoted by $f_1,\dots,f_{\log t}$.
- There are t cheat sheets: each cheat sheet is a block of $(\log t \times C(f) \times \log n)$ many bits

- Let $x_1, \ldots, x_{\log t} \in \{0,1\}^n$ denote the input to the $\log t$ copies of f and let $Y_1, \ldots, Y_t$ denote the t cheat sheets.
- Let $\ell = (f(x_1), \ldots, f(x_{\log t}))$. $f_{CS,t}$ evaluates to 1 if and only if Y_ℓ is a valid cheat sheet.

Separations between various complexity measures was shown in [ABK16] using the cheat sheet framework. In [ABK16], the separations that lower bound bounded-error quantum query complexity in terms of other complexity measures, for example degree, are obtained as follows:

1. Start with a total function $f : \{0,1\}^n \to \{0,1\}$ that has quadratic separation between quantum query complexity and certificate complexity: $\mathrm{Q}(f) = \widetilde{\Omega}(n)$ and $C(f) = \widetilde{O}(\sqrt{n})$. Consider the cheat sheet version of this function $f_{CS,t}$, with $t = n^{10}$.
2. Lower bound $Q(f_{CS,t})$, for $t = n^{10}$, by $\mathrm{Q}(f)$. This uses the hybrid method ([BBBV97]) and strong direct product theorem ([LR13]).
3. Upper bound degree of $f_{CS,t}$ by using the upper bound on the unambiguous certificate complexity of $f_{CS,t}$.

Instead of degree, one might use approximate degree in the third step above for a suitable choice of f (see [ABK16] for details).

A natural approach to obtain a transitive function with gap between a pair of complexity measures is to modify the cheat sheet framework to make it transitive. One possible modification is to allow a poly-logarithmic blowup in the input size of the resulting transitive function while preserving complexity measures of the cheat sheet function that are of interest (upto poly-logarithmic factors). We show, however, that it is not possible to obtain a quadratic separation between degree and quantum query complexity for transitive functions by modifying the cheat sheet function using the techniques in [ABK16] which go via unambiguous certificate complexity. The reason for this is that the unambiguous certificate complexity of a transitive cheat sheet function on N-bits is $\Omega(\sqrt{N})$ (see Observation 5.6) whereas we show (see Lemma 5.7) that the quantum query complexity of such a function is $o(N)$.

Note that this does not mean that cheat sheet framework can not be made transitive to show such a quadratic gap. If the cheat sheet version of a function that is being made transitive has a better degree upper bound than that given by unambiguous certificate complexity then a better gap might be possible.

To formalize the above discussion we first need the following observation that lower bounds the certificate complexity of any transitive function.

Observation 5.6 ([SYZ04]). *Let $f : \{0,1\}^N \to \{0,1\}$ be a transitive function, then $\mathsf{C}(f) \geq \sqrt{N}$.*

Next, we upper bound on quantum query complexity of cheat sheet function using quantum amplitude amplification ([BHMT02]). The details of proof of the following lemma can be found in the full version of this paper [CKP21].

Lemma 5.7. *The quantum query complexity of $f_{CS,t}$ is $O(\sqrt{t} \times \log t \times \sqrt{n} \times \log n)$.*

The cheat sheet version of f, $f_{CS,t}$, is a function on $\widetilde{\Theta}(n + C(f)t)$ many variables, where t is polynomial in n. From the cheat sheet property the unambiguous certificate complexity of $f_{CS,t}$, denoted by $\mathsf{UC}(f_{CS,t})$, is $\widetilde{\Theta}(\mathsf{C}(f))$.

Let $\widetilde{f_{CS,t}}$ be a modified transitive version of $f_{CS,t}$ that preserves the quantum query complexity and certificate complexity of $f_{CS,t}$ upto poly-logarithmic factors, respectively. From Observation 5.6 it follows that $\mathsf{UC}(\widetilde{f_{CS,t}}) = \widetilde{\Omega}(\sqrt{n + \mathsf{C}(f)t})$. On the other hand, since $\widetilde{f_{CS,t}}$ preserves the certificate complexity upto poly-logarithmic factors, $\mathsf{UC}(\widetilde{f_{CS,t}}) = \widetilde{O}(\mathsf{C}(f))$. This implies that $t = \widetilde{O}(\mathsf{C}(f))$. Lemma 5.7 that $\mathsf{Q}(f_{CS,t})$ is at most $\widetilde{O}(C(f)\sqrt{t})$. Thus in order to achieve quadratic separation between UC and Q, t has to be $\widetilde{\Omega}(\mathsf{C}(f)^2)$.

We end this section by giving a concrete approach towards showing separation between degree and quantum query complexity for a transitive functions using the cheat sheet method. We believe the it is possible to start with $f_{CS,t}$, for transitive function f and $t = \sqrt{n}$ and convert it to a transitive function that preserves the unambiguous certificate complexity and quantum query complexity upto poly-logarithmic factors, while incurring a poly-logarithmic blowup in the input size. However, we do not know how to prove quantum query complexity lower bound matching our upper bound from Lemma 5.7 for $t = \sqrt{n}$. We make the following conjecture towards this end.

Conjecture 5.8. *There exists a transitive function $f : \{0,1\}^n \to \{0,1\}$ with $\mathsf{C}(f) = \widetilde{O}(\sqrt{n})$ and $\mathsf{Q}(f) = \widetilde{\Omega}(n)$. Let $f_{CS,\sqrt{n}}$ be the cheat sheet version of f with $\sqrt{n}$ cheat sheets. Then $\mathsf{Q}(f_{CS,\sqrt{n}}) = \Omega(n^{3/4})$.*

If true, the above conjecture should implies that for a transitive function f, $\mathsf{Q}(f) = \widetilde{\Omega}(\deg(f)^{4/3})$.

It was showed in [ABK16] that the quantum query complexity of the cheat function $f_{CS,t}$, i.e. $\mathsf{Q}(f_{CS,t})$, is lower bounded by $\mathsf{Q}(f)$, when $t = n^{10}$. Their proof goes via he hybrid method ([BBBV97]) and strong direct product theorem ([LR13]). Is is interesting to find the the constant smallest c such that $\mathsf{Q}(f_{CS,n^c}) = \Omega(\mathsf{Q}(f))$. We know that such a c must be at least than 1 (from Lemma 5.7) and is at most 10 (from [ABK16]). We state this formally below:

Question 5.9. *Let $f : \{0,1\}^n \to \{0,1\}$ be a non-constant Boolean function and let f_{CS,n^c} be its cheat sheet version with n^c cheat sheets. What is the smallest c such that the following is true $\mathsf{Q}(f_{CS,n^c}) = \Omega(\mathsf{Q}(f))$.*

5.3 State of art of the study of Transitive Boolean functions

Transitive functions have been studied for many years and there are results for some sub-classes of transitive functions which are strictly larger than the classes of Symmetric functions. Here we mention some of the important results related to transitive function as well as the state of the art of this area.

5.3.1 Known lower bounds for complexity measures for the class of transitive function

Corollary 5.10 ([KT16])**.** $\mathsf{fbs}(f) = \Omega(\sqrt{n})$ *for any transitive function f.*

From [BHT17] it is also known that,

Lemma 5.11. *For any Boolean function f,* $\mathsf{fbs}(f) = 2\mathsf{UC}_{\mathsf{min}}(f) - 1$.

Which implies the following corollary,

Corollary 5.12. $\mathsf{UC}_{min}(f) = \Omega(\sqrt{n})$ *and* $\mathsf{R}(f) = \Omega(\sqrt{n})$ *for any transitive function $f : \{0,1\}^n \to \{0,1\}$.*

From above it follows that $\mathsf{D}(f) \leq \mathsf{R}(f)^2$.

The following table represents the individual known separations and the known example for different complexity measures for the class of transitive function:

5.3.2 Remaining cells of Table 5.2

In this section we prove that all function in white and yellow cell of Table 5.2 are transitive. Since PARITY ($\oplus$) and AND ($\wedge$) are symmetric they are also transitive by definition. The other functions that appear in the table are those in Definition 2.30 (NW^d for finite $d \in \mathbb{N}$), Definition 2.28 (RUB), Definition 2.32 (GSS_1^d) and Definition 2.33 (GSS_2).

We have the following corollary to the above observation:

Measure	**known lower bounds**	**Known example**
D	$\Omega(\sqrt{N})$ [SYZ04]	$O(\sqrt{N})$ [SYZ04]
$\mathsf{R_0}$	$\Omega(\sqrt{N})$ [SYZ04]	$O(\sqrt{N})$ [SYZ04]
R	$\Omega(N^{\frac{1}{3}})$ $\mathsf{bs(f)} = O(\mathsf{R}(f))$	$O(\sqrt{N})$ [SYZ04]
C	$\Omega(\sqrt{N})$	$O(\sqrt{N})$ $Tribe(\sqrt{N},\sqrt{N})$
RC	$\Omega(\sqrt{N})$ $\mathsf{fbs(f)} = \Theta(\mathsf{RC}(f))$	$O(\sqrt{N})$ $Tribe(\sqrt{N},\sqrt{N})$
bs	$\Omega(N^{\frac{1}{3}})$ [Sun07]	$\widetilde{O}(N^{\frac{3}{7}})$ [Sun07], [Dru11]
s	$\Omega(N^{\frac{1}{8}})$ $\deg(f) = O(\mathsf{s(f)})^2$	$\Theta(N^{\frac{1}{3}})$ [Cha11]
λ	$\Omega(N^{\frac{1}{12}})$ $\mathsf{C(f)} = O(\lambda(\mathsf{f}))^6$	$\Theta(N^{\frac{1}{3}})$ [Cha11]
$\mathsf{Q_E}$	$\Omega(N^{\frac{1}{4}})$ $\mathsf{Q}(f) = O(\mathsf{Q_E}(f))$	$O(\sqrt{N})$ [SYZ04]
deg	$\Omega(N^{\frac{1}{4}})$ $\deg(f) = \Omega(\widetilde{\deg}(f))$	$O(\sqrt{N})$ [SYZ04]
Q	$\Omega(N^{\frac{1}{4}})$ [SYZ04]	$\widetilde{O}(N^{\frac{1}{4}})$ [SYZ04]
$\widetilde{\deg}$	$\Omega(N^{\frac{1}{4}})$ [KT16]	$\widetilde{O}(N^{\frac{1}{4}})$ [SYZ04]

Table 5.1 Upper and Lower bounds on Complexity measures for Transitive Boolean functions

In each row, for the measure A, the two entries a, b represents:
(1) (Known lower bound) for all transitive Boolean function f, $A(f) = \Omega(a)$, and
(2) (Known example) there exists a *transitive* function g such that $A(g) = O(b)$, where a and b are some polynomial in N.

Observation 5.13. *The following functions are transitive:*

1. $\mathsf{OR}_n \circ \mathsf{AND}_n$
2. *$\widetilde{\wedge}$-tree for depth d for finite $d \in \mathbb{N}$,*

Table 5.2 Best known separations between combinatorial measures for transitive functions.

	D	R_0	R	C	RC	bs	s	λ	Q_E	deg	Q	$\widetilde{\deg}$
D		2 ; 2 T:1.4	2 ; 2 T:1.4	2 ; 2 $\wedge \circ \vee$	2 ; 2 $\wedge \circ \vee$	2 ; 3 $\wedge \circ \vee$	3 ; 6 T:5.2	4 ; 6 T:4.3	2 ; 3 T:4.2	2 ; 3 T:1.4	4 ; 4 T:1.4	4 ; 4 T:4.3
R_0	1,1 $\oplus$		2 ; 2 T:4.2	2 ; 2 $\wedge \circ \vee$	2 ; 2 $\wedge \circ \vee$	2 ; 3 $\wedge \circ \vee$	3 ; 6 T:5.2	4 ; 6 T:4.3	2 ; 3 T:4.2	2 ; 3 T:4.2	3 ; 4 T:4.4	4 ; 4 T:4.3
R	1 ; 1 $\oplus$	1 ; 1 $\oplus$		2 ; 2 $\wedge \circ \vee$	2 ; 2 $\wedge \circ \vee$	2 ; 3 $\wedge \circ \vee$	3 ; 6 T:5.2	4 ; 6 T:4.3	1.5 ; 3 T:4.5	2 ; 3 T:4.3	2 ; 4 $\wedge$	4 ; 4 T:4.3
C	1 ; 1 $\oplus$	1 ; 1 $\oplus$	1 ; 2 $\oplus$		2 ; 2 [GSS16]	2 ; 2 [GSS16]	2 ; 5 [Rub95]	2 ; 6 $\wedge$	1.15 ; 3 [Amb16]	1.63 ; 3 [NW95]	2 ; 4 $\wedge$	2 ; 4 $\wedge$
RC	1 ; 1 $\oplus$	1 ; 1 $\oplus$	1 ; 1 $\oplus$	1 ; 1 $\oplus$		1.5 ; 2 [GSS16]	2 ; 4 [Rub95]	2 ; 4 $\wedge$	1.15 ; 2 [Amb16]	1.63 ; 2 [NW95]	2 ; 2 $\wedge$	2 ; 2 $\wedge$
bs	1 ; 1 $\oplus$	1 ; 1 $\oplus$	1 ; 1 $\oplus$	1 ; 1 $\oplus$	1 ; 1 $\oplus$		2 ; 4 [Rub95, AS11]	2 ; 4 $\wedge$	1.15 ; 2 [Amb16]	1.63 ; 2 [NW95]	2, 2 $\wedge$	2 ; 2 $\wedge$
s	1 ; 1 $\oplus$	1 ; 1 $\oplus$	1 ; 1 $\oplus$	1 ; 1 $\oplus$	1 ; 1 $\oplus$	1 ; 1 $\oplus$		2 ; 2 $\wedge$	1.15 ; 2 [Amb16]	1.63 ; 2 [NW95]	2, 2 $\wedge$	2 ; 2 $\wedge$
λ	1 ; 1 $\oplus$	1 ; 1 $\oplus$	1 ; 1 $\oplus$	1 ; 1 $\oplus$	1 ; 1 $\oplus$	1 ; 1 $\oplus$	1 ; 1 $\oplus$		1 ; 1 $\oplus$	1 ; 1 $\oplus$	1 ; 1 $\oplus$	1 ; 1 $\oplus$
Q_E	1 ; 1 $\oplus$	1.33 ; 2 $\tilde{\wedge}$-tree	1.33 ; 3 $\tilde{\wedge}$-tree	2 ; 2 $\wedge \circ \vee$	2 ; 2 $\wedge \circ \vee$	2 ; 3 $\wedge \circ \vee$	2 ; 6 T:1.5	2 ; 6 T:1.5		1 ; 3 $\oplus$	2 ; 4 $\wedge$	1 ; 4 $\oplus$
deg	1 ; 1 $\oplus$	1.33 ; 2 $\tilde{\wedge}$-tree	1.33 ; 2 $\tilde{\wedge}$-tree	2 ; 2 $\wedge \circ \vee$	2 ; 2 $\wedge \circ \vee$	2 ; 2 $\wedge \circ \vee$	2 ; 2 $\wedge \circ \vee$	2 ; 2 $\wedge$	1 ; 1 $\oplus$		2 ; 2 $\wedge$	2 ; 2 $\wedge$
Q	1 ; 1 $\oplus$	1 ; 1 $\oplus$	1 ; 1 $\oplus$	2 ; 2 T:1.5	2 ; 2 T:1.5	2 ; 3 T:1.5	2 ; 6 T:1.5	2 ; 6 T:1.5	1, 1 $\oplus$	1 ; 3 $\oplus$		1 ; 4 $\oplus$
$\widetilde{\deg}$	1 ; 1 $\oplus$	1 ; 1 $\oplus$	1 ; 1 $\oplus$	1 ; 2 $\oplus$	1 ; 2 $\oplus$	1 ; 2 $\oplus$	1 ; 2 $\oplus$	1 ; 2 $\oplus$	1 ; 1 $\oplus$	1 ; 1 $\oplus$	1 ; 1 $\oplus$	

(1) Entry $a;b$ in row A and column B represents: (1) ('Relationships') for any Boolean function f , $A(f) = O(B(f))^{b+o(1)}$, and (2) ('Separations') there exists a *transitive* function g such that $A(g) = \Omega(B(g))^a$.

(2) Cells with a green background are those for which we constructed new *transitive functions* to demonstrate 'separations' that match the known 'separations' for general functions. The previously known functions that gave the strongest 'separations' were not transitive. The second row gives the reference to the Theorems where the 'separation' is proved. Although for these green cells the bounds match that of the general functions, for some cells (with light green color) there is a gap between the known 'relationships' and 'separations'.

(3) In the cells a with white background the best-known examples for the corresponding separation was already *transitive functions*. For these cells, the second row either contains the function that demonstrates the separation or a reference to the paper where the separation was proved. So for these cells, the 'separations' for transitive functions matched the current best known 'separations' for general functions. Note that for some of these cell the don't match.

(4) Cells with a yellow background are those where the 'separations' do not match the best known 'separations' for general functions.

(5) The various functions mentioned/referred in this table are defined in Section 2.2.

3. NW^d,
4. RUB,
5. GSS_1^d, *and*
6. GSS_2.

Proof. The transitivity of the functions either follows directly from their definitions or from the fact that they are composition of two transitive functions and hence the function is transitive using Observation 4.10. □

5.4 Conclusion

This chapter presents a thorough investigation of the separations between various pairs of complexity measures for transitive function. The main technical contribution of this paper is to define transitive versions of pointer functions of [ABK16] that have similar complexity measures as that of the original pointer functions while incurring only a poly-logarithmic blowup in the input size. The current best known 'relationships' and 'separations' between various pairs of measures for transitive functions are summarized in the Table 5.2.

Unfortunately, a number of cells in the table are not tight. In this context, we would like to point out two important directions:

- For some of these cells, the 'separation' results for transitive functions are weaker than that of the general functions. A natural question is the following: why can't we design a transitive version of the general functions that achieve the same separation? For some cases, like the cheat sheet-based functions, we discuss the difficulties and possible directions in Chapter 5 Section 5.2. For these cases, the natural question would be to obtain some different collection of functions (maybe not transitive) that achieves similar separations.
- In some of the cells in the Table 5.2 tight bounds are not known, even for the case of general functions. Can the 'relationships' results in these cases possibly be improved for the case of transitive functions?

In the Table 5.1 we summarize the results on how low can individual complexity measures go for transitive function. Even with the recent results of Huang [Hua19] and Aaronson et al. [ABK$^+$21b], there are significant gaps between the 'relationships' and 'separations' in this case.

Finally, we would like to ask the question of how the amount of symmetry affects the relationship between various measures. This is in the lines of the recent work [BCG$^+$20]. The study about the different types of symmetries like graph properties, cyclically invariant functions, min-term transitive functions, etc. is not new in this area, but a more elaborate analysis is required to quantify the relationship between the measures and the amount of symmetry.

Chapter 6

Non-deterministic degree of Symmetric Boolean functions

6.1 Introduction

The non-deterministic degree $ndeg(f)$ is the minimal degree of a non-deterministic polynomial for f , which is required to be nonzero iff $f(x) = 1$. In the area of quantum algorithms, non-deterministic degree is a useful tool to characterize the non-deterministic version of a quantum algorithm. Note that classical degree is equivalent to Quantum exact query algorithm while non-deterministic degree is nothing but a way to characterize the nondeterministic version of a quantum algorithm. For further details, we refer to [Mid04, dW00].

Now we will define another classical problem from discrete geometry known as covering points from hypercube. Given a natural number $n \in \mathbb{N}$, $[n]$ denotes the set $\{1,\ldots,n\}$. For any non-zero vector $a \in \mathbb{R}^n$ and b in $\mathbb{R}$, the set of solutions to the affine equation $H(x) := \langle a,x\rangle - b = 0$[1] defines a hyperplane in $\mathbb{R}^n$. We say a point u in $\mathbb{R}^n$ is covered by a hyperplane H if u *lies* on the hyperplane H, that is if $H(u) = 0$. A family of hyperplanes in $\mathbb{R}^n$ is said to cover a set $S \subset \mathbb{R}^n$ if for every $u \in S$ there exists a hyperplane in the family that covers u. Similarly, we say u is covered by a polynomial $P \in \mathbb{R}[x_1,\ldots,x_n]$ if $P(u) = 0$. Essentially the polynomial we will get from the covering problem is the non-deterministic polynomial for a Boolean function. Here in this chapter, we will investigate the polynomial covering for symmetric subsets of the Boolean cube which will lead us to the non-deterministic degree of a polynomial that represents a symmetric Boolean function in the non-deterministic set-up.

[1]For all a, b in $\mathbb{R}^n$, $\langle a,b\rangle$ will denote the standard *inner product* between a and b.

Using Combinatorial Nullstellensatz [AT92, Alo99], Alon and Füredi [AF93] proved the following lower bound on the natural except the origin $(0,\dots,0)$.

Theorem 6.1 (Alon and Füredi [AF93]). *Let P be a polynomial in $\mathbb{R}[x_1,\dots,x_n]$ such that P covers every point of $\mathcal{Q}^n$ expect the origin $(0,\dots,0)$. Then $deg(P) \geq n$.*

Now suppose that we are given an n-cube $\mathcal{Q}^n = \{0,1\}^n$, and we want to cover all its vertices using minimum number of hyperplanes. Observe that, using only two hyperplanes (namely $x_k = 0$ and $x_k - 1 = 0$ for any $k \in [n]$), one can cover all the vertices of the n-cube. Also, note that at least two hyperplanes are required to cover all the vertices of $\mathcal{Q}^n$.

Bárány asked about the minimum number m such that there exists a family of m hyperplanes in $\mathbb{R}^n$ covering every point of the hypercube $\mathcal{Q}^n$ except the origin $(0,\dots,0)$. Bárány extracted this problem from a paper on an infinite extension of Rado's Theorem by Komjáth [Kom94]. Komjáth [Kom94] proved that $m \geq \log_2 n - \log_2 \log_2 n$ for all $n \geq 2$. Also, note that $m \leq n$, because the hyperplanes $H_1,\dots,H_n$, where $H_i(x) := x_i - 1 = 0$ for all $i \in [n]$, cover every point of $\mathcal{Q}^n$ except $(0,\dots,0)$. Alon and Füredi proved Theorem 6.1 in the context of solving this covering problem. As a direct consequence of Theorem 6.1, we get the following celebrated result in combinatorial geometry.

Theorem 6.2 (Alon and Füredi [AF93]). *Let m be the least positive integer such that there exists a family of m hyperplanes covering the n-cube $\mathcal{Q}^n = \{0,1\}^n$ leaving out only the origin. Then $m = n$.*

Combinatorial Nullstellensatz [AT92], and Alon and Furedi's covering result [AF93] have found multiple extensions and applications in areas like finite geometry, coding theory, combinatorial geometry, and extremal combinatorics, see [Alo99, BS09, KMR11, KR12, DKSS13, BCPS18, BB20, CH20, SW20, BBDM21].

We say that a polynomial $P \in \mathbb{R}[x_1,\dots,x_n]$ has a zero of *multiplicity* at least t at a point $v \in \mathbb{R}^n$ if all derivatives of P up to order $t-1$ vanish at v and $P(v) = 0$. In this paper, we prove the following generalization of Theorem 6.1.

Theorem 6.3 (Restatement of Theorem 6.5). *Given $t \in \mathbb{N}$ and $k \in [n]$, let $P \in \mathbb{R}[x_1,\dots,x_n]$ be a polynomial such that at each point $u \in \mathcal{Q}^n \setminus \mathcal{Q}^n_k$, P has a zero of multiplicity at least t and at each point $v \in \mathcal{Q}^n_k$, P has a zero of multiplicity exactly $(t-1)$. Then*

$$\deg(P) \geq \max\{k, n-k\} + 2t - 2.$$

We say a family of hyperplanes $H_1, \ldots, H_m$ in $\mathbb{R}^n$ is said to *cover t times* a point u in $\mathbb{R}^n$ if t hyperplanes from the family cover u. Note that the following corollary is a direct consequence of the above theorem.

Corollary 6.4. *Given $t \in \mathbb{N}$ and $k \in [n]$, suppose $H_1, \ldots, H_m$ in $\mathbb{R}^n$ is a family of hyperplanes such that every point of $\mathcal{Q}^n_k$ of the hypercube is covered exactly $(t-1)$ times and every point of $\mathcal{Q}^n \setminus \mathcal{Q}^n_k$ is covered at least t times. Then*

$$m \geq \max\{k, n-k\} + 2t - 2.$$

Additionally, we give an explicit construction of a family of hyperplanes matching the lower bound of Corollary 6.4.

Theorem 6.5. *Given $t \in \mathbb{N}$ and $k \in [n]$, there exists a family of hyperplanes $H_1, \ldots, H_m$ in $\mathbb{R}^n$, where $m = \max\{k, n-k\} + 2t - 2$, such that every point of $\mathcal{Q}^n_k$ of the hypercube is covered exactly $(t-1)$ times and every point of $\mathcal{Q}^n \setminus \mathcal{Q}^n_k$ is covered at least t times.*

Observe that Theorem 6.5 directly implies that the lower bounds obtained in Theorem 1.6 and Corollary 6.4 are tight.

Corollary 6.6.

- *Given $t \in \mathbb{N}$ and $k \in [n]$, if $P \in \mathbb{R}[x_1, \ldots, x_n]$ is a polynomial of minimum degree such that at each point $u \in \mathcal{Q}^n \setminus \mathcal{Q}^n_k$, P has a zero of multiplicity at least t and at each point $v \in \mathcal{Q}^n_k$, P has a zero of multiplicity exactly $(t-1)$, then deg$(P) = \max\{k, n-k\} + 2t - 2$.*
- *Let m be the least positive integer such that there exists a family of m hyperplanes covering each point $\mathcal{Q}^n \setminus \mathcal{Q}^n_k$ at least t times and each point of $\mathcal{Q}^n_k$ exactly $t-1$ times. Then $m = \max\{k, n-k\} + 2t - 2$.*

Using ideas from the proofs of Theorems 6.5 and 1.6, we also study a new variant of *restricted sumset* problem and properties of polynomials vanishing on a grid.

6.2 Related Work

Given a subset S of $\mathcal{Q}^n$, we will denote by ec$(\{0,1\}^n \setminus S)$ the minimum number of hyperplanes required to cover all the points in $\{0,1\}^n \setminus S$ while leaving out all the points in S. A natural question to ask, after Alon-Furedi's covering result, is the following:

Problem 6.7 (Exact cover problem with respect to a subset). *For a given subset $S \subset \mathcal{Q}^n$ with $|S| > 1$, what is the value of* ec$(\{0,1\}^n \setminus S)$*?*

Aaronson et al. [AGG$^+$21] investigated the above problem first. They gave the exact value of $\mathrm{ec}(\{0,1\}^n \setminus S)$ when the forbidden set S has cardinality at most 4, and for higher cardinalities of the forbidden set S, they gave a range of values for $\mathrm{ec}(\{0,1\}^n \setminus S)$.

Theorem 6.8 (Aaronson et al. [AGG$^+$21]). *Suppose $S \subseteq \{0,1\}^n$ and $k \in [2^n]$.*

(a) If $|S| \in \{2,3\}$, then $ec(\{0,1\}^n \setminus S) = n-1$.

(b) If $|S| = 4$, then $ec(\{0,1\}^n \setminus S) = n-1$, if there is a hyperplane Q with $|Q \cap S| = 3$, and $ec(\{0,1\}^n \setminus S) = n-2$, otherwise.

(c) Let $\mathrm{ec}(n,k) := \max\{\mathrm{ec}(\{0,1\}^n \setminus S) \mid S \subseteq \{0,1\}^n$ and $|S| = k\}$. Then,

$$n - \log_2(k) \leq \mathrm{ec}(n,k) \leq n - 2^k + \mathrm{ec}(2^k, k).$$

In more recent work, Diamond and Yehudayoff [DY22] showed the existence of a subset S of the hypercube for which $\mathrm{ec}(S)$ is exponential in n.

Theorem 6.9 (Diamond and Yehudayoff [DY22]). *There exists a subset D_n of $\{0,1\}^n$ such that $\mathrm{ec}(D_n) \geq 2^{\Omega(n)}$.*

In this paper, we shall give an improved lower bound by introducing a new combinatorial measure of the forbidden set S and show that our bound is tight also.

A subset S of $\{0,1\}^n$ is called *symmetric* if it is closed under permutations of coordinates. Given a subset S of hypercube $\mathcal{Q}^n$, define

$$W(S) := \left| \left\{ \sum_{i=1}^{n} u_i \mid u = (u_1, \ldots, u_n) \in S \right\} \right|.$$

For any $i \in \{0, \ldots, n\}$, $T_{n,i}$ denotes the set

$$T_{n,i} := \{0, \ldots, i-1\} \cup \{n-i+1, \ldots, n\}.$$

Venkitesh [Ven22] proved the following variant of the covering problem:

Theorem 6.10 (Venkitesh [Ven22]). *Let S be a symmetric subset of the hypercube $\mathcal{Q}^n$, and $P \in \mathbb{R}[x_1, \ldots, x_n]$ be a polynomial of minimum degree that covers all the points in S and no points in $\mathcal{Q}^n \setminus S$. Then $deg(P) = W(S) - \max\{i \in \{0, \ldots, n\} \mid T_{n,i} \subseteq W(S)\}$.*

If S is a symmetric subset of $\mathcal{Q}^n$, then as a direct consequence of the above theorem we get

$$\mathrm{ec}(S) \geq W(S) - \max\{i \in \{0,\ldots,n\} \mid T_{n,i} \subseteq W(S)\}.$$

Also, if we substitute $S = \mathcal{Q}^n \setminus \mathcal{Q}^n_k$ in the above theorem then we get

$$deg(P) = \max\{n-k,k\},$$

which is a special case of our Corollary 6.6. Venkitesh [Ven22] made the following conjecture about covering symmetric subsets of hypercube using hyperplanes.

Conjecture 6.11 (Venkitesh [Ven22])**.** *Suppose S is a symmetric subset* $\{0,1\}^n$, $n \geq 4$ *such that* $W(S) \subseteq T_{n,2}$. *Then* $\mathrm{ec}(S) = W(S) - 2$.

Clifton and Huang introduced the notion of *multiple* covering by hyperplanes.

Theorem 6.12 (Clifton and Huang [CH20])**.** *Let* $f(n,t)$ *denote the minimum number of hyperplanes required to cover every vertex of the hypercube* $\mathcal{Q}^n$ *at least t times, while the origin* $(0,\ldots,0)$ *remains uncovered. Then*

1. $\forall n \geq 2$, $f(n,2) = n+1$
2. $\forall n \geq 2$, $f(n,3) = n+3$
3. $\forall t \geq 4$ *and* $\forall n \geq 3$, $n+t+1 \leq f(n,t) \leq n + \binom{t}{2}$

Sauermann and Wigderson first studied the following *higher multiplicities* version of the polynomial covering problem.

