Algorithms for property testing and related problems

Yonatan Goldhirsh

Algorithms for property testing and related problems

Research Thesis

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy

Yonatan Goldhirsh

Submitted to the Senate
of the Technion — Israel Institute of Technology
Iyar 5775 Haifa April 2015

This research was carried out under the supervision of Prof. Eldar Fischer, in the Faculty of Computer Science.

Some results in this thesis have been published as articles by the author and research collaborators in conferences and journals during the course of the author's doctoral research period, the most up-to-date versions of which being:

Sourav Chakraborty, Eldar Fischer, Yonatan Goldhirsh, and Arie Matsliah. On the power of conditional samples in distribution testing. In Robert D. Kleinberg, editor, *ITCS*, pages 561–580. ACM, 2013.

Eldar Fischer, Yonatan Goldhirsh, and Oded Lachish. Testing formula satisfaction. In Fedor V. Fomin and Petteri Kaski, editors, *SWAT*, volume 7357 of *Lecture Notes in Computer Science*, pages 376–387. Springer, 2012.

Eldar Fischer, Yonatan Goldhirsh, and Oded Lachish. Partial tests, universal tests and decomposability. In Moni Naor, editor, *ITCS*, pages 483–500. ACM, 2014.

Acknowledgements

I would like to thank Prof. Eldar Fischer, my advisor, for his encouragement and support throughout the years of my work. It was an exciting and rewarding period, and I had the good fortune of having Eldar as my guide to the world of research. I am also deeply indebted to my friend Ami Paz, who played a crucial role in getting me to actually write this thesis. Ami picked me up when I was down and helped me transform the thesis from an insurmountable obstacle to a done project.

This work is dedicated to my beloved wife, Sarit, who is always there for support during hard times, and for sharing the joys of success.

The Technion's funding of this research is hereby acknowledged.

Contents

Abstract **1**

1 Introduction **3**

1.1 Property testing . . . 4

1.2 Summary of results . . . 7

1.2.1 Testing formula satisfaction . . . 7

1.2.2 Testing tree coloring properties . . . 7

1.2.3 Partial property testing . . . 8

1.2.4 Distribution testing with conditional samples . . . 9

2 Preliminaries **11**

2.1 Directed ordered trees . . . 13

3 Testing Formula Satisfaction **15**

3.1 Introduction . . . 15

3.2 Preliminaries . . . 16

3.2.1 Formulas, evaluations and testing . . . 16

3.2.2 Basic formula simplification and handling . . . 18

3.2.3 Observations about subformulas and distance . . . 20

3.2.4 Heavy and light children in general gates . . . 21

3.3 Upper bound for general bounded arity formula . . . 21

3.4 Estimator for monotone formula of bounded arity . . . 25

3.5 Quasi-polynomial upper bound for basic-formulas . . . 29

3.5.1 Critical and important . . . 29

3.5.2 Algorithm . . . 31

3.6 The computational complexity of the testers and estimator . . . 34

3.7 The untestable formulas . . . 35

3.7.1 The 4-valued formula . . . 35

3.7.2 Two distributions . . . 37

3.7.3 Proving non-testability . . . 38

3.7.4 An untestable 5-valued monotone formula . . . 38

4 Testing Tree Coloring Properties **41**
4.1 Introduction . . . 41
4.1.1 Topological subtrees . . . 42
4.2 Preliminaries and statement of main result . . . 43
4.3 Overview . . . 44
4.4 Testing for a family of forbidden topological subtrees in bounded degree trees . . 46
4.4.1 AllColors . . . 47
4.4.2 MissingColor . . . 50
4.4.3 Query complexity . . . 52
4.4.4 Reduction from a general tree to a full tree . . . 53
4.4.5 Extending AllColors to trees with unsorted children . . . 54
4.5 A lower bound for testing against a family of forbidden induced subtrees in a balanced binary tree . . . 55

5 Partial Property Testing **57**
5.1 Introduction . . . 57
5.1.1 Related work . . . 61
5.2 General themes . . . 63
5.2.1 Proving a bound against non-adaptive algorithms . . . 63
5.2.2 Proving a bound against adaptive algorithms . . . 64
5.2.3 Testing decomposable properties through universal testing . . . 65
5.2.4 A non-testable property that is decomposable to testable ones . . . 66
5.3 Preliminaries . . . 66
5.4 Nonadaptive lower bound . . . 68
5.5 Adaptive lower bound . . . 71
5.6 Properties with a proximity oblivious 2-test decomposition . . . 76
5.7 Properties with a proximity oblivious q-test decomposition . . . 80
5.8 Highly decomposable properties . . . 85

6 Distribution Testing with Conditional Samples **89**
6.1 Introduction . . . 89
6.1.1 Some motivating examples . . . 90
6.1.2 Informal description of results . . . 91
6.1.3 Related work . . . 93
6.2 Preliminaries . . . 95
6.2.1 The conditional distribution testing model . . . 95
6.2.2 Tools from previous works . . . 96
6.3 Adaptive testing for uniformity and identity . . . 97
6.3.1 Testing uniformity . . . 97
6.3.2 Testing identity to a known distribution . . . 99
6.4 Non-adaptive testing for uniformity and identity . . . 102

6.4.1 Testing uniformity . . . 102
6.4.2 Testing identity to a known distribution . . . 104
6.5 Explicit persistent samplers . . . 105
6.5.1 Ratio trees and reconstituted distributions . . . 105
6.5.2 Distribution samplers and learning . . . 107
6.6 Testing any label-invariant property . . . 108
6.6.1 Bucketing and approximations . . . 109
6.6.2 From bucketing to learning . . . 111
6.7 Lower bounds for label invariant properties . . . 113
6.7.1 Preliminary definitions . . . 113
6.7.2 Uniformity has no constant sample non-adaptive test . . . 116
6.7.3 A label-invariant property with no constant sample adaptive test . . . 119
6.8 A lower bound for testing general properties of distributions . . . 124
6.8.1 The reduction lemma . . . 124

7 Open Questions **127**
7.1 Testing formula satisfaction . . . 127
7.2 Partial property testing . . . 127
7.3 Distribution testing with conditional samples . . . 128

Hebrew Abstract **i**

Abstract

In this thesis we study property testers from an algorithmic point of view. Namely, property testers with a substantial algorithmic component, and the power of algorithmic approaches to property testing problems.

First we study property testing in the massively parametrized model. The first problem we study in this model is testing read once formula satisfaction, in several flavors. For general formulas with gates of bounded arity we give a constant query property tester. For formulas with monotone gates of bounded arity we give a constant query complexity distance estimator. Finally, for and/or formulas we give a property tester with query complexity which is quasi-polynomial in the distance parameter. On the lower bound front, lower bounds against testing read once formula satisfaction over an alphabet of size 4, and against testing read once formula satisfaction for formulas with monotone gates over an alphabet of size 5. We proceed to study tree coloring properties, specifically those defined by freeness from a forbidden family of colored topological subtrees. We give constant query complexity algorithms for this problem, as well as a super constant lower bound for the case of freeness from a forbidden family of colored induced subtrees.

Then we introduce the notion of partial property testing, and the decomposition of a property into partially testable subproperties. A property P is P'-partially testable if there is a property tester that accepts inputs in P' and rejects inputs far from P. We show that there exists a property which is untestable, but has a decomposition into a few subproperties, for each it is partially testable. On the other hand we show that there exist untestable properties for which any decomposition into partially testable subproperties has an exponential number of sets. For this we develop new, entropy based techniques for proving property testing lower bounds. To complete the picture, we show that if we have a decomposition into a few subsets for which we have proximity oblivious testers then we can obtain a sublinear query complexity tester for the entire property. We do this by developing a “universal tester” and prove it tests such properties. This also resolves an open question regarding “sample based testers” posed by Goldreich and Ron [GR13].

Distribution testers are always non-adaptive, since all samples are obtained in the same manner and there is no place for an algorithmic decision. This is changed by introducing the model of distribution testing with conditional samples. In this model, the testing algorithm may obtain samples conditioned on any subset of the universe. This creates a much richer model, and in particular enables adaptive algorithms. We show that many problems are easier to test with conditional samples, such as uniformity, testing closeness to a known distribution, and any label invariant property of distributions. On the other hand, we show that there still exist problems in this model which require a linear number of samples. Finally, we also show a separation between adaptive and non-adaptive distribution testing algorithms with conditional samples.

Chapter 1

Introduction

Classic algorithmic theory considers run time as the most important measure of algorithmic efficiency. An algorithm is considered efficient if its run time, or time complexity, grows at most polynomially, as was first suggested in the seminal work of Edmonds [Edm65]. In recent decades, the amount of data that algorithms are required to process is growing in a breathtaking pace. Google announced in 2012 that its Gmail service has 425 million active users [Gmaa]. Keeping in mind that every user is currently allocated at least 15GB of storage [Gmab], one can quickly see that any polynomial time computation over this dataset is infeasible. In fact, any computation that attempts to read this entire dataset in a straightforward manner is beyond our reach.

These developments have lead to research of algorithms for massive data sets. Many frameworks and approaches have been suggested and implemented: A streaming algorithm [Mut05] only reads the data set once, in some simple order, and only maintains a very small memory footprint throughout the computation. The Map-Reduce framework [DG08] is based on creating very localized computations that lend themselves to extreme parallelism. Sublinear time algorithms, which include most property testing algorithms, take a formalistic approach to the issue — if the data sets are so big, let us design algorithms with time complexity that increases less than linearly in the size of the input.

This approach does face a significant challenge compared to the other approaches: if computation time increases more slowly than the size of the input, then the computation cannot expect to read the entire input. This is where the terms "sublinear algorithms" and "property testing" diverge. While sublinear algorithms are concerned with sublinear time complexity, property testing focuses on the problem of decision making when reading only a negligible portion of the input.

When restricting our attention to algorithms that cannot read the entire input, we are quickly faced with some deep issues of problem definition. Suppose that our input is a binary string, and we want an algorithm that accepts the string if and only if it is the all-1 string. An algorithm that does not read the entire input can never be guaranteed to answer correctly, and so we must introduce a notion of *approximation*. This notion will be tied to an approximation parameter $1 > \epsilon > 0$, and we will only require that the algorithm rejects a string if at least an ϵ fraction of its inputs must be changed for it to be in the set of allowed strings (in our example, this set is the singleton set comprised of the all-1 string). But even if we allow such an approximation, we still have a problem. We expect a property testing algorithm to read a negligible portion of the input. This means that if an adversary knows which bits will be read, it can cause the algorithm to accept inputs it should reject. For this reason property testing algorithms must be *probabilistic*, and their output will only be required to be correct *with high probability*.

To reiterate, we have some universe of objects U, each of size n, and a property that some of these objects posses $P \subseteq U$. In the example above, if we choose some n, then $U = \{0,1\}^n$ and $P = \{1111\ldots1\}$. We also have some notion of *distance* over objects in U, where a fitting choice for the example above is the normalized hamming distance, and a notion of *query* into the objects in U, where in the example above it would just be reading a requested bit. A randomized algorithm A is an *ϵ-property tester* for P with *query complexity* q if given an input x, it performs at most q queries into x, and then *accepts* with probability at least $2/3$ if $x \in P$, and *rejects* with probability at least $2/3$ if the distance between x and any member of P is at least ϵ.

We will make several distinctions between different property testing algorithms. An algorithm will be *one-sided* if it accepts all inputs $x \in P$ with probability 1, and otherwise it will be *two-sided*. An algorithm will be *non-adaptive* if the set of queries it makes only depends on the input size and its internal randomness (and not on the answers to previous queries), and it will be *adaptive* otherwise.

1.1 Property testing

A large part of property testing research concerns itself with algorithms of *constant* query complexity, that is, query complexity that only depends on the parameter ϵ. Therefore, when we use terms such as "testable" or "has a property tester", we implicitly mean "with constant query complexity".

The notion of property testing was first defined by Rubinfeld and Sudan [RS96] in an algebraic setting, though similar problems were addressed by Blum, Luby and Rubinfeld [BLR93]. They concerned themselves with questions like "is a given function $f : \mathbb{F}^n \to \mathbb{F}$ linear?" or more generally "is it a low degree polynomial?". Such questions have many applications for error correcting codes. The original application that Rubinfeld and Sudan had in mind was quick testing of program correctness. The overarching theme in this work was showing that a property has a *robust local characterization*, meaning that if we have a function $f : \{0,1\}^n \to \{0,1\}$ such that with high probability when picking $x \in \{0,1\}^n$ and $y \in \{0,1\}^n$, we have that $f(x) + f(y) = f(x+y)$, then this function cannot be far from a linear function. This naturally lends itself to a one-sided, nonadaptive property testing algorithm. Moreover, this algorithm is extremely simple, just sampling and checking instances of this local characterization of the property. The generality of this approach was expanded upon in later works on the role of invariance in algebraic property testing [KS08, Sud11, BGS13, BFH$^+$13]

Later, Goldreich, Goldwasser and Ron [GGR98] introduced the notion of *combinatorial property testing*, that is, property testing for combinatorial objects, chiefly graphs. Goldreich, Goldwasser and Ron introduced the *dense graph model*, where we see a graph as being represented by its adjacency matrix. The possible queries to make are of the sort "are the vertices u and v adjacent?", and the distance between two graphs is simply the normalized hamming distance between their respective adjacency matrices. Note that in this model, any two graphs with $o(V^2)$ edges are always close to one another, and for this reason this model is only useful for discussing dense graphs. Goldreich, Goldwasser and Ron studied properties such as being bipartite, k-colorable or having a ρ-clique. They also noted that algorithms for all of the problems considered in their paper can be derived from a general algorithm for testing whether a graph has a partition of the vertices into sets of certain size range with lower and upper bounds on the number of edges between them. This journey culminated in the papers of Alon et. al. [AFNS09] and Borgs et. al. [BCL$^+$06] which proved that in fact, testable dense graph properties can be characterized by the existence of regular partitions.

Algebraic property testing and property testing in the dense graph model use the symmetry of the underlying models to craft concise and elegant testing algorithms. Other models for property testing problems require a "messier", more algorithmic approach. A prime examples is the *bounded-degree graph model*. In this model, introduced by Goldreich and Ron [GR97], we again test properties of graphs, but this time a graph of n vertices and maximal degree d is represented by an $n \times d$ array, where the (v, i) entry is the ith neighbor of the vertex v. This models lends itself to queries such as "who is the ith neighbor of v", but usually also allows queries of the form "how many neighbors does v have?". Usually d is seen as a constant. This model has seen a lot of fruitful research, including algorithms for bipartiteness [GR99], expansion [CS10] and other problems. Unlike the dense graph model, most of the algorithms in the bounded-degree model have query complexity sublinear in n, but not constant. Also, there is little use for the approach of sampling a set of vertices and considering the induced subgraph, as it is most likely empty. Therefore, most algorithms take a more algorithmic approach using various graph searching algorithms, from BFS and DFS through random walks to other ad-hoc approaches.

The property testing approaches described above assume that the algorithm is completely ignorant of the input and its structure. In many scenarios this is not the case. Consider a driving directions server — it has full knowledge of the road network, which is essentially immutable. On the other hand, it cannot expect to have full knowledge of traffic congestion in every road, since this data changes with high frequency. Another example is a hardware verification program. It has full knowledge of the structure of the verified hardware, but given a very large input, it might not want to run all of it through the hardware, and only extract small interesting bits out of it. The *massively parametrized model* gives a framework to describe property testing approaches to such problems. In this model, there is an underlying combinatorial structure S, and the universe is actually a universe of functions $f : S \to D$. This means that the properties discussed are properties of such functions.

The first work to consider the massively parameterized model, albeit implicitly, is that of Newman [New02], which proved testability of properties defined by bounded-width read-once branching programs. Note that for an input of n bits, a branching program of width w has

$O(wn)$ states and transitions. The algorithm given by Newman is given a branching program for a property P, and uses it in a testing algorithm for the property of being accepted by the program. That is, the algorithm has full knowledge of an object as big as the input. Another work where the concept of a massively parametrized model was used implicitly was that of Fischer et. al. [FLN$^+$02] which studied monotonicity over general posets. Here the algorithm is given a representation of the poset as a DAG, and has to decide using queries whether a given function is monotone relative to this poset. Again, the representation of the DAG can be as big as the input, but while the algorithm queries the input, it has full knowledge of the DAG. Another important work in the same spirit is that of Halevy et. al. [HLNT07], where they studied properties defined by constraint graphs. In a constraint graph every edge is labeled by a boolean variable, and every vertex is a boolean predicate on its incident edges. An assignment to the edge variables is satisfying if all of the vertex predicates are satisfied. A property testing algorithm in this setting has full knowledge of the graph structure and the vertex predicates, and only queries the variable values. Again, the algorithm has full knowledge of a structure that is at least as big as the input.

The *orientation model* is a well studied massively parametrized model. In it the underlying structure is some undirected graph G, and the functions define orientation on its edges. This means that the properties considered are properties of possible orientations of a given graph. This also means that a property testing algorithm has full knowledge of the underlying graph G, and only queries edges for their orientation. This model in fact originated in the work of Halevy et. al. [HLNT07] discussed above, since a boolean variable on an edge can be seen as assigning it a direction. It was later studied in several works, such as that of Fischer et. al. [FLM$^+$12] which studied the property of the orientation being Eulerian, and that of Chakraborty et. al. [CFL$^+$07], which considered the property of the existence of a directed path between two given vertices. There are other instances of the massively parameterized model, such as the tree coloring model [FY11] (also considered in this thesis), further work on branching programs [FNS04], graph isomorphism where one graph is given in advance [Fis05] and others. Also see the survey by Newman [New10]. Such algorithms cannot perform any blind sampling, as the queries they perform are intrinsically related to the structure of the underlying combinatorial structure. Thus they must explore this structure in tandem with performing queries to the unknown function.

Another model which is studied in this thesis is distribution property testing. While strictly speaking it is not an instance of property testing, it is a spiritual relative. In *distribution-property testing*, the goal is to distinguish the case where the samples come from a distribution that has a certain property $\mathcal{P}$ from the case where the samples come from a distribution that is far, in the variation distance, from any distribution that has the property $\mathcal{P}$ (the variation distance between two distributions μ and μ' over a common set B is $\frac{1}{2}\sum_{i\in B}|\Pr_\mu[i] - \Pr_{\mu'}[i]|$, which is equal to the maximum difference in probability between the distributions for any possible event). In the traditional setting no access is provided to the distribution apart from the ability to take independent samples, and the two cases should be distinguished using as few samples as possible. There are several natural distribution properties that were studied in this context: testing whether a distribution is uniform [GR11], testing identity between distributions (taking samples from both) [BFR+10, LRR10], testing whether a joint distribution is independent (a product of two distributions) [BFR+10] and more. Some useful general techniques have also been designed to obtain nearly tight lower bounds on various distribution-property testing problems [Val11]. Other tightly related works study the problems of estimating various measures of distributions, such as entropy [BDKR05, GMV09] or support size [RRSS09].

For more information on property testing and it's various branches, see the surveys [Fis04a, Gol10a, Ron08, Ron09, Gol10b].

1.2 Summary of results

This thesis is concerned with property testing problems where there is a substantial algorithmic component. In this section we will give an informal overview of our main results.

1.2.1 Testing formula satisfaction

An important family of problems in the massively parameterized model is satisfaction of nonuniform computational models. This includes the branching programs studied by Newman [New02] and the constraint graphs studied by Halevy et. al. [HLNT07]. Continuing in this line of research, we consider the problem of testing read-once formula satisfaction in Chapter 3.

When testing read-once formula satisfaction the massive parameter is a formula, given as a directed rooted tree where the inner nodes are labeled as logical gates, and the leaves are labeled as input variables. The tested property then is that of a given input satisfying the formula. We give several algorithms of varying generality. First, an algorithm to test satisfaction of read-once formulas of general boolean gates. Second, an algorithm to estimate the distance of a given assignment to satisfying a read-once formulas with monotone gates. Lastly, a more efficient algorithm to test satisfaction of read-once and/or formulas.

Finally, we give a lower bound for cases involving non-boolean values. We give two lower bounds, the first shows that read-once formula satisfaction over an alphabet of size 4 cannot be tested with a constant query complexity, and the second showing that even if we require all the gates to be monotone, read-once formula satisfaction over an alphabet of size 5 cannot be tested with a constant query complexity.

1.2.2 Testing tree coloring properties

Tree coloring properties are another instance of the massively parameterized model. In this case the underlying structure is a rooted, directed tree T, and the functions whose properties we test are coloring functions over the vertices. Probably the first instance of this model was in the context of monotonicity testing, in the work of Fischer et. al. [FLN+02].

A very general problem in this setting, is the problem of testing against forbidden topological subtrees. In this problem the property is defined by a set of forbidden colored trees $\mathcal{F}$, and a coloring C has the property if it does not contain any of these forbidden trees as a colored topological subtree. An exact definition of a colored topological subtree is found in Section 4.2, informally it means that there is a mapping from a forbidden tree $t \in \mathcal{F}$ to T, where vertices are mapped to other vertices of the same color, and edges are mapped to disjoint paths. This problem was previously considered by Yahalom [Yah12], which developed constant query complexity algorithms for the case where there is only one forbidden topological subtree. In this thesis we give an algorithm for testing freeness from a finite family of forbidden topological subtrees, with constant query complexity. Note that in general, testability is not preserved under property intersection, so moving from a single forbidden tree to a family of such trees is non-trivial. The basis of the algorithm is in a divide and conquer approach, with careful handling of ramsey-like phenomena and the kind of transformation steps that can be done on the forbidden family to "break it down".

Finally, we give a family of forbidden subtrees, such that freeness from which, as a colored induced subtree, cannot be tested in a constant query complexity.

1.2.3 Partial property testing

Let P be some property, and $P' \subseteq P$. We say that P *is* P'*-partially testable* if there exists a probabilistic algorithm that accepts inputs in P' with high probability, and rejects inputs far from P with high probability. In essence, this is a generalization of the usual definition of property testing. We initiate the study of this model in Chapter 5.

First, we give a construction, based on error correcting codes, of a property P such that any subproperty P' for which P is P'-partially testable is very small. This is based on introducing several new techniques for analyzing property testing algorithms and proving lower bounds for query complexity.

One of the reasons that the notion of partial testing is interesting, is that many untestable properties can be decomposed into relatively few subproperties, for which they are partially testable. In particular, the construction described above actually shows that for some untestable properties, any such decomposition must be to many sets.

Another way to view the idea of decomposition into partially testable subproperties is as "property testing with a proof", where the testing algorithm is given a proof π that it can read in its entirety. The requirement is that for every input with the property, there is a proof that causes the tester to accept with high probability, and for any input that is far from the property, the tester will reject it with high probability no matter the proof. The correspondence between the two definitions is that a proof can be seen as specifying a subpropery in the decomposition. The notion of property testing with a proof was concurrently introduced by Gur and Rothblum [GR15].

Next, we restrict our attention to proximity oblivious algorithms. Informally, these are property testers which are based on a "mini-algorithm" with query complexity which is an absolute constant (independent of ϵ), with success probability that depends on ϵ, and the property testing algorithm is based on performing several repetitions of this mini-algorithm (the number of repetitions depends on ϵ). We show that if a property can be decomposed into a few sets for which it is partially testable with a proximity oblivious algorithm, then there is also a sublinear algorithm for testing the entire property. We give two flavors of this results, a more efficient one where the proximity oblivious algorithm uses 2 queries, and a less efficient one for any constant number of queries. This shows a general framework for creating sublinear query complexity property testers by developing testers for subproperties of it. The algorithm is created by converting a proximity oblivious property tester to a *sample-based property tester* as defined by Goldreich and Ron [GR13], thus partially answering an open question posed by them.

Finally, we return our attention to the general case of finding decompositions, and construct a property that is hard to test, but has excellent decomposition into relatively few subproperties for which it is partially testable.

1.2.4 Distribution testing with conditional samples

On its face, distribution testing seems like the least algorithmic of the various branches of property testing. Since the algorithm just obtains independent samples, there is little for it to decide other than the number of samples obtained and the final acceptance criterion. While most previous works are focused on the ordinary sampling oracle, other stronger oracles were considered too. A major reason is that the number of required samples, while sublinear, is still very large in the original model. The most notable example of a strong oracle is the one from [BDKR05], that also allows querying the exact probability weight of any element from the domain. Another research direction involved restricting the problem further, for example by adding the promise of the distribution being monotone [BKR04].

In this thesis we initiate the study of a new model, where the algorithm can condition its samples on being in a certain subset S of the universe. This creates a wealth of algorithmic possibilities for the algorithm, including the possibility of creating adaptive algorithms. A very similar model was proposed and studied independently by [CRS14].

We give several results in this model. First, we give adaptive and non-adaptive algorithms for testing uniformity and identity to a known distribution. As expected, the adaptive algorithms are of better query complexity. Second, we give a general algorithm to learn distributions using an explicit sampling oracle, in the sense of being able to produce samples and answer probability questions for an approximation of the distribution. Using this, we also develop testing algorithms for any label-invariant property. Many such properties were studied in the literature, such as having a small support [RRSS09].

Finally, we give a comprehensive collection of lower bounds. First, a lower bound showing that uniformity has no constant sample complexity non-adaptive test, separating the strength of adaptive and non-adaptive algorithms in this model. Second, we show that there exist label-invariant properties with no constant sample complexity adaptive test, showing that the general problem is harder than uniformity. Lastly, we show a general mechanism for transforming lower bounds for properties of boolean strings into lower bounds in the conditional sampling model.

Chapter 2

Preliminaries

We use $[k]$ to denote the set $\{1, \ldots, k\}$. We will also use the following Chernoff-type inequality, which appears as Theorem A.1.11 and Theorem A.1.13 in [AS08]:

Lemma 2.0.1. *Let $p_1, \ldots, p_n \in [0,1]$, $X_1, \ldots, X_n$ be fully independent random variables with $\Pr[X_i = 1 - p_i] = p_i$ and $\Pr[X_i = -p_i] = 1 - p_i$, and let $p = \frac{1}{n}\sum_{i=1}^{n} p_i$ and $X = \sum_{i=1}^{n} X_i$. Then $\Pr[|X| > a] < 2\exp(-a^2/2pn)$.*

When using this lemma we interpret $X + pn = \sum_{i=1}^{n}(X_i + p_i)$ as the number of successes in n independent trials where the probability of success in the ith trial is p_i.

Let us now introduce a general definition of property testing.

Definition 2.0.2 (Property Tester)**.** Let $\{U_n\}_{n=1}^{\infty}$ be a sequence of *universes* of objects, indexed by n. Assume that there is some notion of *query* defined for the objects of the universe, and a measure of distance $d_n : U_n \times U_n \to [0,1]$. Let $\{P_n\}_{n=1}^{\infty}$ be a sequence of *properties* where for each m, $P_m \subseteq U_m$. For a real *approximation parameter* $\epsilon \in [0,1]$ and a real *confidence parameter* $\delta \in [0,1]$, an (ϵ, δ)*-tester* with query complexity $q \in \mathbb{N}$ is a (possibly randomized) algorithm A such that for any n:

- For an input $x \in P_n$, A accepts with probability at least $1 - \delta$.
- For an input $x \in U_n$, such that for any $y \in P_n$, $d_n(x,y) \geq \epsilon$, A rejects with probability at least $1 - \delta$.
- For any input, A performs at most q queries.

If furthermore for an input $x \in P_n$, A accepts with probability 1, then we say that the algorithm is *one-sided*, and otherwise we say that it is *two-sided*. If all of the queries performed by the algorithm can be determined before any of them are made, then the algorithm is said to be *non-adaptive*, and otherwise it is *adaptive*.

While the generality in the above definition is mathematically satisfying, we will mostly concern ourselves with cases where the universes are boolean strings or combinatorial objects. In these cases the notion of a query corresponds to unveiling the value of a certain index, existence of an edge etc. Additionally, it will be cumbersome and unenlightening to discuss sequences of properties and universes, and we will usually fix n for the discussion, without loss of generality. For the rest of this section we will limit the definition to the case of boolean strings when it is more convenient to do so.

Definition 2.0.3 (Normalized Hamming Distance)**.** For $x, y \in \{0,1\}^n$, the *normalized hamming distance* between them is $d(x,y) = \frac{1}{n}|\{i \in [n] | x_i \neq y_i\}|$, that is, the fraction of indexes on which the strings disagree. For $S \subseteq \{0,1\}^n$, we define $d(x,S) = \min_{y \in S} d(x,y)$. When $d(x,S) \geq \epsilon$, we say that x is *ϵ-far* from S, otherwise we say that it is *ϵ-close* to S.

In each chapter we will introduce a more specific notion of testing suitable to the universe discussed. In cases where we do not discuss boolean strings, a suitable notion of distance will also be introduced, but it will always be similar to the normalized hamming distance.

We proceed with some general notions relevant to one-sided algorithms:

Definition 2.0.4 (0-witness)**.** A 0-witness for a boolean function $f : \{0,1\}^n \to \{0,1\}$ is a subset of coordinates $W \subseteq [n]$, which is minimal (by inclusion) amongst all subsets for which there exists an assignment $\sigma : W \to \{0,1\}$, such that for every $x \in \{0,1\}^n$ which agrees with σ (that is, for all $i \in W$, we have that $x_i = \sigma(i)$) we have that $f(x) = 0$.

This notion is important because of the following observation, in which it is useful to reinterpret the function f from the above definition as an indicator function for a set P:

Observation 2.0.5. Let A be a one-sided tester for a property P. A cannot reject unless the queries it made contain a 0-witness.

This simple observation proves very useful for proving lower bounds against one-sided testers.

We will now briefly discuss the main technique for proving lower bounds against two-sided testers. First, let us introduce a definition of distance between distribution, which will also be useful to use when testing properties of distributions:

Definition 2.0.6. [Variation Distance] Let p and q be two distributions over the domain $\mathcal{D}$. The *variation distance* between p and q is defined to be

$$d_{TV}(p,q) = \frac{1}{2} \sum_{i \in \mathcal{D}} |p(i) - q(i)|.$$

We will follow the description of Yao's method from Fischer [Fis04b]. The basic method is that instead of trying to find a worst case input for every possible randomized algorithm, we find a distribution over inputs such that every deterministic algorithm will fail with probability greater than δ on an input taken according to this distribution. Let U be our universe of inputs, where every input is a function $f : D \to \{0,1\}$ for some domain D, and $P \subseteq U$ be the property.

Definition 2.0.7 (Restrictions, Definition 4 in [Fis04b])**.** Let μ be a distribution over inputs, and let $Q \subseteq D$. The restriction $\mu|_Q$ of μ to Q is the distribution defined over functions of the type $g : Q \to \{0,1\}$, that results from choosing a function $f : D \to \{0,1\}$ according to μ, and setting g to be $f|_Q$, the restriction of f to Q.

We can now introduce the version of Yao's Lemma for property testing algorithms.

Lemma 2.0.8 (Yao's Lemma, Lemma 8.1 in [Fis04b]). *Suppose that there exists a distribution μ_P on inputs from U that satisfy a property P, and a distribution μ_N on inputs that are ϵ-far from satisfying the property, and suppose furthermore that for any $Q \subseteq D$ of size q, the variation distance between $\mu_P|_Q$ and $\mu_N|_Q$ is less than δ. Then it is not possible for a non-adaptive algorithm making q (or less) queries to be an (ϵ, δ)-tester for P.*

For adaptive algorithms, we have two options. First, there is a straightforward approach that relies on the following observations:

Observation 2.0.9. Let $P \subseteq \{0,1\}^n$ be a property, and let A be an adaptive (ϵ, δ)-tester for P with query complexity q. Then there exists a non-adaptive (ϵ, δ)-tester with query complexity 2^q.

This observation is obtained by non-adaptively querying the entire decision tree used by A, and then simulating A over the queried values. The contrapositive of it implies that if we know that no non-adaptive algorithm making q queries can be an (ϵ, δ)-tester for P, neither can any adaptive algorithm making $\log q$ queries.

Yao's Lemma also has an adaptive flavor. The following is similar to the form that appears in [Fis04b] (but was developed earlier).

Lemma 2.0.10. *Suppose that μ_P and μ_N are two distributions over $\{0,1\}$. For an index set $Q \subset \{1, \ldots, n\}$ of size q and a word $v \in \{0,1\}^q$, let $\alpha(Q, v)$ be the probability that a word $w \in \{0,1\}$ drawn according to μ_P agrees with v over Q (i.e., that setting $i_1, \ldots, i_q$ to be the members of Q in sorted order, we have $w_{i_j} = v_j$ for all $1 \leq j \leq q$). Define $\beta(Q, v)$ similarly with μ_N instead of μ_P.*

If for every Q of size q and every $v \in \{0,1\}^q$ we have that $\alpha(Q, v) \geq (1 - \eta)\beta(Q, v)$, then no algorithm making up to q queries can distinguish with probability more than η (even an adaptive one and in a 2-sided manner) between the case where w was drawn according to μ_P and the case where it was drawn according to μ_N.

2.1 Directed ordered trees

Both Chapter 3 and Chapter 4 will concern algorithms over directed ordered trees. We introduce here some common notations and terminology.

Definition 2.1.1. A *digraph* G is a pair (V, E) such that $E \subseteq V \times V$.

For every $v \in V$ we set $\texttt{out-deg}(v) = |\{u \in V \mid (v, u) \in E\}|$.

Definition 2.1.2. A *path* is a tuple $(u_1, \ldots, u_k) \in |V|^k$ such that $u_1, \ldots, u_k$ are all distinct and $(u_i, u_{i+1}) \in E$ for every $i \in [k-1]$. The *length* of a path $(u_1, \ldots, u_k) \in |V|^k$ is $k - 1$.

We say that there is a path from u to v if there exists a path $(u_1, \dots, u_k)$ in G such that $u_1 = u$, and $u_k = v$.

Definition 2.1.3. The *distance* from $u \in V$ to $v \in V$, denoted $\texttt{dist}(u, v)$, is the length of the shortest path from u to v if one exists and infinity otherwise.

We use the standard terminology for outward-directed rooted trees.

Definition 2.1.4. A *rooted directed tree* is a tuple (V, E, r), where (V, E) is a digraph, $r \in V$ and for every $v \in V$ there is a unique path from r to v. Let $u, v \in V$. If $\texttt{out-deg}(v) = 0$ then we call v a leaf. We say that u is an *ancestor* of v and v is a *descendant* of u if there is a path from u to v. We say that u is a *child* of v and v is a *parent* of u if $(v, u) \in E$, and set $\texttt{Children}(v) = \{w \in V \mid w \text{ is a child of } v\}$.

Definition 2.1.5. An *ordered tree* is a rooted (directed) tree such that, for every vertex, there is a linear order on its child vertices. A *colored ordered tree* is an ordered tree T supplied with a function $c : V(T) \to C$, called a *coloring* of the tree T.

Our trees will be such that every inner vertex is of degree d. If a vertex v has less than d children, then we still see them as a d-tuple, with several entries set to null. Formally:

Definition 2.1.6. The *children vector* of a vertex v in an ordered tree T is a vector $x \in (V(T) \cup \{\text{null}\})^d$, where x_i is the ith child vertex of v if one exists, and *null* otherwise.

Note that the tree is ordered, and we will indeed refer to it in this way. The tree also has the *descendence relation* over its vertices. The descendant order and the ordering of the children together induce a partial ordering comparing vertex pairs that have no descendence betweem them, based on their lowest common ancestor. Note that a vertex is a descendant (and ancestor) of itself.

Definition 2.1.7. Given an ordered tree T and two vertices $u, v \in V(T)$, we say that u is *left* of v if there exists vertex $w \in V(T)$ such that u is a descendant of the ith child of w, v is a descendant of the jth child of w, and $i < j$.

Although this is not explicitly required, w in the above definition will always be the lowest common ancestor of u and v.

Chapter 3

Testing Formula Satisfaction

3.1 Introduction

A central area of research in Property-Testing in general and Massively-Parametrized Testing in particular is to associate the query complexity of problems to their other measures of complexity. There are a number of results in this direction, to name some examples see [AKNS00, New02, FNS04]. In [BSHR05] the study of formula satisfiability was initiated. There it was shown that there exists a property that is defined by a 3-CNF formula and yet has a query complexity that is linear in the size of the input. This implies that knowing that a specific property is accepted by a 3-CNF formula does not give any information about its query complexity. In [HLNT07] it was shown that if a property is accepted by a read-twice CNF formula, then the property is testable.

In this chapter we study the query complexity of properties that are accepted by read once formulas. These can be described as computational trees, with the tested input values at the leaves and logic gates at the other nodes, where for an input to be in the property a certain value must result when the calculation is concluded at the root.

Section 3.2 contains preliminaries. First we define the properties that we test, and then we introduce numerous definitions and lemmas about bringing the formulas whose satisfaction is tested into a normalized "basic form". These are important and in fact implicitly form a preprocessing part of our algorithms. Once the formula is put in a basic form, testing an assignment to the formula becomes manageable.

In Section 3.3 we show the testability of properties defined by formulas involving arbitrary Boolean gates of bounded arity. For such formulas involving only monotone gates, we provide an *estimation* algorithm in Section 3.4, that is an algorithm that not only tests for the property, but with high probability outputs a real number η such that the true distance of the tested input from the property is between $\eta - \epsilon$ and $\eta + \epsilon$. In Section 3.5 we show that when restricted to And/Or gates, we can provide a test whose query complexity is quasipolynomial in ϵ. We supply a brief analysis of the running times of the algorithms in Section 3.6.

On the other hand, we prove in Section 3.7 that these results can not be generalized to alphabets that have at least four different letters. We construct a formula utilizing only one (symmetric and binary) gate type over an alphabet of size 4, such that the resulting property requires a number of queries depending on the formula (and input) size for a 1/4-test. We also prove that for the cost of one additional alphabet symbol, we can construct a non-testable explicitly monotone property (both the gate used and the acceptance condition are monotone).

Results such as these may have interesting applications in computational complexity. One interesting implication of the testability results here is that any read-once formula accepting an untestable Boolean property must use unbounded arity gates other than And/Or. By proving that properties defined by formulas of a simple form admit efficient property testers, one also paves a path for proving that certain properties cannot be defined by formulas of a simple form — just show that these properties cannot be efficiently testable. Since property testing lower bounds are in general easier to prove than computational complexity lower bounds, this may become a useful approach.

3.2 Preliminaries

3.2.1 Formulas, evaluations and testing

With the terminology of rooted trees given in Chapter 2 we now define our properties; first we define what is a formula and then we define what it means to satisfy one.

Definition 3.2.1 (Formula). A *Formula* is a tuple $\Phi = (V, E, r, X, \kappa, B, \Sigma)$, where (V, E, r) is a rooted directed tree, Σ is an alphabet, X is a set of variables (later on they will take values in Σ), $B \subseteq \bigcup_{k<\infty}\{\Sigma^k \mapsto \Sigma\}$ is a set of functions over Σ, and $\kappa : V \to B \cup X \cup \Sigma$ satisfies the following (we abuse notation somewhat by writing κ_v for $\kappa(v)$).

- For every leaf $v \in V$ we have that $\kappa_v \in X \cup \Sigma$.
- For every v that is not a leaf $\kappa_v \in B$ is a function whose arity is $|\texttt{Children}(v)|$.

In the case where B contains functions that are not symmetric, we additionally assume that the tree is ordered, i.e. for every $v \in V$ there is an ordering of $\texttt{Children}(v) = (u_1, \ldots, u_k)$.

In the special case where Σ is the Boolean alphabet $\{0, 1\}$, we say that Φ is *Boolean*. Unless stated otherwise $\Sigma = \{0, 1\}$, in which case we shall omit Σ from the definition of formulas. A formula $\Phi = (V, E, r, X, \kappa, B, \Sigma)$ is called *read k-times* if for every $x \in X$ there are at most k vertices $v \in V$, where $\kappa_v \equiv x$. We call Φ a *read-once-formula* if it is read 1-times. A formula $\Phi = (V, E, r, X, \kappa, B, \Sigma)$ is called *k-ary* if the arity (number of children) of all its vertices is at most k. If a formula is 2-ary (all functions in B have arity at most 2) then we call it *binary*. A function $f : \{0, 1\}^n \to \{0, 1\}$ is *monotone* if whenever $x \in \{0, 1\}^n$ is such that $f(x) = 1$, for every $y \in \{0, 1\}^n$ such that $x \leq y$ (coordinate-wise) we have $f(y) = 1$ as well. If all the functions in B are monotone then we say that Φ is (explicitly) *monotone*. We denote $|\Phi| = |X|$ and call it the *formula size* (this makes sense for read-once formulas). Note that this is different from another notion of formula size that refers to the number of operators. In our case, the formula size is the size of its input.

Definition 3.2.2 (Sub-Formula)**.** Let $\Phi = (V, E, r, X, \kappa, B)$ be a formula and $u \in V$. The formula $\Phi_u = (V_u, E_u, u, X_u, \kappa, B)$, is such that $V_u \subseteq V$, with $v \in V_u$ if and only if $\texttt{dist}(u, v)$ is finite, and $(v, w) \in E_u$ if and only if $v, w \in V_u$ and $(v, w) \in E$. X_u is the set of all $\kappa_v \in X$ such that $v \in V_u$. If $u \neq r$ then we call Φ_u a *strict sub-formula*. We define $|\Phi_u|$ to be the number of variables in V_u, that is $|\Phi_u| = |X_u|$, and the *weight* of u with respect to its parent v is defined as $|\Phi_u|/|\Phi_v|$.

Definition 3.2.3 (assignment to and evaluation of a formula)**.** An *assignment* σ to a formula $\Phi = (V, E, r, X, \kappa, B, \Sigma)$ is a mapping from X to Σ. The *evaluation* of Φ given σ, denoted (abusing notation somewhat) by $\sigma(\Phi)$, is defined as $\sigma(r)$ where $\sigma : V \to \Sigma$ is recursively defined as follows.

- If $\kappa_v \in \Sigma$ then $\sigma(v) = \kappa_v$.
- If $\kappa_v \in X$ then $\sigma(v) = \sigma(\kappa_v)$.
- Otherwise ($\kappa_v \in B$), denote the members of the set $\texttt{Children}(v)$ by $(u_1, \ldots, u_k)$, and set $\sigma(v) = \kappa_v(\sigma(u_1), \ldots, \sigma(u_k))$.

Given an assignment $\sigma : X \to \Sigma$ and $u \in V$, we let σ_u denote its restriction to X_u, but whenever there is no confusion we just use σ also for the restriction (as an assignment to Φ_u).

We set $SAT(\Phi = b)$ to be the set of all the assignments σ to Φ such that $\sigma(\Phi) = b$. For Boolean formulas, when $b = 1$ and we do not consider the case $b = 0$ in that context, we simply denote these assignments by $SAT(\Phi)$. If $\sigma \in SAT(\Phi)$ then we say that σ *satisfies* Φ. Let σ_1, σ_2 be assignments to Φ. We define $\texttt{dist}_\Phi(\sigma_1, \sigma_2)$ to be the relative Hamming distance between the two assignments. That is, $\texttt{dist}_\Phi(\sigma_1, \sigma_2) = |\{x \in X \mid \sigma_1(x) \neq \sigma_2(x)\}|/|\Phi|$. For every assignment σ to Φ and every subset S of assignments to Φ we define $\texttt{dist}_\Phi(\sigma, S) = \min\{\texttt{dist}_\Phi(\sigma, \sigma') \mid \sigma' \in S\}$. If $\texttt{dist}_\Phi(\sigma, S) > \epsilon$ then σ is ϵ*-far* from S and otherwise it is ϵ*-close* to S.

We now have the ingredients to define testing of assignments to formulas in a massively parametrized model as it is analyzed in this chapter. Namely, the formula Φ is the parameter that is known to the algorithm in advance and may not change, while the assignment $\sigma : X \to \Sigma$ must be queried using as few queries as possible, and distance is measured with respect to the fraction of alterations it requires.

Definition 3.2.4. [(ϵ, q)**-test**] An (ϵ, q)*-test* for $SAT(\Phi)$ is a randomized algorithm $\mathcal{A}$ with free access to Φ, that given oracle access to an assignment σ to Φ operates as follows.

- $\mathcal{A}$ makes at most q queries to σ (where on a query $x \in X$ it receives σ_x as the answer).
- If $\sigma \in SAT(\Phi)$, then $\mathcal{A}$ accepts (returns 1) with probability at least $2/3$.
- If σ is ϵ-far from $SAT(\Phi)$, then $\mathcal{A}$ rejects (returns 0) with probability at least $2/3$. Recall that σ is ϵ-far from $SAT(\Phi)$ if its relative Hamming distance from every assignment in $SAT(\Phi)$ is at least ϵ.

We say that $\mathcal{A}$ is *non-adaptive* if its choice of queries is independent of their values (and may depend only on Φ). We say that $\mathcal{A}$ has 1*-sided error* if given oracle access to $\sigma \in SAT(\Phi)$, it accepts (returns 1) with probability 1. We say that $\mathcal{A}$ is an (ϵ, q)*-estimator* if it returns a value η such that with probability at least $2/3$, σ is both $(\eta + \epsilon)$-close and $(\eta - \epsilon)$-far from $SAT(\Phi)$.