Theorem 6.13 (Sauermann and Wigderson [SW20])**.** *Let* $t \geq 2$, $n \geq 2t-3$ *and* $P \in \mathbb{R}[x_1,\ldots,x_n]$ *be a minimum degree polynomial having zeros of multiplicity at least t at all points in* $\{0,1\}^n \setminus \{(0,\ldots,0)\}$ *and* $P(0,\ldots,0) \neq 0$. *Then* $\deg(P) = n+2t-3$.

Sauermann and Wigderson also studied the following interesting variant of the polynomial covering problem with *higher multiplicities*.

Theorem 6.14 (Sauermann and Wigderson [SW20])**.** *Let* $t \geq 2$, $n \geq 2t-3$ *and* $P \in \mathbb{R}[x_1,\ldots,x_n]$ *be a minimum degree polynomial having zeros of multiplicity at least t at all points in* $\{0,1\}^n \setminus \{(0,\ldots,0)\}$ *and a zero of multiplicity exactly* $t-1$ *at* $(0,\ldots,0)$. *Then* $\deg(P) = n+2t-2$.

6.3 Covering Hypercube with Multiplicities

In this section, we will prove Theorems 1.6 and 6.5. For a natural number t and a subset S of $\mathscr{Q}^n$, we will say a family of hyperplanes (or a polynomial) is a $(t,t-1)$-*cover* of $\mathscr{Q}^n \setminus S$ if the family of hyperplanes (or a polynomial) cover every point of $\mathscr{Q}^n \setminus S$ at least t-times and every point of S exactly $(t-1)$-times. If $t = 1$, we will say the family is an *exact cover* of $\mathscr{Q}^n \setminus S$.

Here we shall first define a new combinatorial measure, namely *index complexity* of a subset S of the n-cube $\mathscr{Q}^n$ and then give an optimal lower bound depending upon the index complexity of the restricted set S for the following question: what is the minimum size of a $(t,t-1)$-cover of $\mathscr{Q}^n \setminus S$?

Definition 6.15 (Index complexity)**.** *We denote the index complexity of a subset S of the n cube $\mathscr{Q}^n$ by $r(S)$ and for $|S| > 1$, we define the index complexity $r(S)$ of S to be the smallest positive integer such that the following holds: $\exists I \subset [n]$ with $|I| = r(S)$ and $\exists v \in S$ such that for each $s \in S \setminus \{v\}$, $s_i \neq v_i$, for some $i \in I$. For singleton subset S, we define $r(S) = 0$.*

Observe that

- if $S = \{u \in \{0,1\}^n : u_1 = 1\}$ then $r(S) = n-1$, and
- if $S' = \{u \in \{0,1\}^n : u_1 = 1 \text{ and } \sum_{i=2}^n x_i < n-1\} \cup \{(0,\dots,0)\}$ then $r(S') = 1$.

The following result gives a lower bound of the size of $(t,t-1)$-*cover* of $\mathscr{Q}^n \setminus S$ in terms of index complexity of the set S.

Theorem 6.16. *Let $\mathscr{Q}^n = \{0,1\}^n$ and $S \subset \mathscr{Q}^n$ with size at least 2 and $P \in \mathbb{R}[x_1,\dots,x_n]$ be a polynomial, which is a $(t,t-1)$*-cover *of $\mathscr{Q}^n \setminus S$. Then*

$$\deg(P) \geq n - r(S) + 2t - 2.$$

Recall that $r(S)$ is the index complexity *of S.*

Proof. By the definition of index complexity, $\exists v = (v_1,\dots,v_n) \in S$ and $\exists I \subset [n]$ with $|I| = r(S)$ such that $\forall u = (u_1,\dots,u_n) \in S \setminus \{v\}$, $\exists i \in I$ for which $u_i \neq v_i$. We define $g \in \mathbb{R}[x_1,\dots,x_n]$ to be the polynomial

$$g(x) := \prod_{i \in I}(x_i - \bar{v}_i),$$

where $\bar{v}_i = 1 - v_i, \forall i \in I$. Then $\forall u \in S \setminus \{v\}$, $g(u) = 0$, $g(v) \neq 0$ and $\deg(g) = r(S)$.

Again, Pg is a polynomial having a zero of multiplicity at least t on every point of $\mathcal{Q}^n \setminus \{v\}$ and a zero of multiplicity exactly $(t-1)$ on v. So by Theorem 6.14 we have $\deg(Pg) \geq n+2t-2$. Hence

$$\deg(P) \geq n - r(S) + 2t - 2.$$

□

Corollary 6.17. *Let $\mathcal{Q}^n = \{0,1\}^n$ and $S \subset \mathcal{Q}^n$ with size at least 2 and $H_1, \dots, H_m$ be a family of hyperplanes, which is a $(t,t-1)$-cover of $\mathcal{Q}^n \setminus S$. Then*

$$m \geq n - r(S) + 2t - 2.$$

Proof. Observe that the polynomial $P \in \mathbb{R}[x_1, \dots, x_n]$ defined as $P(x) = \prod_{i=1}^m H_i(x)$ is a $(t,t-1)$-cover of $\mathcal{Q}^n \setminus S$ and $\deg(P) = m$. The result now directly follows from Theorem 6.16. □

Using Theorem 6.16 we will now derive a lower bound on the degree of the polynomial in terms of the size of the forbidden set.

Theorem 6.18. *Suppose $\mathcal{Q}^n = \{0,1\}^n$ and $S \subset \mathcal{Q}^n$ with size at least 2 and $P \in \mathbb{R}[x_1, \dots, x_n]$ is a polynomial, which is a $(t,t-1)$-cover of $\mathcal{Q}^n \setminus S$. Then*

$$\deg(P) \geq n - \lfloor \log_2 |S| \rfloor + 2t - 2.$$

Proof. As $|S| \geq 2$, we get at least one $i_1 \in [n]$ such that there exists $u_1, v_1 \in S$, $\langle u_1, e_{i_1} \rangle \neq \langle v_1, e_{i_1} \rangle$. Suppose $S_{(i_1,1)}$ and $S_{(i_1,0)}$ are subsets of S such that

$$S_{(i_1,1)} = \{s \in S \mid \langle s, e_{i_1} \rangle = 1\} \text{ and } S_{(i_1,0)} = \{s \in S \mid \langle s, e_{i_1} \rangle = 0\}.$$

Without loss of generality we may assume that $|S_{(i_1,1)}| \geq |S_{(i_1,0)}|$ and we denote $S_1 = S_{(i_1,0)}$. Then clearly $S_1 \subset S$ and $|S_1| \leq |S|/2$.

If $|S_1| = 1$ then we are done. Otherwise, there exist i_2 in $[n] \setminus \{i_1\}$, and u_2 and v_2 in S_1 such that $\langle u_2, e_{i_2} \rangle \neq \langle v_2, e_{i_2} \rangle$. Again, let $S_{(i_2,1)}$ and $S_{(i_2,0)}$ be subsets of S_1 such that

$$S_{(i_2,1)} = \{s \in S_1 \mid \langle s, e_{i_2} \rangle = 1\} \text{ and } S_{(i_2,0)} = \{s \in S_1 \mid \langle s, e_{i_2} \rangle = 0\}.$$

Without loss of generality let us assume that $|S_{(i_2,1)}| \geq |S_{(i_2,0)}|$ and we denote $S_2 = S_{(i_2,0)}$. Observe that $S_2 \subset S_1 \subset S$ and

$$|S_2| \leq \frac{|S_1|}{2} \leq \frac{|S|}{2^2}.$$

Again, if $|S_2| = 1$, we are done. Otherwise, we will divide S_2 and continue the above process until we get

$$S_k \subset S_{k-1} \subset \cdots \subset S_2 \subset S_1 \subset S$$

such that $\forall \ell \in [k]$, we have $|S_\ell| \leq |S_{\ell-1}|/2$ and $|S_k| = 1$. As $|S_k| \leq |S|/2^k$ and $|S_k| = 1$, we get that $k \leq \lfloor \log_2 |S| \rfloor$.

If $S_k = \{v\}$ and $I = \{i_1, \ldots, i_k\} \subset [n]$, then by the above construction we get that $\forall s \in S \setminus \{v\}$, $\exists i \in I$ such that $\langle s, e_i \rangle \neq \langle v, e_i \rangle$. Now using Theorem 6.16, we get that

$$m \geq n - k + 2t - 2 \geq n - \lfloor \log |S| \rfloor + 2t - 2.$$

$\square$

Corollary 6.19. *Let $\mathcal{Q}^n = \{0,1\}^n$ and $S \subset \mathcal{Q}^n$ with size at least 2, and $H_1, \ldots, H_m$ be a family of hyperplanes, which is a $(t, t-1)$-cover of $\mathcal{Q}^n \setminus S$. Then,*

$$m \geq n - \lfloor \log_2 |S| \rfloor + 2t - 2.$$

Remark 6.20. *Note that Aaronson et. al [AGG+21, Theorem 6.8] proved a special case ($t = 1$) of the above corollary.*

Even though Theorem 6.18, unlike Theorem 6.16, gives an explicit bound, we will now show that Theorem 6.16 is strictly stronger than Theorem 6.18. Consider the following subset of $\mathcal{Q}^n$:

$$S = \left\{ x \in \mathcal{Q}^n \mid x_1 = 1 \text{ and } \sum_{i=2}^{n} x_i < n - 1 \right\} \bigcup \left\{ (0, \ldots, 0) \right\}$$

Observe that as $|S| = 2^{n-1}$, Theorem 6.18 implies that size of a $(t, t-1)$-cover of $\mathcal{Q}^n \setminus S$ is at least $1 + 2t - 2$. We can show that at least $n - 1 + 2t - 2$ hyperplanes will be required, as index complexity $r(S) = 1$. Moreover, this lower bound is tight. Consider the following set of $n - 1$ hyperplanes:

$$H_j(x) := nx_1 + \sum_{i=2}^{n} x_i - j = 0, \quad \forall j \in [n-2]$$

$$H_{n-1}(x) := \sum_{i=2}^{n} x_i - n - 1 = 0$$

These $(n-1)$ hyperplanes form an exact cover of $\mathcal{Q}^n \setminus S$. So these $n-1$ hyperplanes along with $t-1$ copies of the hyperplane $x_1 = 0$ and $(t-1)$ copies of the hyperplane $x_1 = 1$ form a $(t,t-1)$-cover of $\mathcal{Q}^n \setminus S$.

Now a natural subset of the hypercube $\mathcal{Q}^n$ is the k-th layer $\mathcal{Q}^n_k$, that is, the set of all points in $\mathcal{Q}^n$ whose co-ordinates contain exactly k many ones. Consider the following family of hyperplanes

$$G_j(x) := \sum_{i=1}^{n} x_i - j = 0, \tag{6.1}$$

where $j \in \{0,1,2,\dots,n\}$. Observe that the n hyperplanes $G_j(x)$, where $j \in \{0,1,2,\dots,n\} \setminus \{k\}$, along with $(t-1)$ copies of the hyperplane $x_1 = 0$ and $(t-1)$ copies of the hyperplane $x_1 = 1$ form a $(t,t-1)$-cover of $\mathcal{Q}^n \setminus \mathcal{Q}^n_k$. We will show that the size of a $(t,t-1)$-cover of $\mathcal{Q}^n \setminus \mathcal{Q}^n_k$ can be much smaller. But before we do that we will first prove a lower bound, using Theorem 6.16, on the size of a $(t,t-1)$-cover of $\mathcal{Q}^n \setminus \mathcal{Q}^n_k$.

Theorem 6.21 (Restatement of Theorem 1.6)**.** *Let $P \in \mathbb{R}[x_1,\dots,x_n]$ be a polynomial, which is a $(t,t-1)$-cover of $\mathcal{Q}^n \setminus \mathcal{Q}^n_k$. Then*

$$deg(P) \geq \max\{k, n-k\} + 2t - 2.$$

Proof. We will first consider the case where $k \leq n/2$. Let v be the point in $\mathcal{Q}^n_k$ whose first k coordinates are ones and rest of them are zeros. Now observe that for each $u \in \mathcal{Q}^n_k \setminus \{v\}$ there exists $i \in [k]$ such that $1 = \langle v, e_i \rangle \neq \langle u, e_i \rangle = 0$. Therefore by definition of index complexity 6.15, we get $r(\mathcal{Q}^n_k) \leq k$ and so from Theorem 6.16, we get that $\deg(P) \geq n-k+2t-2$. The case where $k > n/2$ can be handled in a similar way. □

Corollary 6.22. *Let $H_1,\dots,H_m$ be a family of hyperplanes, which is a $(t,t-1)$-cover of $\mathcal{Q}^n \setminus \mathcal{Q}^n_k$. Then*

$$m \geq \max\{k, n-k\} + 2t - 2.$$

6.4 Conclusion

Note that we give a lower bound on the minimum degree of a polynomial that is zero at every point of the Boolean cube leaving a layer of the cube uncovered. But this is only a special kind of symmetric set. So it is natural to ask what happens to the bound in the case of general symmetric sets as well as partial symmetric sets?

Another interesting problem is trying to match the upper bound part. That is coming up with polynomials that have the mentioned properties and match the lower bound part.

Part II

Composition Theorem for the functions with symmetry

Chapter 7

Approximate degree composition in terms of Block sensitivity and some applications

7.1 Introduction

For studying the complexity of Boolean functions, a number of simple complexity measures (like decision tree complexity, randomized query complexity, degree, certificate complexity and so on) have been studied over the years. (Refer to the survey [Bd02] for an introduction to complexity measures of Boolean functions.) Understanding how these measures are related to each other [ABK16, ABK$^+$21b, ABB$^+$17, Hua19], and how they behave for various classes of Boolean functions has been a central area of research in complexity theory [Pat92, Dru11, Sun07].

A crucial step towards understanding a complexity measure is: how does the complexity measure behave when two Boolean functions are combined to obtain a new function (i.e., what is the relationship between the measure of the resulting function and the measures of the two individual functions) [BKT19, BDGKW20, GSS16, Tal13]? One particularly natural combination of functions is called *composition* (defined in the Introduction) which takes a central role in complexity theory. Let us briefly revisit the definition of composition. For any two Boolean functions $f : \{0,1\}^n \to \{0,1\}$ and $g : \{0,1\}^m \to \{0,1\}$, the composed function $f \circ g : \{0,1\}^{nm} \to \{0,1\}$ is defined as the function

$$f \circ g(x_1, \ldots, x_n) = f(g(x_1), \ldots, g(x_n)),$$

where $x_i \in \{0,1\}^m$ for $i \in [n]$. For the function $f \circ g$, the function f is called the outer function and g is called the inner function. See Definition 7.8 for a natural extension to partial functions.

Let $M(\cdot)$ be a complexity measure of Boolean functions. A central question in complexity theory is the following.

Question 7.1 (Composition question for M)**.** *Is the following true for all Boolean functions f and g:*

$$M(f \circ g) = \widetilde{\Theta}(M(f) \cdot M(g))?$$

The notation $\widetilde{\Theta}(\cdot)$ hides poly-logarithmic factors of the arity of the outer function f.

Composition of Boolean functions with respect to different complexity measures is a very important and useful tool in areas like communication complexity, circuit complexity and many more. To take an example, a popular application of composition is to create new functions demonstrating better separations (refer to [NS94, Tal13, Amb05, GSS16] for some other results of similar flavour).

It is known that the answer to the composition question is in the affirmative for complexity measures like deterministic decision tree complexity [Sav02, Tal13, Mon14], degree [Tal13] and quantum query complexity [Rei11a, LMR$^+$11, Kim13a]. While it is well understood how several complexity measures behave under composition, there are two important measures (even though well studied) for which this problems remains wide open: randomized query complexity (denoted by R) and approximate degree (denoted by $\widetilde{\deg}$) [She12, NS94, Amb05, She13a, BT13, She13b]. (See Definition 2.16 for their respective formal definitions.)

For both R and $\widetilde{\deg}$ the upper bound inequality is known, i.e., $\mathrm{R}(f \circ g) = \widetilde{O}(\mathrm{R}(f) \cdot \mathrm{R}(g))$ (folklore) and $\widetilde{\deg}(f \circ g) = O(\widetilde{\deg}(f) \cdot \widetilde{\deg}(g))$ [She13d]. Thus it is enough to prove the lower bound on the complexity of composed function in terms of the individual functions. Most of the attempts to prove this direction of the question have focused on special cases when the outer function has certain special properties[1].

For the composition of $\widetilde{\deg}$, Sherstov [She12] already showed that $\widetilde{\deg}(f \circ g)$ composes when the approximate degree of the outer function f is $\Theta(n)$, where n is the arity of the outer function. Thus approximate degree composes for several symmetric functions (having approximate degree $\Theta(n)$, like Majority and Parity). Though, until recently it was not

[1]We note that some works have also studied the composition of R and $\widetilde{\deg}$ when the inner functions have special properties, for example, [ABK16, BK18, AGJ$^+$17, GLSS19, Li21, BDBGM22].

even clear if $\widetilde{\deg}(\mathsf{OR} \circ \mathsf{AND}) = \Omega(\widetilde{\deg}(\mathsf{OR})\,\widetilde{\deg}(\mathsf{AND}))$ (arguably the simplest of composed functions). OR has approximate degree $O(\sqrt{n})$, and thus the result of [She12] does not prove $\widetilde{\deg}$ composition when the outer function is OR (similarly for AND). In a long series of work by [NS94, Amb05, She13a, BT13, She13b], the question was finally resolved; it was later generalized to the case when the outer function is symmetric [BBGK18a].

In contrast to R composition, no lower bound on the approximate degree of composed function is known with a weaker measure on the outer function. It is well known that for any Boolean function f, $\sqrt{\mathsf{s}(f)} \leq \sqrt{\mathrm{bs}(f)} = O(\widetilde{\deg}(f))$ [NS94]. So a natural step towards proving $\widetilde{\deg}$ composition is: prove a lower bound on $\widetilde{\deg}(f \circ g)$ by $\sqrt{\mathrm{bs}(f)} \cdot \widetilde{\deg}(g)$. We show this result by generalizing the approach of [BBGK18a].

While the techniques used for the composition of R and $\widetilde{\deg}$ are quite different, one can still observe the similarities between the classes of outer functions for which the composition of R and $\widetilde{\deg}$ is known to hold respectively. We further expand these similarities, by extending the classes of outer functions for which the composition theorem holds.

7.2 Our Results and Techniques

It is well-known, by amplification, that $\mathrm{R}(f \circ g) = O(\mathrm{R}(f) \cdot \mathrm{R}(g) \cdot \log \mathrm{R}(f))$. In the case of approximate degree, Shrestov [She13d] showed that $\widetilde{\deg}(f \circ g) = O(\widetilde{\deg}(f) \cdot \widetilde{\deg}(g))$. So, to answer the composition question for R (or $\widetilde{\deg}$), we are only concerned about proving a lower bound on $\mathrm{R}(f \circ g)$ (or $\widetilde{\deg}(f \circ g)$) in terms of $\mathrm{R}(f)$ and $\mathrm{R}(g)$ (or $\widetilde{\deg}(f)$ and $\widetilde{\deg}(g)$) respectively.

We split our results into three parts. In the first part we prove the tight lower bound on $\mathrm{R}(f \circ g)$ when the outer function has full randomized complexity. In the second part we provide a tight lower bound on $\widetilde{\deg}(f \circ g)$ in terms of $\mathrm{bs}(f)$ and $\widetilde{\deg}(g)$. Finally, we also prove composition theorems for R and $\widetilde{\deg}$ when the outer functions have a slightly relaxed notion of symmetry.

7.2.1 Lower bound on $\widetilde{\deg}(f \circ g)$ in terms of block sensitivity of f and $\widetilde{\deg}(g)$

As discussed in the introduction, the composition question for $\widetilde{\deg}$ is only known to hold when the outer function f is symmetric [BBGK18a] or has high approximate degree [She12]. There are also no known lower bounds on $\widetilde{\deg}(f \circ g)$ in terms of weaker measures of f and $\widetilde{\deg}(g)$. Compare this with the situation with respect to composition of R. It was shown in [GJPW18a] that $\mathrm{R}(f \circ g) = \Omega(\mathsf{s}(f)\,\mathrm{R}(g))$, where $\mathsf{s}(f)$ denotes the sensitivity of f. This was later strengthened to $\Omega(\mathsf{fbs}(f)\,\mathrm{R}(g))$ [BDG+20, BDB20a], where $\mathsf{fbs}(f)$ is the fractional block sensitivity of f.

In this second part we show analogous lower bounds on approximate degree of composed function $f \circ g$. Our main result here is the following.

Theorem 7.2. *For all non-constant (possibly partial)*[2] *Boolean functions $f : \{0,1\}^n \to \{0,1\}$ and $g : \{0,1\}^m \to \{0,1\}$, we have*

$$\widetilde{\deg}(f \circ g) = \widetilde{\Omega}(\sqrt{\mathrm{bs}(f)} \cdot \widetilde{\deg}(g)).$$

We first note that the above theorem is tight in terms of block sensitivity, i.e., we cannot have $\widetilde{\deg}(f \circ g) = \widetilde{\Omega}(\mathrm{bs}(f)^c \cdot \widetilde{\deg}(g))$ for any $c > 1/2$. This is because the OR function over n bits witnesses the tight quadratic separation between $\widetilde{\deg}$ and bs, i.e., $\widetilde{\deg}(\mathsf{OR}_n) = \Theta(\sqrt{n}) = \Theta(\sqrt{\mathrm{bs}(\mathsf{OR}_n)})$ [NS94].

We also get the following composition theorem as a corollary. It says that the composition for $\widetilde{\deg}$ holds when the outer function has minimal approximate degree with respect to its block sensitivity. Recall, $\widetilde{\deg}(f) = \Omega(\sqrt{\mathrm{bs}(f)})$ [NS94].

Corollary 7.3. *For all Boolean function $f : \{0,1\}^n \to \{0,1\}$ with $\widetilde{\deg}(f) = \Theta(\sqrt{\mathrm{bs}(f)})$ and for all $g : \{0,1\}^m \to \{0,1\}$, we have $\widetilde{\deg}(f \circ g) = \widetilde{\Theta}(\widetilde{\deg}(f) \cdot \widetilde{\deg}(g))$.*

This complements a result of Sherstov [She12, Theorem 6.6], which shows that composition of $\widetilde{\deg}$ holds when the outer function has maximal $\widetilde{\deg}$ with respect to its arity.

We further note that Corollary 7.3 covers new set of composed functions $f \circ g$ for which the composition theorem for $\widetilde{\deg}$ doesn't follow from the known results [BBGK18a, She12]. For example, consider the Rubinstein function RUB with arity n (Definition 7.16) as the outer function f. It is clearly not a symmetric function. It also doesn't have high approximate degree, i.e., $\widetilde{\deg}(\mathsf{RUB}) = O(\sqrt{n}\log n)$ (Lemma 7.21). Therefore, the composition of $\widetilde{\deg}(\mathsf{RUB} \circ g)$ doesn't follow from the existing results. However, it follows from Corollary 7.3, since $\mathrm{bs}(\mathsf{RUB}) = \Omega(n)$ and so $\widetilde{\deg}(\mathsf{RUB}) = \widetilde{\Theta}(\sqrt{\mathrm{bs}(\mathsf{RUB})})$.

Another example is the sink function SINK over $\binom{n}{2}$ variables (Definition 7.15), which is also not a symmetric function. Furthermore, its approximate degree is $O(\sqrt{n}\log n)$ (Lemma 7.21). Therefore, the composition of $\widetilde{\deg}(\mathsf{SINK} \circ g)$ also doesn't follow from

[2] For definitions of block sensitivity and approximate degree in the context of partial functions, please see Definitions 7.10 and 7.11.

the existing results. Again, it follows from Corollary 7.3, since $\mathrm{bs}(\mathsf{SINK}) = \Theta(n)$ (Observation 7.17) and $\widetilde{\deg}(\mathsf{SINK}) = \widetilde{\Theta}(\sqrt{n})$.

Ideas behind proof of Theorem 7.2 We will first sketch the proof ideas in the case when f and g are total Boolean functions, and then explain how to extend it to partial functions too.

Our starting point is the well known Nisan-Szegedy's embedding of PrOR over $\mathrm{bs}(f)$ many bits in a Boolean function f [NS94]. Carrying out this transformation in $f \circ g$ embeds $\mathsf{PrOR}_{\mathrm{bs}(f)} \circ (g_1, \ldots, g_{\mathrm{bs}(f)})$ into $f \circ g$, where $g_1, \ldots, g_{\mathrm{bs}(f)}$ are different partial functions such that $\mathsf{bdeg}(g_i) \geq \widetilde{\deg}(g)$ for all $i \in [\mathrm{bs}(f)]$[3]. Since the transformation is just substitutions of variables by constants, we further have

$$\widetilde{\deg}(f \circ g) \geq \mathsf{bdeg}(\mathsf{PrOR}_{\mathrm{bs}(f)} \circ (g_1, \ldots, g_{\mathrm{bs}(f)})). \tag{7.1}$$

It now looks like that we can appeal to the composition theorem for PrOR (Theorem 7.13) [BBGK18a] to obtain our theorem. However, there is a technical difficulty – Theorem 7.13 doesn't hold for *different* inner *partial* functions. It only deals with a single total inner function. We therefore generalize the proof of Theorem 7.13 to obtain the following general version of the composition theorem for PrOR.

Theorem 7.4. *For any partial Boolean functions $g_1, g_2, \ldots, g_n$, we have*

$$\mathsf{bdeg}\left(\mathsf{PrOR}_n \circ (g_1, g_2, \ldots, g_n)\right) = \Omega\left(\frac{\sqrt{n} \cdot \min_{i=1}^{n} \mathsf{bdeg}(g_i)}{\log n}\right).$$

We can now obtain our main theorem from Eq. (7.1) and Theorem 7.4. The proof of Theorem 7.4 is a generalization of Theorem 7.13.

We end this part with a comment on how to extend our main theorem to partial functions. Note that with the appropriate definition of block sensitivity (Definition 7.10), the embedding of Nisan-Szegedy's carries through and then the rest of the proof is identical.

7.2.2 Composition results when the outer functions has some symmetry

The class of symmetric functions capture many important function like OR, AND, Parity and Majority. Recall that a function is symmetric when the function value only depends on the Hamming weight of the input; in other words, a function is symmetric iff its value on an

[3] bdeg is the notion of approximate degree in the context of partial functions. For a formal definition, see Definition 7.11.

input remains unchanged even after permuting the bits of the input. As noted earlier, both for R and $\widetilde{\deg}$, composition was known to hold when the outer function was symmetric.

A natural question is, whether one can prove composition theorems when the outer function is *weakly* symmetric (it is symmetric with respect to a weaker notion of symmetry). In this paper we consider one such notion of symmetry – junta-symmetric functions.

Definition 7.5 (k-junta symmetric function)**.** *A function $f : \{0,1\}^n \to \{0,1\}$ is called a k-junta symmetric function if there exists a set $\mathscr{J}$ of size k of variables such that the function value depends on assignments to the variables in $\mathscr{J}$ as well as on the Hamming weight of the whole input.*

k-junta symmetric functions can be seen as a mixture of symmetric functions and k-juntas. This class of functions has been considered previously in literature, particularly in [CFGM12, BWY15] where these functions plays a crucial role. [CFGM12] even presents multiple characterisations of k-junta symmetric functions for constant k. Note that by definition an arbitrary k-junta (i.e., a function that depend on k variables) is also a k-junta symmetric function, since we can consider the dependence on Hamming weight to be trivial. Thus, this notion loses out on the symmetry of the function considered. We, therefore, consider the class of *strongly* k-junta symmetric functions.

Definition 7.6 (Strongly k-junta symmetric function)**.** *A k-junta symmetric function is called strongly k-junta symmetric if every variable is influential. In other words, there exists a setting to the junta variables such that the function value depends on the Hamming weight of the whole input in a non-trivial way.*

We prove that if the outer function is strongly $\sqrt{n}$-junta symmetric ("strongly" indicating that the dependence on the Hamming weight is non-trivial) then $\widetilde{\deg}$ composes.

Theorem 7.7 (Restatement of Theorem 1.12)**.** *Let $k = O(\sqrt{n})$. For any strongly k-junta symmetric function $f : \{0,1\}^n \to \{0,1\}$ and any Boolean function $g : \{0,1\}^m \to \{0,1\}$, we have*

- $\widetilde{\deg}(f \circ g) = \widetilde{\Theta}(\widetilde{\deg}(f) \cdot \widetilde{\deg}(g))$.

Note that if one is able to prove the above theorem for k-junta-symmetric functions (without the requirement of "strongly") for any non-constant k then we would have the full composition theorem.

7.3 Prelimaniries

Notations: We will use $[n]$ to represent the set $\{1,\dots,n\}$. For any (possibly partial) Boolean function $f:\{0,1\}^n \to \{0,1,*\}$ we will denote by $\mathrm{Dom}(f)$ the set $f^{-1}(\{0,1\})$. The arity of f is the number of variables - in this case n. A Boolean function $f:\{0,1\}^n \to \{0,1,*\}$ is said to be total if $\mathrm{Dom}(f) = \{0,1\}^n$. Any function (not otherwise stated) will be a total Boolean function.

For any $x \in \{0,1\}^n$, we will use $|x|$ to denote the number of 1s in x, that is, the Hamming weight of the string x. The string x^i denotes the modified string x with the i-th bit flipped. Similarly, x^B is defined to be the string such that all the bits whose index is contained in the set $B \subseteq [n]$ are flipped in x.

Following is a formal definition of (partial) function composition.

Definition 7.8 (Generalized composition of functions)**.** *For any (possibly partial) Boolean function* $f:\{0,1\}^n \to \{0,1,*\}$ *and* n *(possibly partial) Boolean functions* $g_1, g_2, \dots, g_n$*, define the (possibly partial) composed function*

$$f \circ (g_1, g_2, \dots, g_n)(x_1, x_2, \dots, x_n) = f(g_1(x_1), g_2(x_2), \dots, g_n(x_n)),$$

where g_i*'s can have different arities and, moreover, if* $x_i \notin \mathrm{Dom}(g_i)$ *for any* $i \in [n]$ *or the string* $(g_1(x_1), g_2(x_2), \dots, g_n(x_n)) \notin \mathrm{Dom}(f)$*, then the function* $f \circ g$ *outputs* $*$*.*

In this chapter, we use the standard definitions of various complexity measures like sensitivity, block-sensitivity, fractional block sensitivity, approximate degree and bounded approximate degree, and the partial function Promise-OR. We present the formal definitions in the following subsections.

7.3.1 Standard definitions and functions

Standard definition of complexity measures

We will start by recalling the formal definition of $\widetilde{\deg}$.