We can now summarize the contributions of the chapter in the following theorem:

Theorem 3.1 (Main Theorem)**.** *The following statements all hold for all constant k:*

- *For any read-once formula Φ where B is the set of all functions of arity at most k there exists a 1-sided (ϵ, q)-test for SAT(Φ) with $q = \exp(\text{poly}(\epsilon^{-1}))$ (Theorem 3.2).*
- *For any read-once formula Φ where B is the set of all monotone functions of arity at most k there exists an (ϵ, q)-estimator for SAT(Φ) with $q = \exp(\text{poly}(\epsilon^{-1}))$ (Theorem 3.3).*
- *For any read-once formula Φ where B is the set of all conjunctions and disjunctions of any arity there exists an (ϵ, q)-test for SAT(Φ) with $q = \epsilon^{O(\log \epsilon)}$ (Corollary 3.5 of Theorem 3.4).*
- *There exists an infinite family of 4-valued read-once formulas Φ, where B contains one binary function, and an appropriate $b \in \Sigma$, such that there is no non-adaptive (ϵ, q)-test for SAT($\Phi = b$) with $q = o(\texttt{depth}(\Phi))$, and no adaptive (ϵ, q)-test for SAT(Φ) with $q = o(\log(\texttt{depth}(\Phi)))$; there also exists such a family of 5-valued read-once formulas whose gates and acceptance condition are monotone with respect to a fixed order of the alphabet. (Theorem 3.6 and Theorem 3.7 respectively).*

For the first two items above, the degree of the polynomial is linear in k.

3.2.2 Basic formula simplification and handling

In the following, unless stated otherwise, our formulas will all be read-once and Boolean. For our algorithms to work, we will need a somewhat "canonical" form of such formulas. We say that two formulas Φ and Φ' are *equivalent* if $\sigma(\Phi) = \sigma(\Phi')$ for every assignment $\sigma : X \to \Sigma$.

Definition 3.2.5. A 1-witness for a boolean function $f : \{0,1\}^n \to \{0,1\}$ is a subset of coordinates $W \subseteq [n]$ which is minimal (by inclusion) amongst all subsets for which there exists an assignment $\sigma : W \to \{0,1\}$ such that for every $x \in \{0,1\}^n$ which agrees with σ (that is, where for all $i \in W$, we have that $x_i = \sigma(i)$) we have that $f(x) = 1$.

Note that a function can have several 1-witnesses and that a 1-witness for a monotone function can always use the assignment σ that maps all coordinates to 1.

Definition 3.2.6. The mDNF (monotone disjunctive normal form) of a monotone boolean function $f : \{0,1\}^n \to \{0,1\}$ is a set of terms T where each term in T is a 1-witness for f and for every $x \in \{0,1\}^n$, $f(x) = 1$ if and only if there exists a term $T_j \in T$ such that for all $i \in T_j$, we have that $x_i = 1$.

Observation 3.2.7. Any monotone boolean function $f : \{0,1\}^n \to \{0,1\}$ has a unique mDNF T.

Proof. The corresponding mDNF is the disjunction of f's 1-witnesses. □

Definition 3.2.8. For $u \in V$, $v \in \texttt{Children}(u)$ is called *(a,b)-forceful* if $\sigma(v) = a$ implies $\sigma(u) = b$. v is *forceful* if it is (a,b)-forceful for some $a, b \in \{0,1\}$.

For example, for a $\wedge$ gate all children are $(0,0)$-forceful, and for a $\vee$ gate all children are $(1,1)$-forceful. Forceful variables are variables that cause an "Or-like" or "And-like" behavior in the gate.

Definition 3.2.9. A vertex $v \in V$ in a formula Φ is called *unforceable* if no child of v is forceful.

Definition 3.2.10. A vertex $v \in V$ in a formula Φ is called *trivial* if there exists a constant $c \in \{0,1\}$ such that for every assignment σ, $\sigma(v) = c$.

Definition 3.2.11 (k-x-Basic formula)**.** A read-once formula Φ is *k-x-basic* if it is Boolean and all the functions in B are either:

- Negations,
- unforceable and of arity at least 2 and at most k,
- an $\wedge$ gate or an $\vee$ gate of arity at least 2.

Additionally, Φ must satisfy the following:

- There are no trivial vertices,
- negations may only have leaves as children,
- there is no leaf $v \in V$ such that $\kappa_v \in \{0,1\}$,
- no $\wedge$ is a child of a $\wedge$ and no $\vee$ is a child of a $\vee$,
- every variable may appear at most once in a leaf.

The set of variables that appear negated will be denoted by $\neg X$.

Definition 3.2.12 (k-Basic formula)**.** A read-once formula Φ is a *k-basic* formula if it is k-x-basic, and furthermore all functions in B are also monotone. If B contains only conjunctions and disjunctions then we abbreviate and call the formula *basic.*

Note that a k-Basic formula can obviously only be monotone.

Lemma 3.2.13. *Every read-once formula Φ with gates of arity at most k has an equivalent k-x-basic formula Φ', possibly over a different set of functions B.*

Proof. Suppose for some u that $v \in \texttt{Children}(u)$ is (a,b)-forceful. If $b = 1$ then κ_u can be replaced with an $\vee$ gate, where one input of the $\vee$ gate is v if $a = 1$ or the negation of v if $a = 0$, and the other input is the result of u when fixing $\sigma(\kappa_v) = 1 - a$. If $b = 0$ then κ_u can be replaced with an $\wedge$ gate, where one input of the $\wedge$ gate is v if $a = 0$ or the negation of v if $a = 1$, and the other input is the gate u when fixing $\sigma(\kappa_v) = 1 - a$. After performing this transformation sufficiently many times we have no forceable gates left except for $\wedge$ and $\vee$.

We will now eliminate $\neg$ gates. Any $\neg$ gate in the input or output of a gate which is not $\wedge$ or $\vee$ can be assimilated into the gate. Otherwise, a $\neg$ on the output of an $\vee$ gate can be replaced with an $\wedge$ gate with $\neg$'s on all of its inputs, according to De-Morgan's laws. Also by De-Morgan's laws, a $\neg$ on the output of an $\wedge$ gate can be replaced with an $\vee$ gate with $\neg$'s on all of its inputs.

Finally, any $\vee$ gates that have $\vee$ children can be merged with them, and the same goes for $\wedge$ gates. Now we have achieved an equivalent k-x-basic formula. ☐

Observation 3.2.14. Any formula Φ which is comprised of only monotone k-arity gates has an equivalent k-basic formula Φ'.

Proof. This observation follows by inspecting the above proof, and noticing that monotone gates will never produce negations in the process described, in particular having no $(0,1)$-forceful or $(1,0)$-forceful children. $\square$

3.2.3 Observations about subformulas and distance

Definition 3.2.15 (heaviest child $h(v)$)**.** Let $\Phi = (V, E, r, X, \kappa, B)$ be a formula. For every $v \in V$ we define $h(v)$ to be v if $\texttt{Children}(v) = \emptyset$, and otherwise to be an arbitrarily selected vertex $u \in \texttt{Children}(v)$, such that $|\Phi_u| = \max\{|\Phi_w| \mid w \in \texttt{Children}(v)\}$.

Definition 3.2.16 (vertex depth $\texttt{depth}_\Phi(v)$)**.** Let $\Phi = (V, E, r, X, \kappa, B)$ be a formula. For every $v \in V$ we define $\texttt{depth}_\Phi(v) = \texttt{dist}(r, v)$ and $\texttt{depth}(\Phi) = \max\{\texttt{depth}_\Phi(u) \mid u \in V\}$.

Our first observation is that in "and" gates and similar situations, distance implies distance in subformulas, in a Markov's inequality-like fashion.

Observation 3.2.17. Let $v \in V$ be a vertex with no trivial children, such that either $\kappa_v \equiv \vee$ and its output $b = 0$ or $\kappa_v \equiv \wedge$ and $b = 1$, and $\texttt{dist}(\sigma, SAT(\Phi_v = b)) \geq \epsilon$. For every $1 > \alpha > 0$ there exists $S \subseteq \texttt{Children}(v)$ such that $\sum_{s \in S} |\Phi_s| \geq \epsilon\alpha|\Phi|$ and $\texttt{dist}(\sigma, SAT(\Phi_w = b)) \geq \epsilon(1-\alpha)$ for every $w \in S$. Furthermore, there exists a child $u \in \texttt{Children}(v)$ such that $\texttt{dist}(\sigma, SAT(\Phi_u = 1 - b)) \geq \epsilon$.

Proof. Let T be the maximum subset of $\texttt{Children}(v)$ such that Φ_w is $\epsilon(1-\alpha)$-far from being evaluated to b for every $w \in T$. If $\sum_{t \in T} |\Phi_t| < \epsilon\alpha|\Phi|$ then the distance from having Φ_v evaluate to b is less than $\epsilon\alpha + \epsilon(1-\alpha) = \epsilon$, since we only need to change the $\epsilon\alpha|\Phi_v|$ leaves that descend from the children in S and for the rest, we know that each of them is $\epsilon(1-\alpha)$-close to satisfaction, and therefore only that fraction of inputs in leaves that descend from children outside of S need to be changed. This contradicts the assumption.

For the second statement, note that if no such child exists then Φ_v is ϵ-close to being evaluated to b. $\square$

Observation 3.2.18. Let $v \in V$ be a vertex with no trivial children, such that either $\kappa_v \equiv \vee$ and $b = 1$ or $\kappa_v \equiv \wedge$ and $b = 0$, and $\texttt{dist}(\sigma, SAT(\Phi_v = b)) \geq \epsilon$. For every child $u \in \texttt{Children}(v)$, $|\Phi_u| \geq |\Phi|\epsilon$ and $\texttt{dist}(\sigma, SAT(\Phi_u = b)) \geq \epsilon(1+\epsilon)$. Furthermore, $\epsilon \leq 1/2$, and for any $u \in \texttt{Children}(v) \setminus \{h(v)\}$, $\texttt{dist}(\sigma, SAT(\Phi_u = b)) \geq 2\epsilon$.

Proof. First suppose that the weight of some child u is less than ϵ. In this case, setting u to b makes the formula Φ_v evaluate to b by changing less than an ϵ fraction of inputs, a contradiction.

Since there are at least two children, every child u is of weight at most $1-\epsilon$, and since setting it to b would make Φ_v evaluate to b, it is at least $\epsilon(1+\epsilon)$-far from being evaluated to b.

For the last part, note that since $|\texttt{Children}(v)| > 1$, there exists $u \in \texttt{Children}(v)$ such that $|\Phi_u| \leq |\Phi_v|/2$. Thus every assignment to Φ_v is 1/2-close to an assignment σ' by which Φ_v evaluates to b. Also note that any $u \in \texttt{Children}(v) \setminus \{h(v)\}$ satisfies $|\Phi_u| \leq |\Phi_v|/2$, and therefore if Φ_u were 2ϵ-close to being evaluated to b, Φ_v would be ϵ-close to being evaluated to b. $\square$

3.2.4 Heavy and light children in general gates

We would like to pick the heaviest child of a general gate, same as we did above. The problem is that since we will use this for unforceable gates, we will simultaneously want the heaviest child or children not to be "too heavy". This brings us to the following definition.

Definition 3.2.19. Given a k-x-basic formula Φ, a parameter ϵ and a vertex u, we let $\ell = \ell(u, \epsilon)$ be the smallest integer such that the size of the ℓ'th largest child of u is less than $|\Phi|(4k/\epsilon)^{-\ell}$ if such an integer exists, and set $\ell = k + 1$ otherwise. The *heavy* children of u are the $\ell - 1$ largest children of u, and the rest of the children of u are its *light* children.

Note that if there is a really big child, then σ is close to both $SAT(\Phi_v = 1)$ and $SAT(\Phi_v = 0)$. More formally:

Lemma 3.2.20. *If an unforceable vertex v with no trivial children has a child u such that $|\Phi_v|(1 - \epsilon) \leq |\Phi_u|$, then σ is both ϵ-close to $SAT(\Phi_v = 1)$ and ϵ-close to $SAT(\Phi_v = 0)$.*

Proof. The child is unforceful, and therefore it is possible to change the remaining children to obtain any output value. □

Observation 3.2.21. If for a vertex u with no trivial children, $\kappa_u \not\equiv \wedge$, $\kappa_u \not\equiv \vee$, $\kappa_u \notin X$ and σ is ϵ-far from $SAT(\Phi_u = b)$, then it must have at least two heavy children.

Proof. By the definition of ℓ, if there is just one heavy child, then $\ell = 2$ and the total weight of the light children is strictly smaller than ϵ. Therefore by Lemma 3.2.20 there must be more than one heavy child, as otherwise the gate is ϵ-close to both 0 and 1. □

3.3 Upper bound for general bounded arity formula

Algorithm 3.1 tests whether the input is ϵ-close to having output b with 1-sided error, and also receives a confidence parameter δ. The explicit confidence parameter makes the inductive arguments easier and clearer. The algorithm operates by recursively checking the conditions in Observations 3.2.17 and 3.2.18.

Theorem 3.2. *Algorithm 3.1$(\Phi, \epsilon, \delta, \sigma)$ always accepts any input that satisfies the read-once formula Φ, and rejects any input far from satisfying Φ with probability at least $1 - \delta$. Its query complexity (treating k and δ as constant) is always $O(\exp(poly(\epsilon^{-1})))$.*

Proof. Follows from Lemma 3.3.3, Lemma 3.3.4 and Lemma 3.3.2 (in that order) below. □

Algorithm 3.1 Test satisfiability of read-once formula

```
Input: read-once $k$-$x$-basic formula $\Phi = (V, E, r, X, \kappa)$, parameters $\epsilon, \delta > 0, b \in \{0,1\}$, oracle to $\sigma$.
Output: "true" or "false".
1: if $\epsilon > 1$ then return "true"
2: if $\kappa_r \in X$ then return the truth value of $\sigma(r) = b$
3: if $\kappa_r \in \neg X$ then return the truth value of $\sigma(r) = 1 - b$
4: if $(\kappa_r \equiv \wedge$ and $b = 1)$ or $(\kappa_r \equiv \vee$ and $b = 0)$ then
5:     $y \longleftarrow$ "true"
6:     for $i = 1$ to $l = 32(2k/\epsilon)^{2k}\log(\delta^{-1})$ do
7:         $u \longleftarrow$ a vertex in Children($r$) selected independently at random, where the probability
           that $w \in$ Children($r$) is selected is $|\Phi_w|/|\Phi|$
8:         $y \longleftarrow y \wedge$ Algorithm 3.1$(\Phi_u, (\epsilon(1 + (8k/\epsilon)^{-k}/16)), \sigma, \delta/2, b)$
9:     return $y$
10: if $(\kappa_r \equiv \wedge$ and $b = 0)$ or $(\kappa_r \equiv \vee$ and $b = 1)$ then
11:     if there exists a child of weight less than $\epsilon$ then return "true"
12:     $y \longleftarrow$ "false"
13:     for all $u \in$ Children($r$) do $y \longleftarrow y \vee$ Algorithm 3.1$(\Phi_u, (\epsilon(1+\epsilon)), \sigma, \epsilon\delta/2, b)$
14:     return $y$
15: if there is a child of weight at least $1 - \epsilon$ then return "true"
16: for all $u \in$ Children($r$) do
17:     $y_u^0 \longleftarrow$ Algorithm 3.1$(\Phi_u, (\epsilon(1 + (4k/\epsilon)^{-k})), \sigma, \delta/2k, 0)$
18:     $y_u^1 \longleftarrow$ Algorithm 3.1$(\Phi_u, (\epsilon(1 + (4k/\epsilon)^{-k})), \sigma, \delta/2k, 1)$
19: if there exists $x \in \{0,1\}^k$ such that $\kappa_r$ on $x$ evaluates to $b$ and for all $u \in$ Children($r$) we
    have $y_u^{x_u}$ equals "true" then return "true"
20: else return "false"
```

Lemma 3.3.1. *The depth of recursion in Algorithm 3.1 is at most* $16(8k/\epsilon)^k \log(\epsilon^{-1})$.

Proof. If $\epsilon > 1$ then the condition in Line 1 is satisfied and the algorithm returns without any recursion.

All recursive calls occur in Lines 8, 13, 17 and 18. Since Φ is k-x-basic, any call with a subformula whose root is labeled by $\wedge$ results in calls to subformulas, each with a root labeled either by $\vee$ or an unforceable gate, and with the same b value (this is crucial since the b value for which $\wedge$ recurses with a smaller ϵ is the b value for which $\vee$ recurses with a bigger ϵ, and vice-versa). Similarly, any call with a subformula whose root is labeled by $\vee$ results in calls to subformulas, each with a root labeled either by $\wedge$ or an unforceable gate, and with the same b value.

Therefore, in two consecutive recursive calls, there are three options:

1. The first call is made with distance parameter ϵ' where $\epsilon(1+\epsilon) \geq \epsilon' \geq \epsilon(1 + (4k/\epsilon)^{-k})$ and the second call with $\epsilon'' = \epsilon'(1 - (8k/\epsilon')^{-k}/16)$. In this case the distance parameter increases by at least $(1 + (4k/\epsilon)^{-k})(1 - (8k/\epsilon(1+\epsilon))^{-k}/16) \geq (1 + (4k/\epsilon)^{-k}/8)$.

2. The first call is made with distance parameter $\epsilon' = \epsilon(1 - (8k/\epsilon)^{-k}/16)$ and the second call with $\epsilon'' \geq \epsilon'(1 + (4k/\epsilon')^{-k})$. In this case in two consecutive calls the distance parameter increases by at least $\epsilon(1 - (8k/\epsilon)^{-k}/16)(1 + (4k/\epsilon(1 - (8k/\epsilon)^{-k}/16))^{-k}) \geq (1 + (8k/\epsilon)^{-k}/8)$.

3. The first call is made with distance parameter $\epsilon' \geq \epsilon(1+(4k/\epsilon)^{-k})$ and the second call with $\epsilon'' \geq \epsilon'(1+(4k/\epsilon')^{-k})$. In this case two consecutive in calls the distance parameter increases by at least $(1+(4k/\epsilon')^{-k})^2$

Therefore, in all cases, an increase of two in the depth results in an increase of the distance parameter from ϵ to at least $\epsilon(1+(8k/\epsilon)^{-k}/8)$. Thus in recursive calls of depth $16(8k/\epsilon)^k\log(\epsilon^{-1})$ the distance parameter exceeds 1 and the call returns without making any further calls. □

Lemma 3.3.2. *Algorithm 3.1 uses at most* $\exp(\mathrm{poly}(1/\epsilon))$ *queries for a constant* k*.*

Proof. If $\epsilon > 1$ then the condition in Line 1 is satisfied and no queries are made. Therefore assume $\epsilon \leq 1$. Observe that in a specific instantiation at most one query is used, either in Line 2 or Line 3. Therefore the number of queries is upper bounded by the number of instantiations of Algorithm 3.1.

Recall that by Lemma 3.3.1 the depth of the recursion is at most $16(8k/\epsilon)^k\log(\epsilon^{-1})$. Since by the proof of Lemma 3.3.1 every two consecutive recursive calls increase the distance parameter, it is never smaller than $\epsilon' = \epsilon(1-(8k/\epsilon)^{-k}/16)$. In a specific instantiation at most

$$32(2k/\epsilon')^{2k}\log(\delta^{-1}) = 32(2k)^{2k}(\epsilon(1-(8k/\epsilon)^{-k}/16))^{-2k}\log(\delta^{-1})$$

recursive calls are made in total (note that by Line 11 there are at most $1/\epsilon'$ children in the case of the condition in Line 10, and in the case of an unforceable gate there are at most $2k$ recursive calls).

To conclude, we note that the value of the confidence parameter in all these calls is lower bounded by

$$\delta\cdot(\epsilon'/2k)^{16(8k/\epsilon)^k\log(\epsilon^{-1})} \geq \delta\cdot(\epsilon')^{32(8k/\epsilon)^k\log(k\epsilon^{-1})} = \delta\cdot(\epsilon(1-(8k/\epsilon)^{-k}/16))^{32(8k/\epsilon)^k\log(k\epsilon^{-1})}.$$

Therefore at most

$$(32(2k)^{2k}(\epsilon(1-(8k/\epsilon)^{-k}/16))^{-2k}\log(\delta^{-1}(\epsilon(1-(8k/\epsilon)^{-k}/16))^{-32(8k/\epsilon)^k\log(k\epsilon^{-1})}))^{16(8k/\epsilon)^k\log(\epsilon^{-1})}$$

queries are used, which is $\exp(\mathrm{poly}(1/\epsilon))$ for a constant k. □

Lemma 3.3.3. *If* Φ *on* σ *evaluates to* b *then Algorithm 3.1 returns "true" with probability* 1*.*

Proof. If $\epsilon > 1$ then the condition of Line 1 is satisfied and "true" is returned correctly. We proceed with induction over the depth of the formula. If $\texttt{depth}(\Phi) = 0$ then $\kappa_r \in X \cup \neg X$. If $\kappa_r \in X$ then since Φ evaluates to b, $\sigma(r) = b$, and if $\kappa_r \in \neg X$ then $\sigma(r) = 1-b$, and the algorithm returns "true" correctly.

Now assume that $\texttt{depth}(\Phi) > 0$. Obviously, for all $u \in \texttt{Children}(r)$, we have that $\texttt{depth}(\Phi) > \texttt{depth}(\Phi_u)$ and therefore from the induction hypothesis any recursive call with parameter $b' \in \{0,1\}$ on a subformula that evaluates to b' returns "true" with probability 1.

If $\kappa_r \equiv \wedge$ and $b = 1$ or $\kappa_r \equiv \vee$ and $b = 0$, then it must be the case that for all $u \in \texttt{Children}(r)$, Φ_u evaluates to b. By the induction hypothesis all recursive calls will return "true" and y will get the value "true", which will be returned by the algorithm.

Now assume that $\kappa_r \equiv \wedge$ and $b = 0$ or $\kappa_r \equiv \vee$ and $b = 1$. Since Φ evaluates to b then it must be the case that at least for one $u \in \texttt{Children}(r)$, Φ_u evaluates to b. By the induction hypothesis, the recursive call on that u will return "true", and y will get the value "true" which will be returned by the algorithm (unless the algorithm already returned "true" for another reason, e.g. in line 11).

Lastly, assume that r is an unforceable gate. Since Φ evaluates to b, the children of r evaluate to the assignment σ which evaluates to b. By the induction hypothesis, for every $u \in \texttt{Children}(r)$ the recursive call on Φ_u with $\sigma(u)$ will return "true", and thus the assignment σ will, in particular, fill the condition in Line 19 and the algorithm will return "true". $\square$

Lemma 3.3.4. *If σ is ϵ-far from getting Φ to output b then Algorithm 3.1 returns "false" with probability at least $1-\delta$.*

Proof. The proof is by induction over the tree structure, where we partition to cases according to κ_r and b. Note that $\epsilon \le 1$.

If $\kappa_r \in X$ or $\kappa_r \in \neg X$ then by Lines 2 or 3 the algorithm returns "false" whenever σ does not make Φ output b.

If $\kappa_r \equiv \wedge$ and $b = 1$ or $\kappa_r \equiv \vee$ and $b = 0$, since σ is ϵ-far from getting Φ to output b then by Observation 3.2.17 we get that there exists $T \subseteq \texttt{Children}(r)$ for which it holds that $\sum_{t\in T} |\Phi_t| \ge |\Phi|\epsilon((8k/\epsilon)^{-k}/16)$ and each Φ_t is $\epsilon(1+(8k/\epsilon)^{-k}/16$-far from being evaluated to b. Let S be the set of all vertices selected in Line 7. The probability of a vertex from T being selected is at least $\epsilon((8k/\epsilon)^{-k}/16)$. Since this happens at least $32(2k/\epsilon)^{2k}\log(\delta^{-1})$ times independently, with probability at least $1-\delta/2$ we have that $S \cap T \neq \emptyset$. Letting $w \in T \cap S$, the recursive call on it with parameter $\epsilon(1+(8k/\epsilon)^{-k}/16$ will return "false" with probability at least $1-\delta/2$, which will eventually cause the returned value to be "false" as required. Thus the algorithm succeeds with probability at least $1-\delta$.

Now assume that $\kappa_r \equiv \wedge$ and $b = 0$ or $\kappa_r \equiv \vee$ and $b = 1$. Since Φ is ϵ-far from being evaluated to b, Observation 3.2.18 implies that all children are of weight at least ϵ and are $\epsilon(\epsilon+1)$-far from b, and therefore the conditions of Line 11 would not be triggered. Every recursive call on a vertex $v \in \texttt{Children}(r)$ is made with distance parameter $\epsilon(1+\epsilon)$ and so it returns "true" with probability at most $\epsilon\delta/2$. Since there are at most ϵ^{-1} children of r, the probability that none returns "true" is at least $1-\delta/2$ and in that case the algorithm returns "false" successfully.

Now assume that κ_r is some unforceable gate. By Lemma 3.2.20, since Φ is ϵ-far from being satisfied the condition in Line 15 is not triggered. If the algorithm returned "true" then it must be that the condition in Line 19 is satisfied. If there exists some heavy child $u \in \texttt{Children}(r)$ such that y_u^b is "true" and y_u^{1-b} is "false", then by Lemma 3.3.3 the formula Φ_u does evaluate to b and the assignment σ must be such that $\sigma(u) = b$. For the rest of the children of r, assuming the calls succeeded, the subformula rooted in each v is $(\epsilon(1+(4k/\epsilon)^{-k}))$-close to evaluate to $\sigma(v)$. Since u is heavy, the total weight of $\texttt{Children}(r) \setminus \{u\}$ is at most $1-(4k/\epsilon)^{-k}$, and thus by changing at most an $(\epsilon(1+(4k/\epsilon)^{-k}))(1-(4k/\epsilon)^{-k}) \le \epsilon$ fraction of inputs we can get to an assignment where Φ evaluates to b.

If all heavy children u are such that both y_u^b and y_u^{1-b} are "true", then pick some heavy child u arbitrarily. Since r is unforceable, there is an assignment that evaluates to b no matter what the value of Φ_u is. Take such an assignment $\tilde{\sigma}$ that fits the real value of Φ_u. Note that for every heavy child v we have that $y_v^{x_v}$ is "true", and therefore by changing at most an $(\epsilon(1+(4k/\epsilon)^{-k}))$-fraction of the variables in Φ_v we can get it to evaluate to x_v. The weight of u is at least $(4k/\epsilon)^{-\ell+1}$ (recall the definition of ℓ in definition 3.2.19), thus the total weight of the other heavy children is at most $1-(4k/\epsilon)^{-\ell+1}$ and the total weight of the light children is at most $\frac{\epsilon}{4}(4k/\epsilon)^{-\ell}$. So by changing all subformulas to evaluate to the value implied by $\tilde{\sigma}$ we change at most an $(\epsilon(1+(4k/\epsilon)^{-k}))(1-(4k/\epsilon)^{-\ell+1}) + \frac{\epsilon}{4}(4k/\epsilon)^{-\ell} \le \epsilon$ fraction of inputs and get an assignment where Φ evaluates to b. Note that this $\tilde{\sigma}$ does not necessarily correspond to the x found in Line 19.

Thus we have found that finding an assignment x in Line 19, assuming the calls are correct, implies that Φ is ϵ-close to evaluate to b. The probability that all relevant calls to an assignment return "true" incorrectly is at most the probability that any of the $2k$ recursive calls errs, which by the union bound is at most δ, and the algorithm will return "false" correctly with probability at least $1-\delta$. □

3.4 Estimator for monotone formula of bounded arity

Algorithm 3.2 below operates in a recursive manner, and estimates the distance to satisfying the formula rooted in r according to estimates for the subformula rooted in every child of r. The algorithm receives a confidence parameter δ as well as the approximation parameter ϵ, and should with probability at least $1-\delta$ return a number η such that the input is both $(\eta+\epsilon)$-close and $(\eta-\epsilon)$-far from satisfying the given formula.

Algorithm 3.2 Approximate distance to satisfiability of monotone formula

```
Input: read-once $k$-basic formula $\Phi = (V, E, r, X, \kappa)$, parameters $\epsilon, \delta > 0$, oracle to $\sigma$.
Output: $\eta \in [0,1]$.
 1: if $\kappa_r \in X$ then return $1 - \sigma(\kappa_r)$
 2: if $\epsilon > 1$ then return $0$
 3: if $\kappa_r \equiv \vee$ and there exists $u \in$ Children$(r)$ with $|\Phi_u| < \epsilon|\Phi|$ then return $0$
 4: if $\kappa_r \equiv \wedge$ then
 5:     for $i = 1$ to $l = \lceil 1000\epsilon^{-2k-2}(8k)^{2k} \cdot \log(1/\delta)\rceil$ do
 6:         $u \longleftarrow$ a vertex in Children$(r)$ selected independently at random, where the probability
            that $w \in$ Children$(r)$ is selected is $|\Phi_w|/|\Phi|$
 7:         $\alpha_i \longleftarrow$ Algorithm 3.2$(\Phi_u, \epsilon(1-(8k/\epsilon)^{-k}/8), \delta\epsilon(8k/\epsilon)^{-k}/32, \sigma)$
 8:     return $\sum_{i=1}^{l} \alpha_i/l$
 9: else
10:     for every light child $u$ of $r$ set $\alpha_u \longleftarrow 0$
11:     for every heavy child $u$ of $r$ perform a recursive call and use the return value to set
        $\alpha_u \longleftarrow$ Algorithm 3.2$(\Phi_u, \epsilon(1+(4k/\epsilon)^{-k}), \delta/\max\{k, 1/\epsilon\}, \sigma)$
12:     for every term $C$ in the mDNF of $\kappa_r$ set $\alpha_C \longleftarrow \sum_{u\in C} \alpha_u \cdot \frac{|\Phi_u|}{|\Phi|}$
13:     return $\min\{\alpha_C : C \in \mathrm{mDNF}(\kappa_r)\}$
```

The following states that Algorithm 3.2 indeed gives an estimation of the distance. While estimation algorithms cannot have 1-sided error, there is an additional feature of this algorithm that makes it also useful as a 1-sided test (by running it and accepting if it returns $\eta = 0$).

Theorem 3.3. *With probability at least $1-\delta$, the output of Algorithm 3.2$(\Phi, \epsilon, \delta, \sigma)$ is an η such that the assignment σ is both $(\eta+\epsilon)$-close to satisfying Φ and $(\eta-\epsilon)$-far from satisfying it. Additionally, if the assignment σ satisfies Φ then $\eta = 0$ with probability 1. Its query complexity (treating k and δ as constant) is $O(\exp(poly(\epsilon^{-1})))$.*

Proof. The bound on the number of queries is a direct result of Lemma 3.4.2 below. Given that, the correctness proof is done by induction over the height of the formula. The base case (for any

ϵ and δ) is the observation that an instantiation of the algorithm that makes no recursive calls (i.e. triggers the condition in Line 1, 2 or 3) always gives a value that satisfies the assertion.

The induction step uses Lemma 3.4.3 and Lemma 3.4.4 below. Given that the algorithm performs correctly (for any ϵ and δ) for every formula Φ' of height smaller than Φ, the assertions of the lemma corresponding to κ_r (out of the two) are satisfied, and so the correctness for Φ itself follows. □

The dependence on δ can be made into a simple logarithm by a standard amplification technique: Algorithm 3.2 is run $O(1/\delta)$ independent times, each time with a confidence parameter of $2/3$, and then the median of the outputs is taken.

Lemma 3.4.1. *When called with Φ, ϵ, δ, and oracle access to σ, Algorithm 3.2 goes down at most $3(8k/\epsilon)^k \log(1/\epsilon) = \mathrm{poly}(\epsilon)$ recursion levels.*

Proof. Recursion can only happen on Line 7 and Line 11. Moreover, because of the formula being k-basic, recursion cannot follow through Line 7 two recursion levels in a row. In every two consecutive recursive calls there are three options:

1. The first call is made with distance parameter $\epsilon' = \epsilon(1+(4k/\epsilon)^{-k})$ and the second with $\epsilon'' = \epsilon'(1-(8k/\epsilon')^{-k}/8)$. In this case the distance parameter increases by a factor of $(1+(4k/\epsilon)^{-k})(1-(8k/(\epsilon(1+(4k/\epsilon)^{-k})))^{-k}/8) \geq (1+\frac{7}{8}(4k/\epsilon)^{-k})$.

2. The first call is made with distance parameter $\epsilon' = \epsilon(1-(8k/\epsilon)^{-k}/8)$ and the second with $\epsilon'' = \epsilon'(1+(4k/\epsilon')^{-k})$. In this case the distance parameter increases by a factor of $(1-(8k/\epsilon)^{-k}/8)(1+(4k/(\epsilon(1-(8k/\epsilon)^{-k}/8)))^{-k}) \geq (1+\frac{7}{8}(8k/\epsilon)^{-k})$

3. The first call is made with distance parameter $\epsilon' = \epsilon(1+(4k/\epsilon)^{-k})$ and the second with $\epsilon'' = \epsilon'(1+(4k/\epsilon')^{-k})$. In this case the distance parameter increases by a factor of at least $(1+(4k/\epsilon)^{-k})^2$.

Therefore, either way, in every two consecutive levels of the recursion ϵ is increased by a factor of at least $(1+\frac{7}{8}(8k/\epsilon)^{-k})$. After $3(8k/\epsilon)^k\log(1/\epsilon)$ recursive steps, such an increase has occurred at least $\frac{3}{2}(8k/\epsilon)^k\log(1/\epsilon)$ times, and therefore the distance parameter is at least $\epsilon\cdot(1+\frac{7}{8}(8k/\epsilon)^{-k})^{\frac{3}{2}(8k/\epsilon)^k\log(1/\epsilon)} > 1$. In such a case the algorithm immediately returns 0 and the recursion stops. □

Lemma 3.4.2. *When called with Φ, ϵ, δ, and oracle access to σ, Algorithm 3.2 uses a total of at most $\exp(\mathrm{poly}(1/\epsilon))$ queries for any constant k.*

Proof. Denote by ϵ' the smallest value of the distance parameter in any recursive call. Denoting by δ' the smallest value of δ in any recursive call, it holds that $\delta' \geq \delta(\epsilon'(8k/\epsilon')^{-k}/32)^{3(8k/\epsilon)^k\log(1/\epsilon)}$ by Lemma 3.4.1. The number of recursive calls per instantiation of the algorithm is thus at most $l' = \lceil 1000\epsilon'^{-2k-2}(8k)^{2k}\cdot\log(1/\delta')\rceil = \mathrm{poly}(1/\epsilon')$. Now, by the proof of Lemma 3.4.1, every two consecutive recursive calls increase the value of the distance parameter. Since it only decreases in line 7, it holds that $\epsilon' \geq \epsilon(1-(8k/\epsilon)^{-k}/8))$. This means that $l' = \mathrm{poly}(1/\epsilon)$.

Since the algorithm may make at most one query per instantiation, and this only in the case where a recursive call is not performed, the total number of queries is (bounding the recursion depth through Lemma 3.4.1) at most $(l')^{3(8k/\epsilon)^k\log(1/\epsilon)} = \exp(\mathrm{poly}(1/\epsilon))$. □

Lemma 3.4.3. *If $\kappa_r \not\equiv \wedge$ and all recursive calls satisfy the assertion of Theorem 3.3, then with probability at least $1-\delta$ the current instantiation of Algorithm 3.2 provides a value η such that σ is both $(\eta+\epsilon)$-close to satisfying Φ and $(\eta-\epsilon)$-far from satisfying it. Furthermore, if σ satisfies Φ then with probability 1 the output is $\eta = 0$.*

Proof. First we note that Step 3, if triggered, gives a correct value for η (as the σ can be made into a satisfying assignment by changing possibly all variables of the smallest child of r). We also note that if $\kappa_r \equiv \vee$ and Step 3 was not triggered, then by definition all of r's children are heavy, and there are no more than $1/\epsilon$ of them.

Let us consider the cost of fixing input bits in order to make σ satisfy Φ. Note that any such fix must make all of the children in some term C in the mDNF evaluate to 1, since these terms are all of the 1-witnesses. Additionally, making all of the children of one term evaluate to 1 is sufficient. Therefore, the distance of σ from Φ is the minimum over all terms C in κ_r of the adjusted cost of making all children of C evaluate to 1, which is $\sum_{u\in C}\texttt{dist}(\sigma, SAT(\Phi_u))\cdot\frac{|\Phi_u|}{|\Phi|}$. Now in this case there are clearly no more than $\max\{k,\epsilon^{-1}\}$ children, so by the union bound, with probability at least $1-\delta$, every call done through Line 7 gave a value η_u so that indeed σ is $(\eta_u+\epsilon(1+(4k/\epsilon)^{-k}))$-close and $(\eta_u-\epsilon(1+(4k/\epsilon)^{-k}))$-far from Φ_u.

Now let D_i denote C_i minus any light children that it may contain, since the approximation ignores these. It may be that some D_i's contain all heavy children of C_i, where "heavy children" refers to the children of r. Since there are no forcing children (and there exist heavy children) it must be the case that some D_i's do not contain all heavy children, since if a heavy child appears in all D_is, then it appears in all C_is and therefore by setting it to 0 we force a 0 in the output. The D_is that do not contain all heavy children will dominate the expression in Line 13. Note that $\sum_{u\in D_i}|\Phi_u| \le (1-(4k/\epsilon)^{k-\ell})|\Phi|$ for any D_i not containing a heavy child. This implies by bounding $(1+(4k/\epsilon)^{-k}))\cdot(1-(4k/\epsilon)^{k-\ell})$:

$$\sum_{u\in D_i}\texttt{dist}(\sigma, SAT(\Phi_u))\cdot\frac{|\Phi_u|}{|\Phi|}-\epsilon < \frac{\sum_{u\in D_i}\eta_u|\Phi_u|}{|\Phi|}$$
$$< \sum_{u\in D_i}\texttt{dist}(\sigma, SAT(\Phi_u))\cdot\frac{|\Phi_u|}{|\Phi|}+\epsilon-2k(4k/\epsilon)^{-\ell}$$

Now the true distance of C_i not containing all heavy children is at least that of D_i, and at most that of D_i plus the added distance of making all light children evaluate to 1, which is bounded by $k(4k/\epsilon)^{-\ell}$. This means that for such a C_i we have:

$$\sum_{u\in C_i}\texttt{dist}(\sigma, SAT(\Phi_u))\cdot\frac{|\Phi_u|}{|\Phi|}-\epsilon < \frac{\sum_{u\in D_i}\eta_u|\Phi_u|}{|\Phi|}$$
$$< \sum_{u\in C_i}\texttt{dist}(\sigma, SAT(\Phi_u))\cdot\frac{|\Phi_u|}{|\Phi|}+\epsilon-k(4k/\epsilon)^{-\ell}$$

The value returned as η is the minimum over terms C_i in κ_r of $\eta_u\cdot\frac{\sum_{u\in D_i}|\Phi_u|}{|\Phi|}$. We also know that this minimum is reached by some C_j which does not contain all heavy children, but it may be that in fact $\texttt{dist}(\sigma, SAT(\Phi)) = \sum_{u\in C_i}\texttt{dist}(\sigma, SAT(\Phi_u))\cdot\frac{|\Phi_u|}{|\Phi|}$ for some $i\ne j$ (the true distance is the minimum of the total distance of each clause, but it may be reached by a different clause).

By our assumptions

$$\texttt{dist}(\sigma, SAT(\Phi)) - \epsilon = \sum_{u \in C_i} \texttt{dist}(\sigma, SAT(\Phi_u)) \cdot \frac{|\Phi_u|}{|\Phi|} - \epsilon$$

$$\leq \sum_{u \in C_j} \texttt{dist}(\sigma, SAT(\Phi_u)) \cdot \frac{|\Phi_u|}{|\Phi|} - \epsilon < \eta$$

so we have one side of the required bound. For the other side, we split into cases. If C_i also does not contain all heavy children then we use the way we calculated η as the minimum over the corresponding sums:

$$\eta = \frac{\sum_{u \in D_j} \eta_u |\Phi_u|}{|\Phi|} \leq \frac{\sum_{u \in D_i} \eta_u |\Phi_u|}{|\Phi|} < \texttt{dist}(\sigma, SAT(\Phi)) + \epsilon$$

In the final case, we note that by the assumptions on the light children we will always have (recalling that C_i will in particular have all heavy children of C_j):

$$\eta = \frac{\sum_{u \in D_j} \eta_u |\Phi_u|}{|\Phi|} < \sum_{u \in C_j} \texttt{dist}(\sigma, SAT(\Phi_u)) \cdot \frac{|\Phi_u|}{|\Phi|} + \epsilon - k(4k/\epsilon)^{-\ell}$$

$$\leq \sum_{u \in C_i} \texttt{dist}(\sigma, SAT(\Phi_u)) \cdot \frac{|\Phi_u|}{|\Phi|} + \epsilon$$

where the rightmost term equals $\texttt{dist}(\sigma, SAT(\Phi)) + \epsilon$ as required.

For the last part of the claim, note that if σ satisfies Φ, then in particular, one of the terms C of κ_r must be satisfied. By the induction hypothesis, for all $u \in C$ we would have $\eta_u = 0$ and therefore $\eta_C = 0$, and since η is taken as a minimum over all terms we would have $\eta = 0$. □

Lemma 3.4.4. *If $\kappa_r \equiv \wedge$ and all recursive calls satisfy the assertion of Theorem 3.3, then with probability at least $1 - \delta$ the current instantiation of Algorithm 3.2 provides a value η such that σ is both $(\eta + \epsilon)$-close to satisfying Φ and $(\eta - \epsilon)$-far from satisfying it. If σ satisfies Φ then with probability 1 the output is $\eta = 0$.*

Proof. First note that if we sample a vertex w according to the distribution of Line 5 and then take the true distance $\texttt{dist}(\sigma, SAT(\Phi_w))$, then the expectation (but not the value) of this equals $\texttt{dist}(\sigma, SAT(\Phi))$. This is because to make σ evaluate to 1 at the root, we need to make all its children evaluate to 1, an operation whose adjusted cost is given by the weighted sum of distances that corresponds to the expectation above.

Thus, denoting by X_i the random variable whose value is $\texttt{dist}(\sigma, SAT(\Phi_{w_i}))$ where w_i is the vertex picked in the ith iteration, we have $\mathrm{E}[X_i] = \texttt{dist}(\sigma, SAT(\Phi))$. By a Chernoff type bound, with probability at least $1 - \delta/2$, the average X of $X_1, \ldots, X_l$ is no more than $\epsilon^{k+1}(4k)^{-k}/16$ away from $\mathrm{E}[X_i]$ and hence satisfies:

$$\texttt{dist}(\sigma, SAT(\Phi)) - \epsilon^{k+1}(4k)^{-k}/16 < X < \texttt{dist}(\sigma, SAT(\Phi)) + \epsilon^{k+1}(4k)^{-k}/16$$

Then note that by the Markov inequality, the assertion of the lemma means that with probability at least $1 - \delta/2$, all calls done in Line 11 but at most $\epsilon(4k/\epsilon)^{-k}/16$ of them return a value η_w so that σ is $(\eta_w + \epsilon(1 - (4k/\epsilon)^{-k}/16))$-close and $(\eta_w - \epsilon(1 - (4k/\epsilon)^{-k}/16))$-far from Φ_w.

When this happens, at least a $(1-\epsilon(4k/\epsilon)^{-k}/16)$ fraction of the answers α_i of the calls are up to $\epsilon(1-(4k/\epsilon)^{-k}/16))$ away from each corresponding X_i, while at most a $\epsilon(4k/\epsilon)^{-k}/16$ fraction of the answers α_i are (trivially) up to 1 away from each corresponding X_i. Summing up these deviations, the final answer average η satisfies

$$X-\epsilon(1-(4k/\epsilon)^{-k}/8)-\epsilon(4k/\epsilon)^{-k}/16<\eta<X+\epsilon(1-(4k/\epsilon)^{-k}/8)+\epsilon(4k/\epsilon)^{-k}/16$$

With probability at least $1-\delta$ both of the above events occur, and summing up the two inequalities we obtain the required bound

$$\texttt{dist}(\sigma, SAT(\Phi))-\epsilon \le \eta < \texttt{dist}(\sigma, SAT(\Phi))+\epsilon$$

3.5 Quasi-polynomial upper bound for basic-formulas

Let $\Phi=(V,E,r,X,\kappa,B)$ be a basic formula and σ be an assignment to Φ.

The main idea of the algorithm is to randomly choose a full root to leaf path, and recurs over all the children of "$\vee$" vertices on this path that go outside of it, if they are not too many. The main technical part is in proving that if σ is indeed ϵ-far from satisfying Φ, then many of these paths have few such children (few enough to recurs over all of them), where additionally the distance of σ from satisfying the corresponding sub-formulas is significantly larger. An interesting combinatorial corollary of this is that formulas, for which there are not a lot of leaves whose corresponding paths have few such children, do not admit ϵ-far assignments at all.

3.5.1 Critical and important

To understand the intuition behind the following definitions, it is useful to first consider what happens if we could locate a vertex that is "(ϵ,σ)-critical" in the sense that is defined next.

Definition 3.5.1. [(ϵ,σ)***-important,*** (ϵ,σ)***-critical***] A vertex $v\in V$ is (ϵ,σ)*-important* if $\sigma\notin SAT(\Phi)$, and for every u that is either v or an ancestor of v, we have that

- $\texttt{dist}(\sigma, SAT(\Phi_u)) \ge (2\epsilon/3)(1+2\epsilon/3)^{\lfloor \texttt{depth}_\Phi(u)/3\rfloor}$
- If $\kappa_u\equiv\vee$ and $u\neq v$ then the heaviest child of u, $h(u)$ is either v or an ancestor of v.

An (ϵ,σ)-*critical* vertex v is an (ϵ,σ)-important vertex v for which $\kappa_v\in X$.

Note that such a vertex is never too deep, since $\texttt{dist}(\sigma, SAT(\Phi_u))$ is always at most 1. Hence the following observation follows from Definition 3.5.1.

Observation 3.5.2. If v is (ϵ,σ)-important, then $\texttt{depth}_\Phi(v)\le 4\epsilon^{-1}\log(2\epsilon^{-1})$.

A hypothetical oracle that provides a critical vertex can be used as follows. If v is the vertex returned by such an oracle, then for every ancestor u of v such that $\kappa_u=\vee$, and every $w\in\texttt{Children}(u)$ that is not an ancestor of v, a number of recursive calls with Φ_w and distance parameter significantly larger than ϵ are used. The following lemma implies that if for each of these vertices one of the recursive calls returned 0, then we know that $\sigma\notin SAT(\Phi)$.