Definition 7.9 (Approximate degree ($\widetilde{\deg}$))**.** *A polynomial* $p:\mathbb{R}^n \to \mathbb{R}$ *is said to approximate a Boolean function* $f:\{0,1\}^n \to \{0,1\}$ *if* $|p(x) - f(x)| \le 1/3, \quad \forall x \in \{0,1\}^n$*. The approximate degree of* f*,* $\widetilde{\deg}(f)$*, is the minimum possible degree of a polynomial that approximates* f*.*

Note that the constant $1/3$ in the above definitions can be replaced by any constant strictly smaller than $1/2$ which changes $\widetilde{\deg}(f)$ by only a constant factor.

Other than R and $\widetilde{\deg}$, two important related measures are sensitivity ($\mathsf{s}(f)$) and block sensitivity ($\mathrm{bs}(f)$). While the sensitivity and block sensitivity of a total function is well defined, we note that for the case of partial functions there are at least two valid ways of extending the definition from total functions to partial functions. All our results in this paper will hold for partial functions with the following definitions of sensitivity and block sensitivity.

Definition 7.10. *The sensitivity* $\mathsf{s}(f,x)$ *of a function* $f : \{0,1\} \to \{0,1,*\}$ *on x is the maximum number s such that there are indices* $i_1, i_2, \ldots, i_s \in [n]$ *with* $f(x^{i_j}) = 1 - f(x)$*, for all* $1 \le j \le s$. *Here x^i is obtained from x by flipping the i^{th} bit. The sensitivity of f is defined to be* $\mathsf{s}(f) = \max_{x \in \mathrm{Dom}(f)} \mathsf{s}(f,x)$.

Similarly, the block sensitivity $\mathrm{bs}(f,x)$ *of a function* $f : \{0,1\} \to \{0,1,*\}$ *on x is the maximum number b such that there are disjoint sets* $B_1, B_2, \ldots, B_b \subseteq [n]$ *with* $f(x^{B_j}) = 1 - f(x)$ *for all* $1 \le j \le b$. *Recall x^{B_j} is obtained from x by flipping all bits inside B_j. The block sensitivity of f is defined to be* $\mathrm{bs}(f) = \max_{x \in \mathrm{Dom}(f)} \mathrm{bs}(f,x)$.

In the definition of block sensitivity, the constraint that the blocks has to be disjoint can be relaxed by extending the definition to "fractional blocks". This gives the measure of fractional block sensitivity. Please check Definition 2.12 in Chapter 2 for a formal definition of fractional block sensitivity.

For the composition of $\widetilde{\deg}$ The definition of $\widetilde{\deg}$ can naturally be extended to partial functions f by restricting the definition to hold only for inputs in $\mathrm{Dom}(f)$, i.e., it doesn't specify the value of the approximating polynomial on inputs *not* in $\mathrm{Dom}(f)$. So the approximating polynomial can take arbitrarily large values on points outside the domain. However, for the purpose of understanding the composition of approximate degree of Boolean functions (or even total Boolean functions) one need a measure of approximate degree of partial Boolean functions which is bounded on all the points of the Boolean cube.

Definition 7.11 (Bounded approximate degree (bdeg)). *For a partial Boolean function* $f : \{0,1\}^n \to \{0,1,*\}$*, the bounded approximate degree of f* ($\mathsf{bdeg}(f)$) *is the minimum possible degree of a polynomial p such that*

- $|p(x) - f(x)| \le 1/3, \quad \forall x \in \mathrm{Dom}(f)$*, and*
- $0 \le p(x) \le 1 \quad \forall x \in \{0,1\}^n$.

In other words, we take the minimum possible degree of a polynomial which is bounded for all possible inputs ($p(x) \in [0,1]$ for all $x \in \{0,1\}^n$), and it approximates f in the usual sense over $\mathrm{Dom}(f)$.

Over the years people have tried to study the composition of $\widetilde{\deg}$ with different outer functions. In this context the following restriction of OR is an important partial function:

Definition 7.12 (Promise-OR)**.** *Promise-*OR *(denoted by* PrOR_n*) is the function* $\mathsf{PrOR}_n : \{0,1\}^n \to \{0,1,*\}$ *such that:*

$$\mathsf{PrOR}_n(x) = \begin{cases} 0 & \textit{if } |x| = 0, \\ 1 & \textit{if } |x| = 1, \\ * & \textit{otherwise.} \end{cases}$$

We will also be using the following theorem of [BBGK18a] regarding the composition question of bdeg when the outer function is PrOR_n. Informally, we will call it the Promise-OR composition theorem.

Theorem 7.13 ([BBGK18a])**.** *For any Boolean function* $g : \{0,1\}^m \to \{0,1\}$ *we have,*

$$\mathsf{bdeg}\left(\mathsf{PrOR}_n \circ g\right) = \Omega\left(\sqrt{n} \cdot \widetilde{\deg}(g) / \log n\right).$$

7.3.2 Some important Boolean functions

In this section, we define some important Boolean functions that have been used in the paper. We start with Multiplexer Function or Addressing Function.

Definition 7.14 (Multiplexer Function or Addressing Function)**.** *The function* $\mathsf{MUX} : \{0,1\}^{k+2^k} \to \{0,1\}$ *with input* $(x_0, \ldots, x_{k-1}, y_0, \ldots, y_{2^k-1})$ *outputs the bit* y_t*, where* $t = \sum_{i=0}^{k-1} x_i 2^i$*.*

The following function was defined in [CMS20].

Definition 7.15 (SINK)**.** *Consider a tournament defined on k vertices with* $\binom{k}{2}$ *variables such that, for* $i < j$*, if* $x_{ij} = 1$ *then there is an outgoing edge from i to j. A vertex* $i \in [n]$ *is a sink-vertex if all edges incident to it are incoming edges. For* $x \in \binom{k}{2}$, $\mathsf{SINK}(x) = 1$ *if there a vertex* $i \in [n]$ *such that i is a sink-vertex, and* 0 *otherwise.*

Definition 7.16 (Rubinstein function [Rub95])**.** *Let* $g : \{0,1\}^k \to \{0,1\}$ *be such that* $g(x) = 1$ *iff x contains two consecutive ones and the rest of the bits are* 0*. The Rubinstein function, denoted by* $\mathsf{RUB} : \{0,1\}^{k^2} \to \{0,1\}$*, is defined to be* $\mathsf{RUB} = \mathsf{OR}_k \circ g$*.*

7.3.3 Some properties of SINK and Rubinstein function

Following is an easy observation.

Observation 7.17. *The sensitivity of* $\mathsf{SINK} : \{0,1\}^{\binom{k}{2}} \to \{0,1\}$ *is at least* $(k-1)$. *Consider a tournament on vertices* $v_1, \dots, v_k$ *such that* v_1 *is a sink-vertex and* $(v_2, \dots, v_k, v_2)$ *is a cycle. Observe that flipping any edge incident to* v_1 *changes the value of the function from* 1 *to* 0. *In particular,* $\mathsf{bs}(\mathsf{SINK}) \geq \mathsf{s}(\mathsf{SINK}) = \Omega(k)$.

We now want to give an upper bound on the approximate degree of these two functions. For this, we first need the following generalization of approximate degree.

Definition 7.18 (ε-Error Approximate Degree ($\widetilde{\deg}_\varepsilon$)). *A polynomial* $p : \mathbb{R}^n \to \mathbb{R}$ *is said to* ε*-approximate a Boolean function* $f : \{0,1\}^n \to \{0,1\}$ *if*

$$|p(x) - f(x)| \leq \varepsilon, \quad \forall x \in \{0,1\}^n.$$

The ε*-approximate degree of* f $(\widetilde{\deg}_\varepsilon(f))$ *is the minimum possible degree of a polynomial which* ε*-approximates* f.

It is known (see [BNRdW07a] also [Tal14, Appendix A]) that:

Lemma 7.19 ([BNRdW07a, Tal14]). *Given a polynomial* p *that approximates a Boolean function* f *to error* $1/3$, *one can come up with a polynomial* p' *that* ε*-approximates* f *such that* $\deg(p') = O(\deg(p) \log(1/\varepsilon))$.

Also, the following theorem will be used in this section.

Theorem 7.20 ([She13d]). *For all Boolean functions* f *and* g, $\widetilde{\deg}(f \circ g) = O(\widetilde{\deg}(f)\widetilde{\deg}(g))$.

We now upper bound the approximate degree of SINK and RUB.

Lemma 7.21. *For the sink function* $\mathsf{SINK} : \{0,1\}^{\binom{k}{2}} \to \{0,1\}$ *and the Rubinstein function* $\mathsf{RUB} : \{0,1\}^{k^2} \to \{0,1\}$ *we have*

1. $\widetilde{\deg}(\mathsf{SINK}) = O(\sqrt{k} \log k)$, *and*
2. $\widetilde{\deg}(\mathsf{RUB}) = O(k \log k)$.

Proof. We first upper bound the approximate degree of SINK. For every vertex $i \in [n]$, there is a sink at that vertex if and only if all edges incident to that vertex are incoming. This is equivalent to saying that the variables x_{ij}, when $j > i$, are 0 and x_{ji}, when $j < i$, are 1. This can be computed by an AND_k function and we call this function $\mathsf{AND}_k^{(i)}$.

Also, since any tournament has at most one sink vertex, SINK can be expressed as the sum of $\mathsf{AND}_k^{(i)}$'s, for $i \in [k]$. Recall that the approximate degree of AND_k is $O(\sqrt{k})$, which, from Theorem 7.19, implies that $1/(3k)$-error approximate degree of AND_k is $O(\sqrt{k}\log k)$[4]. Replacing each $\mathsf{AND}_k^{(i)}$ with $1/(3k)$-error approximating polynomial gives a $1/3$-error approximating polynomial for SINK with degree $O(\sqrt{k}\log k)$.

Now we upper bound the approximate degree of Rubinstein function. Recall that $\mathsf{RUB} : \{0,1\}^{k^2} \to \{0,1\}$ is defined as $\mathsf{RUB} = \mathsf{OR}_k \circ g$, where $g : \{0,1\}^k \to \{0,1\}$ is a function such that $g(x) = 1$ if and only if x contains two consecutive 1's and the rest of the bits are 0. Observe that there are only $(k-1)$ inputs on which g takes value 1, call these inputs $z_1, \dots, z_{k-1}$. Let $\mathsf{AND}_k^{(i)}$ be the Boolean function which evaluates to 1 if and only if its input is z_i. Thus g can be expressed as the sum of $\mathsf{AND}_k^{(i)}$'s, for $i \in [k]$. By a similar argument as in the last paragraph, the $1/3$-error approximate degree of g can be bounded by $O(\sqrt{k}\log k)$. From Theorem 7.20, this implies an $O(k \log k)$ upper bound on the approximate degree of RUB. □

7.4 Composition of approximate degree in terms of block sensitivity

In this section, we study the composition question for approximate degree. Recall that the composition question asks: whether for all Boolean functions f and g

$$\widetilde{\deg}(f \circ g) = \widetilde{\Omega}(\widetilde{\deg}(f)\,\widetilde{\deg}(g))?$$

Following our discussion from the introduction, we know that the above composition is known to hold for only two sub-classes of outer functions, namely symmetric functions [BBGK18a] and functions with high approximate degree [She12]. It is thus natural to seek weaker lower bounds to make progress towards the composition question. One way to weaken the expression on the right-hand side would be to replace the measure $\widetilde{\deg}(f)$ by a weaker measure (like $\sqrt{\mathsf{s}(f)}$, $\sqrt{\mathrm{bs}(f)}$ or $\sqrt{\mathsf{fbs}(f)}$). Here we will establish one such lower bound of $\sqrt{\mathrm{bs}(f)}\,\widetilde{\deg}(g)$.

We restate our result below:

[4]Note that $1/(3k)$-error approximate degree of AND_k can be even lower $O(\sqrt{k\log k})$ (see [dW08] for more details). We are dealing with the upper bound and for our purpose, it is sufficient to use $\sqrt{k}\log k$ bound.

For all non-constant (possibly partial)[5] Boolean functions $f : \{0,1\}^n \to \{0,1\}$ and $g : \{0,1\}^m \to \{0,1\}$, we have

$$\widetilde{\deg}(f \circ g) = \widetilde{\Omega}(\sqrt{\mathrm{bs}(f)} \cdot \widetilde{\deg}(g)).$$

We note that many analogous results are known in the setting of the composition of R; see, for example, [GJPW18a, BDG$^+$20, BDB20a, BK18, AGJ$^+$17, GLSS19, BDBGM22]. To the best of our knowledge, this is the first such result in the setting of $\widetilde{\deg}$. We present only a proof sketch here; most of the technical parts of the proof appear in Section 7.5.

Further, we present the sketch of the proof in two parts. For simplicity, in the first part we sketch a proof of the lower bound $\sqrt{\mathsf{s}(f)}\,\widetilde{\deg}(g)$ for total function f, and then in the second part we modify the arguments to obtain Theorem 7.2.

We begin with a proof sketch for a lower bound of $\sqrt{\mathsf{s}(f)}\,\widetilde{\deg}(g)$. Let $x \in \{0,1\}^n$ be an input having the maximum sensitivity with respect to f, and $S \subseteq [n]$ be the set of sensitive bits at x ($|S| = \mathsf{s}(f)$). Consider the subfunction f' obtained from f by fixing the set of variables *not* in S according to x. By construction, f' is defined over $\mathsf{s}(f)$ many variables and is fully sensitive at the input $x|_S$ given by x restricted to the indices in S. Since f' is a subfunction of f and g is non-constant, we have $\widetilde{\deg}(f \circ g) \geq \widetilde{\deg}(f' \circ g)$.

Notice that f' at the neighborhood of x, in the Boolean cube, is the partial function

PrOR (Definition 7.12) or its negation. Therefore, we have $\widetilde{\deg}(f \circ g) \geq \widetilde{\deg}(f' \circ g) \geq \mathsf{bdeg}(\mathsf{PrOR}_{|S|} \circ g)$ (see Definition 7.11 for a definition of the bounded approximate degree). We can now invoke the composition theorem for PrOR (Theorem 7.13) [BBGK18a] to obtain our lower bound:

$$\widetilde{\deg}(f \circ g) \geq \widetilde{\deg}(f' \circ g) \geq \mathsf{bdeg}(\mathsf{PrOR}_{|S|} \circ g) = \widetilde{\Omega}(\sqrt{\mathsf{s}(f)}\,\widetilde{\deg}(g)).$$

However, there is a technical issue with our argument above. When we claimed that f' looks like a PrOR function we were not quite correct. Technically, it is a Shifted-PrOR function $\mathsf{PrOR}_{|S|}^{x|_S}$, where $\mathsf{PrOR}_n^a(y_1, y_2, \ldots, y_n) := \mathsf{PrOR}_n(y_1 \oplus a_1, y_2 \oplus a_2, \ldots, y_n \oplus a_n)$ for $a \in \{0,1\}^n$. Formally, we have

$$\widetilde{\deg}(f \circ g) \geq \widetilde{\deg}(f' \circ g) \geq \mathsf{bdeg}(\mathsf{PrOR}_{|S|}^{x|_S} \circ g) = \mathsf{bdeg}(\mathsf{PrOR}_{|S|} \circ (g_1, \ldots, g_{|S|})), \tag{7.2}$$

[5]For definitions of block sensitivity and approximate degree in the context of partial functions, please see Definitions 7.10 and 7.11.

where $g_i = g$ or $\neg g$ depending on the corresponding i-th bit in $x|_S$.

We, therefore, need a composition theorem for PrOR with *different* inner functions, while Theorem 7.13 requires that all the inner functions be same. In fact, we would need a more general composition theorem with *different* inner *partial* functions, which we restate below. This generalization is crucially used when dealing with block sensitivity.

The proof of Theorem 7.4 is a generalization of proof of Theorem 7.13. For the sake of completeness and reader's convenience, we present the proof in Section 7.5 (Theorem 7.29)[6].

Now returning to Eq. (7.2) and using Theorem 7.4, we obtain the desired lower bound:

$$\widetilde{\deg}(f \circ g) \geq \widetilde{\deg}(f' \circ g) \geq \mathsf{bdeg}(\mathsf{PrOR}_{|S|}^{x|_S} \circ g) = \widetilde{\Omega}(\sqrt{\mathsf{s}(f)}\,\widetilde{\deg}(g)).$$

We are now ready to present the modifications required to prove the lower bound $\widetilde{\Omega}(\sqrt{\mathrm{bs}(f)}\,\widetilde{\deg}(g))$.

Proof of Theorem 7.2. Let $b = \mathrm{bs}(f)$ and $a = (a_1, a_2, \ldots, a_n)$ be an input where f achieves the maximum block sensitivity. Further, let $B_1, B_2, \ldots, B_b$ be disjoint minimal sets of variables that achieves the block sensitivity at a, i.e., $f(a) \neq f(a^{B_i})$ for all $i \in [b]$. Recall, a^{B_i} denotes the Boolean string obtained from a by flipping the bits at all the indices given by B_i. Define a partial function $f' : \{0,1\}^n \to \{0,1,*\}$ such that,

$$f'(x) = \begin{cases} 0 & \text{if } x = a, \\ 1 & \text{if } x = a^{B_i}, \text{ for some } i \in [b], \\ * & \text{otherwise.} \end{cases}$$

Note that f contains f' or its negation as a sub function. Thus, $\widetilde{\deg}(f \circ g) \geq \mathsf{bdeg}(f' \circ g)$.

Since g is non-constant, we can fix the indices *not* in $\bigcup_{i=1}^{b} B_i$ according to a to obtain $f'' \circ g$. We would now like to embed PrOR_b over the remaining variables in f''. For this purpose we define the following partial functions: for every $i \in [b]$, let $I_i : \{0,1\}^{B_i} \to \{0,1,*\}$ be such that

$$I_i(x) = \begin{cases} 0 & \text{if } x = a|_{B_i}, \\ 1 & \text{if } x = a^{B_i}|_{B_i}, \\ * & \text{otherwise.} \end{cases}$$

[6] For a nearly optimal generalization see Theorem 7.30 in Section 7.5.

Now observe that $f'' \circ g$ can be rewritten as $\mathsf{PrOR}_b \circ (I_1 \circ g, \ldots, I_b \circ g)$. We therefore have

$$\begin{aligned}\widetilde{\deg}(f \circ g) \geq \mathsf{bdeg}(f' \circ g) \geq \mathsf{bdeg}(f'' \circ g) &= \mathsf{bdeg}(\mathsf{PrOR}_b \circ (I_1 \circ g, \ldots, I_b \circ g)) \\ &= \Omega\left(\frac{\sqrt{b} \cdot \min_i \mathsf{bdeg}(I_i \circ g)}{\log b}\right) = \widetilde{\Omega}\left(\sqrt{b} \cdot \widetilde{\deg}(g)\right),\end{aligned}$$

where the second last equality follows from Theorem 7.4 and the last equality uses the fact $\mathsf{bdeg}(I_i \circ g) \geq \widetilde{\deg}(g)$ for all i, which in turn follows from each I_i being non-constant. □

We end this section with few final remarks. As a corollary to Theorem 7.2 we have the following composition for $\widetilde{\deg}$ when the outer function has minimal approximate degree with respect to its block sensitivity.

Corollary 7.22 (Restatement of Corollary 7.3). *For all Boolean function $f : \{0,1\}^n \to \{0,1\}$ with $\widetilde{\deg}(f) = \Theta(\sqrt{\mathrm{bs}(f)})$ and for all $g : \{0,1\}^m \to \{0,1\}$, we have $\widetilde{\deg}(f \circ g) = \widetilde{\Theta}(\widetilde{\deg}(f) \cdot \widetilde{\deg}(g))$.*

We also note that the set of Boolean functions with $\widetilde{\deg}(f) = \Theta(\sqrt{\mathrm{bs}(f)})$ includes examples of *non-symmetric* functions f with *low* approximate degree. In other words, when such functions are outer function in a composed function then the composition of $\widetilde{\deg}$ doesn't follow from the known results [BBGK18a, She12]. For example, consider the Rubinstein function RUB with arity n (Definition 7.16). It is a non-symmetric function with $\widetilde{\deg}(\mathsf{RUB}) = O(\sqrt{n}\log n)$ (Lemma 7.21). Thus, the composition of $\widetilde{\deg}(\mathsf{RUB} \circ g)$ does not follow from [BBGK18a] or [She12]. However, $\widetilde{\deg}(\mathsf{RUB}) = \widetilde{\Theta}(\sqrt{\mathrm{bs}(\mathsf{RUB})})$, and hence from Corollary 7.3, $\widetilde{\deg}(\mathsf{RUB} \circ g) = \widetilde{\Theta}(\widetilde{\deg}(\mathsf{RUB})\,\widetilde{\deg}(g))$.

Another example is given by the SINK function on $\Theta(n^2)$ variables (Definition 7.15). Its approximate degree is $O(\sqrt{n}\log n)$ (Lemma 7.21), while $\mathrm{bs}(\mathsf{SINK}) = \Theta(n)$. Thus, $\widetilde{\deg}(\mathsf{SINK} \circ g) = \widetilde{\Theta}(\widetilde{\deg}(\mathsf{SINK})\,\widetilde{\deg}(g))$ follows from Corollary 7.3.

As stated in the introduction, we recall that Theorem 7.2 is tight in terms of block-sensitivity, i.e., the lower bound can not be improved to $\widetilde{\Omega}(\mathrm{bs}(f)^c \cdot \widetilde{\deg}(g))$ for some $c > 1/2$.

7.5 Approximate degree of Promise-OR composed with different inner functions

In this section we show that the approximate degree composes when the outer function is PrOR and the inner functions are (possibly) different partial functions. The proof is essentially a straightforward generalization of the proof of Theorem 7.13 [BBGK18a, Theorem 16 (arXiv version)]. However, for the sake of completeness and reader's convenience, we give an overview of the proof here. We will need some definitions and theorems from [BBGK18a] which we state now. We start with the definition of a problem called "singleton combinatorial group testing". It generalizes the combinatorial group testing problem.

Definition 7.23 (Singleton CGT). *Let D be the set of all $w \in \{0,1\}^{2^n}$ for which there exists an $x \in \{0,1\}^n$ such that for all $S \subseteq [n]$ satisfying $\sum_{i\in S} x_i \in \{0,1\}$, we have $\sum_{i\in S} x_i = w_S$. Note that for all $w \in D$, the string x is uniquely defined by $x_i = w_{\{i\}}$. Let us denote this string by $x(w)$. we then define the partial function $\mathsf{SCGT}_{2^n} : D \to \{0,1\}^n$ by $\mathsf{SCGT}_{2^n}(w) = x(w)$.*

The following theorem is from Theorem 19 from [BBGK18a] (arXiv version), previously proved by [Bel15]:

Theorem 7.24 ([Bel15]). *The bounded-error quantum query complexity of SCGT_{2^n} is $\Theta(\sqrt{n})$.*

For a formal Definition of bounded error quantum query complexity we refer the survey by [Bd02]. Before we state the next result that we need from [BBGK18a] we are defining robustness of a polynomial to input noise.

Definition 7.25 (Robustness to input noise). *For any function $f : \{0,1\}^n \to \{0,1,*\}$ we say a polynomial $p : \{0,1\}^n \to \mathrm{R}$ approximately computes f with δ-robustness where $\delta \in [0, \frac{1}{2})$ if for any $x \in \mathsf{Dom}(f)$ and $\Delta \in [-\delta, \delta]^n$, we have $|f(x) - p(\Delta + x)| \leq \frac{1}{3}$.*

Now we are ready to state the next result.

Theorem 7.26 ([BBGK18a, Theorem 17 (arXiv version)]). *For a partial Boolean function f, there exists a bounded multilinear polynomial p of degree $O(\mathsf{Q}(f))$ that approximates f with robustness $\Omega(1/\mathsf{Q}(f)^2)$ where $\mathsf{Q}(f)$ is the bounded error quantum query complexity of the function f.*

We refer [BBGK18a] for more details about robustness of a polynomial induces by quantum algorithm. We also need the existence of a multilinear robust polynomial for $\mathsf{XOR}_n \circ \mathsf{SCGT}_{2^n}$, which follows from Theorems 7.24 and 7.26 above, where $\mathsf{XOR}_n \circ \mathsf{SCGT}_{2^n}$ is the parity of n output bits of SCGT_{2^n}.

Theorem 7.27 ([BBGK18a, Theorem 20 (arXiv version)]). *There is a real polynomial p of degree $O(\sqrt{n})$ over 2^n variables $\{w_S\}_{S\subseteq[n]}$ and a constant $c \geq 10^{-5}$ such that for any input $w \in \{0,1\}^{2^n}$ with $\mathsf{XOR}_n \circ \mathsf{SCGT}_{2^n}(w) \neq *$ and any $\Delta \in [-c/n, c/n]^{2^n}$,*

$$|p(w+\Delta) - \mathsf{XOR}_n \circ \mathsf{SCGT}_{2^n}(w)| \leq 1/3.$$

Furthermore, p is multilinear and for all $w \in \{0,1\}^{2^n}$, $p(w) \in [0,1]$.

We also need the following result of Sherstov that shows composition holds for the approximate degree of the parity of n different functions.

Theorem 7.28 ([She12, Theorem 5.9]). *For any partial Boolean functions $f_1, \dots, f_n$, we have*

$$\mathsf{bdeg}(\mathsf{XOR} \circ (f_1, \dots, f_n)) = \Omega\left(\sum_{i=1}^{n} \mathsf{bdeg}(f_i)\right).$$

We are now ready to prove Theorem 7.4 which we restate below.

Theorem 7.29. *For any partial Boolean functions $f_1, f_2, \dots, f_n$, we have*

$$\mathsf{bdeg}\left(\mathsf{PrOR}_n \circ (f_1, f_2, \dots, f_n)\right) = \Omega\left(\frac{\sqrt{n} \cdot \min_{i=1}^{n} \mathsf{bdeg}(f_i)}{\log n}\right).$$

Furthermore the following upper bound also holds,

$$\mathsf{bdeg}\left(\mathsf{PrOR}_n \circ (f_1, f_2, \dots, f_n)\right) = O\left(\sqrt{n} \cdot \max_{i=1}^{n} \mathsf{bdeg}(f_i) \cdot \log n\right).$$

Proof. The upper bound follows by first amplifying the approximation of inner function to within error $\Theta(1/n)$ and then composing with the polynomial given by Theorem 7.26 for PrOR.

For amplification one can use the univariate amplification polynomial of degree $O(\log(1/\varepsilon))$ that maps $[0,1/3]$ to $[0,\varepsilon]$, $[2/3,1]$ to $[1-\varepsilon,1]$, and $[1/3,2/3]$ to $[0,1]$ given by [BNRdW07a, Lemma 1].

We now give an overview of the lower bound proof following [BBGK18a].

Let q be an approximating polynomial for $\mathsf{PrOR} \circ (f_1, \dots, f_n)$ of degree $T := \mathsf{bdeg}(\mathsf{PrOR} \circ (f_1, \dots, f_n))$, which is also bounded on all Boolean inputs outside the promise. Let q' be the polynomial obtained from q by amplifying the approximation to within error c/n on all inputs in the promise, where c is the constant from Theorem 7.27. Note that the degree of q' is $O(T \cdot \log n)$, and it remains bounded on all possible Boolean inputs.

We can assume without loss of generality that $f_1, f_2, \dots, f_n$ are non-constant partial Boolean functions. Therefore for each f_i there exists an input y such that $f_i(y) = 0$. For all $S \subseteq [n]$, we now define a polynomial q'_S using q that approximates $\mathsf{PrOR} \circ (f_i)_{i \in S}$ by setting the variables of each f_i, $i \notin S$, to an input where it evaluates to 0. Note that the degree of q'_S is bounded by the degree of q'.

Now consider the polynomial p over 2^n variables $\{w_S\}_{S\subseteq[n]}$ given by Theorem 7.27. Let r be the polynomial obtained from p by replacing the variables w_S by polynomials q'_S, i.e., $r = p \circ (q'_S)_{S\subseteq[n]}$. Clearly the degree of r is $O(T\sqrt{n}\log n)$.

It can now be argued that r approximates $\mathsf{XOR} \circ (f_1,\dots,f_n)$ to error within $1/3$. A slight care is needed in this argument for p expects inputs which are Δ-close to Boolean values $\{0,1\}$. However, it may happen that some q'_S, though bounded in $[0,1]$, is not close to Boolean values $\{0,1\}$. This is where we will use the fact that p is also *multilinear*, and hence the value of p on a convex combination of Boolean inputs is equal to the convex combination of values of p on the Boolean inputs. Hence, if p works correctly when all inputs are in $\{0,1\}$, then it must also be correct on inputs in $[0,1]$. A final thing to note is that any invalid input (to p) in $[0,1]^{2^n}$ can be written as a convex combination of valid inputs.

Thus, we have

$$\mathsf{bdeg}(\mathsf{XOR} \circ (f_1,\dots,f_n)) = O(T\sqrt{n}\log n) = O(\sqrt{n}\log n \cdot \mathsf{bdeg}(\mathsf{PrOR} \circ (f_1,\dots,f_n))).$$

Whereas from Theorem 7.28 we have

$$\mathsf{bdeg}(\mathsf{XOR} \circ (f_1,\dots,f_n)) = \Omega\left(\sum_{i=1}^{n} \mathsf{bdeg}(f_i)\right) = \Omega\left(n \cdot \min_i \mathsf{bdeg}(f_i)\right).$$

Combining the two, we obtain the lower bound

$$\mathsf{bdeg}(\mathsf{PrOR} \circ (f_1,\dots,f_n)) = \Omega\left(\frac{\sqrt{n}\cdot \min_{i=1}^{n} \mathsf{bdeg}(f_i)}{\log n}\right).$$

□

We will now use this weak bound to establish nearly optimal bound for the approximate degree of PrOR composed with n different *partial* functions. This will again be a simple generalization of OR composed with different functions [BBGK18a, Theorem 37]. For the sake of completeness, we work out some of the details.

Theorem 7.30. *For any partial Boolean functions $f_1, f_2, \ldots, f_n$, we have*

$$\mathsf{bdeg}\left(\mathsf{PrOR}_n \circ (f_1, f_2, \ldots, f_n)\right) = \widetilde{\Theta}\left(\sqrt{\sum_{i=1}^{n} \mathsf{bdeg}(f_i)^2}\right),$$

when the lcm of $\mathsf{bdeg}(f_i)^2$ *for* $i \in [n]$ *is* $\Theta(\max_i \ \mathsf{bdeg}(f_i)^2)$.

Proof. As mentioned before, the proof is merely working out the details of [BBGK18a, Theorem 37] while keeping in mind that we are working with *partial* functions.

Let $F = \mathsf{PrOR}_n \circ (f_1, f_2, \ldots, f_n)$, $d_i = \mathsf{bdeg}(f_i)^2$ for $i \in [n]$, and ℓ be the lcm of d_i's. Now consider the function $G = \mathsf{PrOR}_\ell \circ F$. From Theorem 7.29, we have the following bounds on $\mathsf{bdeg}(G)$ up to constants

$$\frac{\sqrt{\ell} \cdot \mathsf{bdeg}(F)}{\log \ell} \leq \mathsf{bdeg}(G) \leq \sqrt{\ell} \cdot \mathsf{bdeg}(F) \cdot \log \ell. \tag{7.3}$$

Now using the associativity of PrOR we can rewrite G as

$$G = \mathsf{PrOR}_{n\ell} \circ (\underbrace{f_1, \ldots, f_1}_{\ell \text{ times}}, \ldots, \underbrace{f_n, \ldots, f_n}_{\ell \text{ times}}). \tag{7.4}$$

Further regrouping f_i's, we can rewrite G as follows

$$G = \mathsf{PrOR}_d \circ (\underbrace{\mathsf{PrOR}_{\ell/d_1} \circ f_1, \ldots, \mathsf{PrOR}_{\ell/d_1} \circ f_1}_{d_1 \text{ times}}, \ldots, \underbrace{\mathsf{PrOR}_{\ell/d_n} \circ f_n, \ldots, \mathsf{PrOR}_{\ell/d_n} \circ f_n}_{d_n \text{ times}}), \tag{7.5}$$

where $d = \sum_{i=1}^{n} d_i$. Now using Theorem 7.29 and $\sqrt{d_i} = \mathsf{bdeg}(f_i)$, we obtain following bounds for $\mathsf{PrOR}_{\ell/d_i} \circ f_i$ (up to constants)

$$\frac{\sqrt{\ell}}{\log(\ell/d_i)} \leq \mathsf{bdeg}(\mathsf{PrOR}_{\ell/d_i} \circ f_i) \leq \sqrt{\ell} \cdot \log(\ell/d_i). \tag{7.6}$$

Now consider (7.5) and using Theorem 7.29 along with (7.6), we obtain

$$\frac{\sqrt{d\ell}}{\log d \cdot \log \ell} \leq \mathsf{bdeg}(G) \leq \sqrt{d\ell} \cdot \log \ell \cdot \log d \tag{7.7}$$

Now from (7.7) and (7.3) it follows

$$\frac{\sqrt{d}}{\log d \cdot \log^2 \ell} \leq \mathsf{bdeg}(F) \leq \sqrt{d} \cdot \log^2 \ell \cdot \log d.$$

□

7.6 Composition results when the outer functions have some symmetry

7.6.1 Composition theorem for junta-symmetric functions

The class of symmetric functions capture many important function like OR, AND, Parity and Majority. Recall that a function is symmetric when the function value only depends on the Hamming weight of the input; in other words, a function is symmetric iff its value on an input remains unchanged even after permuting the bits of the input. As noted earlier, both for R and $\widetilde{\deg}$, composition was known to hold when the outer function was symmetric.

A natural question is, whether one can prove composition theorems when the outer function is *weakly* symmetric (it is symmetric with respect to a weaker notion of symmetry). In this paper we consider one such notion of symmetry – junta-symmetric functions. Check Definition 7.5.

k-junta symmetric functions can be seen as a mixture of symmetric functions and k-juntas. This class of functions has been considered previously in literature, particularly in [CFGM12, BWY15] where these functions plays a crucial role. [CFGM12] even presents multiple characterisations of k-junta symmetric functions for constant k. Note that by definition an arbitrary k-junta (i.e., a function that depend on k variables) is also a k-junta symmetric function, since we can consider the dependence on Hamming weight to be trivial. Thus, this notion loses out on the symmetry of the function considered. We, therefore, consider the class of *strongly* k-junta symmetric functions. A k-junta symmetric function is called strongly k-junta symmetric if every variable is influential. In other words, there exists a setting to the junta variables such that the function value depends on the Hamming weight of the whole input in a non-trivial way.

We prove that if the outer function is strongly $\sqrt{n}$-junta symmetric ("strongly" indicating that the dependence on the Hamming weight is non-trivial) then $\widetilde{\deg}$ composes.

Theorem 7.31 (Restatement of Theorem 1.12)**.** *For any strongly k-junta symmetric function $f : \{0,1\}^n \to \{0,1\}$ and any Boolean function $g : \{0,1\}^m \to \{0,1\}$, we have*

- $\widetilde{\deg}(f \circ g) = \widetilde{\Theta}(\widetilde{\deg}(f) \cdot \widetilde{\deg}(g))$ *where* $k = O(\sqrt{n})$.

Note that if one can prove the above theorem for k-junta-symmetric functions (without the requirement of "strongly") for any non-constant k then we would have the full composition theorem.

7.6.2 Composition of approximate degree for $\sqrt{n}$-junta symmetric functions

In this subsection, we will prove the composition result of $\widetilde{\deg}$ when the outer function has some amount of symmetry. Of course, there are various notions of symmetry. Traditionally a function is said to have the maximum amount of symmetry when the function value is invariant under any permutation of the variables. Such functions are called symmetric. Symmetric functions are very well studied in the literature of Boolean function analysis. In terms of composition theorems of $\widetilde{\deg}$ and R it was proved in [BBGK18a] and [GJPW18a] that $\widetilde{\deg}$ and R respectively composes when the outer function is symmetric.

In terms of weaker notions of symmetry, there are various possible definitions. Here we consider the case of strongly-k-junta symmetric functions. The composition theorem for $\widetilde{\deg}$ when the outer function is strongly-k-junta symmetric.

A crucial result that we use in the proof of composition theorem of $\widetilde{\deg}$ is the following result from [Pat92].

Theorem 7.32 ([Pat92])**.** *For any non-constant symmetric function $f : \{0,1\}^n \to \{0,1\}$, let k be the closest integer to $n/2$ such that f takes different values on inputs of Hamming weight k and $k+1$. Define,*

$$\gamma(f) = \begin{cases} k & \text{if } k \leq n/2, \\ n-k & \text{otherwise.} \end{cases}$$

Then

$$\widetilde{\deg}(f) = \Theta\left(\sqrt{n(\gamma(f)+1)}\right).$$

Using the result of [Pat92] we prove the following proposition about the approximate degree of a k-junta symmetric function. Recall the multiplexer function from Definition 7.14.

Proposition 7.33: For any k-junta symmetric function $f : \{0,1\}^n \to \{0,1\}$, we have $\widetilde{\deg}(f) = \Omega\left(\sqrt{(n-k)\gamma_{\max}}\right)$ and $\widetilde{\deg}(f) = O\left(\max\{k, \sqrt{(n-k)\gamma_{\max}}\}\right)$, where $\gamma_{\max} = \max_{i \in \{0,1\}^k}\{\gamma(f_i)\}$ such that f_i is the symmetric function obtained by restricting the junta variables according to i.

Proof. Fixing the junta variables in f we obtain a symmetric function on $n-k$ variables with approximate degree $\Omega(\sqrt{(n-k)\gamma_{\max}})$ (Theorem 7.32), which in turn implies the same lower bound on $\widetilde{\deg}(f)$.

For the upper bound, we obtain an approximating polynomial for f by composing the (exact) polynomial for the multiplexer function $\mathsf{MUX} : \{0,1\}^{k+2^k} \to \{0,1\}$ with the approximating polynomials for different symmetric functions obtained by restricting the k junta variables. Therefore, $\widetilde{\deg}(f) = k + O(\sqrt{(n-k)\gamma_{\max}}) = O\left(\max\{k, \sqrt{(n-k)\gamma_{\max}}\}\right)$. □

As mentioned earlier, the composition of $\widetilde{\deg}$ when the outer function is symmetric was proved in [BBGK18a]. The following is their result that we crucially use in the proof of Theorem 1.12.

Theorem 7.34 ([BBGK18a]). *For any symmetric Boolean function $f : \{0,1\}^n \to \{0,1\}$ and any Boolean function $g : \{0,1\}^m \to \{0,1\}$ we have,*

$$\widetilde{\deg}(f \circ g) = \widetilde{\Omega}(\widetilde{\deg}(f) \cdot \widetilde{\deg}(g))$$

We now present the proof of Part (i) of Theorem 1.12, that the proof of composition of $\widetilde{\deg}$ when the outer function is strongly-k-junta symmetric.

Proof of Theorem 1.12(Part (i)). Since f is a strongly-k-junta symmetric function so there exists a setting of the k junta variables such that the resulting function is a non-constant symmetric function. Let f' be the symmetric function obtained by restricting the junta variables of f so that f' is non-constant. Then by Theorem 7.32 the approximate degree of f' is $\Omega(\sqrt{(n-k)\gamma_{\max}})$. Then clearly we have

$$\widetilde{\deg}(f \circ g) \geq \widetilde{\deg}(f' \circ g) = \widetilde{\Omega}(\widetilde{\deg}(f') \cdot \widetilde{\deg}(g)) = \widetilde{\Omega}(\sqrt{(n-k)\gamma_{\max}} \cdot \widetilde{\deg}(g)), \qquad (7.8)$$

where the first equality follows from Theorem 7.34. Now from Proposition 7.33 we know that $\widetilde{\deg}(f) = O(\sqrt{(n-k)\gamma_{\max}})$ if $k = O(\sqrt{(n-k)\gamma_{\max}})$, which is satisfied when $k = O(\sqrt{n})$.

Thus from (7.8) we obtain

$$\widetilde{\deg}(f \circ g) = \widetilde{\Omega}(\widetilde{\deg}(f) \cdot \widetilde{\deg}(g)).$$

□

7.7 Conclusion

While our work makes progress on the composition problem for $\widetilde{\deg}$, the main problem of whether $\widetilde{\deg}$ composes for any pair of Boolean functions remains open. In this light, we would like to highlight some questions that can be useful stepping stones towards the main questions.

In case of approximate degree composition, a natural question is whether $\sqrt{\mathrm{bs}(f)}$ can be replaced by some other complexity measures. In this regards we state the following open problems:

Question 7.35. *For all Boolean functions f and g, can we prove either of the following:*

- $\widetilde{\deg}(f \circ g) = \Omega(\sqrt{\deg(f)} \cdot \widetilde{\deg}(g))$*?*
- $\widetilde{\deg}(f \circ g) = \Omega(\sqrt{\mathsf{fbs}(f)} \cdot \widetilde{\deg}(g))$*?*

Recently, in [SYZ04, Sun07, Dru11, Cha11, CKP22], the classes of transitive functions got a lot of attention as a natural generalization of the classes of symmetric functions.

Question 7.36. *Can we prove that* $\widetilde{\deg}$ *and* R *compose when the outer function is transitive?*

Chapter 8

Approximate degree composition for recursive functions

8.1 Introduction

Representing Boolean functions in terms of polynomials has played a pivotal role in theoretical computer science. Various representation concepts, such as degree, approximate degree and sign degree, contribute to understanding the complexity of these functions. The minimum possible degree of a real polynomial representing a Boolean function, $f : \{0,1\}^n \to \{0,1\}$, is called the degree, denoted $\deg(f)$, of the function. For any $0 < \varepsilon < \frac{1}{2}$, the minimum degree of a real polynomial that ε approximates a Boolean function, $f : \{0,1\}^n \to \{0,1\}$, in ℓ_∞, is called the approximate degree of f and is denoted by $\widetilde{\deg}_\varepsilon(f)$. Usually, $\widetilde{\deg}(f)$ is shorthand for $\widetilde{\deg}_{1/3}(f)$.

Different values of ε introduce different representation concepts, each with distinct applications. For instance, as ε approaches $\frac{1}{2}$ we have the sign representation or threshold degree, denoted $\deg_\pm(f)$, of a Boolean function f. Threshold degree has strong connections to designing efficient learning algorithms (PAC learning [KS04, KOS04]). At the same time, the threshold degree lower bounds provide insights into circuit depths, formula sizes, communication complexity and many more areas [BVdW07, Che16]. For $0 < \varepsilon < 1/2$, called the approximate degree of f, $\widetilde{\deg}_\varepsilon(f)$, is a crucial quantity with close connections to quantum query complexity, communication complexity and other areas in theoretical computer science. More specifically, lower bounds on approximate degrees lead to lower bounds in quantum query complexity [BBC$^+$01b, AS04, Aar12], communication complexity [She09, She11a], and circuit complexity [All89]. On the other hand, upper bounds on approximate degree impact learning theory (Agnostic Learning [KKMS08], Attribute-Efficient Learning [KS06, STT12], etc.), approximate inclusion-exclusion [KLS96, She09], differentially private data release [TUV12, CTUW14], etc.

For large error, that is ε of the form $\frac{1-\delta^{-n}}{2}$ where δ is a constant less than 1, approximate degrees have an important role in understanding lower bounds of AC_0 circuits and also plays a vital role in attribute-efficient learning. Notably, large error representation and threshold representation are distinct from each other. For some functions, they may be the same (like OR) but there exist functions for which both representations are signifiantly different. For example, the 'ODD-MAX-BIT' function [Bei94] has constant threshold degree but approximate degree is $n^{\Omega(1)}$ for $\varepsilon = \frac{1-\exp(-n^{\Omega(1)})}{2}$. In this work, the large error representation of a function serves a crucial role in proving lower bounds, specifically for functions termed 'hardness amplifier functions' [BT22]. Before going into further details, first we will describe the problem.

A fundamental aspect in understanding complexity measures is investigating how they behave when two Boolean functions are combined to form a new function [BKT19, BDGKW20, GSS16, Tal13]. One particularly natural form of combination is *composition*: for any two Boolean functions $f : \{0,1\}^n \to \{0,1\}$ and $g : \{0,1\}^m \to \{0,1\}$, the composed function $f \circ g : \{0,1\}^{nm} \to \{0,1\}$ is defined by

$$f \circ g(x_1, \ldots, x_n) := f(g(x_1), \ldots, g(x_n)),$$

where $x_i \in \{0,1\}^m$ for $i \in [n]$. For the function $f \circ g$, the function f is called the outer function, and g is called the inner function. The composition has been employed commonly to create new functions showcasing improved separations (refer to [NS94, Tal13, Amb05, GSS16] for related results).

A big open question in this area is to understand how approximate degree ($\widetilde{\deg}$) behaves when we compose two functions. More specifically, it asks the following: for all Boolean functions $f : \{0,1\}^n \to \{0,1\}$ and $g : \{0,1\}^m \to \{0,1\}$, is

$$\widetilde{\deg}(f \circ g) = \widetilde{\Theta}(\widetilde{\deg}(f) \cdot \widetilde{\deg}(g))?$$

The tilde in the $\widetilde{\Theta}$ notation in the above question hides a polynomial factor of $\log(n+m)$. This problem is often referred to as the approximate degree composition problem.

The upper bound, $\widetilde{\deg}(f \circ g) = O(\widetilde{\deg}(f) \cdot \widetilde{\deg}(g))$ for any Boolean functions f, g, was proved in [She13e]. Thus, proving lower bounds on the approximate degree of the composed function in terms of the approximate degree of the individual functions remains the main problem; in other words, for all f and g, is $\widetilde{\deg}(f \circ g) = \widetilde{\Omega}(\widetilde{\deg}(f) \cdot \widetilde{\deg}(g))$? In this article approximate degree composition problem will refer to only this part of the actual approximate degree composition problem.

Numerous works, including those by [NS94, Amb05, She12, She13b, She13a, BT13, BBGK18b, CKM$^+$23b], actively pursued these lower bounds, leading to newer connections with several important problems in the field. However, establishing the lower bound $\widetilde{\deg}(f \circ g) = \widetilde{\Omega}\left(\widetilde{\deg}(f)\, \widetilde{\deg}(g)\right)$ even for specific functions or restricted classes of functions is often very challenging. For example, consider the composed function $\mathsf{OR} \circ \mathsf{AND}$, it took a long series of work [NS94, Shi02, Amb05, She13b, She13a, BT13] over nearly two decades to prove that $\widetilde{\deg}(\mathsf{OR} \circ \mathsf{AND}) = \Omega\left(\widetilde{\deg}(\mathsf{OR})\, \widetilde{\deg}(\mathsf{AND})\right)$. Till date we know that the approximate degree composes in the following cases:

- when the outer function f has full approximate degree, i.e., $\Theta(n)$ [She12],
- when the outer function f is a symmetric function [BBGK18b],
- when the outer function f has minimal approximate degree with respect to its block sensitivity, i.e., $\widetilde{\deg}(f) = O(\sqrt{\mathrm{bs}(f)})$ [CKM$^+$23b] (Chapter 7, Theorem 1.7), and
- when the sign degree of the inner function is same as its approximate degree [She12, Lee09].

This work focuses on the behavior of approximate degree when *recursive functions* are composed with other general functions (as outer or inner function). Here, by recursive functions we mean the functions of the kind h^d (h composed with itself d times) where the arity of h is small. The function h is often called the base function and the function f is called the a recursive h function.

Recursive functions are an important class of Boolean functions that are studied in various different contexts in the analysis of Boolean function, mainly in proving various lower bounds [Amb05, Sni85, SW86a, NS94, NW95, BHT17]. For example, the Kushilevitz's function [NW95] which is the only known non-trivial example of functions with low degree and high sensitivity is a recursive function of a carefully chosen base function. Recursive majority, Maj_3^d, is another recursive function that has been studied extensively in the literature for its different properties [SW86b, JKS03, Leo13, MNS$^+$16]. Boppana (see, e.g., [SW86b]) used it to provide the first evidence that the randomized query is more powerful than deterministic query [SW86b]. In the same article, they show a similar separation using recursive $\mathsf{AND}_2 \circ \mathsf{OR}_2$ function too. In a different application of recursive $\mathsf{AND}_2 \circ \mathsf{OR}_2$, [JRSW99] show separation between deterministic tree-size complexity and number of monomials in the minimal DNF or CNF.

The approximate degree composition was not known when the outer or inner function is a recursive function, in general. For some special recursive functions, however, it was known that the approximate degree composes. For example, the OR function on $n = 3^d$ bits is same as OR_3^d. After a series of works ([NS94, Amb05, She13a, BT13, She13b]), it was proven that the approximate degree composition holds when the outer function is OR, and in general symmetric [BBGK18b]. Similarly, from the result of [She12, Lee09] it can be observed that the lower bound holds when either the inner or outer function is recursive PARITY. Unfortunately, these results can't be applied in general even when the base function is symmetric or it has full approximate degree.

This scenario leads to the natural question:

Can we prove that $\widetilde{\deg}(f \circ g) = \Omega(\widetilde{\deg}(f) \cdot \widetilde{\deg}(g))$ when the outer function f or the inner function g is recursive?

8.1.1 Our Results

Let $h : \{0,1\}^k \to \{0,1\}$ be a function on k-bits. Let h^d denotes the complete k-ary tree of depth d such that each internal node of the tree is labelled by h and the leaves of tree are labelled by distinct variables. Our main result shows that the composition theorem holds for any h^d (except a few specific h's), either as the outer function with any inner function or as the inner function with any outer function.

Theorem 8.1 (Restatement of Theorem 1.10)**.** *Let $f\colon \{0,1\}^n \to \{0,1\}$ and $g\colon \{0,1\}^m \to \{0,1\}$ be two Boolean functions and $d = \Omega(\log\log n)$. Then,*

$$\widetilde{\deg}(f \circ g) = \Omega\left(\frac{\widetilde{\deg}(f)\widetilde{\deg}(g)}{\mathrm{polylog}(n)}\right),$$

if either of the following conditions hold:

1. *$f = h^d$, for any Boolean function h.*
2. *$g = h^d$, for any Boolean function h with constant arity and not equal to* AND *or* OR.

In light of the above theorem, understanding the composition of approximate degree when the inner function is OR is the central case for making progress towards the general composition question.

We would like to emphasize that there are not many results which prove composition theorem for a general class of inner functions. Theorem 1.10 shows that the composition property holds if the inner function is recursive irrespective of the outer function.

We further note that Theorem 1.10 doesn't follow from the known results even when the composition theorem is known to hold for the base function. Firstly, it is known that the composition lower bound holds when the outer function is symmetric [BBGK18b]; though, a repeated composition of a symmetric function will incur the factor of $(\log n)^d$ (because of the $\log n$ factor hiding in the $\widetilde{\Omega}$ notation). Secondly, while the majority function, Maj_n, has full approximate degree $(\Theta(n))$, Maj_3^d doesn't have full approximate degree. We use spectral sensitivity (check Definition 8.23) and it's perfect composition theorem to analyze the approximate degree of recursive majority. Thus, Sherstov's result [She12] that proves composition theorem holds for functions with full approximate degree cannot be applied in the case of recursive majority. The situation is similar for the inner function as well.

Moving ahead, the proof of Theorem 1.10 uses two ideas.

- We first prove that a similar theorem works for the specific case of $h = \mathrm{Maj}_3$ and $h = \mathsf{AND}_2 \circ \mathsf{OR}_2$ functions.
- Then, we use a general h to *simulate* $\mathsf{AND}_2 \circ \mathsf{OR}_2$; hence, proving composition for the general case.

The case of recursive $h = \mathrm{Maj}_3$ and $h = \mathsf{AND}_2 \circ \mathsf{OR}_2$ functions is in itself very interesting. There have been several works towards exploring the approximate degree and other properties of these two functions [GJ16, KV14, SW86b, JRSW99]. Given their importance, and the fact that it is a central step in our main result (Theorem 1.10), we state the composition theorem for these two functions separately.

Theorem 8.2 (Restatement of Theorem 1.9)**.** *Let f and g be two Boolean functions. Then,*

$$\widetilde{\deg}(f \circ h^d) = \widetilde{\Omega}(\widetilde{\deg}(f)\ \widetilde{\deg}(h^d)) \text{ and } \widetilde{\deg}(h^d \circ g) = \widetilde{\Omega}(\widetilde{\deg}(h^d)\ \widetilde{\deg}(g)),$$

where h is either $\mathrm{Maj}_3 : \{0,1\}^3 \to \{0,1\}$ *or* $\mathsf{AND}_2 \circ \mathsf{OR}_2 : \{0,1\}^4 \to \{0,1\}$*, n is the arity of the outer function, $d \geq C \log\log n$ for a large enough constant C, and $\widetilde{\Omega}(\cdot)$ hides* $\mathrm{polylog}(n)$ *factors.*

To prove Theorem 1.9 we will need the following lemma. Even though the lemma can be obtained from a combination of known results (e.g., [She12] and [BCH$^+$17]) with appropriate parameters, we give a self-contained simpler proof of the lemma, inspired by the primal-dual perspective of [She13a].

Lemma 8.3. *For any Boolean function $f : \{0,1\}^n \to \{0,1\}$ and $g : \{0,1\}^m \to \{0,1\}$,*

$$\widetilde{\deg}(f \circ \mathsf{MAJ}_t \circ g) = \Omega(\widetilde{\deg}(f)\widetilde{\deg}(g)) \tag{8.1}$$

for $t = \Omega(\log n)$.

Note that, Lemma 8.3 gives a way to settle the composition question affirmatively. In particular, if $\widetilde{\deg}(f \circ \mathrm{Maj}_t \circ g) = \widetilde{O}(\widetilde{\deg}(f \circ g))$, where t is $\Theta(\log n)$ and n is the arity of f, then it follows that the composition holds for f and g.

We also highlight that a tighter lower bound can be obtained when the middle function Maj is replaced by an “amplifier function" in Lemma 8.3. Define H to be a *strong hardness amplifier function* for g if

$$\widetilde{\deg}_{\frac{1-2^{-\Omega(t)}}{2}}(H \circ g) = \Omega(\widetilde{\deg}(H) \circ \widetilde{\deg}(g)).$$

In Lemma 8.29 we observe $\widetilde{\deg}(f \circ H \circ g) = \Omega(\widetilde{\deg}(f)\widetilde{\deg}(H)\widetilde{\deg}(g))$, when H is a strong hardness amplifier function for g. We discuss this improvement in Section 8.6.

8.1.2 Proof Ideas

To address the lower bound for the composition of two Boolean functions f and g, $f \circ g$, we will call f to be the ‘outer function’ and g to be the ‘inner function’. In the case of three-layer composed function $(f \circ H \circ g)$, we will call H to be the ‘hardness amplifier’ and f and g to be the outer and inner functions respectively.

Primal dual approach to composition:
Our proof technique is based on the primal-dual view used by [She13a] for proving the composition of $\mathsf{AND}_n \circ \mathsf{OR}_n$. Here, instead of using ‘dual-composition method’ (see [BT13, BT22]) we will be using only the dual witness of the inner function. The primal-dual approach is to construct an approximating polynomial for f with smaller degree than $\widetilde{\deg}(f)$ by applying a linear operator L on the assumed approximating polynomial for $f \circ g$ (say p, with smaller degree than claimed), leading to a contradiction. The linear operator L is defined by taking the input to f, extending it to a probability distribution (which depends upon the dual of g) over the inputs of $f \circ g$ and outputting the expectation.

Let ψ be the dual witness of g, we get μ_0 and μ_1 by restricting ψ on support which takes positive and negative values respectively; by the properties of dual witness, μ_1 (and μ_0) will mostly be supported on inputs x such that $g(x) = 1$ (and $g(x) = 0$ respectively). The input to f is expanded bit by bit using μ_0 and μ_1, creating a distribution on inputs of $f \circ g$.

Formally, L takes a general function $h : \{0,1\}^{mn} \to \{0,1\}$ and gives $Lh : \{0,1\}^n \to$ R.

$$Lh(z_1,\ldots,z_n) = \mathop{\mathbb{E}}_{x_1\sim\mu_{z_1}} \mathop{\mathbb{E}}_{x_2\sim\mu_{z_2}} \cdots \mathop{\mathbb{E}}_{x_n\sim\mu_{z_n}} [h(x_1,x_2,\ldots,x_n)], \tag{8.2}$$

where $x_i \in \{0,1\}^m$ for all $i \in \{1,2,\ldots,n\}$.

To complete the proof, the following two properties of L are required:

1. Showing that the polynomial Lp indeed approximates f in l_∞ norm. Intuitively this happens because the restricted distributions (μ_0 and μ_1) are a pretty good indicator of the value of g.
2. The degree of Lp is small, intuitively because L reduces the degree of every monomial by a factor of $\widetilde{\deg}(g)$.

Problem with the primal dual approach

Unfortunately, the recipe described above doesn't work well in general due to the error introduced by the expectation over μ_0 and μ_1 in the string $(z_1,\ldots,z_n)$. To handle a noisy string in place of a Boolean string, the approximating polynomial p needs to be robust. A polynomial is robust to noise $\frac{1}{3}$, if for all inputs x and for all $\Delta \in [-\frac{1}{3},\frac{1}{3}]^m$, $|p(x)-p(x+\Delta)| < \varepsilon$.

While any polynomial p can be made robust up to error ε with degree at most $\widetilde{\deg}(p) + \log(\frac{1}{\varepsilon})$ (see Theorem 8.10 by [She13e]), such polynomials are not known to be multilinear, making the analysis of expectation difficult. [BNRdW07b] gives a robust multilinear polynomial for any Boolean function $f : \{0,1\}^n \to \{0,1\}$; though, the polynomial is defined on a perturbation matrix of input x instead of x itself. We now discuss how to overcome this problem.

We give the proof ideas of Theorem 1.10, Theorem 1.9 and Lemma 8.3 in the reverse order, the way they are obtained from each other.

Proof idea of Lemma 8.3

We will use Maj_t to get past this difficulty; it helps to reduce the noise in the input of f to error $\frac{1}{n}$. Using the fact that any multilinear polynomial on n variables is robust up to error $\frac{1}{n}$, we have our lower bound for the function $\widetilde{\deg}(f \circ \mathrm{Maj}_t \circ g)$ where $t = \Omega(\log n)$.

Proof idea of Theorem 1.9
Using previously known constructions ([Val84, Gol20]), $\mathrm{Maj}_{\log n}$ can be projected to Maj_3^d and $(\mathsf{AND}_2 \circ \mathsf{OR}_2)^d$, where $d \geq C \log\log n$. We now replace $\mathrm{Maj}_{\log n}$ in Lemma 8.3 with these recursive functions; by using the associativity of the composition of functions and the approximate degree upper bound [She13e], we finish the proof of the theorem. Note that we only lose a factor of $\mathrm{polylog}(n)$ in the lower bound since we only need to simulate $\mathrm{Maj}_{\log n}$.

Now we give the idea about how to replace $\mathsf{AND}_2 \circ \mathsf{OR}_2$ with almost any recursive function to get our main result.

Proof idea of Theorem 1.10
Given Theorem 1.9, it is natural to ask, what other recursive functions satisfy the composition property. We show that almost any h can be used to replace the $\mathsf{AND}_2 \circ \mathsf{OR}_2$ function. This is done by simulating AND_2 and OR_2 using restrictions of h and its powers. The proof of this simulation is divided into two cases: monotone h and non-monotone h.

For the monotone case (except when h is AND or OR): We show that both AND_2 and OR_2 will be present as sub-cubes of the original Boolean hypercube of h.

For the non-monotone case (except when h is PARITY or $\neg\mathsf{PARITY}$): The proof requires more work here because of these two issues. First, there need not be both functions AND_2 and OR_2 as sub-cubes (though, we show that at least one will be present). Second, the sub-cube could be rotated. The resolution to both these issues is same. We use the non-monotonicity to construct the negation function. This allows us to rotate the sub-cube as well as construct $\mathsf{AND}_2/\mathsf{OR}_2$ from the other one.

A slight technical point to note is that when h is a non-constant arity function and h^d is the inner function, then the loss in the lower bound will be larger than $\mathrm{polylog}(n)$. However, even for the case when the base function h has arity that is a “slowly" growing function of n we still obtain a non-trivial lower bound composition result.

The remaining cases of Theorem 1.10, i.e., (i) when f or g equals h^d for $h \in \{\mathsf{PARITY}, \neg\mathsf{PARITY}\}$ follows from [She12], and (ii) when $f = h^d$ and $h \in \{\mathsf{AND}, \mathsf{OR}\}$ follows from [BBGK18b].

8.2 Notations and Preliminaries

In this paper, we will assume a Boolean function has domain $\{0,1\}^n$ and range $\{0,1\}$. We start with some of the important definitions.

Definition 8.4 (Recursive functions). *For any Boolean function $f : \{0,1\}^t \to \{0,1\}$ we define recursive function $f^d : \{0,1\}^{t^d} \to \{0,1\}$ by $f^d = \underbrace{f \circ f \circ \ldots \circ f}_{d \text{ times}}$.*

Definition 8.5 (Approximate degree ($\widetilde{\deg}$)). *For some constant $0 < \varepsilon < 1$, a polynomial $p : \mathbb{R}^n \to \mathbb{R}$ is said to ε-approximate a Boolean function $f : \{0,1\}^n \to \{0,1\}$ if $|p(x) - f(x)| \leq \varepsilon, \quad \forall x \in \{0,1\}^n$. The approximate degree of f, $\widetilde{\deg}_\varepsilon(f)$, is the minimum possible degree of a polynomial that ε-approximates f. Conventionally we use $\widetilde{\deg}(\cdot)$ as the shorthand for $\widetilde{\deg}_{1/3}.(\cdot)$*

Note that the constant ε in the above definitions can be replaced by any constant strictly smaller than $1/2$ which changes $\widetilde{\deg}_\varepsilon(f)$ by only a constant factor.