Definition 3.5.3 (Special relatives)**.** The set of special relatives of $v \in V$ is the set T of every u that is not an ancestor of v or v itself but is a child of an ancestor w of v, where $\kappa_w \equiv \vee$.

Lemma 3.5.4. *If $\sigma \notin SAT(\Phi_u)$ for every $u \in T \cup \{v\}$, then $\sigma \notin SAT(\Phi)$.*

Proof. If $\mathtt{depth}_\Phi(v) = 0$ then $\sigma \notin SAT(\Phi_v)$ implies $\sigma \notin SAT(\Phi)$. Assume by induction that the lemma holds for any formula $\Phi' = (V', E', r', X', \kappa')$, assignment σ' to Φ' and vertex $u \in V'$ such that $0 \leq \mathtt{depth}_{\Phi'}(u) < \mathtt{depth}_\Phi(v)$. Let w be the parent of v. Observe that the special relatives of w are a subset of the special relatives of v and hence by the induction assumption we only need to prove that $\sigma \notin SAT(\Phi_w)$ in order to infer that $\sigma \notin SAT(\Phi)$.

If $\kappa_w \equiv \wedge$, then $\sigma \notin SAT(\Phi_v)$ implies that $\sigma \notin SAT(\Phi_w)$. If $\kappa_w \equiv \vee$, then $\sigma \notin SAT(\Phi_v)$ and $\sigma \notin SAT(\Phi_u)$ for every $u \in T$ implies that $\sigma \notin SAT(\Phi_w)$, since $\mathtt{Children}(w) \setminus \{v\} \subseteq T$. □

The following lemma states that if σ is ϵ-far from $SAT(\Phi)$, then (ϵ, σ)-critical vertices are abundant, and so we can locate one of them by merely sampling a sufficiently large (linear in $1/\epsilon$) number of vertices.

The main part of the proof that this holds is in showing that if σ is only $2\epsilon/3$-far from $SAT(\Phi)$, then there exists an (ϵ, σ)-critical vertex for σ. We first show that this is sufficient to show the claimed abundance of (ϵ, σ)-critical vertices, and then state and prove the required lemma.

Lemma 3.5.5. *If σ is ϵ-far from $SAT(\Phi)$, then $|\{v | v$ is (ϵ, σ)-critical$\}| \geq \epsilon|\Phi|/4$.*

Proof. Set $\mathtt{Critical}_{\epsilon,\sigma} = \{v | v \text{ is } (\epsilon, \sigma)\text{-critical}\}$ and assume the contrary to the lemma statement, that is that $|\mathtt{Critical}_{\epsilon,\sigma}| < \epsilon|\Phi|/4$. Set σ' to be an assignment to X so that for every $s \in V$ where $\kappa_s \in X$, we have that $\sigma'(\kappa_s) = 1$ if $\kappa_s \in \mathtt{Critical}_{\epsilon,\sigma}$ and otherwise $\sigma'(x) = \sigma(x)$. Thus $\mathtt{Critical}_{\epsilon,\sigma'} = \emptyset$. By the triangle inequality we have that

$$\mathtt{dist}(\sigma, SAT(\Phi)) - \mathtt{dist}(\sigma', SAT(\Phi)) \leq \mathtt{dist}(\sigma', \sigma).$$

Finally, since $\mathtt{Critical}_{\epsilon,\sigma'} = \emptyset$, Lemma 3.5.6, which we prove below, asserts the inequality $\mathtt{dist}(\sigma', SAT(\Phi)) < 2\epsilon/3$ and we reach a contradiction. □

Lemma 3.5.6. *If there is no (ϵ, σ)-critical vertex, then σ is $2\epsilon/3$-close to $SAT(\Phi)$.*

Proof. We shall show that if σ is $2\epsilon/3$-far from $SAT(\Phi)$, then there exists an (ϵ, σ)-critical vertex. Assume that σ is $2\epsilon/3$-far from $SAT(\Phi)$. This implies that r is an (ϵ, σ)-important vertex. Hence an (ϵ, σ)-important vertex exists. Let v be an (ϵ, σ)-important vertex such that $\mathtt{depth}_\Phi(v)$ is maximal. Consequently, none of the vertices in $\mathtt{Children}(v)$ are (ϵ, σ)-important. We next prove that v is (ϵ, σ)-critical.

Assume on the contrary that v is not (ϵ, σ)-critical. Consequently $\kappa_v \notin X$ and hence to get a contradiction it is sufficient to show that there exists an (ϵ, σ)-important vertex in $\mathtt{Children}(v)$. If $\kappa_v \equiv \vee$, then by Observation 3.2.18 we get that

$$\mathtt{dist}(\sigma, SAT(\Phi_{h(v)})) \geq (2\epsilon/3)(1 + 2\epsilon/3)^{\lfloor \mathtt{depth}_\Phi(h(v))/3 \rfloor},$$

and hence $h(v)$ is (ϵ, σ)-important.

Assume that $\kappa_v \equiv \wedge$. Let u be such that $\mathtt{dist}(\sigma, SAT(\Phi_u)) \geq \mathtt{dist}(\sigma, SAT(\Phi_v))$. Observation 3.2.17 asserts that such a vertex exists. We assume that $\mathtt{depth}_\Phi(u) > 2$, since otherwise it cannot be the case that $\texttt{dist}(\sigma, SAT(\Phi_u)) < (2\epsilon/3)(1+2\epsilon/3)^0$. Let $w \in V$ be the parent of v. Since w is an ancestor of v it is (ϵ,σ)-important, and hence we have the lower bound $\texttt{dist}(\sigma, SAT(\Phi_w)) \geq (2\epsilon/3)(1+2\epsilon/3)^{\lfloor \texttt{depth}_\Phi(w)/3 \rfloor}$. Since Φ is basic we have that $\kappa_w \equiv \vee$. Thus by Observation 3.2.18 we get that

$$\texttt{dist}(\sigma, SAT(\Phi_v)) \geq (2\epsilon/3)(1+2\epsilon/3)^{1+\lfloor \texttt{depth}_\Phi(w)/3 \rfloor}.$$

Finally since $\texttt{dist}(\sigma, SAT(\Phi_u)) \geq \texttt{dist}(\sigma, SAT(\Phi_v))$ and $\texttt{depth}_\Phi(u) = \texttt{depth}_\Phi(w) + 2$ we get that

$$\texttt{dist}(\sigma, SAT(\Phi_u)) \geq (2\epsilon/3)(1+2\epsilon/3)^{\lfloor \texttt{depth}_\Phi(u)/3 \rfloor}.$$

3.5.2 Algorithm

This algorithm detects far inputs with probability $\Omega(\epsilon)$, but this can be amplified to 2/3 using iterated applications.

Algorithm 3.3 Test satisfiability of basic read-once formula

```
Input: read-once basic formula $\Phi = (V, E, r, X, \kappa)$, a parameter $\epsilon > 0$, oracle to $\sigma$ .
Output: $z \in \{0,1\}$.
 1: if $\epsilon > 1$ then return 1
 2: if $\kappa_r \in X$ then return $\sigma(\kappa_r)$
 3: Pick $s$ uniformly at random from all $v$ such that $\kappa_v \in X$
 4: $A \longleftarrow$ all ancestors $v$ of $s$ such that $\kappa_v \equiv \vee$
 5: $R \longleftarrow (\bigcup_{v\in A} \texttt{Children}(v)) \setminus \{w \mid w \text{ is an ancestor of } s\}$
 6: if $|R| > 3\epsilon^{-2}\log(2\epsilon^{-1})$ then return 1
 7: for all $u \in R$ do
 8:     $y_u \longleftarrow 1$
 9:     for $i = 1$ to $\lceil 20\epsilon^{-1}\log\epsilon^{-1}\rceil$ do $y_u \longleftarrow y_u \wedge$ Algorithm 3.3$(\Phi_u, \sigma, 4\epsilon/3)$
10: return $\sigma(\kappa_s) \vee \bigvee_{u\in R} y_u$
```

We now proceed to prove the correctness of Algorithm 3.3. Algorithm 3.3 is clearly non-adaptive. We first bound its number of queries and next prove that it always returns “1” for an assignment that satisfies the formula, and returns “0” with probability linear in ϵ for an assignment that is ϵ-far from satisfying the formula. Using $O(1/\epsilon)$ independent iterations amplifies the later probability to 2/3.

Lemma 3.5.7. *For $\epsilon > 0$, Algorithm 3.3 halts after using at most $\epsilon^{-16+16\log\epsilon}$ queries, when called with Φ, ϵ and oracle access to σ.*

Proof. The proof is formulated as an inductive argument over the value of the (real) distance parameter ϵ. However, it is formulated in a way that it can be viewed as an inductive argument over the integer valued $\lceil\log(\alpha\epsilon^{-1})\rceil$, for an appropriate global constant α. This is since the value of the distance parameter increases multiplicatively with every recursive call.

If $\epsilon > 1$, then the condition in Line 1 is satisfied, and there are no queries or recursive calls. Hence we assume that $\epsilon \leq 1$. Observe that in a specific instantiation at most one query is used, since a query is only made on Line 2 or on Line 10, and always as part of a “return” command.

Hence the number of queries is upper bounded by the number of calls to Algorithm 3.3 (initial and recursive). We shall show that the number of these calls is at most $\epsilon^{-16+16\log\epsilon}$.

Assume by induction that for some $\eta \leq 1$, for every $\eta \leq \eta' \leq 1$, every formula Φ' and assignment σ' to Φ', on call to Algorithm 3.3 with Φ', η' and an oracle to σ', at most ${\eta'}^{-16+16\log\eta'}$ calls to Algorithm 3.3 are made (including recursive ones).

Assume that $\epsilon > 3\eta/4$. If $\kappa_r \in X$, then the condition on Line 2 is satisfied and hence there are no recursive calls. Thus Algorithm 3.3 is called only once and $1 \leq \epsilon^{-16+16\log\epsilon}$.

Assume that $\kappa_r \notin X$. Note that every recursive call is done by Line 9. By Line 7 and Line 9 at most $|R| \cdot \lceil 20\epsilon^{-1}\log\epsilon^{-1}\rceil$ recursive calls are done. The condition on Line 6 ensures that $|R| \cdot \lceil 20\epsilon^{-1}\log\epsilon^{-1}\rceil \leq 3\epsilon^{-2}\log(2\epsilon^{-1}) \cdot \lceil 20\epsilon^{-1}\log\epsilon^{-1}\rceil$. According to Line 9 each one of these recursive calls is done with distance parameter $4\epsilon/3 > \eta$. Thus by the induction assumption the number of calls to Algorithm 3.3 is at most

$$3\epsilon^{-2}\log(2\epsilon^{-1}) \cdot \lceil 20\epsilon^{-1}\log\epsilon^{-1}\rceil \cdot (4\epsilon/3)^{-16+16\log(4\epsilon/3)}.$$

This is less than $\epsilon^{-16+16\log\epsilon}$. □

The following theorem will be immediate from Lemma 3.5.7 above when coupled with Lemma 3.5.8 and Lemma 3.5.10 below.

Theorem 3.4. *Let $\epsilon > 0$. When Algorithm 3.3 is called with Φ, ϵ and an oracle to σ, it uses at most $\epsilon^{-16+16\log\epsilon}$ queries; if $\sigma \in SAT(\Phi)$ then it always returns 1, and if σ is ϵ-far from $SAT(\Phi)$ then it returns 0 with probability at least $\epsilon/8$.*

Theorem 3.4 does not imply that Algorithm 3.3 is an ϵ-test for $SAT(\Phi)$. However it does imply that in order to get an ϵ-test for $SAT(\Phi)$ it is sufficient to do the following. Call Algorithm 3.3 repeatedly $\lceil 20\epsilon^{-1}\rceil$ times, return 0 if any of the calls returned 0, and otherwise return 1. This only increases the query complexity to the value in the following corollary.

Corollary 3.5. *There exists an ϵ-test for Φ, that uses at most $\epsilon^{-20+16\log\epsilon}$ queries.*

Lemma 3.5.8. *Let $\epsilon > 0$ and $\sigma \in SAT(\Phi)$. Algorithm 3.3 returns 1 when called with Φ, ϵ and an oracle to σ.*

Proof. To prove the lemma we shall show that if Algorithm 3.3 returns 0, when called with Φ, ϵ and oracle access to σ, then $\sigma \notin SAT(\Phi)$. We will show this by induction over $\texttt{depth}(\Phi)$. If $\texttt{depth}(\Phi) = 0$ then the condition in Line 1 is satisfied and $\sigma(\kappa_r)$ is returned. Hence $\sigma(\kappa_r) = 0$ and therefore $\sigma \notin SAT(\Phi)$. Assume that for every $\epsilon' > 0$, Φ' where $\texttt{depth}(\Phi') < \texttt{depth}(\Phi)$, and assignment σ' to Φ', if Algorithm 3.3 returns 0, when called with Φ', ϵ' and oracle access to σ', then $\sigma' \notin SAT(\Phi)$.

Observe that the only other way a 0 can be returned is through Line 10, if it is reached. Let R be the set of vertices on which there was a recursive call in Line 9 and κ_s the variable whose value is queried on Line 10. According to Line 10 a 0 is returned if and only if $\sigma(\kappa_s) = 0$, and for every $u \in R$, there was at least one recursive call with Φ_u and distance parameter $4\epsilon/3$ that returned a 0. By the induction assumption this implies that $\sigma \notin SAT(\Phi_u)$ for every $u \in R$. Note that the set R satisfies the exact same conditions that the set T of special relatives satisfies in Lemma 3.5.4. Hence, Lemma 3.5.4 asserts that $\sigma \notin SAT(\Phi)$. □

We now turn to proving soundness. This depends on first noting that the algorithm will indeed check the paths leading to critical vertices.

Observation 3.5.9. If the vertex s picked in Line 3 is (ϵ,σ)-critical, then it will not trigger the condition of Line 6.

Proof. Definition 3.5.1 in particular implies (see observation 3.2.18) that for every $u \in A$ (as per Line 4) we have $|\texttt{Children}(u)| \leq (3/2\epsilon)(1+2\epsilon/3)^{-\lfloor \texttt{depth}_\Phi(u)/3 \rfloor} \leq 3/2\epsilon$, as otherwise σ will be too close to satisfying Φ_u. Also, from Observation 3.5.2 we know that $\texttt{depth}_\Phi(s) \leq 4\epsilon^{-1}\log(2\epsilon^{-1})$ and so $|A| \leq 2\epsilon^{-1}\log(2\epsilon^{-1}) + 1$.

The two together give us the bound $|R| \leq (3/2\epsilon - 1)(2\epsilon^{-1}\log(2\epsilon^{-1}) + 1) \leq 3\epsilon^{-2}\log(2\epsilon^{-1})$, and so the condition in Line 3 is not triggered. □

Lemma 3.5.10. *Let σ be ϵ-far from $SAT(\Phi)$. If Algorithm 3.3 is called with ϵ, Φ and an oracle to σ, then it returns 0 with probability at least $\epsilon/8$.*

Proof. We will prove this by induction on $\texttt{depth}(\Phi)$.

The base case, $\kappa_r \in X$, is handled correctly by Line 1. Assume next that $\epsilon > 3/4$. Assume first that the vertex s selected in Line 3 is (ϵ,σ)-critical. By Lemma 3.5.5, with probability at least $3/16$ the vertex s selected in Line 3 is indeed (ϵ,σ)-critical. Hence by definition σ is more than $1/2$-far from $SAT(\Phi_u)$ for every ancestor u of s. Thus by Observation 3.2.18 we have that $\kappa_u \equiv \wedge$ for every ancestor u of s. Consequently, by Line 2 and Line 10 the value returned will be $\sigma(\kappa_s)$, and $\sigma(\kappa_s) = 0$ because s is (ϵ,σ)-critical.

Thus, 0 is returned with probability at least $3/16$, which is greater than $\epsilon/8$ when $3/4 < \epsilon \leq 1$.

For all other ϵ we proceed with the induction step. Assume that for any formula Φ' such that $\texttt{depth}(\Phi') < \texttt{depth}(\Phi)$ and any assignment σ' to Φ' that is η-far from $SAT(\Phi')$ (for any η), Algorithm 3.3 returns 0 with probability at least $\eta/8$. Given this we prove that 0 is returned with probability at least $\epsilon/8$ for Φ and σ.

Assume first that the vertex s selected in Line 3 is (ϵ,σ)-critical. Let A, R be the sets from Line 4 and Line 5. Since s is (ϵ,σ)-critical, by definition for every $u \in A$ we have that σ is $2\epsilon/3$-far from $SAT(\Phi_u)$. Also, because s is (ϵ,σ)-critical, by definition for every $u \in A$ and $w \in \texttt{Children}(u) \cap R$ we have that $w \neq h(u)$, and therefore by Observation 3.2.18 we have that σ is $4\epsilon/3$-far from $SAT(\Phi_w)$ for every $w \in R$.

By the induction assumption, for every $w \in R$, with probability at least $1 - (4\epsilon/3)/8$ Algorithm 3.3 returns 0 when called with $4\epsilon/3$, Φ_w and an oracle to σ. Hence, for every $w \in R$, the probability that on $\lceil 20\epsilon^{-1}\log\epsilon^{-1}\rceil$ such independent calls to Algorithm 3.3 the value 0 was never returned is at most $(1-(4\epsilon/3)/8)^{\lceil 20\epsilon^{-1}\log\epsilon^{-1}\rceil}$. This is less than $(\epsilon^{-2}\log(2\epsilon^{-1}))/6$.

Observation 3.5.9 ensures that $|R| \leq 3\epsilon^{-2}\log(2\epsilon^{-1})$, and in particular the condition in Line 6 is not invoked and the calls in Line 9 indeed take place. By the union bound over the vertices of R, with probability at least $1/2$, for every $u \in R$ at least one of calls to Algorithm 3.3 with $4\epsilon/3$, Φ_u and an oracle to σ returned the value 0. This means that for every $u \in R$, y_u in Line 9 was set to 0. Consequently this is the value returned in Line 10

Finally, since σ is ϵ-far from $SAT(\Phi)$, by Lemma 3.5.5 the vertex s selected in Line 3 is (ϵ,σ)-critical with probability at least $\epsilon/4$. Therefore 0 is returned with probability at least $\epsilon/8$. □

3.6 The computational complexity of the testers and estimator

There are two parts to analyzing the computational complexity (as opposed to query complexity) of a test for a massively parametrized property. The first part is the running time of the preprocessing phase, which reads the entire parameter part of the input, in our case the formula, but has no access yet to the tested part, in our case the assignment. This part is subject to traditional running time and working space definitions, and ideally should have a running time that is quasi-linear or at least polynomial in the size of its input (the "massive parameter"). The second part is the testing part, which ideally should take a time that is logarithmic in the input size for every query it makes (as a very basic example, even a tester that just makes independent uniformly random queries over the input would require such a time to draw the necessary $\log(n)$ random coins for each query).

In our case, the preprocessing part would need to take a k-ary formula and convert it to the basic form corresponding to the algorithm that we run. We may assume that the formula is represented as a graph with additional information stored in the vertices.

Constructing the basic form by itself can be done very efficiently (and also have an output size linear in its input size). For example, if the input formula has only "$\wedge$" and "$\vee$" gates, then a Depth First Search over the input would do nicely, where the output would follow this traversal, but create a new child gate in the output only when it is different than its parent (otherwise it would continue traversing the input while remaining in the same output node). With more general monotone gates, a first pass would convert them to unforceable gates by "splitting off" forceful children as in the proof of Lemma 3.2.13. It is not hard to efficiently handle "$\neg$" gates using De-Morgan's law too.

Aside from the basic form of the formula, the preprocessing part should construct several additional data structures to make the second part (the test itself) as efficient as possible.

For Algorithm 3.1, we would need to quickly pick a child of a vertex with probability proportional to its sub-formula size, and know who are the light children as well as what is the relative size of the smallest child. This mainly requires storing the size of every sub-formula for every vertex of the tree, as well as sorting the children of every vertex by their sizes and storing the value of the corresponding "ℓ". Algorithm 3.2 requires very much the same additional data as Algorithm 3.1. This information can be stored in the vertices of the graph while performing a depth-first traversal of it, starting at the root, requiring a time linear in the size of the basic formula.

For Algorithm 3.3, we would need to navigate the tree both downwards and upwards (for finding the ancestors of a vertex), as well as the ability to pick a vertex corresponding to a variable at random, which in itself does not require special preprocessing but does require generating a list of all such vertices. Constructing the set of ancestors is simply following the path from the vertex to the root, requiring time linear in the depth of the vertex in the tree, can be deferred to the testing part and charged to the recursive calls, while in the preprocessing stage we only store parents.

The only part in the algorithms above that depends on ϵ is designating the light children, but this can also be done "for all ϵ" at a low cost by storing the range of ϵ for every positive ℓ. Since ℓ is always an integer no larger than $k+1$, this requires an array of such size in every vertex.

Let us turn to analyzing the running time complexity of the second part, namely the testing algorithm. Once the above preprocessing is performed, the time per instantiation (and thus per query) of the algorithm will be very small (where we charge the time it takes to calculate a recursive call to the recursive instantiation). In Algorithm 3.1, the cost in every instantiation is at most the cost of selecting a child vertex at random for each iteration of the loop in line 6, a cost linear in k for performing the calls in Lines 17 and 18 and a cost of $O(2^k)$ for searching the space of possible x's in Line 19. This would make it a cost logarithmic in the input size per query (multiplied by the time it takes to write and read an address) – where the log incurrence is in fact only when we need to randomly choose a child according to its weight. The case of Algorithm 3.2 is similar, except that while we don't have the cost of iterating over possible assignments to the root, there is an additional constant cost for every term in the mDNF, of which there are at most 2^k.

For Algorithm 3.3, every instantiation requires iterating over all the ancestors of one vertex picked at random. This requires time linear in the depth of the formula and logarithmic in the input size per query, where the depth only depends on the distance parameter (see observation 3.5.2).

3.7 The untestable formulas

We describe here a read-once formula over an alphabet with 4 values, defining a property that cannot be 1/4-tested using a constant number of queries. The formula will have a very simple structure, with only one gate type. Then, building on this construction, we describe a read-once formula over an alphabet with 5-values that cannot be 1/12-tested, which satisfies an additional monotonicity condition: All gates as well as the acceptance condition are monotone with respect to a fixed ordering of the alphabet.

3.7.1 The 4-valued formula

For convenience we denote our alphabet by $\Sigma = \{0, 1, P, F\}$. An input is said to be *accepted* by the formula if, after performing the calculations in the gates, the value received at the root of the tree is not "F". We restrict the input variables to $\{0, 1\}$, although it is easy to see that the following argument holds also if we allow other values to the input variables (and also if we change the acceptance condition to the value at the root having to be "P").

Definition 3.7.1. The *balancing gate* is the gate that receives two inputs from Σ and outputs the following.

- For $(0, 0)$ the output is 0 and for $(1, 1)$ the output is 1.
- For $(1, 0)$ and $(0, 1)$ the output is P.
- For (P, P) the output is P,

- For anything else the output is F.

For a fixed $h > 0$, the *balancing formula* of height h is the formula defined by the following.

- The tree is the full balanced binary tree of height h with variables at the leaves, and hence there are 2^h variables.
- All gates are set to the balancing gate.
- The formula accepts if the value output at the root is not "F".

We denote the variables of the formula in their order by $x_0, \ldots, x_{2^h-1}$. The following is easy.

Lemma 3.7.2. *An assignment $a_0 \in \{0,1\}, \ldots, a_{2^h-1} \in \{0,1\}$ to $x_0, \ldots, x_{2^h-1}$ is accepted by the formula if and only if for every $0 < k \leq h$ and every $0 \leq i < 2^{h-k}$, the number of 1 values in $a_{i2^k}, \ldots, a_{(i+1)2^k-1}$ is either 0, 2^k or 2^{k-1}.*

Proof. Denote the number of 1 values in variables descending from a gate u by $\text{num}_1(u)$. Note that $a_{i2^k}, \ldots, a_{(i+1)2^k-1}$ are the set of descendant leaves of a single vertex, denote it by v. We prove by induction on k that:

- $\text{num}_1(v) = 0$ if and only if the value of v is 0.
- $\text{num}_1(v) = 2^k$ if and only if the value of v is 1.
- If the value of v is P then $\text{num}_1(v) = 2^{k-1}$.
- If $\text{num}_1(v) \notin \{0, 2^{k-1}, 2^k\}$ then the value of v is F.

For $k = 1$ we have the two inputs of v, and by the definition of the balancing gate the claim follows.

For $k > 1$, if at least one of the children of v evaluates to F then so was v (and so does the entire formula) and by the induction hypothesis one of the descendants of its children doesn't have the correct number of 1 values. If neither of them evaluates to F then by the induction hypothesis for both children of v, denoted u, w, we have that $\text{num}_1(u), \text{num}_1(w) \in \{0, 2^{k-2}, 2^{k-1}\}$ and that this determines their value. If $\text{num}_1(w) = \text{num}_1(u) = 0$ then they both evaluate to 0 and so does v. Similarly, if $\text{num}_1(w) = \text{num}_1(u) = 2^{k-1}$ then both evaluate to 1 and so does v. If $\text{num}_1(u) = 2^{k-1}$ and $\text{num}_1(w) = 0$, then u evaluates to 1 and w to 0, and indeed v evaluates to P (and similarly for the symmetric case). If $\text{num}_1(u) = \text{num}_1(w) = 2^{k-2}$, then both evaluate to P and so does v. The remaining case is $\text{num}_1(u) \in \{0, 2^{k-1}\}$ and $\text{num}_1(w) = 2^{k-2}$ (and the symmetric case). Here the induction hypothesis and the definition of the balancing gate implies that v evaluates to F and the formula is unsatisfied, while the interval of all descendant variables of v does not have the correct number of 1 values. □

In other words, for every satisfying assignment every "binary search interval" is either all 0, or all 1, or has the same number of 0 and 1. This will allow us to easily prove that certain inputs are far from satisfying the property.

3.7.2 Two distributions

We now define two distributions, one over satisfying inputs and the other over far inputs.

Definition 3.7.3. The distribution D_Y is defined by the following process.

- Uniformly pick $2 \le k \le h$.
- For every $0 \le i < 2^{h-k}$, independently pick either $(y_{i,0}, y_{i,1}) = (0,1)$ or $(y_{i,0}, y_{i,1}) = (1,0)$ (each with probability $1/2$).
- For every $0 \le i < 2^{h-k}$, set

$$x_{i2^k} = \cdots = x_{i2^k+2^{k-1}-1} = y_{i,0}; \qquad x_{i2^k+2^{k-1}} = \cdots = x_{(i+1)2^k-1} = y_{i,1}.$$

Definition 3.7.4. The distribution D_N is defined by the following process.

- Uniformly pick $2 \le k \le h$.
- For every $0 \le i < 2^{h-k}$, independently choose $(z_{i,0}, z_{i,1}, z_{i,2}, z_{i,3})$ to have either one 1 and three 0 or one 0 and three 1 (each of the 8 possibilities with probability $1/8$).
- For every $0 \le i < 2^{h-k}$, set

$$x_{i2^k} = \cdots = x_{i2^k+2^{k-2}-1} = z_{i,0}; \qquad x_{i2^k+2^{k-2}} = \cdots = x_{i2^k+2^{k-1}-1} = z_{i,1};$$

$$x_{i2^k+2^{k-1}} = \cdots = x_{i2^k+2^{k-1}+2^{k-2}-1} = z_{i,2}; \qquad x_{i2^k+2^{k-1}+2^{k-2}} = \cdots = x_{(i+1)2^k-1} = z_{i,3}.$$

It is easier to illustrate this by considering the calculation that results from the distributions. In both distributions we can think of a randomly selected level k (counted from the bottom, where the leaf level 0 and the level above it, 1, are never selected). In D_Y, the output of all gates at or above level k is "P", while the inputs to every gate at level k will be either $(0,1)$ or $(1,0)$, chosen uniformly at random.

In D_N all gates at level k will output "F" (note however that we cannot query a gate output directly); looking two levels below, every gate as above holds the result from a quadruple chosen uniformly from the 8 choices described in the definition of D_N (the quadruple $(z_{i,0}, z_{i,1}, z_{i,2}, z_{i,3})$). At level $k-2$ or lower the gate outputs are 0 and 1 and their distribution resembles very much the distribution as in the case for D_Y, as long as we cannot "focus" on the transition level k. This is formalized in terms of lowest common ancestors below.

Lemma 3.7.5. *Let $Q \subset \{1, \ldots, 2^h\}$ be a set of queries, and let $H \subset \{0, \ldots, h\}$ be the set of levels containing lowest common ancestors of subsets of Q. Conditioned on neither k nor $k-1$ being in H, both D_Y and D_N induce exactly the same distribution over the outcome of querying Q.*

Proof. Let us condition the two distributions on a specific value of k satisfying the above. For two queries $q, q' \in Q$ whose lowest common ancestor is on a level below $k-1$, with probability 1 they will receive the exact same value (this holds for both D_N and D_Y). The reason is clear from the construction – their values will come from the same $y_{i,j}$ or $z_{i,j}$.

Now let Q' contain one representative from every set of queries in Q that must receive the same value by the above argument. For any $q, q' \in Q'$, their lowest common ancestor is on a level above k. For D_Y it means that x_q takes its value from some $y_{i,j}$ and $x_{q'}$ takes its value from some $y_{i',j'}$ where $i \neq i'$. Because each pair $(y_{i,0}, y_{i,1})$ is chosen independently from all other pairs, this means that the outcome of the queries in Q' is uniformly distributed among all $2^{|Q'|}$ possibilities. The same argument (with $z_{i,j}$ and $z_{i',j'}$ instead of $y_{i,j}$ and $y_{i',j'}$) holds for D_N. Hence the distribution of outcomes over Q' is the same for both distributions, and by extension this holds over Q. □

On the other hand, the two distributions are very different with respect to satisfying the formula.

Lemma 3.7.6. *An input chosen according to D_Y always satisfies the balancing formula, while an input chosen according to D_N is always $1/4$-far from satisfying it.*

Proof. By Lemma 3.7.2, the assignment constructed in D_Y will always be satisfied. This is since for every vertex in a level lower than k, all of its descendant variables will be of the same value, while for every vertex in level k or above, exactly half of the variables will have each value.

Note that in an input constructed according to D_N, every vertex at level k has one quarter of its descendant variables of one value, while the rest are of the other value. By averaging, if one were to change less than $1/4$ of the input values, we would have a vertex v at level k for which less than $1/4$ of the values of its descendant variables were changed. This means that v cannot satisfy the requirements in Lemma 3.7.2, and therefor it and hence the entire formula evaluate to F. □

3.7.3 Proving non-testability

We use here Lemma **??** introduced in Chapter 2 to conclude the proof.

Theorem 3.6. *Testing for being a satisfying assignment of the balancing formula of height h requires at least $\Omega(h)$ queries for a non-adaptive test and $\Omega(\log h)$ queries for a possibly adaptive one.*

Proof. We note that for any set of queries Q, the size of the set of lowest common ancestors outside Q itself is less than $|Q|$, and hence (in the notation of Lemma 3.7.5) we have $|H| \leq 2|Q|$. If $|Q| = o(h)$, then the event of Lemma 3.7.5 happens with probability $1 - o(1)$, and hence the variation distance between the two (unconditional) distributions over outcomes is $o(1)$. Together with Lemma 3.7.6 this fulfills the conditions for Lemma **??** for concluding the proof for non-adaptive algorithms.

For adaptive algorithms the bound follows by the standard procedure that makes an adaptive algorithm into a non-adaptive one at an exponential cost, by querying in advance the algorithm's entire decision tree given its internal coin tosses. □

3.7.4 An untestable 5-valued monotone formula

While the lower bound given above uses a gate which is highly non-monotone, we can also give a similar construction where the alphabet is of size 5 and the gates are monotone (that is, where increasing any input of the gate according to the order of the alphabet does not decrease its input).

Instead of just "$\{1,\ldots,5\}$" we denote our alphabet by $\Sigma = \{0, F_0, P, F_1, 1\}$ in that order. We will restrict the input variables to $\{0,1\}$, although it is not hard to generalize to the case where the input variables may take any value in the alphabet. At first we analyze a formula that has a non-monotone satisfying condition.

Definition 3.7.7. The *monotone balancing gate* is the gate that receives two inputs from Σ and outputs the following.

- For $(0,0)$ the output is 0 and for $(1,1)$ the output is 1.
- For $(1,0)$ and $(0,1)$ the output is P.
- For (P,P) the output is P.
- For $(0,P)$ and $(P,0)$ the output is F_0.
- For $(1,P)$ and $(P,1)$ the output is F_1.
- For (P,F_0), (F_0,P), $(F_0,0)$, $(0,F_0)$ and (F_0,F_0) the output is F_0.
- For $(F_0,1)$ and $(1,F_0)$ the output is F_1.
- For any pair of inputs containing F_1, the output is F_1.

For a fixed $h > 0$, the *almost-monotone balancing formula* of height h is the formula defined by the following.

- The tree is the full balanced binary tree of height h with variables at the leaves, and hence there are 2^h variables.
- All gates are set to the monotone balancing gate.
- The formula accepts if the value output at the root is not "F_0" or "F_1".

The following observation is easy by just running over all possible outcomes of the gate.

Observation 3.7.8. The monotone balancing gate is monotone. Additionally, if the values F_0 and F_1 are unified then the gate is still well-defined, and is isomorphic to the 4-valued balancing gate.

In particular, the above observation implies that the almost-monotone balancing formula has the same property testing lower bound as that of the balancing formula, using the same proof with the same distributions D_Y and D_N. However, we would like a completely monotone formula. For that we use a monotone decreasing acceptance condition; we note that a formula with a monotone increasing acceptance condition can be obtained from it by just "reversing" the order over the alphabet.

Definition 3.7.9. The *monotone sub-balancing formula* is defined the same as the almost-monotone balancing formula, with the exception that the formula accepts if and only if the value output at the root is not F_1 or 1.

By Observation 3.7.8, the distribution D_Y is also supported by inputs satisfying the monotone sub-balancing formula. To analyze D_N, note the following.

Lemma 3.7.10. *An assignment $a_0 \in \{0,1\}, \dots, a_{2^h-1} \in \{0,1\}$ to $x_0, \dots, x_{2^h-1}$, for which for some $0 < k \leq h$ and some $0 \leq i < 2^{h-k}$ the number of 1 values in $a_{i2^k}, \dots, a_{(i+1)2^k-1}$ is more than 2^{k-1} and less than 2^k, cannot be accepted by the formula.*

Proof. We set u to be the gate whose descendant variables are exactly $a_{i2^k}, \dots, a_{(i+1)2^k-1}$. We first note that it is enough to prove that u evaluates to F_1, because then by the definition of the gates the root will also evaluate to F_1. We then use induction over k, while referring to Observation 3.7.8 and the proof of Lemma 3.7.2. The base case $k = 1$ is true because then no assignment satisfies the conditions of the lemma.

If any of the two children of u evaluates to F_1 then we are also done by the definition of the gate. The only other possible scenario (using induction) is when one of the children v of u must evaluate to 1, and hence all of its 2^{k-1} descendant variables are 1, while for the other child w of u some of the descendant variables are 0 and some are 1. But this means that w does not evaluate to either 0 or 1, which again means that u evaluates to F_1. □

This yields the following.

Lemma 3.7.11. *With probability $1 - o(1)$, an input chosen according to D_N will be $1/12$-far from satisfying the monotone sub-balancing formula.*

Proof. This is almost immediate from Lemma 3.7.10, as a large deviation inequality implies that with probability $1 - o(1)$, more than $1/3$ of the quadruples $(z_{i,0}, z_{i,1}, z_{i,2}, z_{i,3})$ as per the definition of D_N will have three 1's and one 0. □

Now we can prove a final lower bound.

Theorem 3.7. *Testing for being a satisfying assignment of the monotone sub-balancing formula of height h requires at least $\Omega(h)$ queries for a non-adaptive test and $\Omega(\log h)$ queries for a possibly adaptive one.*

Proof. This follows exactly the proof of the lower bound for the balancing formula. Due to Observation 3.7.8 and Lemma 3.7.11 we can use the same D_Y and D_N, since the $o(1)$ probability of D_N not producing a far input makes no essential difference for the use of Yao's method. □

Chapter 4

Testing Tree Coloring Properties

4.1 Introduction

Fixing a tree T and some finite color set C, a *tree coloring property* of T is a subset P of all possible functions $c : V(T) \to C$ assigning the vertices of T with colors from C. The tree T is a parameter to the algorithm, and is fully known, whereas the coloring function c is unknown and has to be queried. The goal of the algorithm is to decide whether c has the property or is far from it using as few queries as possible. The distance of c from the property is measured by the fraction of entries that need to be changed so that c would be in P, while T itself is immutable. This is an instance of a *massively parameterized model.*

Possibly the first case where coloring properties were considered, albeit implicitly, was in the work of Fischer et. al. where they considered monotonicity testing over general posets [FLN+02]. The most relevant to our case is tree-monotonicity, where we fix a rooted tree T and take $C = \{1, 2, \dots, k\}$, where the property we demand from the coloring is that no vertex v colored $c(v)$ has a descendant with a color strictly smaller than its own. This can be tested using $O(\epsilon^{-1})$ queries [FLN+02].

Several works were published dealing explicitly with tree coloring properties. Fischer and Yahalom [FY11] studied convexity and related properties. A coloring of a tree is *convex* if the preimage of every color is a connected subtree. Fischer and Yahalom show that this property, as well as several related properties and generalizations, can be tested using a number of queries depending only on the distance parameter. Note that when the tree is rooted, testing convexity actually amounts to testing freeness from certain topological subtrees (as do the other problems considered in [FY11]). Later, Yahalom [Yah08, Yah12] considered a more abstract tree coloring property, namely, the property of a tree coloring being free from a family of forbidden paths, and the property of a tree coloring being free from a single tree minor.

Tree coloring properties were also considered in a more applied context. Ndione et. al. [NNL13] studied DTD validity properties. These are properties of colored trees, where the tree is not known in advance. The distance in this case is with respect to three operations: changing a vertex's color, removing a leaf from the tree and adding a leaf to the tree. A DTD property specifies for every possible color a regular expression for the allowed coloring of its children. In particular, the lower bound we give in Section 4.5 can be put in terms of a DTD property, the difference being that in our lower bound the underlying tree is known to the algorithm in advance.

Czumaj et. al. [CGR$^+$14] studied testing of subtree minor freeness in a different setting. Czumaj et. al. develop an algorithm for testing whether a given tree H is a subtree minor of a bounded degree graph G. Their model is different in several key aspects: (1) The trees and graphs in the work of Czumaj et. al. are unlabeled, (2) they look for the graph H in a bounded degree graph G rather than a tree, (3) they measures distance by changes to the graph G, and our work sees the underlying tree T as immutable and measures distance by changes to only the coloring of T.

Tree coloring can also be an abstraction of formula satisfaction problems. In such an abstraction, the coloring of the tree is the values given to the input nodes, as well as the values calculated in the intermediate nodes. The property to be tested is that of the coloring corresponding to an accepting input. This viewpoint was taken by Halevy et. al. [HLNT07], who used a different model of graph coloring problems (in which edges are colored) to prove that every property of boolean strings that can be represented by a read-twice CNF formula is testable.

4.1.1 Topological subtrees

The notion of a *Topological subtree* (and more generally a topological subgraph) is a classic notion in graph theory, appearing under various names, which has found much use in computer science. Roughly, a tree H is a topological subtree of a tree T if we can map the vertices of H to those of T such that every edge will be mapped to a path, with all of these paths being disjoint outside their end-nodes. In applications we usually view the subtree H as representing some *partial information* that we are trying to match to the information in T.

One such example is in incomplete information queries to heterogenous data-centric XML databases [NNL13]. Suppose we have various entries representing books, and we are interested in finding books with contributions by “Joseph Conrad” discussing “boats”. The problem is that “Joseph Conrad” might be listed as an author, editor, or perhaps the author of a chapter. The fact that the book discusses “boats” might appear in a keyword list, in a list of themes, or maybe just on the title. What we would like to query is whether we can find a topological subtree, where the root is a book entry, one of the children is an entry labeled “Joseph Conrad” and another is labeled “boats” (this example is drawn from Schlieder and Naumann [SN00]). In this case we need to use the notion of a topological subtree because we only have partial information on where we expect the different pieces of information to appear relative to one another. This notion is sometimes called *tree inclusion* [KM93] or *constrained tree inclusion* [Val05] in the information retrieval and pattern matching literature.

An important thing to note about this example is that it actually requires the appearance of a *labeled* topological subtree, rather than just the subtree structure itself. Also note that in some applications we might not require that the mapping sends the edges to disjoint paths, in which case we call it a *subtree minor*.

In this work we give an algorithm for testing whether a coloring is free from a constant sized family of forbidden topological subtrees, for the case where the underlying tree T is of bounded degree. This also solves the problem for the case of a constant sized family of forbidden subtree minors, by a manipulation of the family. In this work we view the underlying tree T as being rooted and ordered, but it can also be made unordered, again by manipulating the forbidden family.

4.2 Preliminaries and statement of main result

We will present an algorithm to test certain properties of tree colorings. For this we need to define the relevant notion of distance between colorings, and follow it with a definition of the notion of property testing as adapted for tree colorings. For defining distance we use the normalized Hamming norm. We rely on definitions of the basic notions of ordered trees from subsection 2.1.

Definition 4.2.1. Given two colorings c_1, c_2 of T, the *distance* between c_1 and c_2 is the fraction of vertices on which c_1 and c_2 disagree. When considering a coloring c and a set of colorings P, we will define the distance between c and P to be the minimal distance between c and a member of P.

The properties considered are sets of colorings of a fixed tree.

Definition 4.2.2. Given an ordered tree T, a *property* of colorings of T with C colors is a subset P of all possible colorings of T. That is, $\mathrm{P} \subseteq C^{V(T)}$.

Note that we do not consider colorings of different underlying trees. One can see colorings of different trees to be of infinite distance from each other. The algorithms in this setting are supplied with query access to the coloring function $c : V(T) \to C$, and the number of queries they require will be the main computational resource that we consider.

Definition 4.2.3. Given an ordered tree T and a property P of colorings of T with C colors, a *property tester* for P is a randomized algorithm A such that for any $\epsilon > 0, 1 > \delta > 0$, given a coloring $c : V(T) \to C$, if $c \in \mathrm{P}$ then A accepts with probability at least $1 - \delta$, and if c is ϵ-far from P then A rejects with probability at least $1 - \delta$. The algorithm A is given T, ϵ and δ in advance.

In the case where the algorithm never rejects a coloring $c \in \mathrm{P}$ we say that it is *one-sided*, and otherwise the algorithm is *two-sided*.

The algorithms in this section will all be one-sided. The properties of colorings that we consider will be defined by freeness from some set of forbidden topological subgraphs. Let us define this formally:

Definition 4.2.4. Given a colored ordered binary tree T_1 and a "forbidden" colored ordered binary tree T_2, we say that T_1 *contains* T_2 if there exists an injective mapping $\mathrm{I} : V(T_2) \to V(T_1)$ such that the following holds for all $u, v \in V(T_2)$:

- $c(v) = c(\mathrm{I}(v))$

- u is a descendant of the ith child of v if and only if $\mathrm{I}(u)$ is a descendant of the ith child of $\mathrm{I}(v)$
- If w is the lowest common ancestor of u and v then $\mathrm{I}(w)$ is the lowest common ancestor of $\mathrm{I}(u)$ and $\mathrm{I}(v)$

That is, the function I preserves color, and respects the tree orders and lowest common ancestry.

This can alternatively be stated as "T_2 appears in T_1 as an order preserving topological subgraph with matching colors". We can now formally define being free of the forbidden subgraphs:

Definition 4.2.5. Given a finite family of colored trees $\mathcal{F}$, we say that a coloring c of an ordered tree T is *free* of $\mathcal{F}$ if it does not contain any of the elements of $\mathcal{F}$ in the sense of Definition 4.2.4. The set of all colorings of T that are free from $\mathcal{F}$ will be denoted by $\mathrm{P}_{\mathcal{F}}$.

Note that actually $\mathrm{P}_{\mathcal{F}}$ also depends on T, but we will keep this implicit. When dealing with families of forbidden colored trees, we will denote the number of trees in the family by $|\mathcal{F}|$ and the set of all vertices of trees in the family by $V(\mathcal{F})$ (where the trees are vertex disjoint).

The main result that we prove is the following:

Theorem 4.1. *For every $\epsilon > 0$ and $1 > \delta > 0$, every tree T with degree bound d, and every forbidden family $\mathcal{F}$, there exists an algorithm that queries the colors of at most $q(\epsilon, \delta, d, |V(\mathcal{F})|)$ vertices, always accepts colorings in $\mathrm{P}_{\mathcal{F}}$, and rejects all colorings ϵ-far from $\mathrm{P}_{\mathcal{F}}$ with probability at least $1 - \delta$.*

4.3 Overview

The general method of the algorithm is one of divide and conquer. The algorithm samples a vertex v (the manner by which it is sampled will depend on circumstances as defined below) and then attempts to find a some evidence that a tree from $\mathcal{F}$ cannot appear rooted in v. It does so by considering several different forbidden families on the subtrees rooted in the children of v. Let us now explain what are those families.

We denote by $\mathcal{F}[i]$ the subset of $\mathcal{F}$ containing only those trees whose root is colored i. We denote by $R(\mathcal{F})$ the set of all colors of roots of trees in $\mathcal{F}$. The set of colors used in the property, that is, the range of c, is denoted by C.