We also need the following facts about error reduction in approximating polynomials.

Lemma 8.6 (Error reduction [She11b]). *For any $\varepsilon > 0$, $\widetilde{\deg}_\varepsilon(f) = \Theta_\varepsilon(\widetilde{\deg}(f))$, where $\Theta_\varepsilon(\cdot)$ denotes that the constant depends on ε.*

Lemma 8.7 ([She11a, She12]). *Let $f : \{0,1\}^n \to \mathbb{R}$ be a function and $\varepsilon > 0$. Then, $\widetilde{\deg}_\varepsilon(f) \geq d$ iff there exists a function $\psi : \{0,1\}^n \to \mathbb{R}$ such that*

$$\sum_{x \in \{0,1\}^n} |\psi(x)| = 1, \tag{8.3}$$

$$\sum_{x \in \{0,1\}^n} \psi(x) \cdot f(x) > \varepsilon, \tag{8.4}$$

$$\sum_{x \in \{0,1\}^n} \psi(x) \cdot p(x) = 0 \tag{8.5}$$

for every polynomial p of degree $< d$.

In a seminal work, Sherstov [She13e] showed that approximate degree can increase at most multiplicatively under composition.

Theorem 8.8 ([She13e]). *For all Boolean function $f : \{0,1\}^n \to \{0,1\}$ and $g : \{0,1\}^m \to \{0,1\}$, $\widetilde{\deg}(f \circ g) = O(\widetilde{\deg}(f) \cdot \widetilde{\deg}(g))$.*

We will be working with inputs that are not Boolean *but* are close to Boolean. We need the notion of *robust* approximating polynomials.

Definition 8.9 ((δ,ε)-robust approximating polynomial)**.** *Let $p:\{0,1\}^m \to \{0,1\}$ be a polynomial. Then, for $\delta,\varepsilon>0$, a (δ,ε)-robust approximating polynomial for p is a polynomial $p_{robust}:\mathbb{R}^m \to \mathbb{R}$ such that for all $x\in\{0,1\}^m$ and for all $\Delta\in[-\delta,\delta]^m$,*

$$|p(x)-p_{robust}(x+\Delta)|<\varepsilon.$$

Note that robust polynomial need not to be multilinear. [She13e] proved that for any Boolean function $f:\{0,1\}^n\to\{0,1\}$ there exists a robust polynomial with degree at most $(\deg(p)+\log(1/\varepsilon))$.

Theorem 8.10 (Sherstov [She13e])**.** *A (δ,ε)-robust approximating polynomial for p of degree $O_\delta(\deg(p)+\log(1/\varepsilon))$ exists. Here $O_\delta(\cdot)$ denotes that the constant in $O(\cdot)$ depends on δ.*

For our purposes, we need a multilinear robust approximating polynomial.

Theorem 8.11 (Folklore)**.** *Any multilinear polynomial $p:\{0,1\}^n\to\{0,1\}$ is $(\frac{\delta}{n},\delta)$-robust.*

We also need the following theorems about computing Maj_n using recursive functions.

Theorem 8.12 ([Gol20])**.** *There exists a constant $C>0$, such that $\mathrm{Maj}_n\colon\{0,1\}^n\to\{0,1\}$ is a projection of Maj_3^d where $d=C\log n$.*

Theorem 8.13 ([Val84])**.** *There exists a constant $C>0$, such that $\mathrm{Maj}_n\colon\{0,1\}^n\to\{0,1\}$ is a projection of $(\mathsf{AND}_2\circ\mathsf{OR}_2)^d$ where $d=C\log n$.*

8.3 Composition theorem for recursive Majority and alternating AND-OR trees

In this section we give a proof of Theorem 1.9. We being with a proof highlight of Lemma 8.3.

8.3.1 Proof of Lemma 8.3

Lemma 8.14. *For any Boolean function $f:\{0,1\}^n\to\{0,1\}$ and $g:\{0,1\}^m\to\{0,1\}$,*

$$\widetilde{\deg}(f\circ\mathsf{MAJ}_t\circ g)=\Omega(\widetilde{\deg}(f)\widetilde{\deg}(g)) \quad (8.1)$$

for $t=\Omega(\log n)$.

Proof. We will present a proof inspired by the primal-dual view of [She13a]. Fix any constant $0 < \varepsilon < 1/2$. Let $h := f \circ \mathrm{Maj}_t \circ g$ be the composed function, and $p_h : \{0,1\}^{ntm} \to \mathbb{R}$ be an ε-approximating polynomial for h.

Further, define $d := \widetilde{\deg}_{\frac{1-\varepsilon}{2}}(g)$. Then, by Lemma 8.7, there exists a function $\psi : \{0,1\}^m \to \mathbb{R}$ such that

$$\sum_{x \in \{0,1\}^m} |\psi(x)| = 1, \tag{8.6}$$

$$\sum_{x \in \{0,1\}^m} \psi(x) \cdot g(x) > \frac{1-\varepsilon}{2}, \tag{8.7}$$

and

$$\sum_{x \in \{0,1\}^m} \psi(x) \cdot p(x) = 0 \tag{8.8}$$

for every polynomial p of degree $< d$.

Let μ be the probability distribution on $\{0,1\}^m$ given by $\mu(x) = |\psi(x)|$ for $x \in \{0,1\}^m$. From (8.8), we have $\sum_{x \in \{0,1\}^m} \psi(x) = 0$. Therefore, the sets $\{x \mid \psi(x) < 0\}$ and $\{x \mid \psi(x) > 0\}$ are weighted equally by μ. Let μ_0 and μ_1 be the probability distributions obtained by conditioning μ on the sets $\{x \mid \psi(x) < 0\}$ and $\{x \mid \psi(x) > 0\}$ respectively. Hence,

$$\mu = \frac{1}{2}\mu_0 + \frac{1}{2}\mu_1, \quad \text{and} \quad \psi = \frac{1}{2}\mu_1 - \frac{1}{2}\mu_0.$$

We note an important property of the distributions μ_0 and μ_1 which shows that the error between $\mathsf{sign}(\psi(x))$ and $g(x)$ is low.

Lemma 8.15. $\mathbb{E}_{x \sim \mu_1}[g(x)] > 1 - \varepsilon$.

Lemma 8.16. $\mathbb{E}_{x \sim \mu_0}[g(x)] < \varepsilon$.

Consider the following linear operator L that maps functions $h : \{0,1\}^{ntm} \to \mathbb{R}$ to functions $Lh : \{0,1\}^n \to \mathbb{R}$,

$$Lh(z) = \mathop{\mathbb{E}}_{\substack{x_{11} \sim \mu_{z_1} \\ x_{12} \sim \mu_{z_1} \\ \vdots \\ x_{1t} \sim \mu_{z_1}}} \mathop{\mathbb{E}}_{\substack{x_{21} \sim \mu_{z_2} \\ x_{22} \sim \mu_{z_2} \\ \vdots \\ x_{2t} \sim \mu_{z_2}}} \cdots \mathop{\mathbb{E}}_{\substack{x_{n1} \sim \mu_{z_n} \\ x_{n2} \sim \mu_{z_n} \\ \vdots \\ x_{nt} \sim \mu_{z_n}}} [h(x_{11}, \ldots, x_{1t}, x_{21}, \ldots, x_{2t}, \ldots, x_{n1}, \ldots, x_{nt})]. \tag{8.9}$$

Recall $h = f \circ \mathrm{Maj}_t \circ g$ and p_h be ε-approximating polynomial for h. Thus by linearity of L we have $\|L(h - p_h)\|_\infty \leq \varepsilon$. We will now observe some useful properties of the linear operator L.

Lemma 8.17. *$\deg(Lp_h) \leq \deg(p_h)/d$, where $d = \widetilde{\deg}_{\frac{1-\varepsilon}{2}}(g)$.*

We now show that Lp_h is in fact an approximating polynomial for f.

Lemma 8.18. *Fix $0 < \delta < 1/2$. Recall p_h is an ε-approximating polynomial for $h = f \circ \mathrm{Maj}_t \circ g$. Let $t = \Theta(\log n + \log(1/\delta))$ where the constant in $\Theta(\cdot)$ depends on ε. Then, Lp_h is a $(\delta + \varepsilon)$-approximating polynomial for f. That is,*

$$\|f - Lp_h\|_\infty \leq \|f - Lh\|_\infty + \|Lh - Lp_h\|_\infty \leq \delta + \varepsilon.$$

Proof. It suffices to show $\|f - Lh\|_\infty \leq \delta$. To this end, consider $Lh(z)$.

$$\begin{aligned} Lh(z) &= \mathop{\mathbb{E}}_{\substack{x_{11}\sim\mu_{z_1} \\ x_{12}\sim\mu_{z_1} \\ \vdots \\ x_{1t}\sim\mu_{z_1}}} \mathop{\mathbb{E}}_{\substack{x_{21}\sim\mu_{z_2} \\ x_{22}\sim\mu_{z_2} \\ \vdots \\ x_{2t}\sim\mu_{z_2}}} \cdots \mathop{\mathbb{E}}_{\substack{x_{n1}\sim\mu_{z_n} \\ x_{n2}\sim\mu_{z_n} \\ \vdots \\ x_{nt}\sim\mu_{z_n}}} [f \circ \mathrm{Maj}_t \circ g(x_{11}, \ldots, x_{1t}, x_{21}, \ldots, x_{2t}, \ldots, x_{n1}, \ldots, x_{nt})] \\ &= f\left(\mathrm{Maj}_t\left(\mathop{\mathbb{E}}_{\mu_{z_1}}[g], \ldots, \mathop{\mathbb{E}}_{\mu_{z_1}}[g]\right), \mathrm{Maj}_t\left(\mathop{\mathbb{E}}_{\mu_{z_2}}[g], \ldots, \mathop{\mathbb{E}}_{\mu_{z_2}}[g]\right), \ldots, \mathrm{Maj}_t\left(\mathop{\mathbb{E}}_{\mu_{z_n}}[g], \ldots, \mathop{\mathbb{E}}_{\mu_{z_n}}[g]\right)\right) \\ &= f(z'_1, z'_2, \ldots, z'_n), \end{aligned}$$

where $\|z - z'\|_\infty \leq \delta/n$ because $t = \Theta(\log n + \log(1/\delta))$ and Lemmas 8.16 and 8.15.

Therefore, for any $z \in \{0,1\}^n$, $|f(z) - Lh(z)| = |f(z) - f(z')| \leq \delta$, since $\|z - z'\|_\infty \leq \delta/n$ and Lemma 8.11. $\square$

Since Lp_h is a $(\delta + \varepsilon)$-approximating polynomial for f, we also have $\deg(Lp_h) \geq \widetilde{\deg}_{\delta+\varepsilon}(f)$. We therefore have the following inequalities

$$\widetilde{\deg}_{\delta+\varepsilon}(f) \leq \deg(Lp_h) \leq \frac{\deg(p_h)}{\widetilde{\deg}_{\frac{1-\varepsilon}{2}}(g)}.$$

Rewriting we have

$$\widetilde{\deg}_\varepsilon(f \circ \mathrm{Maj}_t \circ g) = \deg(p_h) \geq \widetilde{\deg}_{\delta+\varepsilon}(f) \cdot \widetilde{\deg}_{\frac{1-\varepsilon}{2}}(g). \tag{8.10}$$

This completes the proof of Lemma 8.3. $\square$

8.3.2 Proof of Theorem 1.9

Let $f : \{0,1\}^n \to \mathbb{R}$ and $g \colon \{0,1\}^m \to \mathbb{R}$ be two functions. We say that f is a *projection* of g, denoted $f \leq_{\text{proj}} g$, iff

$$f(x_1,\dots,x_n) = g(a_1,\dots,a_m)$$

for some $a_i \in \{0,1\} \cup \{x_1,x_2,\dots,x_n\}$. That is, f is obtained from g by substitutions of variables by variables of f or constants in $\{0,1\}$. We note an easy to observe fact about approximate-degree of projections of functions.

Fact 8.19. *Let $f \colon \{0,1\}^n \to \{0,1\}$ and $g \colon \{0,1\}^m \to \{0,1\}$ be such that $f \leq_{proj} g$, i.e., f is a projection of g. Then, $\widetilde{\deg}(f) \leq \widetilde{\deg}(g)$.*

Consider the recursive-majority function Maj_3^d given by the complete 3-ary tree of height d with internal nodes labeled by Maj_3 and the leaves are labeled by distinct variables. Fix $d \geq C \log\log n$ for a large enough constant C.

Firstly observe that Maj_3^d is *not* a symmetric function. Secondly, it also doesn't have *full* approximate degree ([RS12]). Thirdly, and finally, its approximate degree is also *not* equal to $\Theta\left(\sqrt{\mathrm{bs}(\mathrm{Maj}_3^d)}\right)$ (see Lemma 8.26, it follows from the fact that $\mathrm{bs}(\mathrm{Maj}_3^d)$ is linear with $\widetilde{\deg}(\mathrm{Maj}_3^d)$). Thus, none of the previous works [She12, BBGK18b, CKM$^+$23b] imply that approximate degree composes when one of the (inner or outer) functions is recursive-majority Maj_3^d.

Proof of Theorem 1.9. Let Maj_3^d be the recursive-majority function obtained by the complete 3-ary tree of height d with internal nodes labeled by Maj_3 and the leaves are labeled by distinct variables. Let $f \colon \{0,1\}^n \to \{0,1\}$ be an arbitrary function and consider the approximate degree of the composed function $f \circ \mathrm{Maj}_t \circ \mathrm{Maj}_3^d$ where $t = \Theta(\log n)$.

$$\begin{aligned}
\widetilde{\deg}(f \circ \mathrm{Maj}_t \circ \mathrm{Maj}_3^d) &\leq \widetilde{\deg}(f \circ \mathrm{Maj}_3^{C\log t} \circ \mathrm{Maj}_3^d) = \widetilde{\deg}(f \circ \mathrm{Maj}_3^d \circ \mathrm{Maj}_3^{C\log t}) && (8.11)\\
&= O(\widetilde{\deg}(f \circ \mathrm{Maj}_3^d) \cdot \widetilde{\deg}(\mathrm{Maj}_3^{C\log t})) && (8.12)\\
&= O(\widetilde{\deg}(f \circ \mathrm{Maj}_3^d) \cdot \mathrm{poly}(t)). && (8.13)
\end{aligned}$$

The first inequality in (8.11) follows from the fact that Maj_t is a projection of $\mathrm{Maj}_3^{C\log t}$ (Theorem 8.12) and Fact 8.19. Then (8.12) follows from Theorem 8.8.

On the other hand, from Lemma 8.3, for $t = \Omega(\log n)$ we have

$$\widetilde{\deg}(f \circ \mathrm{Maj}_t \circ \mathrm{Maj}_3^d) = \Omega(\widetilde{\deg}(f) \cdot \widetilde{\deg}(\mathrm{Maj}_3^d)).$$

Combining with (8.13), we obtain the lower bound

$$\widetilde{\deg}(f \circ \mathrm{Maj}_3^d) = \Omega\left(\frac{\widetilde{\deg}(f) \cdot \widetilde{\deg}(\mathrm{Maj}_3^d)}{\mathrm{polylog}(n)}\right).$$

A similar argument shows the following inequalities, where in the last two inequalities we use Theorem 8.13 instead of Theorem 8.12, for $d = \Omega(\log n)$,

- $\widetilde{\deg}(\mathrm{Maj}_3^d \circ f) = \widetilde{\Omega}(\widetilde{\deg}(f) \cdot \widetilde{\deg}(\mathrm{Maj}_3^d))$,
- $\widetilde{\deg}(f \circ (\mathsf{AND}_2 \circ \mathsf{OR}_2)^d) = \widetilde{\Omega}(\widetilde{\deg}(f) \cdot \widetilde{\deg}((\mathsf{AND}_2 \circ \mathsf{OR}_2)^d))$, and
- $\widetilde{\deg}((\mathsf{AND}_2 \circ \mathsf{OR}_2)^d \circ f) = \widetilde{\Omega}(\widetilde{\deg}(f) \cdot \widetilde{\deg}((\mathsf{AND}_2 \circ \mathsf{OR}_2)^d))$.

$\square$

8.4 Composition theorem for recursive functions

In this section we prove our main theorem (Theorem 1.10). It shows that the approximate degree composes when either the inner function or the outer function is a recursive function. More formally,

The following cases of Theorem 1.10 follows from prior works:

1. f or g equals h^d for $h \in \{\mathsf{PARITY}, \neg\mathsf{PARITY}\}$ [She12].
2. $f = h^d$ and $h \in \{\mathsf{AND}, \mathsf{OR}\}$ [BBGK18b].

Therefore, it remains to prove Theorem 1.10 when $h \notin \{\mathsf{PARITY}, \neg\mathsf{PARITY}, \mathsf{AND}, \mathsf{OR}\}$. A crucial technical insight that makes the proof work is that when $h \notin \{\mathsf{PARITY}, \neg\mathsf{PARITY}, \mathsf{AND}, \mathsf{OR}\}$ then AND_2 and OR_2 are projections of h^3. We can thus simulate Maj using a small power of h. Thereafter, Lemma 8.3 is used to conclude Theorem 1.10. We now work out the details. We first state the main technical lemma we need for Theorem 1.10 and then complete the proof of the theorem. Finally, we prove the technical lemma in Section 8.4.1.

Lemma 8.20. *Let $h : \{0,1\}^t \to \{0,1\}$ (where $t \geq 2$) be a Boolean function which depends on all t variables and is not equal to* $\mathsf{PARITY}/\mathsf{OR}/\mathsf{AND}$. *The function* AND_2 *(and similarly* OR_2*) can be obtained by setting all but two variables to constants in h^k for $k \leq 3$.*

We now present the proof of Theorem 1.10 using Lemma 8.20.

Proof of Theorem 1.10. Let $h\colon \{0,1\}^t \to \{0,1\}$ be such that $h \notin \{\mathsf{PARITY}, \neg\mathsf{PARITY}, \mathsf{AND}, \mathsf{OR}\}$. We know from Lemma 8.3 that $\widetilde{\deg}(f \circ \mathrm{Maj}_k \circ h^d) = \Omega(\widetilde{\deg}(f)\widetilde{\deg}(h^d))$ where $k = \Theta(\log n)$.

Like in the proof of Theorem 1.9, we will simulate Maj_k using h^ℓ for sufficiently large ℓ. From Lemma 8.20, it follows that $(\mathsf{AND}_2 \circ \mathsf{OR}_2)^\ell$ is a projection of $h^{6\ell}$. Therefore, we obtain from Theorem 8.13 that Maj_k is a projection of $h^{C\log k}$ for some constant $C > 0$. We thus have the following sequence of inequalities,

$$\begin{aligned}\widetilde{\deg}(f \circ h^d) \geq \widetilde{\deg}(f \circ \mathrm{Maj}_k \circ h^{(d-C\log k)}) &= \Omega(\widetilde{\deg}(f)\widetilde{\deg}(h^{(d-C\log k)})) \\ &= \Omega\left(\frac{\widetilde{\deg}(f)\widetilde{\deg}(h^d)}{t^{C\log k}}\right) = \Omega\left(\frac{\widetilde{\deg}(f)\widetilde{\deg}(h^d)}{\mathrm{polylog}(n)}\right).\end{aligned}$$

Note that the last equality above uses the fact that t is a constant. When h^d is the outer function then we don't need t to be a constant, while the rest of the argument remains the same to give

$$\widetilde{\deg}(h^d \circ g) = \Omega\left(\frac{\widetilde{\deg}(h^d)\widetilde{\deg}(g)}{\mathrm{polylog}(n)}\right).$$

□

This completes the proof of the main theorem. We now present a proof of Lemma 8.20.

8.4.1 Proof of the main technical lemma (Lemma 8.20)

We proceed by proving an intermediate result (Lemma 8.21) before going to the proof of Lemma 8.20.

Suppose we are allowed to *modify* a Boolean function by two operations: negating some of its variables, and restricting some of the variables to constant values. Lemma 8.21 proves that almost every Boolean function can be modified to either an AND_2 or an OR_2 function. A restriction of the variables amounts to looking at a smaller hypercube translated to a new point, and negating a variable amounts to rotating the smaller hypercube. In other words, we want to show that there is a *shifted* AND_2 or OR_2 in the Boolean hypercube of h (see Figure 8.1 for an example).

This shifted $\mathsf{AND}_2/\mathsf{OR}_2$ in the Boolean hypercube of a Boolean function can be concretely defined by the concept of a sensitive block. For a block of variables $S \subseteq [n]$ and an input $x \in \{0,1\}^n$, define $x^{\oplus S} \in \{0,1\}^n$ to be the input which flips exactly the variables in S at the input x. Given a Boolean function $f : \{0,1\}^n \to \{0,1\}$, a block S is called sensitive on x iff $f(x) \neq f(x^{\oplus S})$. A block S is called *minimal sensitive* for x at f, if no subset of S is sensitive for x at f.

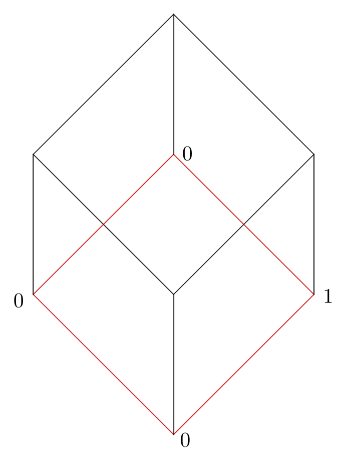

Fig. 8.1 A function on 3 bits with a shifted OR marked with red edges.

Notice that a shifted $\mathsf{AND}_2/\mathsf{OR}_2$ is a square with three vertices labelled 0 and one vertex labelled 1 or vice versa. This gives us a minimal sensitive block on the vertex opposite to the unique value. It can be easily verified that the converse is also true. So, we define a function to have a shifted $\mathsf{AND}_2/\mathsf{OR}_2$ iff it has a minimal sensitive block of size 2.

We show below that almost all functions have a minimal sensitive block of size 2.

Lemma 8.21. *Let $h : \{0,1\}^t \to \{0,1\}$ (where $t \geq 2$) be a Boolean function which depends on all t variables and is not equal to* PARITY. *Then, there exists an $x \in \{0,1\}^t$ such that h has a minimal sensitive block of size 2 on x.*

Proof. We will prove the result using induction on the variables. The statement can be easily verified for $t = 2$.

Define g_0 (and g_1) to be the restrictions of h by setting $x_t = 0$ (and $x_t = 1$) respectively. Let e_y be the edge $((y,0),(y,1))$ in the Boolean hypercube, and $S_t := \{e_y : y \in \{0,1\}^{t-1}\}$. Color an edge e_y red if $g_0(y) = g_1(y)$, and blue otherwise.

Notice that not all the edges in S_t can be red, otherwise h does not depend on x_t. Suppose all the edges in S_t are blue, i.e, $g_1 = \neg g_0$ (in other words, $h = g_0 \oplus x_t$). Since h depends on all variables, then g_0 depends on all variables $x_1, x_2, \cdots, x_{t-1}$. If g_0 is PARITY, then h is also PARITY. Implying that g_0 is dependent on all its variables and is not PARITY. By induction, there exists a minimal sensitive block of size 2 for g_0 (and hence h).

For the rest of the proof, we can assume that there exists both a red and a blue edge in S_t.

Let e_x be red and e_y be blue, this means that $g_0(x) = g_1(x)$ but $g_0(y) \neq g_1(y)$. If x and y were at Hamming distance 1, then vertices $(x,0),(x,1),(y,0)$ and $(y,1)$ will give us the required minimal sensitive block of size 2.

If x,y are not at Hamming distance 1, look at any path from x to y in the $t-1$ dimensional hypercube, say $z_0 = x, z_1, z_2, \cdots, z_l = y$. The edge e_{z_0} is red and e_{z_l} is blue. Since the color needs to switch at some point, there exist z_i, z_{i+1} at Hamming distance 1 such that e_{z_i} is red and $e_{z_{i+1}}$ is blue. Again, the vertices $(z_i,0),(z_i,1),(z_{1+1},0)$ and $(z_{i+1},1)$ will give us the required minimal sensitive block of size 2.

□

We are prepared to prove Lemma 8.20 which shows: given a Boolean function h, AND_2 (and OR_2) can be obtained by restricting some of the variables to constants in a very small power of h. Compared to Lemma 8.21, we need to remove negation and simulate both AND_2 and OR_2 and not just one of them.

We just show how to obtain AND_2, the case for OR_2 is similar. We handle the case of h being monotone and non-monotone separately.

Monotone h: This case is simpler, and AND_2 can be obtained as a restriction of h itself. Let a minimal 1-input be a $x \in \{0,1\}^t$ such that setting any 1 bit of x to 0 changes the value of h. If there is a minimal 1-input x of Hamming weight more than 2, we get a AND_2 by choosing any two indices which are 1 in x. The following claim finishes the proof for monotone functions.

Claim 8.22. *Let $h : \{0,1\}^t \to \{0,1\}$ be a monotone Boolean function which depends on all variables. If there is no minimal 1-input with Hamming weight more than 2, then h is the* OR *function.*

Proof. By abusing the notation, let 0 denote the all 0 input. Since the function is monotone but not constant, we know that $h(0) = 0$. Let $S \subseteq [t]$ capture the indices such that the corresponding Hamming weight 1-input has function value 0,

$$S = \{i : h(0^{\oplus i}) = 0\}.$$

For a $y \in \{0,1\}^t$, if the set of 1-indices are not a subset of S, then $h(y) = 1$ by monotonicity. If the set of 1-indices are a subset of S, then $h(y) = 0$ because there is no minimal 1-input with Hamming weight more than 2.

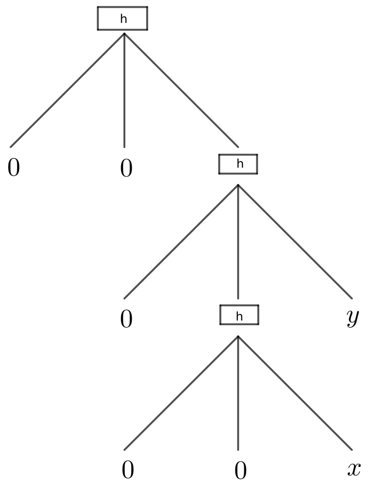

Fig. 8.2 An example for constructing AND_2 using a non-monotone function. Let $\mathsf{h} : \{0,1\}^3 \to \{0,1\}$ be 0 at $x = 001$ and 1 otherwise. We use the shifted OR_2/minimal sensitive block at 001 with indices $\{2,3\}$.

In other words, h is the OR function on the remaining $[t] - S$ variables. Since h depends on all the t variables, h is the OR function.

□

Non-monotone h: Since h is a non monotone function, there exists an input $a \in \{0,1\}^t$ and an index $i \in [t]$ such that $h(a) = 1$, $a_i = 0$ and $h(a^{\oplus i}) = 0$. Restricting the variables according to a (except the i-th bit) gives $h_1(x_i) = \neg x_i$.

From Lemma 8.21, there exists a $b \in \{0,1\}^t$ such that h has a minimal sensitive block of size 2 on b (shifted AND_2/OR_2). The main idea of this proof is to use negation and this shifted AND_2/OR_2 (Figure 8.2 gives an example).

For the formal proof, without loss of generality assume that the block have indices $1,2$ (that means $h(b) = h(b^{\oplus\{1\}}) = h(b^{\oplus\{2\}}) \neq h(b^{\oplus\{1,2\}})$). We will finish the proof by considering the two cases $h(b) = 0$ and $h(b) = 1$.

- $h(b) = 0$ (shifted AND_2): Suppose $b_1 = 0$ and $b_2 = 1$ (other cases can be handled similarly). Notice that $\mathsf{AND}_2(x,y) = h(x,\neg y,b_3,\cdots,b_t)$, giving us $\mathsf{AND}_2(x,y) = h(x,h_1(y),b_3,\cdots,b_t)$.
- $h(b) = 1$ (shifted OR_2): Suppose $b_1 = 1$ and $b_2 = 0$ (other cases can be handled similarly). Notice that $\mathsf{OR}_2(x,y) = h(x,\neg y,b_3,\cdots,b_t)$; using De Morgan's law,

$$\mathsf{AND}_2(x,y) = \neg\mathsf{OR}_2(\neg x,\neg y) = \neg h(\neg x,y,b_3,\cdots,b_t) = h_1(h(h_1(x),y,b_3,\cdots,b_t))$$

Since h_1 is also a restriction of h, the proof is complete.

8.5 Approximate Degree upper and lower bound on Recursive Majority and AND-OR tree

Spectral sensitivity is a nice complexity measure that gives lower bound on $\widetilde{\deg}$. It was used to prove the celebrated conjecture 'sensitivity conjecture' by [Hua19]. First we will define spectral sensitivity and use it to prove approximate degree lower bound for Maj_3^d and $(\mathsf{AND}_2 \circ \mathsf{OR}_2)^d$.

We follow the definition from [ABK$^+$21a] and also state a result from [ABK$^+$21a] where it was proved that spectral sensitivity lower bounds approximate degree of a function.

Definition 8.23 (Spectral Sensitivity). *Let $f : \{0,1\}^n \to \{0,1\}$ be a Boolean function. The sensitivity graph of f, $G_f = (V,E)$ is a subgraph of the Boolean hypercube, where $V = 0,1^n$, and $E = \{(x \oplus e_i) \in V \times V : i \in [n], f(x) \neq f(x \oplus e_i)\}$. That is, E is the set of edges between neighbors on the hypercube that have different f-value. Let A_f be the adjacency matrix of the graph G_f . We define the spectral sensitivity of f as $\lambda(f) = \|A_f\|$.*

It is well-known that spectral sensitivity is a nice quantity that composes exactly.