Since the algorithm will be traversing the tree T and the trees in $\mathcal{F}$, it will be useful to introduce some notation for that. Given a tree T and a vertex $v \in V(T)$, we denote by T_v the subtree rooted in v. v's ith child vertex will be denoted v_i. We call the subtree rooted in the ith child of a vertex $v \in V(T)$ "the ith subtree of v", and denote it by T_{v_i}. If v is the root of T, then we also denote its subtrees by T_i for $i \in \{1, \dots, d\}$. Note that if, for example, v has no first child then T_{v_1} is the empty tree.

Definition 4.3.1. Given two families of ordered colored trees, $\mathcal{G}, \mathcal{F}$, we will say that $\mathcal{G}$ is *stronger* than $\mathcal{F}$ if $\mathcal{F} \neq \mathcal{G}$ and for any tree $t \in \mathcal{F}$, either it or some subtree of it is in $\mathcal{G}$, and conversely all trees in $\mathcal{G}$ are subtrees of trees in $\mathcal{F}$. We denote this by $\mathcal{F} \prec \mathcal{G}$

A simple observation which will be useful and serves as a sanity test is the following:

Observation 4.2. If a tree T is free from a family $\mathcal{G}$ and $\mathcal{F} \prec \mathcal{G}$ then T is also free from the family $\mathcal{F}$.

The algorithm will operate recursively, mostly using strictly stronger families. We will need to spend much energy to deal with the case where we will not be able to use strictly stronger families.

We can now explain in what manner we construct our stronger families. Let c be the color of v. Replace some of the trees in $\mathcal{F}$ having a root colored c with their subtrees rooted in the ith child of the root. This obviously creates a stronger family, and will be called *strengthening $\mathcal{F}$ to the ith children of the trees S*, where $S \subseteq \mathcal{F}[c]$. We denote it by $\mathcal{F}_i^S$. If $S = \mathcal{F}[c]$ then we just denote it by $\mathcal{F}_i^c$. Note that in this process we keep empty trees if they are generated in the new families, where such trees mean that any coloring of any tree will contain this family. We may also use one subtree common to several trees in the original family, so the number of trees may actually decrease in a strengthening. Note that a sequence of such strengthenings may only continue for $|V(\mathcal{F})|$ steps, as we eliminate at least one vertex each time. We will now need to perform this construction in a coordinated manner:

Definition 4.3.2. Let $\mathcal{F}$ be a family of colored trees with degree bound d. Let c be some color, and let $(S_1, S_2, \ldots, S_d)$ be an ordered partition of $\mathcal{F}[c]$. A d-tuple of stronger families $(\mathcal{F}_1^{S_1}, \mathcal{F}_2^{S_2}, \ldots, \mathcal{F}_d^{S_d})$ will be called a *c-useful* d-tuple. If at least one of the sets in the partition is empty, then we say that it is *null*, and otherwise we say that it is *non-null*.

The following lemma shows that such d-tuples are exactly the forbidden families we are interested in:

Lemma 4.3.3. *Given a tree T rooted in v, a coloring of T is in $\mathrm{P}_{\mathcal{F}}$ if and only if there exists a $c(v)$-useful d-tuple $(\mathcal{F}_1^{S_1}, \mathcal{F}_2^{S_2}, \ldots, \mathcal{F}_d^{S_d})$ such that for all $1 \le i \le d$ we have that T_{v_i} is in $\mathrm{P}_{\mathcal{F}_i^{S_i}}$.*

Proof. If the coloring is in $\mathrm{P}_{\mathcal{F}}$ then for every tree $t \in \mathcal{F}[c(v)]$ there exists some $1 \le i \le d$ such that T_{v_i} is in $\mathrm{P}_{\mathcal{F}_i^{\{t\}}}$, as otherwise either t would appear with v being its root, or another tree from $\mathcal{F}$ would appear in T_{v_i}. For every such tree t, assign it to a set S_i for an appropriate i (choosing arbitrarily if there are several possibilities). Since no tree from $\mathcal{F}$ appears in T, this is indeed a partition of $\mathcal{F}[c(v)]$, and a useful d-tuple exists as required.

Now if there exists a useful d-tuple as above, then no tree from $\mathcal{F}$ may appear in T — for all $1 \le i \le d$ we have that $\mathrm{P}_{\mathcal{F}_i^{S_i}}$ is either equal to or stronger than $\mathcal{F}$ and thus no tree from $\mathcal{F}$ appears in T_{v_i}, and finally for every tree $t \in \mathcal{F}[c(v)]$ there exists $1 \le i \le d$ such that $t \in S_i$, and therefore t_i cannot appear rooted in T_{v_i} and t cannot appear rooted in v. There is also no other possibility for a tree not contained in T_{v_i} for some $1 \le i \le d$, as such trees must be rooted in v.□

In our soundness proof we mainly use the following implication:

Lemma 4.3.4. *Given a tree T rooted in v, if there exists a $c(v)$-useful d-tuple $(\mathcal{F}_1^{S_1}, \mathcal{F}_2^{S_2}, \ldots, \mathcal{F}_d^{S_d})$ such that for all $1 \le i \le d$ we have that T_{v_i} is $\alpha_i\epsilon$-close to $\mathrm{P}_{\mathcal{F}_i^{S_i}}$, then the coloring of T is $\frac{\sum_{i=1}^{d} \alpha_i |V(T_{v_i})|}{|V(T)|}\epsilon$-close to $\mathrm{P}_{\mathcal{F}}$.*

Proof. For each $1 \leq i \leq d$, recolor T_{v_i} to be free from $\mathcal{F}_i^{S_i}$. This is possible by our assumption and recolors at most a $\frac{\sum_{i=1}^{d} \alpha_i |V(T_{v_i})|}{|V(T)|}\epsilon$-fraction of the vertices in the tree. Now we can appeal to Lemma 4.3.3 to see that the new coloring is free from $\mathcal{F}$. □

4.4 Testing for a family of forbidden topological subtrees in bounded degree trees

In this section we present and analyze an algorithm for testing whether a coloring of a given ordered tree of degree at most d is free from a family of forbidden topological subgraphs, in the sense of Definition 4.2.4, or is ϵ-far from any such coloring.

The algorithm works by trying to find evidence that the subtree it currently considers is far from being free from the forbidden family. It does so by finding "suspicious" vertices and considering all useful d-tuples of stronger families.

The main problem is in handling the case of null useful d-tuples. When not all colors appear in the roots of the forbidden family, that is, when $R \neq C$, an ϵ-far coloring also implies that there are many roots of forbidden trees, and therefore we can allow ourselves to assume that there is no need to recurse on the subtrees corresponding to the empty sets in the partition. This is since if there is any trouble there, then we are likely to sample it.

If all colors appear in the roots of the forbidden family, that is, $R = C$, then we cannot avoid recursion to all subtrees even for the null useful d-tuples. The key here is to note that even in a null useful d-tuple, at least one child will be tested with a strictly stronger family, and for this reason the distance for other children can be amplified. For notational simplicity, we will assume in the proof that the children of v are sorted in increasing order by the size of the subtree rooted in them, but it is easily extensible to trees where this is not the case.

One more important thing to note is that there may be cases where the forbidden family $\mathcal{F}$ is such that no coloring of T is free from it. We call such a family *unavoidable*. One important example of an unavoidable family is is a family containing an empty tree.

We divide the algorithm into three subroutines: MissingColor solves the case where $R \neq C$, AllColors solves the case where $R = C$, and GeneralTest glues them together. Parameters for all the tests are the input tree T, the forbidden family $\mathcal{F}$, the distance parameter ϵ and the required confidence parameter δ.

Algorithm 4.1 GeneralTest(T,$\mathcal{F}$,ϵ,δ)

```
  if $\mathcal{F}$ is unavoidable then reject $T$ and terminate
  if $\mathcal{F}$ is empty, $T$ is empty or $\epsilon \geq 1$ then accept $T$ and terminate
  if $R(\mathcal{F}) = C$ then call AllColors($T$,$\mathcal{F}$,$\epsilon$,$\delta$), answer as it did and terminate
  else call MissingColor($T$,$\mathcal{F}$,$\delta$,$\epsilon$), answer as it did and terminate
```

We first prove the correctness of GeneralTest assuming the correctness of the two subprocedures. We then provide the AllColors subprocedure and prove its correctness in Subsection 4.4.1. MissingColor and its proof of correctness is found in Subsection 4.4.2. We calculate the query complexity in Subsection 4.4.3. For the entire analysis, we assume that the tree is full (every vertex is either a leaf or has d children); in Subsection 4.4.4 we provide a simple reduction showing that this does not limit generality (we do not make this assumption for the trees in $\mathcal{F}$).

Lemma 4.4.1. *Procedure GeneralTest always accepts if T is in $\mathrm{P}_{\mathcal{F}}$, and rejects if it is ϵ-far from $\mathrm{P}_{\mathcal{F}}$ with probability at least $1-\delta$.*

Proof. If $\mathcal{F}$ is unavoidable, then T must contain it and therefore the coloring is not in $\mathrm{P}_{\mathcal{F}}$ and the algorithm correctly rejects. If $\mathcal{F}=\emptyset$ then any coloring is in $\mathrm{P}_{\mathcal{F}}$ and the algorithm correctly accepts. If T is an empty tree and $\mathcal{F}$ is not unavoidable, then T cannot contain any tree from $\mathcal{F}$ and is thus in $\mathrm{P}_{\mathcal{F}}$ and the algorithm correctly accepts. Finally if $\epsilon \geq 1$ and the family $\mathcal{F}$ is not unavoidable then any coloring is ϵ-close to $\mathrm{P}_{\mathcal{F}}$ and we can accept.

Procedure GeneralTest invokes procedure MissingColor if there is a color missing in the roots of the forbidden family, and by Lemma 4.4.17 this is correct. If all colors are present at the roots of the forbidden family, procedure GeneralTest invokes procedure AllColors, which is correct by Lemmas 4.4.11 and 4.4.12. Both will accept an ϵ-far tree with probability at most δ. □

4.4.1 AllColors

To simplify notation, in this analysis we will assume that the children of every vertex are ordered by increasing subtree size. That is, for every $v \in V(T)$, we have that $|V(T_{v_1})| \leq |V(T_{v_2})| \leq \ldots \leq |V(T_{v_d})|$. We explain how to generalize the algorithm and proof to trees not satisfying this requirement in Subsection 4.4.5, at the cost of clumsier notation.

The main idea will be to use the lighter subtrees of T_v to "fix" it. The problem is that for some vertices this will not work:

Definition 4.4.2. A vertex v is *structurally limiting* if in any coloring, T_{v_1} contains a tree from $\mathcal{F}_1^{\mathcal{F}[c(v)]}$. Recall that $\mathcal{F}[c(v)]$ is the subset of $\mathcal{F}$ containing only those trees with a root colored $c(v)$, and that $\mathcal{F}_1^{\mathcal{F}[c(v)]}$ is the strengthening of $\mathcal{F}$ to the first child of the trees in $\mathcal{F}[c(v)]$. That is, this is the family of trees created from $\mathcal{F}$ by replacing all of the trees in $\mathcal{F}[c(v)]$ with the subtree rooted in their first child. Also recall that we assume throughout that v_1 is the lightest child of v.

We will only need the "topmost" such vertices:

Definition 4.4.3. The set of *significant* structurally limiting vertices is the minimal set $L \subseteq V(T)$ such that for any structurally limiting vertex v, either $v \in L$ or there exists a $u \in L$ which is an ancestor of v.

Note that this is well-defined since T is a tree. We can now proceed to explain our probability distribution.

Definition 4.4.4. The *heaviest subtree* of T, denoted $H(T)$, is the subtree created by removing, for every $v \in V(T)$, the subtree T_{v_1} from the tree.

Definition 4.4.5. Let L be the set of significant structurally limiting vertices in the coloring of T. Define the probability vector $l : L \to [0,1]$ by $l(v) = \frac{|V(T_v)|}{\sum_{u\in L}|V(T_u)|}$.

Definition 4.4.6. Define the probability vector $p : H(T) \to [0,1]$ by $p(v) = \frac{|V(T_{v_1})|}{\sum_{u\in H(T)}|V(T_{u_1})|}$.

Now, to perform the "fixing" part of the proof, we will need useful d-tuples with a certain structure, that corresponds to not requiring null-recursions on the heaviest "relevant" child.

Definition 4.4.7. A $c(v)$-useful d-tuple $(\mathcal{F}_1^{S_1}, \dots, \mathcal{F}_d^{S_d})$ is *i-proper* for $v \in V(T)$ and $1 \le i \le d$ if $\bigcup_{j \le i} S_j = \mathcal{F}[c(v)]$, $S_i \ne \emptyset$ and for all $1 \le j \le i$, if $S_j \ne \emptyset$ then T_{v_j} is $\frac{\epsilon}{100d}$-close to $\mathrm{P}(\mathcal{F}_j^{S_j})$ and if $S_j = \emptyset$ then T_{v_j} is $\epsilon + \frac{\epsilon}{100d}$-close to $\mathrm{P}(\mathcal{F})$.

Note that Lemma 4.3.4 applies to the above definition. Now, these are enough if we are only interested in roots of forbidden trees:

Observation 4.4.8. For $v \in V(T)$, if there exists a $c(v)$-useful d-tuple $(\mathcal{F}_1^{S_1}, \dots, \mathcal{F}_d^{S_d})$ such that for every $1 \le i \le d$ where $S_i \ne \emptyset$ we have that T_{v_i} is free from $\mathcal{F}_i^{S_i}$, then in the current coloring of T, v cannot be the root of a tree from $\mathcal{F}$.

We use the term "bad" to define those vertices that make the algorithm reject:

Definition 4.4.9. A vertex $v \in V(H(T))$ is *bad* if for all $1 \le i \le d$, there exists no i-proper $c(v)$-useful d-tuple for T_v.

What follows is the core of the algorithm's correctness. It shows that if we have a statistically insignificant portion of bad vertices, both structurally limiting and others, then the coloring is close to being free from $\mathcal{F}$.

Lemma 4.4.10. *Let B be the set of bad vertices which are not structurally limiting, and BL be the set of bad vertices which are structurally limiting and significant. If $\sum_{v \in B} |V(T_{v_1})| \le \frac{\epsilon}{4d} |V(T)|$, $\sum_{v \in BL} |V(T_v)| \le \frac{\epsilon}{4d} \sum_{v \in BL} |V(T_v)|$, and $\mathrm{P}(\mathcal{F}) \ne \emptyset$, then the coloring of T is ϵ-close to $\mathrm{P}(\mathcal{F})$.*

Proof. Perform the following process recursively for a vertex v, starting at the root: If $v \notin (B \cup BL)$, then find the maximal i for which v has an i-proper d-tuple $(\mathcal{F}_1^{S_1}, \dots, \mathcal{F}_d^{S_d})$. For every $1 \le j \le i$, recolor T_{v_j} to be free from $\mathcal{F}_j^{S_j}$. For every $i < k \le d$, recursively repeat the whole process for v_k. Let us examine what fraction of the total tree is recolored, in addition to what is recolored by the recursive calls. Note that we recolor at most an $\frac{\epsilon}{100d}$ fraction of T_{v_i}, by the definition of an i-proper d-tuple. Also notice that $\left|\bigcup_{1 \le j < i} V(T_{v_j})\right| \le (1 - \frac{1}{d}) \left|\bigcup_{1 \le j \le i} V(T_{v_j})\right|$. Thus the total fraction of $T_{v_1}, \dots, T_{v_i}$ we have recolored is at most $\frac{\epsilon}{100d} + (1 - \frac{1}{d})(\epsilon + \frac{\epsilon}{100d}) < \frac{2d-1}{2d}\epsilon$.

For a vertex $v \in B$, by the fact that it is not structurally limiting, we can recolor its entire light subtree, T_{v_1}, to be free from $\mathcal{F}_1^{\mathcal{F}[c(v)]}$ and then recurse on the other children.

For a vertex $v \in BL$, just recolor T_v to be in $\mathrm{P}(\mathcal{F})$.

First note that we have recolored at most an ϵ fraction of the vertices. The total of all calls on non-bad vertices recolors at most a $\frac{2d-1}{2d}\epsilon$ fraction of the tree by the previous calculation, the calls on bad vertices recolor at most an $\epsilon/4d$ fraction of the tree, and the calls on structurally limiting vertices recolor at most an $\epsilon/4d$ fraction of the tree, both by the assumptions of the lemma.

To show that the new coloring is in $\mathrm{P}(\mathcal{F})$, we will demonstrate that no vertex in T can serve as the root of a forbidden tree. Suppose that a tree $t \in \mathcal{F}$ appears rooted in v. If $v \in B$, then t cannot appear rooted in v, since T_{v_1} is free from $\mathcal{F}_1^{\mathcal{F}[c(v)]}$ and thus the corresponding subtree of t cannot appear in T_{v_1}. If $v \notin (B \cup BL)$ and the process was invoked on it, let $(\mathcal{F}_1^{S_1}, \dots, \mathcal{F}_d^{S_d})$ be the i-proper d-tuple according to which its subtrees were recolored. Since $\bigcup_{j \le i} S_j = \mathcal{F}[c(v)]$, there exists some $j \le i$ such that $t \in S_j$, and therefore the subtree rooted in the jth child of the root of t cannot appear in T_{v_j}. Obviously it cannot be that $v \in BL$, since then T_v was recolored to be free from $\mathcal{F}$. The last remaining case are those $v \in V(T)$ that the recoloring procedure was not invoked on. Such a v must be in a subtree $\bar{T}$ of a u that the recoloring was invoked on, but the recoloring was not invoked on any vertex below u. This implies that the recoloring algorithm recolored all of $\bar{T}$ to be free of $\mathcal{F}$ or some family stronger than it, and so t cannot appear rooted in v. Therefore t cannot appear anywhere in T. □

Algorithm 4.2 AllColors(T,$\mathcal{F}$,ϵ,δ)

```
Sample $\frac{8d\log(\delta^{-1})}{\epsilon}$ vertices $A_B$ according to $p$ {Locating non-limiting bad vertices}
Sample $\frac{8d\log(\delta^{-1})}{\epsilon}$ vertices $A_L$ according to $l$ {Locating structurally limiting bad vertices}
for every $v \in A_B \cup A_L$ do
  Query $c[v]$
  for every $S \in (\mathcal{P}(\mathcal{F}[c(v)]) \setminus \{\emptyset\})$ and $1 \le i \le d$ do
    {Non-null-recursions}
    Call GeneralTest($T_{v_i}$,$\mathcal{F}_i^S$,$\frac{\epsilon}{100d}$, $\frac{\delta}{d\cdot 2^{|V(\mathcal{F})|+1}}$)
  for every $1 \le i < d$ do
    {Null-recursions to all children apart from the heaviest}
    Call GeneralTest($T_{v_i}$,$\mathcal{F}$,$\epsilon(1+\frac{1}{100d})$, $\frac{\delta}{d\cdot 2^{|V(\mathcal{F})|+1}}$)
  if there exists a proper $d$-tuple for $v$ then accept $v$,
  else reject $T$ and terminate
Accept $T$ {No $v$ was rejected}
```

We can now prove the correctness of AllColors by induction over the size of T. This could supposedly imply that the algorithm reads the entire coloring of the tree, but we will later analyze the use of stronger families and larger distance parameter to prove that this is not the case.

Lemma 4.4.11. *Assume that all recursive calls to GeneralTest with a tree T' such that: $|V(T')| < |V(T)|$, with any forbidden family $\mathcal{F}'$, distance parameter ϵ' and confidence parameter δ', accept colorings of T' which are in $\mathrm{P}(\mathcal{F})$ with probability 1. Then given a tree T, if the coloring is in $\mathrm{P}(\mathcal{F})$ then AllColors accepts with probability 1.*

Proof. Let us show that we accept every vertex v sampled. Since the coloring is in $\mathrm{P}(\mathcal{F})$, so is the coloring of T_v. Therefore, by Lemma 4.3.4 we know there exists a $c(v)$-useful d-tuple $(\mathcal{F}_1^{S_1}, \mathcal{F}_2^{S_2}, \dots, \mathcal{F}_d^{S_d})$ such that for all $1 \le i \le d$ we have that T_{v_i} is in $\mathrm{P}_{\mathcal{F}_i^{S_i}}$. By our assumption, all recursive calls corresponding to this d-tuple will accept with probability 1. Letting j be the largest index such that $S_j \ne \emptyset$, this is also a j-proper d-tuple, and thus AllColors will accept v with probability 1. Since this reasoning is true for all sampled vertices, AllColors will accept with probability 1. □

We now use Lemma 4.4.10 to prove the soundness of AllColors:

Lemma 4.4.12. *Assume that all recursive calls to GeneralTest with a tree T' such that: $|V(T')| < |V(T)|$, with any forbidden family $\mathcal{F}'$, distance parameter ϵ' and confidence parameter δ', reject all colorings ϵ'-far from $\mathrm{P}(\mathcal{F})$ with probability at least $1-\delta'$. Then given a tree T, a distance parameter ϵ and confidence parameter δ, if the coloring is ϵ-far from $\mathrm{P}(\mathcal{F})$ then AllColors rejects with probability at least $1-\delta$.*

Proof. Denote the set of bad significant structurally limiting vertices by BL, and the set of bad vertices which are not structurally limiting by B. By Lemma 4.4.10, if the coloring is ϵ-far from $\mathrm{P}(\mathcal{F})$, then either $\sum_{v \in B} |V(T_{v_1})| > \frac{\epsilon}{4d}|V(T)|$ or $\sum_{v \in BL} |V(T_v)| > \frac{\epsilon}{4d} \sum_{v \in BL} |V(T_v)|$.

In either case, we will sample a bad vertex v with probability at least $1 - \frac{\delta}{2}$. Since v is a bad vertex, it has no i-proper $c(v)$-useful d-tuple for any i, and therefore, for every candidate d-tuple there will be at least one recursive call that will reject with probability $1 - \delta'$, where δ' is the confidence parameter fed to that call. By the union bound, the probability that any call incorrectly accepts and causes us to falsely discover a proper d-tuple for v is at most $\delta/2$, and thus we will correctly reject with probability at least $1 - \delta$. □

4.4.2 MissingColor

We will start by explaining the kind of rejection witnesses we are looking for. We re-use the term "bad" to define those vertices that will make the algorithm reject.

Definition 4.4.13. Let v be a vertex colored $c \in R$. We say that v is *bad* if for any c-useful d-tuple $(\mathcal{F}_1^{S_1}, \mathcal{F}_2^{S_2}, \dots, \mathcal{F}_d^{S_d})$ there exists some $1 \le i \le d$ such that $S_i \ne \emptyset$ and T_{v_i} is $\epsilon/2$-far from $\mathrm{P}_{\mathcal{F}_i^{S_i}}$.

Note that a bad vertex can be easily "fixed" by changing its color to a member of $C \setminus R$. Now for the converse definition:

Definition 4.4.14. Suppose that v is a vertex such that $c(v) \in R$. Further suppose that we have a $c(v)$-useful d-tuple $(\mathcal{F}_1^{S_1}, \mathcal{F}_2^{S_2}, \dots, \mathcal{F}_d^{S_d})$ such that for all $1 \le i \le d$ either $S_i = \emptyset$ or the coloring of T_{v_i} is $\epsilon/2$-close to $\mathrm{P}_{\mathcal{F}_i^{S_i}}$. Denote the indices $1 \le i \le d$ corresponding to $S_i \ne \emptyset$ by I. The vertex v is called *I-good*.

In essence, we prove that if a tree is far from being free from the forbidden family, then there are many bad vertices.

Lemma 4.4.15. *If $R \ne C$ and there are at most $\frac{\epsilon}{2}|T|$ bad vertices in T, then T is ϵ-close to $\mathrm{P}_{\mathcal{F}}$.*

Proof. We show that by changing the color of at most an ϵ fraction of the vertices, no tree in $\mathcal{F}$ can appear in T. Let us consider the current coloring of T. Considering the root of any forbidden tree appearing in T, it can be either bad or I-good for some $I \subseteq \{1, \dots, d\}$.

Consider the following set U: for any I-good vertex v, add all of its children corresponding to indices from I to U. If a vertex is I-good for more then one set $I \ne \{1, \dots, d\}$, pick one arbitrarily. Now remove from U any vertex $v \in U$ which is a descendant of another vertex in U. Since these are vertices in a rooted tree, this is possible and uniquely determined. Note that for every vertex in U, the subtree rooted in it is $\epsilon/2$-close to being free from a corresponding family which is stronger than $\mathcal{F}$.

Define the new coloring: Every bad vertex which is not a descendant of a vertex in U is recolored to a color in $C \setminus R$, this recolors at most an $\epsilon/2$ fraction of the tree's vertices. For each vertex $v \in U$ recolor the subtree rooted in it to be free from the corresponding strengthening of $\mathcal{F}$. Since the subtrees rooted in the vertices of U are pairwise disjoint, we have recolored at most another $\epsilon/2$ fraction of the tree's vertices.

Suppose that a forbidden tree t appears in the new coloring. Its root must appear as either a bad vertex or an I-good vertex, as it obviously cannot be a descendant of a vertex from U.

The root of t cannot appear in an I-good vertex v, since there exists some $i \in I$ for which t is in S_i, and therefore T_{v_i} is free from $\mathcal{F}_i^{S_i}$. The root of t also cannot appear in a bad vertex, since we have recolored all bad vertices which are not descendants of U to a color in $C \setminus R$. Thus there can be no forbidden topological subtrees in the new coloring.

We have recolored at most an ϵ fraction of the vertices to get a coloring in $\mathrm{P}_{\mathcal{F}}$, and thus the original coloring is ϵ-close to $\mathrm{P}_{\mathcal{F}}$. □

In the following we will use the converse of the above:

Lemma 4.4.16. *If F is ϵ-far from $\mathrm{P}_{\mathcal{F}}$, then there are at least an $\epsilon/2$ fraction of bad vertices.*

We can now state the algorithm and prove its correctness.

Algorithm 4.3 MissingColor(T,$\mathcal{F}$,ϵ,δ)

```
Sample $10\epsilon^{-1}\log(\delta^{-1})$ vertices uniformly and independently
for every sampled vertex $v$ do
  Query $c[v]$
  if $\mathcal{F}[c(v)] = \emptyset$ then accept the vertex $v$
  for every $S \in (\mathcal{P}(\mathcal{F}[c(v)]) \setminus \{\emptyset\})$ and $1 \le i \le d$ do
    Call GeneralTest($T_{v_i}$,$\mathcal{F}_i^S$,$\epsilon/2$, $\frac{1}{2d}\frac{\delta}{2^{|V(\mathcal{F}|)}}$)
  if there exists a $c(v)$-useful $d$-tuple $(\mathcal{F}_1^{S_1}, \mathcal{F}_2^{S_2}, \ldots, \mathcal{F}_d^{S_d})$ such that for each $1 \le i \le d$ we
  either accepted a call on $T_{v_i}$ with $\mathcal{F}_i^{S_i}$ or $S_i = \emptyset$ then
    Accept the vertex $v$ {$v$ is an $I$-good vertex}
    Continue with the loop to the next vertex
  else reject $T$ and terminate {$v$ is a bad vertex}
if there was no rejection then accept $T$ and terminate
```

Lemma 4.4.17. *Assuming by induction that calls to GeneralTest on stronger families are correct (with 1-sided error whose probability is not more than the corresponding call parameter), MissingColor always accepts colorings in $\mathrm{P}_{\mathcal{F}}$, and rejects colorings which are ϵ-far from $\mathrm{P}_{\mathcal{F}}$ with probability at least $1 - \delta$.*

Proof. Note that if the coloring is in $\mathrm{P}_{\mathcal{F}}$ then there are no bad vertices and MissingColor will always accept. According to Lemma 4.4.16, if T is ϵ-far from $\mathrm{P}_{\mathcal{F}}$ then there exist at least $\epsilon/2 \cdot |T|$ bad vertices. Thus the probability of getting one in a sample is at least $\epsilon/2$, and the probability of not finding such a vertex in k samples is at most $(1 - \epsilon/2)^k < \exp(-k\epsilon/2)$, and with $k = 10\epsilon^{-1}\log(\delta^{-1})$ samples the probability of not sampling a bad vertex drops below $\delta/2$.

If a bad vertex v colored c is sampled, then for any useful d-tuple $(\mathcal{F}_1^{S_1}, \mathcal{F}_2^{S_2}, \ldots, \mathcal{F}_d^{S_d})$ there exists some $1 \le i \le d$ such that $S_i \neq \emptyset$ and T_{v_i} is $\epsilon/2$-far from $\mathrm{P}_{\mathcal{F}_i^{S_i}}$. By the induction hypothesis all calls over subtrees with stronger families (and parameter $\epsilon/2$) will reject with probabilities as required. To accept a false positive we need a useful d-tuple $(\mathcal{F}_1^{S_1}, \mathcal{F}_2^{S_2}, \ldots, \mathcal{F}_d^{S_d})$ for which we accepted for some $1 \le i \le d$ such that $S_i \neq \emptyset$. By a union bound the probability that this occurs is at most $\delta/2$.

Concluding the proof we have that the probability for either failing to sample a bad vertex or failing to detect that it is bad is at most δ. □

4.4.3 Query complexity

We will conclude the proof of Theorem 4.1 by calculating the query complexity of the algorithm. Note that it is independent of n and quasipolynomial in ϵ^{-1} for any fixed $\mathcal{F}$ and d.

Lemma 4.4.18. *Given a full degree d tree T, a forbidden family $\mathcal{F}$ such that $m = |V(\mathcal{F})|$, a distance parameter ϵ, confidence parameter δ and an oracle access to a coloring function for the vertices of the tree $c : V(T) \to C$, the number of queries performed by GeneralTest is at most $\epsilon^{-(md)^{O(1)}\log(\epsilon^{-1})\log\log(\delta^{-1})}$*

Proof. Denote the worst-case query complexities of GeneralTest, MissingColor and AllColors with distance parameter ϵ, confidence parameter δ, degree bound d and family $\mathcal{F}$ with $m = |V(\mathcal{F})|$ vertices in total by $G(\epsilon, \delta, d, m), M(\epsilon, \delta, d, m)$ and $A(\epsilon, \delta, d, m)$ respectively. Note that unless $\{\emptyset\} \in \mathcal{F}$, $|\mathcal{F}| \leq |V(\mathcal{F})|$, and thus we will regard both as bounded from above by m. By the definition of GeneralTest we have

$$G(\epsilon, \delta, d, m) \leq \max\{A(\epsilon, \delta, d, m), M(\epsilon, \delta, d, m)\}$$

First let us deal with the base case where $m = 0$ and $\delta > 0, \epsilon > 0$. In this case we either have an empty family and accept without performing any queries, or we have an empty tree and reject without performing any queries.

Now let us consider the case of $m > 0$ and $\delta > 0, \epsilon > 0$. Each call to MissingColor entails the query of $10\epsilon^{-1}\log(\delta^{-1})$ vertices. Each vertex sampled during the execution of MissingColor generates $d \cdot 2^m$ calls to GeneralTest with distance parameter $\epsilon/2$ and confidence parameter $\frac{1}{2d}\frac{\delta}{2^m}$. That is,

$$M(\epsilon, \delta, d, m) \leq 10\epsilon^{-1}\log(\delta^{-1}) + 10\epsilon^{-1}\log(\delta^{-1}) \cdot d2^m \cdot G\left(\epsilon/2, \frac{1}{2d}\frac{\delta}{2^m}, m-1\right)$$

To analyze the query complexity of AllColors, consider the recursion tree. That is, the directed tree where every vertex is an instantiation of the algorithm, and an edge (u, v) exists when u invoked v. The *null-recursion tree* Y of $\mathcal{F}$ is the subtree of the recursion tree comprised of null recursive calls with $\mathcal{F}$. Note that all calls that go from Y to a vertex outside of it are with a smaller value of m.

Since every recursive call within the null-recursion tree multiplies the distance parameter by $\left(1 + \frac{1}{100d}\right)$, its depth is bounded by $100d \cdot \log(\epsilon^{-1})$. The minimal distance parameter used in the null-recursion tree is ϵ, and the minimal confidence parameter is $\delta \cdot (2d)^{-100d\cdot\log(\epsilon^{-1})} = \delta\epsilon^{100d\log d}$, and therefore the function call for every vertex in the null-recursion tree performs at most $\frac{16d\log(\delta\epsilon^{100d\log d^{-1}})}{\epsilon} \leq 2^{11}d^3\epsilon^{-2}\log(\delta^{-1})$ queries. AllColors performs at most $(d-1)$ null-recursive calls for every vertex sampled, and thus every vertex in the null-recursion tree has at most $2^{11}d^4\epsilon^{-2}\log(\delta^{-1})$ children. Therefore, there are at most $\left(2^{11}d^4\epsilon^{-2}\log(\delta^{-1})\right)^{100d\cdot\log(\epsilon^{-1})}$ instantiations in the null-recursion tree. Additionally, every such instantiation performs at most $d2^m$ calls to GeneralTest with distance parameter at least $\frac{\epsilon}{100d}$, confidence parameter at least $\frac{\delta\epsilon^{100d\log d}}{d\cdot 2^{|V(\mathcal{F})|+1}}$ and family with at most $m-1$ vertices, for very vertex queried.

Thus we can bound the number of queries used in the entire null-recursion tree by:

$$\left(2^{11}d^4\epsilon^{-2}\log(\delta^{-1})\right)^{100d\cdot\log(\epsilon^{-1})+1} + \left(2^{11}d^4\epsilon^{-2}\log(\delta^{-1})\right)^{100d\cdot\log(\epsilon^{-1})} \cdot d2^m G\left(\frac{\epsilon}{100d}, \delta\epsilon^{100d\log d}, m-1\right)$$

And thus we can bound $G(\epsilon, \delta, d, m) \le \max\{A(\epsilon, \delta, m), M(\epsilon, \delta, m)\}$:

$$G(\epsilon, \delta, d, m) \le \left(2^{11} d^4 \epsilon^{-2} \log(\delta^{-1})\right)^{100d \cdot \log(\epsilon^{-1})+1} \left(1 + d2^m \cdot G\left(\frac{\epsilon}{100d}, \delta\epsilon^{100d\log d}, m-1\right)\right)$$

Now unfold the recurrence relation to get

$$\begin{aligned} G(\epsilon, \delta, d, m) &\le m\left(2^{11} d^4 \epsilon^{-2} \log(\delta^{-1}\epsilon^{-100md\log d})\right)^{101md \cdot \log(\epsilon^{-1})} \\ &\le m\left(2^{18} m d^5 \epsilon^{-3} \log d \log(\delta^{-1})\right)^{101md\cdot\log(\epsilon^{-1})} \\ &= \epsilon^{-(md)^{O(1)} \log(\epsilon^{-1}) \log\log(\delta^{-1})} \end{aligned}$$

We note here that the dependency on δ can always be made logarithmic by first running the algorithm with $1/2$ as the confidence parameter, and then repeating the whole test $\log(\delta^{-1})$ independent times.

4.4.4 Reduction from a general tree to a full tree

In the previous subsections we assumed that in the d degree tree T all vertices have an out-degree of d or 0. In this section we will describe how to reduce the case of a general degree d tree T (that is, where some vertices might have degree $1, \dots, d-1$) to the case of a full tree.

We define a new color, c_l, which we think of as the color given to the extra leaves. For any vertex in T with an out-degree smaller than d we add child vertices colored c_t for all of the missing children. For the leaves in T we also add d child vertices colored c_t. For any d colors $c_1, \dots, c_d \in C \cup \{c_t\}$ (this is an ordered choice with repetition), we add a tree to $\mathcal{F}$ which has only a root colored c_t with d children where the ith of which is colored c_i for all $1 \le i \le d$.

Claim 4.4.19. *Given the colored tree T and the family $\mathcal{F}$, denote the new colored tree by T_b and the new family by $\mathcal{F}_b$. The following holds:*

- *If a coloring of T is free from $\mathcal{F}$ then the coloring of T_b described above is free from $\mathcal{F}_b$ and vice versa, that is $T \in \mathrm{P}_{\mathcal{F}} \iff T_b \in \mathrm{P}_{\mathcal{F}_b}$.*
- *The distance of the colored tree T_b from being free of $\mathcal{F}_b$ is at least $\frac{1}{d+1}$ the distance of T from being free of $\mathcal{F}$, that is $\frac{1}{d+1} d(T, \mathrm{P}_{\mathcal{F}}) \le d(T_b, \mathrm{P}_{\mathcal{F}_b})$.*

Proof. For the first item suppose first that $T \in \mathrm{P}_{\mathcal{F}}$, and assume that there exists a tree $T_f \in \mathcal{F}_b$ that appears in T_b. By its construction, all the vertices colored c_t are in its leaves, and thus $T_f \in \mathcal{F}$, and T contains this tree from $\mathcal{F}$, as T_f does not have a vertex colored c_t, a contradiction. The other direction of this item is a consequence of the second item, proved next.

For the second item first note that the tree $T' \in \mathrm{P}_{\mathcal{F}_b}$ which obtains the distance of T_b to $\mathrm{P}_{\mathcal{F}_b}$, that is, the tree that has $d(T_b, T') = d(T_b, \mathrm{P}_{\mathcal{F}_b})$, is such that all of the vertices colored c_t in T' are leaves, and if we remove them all we obtain a tree in $\mathrm{P}_{\mathcal{F}}$. Supposing this was not the case, if T' had vertices colored c_t which were not leaves, then it would not be in $\mathrm{P}_{\mathcal{F}_b}$ since it would contain one of the new trees introduced to $\mathcal{F}_b$. On the other hand, it is easy to see that all the leaves in T_b, which were colored c_t, retain their color in T' (since with this color they can never be in any $T_f \in \mathcal{F}_b$). If after removing all of the vertices colored c_t from T' we do not obtain a tree in $\mathrm{P}_{\mathcal{F}}$ then there must be some tree $T_f \in \mathcal{F}$ which appears in T', and thus its corresponding tree in $\mathcal{F}_b$ appears in T', in contradiction to the assumption that $T' \in \mathrm{P}_{\mathcal{F}_b}$. Therefore, if we recolor a $d(T_b, \mathrm{P}_{\mathcal{F}_b})$ fraction of the vertices in T_b to be free from $\mathcal{F}_b$, and then remove the vertices colored c_t, we will get a coloring of T which is free from $\mathcal{F}$. Since T_b may have at most as many as $d+1$ times the vertices of T, we get the second item. $\square$

4.4.5 Extending AllColors to trees with unsorted children

Recall that in the analysis and statement of the AllColors algorithm, we assumed that for every vertex $v \in V(T)$, its children are sorted by increasing subtree size, that is that we have $|V(T_{v_1})| \leq |V(T_{v_2})| \leq \ldots \leq |V(T_{v_d})|$. This assumption helps clarify the discussion, as it helps to make for an easier definition and use of i-proper useful d-tuples. In this section we will explain how to avoid this assumption. Note that we are here under the assumption that the tree is full, see Subsection 4.4.4 above for proof that this assumption does not limit generality.

First, we will need to define a "sorting function" for the children of each vertex:

Definition 4.4.20. Let $v \in V(T)$ be a vertex, and fix some sorting S of its children by non-decreasing subtree size. The sorting function of v, $\sigma_v : [d] \to [d]$ is the function mapping the positions in the sorting S to the original children of v. That is, $\sigma_v(i) = j$ means that the i'th smallest child of v is v_j.

Note that σ_v truly depends on v as well as T. A simple example of how this affects definitions is the notion of a structurally limiting vertex.

Definition 4.4.21. A vertex $v \in V(T)$ is *structurally limiting* if any coloring of $T_{v_{\sigma_v(1)}}$ contains a tree from $\mathcal{F}_{\sigma_v(1)}^{\mathcal{F}[c(v)]}$.

The definition of the heaviest subtree changes in a similar manner.

Definition 4.4.22. The *heaviest subtree* of T, denoted $H(T)$, is the subtree created by removing, for every $v \in V(T)$, the subtree $T_{v_{\sigma_v(1)}}$ from the tree.

The main change in the proof of the testability of the coloring property is the following new definition of an i-proper d-tuple, and the corresponding changes wherever it is used.

Definition 4.4.23. A $c(v)$-useful d-tuple $(\mathcal{F}_1^{S_1}, \ldots, \mathcal{F}_d^{S_d})$ for $v \in V(T)$ is *i-proper* if it holds that $\bigcup_{j \leq i} S_{\sigma_v(j)} = \mathcal{F}[c(v)]$, $S_{\sigma_v(i)} \neq \emptyset$ and for all j such that $1 \leq j \leq i$, if $S_{\sigma_v(j)} \neq \emptyset$ then $T_{v_{\sigma_v(j)}}$ is $\frac{\epsilon}{100d}$-close to $\mathrm{P}(\mathcal{F}_{\sigma_v(j)}^{S_{\sigma_v(j)}})$ and if $S_{\sigma_v(j)} = \emptyset$ then $T_{v_{\sigma_v(j)}}$ is $\epsilon + \frac{\epsilon}{100d}$-close to $\mathrm{P}(\mathcal{F})$.

The algorithms, definitions and proofs now follow those of subsections 4.4.1, 4.4.2, 4.4.3 and 4.4.4 almost verbatim, using the new definitions and notation.

4.5 A lower bound for testing against a family of forbidden induced subtrees in a balanced binary tree

We sketch here an adaptation of the lower bound from Section 3.7 for testing for a family of forbidden subgraphs in a balanced binary tree. The proof uses Yao's principle, which is the standard tool in this setting. Let us first state the lower bound formally:

Definition 4.5.1. Let $\mathcal{F}$ be a set of colored full balanced binary trees, and let T be a fixed full balanced binary tree. We denote by $\mathrm{P}_{\mathcal{F}}^{sub}$ the set of all colorings of T which do not contain any of the colored trees from $\mathcal{F}$ as an induced subgraph.

Theorem 4.3. *There exists a family of* 3*-colored trees* $\mathcal{F}$ *such that any nonadaptive algorithm* A*, which accepts all colorings in* $\mathrm{P}_{\mathcal{F}}^{sub}$ *and rejects all colorings which are* $1/5$*-far from it with probability at least* $2/3$*, must perform at least* $\Omega(h)$ *non-adaptive queries, or at least* $\Omega(\log(h))$ *adaptive queries, where* h *is the height of* T*.*

Proof sketch. To present the property it is convenient to use the set of colors $\{1, 2, X\}$. The set of forbidden subtrees includes all trees with one root and two children, *apart* from the following: The three possibilities where the two children have the same color as the root, and the two possibilities where the root is colored "X" and the children are colored one with "1" and one with "2".

We present two input distributions D_{good} and D_{bad}, such that the distribution D_{good} is supported on inputs which are in $\mathrm{P}_{\mathcal{F}}^{sub}$ and D_{bad} is supported on inputs which are $1/5$-far from $\mathrm{P}_{\mathcal{F}}^{sub}$. We then prove that for any set Q of at most $q = \alpha \cdot h$ vertices of T, for some fixed α, the variation distance between D_{good} and D_{bad} when projected on Q is at most $1/3$. The fact that the two distributions are indeed supported on the correct sets and the proof that projections of them on small subsets have small variation distance is almost identical to that of Section 3.7.

Both distributions are constructed by first choosing, randomly and uniformly, a level l in the balanced tree, except for the two bottom levels (that is, we choose a number l between 1 and $h-2$). In both distributions all vertices at or above Level l are colored X. In D_{good}, at level $l+1$ each pair of siblings is uniformly and independently colored with either 1 and 2 or 2 and 1 (in that order). In D_{bad}, at level $l+1$ one vertex of each pair of siblings is colored with X and the other vertex with 1 or 2 (we uniformly and independently choose among the four options), while at level $l+2$ each pair of siblings whose father was colored with X is uniformly and independently colored with either 1 and 2 or 2 and 1. In both distributions, every vertex not already colored by the above is colored by 1 or 2 according to the color of its lowest colored ancestor.

The main thing to note about the family $\mathcal{F}$ is that colorings in it can be characterized as follows: Pick some vertex v at level $k \leq h$. Then it must be that the number of 1s appearing in the leaves descendant from v must be either $0, 2^k$ or 2^{k-1}, and none of them may be equal to X unless all of them are. That is, every "binary search interval" is either all 1, all 2, exactly one half of each, or all X.

Note that in D_{bad}, the leaves will contain disjoint intervals far from following the above characterization, and therefore the coloring will be far from $\mathrm{P}_{\mathcal{F}}^{sub}$.

We refer to Section 3.7 for the full proof that the two distributions are indistinguishable with a small number of queries. The general idea is that if none of the queries or their lowest common ancestors is in Level l or $l+1$, then the two distributions look the same. While in Section 3.7 only leaf queries are allowed, this does not significantly change the proof. □

Note that one can construct an algorithm for distinguishing between the two distributions in the proof sketch with $O(\log(h))$ queries, by picking a root to leaf path in the tree and then using binary search to find the level l and check the computation there. In fact, a similar procedure (over a random sample of leafs) gives an $O(\log(h))$ adaptive test or an $O(h)$ non-adaptive test (for a any fixed ϵ) for the property itself, over full balanced binary trees of height h.

Chapter 5

Partial Property Testing

5.1 Introduction

When proving that testing a property requires many queries, one might ask "how strong is this requirement?", which can be illustrated with an example. Alon et. al. [AKNS00] studied the testability of formal languages, and proved that the language $L = \{uu^Rvv^R | u, v \in \{0,1\}^*\}$ requires at least $\Omega(\sqrt{n})$ queries to test (formally, the property $L \cap \{0,1\}^n$ requires that many queries to test). Informally, one may say that the "reason" for this language being untestable is the difficulty in guessing the length of uu^R. This can be made formal by considering the languages $L_i = \{uu^Rvv^R | u, v \in \{0,1\}^*, |u| = i\}$, which form a partition of L. A simple sampling algorithm can perform $O(\epsilon^{-1})$ queries to an input and distinguish between inputs in L_i and inputs ϵ-far from L. It is also important to note that $|L \cap \{0,1\}^n| = 2^{\Theta(n)}$, but its partition $L_0 \cap \{0,1\}^n, \ldots, L_n \cap \{0,1\}^n$ is only to a number of subsets linear in n.