Theorem 8.24 ([ABK$^+$21a]). *For all Boolean function f, $\lambda(f) = O(\widetilde{\deg}(f))$.*

Lemma 8.25. *$\lambda(\mathrm{Maj}_3^d) = \Theta(2^d)$ where d is the depth of the recursion.*

Proof. First, we will prove spectral sensitivity of Maj_3. Here is the adjacency matrix of the sensitivity graph of Maj_3.

$$\mathscr{S} = \left(\begin{array}{ccccccccc} & (000) & (100) & (010) & (001) & (110) & (101) & (011) & (111) \\ (000) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ (100) & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ (010) & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ (001) & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ (110) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ (101) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ (011) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ (111) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right) \tag{8.14}$$

Since the adjacency matrix is symmetric, the largest eigenvalue of the matrix is also the norm of the matrix. Note that calculating the eigenvalue of the following 3×3 matrix is sufficient for our purpose:

$$\mathscr{A} = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} \tag{8.15}$$

Eigen value of A is 2, consequently $\|\mathscr{A}\| = 2$. Since spectral sensitivity (λ) composes exactly (without any constant overhead) we have $\lambda(\mathrm{Maj}_3^d) = 2^d$. $\square$

Corollary 8.26. $\widetilde{\deg}(\mathrm{Maj}_3^d) = \Theta(2^d)$ *where d is the depth of the recursion.*

Proof. From Lemma 8.25 it follows that $\widetilde{\deg}(\mathrm{Maj}_3^d) = \Omega(2^d)$ since $\widetilde{\deg} \leq \lambda$ for any Boolean function. Also it is known that bounded error quantum query complexity gives upper bound on $\widetilde{\deg}$. [RS12] showed that bounded error quantum query complexity of Maj_3^d is $O(2^d)$. Hence, $\widetilde{\deg}(\mathrm{Maj}_3^d) = \Theta(2^d)$. $\square$

Lemma 8.27. $\lambda(\mathsf{AND}_2 \circ \mathsf{OR}_2)^d = 2^d$.

Proof. Similar to the Maj_3^d proof for calculating the largest eigenvalue of the adjacency matrix of AND_2 the following 3×3 matrix is sufficient for our purpose:

$$\mathscr{A} = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix} \tag{8.16}$$

Eigen value of A is $\sqrt{2}$, consequently $\|\mathscr{A}\| = \sqrt{2}$. Since spectral sensitivity (λ) composes exactly (without any constant overhead) we have $\lambda(\mathsf{AND}_2 \circ \mathsf{OR}_2) = 2$ and $\lambda(\mathsf{AND}_2 \circ \mathsf{OR}_2)^d = 2^d$. $\square$

Approximate degree lower bound of $\mathsf{AND}_2 \circ \mathsf{OR}_2{}^d$ follows from [ABK+21b], for completeness we are stating here.

Corollary 8.28. *$\widetilde{\deg}(\mathsf{AND}_2 \circ \mathsf{OR}_2{}^d) = \Theta(2^d)$ where d is the depth of the recursion.*

Proof. From Lemma 8.27 it follows that $\widetilde{\deg}(\mathsf{AND}_2 \circ \mathsf{OR}_2{}^d) = \Omega(2^d)$ since $\widetilde{\deg} \leq \lambda$ for any Boolean function. It follows from [Rei11b] and from [KV14] that the upper bound on approximate degree of $(\mathsf{AND} \circ \mathsf{OR}^d)$ tree is $O(2^d)$. Hence, $\widetilde{\deg}((\mathsf{AND} \circ \mathsf{OR}^d)) = \Theta(2^d)$. $\square$

8.6 Composition theorem for recursive functions with full sign degree

The aim of this section is to show that any Boolean function with full sign degree can be used as a hardness amplifier, similar to Maj_t in Lemma 8.3.

Lemma 8.29. *For any Boolean function $f : \{0,1\}^n \to \{0,1\}$ and $g : \{0,1\}^m \to \{0,1\}$ we have,*

$$\widetilde{\deg}(f \circ \mathsf{Amp}_t \circ g) = \Omega(\widetilde{\deg}(f) \cdot \widetilde{\deg}(\mathsf{Amp}_t) \cdot \widetilde{\deg}(g))$$

where Amp_t is any Boolean function on t bits with full sign degree.

In this section, we will prove the composition theorem for a few more recursive functions. To prove our theorem we will be using Lemma 8.29 which is a generalization of Lemma 8.3. For the sake of completeness, we also present the proof of Lemma 8.29. [Sak93] and [Ant95] shows that,

Theorem 8.30. *([Sak93] and [Ant95]) Almost all the function $f : \{0,1\}^n \to \{0,1\}$ sign degree is high, $\widetilde{\deg}_{\pm}(f) = \Omega(n)$.*

We will be proving the following results:

Theorem 8.31. *For any Boolean function $f : \{0,1\}^n \to \{0,1\}$ and $g : \{0,1\}^n \to \{0,1\}$ the following holds:*

$\widetilde{\deg}(f \circ g) = \Omega(\widetilde{\deg}(f)\widetilde{\deg}(g))$ when,

- *$f = \mathsf{Amp}_t^d$ and g is any Boolean function,*
- *$g = \mathsf{Amp}_t^d$ and f is any Boolean function.*

where Amp_t is any Boolean function with full sign degree and $t = \Omega(\log n)$.

So, Theorem 8.31 gives a composition theorem for the recursive version of all such functions being inner or outer functions. Note that when the outer function has a full sign degree applying [She12] we get $\widetilde{\deg}(\mathsf{Amp}_t)$ in the lower bound part in Lemma 8.29, using which we can prove composition theorem for some classes of functions where we are not loosing the polylog(n) in the lower bound part compared to Theorem 1.10.

The proof is completely similar to the proof of Theorem 1.9, we are giving the proof for the sake of completeness. Note that here we are not losing the $\log n$ compared to Theorem 1.9.

Proof of Theorem 8.31. From lemma 8.29, we have $\widetilde{\deg}(f \circ \mathsf{Amp}_t \circ g) = \Omega(\widetilde{\deg}(f) \cdot \widetilde{\deg}(\mathsf{Amp}_t) \cdot \widetilde{\deg}(g))$.

If $f = \mathsf{Amp}_t^k$,

$$\widetilde{\deg}(\mathsf{Amp}_t^k \circ \mathsf{Amp}_t \circ g) \geq \widetilde{\deg}(\mathsf{Amp}_t^k) \cdot \widetilde{\deg}(\mathsf{Amp}_t) \cdot \widetilde{\deg}(g) \tag{8.17}$$

.

On the other hand using associativity if composition we have,

$$\widetilde{\deg}(\mathsf{Amp}_t \circ \mathsf{Amp}_t^k \circ g) \geq \widetilde{\deg}(\mathsf{Amp}_t^k \circ \mathsf{Amp}_t \circ g) \tag{8.18}$$

Applying Theorem 8.8 we also have the following,

$$\widetilde{\deg}(\mathsf{Amp}_t \circ \mathsf{Amp}_t^k \circ g) \leq \widetilde{\deg}(\mathsf{Amp}_t) \cdot \widetilde{\deg}(\mathsf{Amp}_t^k \circ g) \leq \widetilde{\deg}(\mathsf{Amp}_t) \cdot \widetilde{\deg}(\mathsf{Amp}_t^k \circ g). \tag{8.19}$$

From equation (8.17), (8.18) it follows that,

$$\widetilde{\deg}(\mathsf{Amp}_t^k \circ g) \geq \widetilde{\deg}(\mathsf{Amp}_t^k) \cdot \widetilde{\deg}(g).$$

For $g = \mathrm{Maj}_t^k$, we can derive the proof in a similar fashion. □

8.6.1 Proof of lemma 8.29

For the proof of lemma 8.29 we will be using the following composition theorem by [She11b] for the classes of outer function with full approximate degree.

Theorem 8.32 ([She11b])**.** *For any Boolean function $h : \{0,1\}^n \to \{0,1\}$ and $g : \{0,1\}^n \to \{0,1\}$,*

$$\widetilde{\deg}_{\frac{1-\varepsilon^{-t}}{2}}(h \circ g) = \Omega(\widetilde{\deg}(h) \cdot \widetilde{\deg}(g))$$

where h is a function with full sign-degree.

Let us start by recalling the lower bound we are going to prove here:

For any Boolean function $f : \{0,1\}^n \to \{0,1\}$ and $g : \{0,1\}^m \to \{0,1\}$ we have,

$$\widetilde{\deg}(f \circ \mathsf{Amp}_t \circ g) = \Omega(\widetilde{\deg}(f) \cdot \widetilde{\deg}(\mathsf{Amp}_t) \cdot \widetilde{\deg}(g)),$$

where Amp_t is any Boolean function on t bits with full sign degree.

The proof will be along with the same line of Lemma 8.3. Instead of using the dual of the inner function g, we will be using the dual of $\mathsf{Amp}_t \circ g$. Fix any constant $0 < \varepsilon < 1$ and consider $\frac{1-\varepsilon^{-t}}{2}$-approximating polynomial of $\mathsf{Amp}_t \circ g$. Let $\widetilde{\deg}_{\frac{1-\varepsilon^{-t}}{2}}(\mathsf{Amp}_t \circ g) =: d$. By Lemma 8.7, there exists a function $\psi : \{0,1\}^{mt} \to \mathrm{R}$ such that

$$\sum_{x \in \{0,1\}^{mt}} |\psi(x)| = 1, \tag{8.20}$$

$$\tag{8.21}$$

$$\sum_{x \in \{0,1\}^{mt}} \psi(x) \cdot g(x) > \frac{1-\varepsilon^{-t}}{2}, \tag{8.22}$$

and

$$\sum_{x \in \{0,1\}^{mt}} \psi(x) \cdot p(x) = 0 \tag{8.23}$$

for every polynomial p of degree $< d$.

Similar to Lemma 8.3 let μ be the probability distribution on $\{0,1\}^{mt}$ given by $\mu(x) = |\psi(x)|$ for $x \in \{0,1\}^{mt}$. Define μ_0 and μ_1 similarly. We have the following properties of μ_0 and μ_1.

Lemma 8.33.

$$\mathop{\mathbb{E}}_{x\sim\mu_1}[\mathsf{Amp}_t \circ g(x)] > 1 - \varepsilon^{-t}, \tag{8.24}$$

and

$$\mathop{\mathbb{E}}_{x\sim\mu_0}[\mathsf{Amp}_t \circ g(x)] < \varepsilon^{-t}. \tag{8.25}$$

Consider the following linear operator L that maps functions $h : \{0,1\}^{ntm} \to \mathrm{R}$ to functions $Lh : \{0,1\}^n \to \mathrm{R}$,

$$Lh(z) = \mathop{\mathbb{E}}_{x_1\sim\mu_{z_1}} \mathop{\mathbb{E}}_{x_2\sim\mu_{z_2}} \cdots \mathop{\mathbb{E}}_{x_n\sim\mu_{z_n}} [h(x_1, x_2, \ldots, x_n)]. \tag{8.26}$$

Let $H := f \circ \mathsf{Amp}_t \circ g$, where $f : \{0,1\}^n \to \{0,1\}$ and $g : \{0,1\}^m \to \{0,1\}$. Further, let $p_H : \{0,1\}^{ntm} \to \mathrm{R}$ be an ε-approximating polynomial for the composed function $H : \{0,1\}^{ntm} \to \{0,1\}$. That is, $||H - p_H||_\infty \leq \varepsilon$. Then, it follows from the linearity that

$$||L(H - p_H)||_\infty \leq \varepsilon.$$

We now explore the behavior of L on H and p_H.

Lemma 8.34.

$$\deg(Lp_H) \leq \frac{\deg(p_H)}{\widetilde{\deg}_{\frac{1-\varepsilon^{-t}}{2}}(g)}.$$

Proof. Follows exactly as Lemma 8.17. ☐

Now it is left to prove that Lp_H is an approximating polynomial for f.

Lemma 8.35. *Fix $0 < \delta < 1$. Recall p_H is an ε-approximating polynomial for $H = f \circ \mathsf{Amp}_t \circ g$. Let $t = \Omega(\log n + \log(1/\delta))$ where the constant in $\Theta(\cdot)$ depends on ε. Then, Lp_H*

is a $(\delta+\varepsilon)$-approximating polynomial for f. That is,

$$\|f - Lp_H\|_\infty \le \|f - LH\|_\infty + \|LH - Lp_H\|_\infty \le \delta + \varepsilon.$$

Proof. It suffices to bound $\|f - Lh\|_\infty$. To this end, consider $Lh(z)$.

$$\begin{aligned}
(Lh)(z) &= \mathop{\mathbb{E}}_{x_1\sim\mu_{z_1}} \mathop{\mathbb{E}}_{x_2\sim\mu_{z_2}} \cdots \mathop{\mathbb{E}}_{x_n\sim\mu_{z_n}} [h(x_1,x_2,\dots,x_n)] && (8.27)\\
&= \mathop{\mathbb{E}}_{x_1\sim\mu_{z_1}} \mathop{\mathbb{E}}_{x_2\sim\mu_{z_2}} \cdots \mathop{\mathbb{E}}_{x_n\sim\mu_{z_n}} [f\circ \mathsf{Amp}_t \circ g(x_1,x_2,\dots,x_n)] && (8.28)\\
&= f\left(\mathop{\mathbb{E}}_{\mu_{z_1}}[\mathsf{Amp}_t\circ g], \mathop{\mathbb{E}}_{\mu_{z_2}}[\mathsf{Amp}_t\circ g], \dots, \mathop{\mathbb{E}}_{\mu_{z_n}}[\mathsf{Amp}_t\circ g]\right) && (8.29)\\
&= f\left(z_1',\dots,z_n'\right) && (8.30)
\end{aligned}$$

where $\|z-z'\|_\infty = O(\delta/n)$ because $t = \Omega(\log n + \log(\frac{1}{\delta}))$ and Lemmas 8.33.

Therefore, for arbitrary $z \in \{0,1\}^n$, $|f(z) - Lh(z)| = |f(z) - f(z')| \le \varepsilon_1$, where $\|z - z'\|_\infty = O(1/n)$.

Since Lp_h is an $(\varepsilon_1+\varepsilon)$-approximating polynomial for f, we also have

$$\deg(Lp_h) \ge \widetilde{\deg}_{\varepsilon_1+\varepsilon}(f).$$

We therefore have the following inequalities

$$\widetilde{\deg}_{\varepsilon_1+\varepsilon}(f) \le \deg(Lp) \le \frac{\deg(p)}{\widetilde{\deg}_{\frac{1-\varepsilon^{-t}}{2}}(\mathsf{Amp}_t\circ g)}.$$

This implies

$$\deg(p) \ge \widetilde{\deg}_{\varepsilon_1+\varepsilon}(f)\cdot \widetilde{\deg}_{\frac{1-\varepsilon^{-t}}{2}}(\mathsf{Amp}_t\circ g) \ge \widetilde{\deg}_{\varepsilon_1+\varepsilon}(f)\cdot\widetilde{\deg}(\mathsf{Amp}_t)\cdot\widetilde{\deg}(g),$$

where the last inequality follows from Theorem 8.32 by [She11b].

$$\widetilde{\deg}_\varepsilon(f\circ\mathsf{Amp}_t\circ g) = \deg(p) \ge \widetilde{\deg}_{\varepsilon_1+\varepsilon}(f)\cdot\widetilde{\deg}(\mathsf{Amp}_t)\cdot\widetilde{\deg}_{1/3}(g) \qquad (8.31)$$

□

From the proof technique, it is clear that if we start with an $(\frac{1-\frac{\delta}{n}}{2})$-approximating polynomial for the inner function g then we are fine with the error accumulated. So, the following known results can be derived as a corollary of the above Lemma.

Corollary 8.36 ([She13c]). *For any Boolean function $f : \{0,1\}^n \to \{0,1\}$ and $g : \{0,1\}^m \to \{0,1\}$ we have $\widetilde{\deg}_\varepsilon(f \circ g) = \Omega(\widetilde{\deg}_{\varepsilon_1+\varepsilon}(f)\widetilde{\deg}_{\frac{1-\frac{\delta}{n}}{2}}(g))$, where ε, ε is some constant in $(0, \frac{1}{2})$.*

From above it also follows

Corollary 8.37 ([She13c]). *For any Boolean function $f : \{0,1\}^n \to \{0,1\}$ and $g : \{0,1\}^m \to \{0,1\}$ we have $\widetilde{\deg}_\varepsilon(f \circ g) = \Omega(\widetilde{\deg}(f)\widetilde{\deg}_\pm(g))$, where $\widetilde{\deg}_\pm(g)$ denotes the sign-degree of g.*

8.7 One-sided approximate degree

Here is some analogous result to Lemma 8.3 in the case of one-sided approximate degree. Let us define the concept of one-sided approximate degree first.

Definition 8.38. *A polynomial p is a one-sided ε-approximation to f if*

1. *for all $x \in f^{-1}(1)$, $|p(x) - 1| \leq \varepsilon$, and*
2. *for all $x \in f^{-1}(0)$, $p(x) \leq \varepsilon$.*

Definition 8.39. *The one-sided ε-approximate degree of f, denoted $\mathrm{odeg}_\varepsilon(f)$, is the least degree of a real polynomial p that is a one-sided ε-approximation to f.*

Lemma 8.40 ([BT22]). *Let $f \colon \{0,1\}^n \to \{0,1\}$ be a Boolean function. Then $\mathrm{odeg}_\varepsilon(f) > d$ if and only if there exists a function $\psi \colon \{0,1\}^n \to \mathrm{R}$ such that*

$$\sum_{x\in\{0,1\}^n} |\psi(x)| = 1,$$
$$\sum_{x\in\{0,1\}^n} \psi(x)f(x) > \varepsilon,$$
$$\sum_{x\in\{0,1\}^n} \psi(x)p(x) = 0 \textit{ for every polynomial } p \textit{ of degree } \leq d, \quad \textit{and}$$
$$\psi(x) \leq 0 \textit{ for all } x \in f^{-1}(0).$$

The dual formulation for one-sided approximate degree is the following:

Corollary 8.41. *For any Boolean function $f : \{0,1\}^n \to \{0,1\}$ we have the following:*

$$\widetilde{\deg}(f \circ \mathsf{AND}_t \circ \mathsf{OR}) = \Omega(\widetilde{\deg}(f) \cdot \widetilde{\deg}(\mathsf{OR}))$$

for $t = \Omega(\log n)$.

Proof. When $g = \mathsf{OR}$, then we can use *one-sidedness* of dual of OR, i.e., $\mathbb{E}_{\mu_1}[\mathsf{OR}] = 1$ and $\mathbb{E}_{\mu_0}[\mathsf{OR}] < \varepsilon$, to amplify using AND instead of Maj. Which gives $\widetilde{\deg}(f \circ \mathsf{AND}_t \circ \mathsf{OR}) = \Omega(\widetilde{\deg}(f) \cdot \widetilde{\deg}(\mathsf{OR}))$. □

Precisely for one sided dual, $\mathbb{E}_{\mu_1}[g] = 1$ and $\mathbb{E}_{\mu_0}[g] < \varepsilon$ which gives the following observation.

Observation 8.42. *For any Boolean function $f : \{0,1\}^n \to \{0,1\}$ and $g : \{0,1\}^m \to \{0,1\}$ we have the following: $\widetilde{\deg}(f \circ \mathsf{AND}_t \circ g) = \Omega(\widetilde{\deg}(f) \cdot \mathrm{odeg}(g))$ for $t = \Omega(\log n)$.*

8.8 Conclusion

Towards the main open problem if $\widetilde{\deg}$ composes or not we have the following conjecture. Can we give an upper bound on $\widetilde{\deg}(f \circ \mathrm{Maj}_t \circ g)$ in terms of $\widetilde{\deg}(f \circ g)$? Precisely,

Conjecture 8.43. *Is $\widetilde{\deg}(f \circ \mathrm{Maj}_t \circ g) = \widetilde{O}(\widetilde{\deg}(f \circ g))$, where $t = \Theta(\log n)$ and n is the arity of the outer function?*

Observe that an affirmative solution to the above question solves the composition question for the approximate degree. Another interesting question is to find other classes of functions for which the analogue of Lemma 8.29 holds.

Question 8.44. *Find non-trivial classes of functions H such that $\widetilde{\deg}(f \circ H \circ g) = \widetilde{\Omega}(\widetilde{\deg}(f) \cdot \widetilde{\deg}(H) \cdot \widetilde{\deg}(g))$?*

It has the following two useful implications. First, this gives composition for functions $h \in H$. In particular, when one of the functions h (inner or outer) belongs to the class H then $\widetilde{\deg}(f \circ h \circ g) = \widetilde{\Omega}(\widetilde{\deg}(f) \cdot \widetilde{\deg}(h) \cdot \widetilde{\deg}(g))$ along with Theorem 8.8 implies

$$\widetilde{\deg}(h \circ g) = \widetilde{\Omega}(\widetilde{\deg}(h) \cdot \widetilde{\deg}(g)) \quad \text{and} \quad \widetilde{\deg}(f \circ h) = \widetilde{\Omega}(\widetilde{\deg}(f) \cdot \widetilde{\deg}(h)).$$

Secondly, A function $h \in H$ can be used as ‘hardness amplifier’ functions.

Chapter 9

Randomized query complexity composition for some classes of functions

9.1 Introduction

For any Boolean functions f and g, the question is whether $\mathrm{R}(f \circ g) = \widetilde{\Theta}(\mathrm{R}(f) \cdot \mathrm{R}(g))$, is known as the composition question for the randomized query complexity. This is one of the most important and well-studied problems in the field of analysis of Boolean functions, and yet we are far from answering them satisfactorily. For R the upper bound inequality is known, i.e., $\mathrm{R}(f \circ g) = \widetilde{O}(\mathrm{R}(f) \cdot \mathrm{R}(g))$ (folklore). Thus it is enough to prove the lower bound on the complexity of composed function in terms of the individual functions. Most of the attempts to prove this direction of the question have focused on special cases when the outer function has certain special properties[1].

The initial steps taken towards answering the composition question for R were to show a lower bound by using a weaker measure of outer function than the randomized query complexity. In particular, it was shown that $\mathrm{R}(f \circ g) = \Omega(\mathsf{s}(f) \cdot \mathrm{R}(g))$ [GJPW18a, AKK16], where $\mathsf{s}(f)$ is the sensitivity of f (Definition 7.10). Since $\mathsf{s}(f) = \Theta(\mathrm{R}(f))$ for any symmetric function[2] f, these results show that R composes when the outer function is a symmetric function (like OR, AND, Majority, Parity, etc.). The lower bound was later improved to obtain $\mathrm{R}(f \circ g) = \Omega(\mathsf{fbs}(f) \cdot \mathrm{R}(g))$ [BDG$^+$20, BDB20a], where $\mathsf{fbs}(f)$ is the fractional block sensitivity of f (Definition 2.12).

[1] We note that some works have also studied the composition of R when the inner functions have special properties, for example, [ABK16, BK18, AGJ$^+$17, GLSS19, Li21, BDBGM22].

[2] Functions that depend only on the Hamming weight of their input.

Unfortunately, at this stage, there could even be a cubic gap between R and fbs; the best known bound is $\mathrm{R}(f) = O(\mathsf{fbs}(f)^3)$ [ABK+21b]. Given that there are already known Boolean functions with quadratic gaps between $\mathsf{fbs}(f)$ and $\mathrm{R}(f)$ (e.g., BKK function [ABK16]), it is natural to consider composition question for randomized query complexity when R is *big* but fbs is *small*. We take a step towards this problem by showing that composition for R holds when the outer function has full randomized query complexity, i.e., $\mathrm{R}(f) = \Theta(n)$, where n is the arity of the outer function f.

It is known that the measures compose if one assumes various properties of the outer function f (or inner function g).

A recent landmark result (Ben-David and Blais, 2020) showed that $\mathrm{R}(f \circ g) = \Omega(\mathrm{noisyR}(f) \cdot \mathrm{R}(g))$. This implies that composition holds whenever $\mathrm{noisyR}(f) = \widetilde{\Theta}(\mathrm{R}(f))$. We show two results:

1. When $\mathrm{R}(f) = \Theta(n)$, then $\mathrm{noisyR}(f) = \Theta(\mathrm{R}(f))$. In other words, composition holds whenever the randomized query complexity of the outer function is full.
2. If R composes with respect to an outer function, then noisyR also composes with respect to the same outer function.

It is already known that both R compose when the outer function is symmetric. We also extend these results to weaker notions of symmetry with respect to the outer function.

9.2 Preliminaries

We will start by describing some of the notions that we will be needing for this chapter.

The function Gap-Majority has played an important role in the study of the composition of R.

Definition 9.1 (Gap-Majority)**.** *The function* $\mathrm{GapMaj}_t : \{0,1\}^t \to \{0,1,*\}$ *is a partial function with arity t such that*

$$\mathrm{GapMaj}_t(x) = \begin{cases} 1 & \textit{if } |x| = t/2 + 2\sqrt{t}, \\ 0 & \textit{if } |x| = t/2 - 2\sqrt{t}, \\ * & \textit{otherwise.} \end{cases}$$

It can be shown that $\mathrm{R}(\mathrm{GapMaj}_t) = \Theta(t)$ [BDB20b].

In regards to the composition question of R, one of the most significant complexity measures (defined by Ben-David and Blais [BDB20b]) is that of noisyR. We first define the noisy oracle model.

Definition 9.2 (Noisy Oracle Model and Noisy Oracle Access to a String ([BDB20b])). *For $b \in \{0,1\}$, a noisy oracle to b takes a parameter $-1 \leq \gamma \leq 1$ as input and returns a bit b' such that $\Pr[b' = b] = (1+\gamma)/2$. The cost of one such query is γ^2. Each query to noisy oracle returns independent bits.*

For $x = (x_1, \ldots, x_n) \in \{0,1\}^n$, noisy oracle access to x is access to n independent noisy oracles, one for each bit x_i, $i \in [n]$.

Next, we define the noisy oracle model of computation.

Definition 9.3 (Noisy Oracle Model of Computation ([BDB20b])). *Let $f : \{0,1\}^n \to \{0,1,*\}$ be a partial Boolean function. A* noisyR *query algorithm A computes f if for all $x \in \mathrm{Dom}(f)$, $\Pr[A(x) \neq f(x)] \leq 1/3$, where A is a randomized algorithm given noisy oracle access to x, and the probability is over both noisy oracle calls and the internal randomness of the algorithm A. The cost of the algorithm A for an input x is the sum of the cost of all noisy oracle calls made by A on x, and the cost of A, cost(A), is the maximum cost over all $x \in \mathrm{Dom}(f)$. The* noisyR *randomized query complexity of f, denoted by* $\mathrm{noisyR}(f)$, *is defined as*

$$\mathrm{noisyR}(f) = \min_{A \text{ computes } f} cost(A).$$

Again, $1/3$ in the above definition can be replaced by any constant $< 1/2$. If only queries with $\gamma = 1$ are allowed in the noisy query model, then we obtain the usual randomized algorithm for f, thus $\mathrm{noisyR}(f) = O(\mathrm{R}(f))$.

Some useful previous results: We will also be crucially using a few results from prior works in our proofs. The following are a couple of useful results on noisyR.

Lemma 9.4 ([BDB20b]). *Let f be a non-constant partial Boolean function then* $\mathrm{noisyR}(f) = \Omega(1)$.

Theorem 9.5 ([BDB20b]). *For all partial functions f and g,* $\mathrm{R}(f \circ g) = \Omega(\mathrm{noisyR}(f) \cdot \mathrm{R}(g))$.

We will also be using the following theorem of [BBGK18a] regarding the composition question of bdeg when the outer function is PrOR_n. Informally, we will call it the Promise-OR composition theorem by [BBGK18a]: For any Boolean function $g : \{0,1\}^m \to \{0,1\}$ we have,

$$\mathsf{bdeg}\left(\mathsf{PrOR}_n \circ g\right) = \Omega\left(\sqrt{n} \cdot \widetilde{\deg}(g)/\log n\right).$$

9.3 Our results and techniques

9.3.1 Lower bounds on $\mathrm{R}(f \circ g)$ when the outer function has full randomized query complexity

Sherstov [She12] proved that $\widetilde{\deg}(f \circ g) = \Omega(\widetilde{\deg}(f) \cdot \widetilde{\deg}(g))$ when the approximate degree of the outer function f is $\Theta(n)$, where n is the arity of f. Though, a corresponding result for the case of randomized query complexity was not known. Our main result is to prove the corresponding theorem for randomized query complexity.

Theorem 9.6. *Let f be a partial Boolean function on n-bits such that* $\mathrm{R}(f) = \Theta(n)$. *Then for all partial functions g, we have*

$$\mathrm{R}(f \circ g) = \Omega(\mathrm{R}(f) \cdot \mathrm{R}(g)).$$

The proof of this theorem is given in Section 9.4. Notice, since $\mathrm{R}(f \circ g) = O(\mathrm{R}(f) \cdot \mathrm{R}(g) \log \mathrm{R}(f))$ (by error reduction), Theorem 9.6 says that composition of R holds when the randomized query complexity of the outer function, f, is $\Theta(n)$. Next, we give main ideas behind the proof of the above theorem.

Ideas behind proof of Theorem 9.6 A crucial complexity measure that we use for the proof of Theorem 9.6 is called the *noisy randomized query complexity*, first introduced by Ben-David and Blais [BDB20b] in order to study randomized query complexity. Noisy randomized query complexity can be seen as a generalization of randomized query complexity where the algorithm can query a bit with any bias and only pays proportionally to the square of the bias in terms of cost (see Definition 9.3). They give the following characterization of $\mathrm{noisyR}(f)$ (the noisy randomized query complexity of f).

Theorem 9.7 (Ben-David and Blais [BDB20b]). *For all partial functions f on n-bits, we have*

$$\mathrm{noisyR}(f) = \Theta\left(\frac{\mathrm{R}(f \circ \mathrm{GapMaj}_n)}{n}\right), \tag{9.1}$$

where GapMaj_n *is the Gap-Majority function on n bits whose input is promised to have Hamming weight either* $(n/2+2\sqrt{n})$ *(in which case the output is* -1*) or* $(n/2-2\sqrt{n})$ *(in which case the output is* 1*).*

We want to point out that the arity of f and Gap-Majority is the *same* in Theorem 9.7. Towards a proof of Theorem 9.6, we first make the following crucial observation.

Observation 9.8. *Let f be a partial Boolean function on n bits. If* $t(n) \geq 1$ *is a non-decreasing function of n and*

$$\mathrm{noisyR}(f) = \Omega\left(\frac{\mathrm{R}(f \circ \mathrm{GapMaj}_{t(n)})}{t(n)}\right),$$

then $\mathrm{R}(f \circ g) = \Omega((\mathrm{R}(f) \cdot \mathrm{R}(g))/t(n))$ *for all partial functions g.*

In particular choosing $t(n)$ to be $(\log n)$, if the outer function f satisfies

$$\mathrm{noisyR}(f) = \Omega\left(\frac{\mathrm{R}(f \circ \mathrm{GapMaj}_{\log n})}{\log n}\right). \tag{9.2}$$

then the above observation gives $\mathrm{R}(f \circ g) = \Omega((\mathrm{R}(f) \cdot \mathrm{R}(g))/(\log n))$ for all partial functions g.

The Observation 9.8 allows us to approach the composition question for randomized query complexity in a conceptually fresh manner. The goal of proving that randomized query complexity composes for a function or a class of functions, say upto $(\log n)$-factor, reduces to showing that Equation 9.2 holds for that function or class of functions for $t(n) = \log n$.

We are able to show that Equation 9.2 holds for all non-decreasing functions $t(n)$.

Theorem 9.9. *Let f be a partial function on n bits and let* $t \geq 1$*, then* $\mathrm{R}(f \circ \mathrm{GapMaj}_t) = O(t \cdot \mathrm{noisyR}(f) + n)$.

Notice that this is a generalization of Ben-David and Blais' characterization of noisyR given by Theorem 9.7 in one direction. To give an idea of the proof, their characterization (Theorem 9.7) shows that any noisy oracle algorithm for f can be simulated using only two biases, 1 and $1/\sqrt{n}$ (where n is the arity of f), with only constant overhead. We generalize this by showing that the same simulation works with a slight overhead even when the bias $1/\sqrt{n}$ is replaced by a bias $1/\sqrt{t}$, for some $t \geq 1$. A proof the above theorem is provided in Section 9.4.1.

This seemingly inconsequential generalization allows us to complete the proof of Theorem 9.6, i.e. if for an n-bit partial function f, $\mathrm{R}(f) = \Theta(n)$, then $\mathrm{R}(f \circ g) = \widetilde{\Theta}(\mathrm{R}(f) \cdot \mathrm{R}(g))$ for all partial functions g (see Section 9.4 for details).

Furthermore, Theorem 9.9 even sheds light on *the composition question for* noisyR. A corollary of this theorem is that if R composes with respect to an outer function, then noisyR also composes with respect to the same outer function (see Section 9.4 for a proof).

Corollary 9.10. *Let f be a partial Boolean function. If* $\mathrm{R}(f \circ g) = \widetilde{\Theta}(\mathrm{R}(f) \cdot \mathrm{R}(g))$ *for all partial functions g then* $\mathrm{noisyR}(f \circ g) = \widetilde{\Theta}(\mathrm{noisyR}(f) \cdot \mathrm{noisyR}(g))$.

9.3.2 Composition theorem when the outer function has some Symmetry

Applying Theorem 9.6 we can prove the composition theorem for some classes of functions that were not known beforehand. Precisely we will prove composition theorem when the function is strongly junta symmetric and the number of junta variable is linear in n which is really high. For the Definition of strongly junta symmetric function please refer Chapter 6.

Theorem 9.11 (Restatement of Theorem **??**). *For any strongly $k-$junta symmetric function $f : \{0,1\}^n \to \{0,1\}$ and any Boolean function $g : \{0,1\}^m \to \{0,1\}$, we have* $\mathsf{R}(f \circ g) = \widetilde{\Omega}(\mathsf{R}(f) \cdot \mathsf{R}(g))$ *where $n-k = \Theta(n)$.*

Proof. There exists an assignment of the k-bits such that the resulting function is a non-constant symmetric function on $(n-k)$ bits. Since the sensitivity of the restricted function is $\Omega(n)$, the randomized query complexity is also $\Omega(n)$ (see [Nis89]). Hence, from Theorem 9.6 the result follows. ☐

9.4 Results about composition of R

This section is devoted to the results related to the composition of randomized query complexity. Our main result states that composition of R holds if the outer function has full randomized query complexity (Theorem 9.6). As mentioned in the proof idea, the proof critically depends on the notion of noisy randomized query complexity and its properties (introduced by Ben-David and Blais [BDB20b]).

Recall the definition of noisy randomized query complexity of a function f from Definition 9.3. As mentioned in the introduction (Theorem 9.7), Ben-David and Blais [BDB20b] proved that

$$\text{noisyR}(f) = \Theta\left(\frac{\text{R}(f \circ \text{GapMaj}_n)}{n}\right), \tag{9.3}$$

where GapMaj_n is the Gap-Majority function on n bits. Note that Ben-David and Blais proved Equation 9.3 when the arity of functions f and Gap-Majority is the same. We show that if Equation 9.3 can be generalized for Gap-Majority functions of arbitrary arity for some outer function f, then randomized query complexity composes for the function f. We restate the following observation from the introduction.

Let f be a partial Boolean function on n bits. If $t(n) \geq 1$ is a non-decreasing function of n and

$$\text{noisyR}(f) = \Omega\left(\frac{\text{R}(f \circ \text{GapMaj}_{t(n)})}{t(n)}\right),$$

then $\text{R}(f \circ g) = \Omega((\text{R}(f) \cdot \text{R}(g))/t(n))$ for all partial functions g.

Proof. Suppose $\text{noisyR}(f) = \Omega\left(\frac{\text{R}(f \circ \text{GapMaj}_{t(n)})}{t(n)}\right)$, since $\text{R}(f \circ \text{GapMaj}_t) \geq \text{R}(f)$, we have, $\text{noisyR}(f) = \Omega(\text{R}(f)/(t(n))$. Theorem 9.7 implies that a lower bound on noisyR translates to a lower bound on $\text{R}(f \circ g)$. We have,

$$\begin{aligned} \text{R}(f \circ g) &= \Omega(\text{noisyR}(f) \cdot \text{R}(g)) && \text{(Theorem 9.5)} \\ &= \Omega\left(\frac{\text{R}(f) \cdot \text{R}(g)}{t(n)}\right). && \square \end{aligned}$$

Observation 9.8 follows from the above observation by choosing $t(n)$ to be a small function of n.

We restate from Section 7.1 our generalized characterization of noisyR (i.e., generalization of Equation 9.3) and proof it in the following subsection.

9.4.1 Proof of Theorem 9.9

Let us start by recalling the theorem we want to prove in this subsection. We will prove the following:

Theorem 9.12. *Let f be a partial function on n bits and let $t \geq 1$, then* $\text{R}(f \circ \text{GapMaj}_t) = O(t \cdot \text{noisyR}(f) + n)$.

We mention two lemmas from Ben-David and Blais [BDB20a] that we need for the proof of this theorem. The first lemma shows that given a noisy oracle with γ bias, it can simulate an oracle of $\gamma' \geq \gamma$ bias by making approximately $\left(\frac{\gamma'}{\gamma}\right)^2$ queries.

Lemma 9.13 ([BDB20a]). *Let $\gamma \in [-1/3, 1/3]$ be nonzero and $k \leq 1/\gamma^2$ be an odd positive integer. If we take k independent samples from Bernoulli$((1+\gamma)/2)$ and take their majority, the resultant distribution is Bernoulli$((1+\gamma')/2)$ where $\mathsf{sign}(\gamma') = \mathsf{sign}(\gamma)$ and*

$$\frac{\sqrt{k}|\gamma|}{3} \cdot \leq |\gamma'| \leq 3\sqrt{k}|\gamma|.$$

The following lemma shows that it is enough to query the noisy oracle with just two biases.

Lemma 9.14 ([BDB20a]). *Let f be a partial function and let A be an optimal noisy randomized algorithm for A of cost* $\mathrm{noisyR}(f)$. *Then there is another noisy randomized algorithm $\widehat{A}$ of cost $O(\mathrm{noisyR}(f))$, which only queries its noisy oracles with parameter either $\gamma = 1$ or $\gamma = \widehat{\gamma}$, where $\widehat{\gamma} > 0$ is the smallest bias that A uses on any input.*

Another important lemma needed for the proof concerns the property of a random walk on a line where the coin is biased with some probability.

Lemma 9.15. *Define a random walk on a line where the coin gives head with probability $\frac{1+\widehat{\gamma}}{2}$. The random walk start at 0 and stops if it reaches T or $-T$. Conditioned on the fact that we reach T before $-T$, let the expected number of steps of the walk be μ_T, then*

$$\mu_T = \frac{T}{\widehat{\gamma}} - \frac{2T}{\widehat{\gamma}}(1-\widehat{\gamma})^T \left(\frac{(1+\widehat{\gamma})^T - (1-\widehat{\gamma})^T}{(1+\widehat{\gamma})^{2T} - (1-\widehat{\gamma})^{2T}}\right). \qquad \text{([Fel91, Chapter XIV])}$$

Moreover, for $T = \Theta(1/(\sqrt{t} \cdot \widehat{\gamma}))$, we have

- *$\mu_{2T} = \Omega(1/(t \cdot \widehat{\gamma}^2))$, and*
- *$\mu_{2T} \leq 12\mu_T$.*

For completeness, we present the proof after the proof of Theorem 9.9

Proof of of Theorem 9.9. From Lemma 9.14, we can assume that f can be computed by a noisy randomized algorithm $\widehat{A}$ of cost $O(\mathrm{noisyR}(f))$ that makes queries with only two biases:

1 and $\widehat{\gamma}$. We will now simulate $\widehat{A}$, that uses bias 1 and $\widehat{\gamma}$, with a noisyR algorithm B that uses bias 1 and $1/\sqrt{t}$, where $t \geq 1$.

Case 1: The first case is when $\widehat{\gamma} \geq 1/\sqrt{t}$. In this case we can use multiple oracle calls of bias $1/\sqrt{t}$ to simulate one oracle call of bias $\widehat{\gamma}$. To do this, algorithm B makes $O(t\widehat{\gamma}^2)$ calls of bias $\widehat{\gamma}$ and takes their majority. By Lemma 9.13, the algorithm B obtains a bit of bias slightly greater than $\widehat{\gamma}$. Then B adds some bias to this bit to obtain a bit of bias exactly $\widehat{\gamma}$. The cost paid by B is $O(t\widehat{\gamma}^2 \cdot 1/t) = O(\widehat{\gamma}^2)$.

Case 2: We now consider the case when $\widehat{\gamma} < 1/\sqrt{t}$. In this case we wish to generate many bits of low bias ($\widehat{\gamma}$) using a single bit of high bias ($1/\sqrt{t}$). Consider the following random walk based sampling procedure.

- **Setup.** Let

$$T = \left\lfloor \frac{1}{5\sqrt{t}\widehat{\gamma}} \right\rfloor . \tag{9.4}$$

  The random walk will take place on a line that is marked with integral multiples of T, with T being right to 0, $-T$ being left to 0 and so on. Also, $(T-1)$ points are marked between every two adjacent integral multiples of T. The random walk starts at 0. Also, let

$$R = \left(\frac{1+\widehat{\gamma}}{1-\widehat{\gamma}}\right)^T .$$

1. Toss a coin of bias
$$\delta' = \frac{R-1}{R+1}.$$
2. If the result of the toss is 1, then sample a sequence of bits using a $\widehat{\gamma}$-bias coin consisting of w 1's and z 0's, conditioned on $w - z = T$. This sampling procedure can be simulated by picking a random walk from the set of $\widehat{\gamma}$-biased random walks on the line which reach T before $-T$ (starting at 0).
3. If the result of the toss is 0, then sample a sequence of bits using a $\widehat{\gamma}$-bias coin consisting of w 1's and z 0's, conditioned on $w - z = -T$. This sampling procedure

can be simulated by picking a random walk from the set of $\widehat{\gamma}$-biased random walks on the line which reach $-T$ before T (starting at 0).

We now prove the correctness of the above protocol. A crucial observation in [BDB20a] is that conditioned on a set of sequences, all of which reach T before $-T$, the probability of choosing each sequence is the same whether we choose bias $\widehat{\gamma}$ or $-\widehat{\gamma}$ for Step 2. and Step 3. above. The reason is that the probability for any sequence of walks with w 1's and z 0's such that $w-z=T$, is R times more likely when we choose a $\widehat{\gamma}$ biased coin than when we choose a $-\widehat{\gamma}$ biased coin. Furthermore, the probability of a $\widehat{\gamma}$-biased walk starting at 0 and reaching T is $R/(R+1)$. This is because if the probability of reaching T is p, then probability of reaching $-T$ is p/R. Since $p+p/R=1$, $p=R/(R+1)$. Similarly, the probability that a $-\widehat{\gamma}$-biased coin reaches T is $1/(R+1)$ and reaches $-T$ is $R/(R+1)$. Thus δ' is chosen such that the probability of being correct is exactly $R/(1+R)$. Also, assuming that $\widehat{\gamma}$ is smaller than $1/10$, the bias δ' can be shown to be smaller than but within a constant factor of $1/\sqrt{t}$ (we refer to [BDB20a, Theorem 4] for details). Thus, by adding noise, the bias $1/\sqrt{t}$ can be converted to bias δ'.

Let μ_T denote the expected number of bits that one run of the above sampling procedure generates. Lemma 9.15 implies that μ_T is lower bounded by $\Omega(1/(t\cdot\widehat{\gamma}^2))$.

In summary, for one iteration of the Step 1. to 3. of the sampling procedure, the following are true: a) The choice of δ' in Step 1. implies correct distribution (from bias $\widehat{\gamma}$) of generating the coin tosses, and b) an expected number of $\Omega(1/(t\widehat{\gamma}^2))$ $\widehat{\gamma}$-biased samples are generated. c) Cost paid is δ^2.

Before moving further let us mention explicitly how algorithm B works: B runs algorithm $\widehat{A}$ on the given input. For each $i\in[n]$:

- If $\widehat{A}$ makes a query of bias 1 then B also queries that bit with bias 1. The value of such bits are known with certainty and no other queries to these bits are made.
- When $\widehat{A}$ makes a query of bias $\widehat{\gamma}$ then
  - If B has $\widehat{\gamma}$ biased bits available, then B uses these bits to run $\widehat{A}$. Otherwise B generates the above sampling procedure to generate $\widehat{\gamma}$ biased bits.

Clearly, the correctness of B follows from the correctness of $\widehat{A}$. In order to upper bound the expected cost of B, we upper bound the expected number of times B runs the above sampling procedure for each index $i\in[n]$. Recall that our goal is to show that the total expected cost is bounded by:

$$O(\mathrm{noisyR}(f)+n/t).$$

Fix a $k \in [n]$ and consider the noisy query algorithm B that uses bias $1/\sqrt{t}$ and 1 to simulate the algorithm $\widehat{A}$ (that uses bias $\widehat{\gamma}$ and 1). For a run, say r, of algorithm $\widehat{A}$

- Let $T_{r,k}$ be the number of times algorithm $\widehat{A}$ queries the kth bit in the r-th run and let

$$T_k = \mathbb{E}_r[T_{r,k}].$$

Observe that

$$\sum_k T_k = O(\text{noisyR}(f)/\widehat{\gamma}^2). \tag{9.5}$$

- Let $X_1, X_2, \ldots, X_{\ell_r}$ be the number of bits sampled by B, while simulating the r-th run of $\widehat{A}$, where B makes ℓ_r independent calls to the above sampling procedure in order to meet the demand of $T_{r,k}$.

Let

$$L_{k,r} = (X_1 + \cdots + X_{\ell_r}) - T_{r,k}, \tag{9.6}$$

denote the number of surplus bits (i.e. not used in simulation) generated by the algorithm B. Thus,

$$\begin{aligned}\mathbb{E}[X_1 + \cdots + X_{\ell_r}] &= \mathbb{E}[T_{r,k}] + \mathbb{E}[L_{k,r}] \\ &= T_k + \mathbb{E}[L_{k,r}], \end{aligned} \tag{9.7}$$

where the expectation in the first equation is over both the run r and the randomness of the sampling procedure.

We now upper bound $\mathbb{E}[L_{k,r}]$. Note that $L_{k,r}$ denotes the number of bits generated by the following walk on line: a $\widehat{\gamma}$-biased random walk on line starts at some point x_r, where $-T < x_r < T$ depends on $X_1, \ldots, X_{\ell_r}$ and $T_{r,k}$, and stops after reaching either $-T$ or T. The expected value of $L_{k,r}$ is upper bounded by the expected value of a $\widehat{\gamma}$-biased random walk on line starts at x_r and stops after reaching either $-(T + |x_r|)$ or $(T + |x_r|)$. Since $-T < x_r < T$, $\mathbb{E}[L_{k,r}]$ is upper bounded by the expected value of a $\widehat{\gamma}$-biased random walk on line starts at 0 and stops after reaching either $-2T$ or $2T$.

From Lemma 9.15 we have the following bound on μ_{2T} in terms of μ_T,

$$\mu_{2T} \leq 12\mu_T.$$

The above expression, combined with Equation 9.7, implies that for every $k \in [n]$

$$\mathbb{E}[X_1 + \cdots + X_{\ell_r}] \leq T_k + 12\mu_T, \tag{9.8}$$

where the expectation is over both r (i.e. randomness used by $\widehat{A}$) and the randomness of the sampling procedure.

Next, we upper bound the expected number of times B calls the sampling procedure, i.e. we upper bound $\mathbb{E}[\ell_r]$. Let I_t denote the random variable such that $I_t = 0$ if $t > \ell_r$ and $I_t = 1$ otherwise. Thus,

$$\begin{aligned}
\mathbb{E}\left[\sum_{i=1}^{\ell_r} X_i\right] &= \mathbb{E}\left[\sum_{i=1}^{\infty} X_i I_i\right] \\
&= \sum_{i=1}^{\infty} \mathbb{E}[X_i I_i] \\
&= \sum_{i=1}^{\infty} \Pr[I_i = 1]\mathbb{E}[X_i \mid I_i = 1] \\
&= \sum_{i=1}^{\infty} \Pr[\ell_r \geq i]\mathbb{E}[X_i \mid \ell_r \geq i] \\
&= \mathbb{E}[X_i] \sum_{i=1}^{\infty} \Pr[\ell_r \geq i] \\
&= \mathbb{E}[X_i]\mathbb{E}[\ell_r].
\end{aligned}$$

In the second last equation, we have used $\mathbb{E}[X_i \mid \ell_r \geq i] = \mathbb{E}[X_i]$. This is because the event $\ell_r \geq i$ depends on $X_1, \ldots, X_{i-1}$ but not on X_i. Since $\mathbb{E}[X_i] = \mu_T$ for all i, along with Equation 9.8 we have

$$\mathbb{E}[\ell_r] \leq T_k/\mu_T + 12.$$

Summing over all k and using Equation 9.5, the expected number of queries made by algorithm B is upper bounded by

$$O\left(\frac{\text{noisyR}(f)}{\widehat{\gamma}^2 \mu_T} + 12n\right).$$

Thus the expected cost of B is upper bounded by

$$O\left(\frac{\text{noisyR}(f)}{t\widehat{\gamma}^2 \mu_T} + \frac{12n}{t}\right).$$

Since $\mu_T = \Omega(1/(t\widehat{\gamma}^2))$, the above quantity is upper bounded by

$$O\left(\text{noisyR}(f) + \frac{n}{t}\right).$$

In order to complete the proof of the theorem, we show how to simulate B to obtain a randomized query algorithm B' for $f \circ \text{GapMaj}_t$ of cost $O(t \cdot \text{noisyR}(f) + n)$. The algorithm B' simulates B in the following manner:

1. if b queries i-th bit of f with bias 1, for some $i \in [n]$, then B' queries all the bits of i-th Gap-Majority inner function.
2. if B queries i-th bit of f with bias $1/\sqrt{t}$, for some $i \in [n]$, then B' queries a random bit of i-th Gap-Majority inner function.

The correctness of B' follows directly from the simulation and the correctness of B. Also, in both the cases (1. and 2.), cost paid by B' is t times the cost paid by B. Thus

$$\text{R}(f \circ \text{GapMaj}_t) = O\left(t \cdot \text{noisyR}(f) + n\right). \tag{9.9}$$

□

Proof of Lemma 9.15. As mentioned in the statement of the lemma, the equality

$$\mu_T = \frac{T}{\widehat{\gamma}} - \frac{2T}{\widehat{\gamma}}(1-\widehat{\gamma})^T \left(\frac{(1+\widehat{\gamma})^T - (1-\widehat{\gamma})^T}{(1+\widehat{\gamma})^{2T} - (1-\widehat{\gamma})^{2T}}\right), \tag{9.10}$$

follows from [Fel91, Chapter XIV].

Next, we prove the second part of the lemma. The following inequalities are easy to observe.

$$\begin{aligned}
(1+\widehat{\gamma})^T - (1-\widehat{\gamma})^T &\leq \frac{2\widehat{\gamma}T}{1-\widehat{\gamma}^2T^2},\\
(1+\widehat{\gamma})^T - (1-\widehat{\gamma})^T &\geq 2\widehat{\gamma}T,\\
(1-\widehat{\gamma})^T &\geq 1 - T\widehat{\gamma}.
\end{aligned}$$

Using the above inequalities and Equation 9.10 we get the following bounds on μ_T

$$\frac{T^2}{(1+\widehat{\gamma}T)} \leq \mu_T \leq \frac{T}{\widehat{\gamma}}\left(\widehat{\gamma}T + 4\widehat{\gamma}^2T^2 - 4\widehat{\gamma}^3T^3\right). \tag{9.11}$$

For $T = \Theta(1/\sqrt{t} \cdot \widehat{\gamma})$, the above expression implies that μ_T is lower bounded by $\Omega(1/(t \cdot \widehat{\gamma}^2))$.

Now we use Equation 9.11 to upper bound μ_{2T} in terms of μ_T:

$$\begin{aligned}
\mu_{2T} &\leq \frac{2T}{\widehat{\gamma}}\left(2\widehat{\gamma}T + 16\widehat{\gamma}^2T^2 - 32\widehat{\gamma}^3T^3\right) \\
&= \frac{T^2}{(1+\widehat{\gamma}T)} \cdot \frac{(1+\widehat{\gamma}T)}{T^2} \cdot \frac{2T}{\widehat{\gamma}}\left(2\widehat{\gamma}T + 16\widehat{\gamma}^2T^2 - 32\widehat{\gamma}^3T^3\right) \\
&\leq \frac{(1+\widehat{\gamma}T)}{T^2} \cdot \frac{2T}{\widehat{\gamma}}\left(2\widehat{\gamma}T + 16\widehat{\gamma}^2T^2 - 32\widehat{\gamma}^3T^3\right) \cdot \mu_T \\
&= 2(1+\widehat{\gamma}T) \cdot \left(2 + 16\widehat{\gamma}T - 32\widehat{\gamma}^2T^2\right) \cdot \mu_T.
\end{aligned}$$

Since $T = \Theta(1/(\sqrt{t} \cdot \widehat{\gamma}))$, for a suitable choice of constant, we have,

$$\mu_{2T} \leq 12\mu_T.$$

□

This allows us to show that if for an n-bit partial function f, $\mathrm{R}(f) = \Theta(n)$, then $\mathrm{R}(f \circ g) = \widetilde{\Theta}(\mathrm{R}(f) \cdot \mathrm{R}(g))$ for all partial functions g (Theorem 9.6).

The proof of Theorem 9.6 is discussed in Section 9.4.2. A corollary of this theorem is that if R composes with respect to an outer function, then noisyR also composes with respect to the same outer function (Corollary 9.10).

We give proof of Theorem 9.6 in the next section and prove Corollary 9.10 in Section 9.4.4. We need the following theorem for these proofs, which lower bounds $\mathrm{R}(f \circ g)$ in terms of $\mathrm{R}(f)$ and $\mathrm{R}(g)$.

Theorem 9.16 ([GLSS19])**.** *Let f and g be partial functions then* $\mathrm{R}(f \circ g) = \Omega(\mathrm{R}(f) \cdot \sqrt{\mathrm{R}(g)})$.

9.4.2 Composition for functions with $\mathrm{R}(f) = \Theta(n)$

We restate the theorem below.

Theorem 9.17. *Let f be a partial Boolean function on n-bits such that* $\mathrm{R}(f) = \Theta(n)$*. Then for all partial functions g, we have*

$$\mathrm{R}(f \circ g) = \Omega(\mathrm{R}(f) \cdot \mathrm{R}(g)).$$

Proof. From Theorem 9.16 we have a lower bound on the randomized query complexity of $(f \circ \mathrm{GapMaj}_t)$:

$$\mathrm{R}(f \circ \mathrm{GapMaj}_t) = \Omega(\mathrm{R}(f) \cdot \sqrt{t}). \tag{9.12}$$

On the other hand, Theorem 9.9 gives an upper bound of $O(t \cdot \mathrm{noisyR}(f) + n)$ on $\mathrm{R}(f \circ \mathrm{GapMaj}_t)$. Thus, choosing $t = \left(\frac{C \cdot n}{\mathrm{noisyR}(f)}\right)$ for a large enough constant C, we have

$$\mathrm{R}(f) \cdot \sqrt{\frac{n}{\mathrm{noisyR}(f)}} = O\left(\frac{n}{\mathrm{noisyR}(f)} \cdot \mathrm{noisyR}(f) + n\right).$$

This implies that

$$\mathrm{R}(f) = O\left(\sqrt{n \cdot \mathrm{noisyR}(f)}\right). \tag{9.13}$$

Thus, if $\mathrm{R}(f) = \Theta(n)$, then $\mathrm{noisyR}(f) = \Omega(\mathrm{R}(f))$, which implies composition from Theorem 9.5. □

Notice that Equation 9.13 is equivalent to the following observation.

Observation 9.18. *Let f be a partial Boolean function on n-bits. Then,* $\mathrm{noisyR}(f) = \Omega\left(\frac{\mathrm{R}(f)^2}{n}\right)$.

When $\mathrm{R}(f) = \Theta(n)$, we have already seen that Observation 9.18 implies composition of randomized query complexity when the outer function is f.

Though, Observation 9.18 implies a more general result. When $\mathrm{R}(f)$ is close to n (arity of f), Observation 9.18 places a limit on the gap between $\mathrm{R}(f)$ and $\mathrm{noisyR}(f)$ (consequently on the violation of composition with outer function being f). These implications are formally discussed in the next Section 9.4.3.

9.4.3 Additional implications of Observation 9.18

Without loss of generality we can assume $\mathrm{R}(f \circ g) = \Omega(\mathrm{R}(g))$ (note that this is true when f is non-constant).

Ben-David and Blais [BDB20b] gave a counterexample for composition, but the arity of the used function was very high compared to the randomized query complexity. They observed that the composition can still be true in the weaker sense:

$$\mathrm{R}(f\circ g)=\Omega\left(\frac{\mathrm{R}(f)\cdot\mathrm{R}(g)}{\log n}\right).$$

Observation 9.18 shows that a much weaker composition result is true.

Corollary 9.19. *Let f and g be partial functions on n and m bits respectively. If* $\mathrm{R}(f\circ g)=\Omega(\mathrm{R}(g))$, *then*

$$\mathrm{R}(f\circ g)=\Omega\left(\frac{\mathrm{R}(f)\cdot\mathrm{R}(g)}{\sqrt{n}}\right).$$

Proof.

$$\begin{aligned}\mathrm{R}(f\circ g)&=\Omega(\mathrm{noisyR}(f)\cdot\mathrm{R}(g)) &&\text{(Theorem 9.5)}\\ &=\Omega\left(\frac{\mathrm{R}(f)^2\cdot\mathrm{R}(g)}{n}\right). &&(9.14)\end{aligned}$$

Where the last equality follows from Observation 9.18 [3] Now there are two cases:

- **Case 1.** $\mathrm{R}(f)=O(\sqrt{n})$. In this case $\mathrm{R}(f)/\sqrt{n}=O(1)$ and since we assumed $\mathrm{R}(f\circ g)=\Omega(\mathrm{R}(g))$, the claim follows from Equation 9.14.
- **Case 2.** $\mathrm{R}(f)=\Theta(n^{1/2}\cdot t(n))$ where $t(n)$ is a strictly increasing function of n. Thus,

$$\frac{\mathrm{R}(f)^2\cdot\mathrm{R}(g)}{n}=\Omega\left(t(n)^2\cdot\mathrm{R}(g)\right)=\Omega\left(\frac{\mathrm{R}(f)\cdot\mathrm{R}(g)}{\sqrt{n}}\right).$$

  Again, the claim follows from Equation 9.14.

□

The weaker composition, Corollary 9.19, implies that if $\mathrm{R}(f)$ and $\mathrm{R}(g)$ are comparable to the arity of these functions, the randomized query complexity of $f\circ g$ is "not far" from the conjectured randomized query complexity $\mathrm{R}(f)\cdot\mathrm{R}(g)$. In other words, if there is a large polynomial separation between $\mathrm{R}(f\circ g)$ and $(\mathrm{R}(f)\cdot\mathrm{R}(g))$, then $\mathrm{R}(f)$ and $\mathrm{R}(g)$ can not be too large.

[3] Sherstov [She12] proved that for Boolean functions f and g, $\widetilde{\deg}(f\circ g)=\Omega((\widetilde{\deg}(f)^2\widetilde{\deg}(g))/n)$. Thus in Equation 9.14 we prove the same result but in the randomized world.

Corollary 9.20. *Let f and g be partial functions such that f is a function on n-bits and g is a function on $t(n)$-bits where $t(n)$ is a strictly increasing function of n. If* $\mathrm{R}(f) = \Theta(n^{\beta})$, $\mathrm{R}(g) = \Theta(n^{\gamma})$ *and* $\mathrm{R}(f \circ g) = O((\mathrm{R}(f) \cdot \mathrm{R}(g))^{\alpha})$*, where $\alpha < 1$ is a constant, then $(1-\alpha)(\alpha+\beta) < 1/2$.*

Proof. For some constants A and B we have

$$A \cdot \frac{\mathrm{R}(f) \cdot \mathrm{R}(g)}{\sqrt{n}} \leq \mathrm{R}(f \circ g) \leq B \cdot (\mathrm{R}(f) \cdot \mathrm{R}(g))^{\alpha},$$

where the first inequality follows from Corollary 9.19 and second from assumption. Assigning the values of $\mathrm{R}(f)$ and $\mathrm{R}(g)$ in terms on n we have,

$$A \cdot n^{\beta+\gamma-1/2} \leq B \cdot n^{\alpha(\beta+\gamma)}$$
$$n^{(1-\alpha)(\beta+\gamma)-1/2} \leq \frac{B}{A}.$$

which implies, for large enough n, $(1-\alpha)(\beta+\gamma) \leq 1/2$. □

A special case of the above corollary is when arity and randomized query complexity of g are superpolynomial in n. In this case a polynomial gap between $\mathrm{R}(f \circ g)$ and $(\mathrm{R}(f) \cdot \mathrm{R}(g))$ is not possible.

Another implication of Theorem 9.6 is that composition of R for an outer function f implies the composition of noisyR for outer function being f (Corollary 9.10).

9.4.4 Proof of Corollary 9.10

First, we need the following lemma which follows from Theorem 9.9, Theorem 9.5 and Lemma 9.4.

Lemma 9.21. *Let f be a partial function on n bits and let $t = \Omega(n)$. Then*

$$\mathrm{noisyR}(f) = \Theta\left(\frac{\mathrm{R}(f \circ \mathrm{GapMaj}_t)}{t}\right).$$

Proof. From Theorem 9.9 we have for all $t \geq 1$, $\mathrm{R}(f \circ \mathrm{GapMaj}_t) = O(t \cdot \mathrm{noisyR}(f) + n)$. Since we have assumed $t = \Omega(n)$ and $\mathrm{noisyR}(f) = \Omega(1)$ (Lemma 9.4), we get $\mathrm{R}(f \circ \mathrm{GapMaj}_t) = O(t \cdot \mathrm{noisyR}(f))$. Thus, $\mathrm{noisyR}(f) = \Omega\left(\frac{\mathrm{R}(f \circ \mathrm{GapMaj}_t)}{t}\right)$.