This phenomenon is not unique to the language considered by Alon et. al. Another example is that of graph isomorphism, first considered in the property testing framework by Alon et. al. [AFKS00] (and later by Fischer and Matsliah [FM08]), and shown to require at least $\Omega(n)$ queries to test. In this setting we consider a pair of unknown graphs given by their adjacency matrices, and we are charged with distinguishing the case where they are isomorphic from the case where more than ϵn^2 of their edges must be changed to make them isomorphic. In this case, the size of the property is $2^{\Theta(n^2)}$, and we can partition the property into $n!$ properties $\{P_\pi | \pi \in S_n\}$, each defined by $P_\pi = \{(G_1, G_2) | \pi(G_1) = G_2\}$, such that a sampling algorithm can perform $O(\epsilon^{-1})$ queries to an input and distinguish between inputs in P_π and inputs ϵ-far from the original property.

Thus it is tempting to ask whether this is a general phenomenon. Can any property P be partitioned into $k = |P|^{o(1)}$ properties $P_1, \ldots, P_k$ such that the task of distinguishing inputs in P_i from inputs far from P can be performed with a number of queries that depends only on ϵ?

This question has a strong connection, in fact a near-equivalence, with the notion of a MAP as defined by Gur and Rothblum [GR15]. They define a MAP (Merlin-Arthur proof of Proximity) as a testing algorithm that first reads a "proof string" in whole, and uses it to test the given input. The requirement is that an input in P will have some corresponding proof that causes high probability acceptance, while for ϵ-far inputs for every proof there will be a high probability of rejection. The connection to our framework is that the proof corresponds to the representation of the alleged i such that the input is in P_i, making the required proof length equal to $\lceil \log k \rceil$ for the optimal k.

The first main result of this chapter is a proof that an efficient decomposition does not always exist. In fact, there exist properties for which any such partition must be to a number of subsets exponential in n (and equivalently they do not admit a MAP with an $o(n)$ proof size for testing with a number of queries independent of n).

To prove this result we in fact show the non-existence of a strictly weaker testing scenario, that would correspond to being able to test just for the biggest P_i in the alleged partition.

Definition 5.1.1 (Partially testable property)**.** For $P \subseteq \{0,1\}^n$ and $P' \subseteq P$, we say that P is *P'-partially testable with q queries* if there exists an algorithm A that gets as input a parameter $\epsilon > 0$ and query access to an input string $x \in \{0,1\}^n$ and outputs *accept* or *reject* such that:

- If $x \in P'$, then A accepts with probability at least $2/3$.
- If $d(x,P) > \epsilon$, then A rejects with probability at least $2/3$.

If furthermore all queries performed to the input can be decided before any of them are made, then the algorithm is *non-adaptive*, and else it is *adaptive*.

Obviously, if P is testable with q queries, then for any subset $P' \subseteq P$ it is P'-partially testable with the same number of queries. On the other extreme, for any property P and any element $x \in P$, we have that P is $\{x\}$-partially testable with $O(\epsilon^{-1})$ queries.

The partitions described above are in fact partitions of P into subsets $P_1, \dots, P_k$ such that P is P_i-partially testable for every $1 \le i \le k$. If there exists such a partition into not too many sets, then there must be at least one set that is relatively large. Our main result shows that there exists a property P for which all subsets $P' \subseteq P$ such that P is P'-partially testable are small. In fact, all linear codes with large dual distance define such properties.

Theorem 5.1. *Let $C \subseteq \{0,1\}^n$ be a linear code of size $|C| \le 2^{\frac{1}{64}n}$ and dual distance Γ. For every $C' \subseteq C$, if C is C'-partially testable with q adaptive queries, then $|C'| \le |C| 2^{-\Theta(\Gamma/q)}$.*

We will first prove, as a warm-up, a weak version of Theorem 5.1 in Section 5.4 which will apply for q non-adaptive queries and imply the bound $|C'| \le |C| 2^{-\Theta(\Gamma/q^3)}$. This proof will use some of the key ideas that will later manifest in the proof of the theorem in its full generality in Section 5.5.

Remark. Theorem 5.1 holds for every property P which is Γ-wise independent. The only use of the linearity of C is in that dual distance Γ implies Γ-wise independence (see Theorem 5.3.8).

An important question is the existence of codes with strong parameters. A random linear code C will have $\Gamma = \Theta(n)$ and $|C| = 2^{\Theta(n)}$ with high probability (this is implied by the Gilbert-Varshamov bound [Gil52, Var57]; MacWilliams et. al. [MST72] showed that this can also be obtained by codes which are self-dual and thus also have good distance), and thus by Theorem 5.1 we will have that for any $C' \subseteq C$ such that C is C'-partially testable with q queries,

$|C'| \leq |C|2^{-\Theta(n/q)}$. For a constant q, this implies that partial testability will only be possible with exponentially small subsets. The best explicit (and reasonable uniform decision complexity) construction known to us is that of [ABI86], which gives $|C| = 2^{\Theta(n)}$ with $\Gamma = \Theta(n/\log n)$, and thus the bound deteriorates to $|C'| \leq n^{O(1)}|C|2^{-\Theta(n/q)}$, which is polynomially worse than the non-explicit bound, but is still a strong upper bound on the size of C'.

Theorem 5.1 implies that there exist properties P that require a lot of queries to test, and that every partition of P into subsets $P_1, \dots, P_k$ such that P is P_i-partially testable for every $1 \leq i \leq k$ requires that k will be very big. One might ask if we can prove a converse. That is, if P can be tested with a few queries, can we find such a partition with a small k?

This problem can also be phrased as whether there exists a general trade-off between testing hardness and partitionability to easily partially testable properties. For the converse direction, of whether partitionability implies an efficient test for the whole property, we present results that revolve around the stricter notion of proximity oblivious testing:

Definition 5.1.2. A *non-adaptive,* 1*-sided proximity-oblivious* q*-test* for a property P with *detection function* $\rho(\epsilon)$ is an algorithm that makes q non-adaptive queries to the input (i.e. the queries are all made before the answers to them are received), and based on those answers accepts or rejects the input in a way that satisfies the following:

- If the input satisfies P then the algorithm accepts with probability 1.
- If the input is ϵ-far from P, then the algorithm rejects with probability at least $\rho(\epsilon)$.

Note that the algorithm is given the input length n in advance, but is not given ϵ. A partial proximity-oblivious q-test is defined in the analogous manner.

The simplest conceivable proximity-oblivious test would be a 2-test, making only 2 queries. Such tests exist for example in some monotonicity testing scenarios. We prove that partitionability into properties that are 2-testable implies a sublinear query test (that is not proximity-oblivious) for the entire property.

Theorem 5.2. *Let $P_1, P_2, \dots, P_k \subseteq \{0,1\}^n$ be properties such that for every $i \in \{1, \dots, k\}$, P_i has a* 1*-sided error proximity-oblivious* 2*-tester with detection function $\rho(\epsilon)$. If $\epsilon > 0$ is such that $\rho(\epsilon/2) > 0$, then for n large enough, as a polynomial function of $1/\rho(\epsilon/2)$, there is a one-sided error non-adaptive ϵ-tester for $P = \bigcup_{i=1}^k P_i$ with query complexity $\tilde{O}(n^{2/3}\epsilon^{-1}) \cdot \log(k)$. This also holds if for every P_i we only require a* 1*-sided error proximity-oblivious P_i-partial* 2*-test for P.*

The converse of the above immediately implies an observation interesting enough to state on its own.

Corollary 5.3. *If a property P requires $\Omega(n^\beta)$ many queries for some fixed $\beta > 2/3$, then there is no way to partition P into polynomially many properties (even not necessarily disjoint) admitting* 1*-sided proximity-oblivious* 2*-tests (or even the corresponding partial tests).*

Theorem 5.2 is proved using a special test that we call a *universal test*, that works by selecting every index i for querying with probability $\tilde{O}(n^{-1/3}\epsilon^{-1}) \cdot \log(k)$, independently of other indexes. We prove in Theorem 5.10 below that such a kind of test will work for any property admitting a proximity oblivious 2-test, regardless of how that 2-test works. This universal test is very close to what is defined as a *sampling based test* in an independent work [GR13] of Goldreich and Ron. In particular, our proof yields the following corollary, which partially addresses a question from [GR13] about whether proximity oblivious tests are translatable to sample-based ones:

Corollary 5.4. *Let P be a property that has a 1-sided error proximity-oblivious 2-tester with detection function $\rho(\epsilon)$. If $\epsilon > 0$ is such that $\rho(\epsilon/2) > 0$, then for n large enough, as a polynomial function of $1/\rho(\epsilon/2)$, there is a 1-sided error sample based test (see [GR13], Definition 2.3) with query complexity $\tilde{O}(n^{2/3}\epsilon^{-1}) \cdot \log(k)$.*

For proximity oblivious q-tests with $q > 2$ the situation is more complex, and we can only prove an analog of Theorem 5.10 (and by it Theorem 5.2) where the power of n in the query complexity depends (rather badly) on both q and $\rho(\epsilon/2)$.

To formulate the theorem achieving this, we say that a set R of indexes is a *witness against the input* for a property P, if the restriction of the input to R is such that it cannot be the restriction of any member of P (or alternatively, this restriction cannot be extended to an alternate input that satisfies P).

Definition 5.1.3. For $\gamma \in (0, 1)$, the *γ-universal sampler* selects a set $R \subseteq [n]$ where, for every $i \in [n]$, $\Pr[i \in R] = n^{-\gamma}$.

We prove that the above sampling technique, essentially that of a sample-based tester as in [GR13], is indeed a core of a "universal test" for any property that has a (possibly "unknown") 1-sided proximity-oblivious q-test.

Theorem 5.5. *For every property P with a proximity oblivious q-test with detection function $\rho(\epsilon)$ there exists γ depending on q and $\rho(\epsilon/2)$ (for every ϵ), so that for n large enough and every ϵ-far input over $\{0,1\}^n$, the γ-universal sampler finds a witness against it with probability $1 - o(1)$.*

Its immediate corollary (through standard probability amplification and union bound) gives us a sub-linear query complexity test for any property decomposable into not too many (at most $\exp(n^{o(1)})$) properties where each of them has a proximity oblivious test, as long as they have the same detection function $\rho(\epsilon)$.

Corollary 5.6. *If $P = \bigcup_{i=1}^{\ell} P_i$ is a property such that every P_i has an oblivious 1-sided error (proximity oblivious) q-test, all with the same detection function $\rho(\epsilon)$ (but not necessarily the same test), then for n large enough the following is a test for P with $O(\log(\ell)n^{1-\gamma})$ query complexity, where we use the γ of Theorem 5.5:*

Select a set $R \subseteq [n]$ that is the union of $2\log(\ell)$ sets, each chosen according to the γ-universal sampler. If $|R| > 4\log(\ell)n^{1-\gamma}$ then accept immediately, and otherwise query the input on all indexes of R, reject if R is a P_i-witness against the input for every $i \in [\ell]$, and accept otherwise.

Finally, we prove a result in the other direction, hinting that maybe some role for proximity oblivious testing is essential. Using a very simple construction we prove the following:

Theorem 5.7. *For every fixed k there is a property P, so that $1/5k$-testing P (even adaptively) requires $\Omega(n^{1-1/k})$ queries, while P is still decomposable to at most n^{k-1} many properties so that each of them is even ϵ-testable in itself with $O(1/\epsilon)$ many queries for every ϵ; in fact each of them will have a proximity-oblivious 1-sided k-test with the detection function $\rho(\epsilon) = O(k\epsilon)$.*

Until now we discussed the relation of Theorem 5.1 to the impossibility of decomposing a property to testable ones. However, there may be use in its stronger statement of not having even one large sub-property for which there exists an efficient test. The proof of Theorem 5.1 immediately gives the following corollary.

Corollary 5.8. *Suppose that P is a property for which $|P| \leq 2^{\frac{1}{64}n}$, and C is any linear code with dual distance Γ so that $|P \cap C| \geq |C|2^{-\Gamma/q}$. Then P requires at least $\Theta(q)$ many queries to test (or even $P \cap C$-partially test).*

5.1.1 Related work

The notion of partial testability, while not defined before, is implicit in previous works on PCPs (Probabilistically Checkable Proofs). The long code tester of Håstad [Hås01] accepts inputs which are codewords in the long code, and rejects inputs which are far from being k-juntas for some k. Since codewords in the long code are dictatorships (1-juntas), this is an instance where the fact that being such a k-junta is dictatorship-partially testable is used to construct PCPs.

Our notion of a partition is similar to existing notions in computational complexity. For a partition $P = P_1 \cup P_2 \cup \ldots \cup P_k$ where for every $1 \leq i \leq k$, P is P_i-partially testable, the designation of P_i can be seen as a "proof" that a certain x is in P. If $x \in P$, then there exists some P_i such that $x \in P_i$ and therefore a P_i-partial tester for P will accept it with high probability. If x is ϵ-far from P, then all P_i-partial testers for P will reject it with high probability.

This is similar to the notion of a *Probabilistically Checkable Proof of Proximity (PCPP)*, first introduced by Ben-Sasson et. al. [BSGH$^+$06] (a precursor to this is found in [Sze99]). PCPPs are to property testing as NP is to P. A q-query PCPP for a property $P \subset U$ is an algorithm that gets as input $x \in U$ and a *proof of proximity* $\pi \in \{0,1\}^l$. The algorithm must perform at most q queries to x and π and fulfill the requirement that if $x \in P$ then there exists a proof π that causes the algorithm to accept with high probability, but when x is ϵ-far from P then for any proof π the algorithm rejects with high probability. In our setting, the algorithm is allowed free access to a proof of length $l = \log(k)$, but we expect l to be sublinear in the size of x.

Rothblum et. al. [RVW13] introduced the notion of an *Interactive Proof of Proximity (IPP)*. In an IPP for a property P, the tester can also communicate with a *prover* in addition to querying the input x. If $x \in P$ then the prover has a strategy that will cause the tester to accept with high probability. When x is ϵ-far from P, the prover cannot make the tester accept with high probability. Rothblum et. al. show that all languages in NC admit such a protocol with $\sqrt{n}$ query and communication complexity and polylog(n) communication rounds. Protocols of this kind are only interesting for the case where the communication complexity is sublinear, or else the prover may just give the input to the tester.

Independently of the research performed towards this thesis, Gur and Rothblum [GR15] weakened the IPP model to create *Merlin-Arthur Proofs of Proximity (MAP).* Gur and Rothblum define a MAP as a proof-system for a property P where for an input x and a proof π the verifier reads the entire proof π and queries q bits from the input x. If $x \in P$, then there exists a proof π such that the verifier accepts with high probability, and if x is far from P, then for every proof π the verifier rejects with high probability. Since we can trivially set $\pi = x$, the only interesting cases are where the length of π is sublinear.

The notion of a MAP with q queries and proofs of length ℓ for a property P is equivalent to the existence of $k = 2^\ell$ sets $P_1, \dots, P_k$ such that $P = P_1 \cup P_2 \cup \dots \cup P_k$ where for every $1 \leq i \leq k$, P is P_i-partially testable with q queries.

Gur and Rothblum give several constructions of properties where a MAP with a sublinear length proof greatly reduces query complexity. Gur and Rothblum also introduce the *Tensor Sum* family of properties, and prove that for every constant $\alpha > 0$ there exists an instantiation of Tensor Sum such that any MAP for it that performs q queries must require a proof of length $\Omega\left(\frac{n^{1-\alpha}}{q}\right)$. This bound is slightly weaker than the implication for decomposability of Theorem 5.1 proved in this chapter for our property (however, their property is not a high dual-distance code, so our result would not apply directly). There is no known bound on the size of a sub-property of the Tensor Sum properties admitting a partial test, only on decomposability.

Their lower bound is proved by an extension of the communication complexity technique of Brody et. al. [BBM12] to *Merlin-Arthur communication complexity.* First proving a lower bound for 1-sided testing this way, they then use a general conversion technique (at some cost to both proof length and query complexity, see below) to 2-sided testing. Gur and Rothblum also prove that this trade-off is almost optimal for the Tensor Sum properties.

Additionally, Gur and Rothblum show separations between the power of MAPs and that of IPPs and PCPPs. For their proofs they also show that 2-sidedness may only give a MAP a polylog(n) factor improvement in proof length and query complexity over a 1-sided algorithm. Their result implies a connection also between 1-sided and 2-sided partial testability, although not one that would preserve O(1)-query partial testability.

Regarding the testing versus proof length trade-off question, they show it for the very simple case of "proof-oblivious" testers, i.e. algorithms that make their queries before reading the alleged proof. By contrast, the main difficulty in proving our preliminary trade-off result is exactly that the tests for different P_i could have differing query distributions (even that each of them in itself is proximity oblivious).

Another angle to our methods related to the above trade-off comes from the recent work of Goldreich and Ron [GR13]. Their work is centered on what they call *sample-based algorithms*, which are testing algorithms that select all their queries uniformly and independently at random. For a number of queries that is a fixed power of n where n is large enough, this is virtually identical to the way our universal tests work, where every index is independently chosen to be queried with some fixed probability. Indeed they raised the question of whether any property that is testable by a proximity-oblivious q-test can also be tested by a sublinear complexity sample-based test, for which we give a partial positive answer for 1-sided error tests. Golreich and Shinkar [GS14] also define 2-sided error proximity oblivious q-tests, which we do not analyze here.

5.2 General themes

For the proofs of our main result we develop new techniques that are in some ways more flexible than the traditional use of Yao's method for proving property testing lower bounds. We believe that these techniques hold promise for other lower bound situations where using Yao's method seems to hit a wall.

As with Yao's method, we contrast the behavior of a supposed test when it is run over an input chosen according to some distribution over "yes" instances, with its behavior when it is run over an input chosen according to some distribution over "no" instances. However, while in the traditional method these two distributions are chosen based only on the property (and should work against all possible algorithms of a given class), here the distributions are in fact *tailor made* for the specific analyzed algorithm. Note that special care must be taken in the definition of such an input distribution. It may not depend on the "real-time" behavior of the algorithm (i.e. it may not adapt itself to the identity of the random queries that the algorithm has made), and is instead constructed based only on the *description* of the algorithm.

The second theme is the use of *Shannon entropy.* Our goal here is to prove that if C is C'-partially testable, then C' cannot be too large. For achieving this we assume that a testing algorithm exists, and then contrast a uniformly random choice of a word in C' with another word chosen from a "dangerous" distribution over words far from C. The assumption that the test in fact distinguishes the two distributions allows us to show that a uniformly random choice of a word in C' has low entropy, and hence C' must be small. Using entropy instead of direct counting is crucial for applying our main method to obtaining a bound against 2-sided error tests, rather than only 1-sided error ones.

A third theme used in the proof against adaptive algorithms is that of first parsing the input through a specially constructed injective mapping, called a "reader", which is crucial for "exposing" low-entropy portions in this setting. We are in fact considering not just one input distribution, but several of them as the reader is constructed.

5.2.1 Proving a bound against non-adaptive algorithms

The bound against non-adaptive algorithms showcases many of the general themes. A supposed C'-partial test with q queries is in essence a distribution over query sets of size q, such that with high probability the chosen query set is one that highlights a difference between members of C' and inputs far from being in C. As a toy example, assume first that the test is additionally 1-sided, and "well-spread" with respect to the probabilities of querying any particular index. In this case, for every ϵ-far input, the high probability of finding a forbidden substructure (as this is the only way a 1-sided test can reject) translates to having many disjoint q-tuples of indexes where in each of them there is a value that a member of C' cannot take (as a hypothetical forbidden structure must exist). This would give a cross product bound on the size of C'.

As our tests are not necessarily "well-spread", we will construct a specialized distribution that depends on the specific testing algorithm (but is independent of any particular running instance). For handling 2-sided tests we use a feature of entropy that allows for bounds analogous to combinatorial cross product bounds, namely the subadditivity of the entropy measure.

To construct a "dangerous" distribution over words far from being in C, we first take note of the "heavy" indexes, which are those bits of the input that are with high probability part of the query subset of the investigated testing algorithm. There will be only a few of those, and our distribution over far words would be that of starting with a restriction of a uniformly random word in C' to the set of heavy indexes, and augmenting it with independently and uniformly chosen values to all other input bits. When contrasted with the uniform distribution over all members of C', we obtain that there must be many query sets that show a distinction between the two distributions over the non-heavy indexes with respect to the heavy ones. This means that the values of the non-heavy indexes in each such query set do not behave like a uniformly independent choice, and thus have a corresponding entropy (conditioned on the heavy index bits) that is significantly less than the maximal possible entropy. Having many such query sets in essence means that we can find many such sets that are disjoint outside the heavy indexes, which in turn leads to an entropy bound by virtue of subadditivity (when coupled with general properties of linear codes).

5.2.2 Proving a bound against adaptive algorithms

An adaptive algorithm cannot be described as a distribution over query sets, but rather as a distribution over small decision trees of height q that determine the queries. Therefore low-entropy index sets cannot be readily found (and in fact do not always exist). To deal with this we employ a new technique, that allows us to "rearrange" the input in a way that preserves entropy, but does admit disjoint low-entropy sets.

This new construction is a *reader*, which in essence is an adaptive algorithm that reads the entire input bit by bit (without repetitions). As this adaptive algorithm always eventually reads the entire input, it defines a bijection between the input to be read and the "reading stream", i.e. the sequence of values in the order that the reader has read them.

The construction of this reader is fully based on the description of the q-query adaptive algorithm that C'-partially tests for C (again we assume that such an algorithm exists). In fact we contrast the uniform distribution over members of C' with not one but many possible distributions over inputs far from C. At every stage we obtain that, as long as our reader has not yet read a large portion of the input, the adaptive test can provide a decision tree over the yet-unread bits that shows a difference between a uniformly random member of C' (conditioned on the values of the bits already read) and an independently uniform random choice of values for the unread bits. Our reader will be the result of "concatenating" such decision trees as long as there are enough unread bits. Thus, in the "reading stream" we have sets of q consecutive bits, each with low entropy (as it is distinguishable from independently uniform values). When there are not enough unread bits left, we read all remaining bits arbitrarily, and use general properties of large dual distance codes to bound the entropy on that final chunk.

The method of constructing a reader not only allows us to do away with the exponential penalty usually associated with moving from non-adaptive to adaptive algorithms, but we additionally obtain better bounds for non-adaptive algorithms as well. This is because a reader can do away also with the penalty of moving from the situation of having many low-entropy query sets to having a family of sets disjoint outside the heavy indexes, in essence by constructing the reader for the uniform distribution over C' based on not one but many "dangerous" input distributions.

5.2.3 Testing decomposable properties through universal testing

Suppose that a property P defined over $\{0,1\}^n$ is decomposable to properties $P_1, \ldots, P_k$, so that each of them is in itself ϵ-testable with $q(\epsilon)$-queries for every $\epsilon > 0$ (the same arguments work also for partial testability, but we restrict the discussion here to proper testability for the sake of explanation). How can we test for all of P at once? The simplest way would be to juxtapose the individual tests for every P_i, which would give a test with $O(kq\log(k))$ many queries (accounting also for the necessary probability amplification). However, in our discussion here k rises too fast with n, so we would like the dependence on it to be at most polylogarithmic, even at the cost of replacing the "base complexity" q with a value that depends (sublinearly) on n.

If the tests for all P_i "behave the same", i.e. have the same query distribution, then instead of querying for every test individually we can do the querying once and feed it to all the tests, and then indeed get a test with $O(q\log(k))$ many queries. This is essentially what is done in the preliminary result from [GR15]. Our goal here is to replace the original test with a "universal" test that would work for *any* property for which an original test with the specified parameters exist, and then use it instead of the original individual tests.

In our first preliminary result we construct such a test whose number of queries is bounded by a fixed power of n, but only if every P_i was testable by the very restricted notion of a 1-sided non-adaptive proximity-oblivious test with 2 queries. Such tests allow for a combinatorial viewpoint through their underlying graphs (where an edge connects two indexes $i, j \in \{1, \ldots, n\}$ if with positive probability the test query set is $\{i, j\}$). This allows for some analysis of the probability of picking a "rejecting edge" when every index ("vertex") is picked and queried with probability $n^{-\beta}$ for an appropriate constant β. The hard part in the proof is when the test has some "heavy indexes", corresponding to high degree vertices.

Our second result handles proximity-oblivious q-tests for any fixed q, but unlike the first result, also the power of n in the resulting test depends on ϵ. We essentially make sure that the sampling is "forceful" enough so that any small "erroneous fragment" of the input cannot "propagate" much if it is altered (the test will detect all possible alterations with large propagations, so such alterations will be forbidden). This in turn allows us to analyze $1/\rho(\epsilon/2)$ many $\epsilon/2$-far inputs derived from the original input, showing that unless the universal test works, they cannot be all rejected by the original test. The propagation requirement allowing us to analyze these inputs is what causes a dependency on $\rho(\epsilon/2)$ of the power of n.

5.2.4 A non-testable property that is decomposable to testable ones

The property of being a concatenation of two palindromes was mentioned above as one that requires $\Omega(\sqrt{n})$ many queries to test, while being decomposable to $O(n)$ many testable properties (in fact properties admitting a proximity oblivious 2-test). The basic idea from this property is carried over to the properties constructed here. A parity condition ensures that instead of having to correlate two strings (an alleged palindrome and its reverse), we would have to correlate k strings, increasing the bound from $\Omega(\sqrt{n})$ to $\Omega(n^{1-1/k})$. As these k strings are allowed to "slide" relative to each other, the number of k-testable properties that we decompose to would be $O(n^{k-1})$, each one corresponding to a fixing of the locations of the strings.

5.3 Preliminaries

Below we introduce the reader to some basic definitions and results regarding entropy and the dual distance of codes. We refer the reader who is interested in a more thorough introduction of entropy to [CT00, Chapter 2].

First, we introduce the notion of the entropy of a random variable, the entropy of a random variable conditioned on another one, and two well-known lemmas.

Definition 5.3.1 (Entropy)**.** Let X be a random variable over the domain $\mathcal{D}$. The *entropy of* X is defined to be $H[X] = -\sum_{i\in\mathcal{D}} \Pr[X = i]\log(\Pr[X = i])$.

Definition 5.3.2 (Conditional entropy)**.** Let X and Y be random variables over the domain $\mathcal{D}$. The *entropy of* X *conditioned on* Y is defined to be $H[X|Y] = \sum_{y\in\mathcal{D}} \Pr[Y = y]H[X|Y = y]$.

Lemma 5.3.3 (The chain rule)**.** *Assume that X and Y are random variables. The entropy of the combined state determined by both random variables is denoted by $H[X, Y]$. This quantity obeys the chain rule $H[X, Y] = H[X|Y] + H[Y]$.*

Lemma 5.3.4 (Subadditivity)**.** *If X and Y are random variables, then $H[X, Y] \leq H[X]+H[Y]$.*

The variation distance is not a natural fit in the context of entropy. A more fitting notion of distance between distributions is divergence (also known as the Kullback-Liebler divergence [KL51]).

Definition 5.3.5 (Divergence)**.** Let p and q be two distributions over $\mathcal{D}$. The *divergence of q from p* is defined to be $D(p\|q) = \sum_{i\in\mathcal{D}} p(i)\log\left(\frac{p(i)}{q(i)}\right)$.

Fortunately, divergence and variation distance are related via Pinsker's inequality. This was originally proved with worse bounds by Pinsker [Pin64] and has seen many subsequent improvements, the current definitive version being that of Reid and Williamson [RW09].

Lemma 5.3.6 (Pinsker's inequality)**.** *Assume that p, q are two distributions over the domain $\mathcal{D}$. The variation distance between p and q is related to the divergence of q from p by the inequality $\sqrt{\frac{1}{2}D(p||q)} \geq d_{TV}(p,q)$.*

We will actually be using a simpler corollary of it.

Lemma 5.3.7 (Corollary of Pinsker's inequality)**.** *Assume that X is a random variable distributed according to p over $\mathcal{D}$, and denote the uniform distribution over $\mathcal{D}$ by p_u. The entropy of X is related to its variation distance from the uniform distribution by the inequality $H[X] \leq \log(|\mathcal{D}|) - 2(d_{TV}(p,p_u))^2$.*

Proof. Follows by

$$\begin{aligned} H[X] &= -\sum_{i\in\mathcal{D}} \Pr[X=i]\log(\Pr[X=i]) \\ &= -\sum_{i\in\mathcal{D}} \Pr[X=i]\log(\Pr[X=i]\cdot\frac{1}{|\mathcal{D}|}\cdot|\mathcal{D}|) \\ &= -\sum_{i\in\mathcal{D}} \Pr[X=i]\log(\frac{1}{|\mathcal{D}|}) - \sum_{i\in\mathcal{D}} \Pr[X=i]\log(\Pr[X=i]\cdot|\mathcal{D}|) \\ &= \log(|\mathcal{D}|) - D(p\|p_u) \leq \log(|\mathcal{D}|) - 2(d_{TV}(p,p_u))^2 \end{aligned}$$

where the last step follows from Pinsker's inequality. □

Let $x \in \{0,1\}^n$ and $J \subseteq [n]$. We use $x[J]$ to denote the restriction of x to the indices in J. That is, the vector $\langle x_j \rangle_{j\in J}$. When $C \subseteq \{0,1\}^n$ we use $C[J] = \{x[J] | x \in C\}$.

Let $C \subseteq \{0,1\}^n$. We denote by $U(C)$ the uniform distribution over C. In accordance with the notation above, when $X \sim U(C)$, $X[J]$ denotes the random variable obtained by drawing uniformly from C and then restricting to the indices in J. As a shorthand we use $U(C)[J]$ for the distribution of $X[J]$. We use $U_J(C)$ to denote the result of first drawing a vector x according to $U(C)$, and then replacing $x[[n]\setminus J]$ with a uniformly random vector in $\{0,1\}^{n-|J|}$. In particular, in many cases we will take C to be a singleton, in which case we drop the curly braces and denote this probability distribution by $U_J(x)$.

We will make inherent use of the following result, which can be found e.g. in [MS77, Chapter 1, Theorem 10].

Lemma 5.3.8. *Let C be a linear code with dual distance Γ. If $J \subseteq [n]$ is such that $|J| < \Gamma$ and $X \sim U(C)$, then $X[J]$ is distributed uniformly over $\{0,1\}^{|J|}$.*

We also need the fact that a mostly random input is far from a given code with high probability.

Lemma 5.3.9. *Let $C \subseteq \{0,1\}^n$ such that $|C| \leq 2^{\frac{1}{64}n}$, $\epsilon < 1/8$, and let $J \subseteq [n]$ be such that $|J| \leq n/2$. $X \sim U_J(C)$ is ϵ-far from C with probability $1-o(1)$. Furthermore, this is still true when conditioned on any value of $X[J]$.*

Proof. By Chernoff bounds, the probability that a random element $X \sim U_J(C)$ will agree with $c \in C$ in more than $(1-\epsilon)n$ coordinates is at most $\exp\big(-n(1/4-\epsilon)^2\big)$. Taking the union bound over all $c \in C$ gives us the probability bound $|C| \cdot \exp\big(-n(1/4-\epsilon)^2\big) = o(1)$. Since this calculation assumes that $X[J]$ always agrees with $c[J]$, it holds when conditioned on any value of $X[J]$. □

We note (and use throughout) that trivially $H[X|X[J]] = H[X[\{1,\dots,n\}\setminus J]|X[J]]$. Finally, we will need to use Lemma 5.3.8 to help us calculate the entropy of uniform random variables in codes.

Lemma 5.3.10. *Let C be a code with dual distance Γ, let $J \subseteq [n]$ be such that $|J| \le \Gamma$, and let $C' \subseteq C$ and $X \sim U(C')$. Then $H[X|X[J]] \le \log|C| - |J|$. Furthermore, this is true when conditioned on any particular value of $X[J]$.*

Proof. We can partition C according to the values of the bits in J:

$$C = \bigcup_{z\in\{0,1\}^{|J|}} \{c \in C | c[J] = z\}$$

By Lemma 5.3.8, all sets on the right hand side are of size $2^{-|J|}|C|$. Obviously, for all $z \in \{0,1\}^{|J|}$, we have $\{c' \in C'|c'[J] = z\} \subseteq \{c \in C|c[J] = z\}$, simply because $C' \subseteq C$. Thus for every $x \in C'[J]$, we have that

$$\begin{aligned} H[X|X[J]=x] &\le \log|\{c' \in C'|c'[J]=z\}| \\ &\le \log|\{c \in C|c[J]=z\}|. \end{aligned}$$

This completes the "furthermore" part of the lemma. To obtain the $X[J]$-conditioned version, note that by the definition of conditional entropy,

$$H[X|X[J]] = \mathrm{E}_{x\sim U(C')[J]} H[X|X[J]=x] \le \log|C| - |J|.$$

Thus concluding the proof. □

5.4 Nonadaptive lower bound

In this section we prove Theorem 5.1 for the case of a non-adaptive tester and with slightly worse quantitative bounds. For the rest of this section, set $C \subset \{0,1\}^n$ to be a code with dual distance Γ and $|C| \le 2^{\frac{1}{64}n}$. Set $\epsilon < 1/8$ and assume that C is C'-partially testable for $C' \subseteq C$ with q non-adaptive queries.

Next we define a non-adaptive tester for a property. This definition is consistent with the standard one.

Definition 5.4.1 (Non-adaptive property tester)**.** A non-adaptive *ϵ-tester* for a binary code $C \subseteq \{0,1\}^n$ with query complexity $q(\epsilon,n)$ is defined by a collection of query sets $\{Q_i\}_{i\in I}$ of size q together with a predicate π_i for each query set and a distribution μ over I which satisfies:

- If $x \in C$, then with probability at least $2/3$ an $i \in I$ is picked such that $\pi_i(x[Q_i]) = 1$.
- If $d(x,C) > \epsilon$, then with probability at least $2/3$ an $i \in I$ is picked such that $\pi_i(x[Q_i]) = 0$.

For a C'-partial tester the first item must hold only for $x \in C'$.

Set a non-adaptive tester for C', and let $\{Q_i\}_{i\in I}$ be its query sets.

We will be interested only in those query sets which are useful for telling a random element in C' from a mostly random element in $\{0,1\}^n$.

Definition 5.4.2 (J-Discerning query set)**.** Let $J \subseteq [n]$ be such that $|J| \leq n/2$. A query set Q_i is a J*-discerning* set if $d_{TV}(U(C')[Q_i], U_J(C')[Q_i]) \geq 1/8$.

Next we prove that a tester must have a lot of such good query sets.

Lemma 5.4.3. *Set $J \subseteq [n]$ such that $|J| \leq n/2$. With probability at least $1/9$ the query set Q_i picked by the tester is a J-discerning set.*

Proof. Assume the contrary, that is, that with probability greater than $8/9$ the query set Q_i picked by the tester is such that $d_{TV}(U(C')[Q_i], U_J(C')[Q_i]) < 1/8$.

Thus for every such Q_i,

$$|\Pr_{U(C')[Q_i]}[\text{tester accepts}] - \Pr_{U_J(C')[Q_i]}[\text{tester accepts}]| < 1/8.$$

For the case where the query set picked is discerning, which occurs with probability smaller than $1/9$, we have no bound (better than 1) on the difference in probability.

Overall, over the randomness of the tester,

$$|\Pr_{U(C')}[\text{tester accepts}] - \Pr_{U_J(C')}[\text{tester accepts}]| < 8/9 \cdot 1/8 + 1/9 = 2/9.$$

But by the correctness of the tester and Lemma 5.3.9, we arrive at $\Pr_{U(C')}[\text{tester accepts}] \geq 2/3$ while simultaneously $\Pr_{U_J(C')}[\text{tester accepts}] \leq 1/3 + o(1)$, a contradiction. □

We will later want to construct a collection of J-discerning sets disjoint outside of a small fixed portion of the input. Towards this end we prove that J-discerning sets show difference between an element in C' and a mostly random element in $\{0,1\}^n$ even when we only look outside of J.

Lemma 5.4.4. *Assume that Q_i is a J-discerning set, draw $Z \sim U(C')[J]$ and then draw $X \sim U(C')[Q_i]$ conditioned on $X[J] = Z$. With probability at least $1/16$ (taken over the choice of Z), the distribution of $X[Q_i \setminus J]$ is $1/15$-far from $U(\{0,1\}^{|Q_i \setminus J|})$.*

Proof. First note that the distance between $U(C')[Q_i]$ and $U_J(C')[Q_i]$ is the expectation over Z of the distance of $X[Q_i \setminus J]$ from $U(\{0,1\}^{|Q_i\setminus J|})$, conditioned on $X[J] = Z$. By definition, this is at least $1/8$. By simple probability bounds, with probability at least $1/16$, Z is such that the distance of $X[Q_i \setminus J]$ from $U(\{0,1\}^{|Q_i\setminus J|})$ conditioned on $X[J] = Z$ is at least $1/15$. □

However, total variation distance is not very handy for counting. We now use Lemma 5.3.7 to transform our variation bounds into "entropy loss" bounds.

Lemma 5.4.5. *If Q_i is a J-discerning set and $X \sim U(C')[Q_i]$, then the following inequality holds: $H[X[Q_i \setminus J] | X[J]] \leq |Q_i \setminus J| - 0.0005$.*

Proof. Let $L \subseteq \{0,1\}^{|J|}$ be the set of values $z \in \{0,1\}^{|J|}$ such that when drawing $X \sim U(C')[Q_i]$ conditioned on having $X[J] = z$, the distribution of $X[Q_i \setminus J]$ is 1/15-far from $U(\{0,1\}^{|Q_i \setminus J|})$.

Since the entropy is non-negative, we can upper bound

$$H[X[Q_i \setminus J]|X[J]]$$
$$\leq \sum_{z \in L} \Pr_{Z \sim U(C')[J]}[Z = z] H[[Q_i \setminus J]|X[J] = z] + \sum_{z \in \{0,1\}^J \setminus L} \Pr_{Z \sim U(C')[J]}[Z = z]|Q_i \setminus J|.$$

To treat the first summand on the right hand side, we invoke Lemma 5.3.7 to obtain

$$H[[Q_i \setminus J]|X[J] = z] \leq |Q_i \setminus J| - 0.008.$$

Overall we get

$$\sum_{z \in L} \Pr_{Z \sim U(C')[J]}[Z = z] H[[Q_i \setminus J]|X[J] = z] + \sum_{z \in \{0,1\}^J \setminus L} \Pr_{Z \sim U(C')[J]}[Z = z]|Q_i \setminus J|$$
$$\leq |Q_i \setminus J| - 0.0005.$$

Next, we would try to cover the indices in $[n]$ with as many discerning sets as possible. We will need these sets to be disjoint outside a not-too-big set, so that the "entropy loss" could be aggregated. This set of "bad" indices will be the set of bits read by the tester with the highest probability.

Definition 5.4.6. Define $B = \{k \in [n] | \Pr_{Q \sim \mu}[k \in Q] \geq \frac{2q}{\Gamma}\}$.

Observation 5.4.7. $|B| \leq \Gamma/2 \leq n/2$. Therefore Lemma 5.3.9 holds with $J = B$.

Now we can prove that we can find many B-discerning sets which are disjoint outside of B.

Lemma 5.4.8. *There exists a set I_D such that:*

- *For all $i \in I_D$, Q_i is a B-discerning set*
- *For all $i, j \in I_D$, $Q_i \setminus B$ and $Q_j \setminus B$ are disjoint*

$D = \cup_{i \in I_D}(Q_i \setminus B)$ satisfies $\Gamma/2 \geq |D| \geq \frac{\Gamma}{18q^2}$. Additionally, $|I_D| \geq \frac{\Gamma}{18q^3}$.

Proof. We construct the set I_D greedily. Suppose that we have discerning sets covering k bits that are disjoint outside of B. Choose a set randomly using the tester's distribution conditioned on it being B-discerning. By Lemma 5.4.3, this condition increases the probability of every query set, and every bit to be in a query set, times at most 9. By the definition of B, if we choose a query set randomly using the tester's distribution, the probability that it intersects our already covered bits outside of B is at most $9\frac{2q^2}{\Gamma}k$. As long as this number is smaller than 1, such a set exists. Therefore, as long as $k < \frac{\Gamma}{18q^2}$ we have a set to add, leading to the bound. To get the upper bound on $|D|$ we can just stop the process before D gets too big.

The lower bound on the size of I_D follows from the lower bound on the size of D. □

Finally, we are ready to bound the entropy of a uniformly random codeword from C'. We use the chain rule to split this into bounding the entropy of the bits in B, the entropy of the bits in D conditioned on the bits of B, and the entropy of everything else conditioned on the bits in $D \cup B$.

Lemma 5.4.9. *If $X \sim U(C')$, then its entropy $H[X]$ is bounded away from the maximal $\log|C|$ by the inequality $H[X] \leq \log|C| - 0.0005\frac{\Gamma}{18q^3}$.*

Proof. First, by the chain rule for entropy and the fact that $D \setminus B = D$,

$$H[X] = H[X|X[D \cup B]] + H[X[D]|X[B]] + H[X[B]]$$

We proceed by bounding each element in the sum. First, trivially:

$$H[X[B]] \leq |B|$$

Next, invoke Lemma 5.3.10 over $D \cup B$, as $|D \cup B| \leq \Gamma$. This gives us:

$$H[X|X[D \cup B]] \leq \log|C| - |D \cup B|$$

Now, recall that $\bigcup_{i \in I_D}(Q_i \setminus B) = D$. Since these sets are disjoint outside of B, we employ subadditivity to get:

$$H[X[D]|X[B]] = H[X[D \setminus B]|X[B]] \leq \sum_{i \in I_D} H[X[Q_i \setminus B]|X[B]]$$

Now, since these are all B-discerning sets, by Lemma 5.4.5 we know that for all $i \in I_D$ we have that $H[X[Q_i \setminus B]|X[B]] \leq |Q_i \setminus B| - 0.0005$. By Lemma 5.4.8 we know that $|I_D| \geq \frac{\Gamma}{18q^3}$. Summing up we get:

$$\sum_{i \in I_D} H[X[Q_i \setminus B]|X[B]] \leq |D| - 0.0005|I_D| \leq |D| - 0.0005\frac{\Gamma}{18q^3}$$

That is, $H[X[D]|X[B]] \leq |D| - 0.0005\frac{\Gamma}{18q^3}$. Summing up the bounds on $H[X|X[D \cup B]]$, $H[X[D]|X[B]]$ and $H[X[B]]$ we get the statement of the lemma. □

From this it follows that:

Theorem 5.9 (Weak form of the main theorem). *For $C' \subseteq C$, if C is C'-partially testable with q non-adaptive queries, then*

$$|C'| = 2^{H[X]} \leq |C| 2^{-0.0005\frac{\Gamma}{18q^3}}.$$

5.5 Adaptive lower bound

In this section we prove Theorem 5.1 in its full generality. We start by introducing the mechanism of a *reader*, which allows us to separate the adaptivity and randomness of the algorithm.

Definition 5.5.1 (Reader). A *k-reader* r is a sequence $r_0, r_1, \ldots, r_{k-1}$, where all the readers $r_i : \{0,1\}^i \to \{1,\ldots,n\}$ satisfy for all $i < j$ and $y \in \{0,1\}^j$ that $r_i(y[\{1,\ldots,i\}]) \neq r_j(y)$.

Given an input $x \in \{0,1\}^n$, the reader defines a sequence of its bits. This is the *reading* of x, defined below.

Definition 5.5.2 (Reading)**.** Given $x \in \{0,1\}^n$ and a k-reader r, the *reading* $R_{r(x)}$ of x according to r is a sequence $y_1, \ldots, y_k$ defined inductively by $y_{i+1} = x_{r_i(y_1,\ldots,y_i)}$. We define $r_i(x)$ to be $r_i(y_1, \ldots, y_i)$. The set of *unread bits* $U_{r(x)}$ is the subset of $\{1, \ldots, n\}$ that did not appear as values of $r_1, \ldots, r_k$ in the reading.

We can now define an adaptive tester as a distribution over readers and decision predicates.

Definition 5.5.3 (Adaptive tester)**.** An *adaptive* ϵ-tester for a code $C \subseteq \{0,1\}^n$ with query complexity $q = q(\epsilon, n)$ is defined by a collection of q-readers $\{r^i\}_{i \in I}$ together with predicates π_i for each reader, and a distribution μ over I which satisfies:

- For all $x \in C$, $\Pr_{i \sim \mu}\left[\pi_i(R_{r^i(x)}) = 1\right] \geq 2/3$.
- For all $x \in \{0,1\}^n$ such that $d(x, C) > \epsilon$, we have that $\Pr_{i \sim \mu}\left[\pi_i(R_{r^i(x)}) = 0\right] \geq 2/3$.

For a C'-partial tester the first item must hold only for $x \in C'$.

Part of the usefulness of readers is that if we can construct a reader that reads the entire input, then reading the property C' through it preserves its size.

Observation 5.5.4. If r is an n-reader, then the function mapping every $x \in \{0,1\}^n$ to its reading $R_{r(x)}$ is a bijection.

Proof. Suppose that $x' \neq x$, and let $i \in \{1, \ldots, n\}$ be the least index such that $x_{r_i(x)} \neq x'_{r_i(x)}$. Such an i must exist since r reads all bits, and $x' \neq x$. Note that $r_i(x) = r_i(x')$, since it is the first bit read to be different (and thus $y_1, \ldots, y_{i-1} = y'_1, \ldots, y'_{i-1}$). Thus $x_{r_i(x)} \neq x'_{r_i(x')}$ and therefore $R_{r(x)} \neq R_{r(x')}$. □

In light of the above, we will construct an n-reader and bound the size of C' when permuted by its reading. However, while the end product of the construction is an n-reader, the intermediate steps might not be k-readers for any k. Thus we need to introduce a more general notion.

Definition 5.5.5 (Generalized reader)**.** A *generalized reader* r is a sequence $r_0, r_1, \ldots, r_{n-1}$ where $r_i : \{0,1\}^i \to \{1, \ldots, n\} \cup \{\star\}$ satisfy for all $i < j$ and $y \in \{0,1\}^j$ one of the following

- $r_i(y[\{1, \ldots, i\}]) \in \{1, \ldots, n\} \setminus r_j(y)$
- $r_i(y[\{1, \ldots, i\}]) = r_j(y) = \star$

Given a generalized reader r, a *terminal sequence* in the reader is $y \in \{0,1\}^i$ such that $r_i(y_1, \ldots, y_i) = \star$, while either $r_{i-1}(y_1, \ldots, y_{i-1}) \neq \star$ or $i = 0$.

If we fix a certain $x \in \{0,1\}^n$ then a generalized reader defines a sequence of non-repeating indices that at some point may degenerate to a constant sequence of $\star$. Note that every k-reader naturally defines a generalized reader by setting all undefined functions to map everything to $\star$.

It is useful to think of a (possibly generalized) reader as a decision tree. With a generalized reader, we will often want to continue the branches of the tree with another reader. This operation is called *grafting*. We start with the notion of a 0-*branch* and a 1-*branch*.

Definition 5.5.6 (0-branch, 1-branch)**.** Let r be a reader, possibly generalized. The *0-branch of r* is the reader r' defined by $r'_i(y_1, \dots, y_i) = r_{i+1}(0, y_1, \dots, y_i)$. Similarly, the 1-branch of r is the reader r'' defined by $r''_i(y_1, \dots, y_i) = r_{i+1}(1, y_1, \dots, y_i)$.

We can now define grafting, and will do so recursively. Informally, grafting a reader t onto r at y means that at every $\star$ in the decision tree of r that can be reached after reading y, we continue the reading according to t. In other words, this is the process of appending a decision tree t to another decision tree r given a certain history of reads y.

Definition 5.5.7 (Grafting)**.** Let r and t be generalized readers and $y \in \{0,1\}^i$ be a terminal sequence in r. The *grafting of t onto r on the branch y* is a new reader $r^{t,y}$ defined as follows.

- If $t_0 \in \{r_0(y_1, \dots, y_i), \dots, r_{i-1}(y_1, \dots, y_i)\}$, graft the y_{t_0}-branch of t onto r at $y_1, \dots, y_i$.
- If $t_0 \notin \{r_0(y_1, \dots, y_i), \dots, r_{i-1}(y_1, \dots, y_i)\}$, then set $r_i(y_1, \dots, y_i) = t_0$, call the new reader r', and then graft the 0-branch of t onto r' at $y_0, \dots, y_i, 0$ and the 1-branch of t onto t at $y_0, \dots, y_i, 1$.

Repeat the above recursively, with the base case being the grafting of an identically $\star$ reader onto r by not changing anything.

Note that the grafting of a generalized reader onto another results in a generalized reader. Note that it is also possible that $r^{t,y} = r$ when all bits that t may read were already read by r on y.

To introduce the notion of a reader that discerns a random input from an input from C', we will first need to formulate a notion of executing a reader, which is inherently adaptive, on a partly random input.

Definition 5.5.8 (J-Simulation of a reader)**.** Let r be a q-reader, $J \subseteq [n]$ and $y \in \{0,1\}^{|J|}$. The *J-simulation of r on y* is the distribution $S(r, y, J)$ over $\{0,1\}^q$ defined to be $R_{r(x)}$ where $x \sim U_J(y)$, that is, $x[J] = y[J]$ and all bits of x outside of J are picked independently and uniformly at random from $\{0,1\}$.

We now introduce the notion of a reader discerning a random input from an input from C'.

Definition 5.5.9 (J-Discerning reader)**.** Let r be a (possibly generalized) reader, $J \subseteq [n]$ and $y \in \{0,1\}^{|J|}$. Let x be a uniform random variable in $\{c \in C' | c[J] = y\}$. We say that r is a *J-discerning reader for y* if it holds that $d_{TV}(R_{r(x)}, S(r, y, J)) \geq 1/8$.

Next, we prove that in a given test many of its readers are discerning.

Lemma 5.5.10. *Set $J \subseteq [n]$ such that $|J| \leq n/2$ and $y \in \{0,1\}^{|J|}$. With probability at least $1/9$ the q-reader r picked by the tester is J-discerning for y.*

Proof. Let r be a reader that is not J-discerning for y. Let B and G be random variables such that $B \sim U_J(y)$ and $G \sim U(\{c \in C' | c[J] = y\})$. Denote by π_r the predicate associated with r. By our assumption,

$$|\Pr[\pi_r(R_{r(B)}) = 1] - \Pr[\pi_r(R_{r(G)}) = 1]| < 1/8.$$

Now assume that with probability greater than $8/9$, the q-reader picked is not J-discerning for y. Now consider the difference in acceptance probability when drawing a reader according to μ.

$$|\Pr_{r\sim\mu}[\pi_r(R_{r(B)})=1]-\Pr_{r\sim\mu}[\pi_r(R_{r(G)})=1]|$$
$$<8/9\cdot 1/8+1/9=2/9.$$

But by Lemma 5.3.9 (the "furthermore" claim) and the correctness of the tester, we have that $\Pr_{r\sim\mu}[\pi_r(R_{r(B)})=1]\leq 1/3$, and by the correctness of the tester $\Pr_{r\sim\mu}[\pi_r(R_{r(G)})=1]\geq 2/3$, a contradiction. □

A common operation will be to graft a discerning reader with additional arbitrary bits. This does not cause a discerning reader to stop being one.

Definition 5.5.11. Let r and s be generalized readers. We say that r *contains* s if for every $x\in\{0,1\}^n$, the sequence of non-$\star$ elements in $R_{s(x)}$ is a prefix of $R_{r(x)}$.

Note that in particular, whenever we graft t onto r along some branch, we obtain a reader that contains r.

Lemma 5.5.12. *Let r and s be generalized readers such that r contains s. Let $J\subseteq[n]$ and $y\in\{0,1\}^{|J|}$. If s is a J-discerning reader for y, then so is r.*

Proof. Let $B\sim U_J(y)$ and $G\sim U(\{c\in C'|c[J]=y\})$. Consider $R_{r(B)}$. Its outcomes can be partitioned according to their $R_{s(B)}$ prefixes. Thus every event defined by values of $R_{s(B)}$ can be written also as some event defined by values of $R_{r(B)}$. The same is true for $R_{s(G)}$ and $R_{r(G)}$, with the same translation. Therefore $d_{TV}(R_{r(x)},S(r,y,J))\geq d_{TV}(R_{s(x)},S(s,y,J))$, implying the lemma. □

To prove that a uniform choice in C' does not have high entropy, we graft discerning readers one onto the other. We will want to make sure that all the branches of the decision tree are of the same height throughout the grafting, and thus we define the notion of a *padded grafting.*

Definition 5.5.13 (q-Padded grafting)**.** Let r be a generalized reader, t be a q-reader and $y\in\{0,1\}^i$ be a terminal sequence in r. The *q-padded grafting of t onto r on the branch y* is defined by the following process. First, let r' be the grafting of t onto r at the branch y. Now perform the following repeatedly: Let $z_1,\dots,z_j$ with $j<q$ be such that $r'_{i+j-1}(y_1,\dots,y_i,z_1,\dots,z_{j-1})\neq\star$ while $r'_{i+j}(y_1,\dots,y_i,z_1,\dots,z_j)=\star$, or $j=0$ and $r'_i(y_1,\dots,y_i)=\star$. Let k be an arbitrary index not in the set of indexes $\{r'_0,\dots,r'_{i+j-1}(y_1,\dots,y_i,z_1,\dots,z_{j-1})\}$, and redefine the reader $r'_{i+j}(y_1,\dots,y_i,z_1,\dots,z_j)=k$. Repeat this process as long as such $z_1,\dots,z_j$ with $j<q$ exist.

The above is basically grafting additional arbitrary reads, so that the end-result will always read exactly q bits after reading the sequence $y_1,\dots,y_i$. The next observation together with Lemma 5.5.12 implies that q-padded grafting of a J-discerning reader is equivalent to a grafting of some other J-discerning reader.

Observation 5.5.14. Let r be a generalized reader, t a q-reader and $y\in\{0,1\}^i$ a terminal sequence in r. There exists a reader s containing t such that the q-padded grafting of t onto r at y is equivalent to the grafting of s onto r at y.

Now we can finally prove the main lemma, by performing repeated q-padded grafting of discerning readers one onto another.

Lemma 5.5.15. *If $X \sim U(C')$, where C is C'-partially testable with q queries, then it holds that $H[X] \le \log|C| - \lfloor \frac{1}{32}\Gamma/q \rfloor$*

Proof. Let us construct an n-reader and consider the entropy of C' when permuted by this reader. We assume that Γ is a multiple of q, otherwise we replace it by $q\lfloor \Gamma/q \rfloor$.

Start with the 0-reader r^0 (i.e. the reader with all functions being identically $\star$). Let s be a $\emptyset$-discerning q-reader for the empty word, which must exist since the adaptive tester must pick one with positive probability. Set r^1 to be the q-padded grafting of s onto r^0 on the branch of the empty word (so in particular r^1 contains s).

Assume that we have constructed the jq-reader r^j. If $jq \ge \Gamma$, graft a reader that reads all remaining bits arbitrarily onto r^j on every branch. Else, perform the following for all branches $y \in \{0,1\}^{jq}$ to obtain r^{j+1} (noting that they are all terminal sequences in r^j):

- If there is no member of C' with the reading $R_{r^j(y)}$, perform a q-padded grafting of an arbitrary q-reader onto r^j at the branch y,
- If such a member exists, let s be a reader which is an $\{r^j_1(y), r^j_2(y), \ldots, r^j_{jq}(y)\}$-discerning reader for y (which exists by Lemma 5.5.10). Perform a q-padded grafting of s onto r^j at the branch y.

Now let r be the resulting n-reader, let $r_{R(C')}$ be the image of C' under the reading of r, and let $X \sim U(r_{R(C')})$. By Observation 5.5.4, the distribution of X is the same as starting with a uniformly random member of C' and then taking its reading according to r. By the chain rule $H[X] = H[X[\{1,\ldots,\Gamma\}]] + H[X|X[\{1,\ldots,\Gamma\}]]$.

Note that in the case of a word from C', the maximal j in the construction is equal to Γ/q. By the chain rule we may write

$$H[X[\{1,\ldots,\Gamma\}]] = \sum_{i=1}^{\Gamma/q} H[X[\{(i-1)q+1,\ldots,iq\}]|X[\{1,\ldots,(i-1)q\}]]$$

and since each sequence of q bits is from the grafting of a reader which is discerning with respect to all the previous ones, we may apply Lemma 5.3.7 to obtain

$$\begin{aligned} H[X[\{1,\ldots,\Gamma\}]] &= \sum_{i=1}^{\Gamma/q} H[X[\{(i-1)q+1,\ldots,iq-1\}]|X[\{1,\ldots,(i-1)q-1\}]] \\ &\le \sum_{i=1}^{\Gamma/q} \left(q - \frac{1}{32}\right) \le \Gamma - \Gamma/q \cdot \frac{1}{32}. \end{aligned}$$ □

By Lemma 5.3.10 (the furthermore part, for every $y \in \{0,1\}^n$ using $J = \{r_1(y),\ldots,r_\Gamma(y)\}$), $H[X|X[\{1,\ldots,\Gamma\}]] \le \log|C| - \Gamma$, so by summing it all up we get $H[X] \le \log|C| - \Gamma/q \cdot \frac{1}{32}$.

Proof of Theorem 5.1. Let C a code with dual distance Γ and suppose that it is C'-partially testable with q queries for some $C' \subseteq C$. By Lemma 5.5.15, if X is uniformly distributed in C', then $H[X] \le \log|C| - \lfloor \frac{1}{32}\Gamma/q \rfloor$. This implies that $|C'| = 2^{H[X]} \le 2^{-\lfloor \Gamma/32q \rfloor} \cdot |C|$. □

5.6 Properties with a proximity oblivious 2-test decomposition

For simplicity of presentation all the proofs here are for a property P which is decomposable to properties $P_1, \dots, P_\ell$ that in themselves admit a proximity oblivious 2-test, rather than just a P_i-partial test for P. A sketch on how to extend this to the more general setting is found at the end of this section.

Definition 5.6.1 (P-witness)**.** Let $P \subseteq \{0,1\}^n$ be a property and $w \in \{0,1\}^n$. A *P-witness* against w is a set $Q \subseteq [n]$ such that for every $w' \in \{0,1\}^n$, if $w'_i = w_i$ for every $i \in Q$, then $w' \notin P$.

The family of witness sets for a specific w is closed under taking supersets. Note that any 1-sided q-test essentially rejects only if their query set is a witness. A proximity oblivious 1-sided test is a non-adaptive one which is also independent of the proximity parameter ϵ, essentially just a probability distributions over query sets of a fixed size q. This means that the following definition of a proximity-oblivious test is in fact equivalent to Definition 5.1.2 from the introduction.

Definition 5.6.2 (Test defined by witnesses)**.** A *proximity oblivious* 1-sided q-test with the *detection function* $\rho(\epsilon)$ is a probability distribution over query sets of a fixed size q, so that for every ϵ-far input w (for every ϵ) the probability of obtaining a witness against w is at least $\rho(\epsilon)$.

Definition 5.6.3 (universal sampler)**.** For parameters $\epsilon, \eta \in (0,1)$, the (ϵ,η)*-universal sampler* selects a set $R \subseteq [n]$ where, for every $i \in [n]$, $\Pr[i \in R] = \alpha^3 n^{-1/3}$, where we set the value $\alpha = 8\epsilon^{-1}\log\epsilon^{-1} \cdot \log\eta^{-1} \cdot \log n$.

Let $P_1, P_2, \dots, P_\ell \subseteq \{0,1\}^n$ be properties, each having an oblivious one-sided error 2-tester with the same detection function $\rho(\epsilon)$. Given oracle access to an input string $w \in \{0,1\}^n$, the *ϵ-universal algorithm* for $\bigcup_{i=1}^{\ell} P_i$ selects a set $R \subseteq [n]$ according to the $(\epsilon, 1/4\ell)$-universal sampler. If $|R| > 2\alpha^3 n^{2/3}$, then it accepts immediately, and otherwise it queries the input on all indices of R, rejects if R is a P_i-witness against w for every $i \in [\ell]$, and accepts otherwise.

This section is devoted to proving the following:

Lemma 5.6.4 (implying Theorem 5.2)**.** *Suppose that we have a sequence $P_1, P_2, \dots, P_\ell \subseteq \{0,1\}^n$ of properties, where for every $i \in [\ell]$ P_i has a 1-sided error oblivious 2-tester with detection function $\rho(\epsilon)$. If $\epsilon > 0$ is such that $\rho(\epsilon/2) > 0$, then for n large enough (as a polynomial function of $1/\rho(\epsilon/2)$) the ϵ-universal algorithm for $\bigcup_{i=1}^{\ell} P_i$ is a 1-sided error non-adaptive ϵ-tester for $\bigcup_{i=1}^{\ell} P_i$ with query complexity bound $O(n^{2/3}(\epsilon^{-1}\log\epsilon^{-1} \cdot \log\eta^{-1} \cdot \log n)^3)$.*

To arrive at the theorem, we first need to "thin out" the possible 2-test queries.

Definition 5.6.5 (ϵ-trap)**.** A set $\mathcal{Q}$ of size-2 subsets of $[n]$ is called an *ϵ-trap* for a property P, if for every word $w \in \{0,1\}^n$ that is ϵ-far from P, there is some set $Q \in \mathcal{Q}$ which is a P-witness against w.

Lemma 5.6.6. *If P has a 1-sided error oblivious 2-tester with the detection function $\rho(\epsilon)$, then for every ϵ it has an ϵ-trap $\mathcal{Q}$ with $|\mathcal{Q}| \le 9n/\rho(\epsilon)$.*

Proof. This is immediate from running the 2-tester for $9n/\rho(\epsilon)$ many times (so that with positive probability it will happen that every possible ϵ-far word is rejected by some iteration of it), and then setting $\mathcal{Q}$ to be the set of all query sets drawn in these iterations. $\square$

We will also use the following.

Observation 5.6.7. For n larger than some universal constant, the ϵ-universal test will execute the "immediate accept" step (due to R being too large) with probability less than $1/12$.

Lemma 5.6.4 and hence Theorem 5.2 follows by first obtaining $\mathcal{Q}_1, \ldots, \mathcal{Q}_\ell$ as $\epsilon/2$-traps for $P_1, \ldots, P_\ell$ respectively, and then using the union bound for the respective applications of the following statement, which is in some way the "true theorem" of this section.

Theorem 5.10. *Let $\epsilon > 0$, $\eta > 0$, $\mathcal{Q}$ be an $\epsilon/2$-trap for a property P, and w be ϵ-far from P. For n larger than some polynomial function of $|\mathcal{Q}|/n$, the set R produced by the (ϵ, η)-universal sampler is a P-witness against w with probability exceeding $1 - \eta$.*

Observation 5.6.8.

1. $(1 - \alpha^3 n^{-1/3})^{\epsilon n/4} < \eta 2^{-2\epsilon n^{2/3}}/3$,
2. $(1 - \alpha^3 n^{-1/3})^{\alpha^{-2} n^{1/3}} < \eta/3n$,
3. $e^{-\epsilon\alpha(\alpha-4)} < \eta$.

From here on we fix P to be a property, $\mathcal{Q}$ to be its $\epsilon/2$-trap, and $w \in \{0,1\}^n$ to be ϵ-far from P.

Definition 5.6.9 (degree)**.** For every $i \in [n]$ and $\mathcal{Q}' \subseteq \mathcal{Q}$, we define the *degree* of an index i by $\deg_{\mathcal{Q}'}(i) = |\{Q \in \mathcal{Q}' \mid i \in Q\}|$.

Definition 5.6.10 ($\mathcal{W}_w$, $\mathcal{L}_w$ and $\mathcal{M}_w$)**.** For every $w \in \{0,1\}^n$

1. $\mathcal{W}_w$ is the set of all members of $\mathcal{Q}$ that are witnesses against w, and for every $i \in [n]$, $\mathcal{W}_w^i$ is the set of all members of $\mathcal{W}_w$ that contain i.
2. $\mathcal{L}_w = \left\{ Q \in \mathcal{W}_w \;\middle|\; \exists j \in Q \text{ s.t. } \deg_{\mathcal{W}_w}(j) > \alpha^{-2} n^{1/3} \right\}$
3. $\mathcal{M}_w = \mathcal{W}_w \setminus \mathcal{L}_w$.

Definition 5.6.11 (the $\Rightarrow$ notation)**.** Let $(i,a),(j,b) \in [n] \times \{0,1\}$ be distinct. We write $(i,a) \Rightarrow (j,b)$ if $\mathcal{Q}$ has no witness against some $w' \in \{0,1\}^n$ such that $w'_j = \neg b$ while $\mathcal{Q}$ has a witness against every $w^* \in \{0,1\}^n$ such that $w^*_i = a$ and $w^*_j = \neg b$.

Definition 5.6.12 (viable sub-string)**.** Let $B \subset [n]$ be a set of indexes and $\sigma_B : B \to \{0,1\}$. σ_B is a *viable sub-string* if there exist no $h \in [n]$, $a \in \{0,1\}$ and $i, j \in B$, that are not necessarily distinct, such that $(i, \sigma_B(i)) \Rightarrow (h, a)$ and $(j, \sigma_B(j)) \Rightarrow (h, \neg a)$, or $(i, \sigma_B(i)) \Rightarrow (h, a)$ and $\mathcal{Q}$ has a witness against every $w^* \in \{0,1\}^n$ such that $w^*_h = a$.

Definition 5.6.13 (witness against sub-string)**.** Let $B \subset [n]$ and $\sigma_B : B \to \{0,1\}$ be a viable sub-string. $i \in [n]$ is a *witness against* σ_B in $w \in \{0,1\}^n$, if $i \in B$ and $w_i \neq \sigma_B(i)$, or if there exists $j \in B$ such that $(j, \sigma_B(j)) \Rightarrow (i, \neg w_i)$.

Definition 5.6.14 (Inf_{σ_B}, σ_B^{Inf})**.** Let $B \subset [n]$ and $\sigma_B : B \to \{0,1\}$ be a viable sub-string. We define Inf_{σ_B} to be the set containing all the possible witnesses against σ_B. We define $\sigma_B^{\mathrm{Inf}} : \mathrm{Inf}_{\sigma_B} \to \{0,1\}$ so that for every $i \in B$ and $j \in \mathrm{Inf}_{\sigma_B}$, if $(i, \sigma_B(i)) \Rightarrow (j, a)$, then $a = \sigma_B^{\mathrm{Inf}}(j)$.

Lemma 5.6.15. *Let $B \subset [n]$ and σ_B be a viable sub-string. For every $w^* \in \{0,1\}$ such that $w_i^* = \sigma_B^{\mathrm{Inf}}(i)$, for every $i \in \mathrm{Inf}_{\sigma_B}$, all of the members of $\mathcal{W}_{w^*}$ are disjoint from Inf_{σ_B}.*

Proof. Assume for the sake of contradiction that the lemma does not hold. If there exists a member of $\mathcal{W}_{w^*}$ that is contained in Inf_{σ_B}, then σ_B is not a viable sub-string and hence the contradiction. If $\mathcal{W}_{w^*}$ has a member $\{i, j\}$ such that $i \in \mathrm{Inf}_{\sigma_B}$ and $j \notin \mathrm{Inf}_{\sigma_B}$, then there exists $h \in B$ such that $(h, \sigma_B(h)) \Rightarrow (j, \neg w_j^*)$ (we take the h for which $(h, \sigma_B(h)) \Rightarrow (i, w_i^*)$). This is a contradiction to the definition of Inf_{σ_B} as containing all such j. □

Lemma 5.6.16. *Let w be $3\epsilon/4$-far from P. If $\mathcal{W}_w \subseteq 2^B$, then Inf_{σ_B} contains at least $\epsilon n/4$ witnesses against σ_B for any viable sub-string $\sigma_B : B \to \{0,1\}$.*

Proof. Assume for the sake of contradiction that Inf_{σ_B} contains less than $\epsilon n/4$ witnesses against σ_B. Let $w^* \in \{0,1\}$ be such that $w_i^* = \sigma_B^{\mathrm{Inf}}(i)$ if $i \in \mathrm{Inf}_{\sigma_B}$ and otherwise $w_i^* = w_i$. Obviously, w^* is $\epsilon/2$-far from P.

By Lemma 5.6.15, $\mathcal{W}_{w^*}$ does not have any sets that intersect Inf_{σ_B}. Since $\mathcal{W}_w \subseteq 2^B$, $\mathcal{W}_{w^*} \cap 2^{[n] \setminus \mathrm{Inf}_{\sigma_B}} = \emptyset$. Thus, $\mathcal{W}_{w^*} = \emptyset$ and hence $\mathcal{Q}$ has no witness against w. This is a contradiction to $\mathcal{Q}$ being an $\epsilon/2$-trap for P. □

Lemma 5.6.17. *If $|\bigcup_{Q \in \mathcal{W}_w} Q| \le 2\epsilon n^{2/3}$, then $\Pr[R$ is a witness against $w] > 1 - \eta/3$, even if w is only $3\epsilon/4$-far from P.*

Proof. Let $W = \bigcup_{Q \in \mathcal{W}_w} Q$. By assumption, $|W| \le 2\epsilon n^{2/3}$. Let σ_W be a viable sub-string. Since $\mathcal{W}_w \subseteq 2^W$, by Lemma 5.6.16, there are at least $\epsilon n/4$ witnesses against σ_W.

The probability that such a witness is not selected is at most $(1 - \alpha^3 n^{-1/3})^{\epsilon n/4} < \eta 2^{-2\epsilon n^{2/3}}/3$, where the inequality is by Observation 5.6.8. The lemma follows by the union bound over all viable sub-strings for W. □

Lemma 5.6.18. *If $|\bigcup_{Q \in \mathcal{W}_w} Q| > 2\epsilon n^{2/3}$ and $|\bigcup_{Q \in \mathcal{M}_w} Q| < \epsilon n^{2/3}$, then the probability that R is a witness against w is at least $1 - \eta$, for n larger than some polynomial in $|\mathcal{Q}|/n = O(1/\rho(\epsilon/2))$.*

Proof. Observe that $|\mathcal{L}_w| \ge \epsilon n^{2/3}$, because by definition we have $\bigcup_{Q \in \mathcal{W}_w} Q = \bigcup_{Q \in \mathcal{M}_w \cup \mathcal{L}_w} Q$. Let $\mathrm{Piv} \subseteq [n]$ be the set of all i such that $\deg_{\mathcal{W}_w}(i) \ge \alpha^{-2} n^{1/3}$, and σ_{Piv} be such that $\sigma_{\mathrm{Piv}}(i) = \neg w_i$ for every $i \in \mathrm{Piv}$. Note that, for every $i \in \mathrm{Piv}$,

$$\Pr[R \text{ does not contain } j_i \text{ such that } \{i, j_i\} \in \mathcal{W}_w]$$

$$\le (1 - \alpha^3 n^{-1/3})^{\alpha^{-2} n^{1/3}} < \eta/3n,$$

where the last inequality is by Observation 5.6.8. Consequently, by the union bound

$$\Pr[\text{for every } i \in \mathrm{Piv},\ \exists j_i \in R \text{ s.t. } \{i, j_i\} \in \mathcal{W}_w] > 1 - \eta/3.$$

When the event above indeed occurs, it is only for $\sigma = \sigma_{\mathrm{Piv}}$ (out of any $\sigma : \mathrm{Piv} \to \{0,1\}$) that it may be the case that $\{j_i : i \in \mathrm{Piv}\}$ is not a witness against σ. In other words, with probability at least $1-\eta/3$ we obtain the event that R contains witnesses against all possible assignments to Piv, apart from possibly σ_{Piv}. To conclude we partition to two cases that depend on the relationship of σ_{Piv} and w.

If w has at least $\epsilon n^{1/3}$ witnesses against σ_{Piv}, then the probability that such a witness is not selected is less than $(1-\alpha^3 n^{-1/3})^{\epsilon n^{1/3}} < \eta/3$, where the inequality is by Observation 5.6.8. Thus, by the union bound, with probability exceeding $1-\eta$, R is a witness against w (as it also contains witnesses against any possible assignment to Piv).

Assume now that there are less than $\epsilon n^{1/3}$ witnesses against σ_{Piv}. Let $w^* \in \{0,1\}^n$ be such that if $i \in \text{Inf}_{\text{Piv}}$, then $w^*_i = \sigma^{\text{Inf}}_{\text{Piv}}(i)$, and otherwise $w^*_i = w_i$. By the triangle inequality, w^* is $3\epsilon/4$-far from P (for n large enough so that $\epsilon n^{1/3} < \epsilon n/4$). By Lemma 5.6.15, none of the sets in $\mathcal{W}_{w^*}$ intersect Inf_{Piv} and hence $\mathcal{W}_{w^*} \subseteq \mathcal{M}_w$ Consequently,

$$|\bigcup_{Q\in\mathcal{W}_{w^*}} Q| < |\bigcup_{Q\in\mathcal{M}_w} Q| \le 2\epsilon n^{2/3}.$$

Thus, by Lemma 5.6.17, with probability exceeding $1-\eta$, R is a witness against w^* and so against w (this case does not even require us to analyze witnesses against the possible assignments to Piv themselves). □

Lemma 5.6.19. *If* $|\bigcup_{Q\in\mathcal{M}_w} Q| \ge \epsilon n^{2/3}$, *then* $\Pr[\mathcal{W}_w \cap 2^R \ne \emptyset] > 1-\eta$.

Proof. Let R' be a random subset of R, where every member of R is in R' independently with probability α^{-2}. We observe that, by the definition of R, for every $i \in [n]$ independently, we have that $i \in R'$, with probability $\alpha n^{-1/3}$. We next prove that $\Pr[\mathcal{M}_w \cap 2^{R'} \ne \emptyset] > 1-\eta$, and since $R' \subseteq R$ and $\mathcal{M}_w \subseteq \mathcal{W}_w$, this implies the Lemma.

For every integer i, let $d(i) = |\{j \in [n] \mid 2^i \le \deg_{\mathcal{M}_w}(j) < 2^{i+1}\}|$. Let Δ be the expected number of pairs of distinct $Q, Q' \in \mathcal{M}_w \cap 2^{R'}$ such that $Q \cap Q' \ne \emptyset$. We observe that,

$$\Delta \le \alpha^3 n^{-1} \sum_{i=1}^{\frac{\log n}{3} - 2\log\alpha} \binom{2^{i+1}}{2} d(i).$$

We observe that $d(i) \le |\mathcal{M}_w| 2^{-i+1}$. Plugging this into the above,

$$\Delta < \alpha^3 n^{-1} \sum_{i=1}^{\frac{\log n}{3} - 2\log\alpha} 2^{1+2i} |\mathcal{M}_w| 2^{-i+1} = 4\alpha^3 n^{-1} |\mathcal{M}_w| \sum_{i=1}^{\frac{\log n}{3} - 2\log\alpha} 2^i \le 8\alpha n^{-\frac{2}{3}} |\mathcal{M}_w|. \tag{5.1}$$

For every $Q \in \mathcal{M}_w$, let X_Q be a random variable that is 1 if $Q \subseteq R'$, and otherwise 0. Let μ be the expected value of $\sum_{Q\in\mathcal{M}_w} X_Q$. Then,

$$\mu = \frac{\alpha^2 |\mathcal{M}_w|}{n^{\frac{2}{3}}}. \tag{5.2}$$

Consequently, by (5.1), (5.2) and Janson's inequality [AS08, Part 8],

$$Pr[\mathcal{M}_w \cap 2^{R'} = \emptyset] \le e^{-n^{-\frac{2}{3}} |\mathcal{M}_w| \alpha(\alpha-4)} < e^{-\epsilon\alpha(\alpha-4)} < \eta,$$

where the second to last inequality follows from $|\mathcal{M}_w| \ge \epsilon n^{\frac{2}{3}}$, and the last inequality follows from Observation 5.6.8. □

Proof of Theorem 5.10. An ϵ-far w must clearly fall under at least one of Lemma 5.6.17, Lemma 5.6.18 and Lemma 5.6.19. □

We conclude this section with a sketch of how to generalize the result for a decomposition admitting only partial sets. The key is in relaxing the definition of a trap. Under the new scheme, for every i, for a word $\epsilon/2$-far from P (rather than P_i), the "partial" trap $\mathcal{Q}_i$ would be required to contain a witness against P_i. The arguments translate almost verbatim to this setting, only one must be careful with the definitions such as Definition 5.6.11 – the exact wording about the (partial) trap containing a witness against the words under consideration becomes even more important.

5.7 Properties with a proximity oblivious q-test decomposition

In the following we assume knowledge of the definitions and methods of Section 5.6. We also assume everywhere that n is large enough for the arguments to follow. To prove Theorem 5.5 we will make crucial use of sunflowers.

Definition 5.7.1. A *sunflower with center A* is a family of subsets $B_1, \dots, B_t \subseteq \{1, \dots, n\}$ so that every B_i contains A, and $B_1, \dots, B_t$ are disjoint outside of A (a completely disjoint family is a sunflower with center $A = \emptyset$).

Lemma 5.7.2 (sunflower theorem, Erdös and Rado [ER60])**.** *Any family of at least $s = q!t^{q+1}$ sets whose sizes are at most q contains a sub-family of size t which is a sunflower.*

In the following q will be the (constant) number of queries of the proximity-oblivious test, and t will be some power of n, so our required s will essentially be another power of n.

We next define fragments.

Definition 5.7.3 (fragments and violations)**.** A fragment $\xi = (A, v)$ consists of a subset of indices $A \subseteq \{1, \dots, n\}$ and a function $v : A \to \{0, 1\}$. The special case where $A = \emptyset$ is called the *null fragment*.

A fragment $\xi_1 = (A_1, v_1)$ *contains* $\xi_2 = (A_2, v_2)$ if $A_2 \subseteq A_1$ and the restriction of v_1 to A_2 is v_2; in this case the *difference fragment* $\xi_3 = (A_3, v_3) = \xi_1 \setminus \xi_2$ is defined where $A_3 = A_1 \setminus A_2$ and v_3 is the restriction of v_1 to A_3.

A fragment $\xi = (A, v)$ is said to be *violated* by the input w if the restriction of w to A is v. A fragment ξ that is not violated is said to be *satisfied* by w.

It will be easier for us to redefine proximity oblivious tests as distributions over fragments.

Definition 5.7.4 (fragment version of a q-test)**.** A *proximity oblivious q-test* for P is a distribution μ over a set Ξ of fragments of sizes bounded by q (some members of Ξ could be with probability 0) satisfying the following:

- If w satisfies P then no fragment is violated (not even probability 0 ones).
- If w is ϵ-far from P then the probability of picking a violated fragment is at least $\rho(\epsilon)$.

When moving from a q-test as in the original definition of finding a witness against w, to a q-test as per the above definition, the original $\rho(\epsilon)$ might be divided by up to 2^q (since every original query set is converted to all corresponding fragments that are possible witnesses against the input).

We now introduce a definition of universal testers suitable for the analysis in this section:

Definition 5.7.5 (universal tester)**.** For parameters $\gamma \in (0,1), q \in \mathbb{N}$ and a set of fragments Ξ, the (γ, q)-*universal tester* operates in q rounds. In each round it picks a set R_j by taking every index i with probablity $n^{-2\gamma}$. Let $R = \bigcup_{j=1}^{q} R_j$. If $|R| > n^{\gamma}$, then it accepts immediately, and otherwise it queries the input on all indices of R, rejects if the values of w form a witness against P, and accepts otherwise.

For what follows, we note that the "immediate acceptance" step occurs with probability $1 - o(1)$, and we will continue the discussion conditioned on the event where it does not occur.

We next define how fragments can be "shortened" sometimes, through either queries or logical deductions.

Definition 5.7.6 (witnesses and refutations)**.** A *witness* for a fragment ξ is a containing fragment ξ', so that the difference fragment $\xi' \setminus \xi$ is violated by the input w (ξ itself does not have to be violated by w).

A *refutation* for a fragment ξ is a set Ξ of fragments, at least one of which containing ξ, so that no possible input that satisfies the entire set Ξ may satisfy ξ.

Note that in particular a set Ξ is a refutation of the null fragment if and only if it is unsatisfiable.

Our main tool of analyzing the universal sampler is the following:

Definition 5.7.7 (R-reduction of a test)**.** Given a q-test for a property P, as a distribution μ over a set Ξ of fragments, and a set of queries $R \subseteq \{1, \dots, n\}$, the *$R$-reduction* of the test is the result of the following process.

1. For every $i \in R$, we add the corresponding satisfied fragment $(i, \neg w(i))$, where w is the input, to Ξ, for the time being with probability 0 (this is essentially "adding the query i").
2. We add to Ξ (still with probability 0) every fragment for which there is a refutation in Ξ (note that, because of the previous item, this also includes fragments for which there is a witness whose corresponding difference was indeed verified to violate w through R).
3. For every fragment $\xi \in \Xi$ which contains another fragment in $\xi' \in \Xi$ (and is hence made "redundant" by it), we remove ξ from Ξ. If $\mu(\xi)$ was non-zero, we modify μ by adding this probability to the contained ξ' (for this procedure we can pick any contained ξ' which in itself does not contain yet another member of Ξ).

The R-reduction in itself is not necessarily a test for P. It may reject members of P, and it may even contain the null fragment (when that happens Ξ will contain only the null fragment and with probability 1; this in particular means that R is a witness for the property against the input, i.e., that the input's restriction to R is not extensible to any possible string satisfying P).

On the other hand, the following is immediate.

Observation 5.7.8. For every possible R, the R-reduction of the test is still a probability distribution over fragments. Moreover, for every possible input w, the probability of rejection (obtaining a violating fragment) by the R-reduction is at least the corresponding probability by the original test.

Our main argument for Theorem 5.5 lies in the following: We prove that certain events concerning R and the resulting R-reduction of the test occur with probability $1 - o(1)$. Given these events, we prove that if the null fragment is not in the resulting Ξ, then it may not be the case that all $\epsilon/2$-far inputs are rejected with probability $\rho(\epsilon/2)$ by the original test (we will construct too many "disjoint" inputs).

In the following, γ (of the universal sampler) will be chosen small enough as a function of all other parameters that will be defined below. First we define the following with respect to a value β to be chosen later (γ will depend on β).

Definition 5.7.9 (sunflowers of fragments)**.** Let $\xi_1 = (A_1, v_1), \dots, \xi_t = (A_t, v_t)$ be a family of fragments. We say that it's a *sunflower with center* $\xi = (A, v)$ if $A_1, \dots, A_t$ is a sunflower (of sets) with center A, and additionally the restriction of every v_i to A is v.

Definition 5.7.10 (generations)**.** Given a q-test with the set of fragments Ξ, all members of Ξ are said to be *generation* 0. By induction, a fragment is said to be *generation* i if it is the center of a sunflower of n^β fragments whose generation is at most $i-1$ and which are all witnesses for it, or it has a refutation using fragments whose generation is at most i (and unless the fragment is already of a smaller generation).

Fragments not having a designated generation by the above are said to be *generation* ∞.

We will only be interested in fragments of generation up to q due to this simple observation.

Observation 5.7.11. A generation i fragment for $i < \infty$ has length at most $q - i$, so in particular all finite generations are at most q.

A central claim is the following:

Lemma 5.7.12. *Let $R = \bigcup_{j=1}^q R_j$ be the result of q rounds, where in each round every index i is independently chosen to be in R_j with probability $n^{-2\gamma}$. With probability $1-o(1)$, after the j'th round, the $\bigcup_{k=1}^j R_k$-reduction of the test contains all generation j fragments or sub-fragments thereof. This is when γ is chosen to be $\beta/(4q)$.*

Proof. This is proved by induction. The base is $j = 0$ (the $\emptyset$-reduction of the test will still have all the original fragments, or sub-fragments thereof if there were meaningful refutations).

Let us assume that the $\bigcup_{k=1}^{j-1} R_k$-reduction of the test includes all generation $j-1$ fragments or sub-fragments thereof. For a generation j fragment $\xi = (A, v)$ that is the center of a sunflower of witnesses, first let $\xi' = (A', v')$ be any member thereof. The probability that R_j contains $A' \setminus A$ is at least $n^{-2q\gamma}$. Note that when this happens, ξ or a sub-fragment thereof will be in the $\bigcup_{k=1}^j R_k$-reduction as required.

Now there are at least n^β members of the sunflower, and the events of each difference to be included in R_j are all independent (as this is a sunflower). Therefor the probability of none of the events happening is at most $(1 - n^{-2q\gamma})^{n^\beta} < \exp(-n^{\beta - 2q\gamma})$, which is $o(n^{-1-q})$ taking

$\gamma = \beta/(4q)$. Noting that there are not more than n^{1+q} fragments in all, we are done for all such flower centers by a union bound.

The case where the generation j fragment has a refutation by other generation j fragments is immediate, once we know that all fragments that are generation j through being a center of a sunflower are included. $\square$

This claim in turn motivates the following definition.

Definition 5.7.13. Given a test (as a distribution over a set of fragments Ξ) and an input w, the *generational reduction* thereof is the result of the following process:

1. We add to Ξ (with probability 0) every fragment which is of generation i for some $i < \infty$ (and hence $i \leq q$).

2. For every fragment $\xi \in \Xi$ which contains another fragment $\xi' \in \Xi$ (and is hence made "redundant" by it), we remove ξ from Ξ. If $\mu(\xi)$ was non-zero, we modify μ by adding this probability to the contained ξ' (we can pick any contained ξ' which in itself does not contain yet another member of Ξ).

Again the following is straightforward.

Observation 5.7.14. The generational reduction of the test is still a probability distribution over fragments. Moreover, for every possible input w, the probability of rejection (obtaining a violating fragment) by the generational reduction is at least the corresponding probability by the original test.

It is important for us to note the following, as the generational reduction has a better structure than just any randomized R-reduction obtained through sampling.

Lemma 5.7.15. *With probability $1 - o(1)$, the R-reduction of the test is also a reduction of the generational reduction of the test.*

Proof. This is equivalent to Lemma 5.7.12 for $j = q$, because it means that with probability $1 - o(1)$ there will be witnesses in R to all finite generation fragments, recalling also Observation 5.7.11. $\square$

In particular, if the generational reduction has the null fragment in its set of fragments, then with probability $1 - o(1)$ the γ-universal testing algorithm will reject the property. To complete the components required for the proof of Theorem 5.5, we will assume that the null fragment is not in this reduction (i.e. it is of generation ∞, which is implied by R not being a witness against the input with high probability) and reach a contradiction. At this point we use the sunflower theorem.

Lemma 5.7.16. *Let Ξ_G denote the set of fragments of the generational reduction of the test, and assume that it does not contain the null fragment. For n large enough, there exists no fragment (regardless of whether it is violated itself) that is contained in more than $n^{(q+2)\beta}$ members of Ξ_G that are witnesses against it.*

Proof. If $\xi = (A, v)$ was such a fragment, and $\xi_1 = (A_1, v_1), \ldots, \xi_t = (A_t, v_t)$ for $t = n^{(q+2)\beta}$ were (containing) members of Ξ_G that witness it, then (for n such that $n^\beta > q!$) by Lemma 5.7.2 there would have been a sunflower of sets $A_{j_1}, \ldots, A_{j_t}$ for $t = n^\beta$, whose center is some set A' that contains A. Now the restrictions of $v_{j_1}, \ldots, v_{j_t}$ to A' are all identical: Over A these are identical to v, and over $A' \setminus A$ these are identical to the restriction of w to this set. Let v' denote the common restriction of $v_{j_1}, \ldots, v_{j_t}$ to A'. $\xi_{j_1}, \ldots, \xi_{j_t}$ are now also a sunflower of fragments, all witnesses to their center $\xi' = (A', v')$. This would have meant that ξ' is a fragment of some finite generation, which is a contradiction to Ξ_G already corresponding to the generational reduction of the test. $\square$

In particular (through the null fragment), the above means that there are no more than $n^{(q+2)\beta}$ members of Ξ_G that are violated by w. However, Ξ_G in itself could still be very large, as for example it could contain many fragments that would be violated by the bit-wise negation of w.

In the following, we assume that w is an ϵ-far word for which the generational reduction does not contain the null fragment. We then do the following construction.

Definition 5.7.17. Assume that Ξ_G does not contain the null fragment (and is hence satisfiable). We define by induction the following sequences, where $w_0 = w$, $\Xi_0 = \emptyset$ and $B_0 = \emptyset$. We let w^* be any word that violates no member of Ξ_G.

- Ξ_i is the set of the members of Ξ_G that are violated by w_{i-1}.
- $B_i = B_{i-1} \cup \bigcup\{A : \xi = (A, v) \in \Xi_i\}$.
- w_i is identical to w^* over B_i and identical to w outside of it.

We are interested in taking $w_0, \ldots, w_r$, $\Xi_1, \ldots, \Xi_{r+1}$ and $B_1, \ldots, B_r$ for $r = 1/\rho(\epsilon/2)$. The following lemma gives us their required properties.

Lemma 5.7.18. *Assume that Ξ_G does not contain the null fragment. All of the following hold for n large enough.*

- *The sets Ξ_i are all disjoint.*
- *$|\Xi_1| \le n^{(q+2)\beta}$ and $|B_1| \le qn^{(q+2)\beta}$.*
- *$|\Xi_i| \le n^{(q+2)\beta}|B_{i-1}|^q$ and additionally $|B_i| \le |B_{i-1}| + qn^{(q+2)\beta}|B_{i-1}|^q$ for $i > 1$.*
- *$|B_k| \le n^{(5q)^k\beta}$ for $k > 1$.*

Proof. The first item is because Ξ_i cannot contain any fragment whose respective set is inside B_{i-1} (because w_{i-1} is identical to w^* there), or any fragment whose respective set is not contained in B_i (because of how B_i was defined), and we have successive containment $B_{i-1} \subseteq B_i$.

The second item is from the discussion after Lemma 5.7.16, noting also that all members of Ξ_G are of length bounded by q.

The third item is by Lemma 5.7.16 again. We note that violated fragments of Ξ_G here can only come from witnesses in Ξ_G for fragments inside B_{i-1}, and there are less than $|B_{i-1}|^q$ relevant fragments (all possible restrictions of w^* to subsets of size at most q of B_{i-1}).

The fourth item is by basic numeric induction. $\square$

Now we finally have all the components for proving Theorem 5.5.

Proof of Theorem 5.5. We take $\beta = \frac{1}{2}(5q)^{-r}$, where $r = 1/\rho(\epsilon/2)$ and n is assumed to be large enough so that $n^{-1/2} \leq \epsilon/2$. By Lemma 5.7.15, with probability $1 - o(1)$ the R-reduction of the test will include also its generational reduction, i.e. it will be a reduction also of Ξ_G. To conclude we prove that such an R is necessarily a witness against the input. Let us assume on the contrary that the R-reduction, and hence also Ξ_G, does not contain the null fragment.

We refer to the construction of Definition 5.7.17. The choice of parameters above and Lemma 5.7.18 ensure that $|B_r| \leq \epsilon n/2$, and so all the inputs $w_0, \ldots, w_r$ are $\epsilon/2$-close to w and hence are $\epsilon/2$-far from the property. Hence the original q-test and its generational reduction have to reject each of those inputs with probability at least $\rho(\epsilon/2)$. However, this means that in Ξ_G there are $r+1$ disjoint subsets $\Xi_1, \ldots, \Xi_{r+1}$ that are all probabilities of at least $\rho(\epsilon/2)$ by the generational reduction, which is the contradiction to the sum of all probabilities being 1. □

5.8 Highly decomposable properties

We prove here Theorem 5.7. The property that we will use will be the following one of being k-paritic.