The upper bound $\mathrm{noisyR}(f) = O\left(\frac{\mathrm{R}(f \circ \mathrm{GapMaj}_t)}{t}\right)$ follows from Theorem 9.5 and the fact that $\mathrm{R}(\mathrm{GapMaj}_t) = \Theta(t)$. □

Now we prove that if R composes for f then noisyR composes for that f. For convenience, we recall the statement of the corollary from the introduction.

Corollary 9.22. *Let f be a partial Boolean function. If* $\mathrm{R}(f \circ g) = \widetilde{\Theta}(\mathrm{R}(f) \cdot \mathrm{R}(g))$ *for all partial functions g then* $\mathrm{noisyR}(f \circ g) = \widetilde{\Theta}(\mathrm{noisyR}(f) \cdot \mathrm{noisyR}(g))$.

Proof. From Theorem 9.7, we have

$$\mathrm{noisyR}(f \circ g) = \Theta\left(\frac{\mathrm{R}\left((f \circ g) \circ \mathrm{GapMaj}_{mn}\right)}{mn}\right).$$

Since $(f \circ g) \circ h = f \circ (g \circ h)$, the right hand side of the above expression is equal to

$$\Theta\left(\frac{\mathrm{R}\left(f \circ (g \circ \mathrm{GapMaj}_{mn})\right)}{mn}\right).$$

The proof follows from the assumption that R composes and Lemma 9.21.

$$\begin{aligned}
\mathrm{noisyR}(f \circ g) &= \Theta\left(\frac{\mathrm{R}(f) \cdot \mathrm{R}\left(g \circ \mathrm{GapMaj}_{mn}\right)}{mn}\right) && \text{(assuming R composes)} \\
&= \Theta\left(\mathrm{R}(f) \cdot \mathrm{noisyR}(g)\right) && \text{(from Lemma 9.21)} \\
&= \Theta\left(\mathrm{noisyR}(f) \cdot \mathrm{noisyR}(g)\right). && \text{(assuming R composes)}
\end{aligned}$$

□

9.5 Conclusion

We showed that the composition question for R is equivalent to the following open question (which is a generalization of Ben-David and Blais [BDB20b] result):

Question 9.23. *Let $f : \{0,1\}^n \to \{0,1,*\}$ be a Boolean function. Then, is it true that for arbitrary t,* $\mathrm{noisyR}(f) = \Theta\left(\mathrm{R}(f \circ \mathrm{GapMaj}_t)/t\right)$*?*

While the main problem remains open here is one interesting open problem that may be small steps towards the main problem as well as better understanding.

Recently, in [SYZ04, Sun07, Dru11, Cha11, CKP22], the classes of transitive functions got a lot of attention as a natural generalization of the classes of symmetric functions. Can the result for symmetric functions be extended to transitive functions?

Question 9.24. *Can we prove that* R *composes when the outer function is transitive?*

List of publications on which this thesis is based

Published

- **Separations between Combinatorial Measures for Transitive Functions**

  Author: Sourav Chakraborty, Chandrima Kayal, Manaswi Paraashar
  Published in International Colloquium on Automata, Languages, and Programming (ICALP) 2022, volume 229, pages 36:1-36:20, 2022.
  https://doi.org/10.4230/LIPIcs.ICALP.2022.36
  Submitted to DMTCS, ID 11133, at 2023-03-30 11:21:42

- **On the Composition of Randomized Query Complexity and Approximate Degree**

  Author: Sourav Chakraborty, Chandrima Kayal, Rajat Mittal, Manaswi Paraashar, Swagato Sanyal, Nitin Saurabh
  Published in RANDOM 2023, volume 275, pages 63:1–63:23, 2023.
  https://doi.org/10.4230/LIPIcs.APPROX/RANDOM.2023.63
  Submitted to journal Computational Complexity on (on Feb 19, 2024)

- **Almost covering all the layers of the hypercube with multiplicities**

  Author: Arijit Ghosh, Chandrima Kayal, Soumi Nandi
  Published: Discrete Mathematics, volume 346(7), 113397, 2023.
  https://doi.org/10.1016/j.disc.2023.113397

- **Approximate degree composition for Recursive functions**
  Author: Sourav Chakraborty, Chandrima Kayal, Rajat Mittal, Manaswi Paraashar and Nitin Saurabh
  Published in RANDOM 2024, volume 317, pages 71:1–71:17, 2024.
  https://doi.org/10.4230/LIPIcs.APPROX/RANDOM.2024.71

References

[AA14] Scott Aaronson and Andris Ambainis. The need for structure in quantum speedups. *Theory Comput.*, 10:133–166, 2014.

[Aar08] Scott Aaronson. Quantum certificate complexity. *Journal of Computer and System Sciences*, 74(3):313–322, 2008.

[Aar12] Scott Aaronson. Impossibility of succinct quantum proofs for collision-freeness. *Quantum Inf. Comput.*, 12(1-2):21–28, 2012.

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, USA, 1st edition, 2009.

[ABB$^+$17] Andris Ambainis, Kaspars Balodis, Aleksandrs Belovs, Troy Lee, Miklos Santha, and Juris Smotrovs. Separations in query complexity based on pointer functions. *Journal of the ACM*, 64(5):32:1–32:24, 2017.

[ABK16] Scott Aaronson, Shalev Ben-David, and Robin Kothari. Separations in query complexity using cheat sheets. In *STOC*, pages 863–876, 2016.

[ABK$^+$21a] Scott Aaronson, Shalev Ben-David, Robin Kothari, Shravas Rao, and Avishay Tal. Degree vs. approximate degree and quantum implications of Huang's sensitivity theorem. In *STOC*, pages 1330–1342, 2021.

[ABK$^+$21b] Scott Aaronson, Shalev Ben-David, Robin Kothari, Shravas Rao, and Avishay Tal. Degree vs. approximate degree and quantum implications of Huang's sensitivity theorem. In *STOC*, pages 1330–1342, 2021.

[AF93] Noga Alon and Zoltán Füredi. Covering the Cube by Affine Hyperplanes. *European Journal of Combinatorics*, 14(2):79–83, 1993.

[AGG$^+$21] James Aaronson, Carla Groenland, Andrzej Grzesik, Tom Johnston, and Bartłomiej Kielak. Exact hyperplane covers for subsets of the hypercube. *Discrete Mathematics*, 344(9):112490, 2021.

[AGJ$^+$17] Anurag Anshu, Dmitry Gavinsky, Rahul Jain, Srijita Kundu, Troy Lee, Priyanka Mukhopadhyay, Miklos Santha, and Swagato Sanyal. A composition theorem for randomized query complexity. In *FSTTCS*, volume 93, pages 10:1–10:13, 2017.

[AKK16] Andris Ambainis, Martins Kokainis, and Robin Kothari. Nearly optimal separations between communication (or query) complexity and partitions. In *CCC*, volume 50, pages 4:1–4:14, 2016.

[All89] Eric Allender. A note on the power of threshold circuits. In *FOCS*, pages 580–584, 1989.

[Alo99] Noga Alon. Combinatorial Nullstellensatz. *Combinatorics, Probability and Computing*, 8(1–2):7–29, 1999.

[Amb05] Andris Ambainis. Polynomial degree and lower bounds in quantum complexity: Collision and element distinctness with small range. *Theory Comput.*, 1(1):37–46, 2005.

[Amb16] Andris Ambainis. Superlinear advantage for exact quantum algorithms. *SIAM Journal on Computing*, pages 617–631, 2016.

[Ant95] Martin Anthony. Classification by polynomial surfaces. *Discret. Appl. Math.*, 61(2):91–103, 1995.

[AS04] Scott Aaronson and Yaoyun Shi. Quantum lower bounds for the collision and the element distinctness problems. *J. ACM*, 51(4):595–605, 2004.

[AS11] Andris Ambainis and Xiaoming Sun. New separation between \$s(f)\$ and \$bs(f)\$. *CoRR*, abs/1108.3494, 2011.

[AT92] Noga Alon and Michael Tarsi. Colorings and Orientations of Graphs. *Combinatorica*, 12(2):125—-134, 1992.

[BB20] Gantsooj Batzaya and Gombodorj Bayarmagnai. A generalized Combinatorial Nullstellensatz for multisets. *European Journal of Combinatorics*, 83, 2020.

[BBBV97] Charles H Bennett, Ethan Bernstein, Gilles Brassard, and Umesh Vazirani. Strengths and weaknesses of quantum computing. *SIAM journal on Computing*, 26(5):1510–1523, 1997.

[BBC$^+$01a] Robert Beals, Harry Buhrman, Richard Cleve, Michele Mosca, and Ronald de Wolf. Quantum lower bounds by polynomials. *J. ACM*, 48(4):778–797, 2001.

[BBC$^+$01b] Robert Beals, Harry Buhrman, Richard Cleve, Michele Mosca, and Ronald de Wolf. Quantum lower bounds by polynomials. *J. ACM*, 48(4):778–797, 2001.

[BBDM21] Anurag Bishnoi, Simona Boyadzhiyska, Shagnik Das, and Tamás Mészáros. Subspace Coverings with Multiplicities. *CoRR*, abs/2101.11947, 2021.

[BBGK18a] Shalev Ben-David, Adam Bouland, Ankit Garg, and Robin Kothari. Classical lower bounds from quantum upper bounds. In *FOCS*, pages 339–349, 2018.

[BBGK18b] Shalev Ben-David, Adam Bouland, Ankit Garg, and Robin Kothari. Classical lower bounds from quantum upper bounds. In *FOCS*, pages 339–349, 2018.

[BCG$^+$20] Shalev Ben-David, Andrew M. Childs, András Gilyén, William Kretschmer, Supartha Podder, and Daochen Wang. Symmetries, graph properties, and quantum speedups. In *FOCS*, pages 649–660, 2020.

[BCH$^+$17] Adam Bouland, Lijie Chen, Dhiraj Holden, Justin Thaler, and Prashant Nalini Vasudevan. On the power of statistical zero knowledge. In *FOCS*, pages 708–719, 2017.

[BCPS18] Anurag Bishnoi, Pete L. Clark, Aditya Potukuchi, and John R. Schmitt. On Zeros of a Polynomial in a Finite Grid. *Combinatorics, Probability and Computing*, 27(3):310–333, 2018.

[Bd02] Harry Buhrman and Ronald de Wolf. Complexity measures and decision tree complexity: a survey. *Theoretical Computer Science*, 288(1):21–43, 2002.

[BDB20a] Shalev Ben-David and Eric Blais. A tight composition theorem for the randomized query complexity of partial functions. *arXiv:2002.10809*, 2020.

[BDB20b] Shalev Ben-David and Eric Blais. A tight composition theorem for the randomized query complexity of partial functions: Extended abstract. In *FOCS*, pages 240–246, 2020.

[BDBGM22] Shalev Ben-David, Eric Blais, Mika Göös, and Gilbert Maystre. Randomised composition and small-bias minimax. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 624–635. IEEE, 2022.

[BDG$^+$20] Andrew Bassilakis, Andrew Drucker, Mika Göös, Lunjia Hu, Weiyun Ma, and Li-Yang Tan. The power of many samples in query complexity. In *ICALP*, volume 168, pages 9:1–9:18, 2020.

[BDGKW20] Shalev Ben-David, Mika Göös, Robin Kothari, and Thomas Watson. When Is Amplification Necessary for Composition in Randomized Query Complexity? In *APPROX/RANDOM 2020*, volume 176, pages 28:1–28:16, 2020.

[BdW02] Harry Buhrman and Ronald de Wolf. Complexity measures and decision tree complexity: a survey. *Theoretical Computer Science*, 288(1):21–43, 2002.

[Bei93] R. Beigel. The polynomial method in circuit complexity. In *Proceedings of the Eigth Annual Structure in Complexity Theory Conference*, pages 82–95, 1993.

[Bei94] Richard Beigel. Perceptrons, pp, and the polynomial hierarchy. *Comput. Complex.*, 4:339–349, 1994.

[Bel15] Aleksandrs Belovs. Quantum algorithms for learning symmetric juntas via the adversary bound. *Comput. Complex.*, 24(2):255–293, 2015.

[BHMT02] Gilles Brassard, Peter Hoyer, Michele Mosca, and Alain Tapp. Quantum amplitude amplification and estimation. *Contemporary Mathematics*, 305:53–74, 2002.

[BHT17] Shalev Ben-David, Pooya Hatami, and Avishay Tal. Low-sensitivity functions from unambiguous certificates. In *ITCS*, volume 67, pages 28:1–28:23, 2017.

[BK18] Shalev Ben-David and Robin Kothari. Randomized query complexity of sabotaged and composed functions. *Theory Comput.*, 14(1):1–27, 2018.

[BKT19] Mark Bun, Robin Kothari, and Justin Thaler. Quantum algorithms and approximating polynomials for composed functions with shared inputs. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 662–678. SIAM, 2019.

[BNRdW07a] Harry Buhrman, Ilan Newman, Hein Röhrig, and Ronald de Wolf. Robust polynomials and quantum algorithms. *Theory Comput. Syst.*, 40(4):379–395, 2007.

[BNRdW07b] Harry Buhrman, Ilan Newman, Hein Röhrig, and Ronald de Wolf. Robust polynomials and quantum algorithms. *Theory Comput. Syst.*, 40(4):379–395, 2007.

[Bop97] Ravi B. Boppana. The average sensitivity of bounded-depth circuits. *Inf. Process. Lett.*, 63(5):257–261, 1997.

[BS09] Simeon Ball and Oriol Serra. Punctured Combinatorial Nullstellensätze. *Combinatorica*, 29(5):511 – 522, 2009.

[BS13] Aleksandrs Belovs and Robert Spalek. Adversary lower bound for the k-sum problem. In *ITCS*, pages 323–328, 2013.

[BT13] Mark Bun and Justin Thaler. Dual lower bounds for approximate degree and Markov-Bernstein inequalities. In *ICALP*, volume 7965, pages 303–314, 2013.

[BT22] Mark Bun and Justin Thaler. Approximate degree in classical and quantum computing. *Found. Trends Theor. Comput. Sci.*, 15(3-4):229–423, 2022.

[BVdW07] Harry Buhrman, Nikolai K. Vereshchagin, and Ronald de Wolf. On computation and communication with small bias. In *(CCC*, pages 24–32, 2007.

[BWY15] Eric Blais, Amit Weinstein, and Yuichi Yoshida. Partially symmetric functions are efficiently isomorphism testable. *SIAM J. Comput.*, 44(2):411–432, 2015.

[CDR86] Stephen A. Cook, Cynthia Dwork, and Rüdiger Reischuk. Upper and lower time bounds for parallel random access machines without simultaneous writes. *SIAM J. Comput.*, 15(1):87–97, 1986.

[CFGM12] Sourav Chakraborty, Eldar Fischer, David García-Soriano, and Arie Matsliah. Junto-symmetric functions, hypergraph isomorphism and crunching. In *CCC*, pages 148–158, 2012.

[CH20] Alexander Clifton and Hao Huang. On Almost k-Covers of Hypercubes. *Combinatorica*, 40:511 – 526, 2020.

[Cha11] Sourav Chakraborty. On the sensitivity of cyclically-invariant Boolean functions. *Discrete Mathematics and Theoretical Computer Science*, 13(4):51–60, 2011.

[Che16] Lijie Chen. Adaptivity vs. postselection, and hardness amplification for polynomial approximation. In *ISAAC*, volume 64 of *LIPIcs*, pages 26:1–26:12, 2016.

[CKM$^+$23a] Sourav Chakraborty, Chandrima Kayal, Rajat Mittal, Manaswi Paraashar, Swagato Sanyal, and Nitin Saurabh. On the composition of randomized query complexity and approximate degree. *Electron. Colloquium Comput. Complex.*, 1(1):37–46, 2023.

[CKM$^+$23b] Sourav Chakraborty, Chandrima Kayal, Rajat Mittal, Manaswi Paraashar, Swagato Sanyal, and Nitin Saurabh. On the composition of randomized query complexity and approximate degree. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2023, September 11-13, 2023, Atlanta, Georgia, USA*, volume 275 of *LIPIcs*, pages 63:1–63:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.

[CKM$^+$24] Sourav Chakraborty, Chandrima Kayal, Rajat Mittal, Manaswi Paraashar, and Nitin Saurabh. Approximate degree composition for recursive functions. In Amit Kumar and Noga Ron-Zewi, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2024, August 28-30, 2024, London School of Economics, London, UK*, volume 317 of *LIPIcs*, pages 71:1–71:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.

[CKP21] Sourav Chakraborty, Chandrima Kayal, and Manaswi Paraashar. Separations between combinatorial measures for transitive functions. *ArXiv*, 2021.

[CKP22] Sourav Chakraborty, Chandrima Kayal, and Manaswi Paraashar. Separations between combinatorial measures for transitive functions. In *ICALP*, volume 229, pages 36:1–36:20, 2022.

[CMS20] Arkadev Chattopadhyay, Nikhil S. Mande, and Suhail Sherif. The log-approximate-rank conjecture is false. *Journal of the ACM*, 67, 2020.

[CTUW14] Karthekeyan Chandrasekaran, Justin Thaler, Jonathan R. Ullman, and Andrew Wan. Faster private release of marginals on small databases. In *ITCS*, pages 387–402, 2014.

[DKSS13] Zeev Dvir, Swastik Kopparty, Shubhangi Saraf, and Madhu Sudan. Extensions to the Method of Multiplicities, with Applications to Kakeya Sets and Mergers. *SIAM J. Comput.*, 42(6):2305–2328, 2013.

[Dru11] Andrew Drucker. Block sensitivity of minterm-transitive functions. *Theoretical Computer Science*, 412(41):5796–5801, 2011.

[dW00] Ronald de Wolf. Nondeterministic quantum query and quantum communication complexities. *CoRR*, cs.CC/0001014, 2000.

[dW08] Ronald de Wolf. A note on quantum algorithms and the minimal degree of ε-error polynomials for symmetric functions. *Quantum Inf. Comput.*, 8(10):943–950, 2008.

[DY22] Benjamin E. Diamond and Amir Yehudayoff. Explicit exponential lower bounds for exact hyperplane covers. *Discrete Mathematics*, 345(11):113114, 2022.

[Fel91] William Feller. *An Introduction to Probability Theory and Its Applications*. Wiley Series in Probability and Statistics, 1991.

[GJ16] Mika Göös and T. S. Jayram. A composition theorem for conical juntas. In Ran Raz, editor, *31st Conference on Computational Complexity, CCC 2016, May 29 to June 1, 2016, Tokyo, Japan*, volume 50 of *LIPIcs*, pages 5:1–5:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.

[GJPW18a] Mika Göös, T. S. Jayram, Toniann Pitassi, and Thomas Watson. Randomized communication versus partition number. *ACM Trans. Comput. Theory*, 10(1), 2018.

[GJPW18b] Mika Göös, T. S. Jayram, Toniann Pitassi, and Thomas Watson. Randomized communication versus partition number. *ACM Transactions on Computation Theory*, 10(1):4:1–4:20, 2018.

[GKN23] Arijit Ghosh, Chandrima Kayal, and Soumi Nandi. Covering almost all the layers of the hypercube with multiplicities. *Discret. Math.*, 346(7):113397, 2023.

[GLSS19] Dmitry Gavinsky, Troy Lee, Miklos Santha, and Swagato Sanyal. A Composition Theorem for Randomized Query Complexity via Max-Conflict Complexity. In *ICALP*, volume 132, pages 64:1–64:13, 2019.

[GMSZ13] Yihan Gao, Jieming Mao, Xiaoming Sun, and Song Zuo. On the sensitivity complexity of bipartite graph properties. *Theoretical Computer Science*, 468:83–91, 2013.

[Gol20] Oded Goldreich. On (valiant's) polynomial-size monotone formula for majority. In *Computational Complexity and Property Testing - On the Interplay Between Randomness and Computation*, volume 12050 of *Lecture Notes in Computer Science*, pages 17–23. Springer, 2020.

[GPW18] Mika Göös, Toniann Pitassi, and Thomas Watson. Deterministic communication vs. partition number. *SIAM Journal on Computing*, 47(6):2435–2450, 2018.

[GSS16] Justin Gilmer, Michael E. Saks, and Srikanth Srinivasan. Composition limits and separating examples for some Boolean function complexity measures. *Combinatorica*, 36(3):265–311, 2016.

[HKP11] Pooya Hatami, Raghav Kulkarni, and Denis Pankratov. Variations on the sensitivity conjecture. *Theory Comput.*, 4:1–27, 2011.

[Hua19] Hao Huang. Induced subgraphs of hypercubes and a proof of the sensitivity conjecture. *Annals of Mathematics*, 190(3):949–955, 2019.

[JKS03] T. S. Jayram, Ravi Kumar, and D. Sivakumar. Two applications of information complexity. In *STOC*, pages 673–682, 2003.

[JRSW99] Stasys Jukna, Alexander A. Razborov, Petr Savický, and Ingo Wegener. On P versus NP cap co-np for decision trees and read-once branching programs. *Comput. Complex.*, 8(4):357–370, 1999.

[Kim13a] Shelby Kimmel. Quantum adversary (upper) bound. *Chicago Journal of Theoretical Computer Science*, 2013(4), April 2013.

[Kim13b] Shelby Kimmel. Quantum adversary (upper) bound. *Chicago Journal of Theoritical Computer Science*, 2013, 2013.

[KKMS08] Adam Tauman Kalai, Adam R. Klivans, Yishay Mansour, and Rocco A. Servedio. Agnostically learning halfspaces. *SIAM J. Comput.*, 37(6):1777–1805, 2008.

[KLS96] Jeff Kahn, Nathan Linial, and Alex Samorodnitsky. Inclusion-exclusion: Exact and approximate. *Comb.*, 16(4):465–477, 1996.

[KMR11] Géza Kós, Tamás Mészáros, and Lajos Rónyai. Some Extensions of Alon's Nullstellensatz. *Publ. Math. Debrecen*, 79:507–519, 2011.

[KN96] Eyal Kushilevitz and Noam Nisan. *Communication Complexity*. Cambridge University Press, 1996.

[Kom94] P. Komjáth. Partitions of Vector Spaces. *Periodica Mathematica Hungarica*, 28(3):187 – 193, 1994.

[KOS04] Adam R. Klivans, Ryan O'Donnell, and Rocco A. Servedio. Learning intersections and thresholds of halfspaces. *J. Comput. Syst. Sci.*, 68(4):808–840, 2004.

[KR12] Géza Kós and Lajos Rónyai. Alon's Nullstellensatz for Multisets. *Combinatorica*, 32(5):589–605, 2012.

[KS04] Adam R. Klivans and Rocco A. Servedio. Learning DNF in time $2^{\tilde{O}(n^{1/3})}$. *J. Comput. Syst. Sci.*, 68(2):303–318, 2004.

[KS06] Adam R. Klivans and Rocco A. Servedio. Toward attribute efficient learning of decision lists and parities. *J. Mach. Learn. Res.*, 7:587–602, 2006.

[KT16] Raghav Kulkarni and Avishay Tal. On fractional block sensitivity. *Chic. J. Theor. Comput. Sci.*, 2016, 2016.

[KV14] Pritish Kamath and Prashant Vasudevan. Approximate degree of and-or trees. 2014.

[Lee09] Troy Lee. A note on the sign degree of formulas. *CoRR*, abs/0909.4607, 2009.

[Leo13] Nikos Leonardos. An improved lower bound for the randomized decision tree complexity of recursive majority,. In *ICALP*, volume 7965, pages 696–708, 2013.

[Li21] Yaqiao Li. Conflict complexity is lower bounded by block sensitivity. *Theor. Comput. Sci.*, 856:169–172, 2021.

[LMR$^+$11] Troy Lee, Rajat Mittal, Ben W. Reichardt, Robert Spalek, and Mario Szegedy. Quantum query complexity of state conversion. In *FOCS*, pages 344–353, 2011.

[LR13] Troy Lee and Jérémie Roland. A strong direct product theorem for quantum query complexity. *Computational Complexity*, 22(2):429–462, 2013.

[LS17] Qian Li and Xiaoming Sun. On the sensitivity complexity of k-uniform hypergraph properties. In *STACS*, volume 66, pages 51:1–51:12, 2017.

[Mid04] G. Midrijanis. Exact quantum query complexity for total Boolean functions. *arXiv:quant-ph/0403168v2*, 2004.

[MNS$^+$16] Frédéric Magniez, Ashwin Nayak, Miklos Santha, Jonah Sherman, Gábor Tardos, and David Xiao. Improved bounds for the randomized decision tree complexity of recursive majority. *Random Struct. Algorithms*, 48(3):612–638, 2016.

[Mon14] Ashley Montanaro. A composition theorem for decision tree complexity. *Chicago Journal of Theoritical Computer Science*, 2014, 2014.

[MS15] Sagnik Mukhopadhyay and Swagato Sanyal. Towards better separation between deterministic and randomized query complexity. In *FSTTCS*, volume 45, pages 206–220, 2015.

[NC10] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010.

[Nis89] Noam Nisan. Crew prams and decision trees. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 327–335, 1989.

[NS94] Noam Nisan and Mario Szegedy. On the degree of Boolean functions as real polynomials. *Computational Complexity*, 4:301–313, 1994.

[NW95] Noam Nisan and Avi Wigderson. On rank vs. communication complexity. *Combinatorica*, 15(4):557–565, 1995.

[OWZ11] Ryan O'Donnell, John Wright, and Yuan Zhou. The fourier entropy-influence conjecture for certain classes of boolean functions. In Luca Aceto, Monika Henzinger, and Jirí Sgall, editors, *Automata, Languages and Programming*

- 38th International Colloquium, ICALP 2011, Zurich, Switzerland, July 4-8, 2011, Proceedings, Part I, volume 6755 of *Lecture Notes in Computer Science*, pages 330–341. Springer, 2011.

[Pat92] Ramamohan Paturi. On the degree of polynomials that approximate symmetric boolean functions. In *STOC*, page 468–474, 1992.

[Rei11a] Ben Reichardt. Reflections for quantum query algorithms. In *SODA*, pages 560–569. SIAM, 2011.

[Rei11b] Ben Reichardt. Reflections for quantum query algorithms. In Dana Randall, editor, *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 560–569. SIAM, 2011.

[RS12] Ben Reichardt and Robert Spalek. Span-program-based quantum algorithm for evaluating formulas. *Theory Comput.*, 8(1):291–319, 2012.

[Rub95] David Rubinstein. Sensitivity vs. block sensitivity of Boolean functions. *Combinatorica*, 15(2):297–299, 1995.

[Sak93] Michael E. Saks. *Slicing the hypercube*, page 211–256. London Mathematical Society Lecture Note Series. Cambridge University Press, 1993.

[Sav02] Petr Savický. On determinism versus unambiquous nondeterminism for decision trees. *Electron. Colloquium Comput. Complex.*, 9(009), 2002.

[She09] Alexander A. Sherstov. Approximate inclusion-exclusion for arbitrary symmetric functions. *Comput. Complex.*, 18(2):219–247, 2009.

[She11a] Alexander A. Sherstov. The pattern matrix method. *SIAM J. Comput.*, 40(6):1969–2000, 2011.

[She11b] Alexander A. Sherstov. Strong direct product theorems for quantum communication and query complexity. In *STOC*, pages 41–50, 2011.

[She12] A. A. Sherstov. Strong direct product theorems for quantum communication and query complexity. *SIAM J. Comput.*, 41(5):1122–1165, 2012.

[She13a] Alexander A. Sherstov. Approximating the AND-OR tree. *Theory Comput.*, 9:653–663, 2013.

[She13b] Alexander A. Sherstov. The intersection of two halfspaces has high threshold degree. *SIAM J. Comput.*, 42(6):2329–2374, 2013.

[She13c] Alexander A. Sherstov. The intersection of two halfspaces has high threshold degree. *SIAM Journal on Computing*, 42(6):2329–2374, 2013.

[She13d] Alexander A. Sherstov. Making polynomials robust to noise. *Theory Comput.*, 9:593–615, 2013.

[She13e] Alexander A. Sherstov. Making polynomials robust to noise. *Theory Comput.*, 9:593–615, 2013.

[Shi02] Yaoyun Shi. Approximating linear restrictions of Boolean functions. Technical report, 2002.

[Sni85] Marc Snir. Lower bounds on probabilistic linear decision trees. *Theoretical Computer Science*, 38:69–82, 1985.

[STT12] Rocco A. Servedio, Li-Yang Tan, and Justin Thaler. Attribute-efficient learning and weight-degree tradeoffs for polynomial threshold functions. In *COLT*, volume 23, pages 14.1–14.19. JMLR.org, 2012.

[Sun07] Xiaoming Sun. Block sensitivity of weakly symmetric functions. *Theoretical Computer Science*, 384(1):87–91, 2007.

[Sun11] Xiaoming Sun. An improved lower bound on the sensitivity complexity of graph properties. *Theoretical Computer Science*, 412(29):3524–3529, 2011.

[SW86a] Michael Saks and Avi Wigderson. Probabilistic boolean decision trees and the complexity of evaluating game trees. In *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, pages 29–38, 1986.

[SW86b] Michael E. Saks and Avi Wigderson. Probabilistic boolean decision trees and the complexity of evaluating game trees. In *FOCS*, pages 29–38, 1986.

[SW20] Lisa Sauermann and Yuval Wigderson. Polynomials that vanish to high order on most of the hypercube. *Journal of the London Mathematical Society*, to appear, 2020.

[SYZ04] Xiaoming Sun, Andrew Chi-Chih Yao, and Shengyu Zhang. Graph properties and circular functions: How low can quantum query complexity go? In *CCC*, pages 286–293, 2004.

[Tal13] Avishay Tal. Properties and applications of Boolean function composition. In *ITCS*, pages 441–454, 2013.

[Tal14] Avishay Tal. Shrinkage of De Morgan formulae by spectral techniques. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 551–560. IEEE, 2014.

[Tur84] György Turán. The critical complexity of graph properties. *Information Processing Letters*, 18(3):151–153, 1984.

[TUV12] Justin Thaler, Jonathan R. Ullman, and Salil P. Vadhan. Faster algorithms for privately releasing marginals. In *ICALP*, volume 7391 of *Lecture Notes in Computer Science*, pages 810–821. Springer, 2012.

[Val84] L.G Valiant. Short monotone formulae for the majority function. *Journal of Algorithms*, 5(3):363–366, 1984.

[Ven22] S. Venkitesh. Covering Symmetric Sets of the Boolean Cube by Affine Hyperplanes. *The Electronic Journal of Combinatorics*, 29(2), 2022.

[vZGR97] Joachim von Zur Gathen and James R. Roche. Polynomials with two values. In *Combinatorica*. Springer, 1997.