Definition 5.8.1 (k-paritic)**.** A string $w = (w_1, \ldots, w_n) \in \{0,1\}^n$ is called *k-paritic* if there exist $i_1, \ldots, i_k$ for which $i_1 = 1$, $i_j + n/2k \leq i_{j+1}$ for all $1 \leq j < k$ and $i_k + n/2k \leq n$, such that for every $0 \leq r < n/2k$ we have $\bigoplus_{j=1}^{k} w_{i_j+r} = 0$.

For fixed $i_1, \ldots, i_k$ as above, we let $P_{i_1,\ldots,i_k}$ denote the property of satisfying $\bigoplus_{j=1}^{k} w_{i_j+r} = 0$ for every $0 \leq r < n/2k$ (for these particular $i_1, \ldots, i_k$).

Theorem 5.7 then immediately follows from Lemma 5.8.2 and Lemma 5.8.4 below.

Lemma 5.8.2. *The property of being k-paritic is decomposable to at most n^{k-1} many properties, so that each of them has a proximity-oblivious 1-sided k-test with detection function $\rho(\epsilon) = \Omega(k\epsilon)$.*

Proof. We decompose the property of being k-paritic to the properties $P_{i_1,\ldots,i_k}$ (as in Definition 5.8.1), where $i_1, \ldots, i_k$ are any indexes such that $i_1 = 1$, $i_j + n/2k \leq i_{j+1}$ for all $1 \leq j < k$ and $i_k + n/2k \leq n$ (note that these properties need not be disjoint). There are less than n^{k-1} such properties (i_1 has one value and every other i_j can have less than n possible values), and their union is clearly the property of being k-paritic.

The proximity-oblivious k-test for every property $P_{i_1,\ldots,i_k}$ is done by taking a uniformly drawn value from $\{0, \ldots, \lceil n/2k \rceil - 1\}$ for r, and checking that the requirement $\bigoplus_{j=1}^{k} w_{i_j+r} = 0$ is satisfied (which uses k queries). To get at the $\Omega(k\epsilon)$ bound on the detection function, we note that for w to be ϵ-far from $P_{i_1,\ldots,i_k}$, at least ϵn values of the possible $\lceil n/2k \rceil$ values for r must be such that $\bigoplus_{j=1}^{k} w_{i_j+r} = 1$, so the probability to get such a value for r is at least $\frac{\epsilon n}{n/2k} = \Omega(k\epsilon)$.□

Before continuing we show that being k-paritic is not too dense.

Lemma 5.8.3. *For every fixed k, a uniformly random member of $\{0,1\}^n$ (each bit being chosen uniformly and independently) is $1/5k$-far from being k-paritic with probability $1 - o(1)$.*

Proof. First we consider a property $P_{i_1,\ldots,i_k}$ for specific $i_1, \ldots, i_k$ as in Definition 5.8.1. For every $0 \leq r < n/2k$, the probability for $\bigoplus_{j=1}^{k} w_{i_j+r} = 1$ is exactly $\frac{1}{2}$, and these events are completely independent for different values of r. Hence, by a straightforward large deviation inequality, with probability at least $1 - 2^{-n/100k}$ it holds that we have a set $R \subset \{0, \dots, \lceil n/2k \rceil - 1\}$ of size at least $n/5k$ so that for every $r \in R$ we have $\bigoplus_{j=1}^{k} w_{i_j+r} = 1$. When this occurs the word $w = (w_1, \dots, w_n)$ is clearly $1/5k$-far from $P_{i_1,\dots,i_k}$.

The lemma now follows from a union bound over all properties $P_{i_1,\dots,i_k}$ (whose number is at most n^{k-1}, see Lemma 5.8.2). □

The following now concludes the proof of Theorem 5.7

Lemma 5.8.4. *For any fixed k, The property of being k-paritic in itself cannot be $1/5k$-tested using $o(n^{1-1/k})$ many queries (not even by 2-sided adaptive algorithms).*

Proof. Here we use Lemma 2.0.10. We will assume that $n = 2kl$ for some integer l, as the move from this to general n involves simple padding. We define two distributions.

- The distribution $\mathcal{D}_P$ starts by first choosing uniformly and independently index values $2l(j-1)+1 \le i_j \le 2l(j-1)+l$ for every $1 < j \le k$, setting $i_1 = 1$. Then we take $w \in \{0,1\}^n$ to be a uniformly random member of $P_{i_1,\dots,i_k}$, the corresponding property defined in the proof of Lemma 5.8.2 (out of the 2^{n-l} members thereof).
- The distribution $\mathcal{D}_N$ is just the uniform distribution over $\{0,1\}^n$.

It is clear that an input drawn according to $\mathcal{D}_P$ is always k-paritic. Also, by Lemma 5.8.3 we have that with probability $1 - o(1)$ an input drawn according to $\mathcal{D}_N$ is $1/5k$-far from being k-paritic.

Also note that for every $v \in \{0,1\}^q$ and every index set $Q \subseteq \{1, \dots, n\}$ of size q, the probability of a word w drawn according to $\mathcal{D}_N$ to agree with v over Q is exactly 2^{-q}. To complete the argument, by Lemma 2.0.10, it remains to show that for every $v \in \{0,1\}^q$ and every $Q \subseteq \{1, \dots, n\}$ of size q where $q = o(n^{1-1/k})$, the probability for such an agreement according to $\mathcal{D}_P$ is at least $(1 - o(1))2^{-q}$.

Let E be the event that there is no $0 \le r < l$ for which $\{i_1 + r, \dots, i_k + r\} \subset Q$. Conditioned on E, the probability of w to agree with v over Q is exactly 2^{-q}, so it remains to show that E occurs with probability $1 - o(1)$. Let $s_1, \dots, s_k$ be members of Q. The only case where there can be a positive probability for the equalities $i_1 + r = s_1, \dots, i_k + r = s_k$ is if $2l(j-1)+1 \le s_j \le 2lj$ for every $1 \le j \le k$. In this case the equalities can happen for at most one value of r (since i_1 is always 1), and then their probability is bounded by $l^{1-k} = (2k/n)^{1-k}$ (as $i_2, \dots, i_k$ are chosen independently).

The number of possible eligible k-tuples $s_1, \dots, s_k$ in Q is at most $(q/k)^k$. By the union bound the probability for E not to occur is then bounded by $(2q/n)^{k-1}(q/k)$. For a fixed k, if $q = o(n^{1-1/k})$ then this probability bound evaluates to $o(1)$, concluding the proof. □

It would be interesting to find out whether there exists a property decomposable into a relatively small number (some power of n) of testable properties that in itself requires a linear number of queries to test. The following standard proposition shows that for being k-paritic, our lower bounds are about as far they can go.

Proposition 5.8.5. *The property of being k-paritic is testable by a non-adaptive 1-sided test, with query complexity of $O(n^{1-1/k}(\log(n)/\epsilon)^{1/k})$, which detects ϵ-far inputs with constant probability.*

Proof. We will use the following algorithm:

- Choose a query set Q by choosing for every $1 \leq j \leq n$ independently whether $j \in Q$, where this occurs with probability $(10kn^{-1}\log(n)/\epsilon)^{1/k}$.
- If $|Q| > 2n(10kn^{-1}\log(n)/\epsilon)^{1/k}$ then accept the input without making any queries (by a large deviation inequality this occurs with probability $o(1)$).
- Otherwise, make all the queries in Q, and accept the input if and only if there exists a k-paritic word $u \in \{0,1\}^n$ whose restriction to Q agrees with all queries made to the input word w.

Clearly, if the input word w is k-paritic then it will always be accepted, either arbitrarily in the second step or by $u = w$ in the third step. It now remains to prove that ϵ-far words are rejected with high probability. The second step assures that the number of queries is always at most $2n(10kn^{-1}\log(n)/\epsilon)^{1/k} = O(n^{1-1/k}(\log(n)/\epsilon)^{1/k})$ (rather than being so only with probability $1 - o(1)$).

We may safely ignore the case where there is acceptance in the second step as it occurs with probability $o(1)$, and henceforth analyze the algorithm as if this step was removed from it. We start by analyzing the property $P_{i_1,\ldots,i_k}$ for specific $i_1, \ldots, i_k$ as in Definition 5.8.1. If w is ϵ-far from $P_{i_1,\ldots,i_k}$, then there is a set $R \subset \{0, \ldots, \lceil n/2k \rceil - 1\}$ of size at least ϵn so that for every $r \in R$ we have $\bigoplus_{j=1}^{k} w_{i_j+r} = 1$. For every fixed $r \in R$, the probability to query its corresponding witness of not being in $P_{i_1,\ldots,i_k}$, i.e. the probability for $\{i_1 + r, \ldots, i_k + r\} \in Q$, is $10kn^{-1}\log(n)/\epsilon$.

The above means that for the specific property $P_{i_1,\ldots,i_k}$, the probability of not detecting a witness for the input not being in the property is at most $(1 - 10kn^{-1}\log(n)/\epsilon)^{\epsilon n}$ which is upper bounded by $\exp(-10k\log(n)) = o(n^{k-1})$. All that remains to do is perform a union bound over all properties $P_{i_1,\ldots,i_k}$, whose union is the property of being k-paritic (see Lemma 5.8.2 and its proof), to see that with probability $1 - o(1)$ our query set is such that there is no k-paritic word u whose restriction to Q agrees with the queries made to w. $\square$

Chapter 6

Distribution Testing with Conditional Samples

6.1 Introduction

In this chapter we study testing of distribution properties in a model where the samples that are obtained from the unknown distribution can be conditioned over specified subsets of the domain. In our setting, we assume that a sampling oracle to the unknown distribution μ over the discrete domain $[n] = \{1, \ldots, n\}$ is provided, that allows us to sample random (according to μ) elements conditioned on any specified subset $S \subseteq [n]$. If the original distribution is described by the probabilities $p_1, \ldots, p_n$ (where the probability for obtaining $i \in [n]$ is p_i), then when restricting to S the probability of sampling $i \in [n]$ is $p_i/(\sum_{j \in S} p_j)$ if $i \in S$ and 0 otherwise (see the formal definition of the model and corresponding testers in Section 6.2).

In various scenarios, conditional samples can be obtained naturally, or come at a low cost relative to that of extracting any sample – see some illustrating examples in Section 6.1.1. This leads to the following natural question: can we reduce the sample complexity of distribution-property testers using conditional samples?

Indeed, conditional sampling is more powerful than the traditional model: We show that with conditional samples several natural distribution properties, such as uniformity, can be tested with a constant number of samples (compared to $\widetilde{\Theta}(\sqrt{n})$ unconditional samples even for uniformity [GR11, BFR$^+$10]). The most general result of this chapter (Section 6.6) is that any label-invariant property of distributions (a symmetric property in the terminology of [Val11]) can be tested using $\mathrm{poly}(\log n)$ conditional samples.[1]

On the other hand, there are properties for which testing remains almost as hard as possible even with conditional samples: We show a property of distributions that requires at least $\Omega(n)$ conditional samples to test (Section 6.8).

Another feature that makes conditional-samples interesting is that in contrast to the testers using ordinary samples, which are non-adaptive by definition, adaptivity (and the algorithmic aspect of testing) in the conditional-sampling model plays an important role. For instance,

[1]We say that $f(\alpha_1, \ldots, \alpha_l) = \mathrm{poly}(g_1(\alpha_1, \ldots, \alpha_l), \ldots, g_k(\alpha_1, \ldots, \alpha_l))$ if there exists a polynomial $p(x_1, \ldots, x_k)$ such that $f \leq p(g_1, \ldots, g_k)$ for all values of $\alpha_1, \ldots, \alpha_l$ in their respective domains.

the aforementioned task of testing uniformity, while still possible with a much better sampling complexity than in the traditional model, cannot be done non-adaptively with a constant number of samples (see Section 6.7.2).

Before we move to some motivating examples, let us address the concern of whether arbitrary conditioning is realistic: While the examples below do relate to arbitrary conditioning, sometimes one would like the conditioning to be more restricted, in some sense describable by fewer than the n bits required to describe the conditioning set S. In fact, many of our algorithms require less than that. For example, the adaptive uniformity test takes only unconditional samples and samples conditioned on a constant size set, so the description size per sample is in fact $O(\log n)$, as there are $n^{O(1)}$ possibilities. The adaptive general label invariant property tester takes only samples conditioned to dyadic intervals of $[n]$, so here the description size is $O(\log n)$ as well. The non-adaptive tests do require general conditioning, as they pick uniformly random sets of prescribed sizes.

6.1.1 Some motivating examples

Lottery machines

The *gravity pick lottery machine* is the most common lottery machine used worldwide to pick random numbers. A set B of balls, each marked with a unique number $i \in \mathbb{N}$, are dropped into the machine while it is spinning, and after certain amount of time the machine allows a single ball to drop out. Ensuring that such a machine is fair is an important real-life problem.[2]

Suppose that, given a machine and set of balls, we wish to test them for being fair. Specifically, we would like to distinguish between the following cases:

- The machine picks the balls uniformly at random, that is, for any subset $B' \subseteq B$ of balls dropped into the machine, and for each $i \in B'$, the probability that the machine picks i is $1/|B'|$;
- The distribution according to which the balls are picked is ϵ-far from uniform (where $\epsilon > 0$ is some fixed constant, and the distance we consider is the standard variation distance defined above).

Suppose furthermore that we wish to distinguish between those cases as quickly as possible, and in particular, within few activations of the machine. Compare the following solutions.

We can use the uniformity tester [GR11] for this task. Obtaining each sample from the underlying distribution requires one activation of the machine (with the entire set B), and we can complete the test using $\widetilde{\Theta}(\sqrt{|B|})$ activations.

Alternatively, using the algorithm we present in Section 6.3.1, using conditional samples we can complete the test using $O(1)$ activations only (the number of activations only has a polynomial dependency on ϵ and is logarithmic in the confidence parameter). Assuming that the drawing probabilities depend only on the physical characteristics of every ball separately, a

[2]As was demonstrated in the the Pennsylvania Lottery scandal, see e.g.
`http://en.wikipedia.org/w/index.php?title=1980_Pennsylvania_Lottery_scandal&oldid=496671681`

conditional sample here corresponds to activating the machine with a specific subset of the balls rather than the entire set B.

This is for testing uniformity. Using the algorithm from Section 6.6, we could also test for any label-invariant property with $\mathrm{poly}(\log |B|)$ activations, which would allow us for example to give an estimation of the actual distance of the distribution from being uniform.

Asymmetric communication scenarios

Suppose that two computers A and B are linked with an asymmetric communication link, in which transmitting information in one of the directions (say from A to B) is much easier than in the other direction (consider e.g. a spacecraft traveling in remote space, with limited energy, computational power and transmitting capability; actually numerous examples of asymmetric communications also exist here on earth). Now assume that B has access to some large data that can be modeled as collection of samples coming from an unknown distribution μ, while A wants to learn or test some properties of μ. We could simulate the standard testing algorithms by sending a request to B whenever a random sample from μ is needed. Assuming that the most important measure of efficiency is how much information is sent by B, it would translate to the sample complexity of the simulated algorithm.

However, if B can also produce conditional samples (for example if it has nearly unlimited cost-free access to samples from the distribution), then any property that is significantly easier to test with conditional samples can be tested with fewer resources here.

Political polls

We mention these here because the modern-day practice of polling actually uses conditional sampling. Rather than taking a random sample of all willing potential participants, the polling population is usually first divided to groups according to common traits, and then each such group is polled separately before the results are re-integrated into the final prediction.

6.1.2 Informal description of results

In all sample-complexity upper bounds listed below there is a hidden factor of $\log(\delta^{-1})$, where δ is the maximal failure probability of the tester. Also, all lower bounds are for a fixed (and not very small) ϵ. The results are summarized in Tables 6.1 and 6.2.

Conditioned upon sets

Testing algorithms in the conditional sampling model may be categorized according to the types of sets they condition upon. This is in addition to the questions of adaptivity and query complexity. The simplest types of sets would be constant sized sets. Another simple type of sets arises when we can endow the probability space with some linear order over the elements, and then only condition on sets which are intervals in this linear order. Actually, all of our adaptive testing algorithms use one of these types of sets. On the other hand, the non-adaptive algorithms seem to require the full generality of the model. This distinction was also made in

[CRS14] (see below about this related work). The different types of sets used for conditional sampling are also summarized in Table 6.1.

Adaptive testing

The first result we prove is that uniformity, and more generally identity to any distribution that is very close to uniform in the ℓ_∞ norm, can be tested (adaptively) with poly(ϵ^{-1}) conditional samples (Theorem 6.4 and Theorem 6.5, respectively). This is done by capturing (for far distributions) both "light" and "heavy" elements in the same small set and then conditioning over it. Our next result is that identity to any known distribution can be tested adaptively with poly$(\log^\star n, \epsilon^{-1})$ conditional samples, where n is the size of the domain (Theorem 6.6). This uses the uniformity result with the bucketing technique of [BFR$^+$10] together with a recursive argument.

A core result is that adaptive conditional samples are enough to construct an *explicit persistent sampler*. Such a sampler is essentially a way to simulate (unconditional) samples from a distribution $\tilde{\mu}$ that is close to μ, and for which we can also provide exact probability queries like the oracle of [BDKR05].

From the construction of the explicit persistent sampler we derive our most general result that any label-invariant (i.e. invariant under permutation of the domain) property of distributions can be tested adaptively with poly$(\log n, \epsilon^{-1})$ conditional samples (Theorem 6.1). In fact, we go further to prove the following stronger result: with poly$(\log n, \epsilon^{-1}, \log(\delta^{-1}))$ conditional samples taken from μ, it is possible to compute a distribution μ' that is ϵ-close to μ up to some permutation of the domain $[n]$ (Theorem 6.2).

Non-adaptive testing

We prove that uniformity can be tested non-adaptively with poly$(\log n, \epsilon^{-1})$ conditional samples. Here too the tester enjoys a certain degree of tolerance, in the sense that it is possible to test identity with any distribution that is close enough to uniform (see Theorems 6.7 and 6.8). This is done by first proving (through bucketing) that a portion of the "total difference" of μ from being uniform is in relatively equal-probability members of $[n]$, and then trying to capture just

Upper bounds	**Adaptive**	
	Sample complexity	Conditioned sets
Uniformity	poly(ϵ^{-1})	Constant size
Identity to known dist.	poly$(\log^\star n, \epsilon^{-1})$	poly$(\log^\star n, \epsilon^{-1})$ size
Label-invariant prop.	poly$(\log n, \epsilon^{-1})$	Dyadic intervals
	Non-adaptive	
	Sample complexity	Conditioned sets
Uniformity	poly$(\log n, \epsilon^{-1})$	General
Identity to known dist.	poly$(\log n, \epsilon^{-1})$	General

Table 6.1: Summary of our upper bounds.

a few of them in a random set of an appropriate size. We also prove (from the uniformity test through standard bucketing arguments) that identity to any known distribution can be tested non-adaptively with $\mathrm{poly}(\log n, \epsilon^{-1})$ conditional samples (Theorem 6.9).

Lower bounds

As already mentioned in the introduction, adaptivity is useful when we have access to conditional sampling. We demonstrate this by proving that testing uniformity non-adaptively requires $\Omega(\log\log n)$ conditional samples, for some fixed $\epsilon > 0$ (Theorem 6.17). We also prove that the tester for any label-invariant property (from our main result) cannot be improved to work with a constant number of conditional samples: There is a label invariant property which requires $\Omega(\sqrt{\log\log n})$ samples to test, whether adaptively or not (Theorem 6.19). Our third lower bound shows that for some properties conditional samples do not help much: There are distribution properties that cannot be tested (even adaptively) with $o(n)$ conditional samples (Theorem 6.1). The first two lower bounds are through a special adaptation of Yao's method, while the last one is through a reduction to general properties of Boolean strings, of which maximally untestable examples are known.

6.1.3 Related work

Independently, Cannone et. al. [CRS14] formulated the distribution testing with conditional samples model as well. In their work they achieve several results. Some of their results overlap with those of this chapter, but most of their work takes a different direction and emphasis.

Uniformity Cannone et. al. give an algorithm for testing uniformity using $\tilde{O}(\epsilon^{-2})$ samples, and also give a lower bound of $\Omega(\epsilon^{-2})$. It is interesting to note that their upper bound only uses conditioning on sets of size 2.

Identity to a known distribution For this problem, Cannone et. al. demonstrate that conditioning on arbitrary sets is stronger than conditioning on sets of size 2. They give an upper bound of $\tilde{O}(\epsilon^{-4}\log^4 n)$ and a lower bound of $\Omega\left(\sqrt{\frac{\log n}{\log\log n}}\right)$ for testing identity to a known distribution using samples conditioned on sets of size 2, and an upper bound of $\tilde{O}(\epsilon^{-4})$ for testing it using samples on arbitrary sets.

Identity between two unknown distributions The case of testing identity between two unknown distributions is an especially interesting one, as it showcases what seems to be a

Lower bounds	**Adaptive**	**Non-adaptive**
Uniformity and identity	—	$\Omega(\log\log n)$
Any label-invariant prop.	$\Omega(\sqrt{\log\log n})$	(follows uniformity)
General properties	$\Omega(n)$	(follows adaptive)

Table 6.2: Summary of our lower bounds.

profound characteristic of the conditional sampling model. In developing an algorithm for this problem, Cannone et. al. introduce the notion of an "approximate EVAL oracle". Such an oracle is given some element $i \in [n]$ and should return a good estimate of the probability of i, while allowed to fail for some small fixed subset of $[n]$. This notion is somewhat reminiscent of the notion of an explicit persistent sampler used in Section 6.6.[3] Using this construction they give an algorithm that uses $\tilde{O}(\epsilon^{-4}\log^5 n)$ conditional samples to test identity between two unknown distributions. They also give an algorithm that uses $\tilde{O}(\epsilon^{-21}\log^6 n)$ samples, but only conditions on sets of size 2.

Estimating the distance from uniformity Cannone et. al. give an algorithm using $\tilde{O}(\epsilon^{-20})$ samples conditioned on sets of size 2 to give an additive approximation for the distance of an unknown distribution from uniformity. It is interesting to note that this is a case of a label invariant property, but for this property their result it outperforms the general algorithm for testing label invariant properties given in this chapter.

Conditional samples over structured domains An interesting research direction arises when one tries to impose some sort of structure on the sets conditioned upon. The simplest case is when limiting their size, but one can imagine other cases. Cannone et. al. consider the case where the universe of elements is linearly ordered, and one may obtain samples conditioned on intervals. For this setting, they give an upper bound of $\tilde{O}(\epsilon^{-3}\log^3 n)$ samples and a lower bound of $\Omega\left(\frac{\log n}{\log\log n}\right)$ samples for testing uniformity. It is interesting to note that our explicit persistent sampler construction is also based only on samples conditioned on intervals (in fact, on the restricted set of dyadic intervals).

Later results Since the research in this chapter was first published, there was a significant amount of follow up work.

Ron and Tsur [RT14] studied the problem of hidden set size approximation. In the relevant variant of this problem, the task is approximating the size of an unknown set S in a known universe U. The algorithm may specify a subset $T \subseteq U$, and if $T \cap S \neq \emptyset$ the algorithm gets a uniform sample in $T \cap S$. This is essentially a special case of the problem of approximating the support size of a distribution. Note that this problem can be solved using the explicit persistent samplers developed in this chapter with sample complexity polylogarithmic in n and the approximation parameter. Ron and Tsur give an adaptive algorithm, polyloglogarithmic in the hidden set size, and a nonadaptive upper bound polylogarithmic in the hidden set size. They also give stronger bounds for the case of interval queries.

Acharya, Cannone and Kamath [ACK14] expand on Cannone et. al.'s results regarding testing identity between two unknown distributions, giving a a lower bound of $\Omega(\sqrt{\log\log n})$ on the sample complexity in the conditional samples model. Acharya et. al. also expand on Ron

[3]A major difference is that our explicit persistent sampler will with high probability conform to exactly one distribution $\tilde{\mu}$ that is close to μ, rather than just give approximate probability values for μ.

and Tsur's results achieving the same complexity bounds for the general problem of support size estimation.

Cannone [Can15] studied the problem of testing monotonicity of distributions in this model. In particular, they give an upper bound of $\tilde{O}(\epsilon^{-22})$ for testing distribution monotonicity with conditional samples.

6.2 Preliminaries

6.2.1 The conditional distribution testing model

Let μ be a distribution over $\{1, \ldots, n\}$, its probabilities denoted by $p_1, \ldots, p_n$, where $p_i = \Pr_\mu[i]$. We will also write $\mu(i)$ for $\Pr_\mu[i]$ where we deal with more then one distribution. The distribution μ is not known to the algorithm explicitly, and may only be accessed by drawing samples. A conditional distribution testing algorithm may submit any set $A \subseteq \{1, \ldots, n\}$ and receive a sample $i \in A$ that is drawn according to μ conditioned on A (and independent of any previous samples).

Thus when a sample is drawn according to μ conditioned on A, the probability of getting j is $\Pr[j|A] = p_j/(\sum_{i\in A} p_i)$ for $j \in A$ and 0 for $j \notin A$. If $\sum_{i\in A} p_i = 0$ then we assume (somewhat arbitrarily) that the algorithm obtains a uniformly drawn member of A.[4]

We measure *distance* using the variation distance (recall Definition 2.0.6): We say that μ is *ϵ-far* from a property $\mathcal{P}$ of distributions over $\{1, \ldots, n\}$, if for every μ' that satisfies $\mathcal{P}$ and is described by $p'_1, \ldots, p'_n$ we have $d_{TV}(\mu, \mu') = \frac{1}{2}\sum_{i=1}^{n} |p_i - p'_i| \geq \epsilon$.

We will consider two types of conditional distribution testing algorithms. Non-adaptive testers, which must decide the conditioned sets to sample from before getting any samples, and adaptive testers, which have no such restriction.

Definition 6.2.1 (Non-adaptive tester)**.** A *non-adaptive distribution tester for a property $\mathcal{P}$ with conditional sample complexity* $t : \mathbb{R} \times \mathbb{R} \times \mathbb{N} \to \mathbb{N}$ is a randomized algorithm, that receives $\epsilon, \delta > 0$, $n \in \mathbb{N}$ and a conditional sampling oracle to a distribution μ over $[n]$, and operates as follows.

1. The algorithm generates a sequence of $t \leq t(\epsilon, \delta, n)$ sets $A_1, \ldots, A_t \subseteq [n]$ (possibly with repetitions).

2. Then it calls the conditional oracle t times with $A_1, \ldots, A_t$ respectively, and receives $j_1, \ldots, j_t$, where every j_i is drawn according to the distribution μ conditioned on A_i, independently of $j_1, \ldots, j_{i-1}$ and any other history.

3. Based on the received elements $j_1, \ldots, j_t$ and its internal coin tosses, the algorithm accepts or rejects the distribution μ.

If μ satisfies $\mathcal{P}$ then the algorithm must accept with probability at least $1 - \delta$, and if μ is ϵ-far from $\mathcal{P}$ then the algorithm must reject with probability at least $1 - \delta$.

[4]See the beginning of Section 6.7 for how to essentially reduce a model without this assumption to this model.

Definition 6.2.2 (Adaptive tester)**.** An *adaptive distribution tester for a property $\mathcal{P}$ with conditional sample complexity* $t : \mathbb{R} \times \mathbb{R} \times \mathbb{N} \to \mathbb{N}$ is a randomized algorithm that receives $\epsilon, \delta > 0$, $n \in \mathbb{N}$ and a conditional sampling oracle to a distribution μ over $[n]$ and operates as follows.

1. For $i \in \{1, \dots, t\}$, where $t = t(\epsilon, \delta, n)$, at the ith phase the algorithm generates a set $A_i \subseteq [n]$, based on $j_1, \dots, j_{i-1}$ and its internal coin tosses, and calls the conditional oracle with A_i to receive an element j_i, drawn according to the distribution μ conditioned on A_i, independently of $j_1, \dots, j_{i-1}$ and any other history.

2. Based on the received elements $j_1, \dots, j_t$ and its internal coin tosses, the algorithm accepts or rejects the distribution μ.

If μ satisfies $\mathcal{P}$ then the algorithm must accept with probability at least $1 - \delta$, and if μ is ϵ-far from $\mathcal{P}$ then the algorithm must reject with probability at least $1 - \delta$.

As is standard in the field of property testing, the primary measure of efficiency of these testers is their sample complexity $t(\epsilon, \delta, n)$.

6.2.2 Tools from previous works

Our algorithms will make use of the Identity Tester of Batu et. al. [BFR+10] (though it is important to note that this result is used mainly as a "primitive" and can be replaced in the sequel with just making enough samples to fully approximate the distribution).

Theorem 6.1 (Identity Tester)**.** *There is an algorithm T for testing identity between an unknown distribution μ' and a known distribution μ, both over $[n]$, with (ordinary) sample complexity $t = \tilde{O}(\sqrt{n}\mathrm{poly}(\epsilon^{-1})\log(\delta^{-1}))$. Namely, T accepts with probability $1 - \delta$ if $\mu' = \mu$ and rejects with probability $1 - \delta$ if μ' is ϵ-far from μ, based on $t(\epsilon, \delta, n)$ independent unconditional samples from μ.*

Bucketing

Bucketing is a general tool, introduced in [BFR+10], that decomposes any explicitly given distribution to a collection of distributions that are almost uniform. In this section we recall the bucketing technique and lemmas from [BFR+10] that we will need for our proofs.

Definition 6.2. Given a distribution μ over $[n]$, and $M \subseteq [n]$ such that $\mu(M) > 0$, the *restriction* $\mu\restriction_M$ is the distribution over M with $\mu\restriction_M(i) = \mu(i)/\mu(M)$ (this is the the same as the conditioning of μ on B, only here we also change the domain).

Given a partition $\mathcal{M} = \{M_0, M_1, \dots, M_k\}$ of $[n]$, we denote by $\mu_{\langle\mathcal{M}\rangle}$ the distribution over $\{0\} \cup [k]$ in which $\mu_{\langle\mathcal{M}\rangle}(i) = \mu(M_i)$. This is the *coarsening* of μ according to $\mathcal{M}$.

Definition 6.3. Given an explicit distribution μ over $[n]$, *Bucket*$(\mu, [n], \epsilon)$ is a procedure that generates a partition $\{M_0, M_1, \dots, M_k\}$ of the domain $[n]$, where $k = \frac{\log n}{\log(1+\epsilon)} < \frac{2}{\epsilon}\log(n)$. This partition satisfies the following conditions:

- $M_0 = \{j \in [n] \mid \mu(j) < \frac{1}{n}\}$;
- for all $i \in [k]$, $M_i = \left\{j \in [n] \mid \frac{(1+\epsilon)^{i-1}}{n} \leq \mu(j) < \frac{(1+\epsilon)^i}{n}\right\}$.

Lemma 6.2.3 (Lemma 8 in [BFR$^+$10])**.** *Let μ be a distribution over $[n]$ and let the buckets be $\{M_0, M_1, \ldots, M_k\} \leftarrow Bucket(\mu, [n], \epsilon)$. Then for all $i \in [k]$, $\|\mu \upharpoonright_{M_i} - U \upharpoonright_{M_i}\|_\infty \leq \epsilon/n$.*

Lemma 6.2.4 (Lemma 6 in [BFR$^+$10])**.** *Let μ, μ' be two distributions over $[n]$ and let the sequence of sets $\mathcal{M} = \{M_0, M_1, \ldots, M_k\}$ be a partition of $[n]$. If $d_{TV}(\mu \upharpoonright_{M_i} - \mu' \upharpoonright_{M_i}) \leq \epsilon_1$ for every $i \in [k]$ and $d_{TV}(\mu_{\langle \mathcal{M} \rangle}, \mu_{\langle \mathcal{M} \rangle}) \leq \epsilon_2$, then $d_{TV}(\mu, \mu') \leq \epsilon_1 + \epsilon_2$. Furthermore, we have the bound $d_{TV}(\mu, \mu') \leq \sum_{0 \leq i \leq k} \mu(M_i) d_{TV}(\mu \upharpoonright_{M_i}, \mu' \upharpoonright_{M_i}) + \epsilon_2$.*

We reproduce the proof to obtain the "furthermore" claim:

Proof. This results from the following.

$$
\begin{aligned}
2d_{TV}(\mu, \mu') &= \sum_{0 \leq i \leq k} \sum_{j \in M_i} |\mu(j) - \mu'(j)| = \sum_{0 \leq i \leq k} \sum_{j \in M_i} |\mu(M_i) \cdot \mu \upharpoonright_{M_i}(j) - \mu'(M_i) \cdot \mu' \upharpoonright_{M_i}(j)| \\
&\leq \sum_{0 \leq i \leq k} \sum_{j \in M_i} |\mu(M_i) \cdot \mu \upharpoonright_{M_i}(j) - \mu(M_i) \cdot \mu' \upharpoonright_{M_i}(j)| \\
&\quad + \sum_{0 \leq i \leq k} \sum_{j \in M_i} |\mu(M_i) \cdot \mu' \upharpoonright_{M_i}(j) - \mu'(M_i) \cdot \mu' \upharpoonright_{M_i}(j)| \\
&= \sum_{0 \leq i \leq k} \sum_{j \in M_i} \mu(M_i) \cdot |\mu \upharpoonright_{M_i}(j) - \mu' \upharpoonright_{M_i}(j)| + \sum_{0 \leq i \leq k} \sum_{j \in M_i} \mu' \upharpoonright_{M_i}(j) \cdot |\mu(M_i) - \mu'(M_i)| \\
&= 2 \sum_{0 \leq i \leq k} \mu(M_i) d_{TV}(\mu \upharpoonright_{M_i}, \mu' \upharpoonright_{M_i}) + \sum_{0 \leq i \leq k} |\mu(M_i) - \mu'(M_i)| \\
&\leq 2 \sum_{0 \leq i \leq k} \mu(M_i) d_{TV}(\mu \upharpoonright_{M_i}, \mu' \upharpoonright_{M_i}) + \epsilon_2
\end{aligned}
$$

This provides the "furthermore" claim. To obtain from the above the original claim note that $2 \sum_{0 \leq i \leq k} \mu(M_i) \sum_{j \in M_i} d_{TV}(\mu \upharpoonright_{M_i}(j), \mu' \upharpoonright_{M_i}(j)) \leq \sum_{0 \leq i \leq k} \mu(M_i) \epsilon_1 = \epsilon_1$. □

6.3 Adaptive testing for uniformity and identity

In the following we formulate our testing algorithms to have a polynomial dependence on $\log(\delta^{-1})$. To make it linear in $\log(\delta^{-1})$ we can first run the algorithm $100 \log(\delta^{-1})$ times with a fixed $\frac{1}{3}$ confidence parameter and then take the majority vote.

6.3.1 Testing uniformity

Theorem 6.4. *There is an (adaptive) algorithm testing uniformity using* $\text{poly}(\epsilon^{-1}, \log(\delta^{-1}))$ *conditional samples independently of n.*

In fact we will prove something slightly stronger, which will be useful in the next sections:

Theorem 6.5 (Near Uniformity Tester)**.** *Let μ be a known distribution over $[n]$ such that $\|\mu - U_n\|_\infty < \frac{\epsilon}{100n}$. Identity with μ can be tested using only* $\text{poly}(\epsilon^{-1}, \log(\delta^{-1}))$ *conditional samples by an adaptive algorithm.*

Proof. This follows from Algorithm 6.1 by Lemmas 6.3.1, 6.3.2 and 6.3.6 below. □

Let μ' be the unknown distribution that is to be sampled from.

Algorithm 6.1 Near Uniformity Tester

Input: Known distribution μ, distance parameter ϵ, confidence parameter δ and universe size n.
Output: "ACCEPT" or "REJECT".

```
1: Take $S$ to be $k = (6/\epsilon)\log(\delta^{-1})$ independent samples according to $\mu'$ (unconditioned).
2: Take $U$ to be $k$ members of $\{1,\ldots,n\}$ chosen uniformly at random.
3: Invoke the Identity Tester of Theorem 6.1 to check whether $\mu' \restriction_{U\cup S}$ is $\frac{\epsilon^2}{600\log(\delta^{-1})}$-close to
   $\mu \restriction_{U\cup S}$ over $U \cup S$ with confidence parameter $\delta/3$, and answer as the tester did.
```

Lemma 6.3.1. *The sample complexity of Algorithm 6.1 is* $\mathrm{poly}(\epsilon^{-1}, \log(\delta^{-1}))$.

Proof. The algorithm draws k samples, and then invokes the closeness tester on a set of size $2k$ and an error parameter polynomial in ϵ^{-1}. Since the sample complexity of the closeness tester is polynomial in the support size and the error parameter, and $k = (6/\epsilon)\log(\delta^{-1})$, the total sample complexity of Algorithm 6.1 is $\mathrm{poly}(\epsilon^{-1}, \log(\delta^{-1}))$. □

Lemma 6.3.2. *If* $d_{TV}(\mu,\mu') = 0$ *then Algorithm 6.1 accepts with probability at least* $1-\delta$.

Proof. If $\|\mu-\mu'\|_1 = 0$ then $\|\mu\restriction_{U\cup S} - \mu'\restriction_{U\cup S}\|_1 = 0$ and then the algorithm will accept if the closeness tester does, which will happen with probability at least $1-\frac{\delta}{3}$. □

Let the individual probabilities for the distribution μ be denoted by $p_1,\ldots,p_n$ and the probabilities for the distribution μ' denoted by $p'_1,\ldots,p'_n$. We first note that

$$2d_{TV}(\mu,\mu') = \|\mu-\mu'\|_1 = \sum_{i=1}^{n} |p_i - p'_i| = 2\sum_{p'_i<p_i}(p_i-p'_i) = 2\sum_{p'_i>p_i}(p'_i-p_i)$$

Assume from now on that this distance is at least 2ϵ (which corresponds to variation distance at least ϵ).

Lemma 6.3.3. *With probability at least* $1-\delta/3$ *we have an* $i\in S$ *for which* $(p'_i - p_i) \geq \frac{\epsilon}{2n}$.

Proof. Clearly $\sum_{p_i<p'_i<p_i+\epsilon/2n}(p'_i-p_i) < \frac{1}{2}\epsilon$. Therefore:

$$\sum_{p'_i\geq p_i+\epsilon/2n} p'_i > \sum_{p'_i\geq p_i+\epsilon/2n}(p'_i-p_i) = \sum_{p'_i>p_i}(p'_i-p_i) - \sum_{p_i<p'_i<p_i+\epsilon/2n}(p'_i-p_i) > \frac{1}{2}\epsilon$$

This means that after $(6/\epsilon)\log(\delta^{-1})$ samples, with probability at least $1-\delta/3$ we will get an i with such a p'_i into S. □

Lemma 6.3.4. *With probability at least* $1-\delta/3$ *we have an* $i\in U$ *for which* $p'_i < p_i$.

Proof. Note that $\sum_{p'_i<p_i}(p_i-p'_i) \leq |\{i : p'_i < p_i\}| \cdot \max\{p_i\}$. Now since $\max_i\{p_i\} < (1+\frac{\epsilon}{100})\frac{1}{n}$ there are at least $(\epsilon/2)n$ such i. A uniformly random choice of $(6/\epsilon)\log(\delta^{-1})$ indexes will get one with probability at least $1-\delta/3$. □

Lemma 6.3.5. *When both events above occur,* $\mu'\restriction_{U\cup S}$ *is at least* $\frac{\epsilon^2}{600\log(\delta^{-1})}$*-far from* $\mu\restriction_{U\cup S}$ *over* $U\cup S$.

Proof. Note that $|S \cup U| = 2k = 2 \cdot (6/\epsilon)\log(\delta^{-1})$, and that the two events above mean that there are i and j in this set such that $p'_i \geq \frac{1+\epsilon/2}{1+\epsilon/100} p'_j$. Denoting the conditional probabilities $q_i = p_i/\mu(S \cup U)$ and $q'_i = p'_i/\mu'(S \cup U)$, we note that we obtain $q'_i \geq \frac{1+\epsilon/2}{1+\epsilon/100} q'_j$, while both q_i and q_j are bounded between $\frac{1-\epsilon/100}{1+\epsilon/100}\frac{1}{2k}$ and $\frac{1+\epsilon/100}{1-\epsilon/100}\frac{1}{2k}$. Therefore, either $q'_i > q_i + \frac{\epsilon}{40k}$ or $q'_j < q_j - \frac{\epsilon}{40k}$. Either way, $d_{TV}(\mu \restriction_{U \cup S}, \mu' \restriction_{U \cup S}) > \frac{\epsilon}{100k}$, which concludes the proof. □

This concludes the soundness proof, as the last step of the algorithm checks the closeness of $\mu' \restriction_{U \cup S}$ to $\mu \restriction_{U \cup S}$ with this approximation parameter. Thus we obtain:

Lemma 6.3.6. *Let μ be a known distribution over $[n]$. Then if $\|\mu - U_n\|_\infty < \frac{\epsilon}{100n}$ and $d_{TV}(\mu, \mu') > \epsilon$ then Algorithm 6.1 rejects with probability at least $1 - \delta$.*

Proof. Follows from a union bound for the events of Lemma 6.3.3 and Lemma 6.3.4, and the failure probability of the test invoked in the last step of the algorithm (due to Lemma 6.3.5). □

6.3.2 Testing identity to a known distribution

Recall that if we define $\log^{(0)}(n) = n$ and by induction $\log^{(k+1)}(n) = \log(\log^{(k)}(n))$, then the $\log^\star$ function is defined by $\log^\star(n) = \min\{k : \log^{(k)}(n) \leq 1\}$.

Theorem 6.6. *Testing identity with a known distribution can be done by an adaptive algorithm using $\mathrm{poly}(\log^\star n, \epsilon^{-1}, \log(\delta^{-1}))$ conditional samples.*

Proof. This follows from Algorithm 6.2 by Lemmas 6.3.11, 6.3.8 and 6.3.10 below. □

Let μ be the known distribution and μ' be the unknown distribution that is accessed by sampling. The following is an algorithm for testing identity to the known distribution μ over $[n]$. In the initial run we feed it $m = n$, but in the recursive runs it keeps track of m as the "original n".

Algorithm 6.2 Identity Test

Input: Known distribution μ, distance parameter ϵ, confidence parameter δ, initial universe size m, current universe size n.

Output: “ACCEPT” or “REJECT”.

```
1: if $n \leq \left(\frac{400\log(1/\epsilon)}{\epsilon}\log^\star m\right)^3$ then
2:     {Perform a brute-force test}
3:     Take $100\log(1/\delta)\epsilon^{-2}n^2\log n$ samples to write a distribution $\tilde{\mu}$ that is $\frac{\epsilon}{2}$-close to $\mu'$ (with probability $1-\delta$)
4:     if $d_{TV}(\tilde{\mu},\mu) \leq \frac{\epsilon}{2}$ then
5:         return ACCEPT
6:     else
7:         return REJECT
8: Let $\mathcal{M} = \{M_0, M_1, \ldots, M_k\} \leftarrow Bucket(\mu, [n], \frac{\epsilon}{200\log^\star m})$.
9: for every bucket $M_i \in \mathcal{M}$ do
10:    Test using the Near Uniformity Test (Theorem 6.5) whether $d_{TV}(\mu\restriction_{M_i}, \mu'\restriction_{M_i}) \geq \frac{\epsilon}{\log^* m}$ with confidence parameter $\frac{\delta\epsilon}{12\log^\star(m)\log(\delta^{-1})}$.
11:    if $d_{TV}(\mu\restriction_{M_i}, \mu'\restriction_{M_i}) \geq \frac{\epsilon}{\log^* m}$ then
12:        return REJECT.
13: Recursively test if $d_{TV}(\mu_{\langle\mathcal{M}\rangle}, \mu'_{\langle\mathcal{M}\rangle}) \leq \epsilon\left(1-\frac{1}{\log^\star m}\right)$ with confidence parameter $\frac{\delta}{3}$. If not then REJECT else ACCEPT.
```

First, we bound the number of recursion levels that can occur.

Lemma 6.3.7. *Algorithm 6.2 never enters more than $2\log^\star(n)$ recursion levels from the initial $n = m$ call.*

Proof. Note that in the first $2\log^\star(n)$ recursion levels, the distance parameter that is passed is still at least $\epsilon\left(1-\frac{1}{\log^\star n}\right)^{2\log^\star(n)} > \frac{\epsilon}{e^2}$, so we will prove the bound on the number of levels even if this is the distance parameter that is used in all but the first level. If $\log(n) \leq \left(\frac{400\log(1/\epsilon)}{\epsilon}\log^\star m\right)$ then after at most one recursion level the test goes to the brute force procedure in Step 3 and ends. Otherwise, note that the recursive call now receives $n' \leq \frac{400e^2\log(n)\log^\star(m)}{\epsilon} \leq \log^3(n)$, and that call itself will make a recursive call with universe size $n'' \leq \frac{1200e^2\log\log(n)\log^\star(m)}{\epsilon} \leq \log n$ (unless it already terminated for some other reason). This is sufficient for the bound. □

Lemma 6.3.8. *If $d_{TV}(\mu,\mu') = 0$ then Algorithm 6.2 accepts with probability at least $1-\delta$.*

Proof. The base case where $n \leq \left(\frac{400\log(1/\epsilon)}{\epsilon}\log^\star m\right)^3$ is clear. Otherwise, if $d_{TV}(\mu,\mu') = 0$ then for all buckets M_i we have $d_{TV}(\mu\restriction_{M_i}, \mu'\restriction_{M_i}) = 0$ and $d_{TV}(\mu_{\langle\mathcal{M}\rangle}, \mu'_{\langle\mathcal{M}\rangle}) = 0$. From Lemma 6.2.3 we know that $\|\mu\restriction_{M_i} - U\restriction_{M_i}\|_\infty \leq \frac{\epsilon}{200\log^\star m}\cdot\frac{1}{n} \leq \frac{\epsilon'}{100n}$, where ϵ' is the distance parameter fed to the Near Uniformity Tester, and hence the Near Uniformity tester (Theorem 6.5) is applicable and will accept with probability $1-\frac{\delta\epsilon}{12\log^\star(m)\log(\delta^{-1})}$. Taking the union bound over the number of samples taken and the probability of failure for the recursive call (recall that a recursive call adds a $\frac{1}{3}$ factor to δ) gives us the desired bound. □

For soundness we need the following lemma.

Lemma 6.3.9. *If $d_{TV}(\mu,\mu') \geq \epsilon$ then for any t at least one of the following two will happen:*

1. $\sum_{\{i:d_{TV}(\mu\restriction_{M_i},\mu'\restriction_{M_i})\geq\epsilon/t\}}\mu(M_i)\geq\epsilon/t$
2. $d_{TV}(\mu_{\langle\mathcal{M}\rangle},\mu'_{\langle\mathcal{M}\rangle})\geq\epsilon(1-1/t)$

Proof. Recall Lemma 6.2.4:

$$d_{TV}(\mu,\mu')\leq\sum_{0\leq i\leq k}\mu(M_i)\cdot d_{TV}(\mu\restriction_{M_i},\mu'\restriction_{M_i})+d_{TV}(\mu_{\langle\mathcal{M}\rangle},\mu'_{\langle\mathcal{M}\rangle})$$

Thus if $d_{TV}(\mu_{\langle\mathcal{M}\rangle},\mu'_{\langle\mathcal{M}\rangle})<\epsilon(1-1/t)$ and $\sum_{\{i:d_{TV}(\mu\restriction_{M_i},\mu'\restriction_{M_i})\geq\epsilon/t\}}\mu(M_i)<\epsilon/t$ then since always $d_{TV}(\mu\restriction_{M_i},\mu'\restriction_{M_i})\leq 1$ we sum up to obtain $d_{TV}(\mu,\mu')<\epsilon$, a contradiction. □

Lemma 6.3.10. *If $d_{TV}(\mu,\mu')>\epsilon$ then Algorithm 6.1 rejects with probability at least $1-\delta$.*

Proof. The base case of $n\leq\left(\frac{400\log(1/\epsilon)}{\epsilon}\log^\star m\right)^3$ is clear. Refer now to Lemma 6.3.9, taking $t=\log^\star m$. Assume that we are in the first case of the lemma, that is the case that $\sum_{\{i:d_{TV}(\mu\restriction_{M_i},\mu'\restriction_{M_i})\geq\epsilon/t\}}\mu(M_i)\geq\epsilon/t$. therefore, the probability of sampling an index for which the test in Line 10 should reject is at least $\frac{\epsilon}{2\log^\star m}$. This implies that the probability that one of the sampled elements is such is at least $\delta/3$, and since the probability that all calls to the Near Uniformity Test fail is at most $\delta/3$ as well, we accept with probability at most $2\delta/3$.

Now assuming that we are in the second case of Lemma 6.3.9, by the induction hypothesis we reject with probability at least $\delta/3$. Thus the overall confidence parameter is at most δ. □

Lemma 6.3.11. *The sample complexity of Algorithm 6.2 is* $\mathrm{poly}(\log^\star n,\epsilon^{-1},\log(\delta^{-1}))$.

Proof. If $n\leq\left(\frac{400\log(1/\epsilon)}{\epsilon}\log^\star m\right)^3$ then it is polynomial in ϵ and $\log^\star m$, and so is the result of substituting it in the number of queries of the brute force check of Step 1, which means that $q_b(\epsilon,\delta,n)=100\log(1/\delta)\epsilon^{-2}n^2\log n$. For analyzing the sample complexity when the above does not hold for $m=n$, let $q(\epsilon,\delta,n,m)$ denote the sample complexity of the algorithm. By the algorithm's definition, we have the following formula, where q_u is the sample complexity of the Near Uniformity Tester:

$$q(\epsilon,\delta,n,m)\leq 4\epsilon^{-1}\log^\star(m)\log(\delta^{-1})\left(1+q_u\left(\frac{\epsilon}{\log^* m},\frac{\delta\epsilon}{12\log^\star(m)\log(\delta^{-1})},n\right)\right)$$

$$+q\left(\epsilon\left(1-\frac{1}{\log^\star m}\right),\frac{\delta}{3},\frac{400\log(n)\log^\star(m)}{\epsilon},m\right)$$

According to Lemma 6.3.7, after at most $2\log^\star n$ recursion levels from the initial $n=m$, the right hand side is now within the realm of the brute force check, and we get a summand bounded by $q_b(\epsilon/e^2,\delta\cdot 3^{-2\log^\star n},\left(\frac{400\log(1/\epsilon)}{\epsilon}\log^\star n\right)^3)=\mathrm{poly}(\log^\star n,\epsilon^{-1},\log(\delta^{-1}))$. Therefore:

$$q(\epsilon,\delta,n,m)\leq 8\epsilon^{-1}(\log^\star m)^2\log(\delta^{-1})\left(1+q_u\left(\frac{\epsilon}{2e^2\log^\star m},\frac{\epsilon\cdot\delta\cdot 3^{-2\log^\star m}}{40e^2(\log^\star m)^2\log(\delta^{-1})},n\right)\right)$$

$$+\mathrm{poly}(\log^\star n,\epsilon^{-1},\log(\delta^{-1}))$$

Since by Lemma 6.3.1, the Near Uniformity Tester has sample complexity polynomial in the distance parameter and polylogarithmic in the confidence parameter, we obtain the statement of the lemma. □

6.4 Non-adaptive testing for uniformity and identity

Recall that a non-adaptive tester must be able to produce all the conditioned upon sets in advance. In this section we show that these weaker testers can still beat testers without conditional sampling.

6.4.1 Testing uniformity

Theorem 6.7. *Testing uniformity can be done using* $\mathrm{poly}(\log n, \epsilon^{-1}, \log(\delta^{-1}))$ *non-adaptive conditional samples.*

Again, we will actually prove the following stronger statement:

Theorem 6.8 (Nonadaptive Near Uniformity Tester)**.** *Let* μ *be a known distribution over* $[n]$*. If* $\|\mu - U_n\|_\infty < \epsilon/8n$ *then identity with* μ *can be tested using* $\mathrm{poly}(\log n, \epsilon^{-1}, \log(\delta^{-1}))$ *conditional samples by a non-adaptive algorithm.*

Proof. For $\delta = 1/3$, this follows from Algorithm 6.3 by Lemmas 6.4.2, 6.4.1 and 6.4.5 below. For general δ we use a standard amplification technique: We repeat the algorithm $\Theta(\log(\delta^{-1}))$ times (with independent probabilities) and take the majority vote. This obviously incurs a multiplicative factor of $\Theta(\log(\delta^{-1}))$ in the sample complexity. □

Let μ' denote the unknown distribution, to which the algorithm has nonadaptive conditional sample access.

Algorithm 6.3 Non-Adaptive Near Uniformity Tester

Input: Known distribution μ, distance parameter ϵ, confidence parameter δ and universe size n.
Output: "ACCEPT" or "REJECT".

```
1: for $\lceil \log(2000\epsilon^{-6}\log^5(n)) \rceil \le j \le \lceil \log(n) \rceil$ do
2:     Set $U_j$ to be a uniformly random set of $\min\{n, 2^j\}$ indices.
3: for $U_j$ do
4:     Perform $64\epsilon^{-2}\log^2(n)$ conditional samples.
5:     if the same index was drawn twice then
6:         return REJECT
7: Uniformly pick a random set $U$ of $9000\epsilon^{-6}\log^5(n)$ elements, and invoke the Identity Tester of
   Theorem 6.1 to test whether $\mu' \restriction_U = \mu \restriction_U$ or $d_{TV}(\mu' \restriction_U, \mu \restriction_U) > \frac{\epsilon}{24|U|}$ with success probability
   $\frac{19}{20}$. In the latter case REJECT.
8: return ACCEPT {unless any of the above testers rejected}
```

Lemma 6.4.1. *If* $d_{TV}(\mu, \mu') = 0$ *then Algorithm 6.1 accepts with probability at least* $2/3$.

Proof. Since $\|\mu - U_n\|_\infty < \epsilon/8n$, the probability that an element will be drawn twice in the jth iteration of Line 4 is at most $\binom{64\epsilon^{-2}\log^2(n)}{2} \cdot \left(\frac{1+\epsilon/8}{1-\epsilon/8}\right)^2 \cdot 2^{-2j}$. Summation over all values of j gives us less than 1/9.

Since $\mu = \mu'$, $\mu' \restriction_U = \mu \restriction_U$ for any $U \subseteq [n]$, and the probability that Line 7 rejects is at most 1/9. This obtains the confidence parameter in the lemma. □

The following is immediate from the algorithm statement and Theorem 6.1:

Lemma 6.4.2. *The sample complexity of Algorithm 6.2 is* $\mathrm{poly}(\log n, \epsilon^{-1})$.

Proof. This follows from the number of samples used in Lines 4 and 7 and the fact that Line 4 is iterated at most $\log n$ times. $\square$

In the following we assume that $d_{TV}(\mu, \mu') > \epsilon$.

Let $M_0, M_1, \ldots, M_k$ be the bucketing of μ and $M'_0, M'_1, \ldots, M'_k$ the bucketing of μ', both with $\epsilon/3$. Denote the individual probabilities by $p_1, \ldots, p_n$ and $p'_1, \ldots, p'_n$ respectively.

Lemma 6.4.3. *Assuming $d_{TV}(\mu, \mu') \geq \epsilon$, then $|M'_0 \cup M'_1| \geq \epsilon n$ and there exists $2 < j \leq k$ such that $|M'_j| \geq \frac{\epsilon^2 n}{96(1+\epsilon/3)^j \log n}$.*

Proof. Note that $[n] = M_0 \cup M_1$ by our requirement from μ. Now following Lemma 6.3.4, $\sum_{p'_i < p_i}(p_i - p'_i) \leq |\{i : p'_i < p_i\}| \cdot \max\{p_i\}$. Now since $\max_i\{p_i\} < (1 + \epsilon/8)\frac{1}{n}$ there are at least $(\epsilon/2)n$ such i.

For the second part we will adapt the proof of Lemma 6.3.3. Clearly we have the inequality $\sum_{p_i < p'_i < p_i + 11\epsilon/12n}(p'_i - p_i) < \frac{11}{12}\epsilon$. Therefore:

$$\sum_{p'_i \geq p_i + 11\epsilon/12n} p'_i > \sum_{p'_i \geq p_i + 11\epsilon/12n} (p'_i - p_i) = \sum_{p'_i > p_i} (p'_i - p_i) - \sum_{p_i < p'_i < p_i + 11\epsilon/12n} (p'_i - p_i) > \frac{1}{12}\epsilon$$

Since $p_i \geq \frac{1-\epsilon/8}{n}$, we know that the p'_i in the left hand side have (assuming $\epsilon < 1/10$)

$$p'_i \geq \frac{1-\epsilon/8}{n} + \frac{11\epsilon}{12n} = \frac{1 + 19\epsilon/24}{n} \geq \frac{(1+\epsilon/3)^2}{n}$$

and therefore all these p'_is are in buckets M'_j for $2 < j \leq k$.

Since $k = \frac{\log n}{\log(1+\epsilon/3)}$, there exists some $2 < j \leq k$ such that $\mu'(M'_j) \geq \frac{\epsilon \log(1+\epsilon/3)}{12 \log n}$. By the definition of the buckets this gives $|M'_j| \geq \frac{\epsilon \log(1+\epsilon/3)}{12 \log n} \cdot \frac{n}{(1+\epsilon/3)^j} > \frac{\epsilon^2 n}{96(1+\epsilon/3)^j \log n}$. $\square$

Lemma 6.4.4. *Given a set B of size l, a set U of $\min\{n, \frac{3n}{l}\}$ indices chosen uniformly at random will with probability more than $\frac{19}{20}$ contain a member of B.*

Proof. The probability is lower bounded by the probability for $3n/l$ indexes chosen uniformly and independently with repetitions from $[n]$ to intersect B, which is $1 - (1 - l/n)^{\frac{3n}{l}} \geq \frac{19}{20}$. $\square$

Lemma 6.4.5. *Let μ be a known distribution over $[n]$. If $\|\mu - U_n\|_\infty < \epsilon/8n$ and $d_{TV}(\mu, \mu') > \epsilon$ then Algorithm 6.1 rejects with probability at least $2/3$.*

Proof. We partition into cases according to the j guaranteed by Lemma 6.4.3.

If $(1 + \frac{\epsilon}{3})^j \leq 40\epsilon^{-4}\log^4 n$, then $|M'_j| \geq \frac{\epsilon^6}{3000 \log^5 n} n$, so by Lemma 6.4.4 with probability $\frac{19}{20}$ the set U in Line 7 will contain a member h of M'_j. Note that $j > 2$ and therefore $\mu'(h) \geq \frac{(1+\epsilon/3)^2}{n}$. By the first part of Lemma 6.4.3 with probability $\frac{19}{20}$ (actually much more than that) we will also sample an element $l \in M'_0 \cup M'_1$. Thus we have $\mu'(h) \geq (1 + \epsilon/3)\mu'(l)$, and also $\mu' \restriction_U (h) \geq (1 + \epsilon/3)\mu' \restriction_U (l)$, while both $\mu \restriction_U (h)$ and $\mu \restriction_U (l)$ are restricted between $\frac{1-\epsilon/8}{1+\epsilon/8}\frac{1}{|U|}$ and $\frac{1+\epsilon/8}{1-\epsilon/8}\frac{1}{|U|}$. Therefore, either $\mu' \restriction_U (h) > \mu \restriction_U (h) + \frac{\epsilon}{12|U|}$ or $\mu' \restriction_U (l) < \mu \restriction_U (l) - \frac{\epsilon}{12|U|}$. Either way $d_{TV}(\mu' \restriction_U, \mu \restriction_U) > \frac{\epsilon}{24|U|}$, which will be identified by the tester of Theorem 6.1 with probability $\frac{19}{20}$. Thus in total we get a rejection probability greater than $\frac{7}{9}$.

Otherwise, let i be such that the value 2^i is between $\min\{n, 300\epsilon^{-2}\log n(1+\frac{\epsilon}{3})^j\}$ and $2\min\{n, 300\epsilon^{-2}\log n(1+\frac{\epsilon}{3})^j\}$ (recall the lower bound on $(1+\frac{\epsilon}{3})^j$). In that case the U_i in Line 4 will with probability at least $\frac{19}{20}$ contain a member a of M'_j. Additionally, the expected value of $\mu'(U_i)$ is $\min\{1, \frac{2^i}{n}\} \le \min\{1, \frac{600}{n}\epsilon^{-2}(1+\frac{\epsilon}{3})^j \log n\}$, thus by Markov's inequality, with probability at least $\frac{8}{9}$ we will have $\mu'(U_i) \le \min\{1, \frac{6000}{n}\epsilon^{-2}(1+\frac{\epsilon}{3})^j \log n\}$. Therefore, $\mu' \restriction_{U_i} (a) \ge \frac{\epsilon^2}{6000(1+\epsilon/3)\log n}$. Thus the expected number of times a is sampled is at least $\frac{\log n}{125}$ and therefore by Lemma 2.0.1 with probability $1-2\exp(-\frac{\log n}{250})$ we will sample a at least twice. Thus in total we get a rejection probability greater than $\frac{7}{9}$ for $n > 2^{253}$ (this lower bound can be reduced for the price of a higher degree polynomial dependence on $\log n$). □

6.4.2 Testing identity to a known distribution

Theorem 6.9. *Identity to a known distribution can be tested using* $\text{poly}(\log n, \epsilon^{-1}, \log(\delta^{-1}))$ *non-adaptive conditional samples.*

Proof. This follows from Algorithm 6.4 by Lemmas 6.4.7, 6.4.6 and 6.4.8 below. □

Let μ be the known distribution and μ' be the unknown distribution that is accessed by sampling. The following is an algorithm for testing identity with the known distribution μ over $[n]$:

Algorithm 6.4 Non-Adaptive Identity Test

Input: Known distribution μ, distance parameter ϵ, confidence parameter δ and universe size n.
Output: "ACCEPT" or "REJECT".

```
1: Let $\mathcal{M} = \{M_0, M_1, \ldots, M_k\} \leftarrow Bucket(\mu, [n], \frac{\epsilon}{8})$.
2: for every bucket $M_1, \ldots, M_k$ do
3:     Test using the Nonadaptive Near Uniformity Test (Theorem 6.8) to check whether $\|\mu \restriction_{M_j} - \mu' \restriction_{M_j}\|_1 \ge \epsilon/2$ with confidence parameter $\frac{\delta \log(1+\epsilon/8)}{2\log n}$, rejecting immediately if any test rejects.
4: Invoke the Identity Tester of Theorem 6.1 to test if $\|\mu_{\langle\mathcal{M}\rangle} - \mu'_{\langle\mathcal{M}\rangle}\|_1 \le \epsilon/2$ with confidence parameter $\delta/2$, answering as the test does.
```

Lemma 6.4.6. *If* $d_{TV}(\mu, \mu') = 0$ *then Algorithm 6.4 accepts with probability at least* $1-\delta$.

Proof. In this case, for all buckets $\|\mu \restriction_{M_j} - \mu' \restriction_{M_j}\|_1 = 0$ and $\|\mu_{\langle\mathcal{M}\rangle} - \mu'_{\langle\mathcal{M}\rangle}\|_1 = 0$, and thus by the union bound we obtain the statement. □

Lemma 6.4.7. *The sample complexity of Algorithm 6.2 is* $\text{poly}(\log n, \epsilon^{-1}, \log(\delta^{-1}))$.

Proof. We invoke the Nonadaptive Near Uniformity Test $\frac{\log n}{\log(1+\epsilon/8)}$ times, and invoke the Closeness Tester with a distribution of support size $\frac{\log n}{\log(1+\epsilon/8)}$. Therefore by Lemma 6.4.2 and Theorem 6.1 we obtain the bound in the statement. □

Lemma 6.4.8. *If* $d_{TV}(\mu, \mu') > \epsilon$, *then Algorithm 6.4 rejects with probability at least* $1-\delta$.

Proof. Assume that the test accepted. If no error was made, then by Lemma 6.2.4 we have that $d_{TV}(\mu, \mu') \le \epsilon$. By the union bound the probability of error is at most δ. □

6.5 Explicit persistent samplers

We exhibit here the strength of the conditional sampling oracle, using it to implement explicit persistent samplers as defined below.

Definition 6.10. Given a distribution over distributions $\mathcal{M}$, a (δ, s)-*explicit persistent sampler* is an algorithm that can be run up to s times (and during each run may store information to be used in subsequent runs), that in every run returns a pair (i, η). It must satisfy that with probability at least $1 - \delta$, the i's for all s runs are independent samples of a single distribution $\tilde{\mu}$ that in itself was drawn according to the distribution over distributions $\mathcal{M}$, and every output pair (i, η) satisfies $\eta = \tilde{\mu}(i)$.

The goal of this section is to construct, for every distribution μ, an explicit persistent sampler for a distribution over distributions that are all close to μ, which uses a conditional sampling oracle for μ.

Note that although the definition does not require it, the explicit persistent samplers we construct will also be able to answer oracle queries of the form "what is the probability of i?".

In all the following we assume that n is a power of 2, as otherwise we can "pad" the probability space with additional zero-probability members.

6.5.1 Ratio trees and reconstituted distributions

The main driving force in our algorithm for constructing an explicit sampler is a way to estimate the ratio between the distribution weight of two disjoint sets. To make it into a weight oracle for a value $i \in [n]$, we will use successive partitions of $[n]$, through a fixed binary tree. Remember that here n is assumed to be a power of 2.

We first define how to "reconstruct" a distribution from a tree with ratios, and afterward show how to put the ratios there.

Definition 6.11. Let T be a (full) balanced binary tree with n leaves labeled by $[n]$. Let U be the set of non-leaf nodes of the tree, and assume that we have a function $\alpha : U \to [0, 1]$. For $u \in U$ denote by $L(u)$ the set of leaves that are descendants of the left child of u, and by $R(u)$ the leaves that are descendants of the right child of u.

The *reconstituted distribution* according to α is the distribution $\tilde{\mu}$ that is calculated for every $i \in [n]$ as follows:

- Let $u_1, \ldots, u_{\log(n)+1}$ be the root to leaf path for i (so in particular $u_{\log(n)+1} = i$).
- For every $1 \le j \le \log n$, set $p_j = \alpha(u_j)$ if i is a descendant of the left child of u_j (that is if $i \in L(u_j)$), and otherwise set $p_j = 1 - \alpha(u_j)$.
- Set $\tilde{\mu}(i) = \prod_{j=1}^{\log n} p_j$.

For intuition, note the following trivial observation.

Observation 6.12. If for a distribution μ we set $\alpha(u) = \frac{\mu(L(u))}{\mu(L(u))+\mu(R(u))}$, using an arbitrary value (say $\frac{1}{2}$) for the case where $\mu(L(u)) + \mu(R(u)) = 0$, then the reconstituted distribution $\tilde{\mu}$ is identical to μ.

However, if we only have conditional oracle access to μ then we cannot know the values $\frac{\mu(L(u))}{\mu(L(u))+\mu(R(u))}$. The best we can do the the following.

Definition 6.13. An (ϵ,δ)*-ratio estimator* for T and a distribution μ is an algorithm A that given a non-leaf vertex $u \in U$ outputs a number r, such that with probability $1-\delta$ we have that $\frac{\mu(L(v))}{\mu(L(v))+\mu(R(v))} - \epsilon \le r \le \frac{\mu(L(v))}{\mu(L(v))+\mu(R(v))} + \epsilon$.

The algorithm is given conditional sample access to a distribution μ.

Algorithm 6.5 Ratio Estimator

Input: A balanced binary tree T with n leaves, a non-leaf vertex $u \in U$ distance parameter ϵ and confidence parameter δ.

Output: Real number.

```
1: Sample $t = 2\epsilon^{-2}\log(\delta^{-1})$ elements according to $\mu\restriction_{L(u)\cup R(u)}$, and let $s$ be the number of samples that are in $L(u)$.
2: return the ratio $\frac{s}{t}$ of the samples that are in $L(u)$ to the total number of samples.
```

Lemma 6.5.1. *For any ϵ,δ Algorithm 6.5 is an (ϵ,δ)-ratio estimator for T and μ which uses $t = 2\epsilon^{-2}\log(\delta^{-1})$ non-adaptive conditional samples from μ.*

Proof. The number of samples used is immediate. Let us now proceed to show that this is indeed an (ϵ,δ)-ratio estimator. The expected value of $\frac{s}{t}$ is $\frac{\mu(L(u))}{\mu(L(u))+\mu(R(u))}$.

By Chernoff's inequality, the probability that $\frac{s}{t}$ deviates from its expected value by an additive term of more than ϵ is at most $2\exp(-2\epsilon^2 \cdot t)$. By our choice of t we obtain the statement. □

If we could "populate" the entire tree T (through the function α) by values that do not deviate by much from the corresponding ratios, then we would be able to create an estimate for μ that is good for most values.

Definition 6.14. The function $\alpha : U \to [0,1]$ is called *ϵ-fine* if $|\alpha(u) - \frac{\mu(L(u))}{\mu(L(u))+\mu(R(u))}| \le (\frac{\epsilon}{2\log(n)})^2$ for every $u \in U$.

We call a distribution $\tilde{\mu}$ *ϵ-fine* if there exists a set B such that $\mu(B) \le \epsilon$, and additionally $\tilde{\mu}(i) = (1 \pm \epsilon)\mu(i)$ for every $i \in [n] \setminus B$.

Lemma 6.5.2. *If α is ϵ-fine then the reconstituted distribution $\tilde{\mu}$ is ϵ-fine.*

Proof. To define the set B, for every i consider the $p_1,\dots,p_{\log n}$ that are set as per Definition 6.11, and set $i \in B$ if and only if there exist some p_j that is smaller than $\frac{\epsilon}{2\log(n)}$. Next, denote by $q_1,\dots,q_k$ the "intended" values, that is $q_j = \frac{\mu(L(u_j))}{\mu(L(u_j))+\mu(R(u_j))}$ if $i \in L(u_j)$ and $q_j = \frac{\mu(R(u_j))}{\mu(L(u_j))+\mu(R(u_j))}$ otherwise. Noting that p_j does not deviates from q_j by more than $(\frac{\epsilon}{2\log(n)})^2$, an induction over $\log n$ (the height of T) gives that $1-\mu(B)$ is at least $(1-\frac{\epsilon}{\log n})^{\log n} > 1-\epsilon$.

For $i \in [n] \setminus B$, we note that in this case $p_j = (1 \pm \frac{\epsilon}{2\log n})q_j$, and hence we deduce that $\tilde{\mu}(i) = \prod_{j=1}^{\log n} p_j = (1 \pm \frac{\epsilon}{2\log n})^{\log n}\prod_{j=1}^{\log n} q_j = (1\pm\epsilon)\mu(i)$. □

We should note here that it is not hard to prove that an ϵ-fine distribution $\tilde{\mu}$ is of distance not more than 4ϵ from the original μ. However, we will in fact refer to yet another distribution which will be easier to estimate, so we will show closeness to it instead.

Definition 6.15. Given an ϵ-fine distribution $\tilde{\mu}$ and its respective set B, its *ϵ-trimmed distribution* $\overline{\mu}$ is a distribution over $[n] \cup \{0\}$ defined by the following.

- For $i \in B \cup \{i : \tilde{\mu}(i) < \frac{\epsilon}{n}\}$ we set $\overline{\mu}(i) = 0$. For such i we also set $j_i = 0$.
- For all other $i \in [n]$ we set j_i to be the largest integer for which $\frac{(1+\epsilon)^{j_i-1}}{n}\epsilon \leq \tilde{\mu}(i)$, and set $\overline{\mu}(i) = \frac{(1+\epsilon)^{j_i-1}}{n}\epsilon$.
- Finally set $\overline{\mu}(0) = 1 - \sum_{i=1}^{n} \overline{\mu}(i)$; note that $\overline{\mu}(i) \leq \tilde{\mu}(i)$ for all $1 \leq i \leq n$ and hence $\overline{\mu}(0) \geq 0$.

The *ϵ-renormalized distribution* $\hat{\mu}$ over $[n]$ is just the conditioning $\overline{\mu} \upharpoonright_{[n]}$.

It will be important later to note that the renormalized distribution is in fact (a permutation of) the tentative distribution according to $m_0, \ldots, m_k$, where for $0 \leq j \leq k$ we set $m_j = |\{i : j_i = j\}|$, as per Definition 6.6 below.

Lemma 6.5.3. *The renormalized distribution $\hat{\mu}$ corresponding to an ϵ-fine distribution $\tilde{\mu}$ is 5ϵ-close to μ.*

Proof. First we consider the trimmed distribution $\overline{\mu}$, and its distance from μ (when we extend it by setting $\mu(0) = 0$). Recalling that this variation distance is equal to $\sum_{\{i:\overline{\mu}(i)<\mu(i)\}}(\mu(i) - \overline{\mu}(i))$, we partition the set of relevant i's into two subsets.

- For those i that are in B (for which $\overline{\mu}(i) = 0$), the total difference is $\mu(B) \leq \epsilon$.
- For $i \notin B$ where $\overline{\mu}(i) < \mu(i)$ and $\tilde{\mu} \geq \frac{\epsilon}{n}$, note that $\overline{\mu}(i) \geq \frac{1}{1+\epsilon}\tilde{\mu}(i) \geq \frac{1-\epsilon}{1+\epsilon}\mu(i) > (1-3\epsilon)\mu(i)$. This means that the sum over differences for all such i is bounded by 3ϵ.
- For $i \notin B$ where $\overline{\mu}(i) < \mu(i)$ and $\tilde{\mu} < \frac{\epsilon}{n}$, the total difference is no more than ϵ.
- We never have $\overline{\mu}(0) < \mu(0)$.

Thus the distance between $\overline{\mu}$ and μ is not more than 4ϵ. As for $\hat{\mu}$, the sum of differences over i for which $\hat{\mu}(i) < \mu(i)$ is only made smaller (the conditioning only increases the probability for every $i > 0$), and so the 4ϵ bound remains. □

6.5.2 Distribution samplers and learning

To construct an explicit sampler we need to not only sample from the distribution μ, but to be able to "report" $\mu(i)$ for every i thus sampled. This we cannot do, but it turns out that we can sample from a close distribution $\tilde{\mu}$ while reporting $\tilde{\mu}(i)$. In fact we will sample from a distribution that in itself will be drawn from the following distribution over distributions.

Definition 6.16. The (ϵ, δ)-*condensation* of μ is the distribution over ϵ-fine distributions (with respect to μ) that is defined by the following process.

- Let T be a (full) balanced binary tree whose leaves are labeled by $[n]$, and U be its set of internal nodes.
- For every $u \in U$, let $\alpha(u)$ be the (randomized) result of running the corresponding $((\frac{\epsilon}{2\log(n)})^2, \delta)$-Ratio Estimator (Algorithm 6.5), when conditioned on this result indeed being of distance not more than $(\frac{\epsilon}{2\log(n)})^2$ away from $\frac{\mu(L(u_j))}{\mu(L(u_j))+\mu(R(u_j))}$. This is done independently for every u.

- The drawn distribution $\tilde{\mu}$ is the reconstituted distribution according to T and α

The algorithm that we define next is an explicit persistent sampler: It is explicit in that it relays information about $\tilde{\mu}(i)$ along with i, and persistent in that it simulates (with high probability) a sequence of s independent samples from the same $\tilde{\mu}$. It operates given conditional sample access to a distribution μ.

Algorithm 6.6 Persistent Sampler

Input: Repetition parameter s, universe size n, distance parameter ϵ and confidence parameter δ.

Output: An index $i \in [n]$ and a real number.

```
1: On the initial run, set $T$ to be a full balanced binary tree with $n$ leaves labeled by $[n]$. Let $v$
   denote the root vertex and $U$ denote the set of non-leaf vertices. $\alpha$ is initially unset.
2: On all runs, set $u_1 = v$.
3: for $l = 1, \ldots, \log n$ do
4:     If $\alpha(u_l)$ is not set yet, set it to the result of the $((\frac{\epsilon}{2\log(n)})^2, \frac{\delta}{s\log n})$-Ratio Estimator
       (Algorithm 6.5); run it independently of prior runs.
5:     Independently of any prior choices, and without sampling from $\mu$, with probability $\alpha(u_l)$
       set $u_{l+1}$ to be the left child of $u_l$ and $p_l = \alpha(u_l)$, and with probability $1 - \alpha(u_l)$ set $u_{l+1}$
       to be the right child of $u_l$ and $p_l = 1 - \alpha(u_l)$.
6: Set $i$ to be the label of the leaf $u_{\log n+1}$ and $\eta = \prod_{l=1}^{\log n} p_l$.
7: return $(i, \eta)$
```

Lemma 6.5.4. *For any ϵ, δ and s, Algorithm 6.6 is a (δ, s)-explicit persistent sampler for the $(\epsilon, \frac{\delta}{s\log n})$-condensation of μ. It uses a total of $2^5 \cdot \epsilon^{-4} \log^5 n \cdot \log(s\delta^{-1}\log n)$ many adaptive conditional samples from μ to output a sample.*

Proof. The calculation of the number of samples is straightforward (but note that these are adaptive now). During s runs, by the union bound with probability at least $1-\delta$ all of the calls to the $((\frac{\epsilon}{2\log(n)})^2, \frac{\delta}{s\log n})$-Ratio Estimator produced results that are not more than $((\frac{\epsilon}{2\log(n)})^2$-away from the actual rations.

Conditioned on the above event, the algorithm acts the same as the algorithm that first chooses for every $u \in U$ the value $\alpha(u)$ according to a run of the $((\frac{\epsilon}{2\log(n)})^2, \frac{\delta}{s\log n})$-Ratio Estimator conditioned on it being successful, and only then traverses the tree T for every required sample. The latter algorithm is identical to picking a distribution $\tilde{\mu}$ according to the $(\epsilon, \frac{\delta}{s\log n})$-condensation of μ, and then (explicitly) sampling from it. □

6.6 Testing any label-invariant property

We show here the following "universal testing" theorem for label-invariant properties.

Theorem 6.1. *Every label-invariant property of distributions can be tested adaptively using at most* $\mathrm{poly}(\log n, \epsilon^{-1}, \log(\delta^{-1}))$ *conditional samples.*

It is in fact a direct corollary of the following learning result.

Theorem 6.2. *There exist an algorithm that uses* $\mathrm{poly}(\log n, \epsilon^{-1}, \log(\delta^{-1}))$ *adaptive conditional samples to output a distribution* $\tilde{\mu}$ *over* $[n]$*, so that with probability at least* $1-\delta$ *some permutation of* $\tilde{\mu}$ *will be* ϵ*-close to* μ*.*

Proof. The required algorithm is Algorithm 6.8 below, by Lemma 6.6.2. □

To derive Theorem 6.1, use Theorem 6.2 to obtain a distribution $\tilde{\mu}$ that is $\epsilon/2$-close to a permutation of μ, and then accept μ if and only if $\tilde{\mu}$ is $\epsilon/2$-close to the tested property.

In a similar manner, one can also derive the following corollaries:

Corollary 6.3. There exist an algorithm that uses $\mathrm{poly}(\log n, \epsilon^{-1}, \log(\delta^{-1}))$ adaptive conditional samples to test whether two unknown distributions are identical up to relabeling.

Corollary 6.4. For every label-invariant property P, there exist an algorithm with adaptive sample complexity $\mathrm{poly}(\log n, \epsilon^{-1}, \log(\delta^{-1}))$ that accepts any distribution $\epsilon/2$-close to P with probability at least $1-\delta$ and rejects any distribution ϵ-far from P with probability at least $1-\delta$.

Remark. The above can be proved to be true for every η, ϵ such that $\eta < \epsilon$. That is, there exists an algorithm that accepts any distribution η-close to P and with probability at least $1-\delta$ and rejects any distribution ϵ-far from P with probability at least $1-\delta$.

The main idea of the proof of Theorem 6.2 is to use a bucketing, and try to approximate the number of members of every bucket, which allows us to construct an approximate distribution. However, there are some roadblocks, the foremost being the fact that we cannot really query the value $\mu(i)$. Instead we will use an explicit persistent sampler as introduced in Section 6.5.

6.6.1 Bucketing and approximations

We need a bucketing that also goes into smaller probabilities than those needed for the other sections.

Definition 6.5. Given an explicit distribution μ over $[n]$, *Bucket*$'(\mu, [n], \epsilon)$ is a procedure that generates a partition $\{M_0, M_1, \ldots, M_k\}$ of the domain $[n]$, where $k = \frac{\log n \log(\epsilon^{-1})}{\log^2(1+\epsilon)}$. This partition satisfies the following conditions:

- $M_0 = \{j \in [n] \mid \mu(j) < \frac{\epsilon}{n}\}$;
- for all $i \in [k]$, $M_i = \left\{j \in [n] \mid \frac{(1+\epsilon)^{i-1}}{n}\epsilon \le \mu(j) < \frac{(1+\epsilon)^i}{n}\epsilon\right\}$.

In the rest of this section, bucketing will always refer to this version. Also, from here on we fix ϵ and $k = \frac{\log n \log(\epsilon^{-1})}{\log^2(1+\epsilon)}$ as above (as well as mostly ignore floor and ceiling signs). We also assume that ϵ is small enough, say smaller than $\frac{1}{100}$.

Suppose that we have $m_0, \ldots, m_k$, where $m_i = |M_i|$ is the size of the i'th set in the bucketing of a distribution μ. Then we can use these to construct a distribution that is guaranteed to be close to some permutation of μ.

Definition 6.6. Given $m_0, \ldots, m_k$ for which $\sum_{j=0}^{k} m_j = n$ and ϵ, the *tentative distribution* over $[n]$ is the one constructed according to the following.

- Set $r_1, \ldots, r_n$ so that $|\{i : r_i = 0\}| = m_0$ and $|\{i : r_i = \frac{(1+\epsilon)^{j-1}}{n}\epsilon\}| = m_j$ for every $1 \le j \le k$ (the order of $r_1, \ldots, r_n$ is arbitrary).

- Set a distribution $\tilde{\mu}$ over $[n]$ by setting $\mu(i)$ equal to $r_i / \sum_{j=1}^{n} r_i$.

To gain some intuition, note the following.

Observation 6.7. If $M_0, \ldots, M_k$ is the bucketing of μ and $\tilde{\mu}$ is the tentative distribution according to $m_0 = |M_0|, \ldots, m_k = |M_k|$, then $\tilde{\mu}$ is 2ϵ-close to some permutation of μ.

Proof. We assume that we have already permuted $\tilde{\mu}$ so that each $\tilde{\mu}(i)$ refers to an r_i set according to the bucket M_j satisfying $i \in M_j$ (such a permutation is possible because here we used the actual sizes of the buckets).

We recall that the distance is in particular equal to $\sum_{\{i:\tilde{\mu}(i)<\mu(i)\}}(\mu(i) - \tilde{\mu}(i))$. Referring to the r_i of the definition above, we note that in this case $\sum_{i=0}^{n} r_i \le \sum_{i=0}^{n} \mu(i) = 1$ and hence $\tilde{\mu}(i) \ge r_i$. For $i \notin M_0$, this means that $\tilde{\mu}(i) \ge (1-\epsilon)\mu(i)$. For the rest we just note that $\sum_{i \in M_0} \mu(i) \le \epsilon$. Together we get the required bound. □

The above observation essentially states that it is enough to find the numbers $m_0, \ldots, m_k$ associated with μ. However, the best we can hope for is to somehow estimate the size, or total probability, of every bucket. The following shows that this is in fact sufficient.

Definition 6.8. Given $\alpha_0, \ldots, \alpha_k$ for which $\sum_{j=0}^{k} \alpha_j = 1$, the *bucketization* thereof is the sequence of integers $\hat{m}_0, \ldots, \hat{m}_k$ defined by the following.

- For any $1 \le j \le k$ let $\hat{m}_j$ be the integer closest to $n\alpha_k$ (where an "exact half" is arbitrarily rounded down).

- If $\sum_{j=1}^{k} \hat{m}_j > n$, then decrease the $\hat{m}_j$ until they sum up to n, each time picking j to be the smallest index for which $\hat{m}_j > 0$ and decreasing that quantity by 1.

- Finally set $\hat{m}_0 = n - \sum_{j=1}^{k} \hat{m}_j$.

We say that the bucketization has *failed* if in the second step we had to decrease any $\hat{m}_j$ for which $\frac{(1+\epsilon)^{j-1}}{n}\epsilon \ge \frac{\epsilon}{k}$.

Lemma 6.6.1. *Suppose that $m_0, \ldots, m_k$, $\alpha_0, \ldots, \alpha_k$ are such that :*

- $\sum_{j=0}^{k} m_j = n$

- $\sum_{j=1}^{k} m_j \frac{(1+\epsilon)^{j-1}}{n}\epsilon \le 1$

- $\sum_{j=0}^{k} \alpha_j = 1$

- $|m_j - \alpha_j| \frac{(1+\epsilon)^{j-1}}{n}\epsilon < \frac{\epsilon}{2k}$ *for all* $1 \le j \le k$

and let $\hat{m}_0, \ldots, \hat{m}_k$ be the bucketization of $\alpha_0, \ldots, \alpha_k$. Then $\hat{m}_0, \ldots, \hat{m}_k$ are all well defined (the bucketization process did not fail), and additionally if $\tilde{\mu}$ is the tentative distribution according to $m_0, \ldots, m_k$ and $\hat{\mu}$ is the tentative distribution according to $\hat{m}_0, \ldots, \hat{m}_k$, then the distance between $\hat{\mu}$ and $\tilde{\mu}$ (after some permutation) is at most 4ϵ.

Proof. The first thing to note is that $m_j = \hat{m}_j$ for all j for which $\frac{(1+\epsilon)^{j-1}}{n}\epsilon \geq \frac{\epsilon}{k}$, before the decreasing step, so there will be no need to decrease these values and the bucketization will not fail.

For all $j \geq 1$, before decreasing some of the $\hat{m}_j$ we have that $|m_j - \hat{m}_j|\frac{(1+\epsilon)^{j-1}}{n}\epsilon < \frac{\epsilon}{k}$ (if $\frac{(1+\epsilon)^{j-1}}{n}\epsilon \leq \frac{\epsilon}{k}$ then the distance is not more than doubled by the rounding, and otherwise it follows from $m_j = \hat{m}_j$). Since the bucketization did not fail, the decreasing step only affects values $\hat{m}_j$ for which $\frac{(1+\epsilon)^{j-1}}{n}\epsilon < \frac{\epsilon}{k}$, and the total required decrease in them was by not more than k (as the rounding in the first step of the bucketization added no more than 1 to each of them), we obtain the total bound $\sum_{j=1}^{k}|m_j - \hat{m}_j|\frac{(1+\epsilon)^{j-1}}{n}\epsilon \leq 2\epsilon$.

Let r_i denote the corresponding values in the definition of $\tilde{\mu}$ being the tentative distribution according to $m_0, \ldots, m_k$, and $\hat{r}_i$ be the analog values in the definition of $\hat{\mu}$ being the tentative distribution according to $\hat{m}_0, \ldots, \hat{m}_k$. By what we already know about $\sum_{j=1}^{k}|m_j - \hat{m}_j|\frac{(1+\epsilon)^{j-1}}{n}$ we have in particular $\sum_{i=1}^{n}\hat{r}_i = \sum_{i=1}^{n} r_i \pm 3\epsilon$. Combined with the known bounds on $\sum_{i=1}^{n} r_i$, we can conclude by finding a permutation for which we can bound $\sum_{i=1}^{n}|r_i - \hat{r}_i|$ by 3ϵ, which will give the 4ϵ bound on the distribution distance $\frac{1}{2}\sum_{i=1}^{n}|\tilde{\mu}(i) - \hat{\mu}(i)|$.

The permutation we take is the one that maximizes the number of i's for which $r_i = \hat{r}_i$; for the value $\frac{(1+\epsilon)^{j-1}}{n}\epsilon$ we can find $\min\{m_j, \hat{m}_j\}$ such i's (for every $1 \leq j \leq k$), and the hypothetical worst case is that whenever $r_i \neq \hat{r}_i$ one of them is zero (sometimes the realizable worst case is in fact not as bad as the hypothetical one). Thus the $\sum_{j=1}^{k}|m_j - \hat{m}_j|\frac{(1+\epsilon)^{j-1}}{n}\epsilon \leq 3\epsilon$ bound leads to the 4ϵ bound on the distribution distance. □

A problem still remains, in that sampling from μ will not obtain a value α_j close enough to the required $m_j\frac{(1+\epsilon)^{j-1}}{n}\epsilon$. The variations in the $\mu(i)$ inside the bucket M_j itself could be higher than the $\frac{\epsilon}{2k}$ that we need here. In the next subsection we will construct not only a "bucket identifying" oracle, but tie it with an explicit persistent sampler that will simulate the approximate distribution rather than the original μ.

6.6.2 From bucketing to learning

An explicit persistent sampler is almost sufficient to learn the distribution. The next step would be to estimate the size of a bucket of the ϵ-fine distribution $\tilde{\mu}$ by explicit sampling (i.e. getting the samples along with their probabilities). However, Lemma 6.6.1 requires an approximation not of $\tilde{\mu}(M_j)$ (where M_j is a bucket of $\tilde{\mu}$) but rather of $|M_j|\frac{(1+\epsilon)^{j-1}}{n}\epsilon$. In other words, we really need to approximate $\overline{\mu}(M_j)$, where $\overline{\mu}$ is the corresponding trimmed distribution.

Therefore we define the following explicit sampler for an ϵ-trimmed distribution. We "bend" the definition a little, as this sampler will not be able to provide the corresponding probability for $i = 0$. It is given conditional sample access to a distribution μ.

Algorithm 6.7 Trimming Sampler

Input: Repetition parameter s, universe size n, distance parameter ϵ and confidence parameter δ.

Output: An index $i \in [n]$ and a real number.

```
1: Run the Persistent Sampler (Algorithm 6.6) with parameters $\epsilon, \delta$ and $s$ to obtain $i$ and $\eta$;
   additionally retain $p_1, \ldots, p_{\log n}$ as calculated during the run of the Persistent Sampler.
2: if there exists $l$ for which $p_l < \frac{\epsilon}{2\log(n)}$ then
3:     return "0"
4: if $\eta < \frac{\epsilon}{n}$ then
5:     return "0"
6: Let $j$ be the largest integer for which $\frac{(1+\epsilon)^{j-1}}{n}\epsilon \leq \eta$, and set $\eta' = \frac{(1+\epsilon)^{j-1}}{n}\epsilon$.
7: With probability $1 - \eta'/\eta$ return "0", and with probability $\eta'/\eta$ return $(i, j)$ (where $j$
   corresponds to $\overline{\mu}(i) = \eta'$).
```

The following observation is now easy.

Observation 6.9. The trimming sampler (Algorithm 6.7) is a (δ, s)-persistent sampler, and explicit whenever the returned sample is not 0, for the distribution over distributions that results from taking the ϵ-trimming of an ϵ-fine distribution $\tilde{\mu}$ and its corresponding B that was drawn according to the $(\epsilon, \frac{\delta}{s\log n})$-condensation of μ. The algorithm uses in total $2^5 \cdot \epsilon^{-4} \log^5 n \cdot \log(s\delta^{-1} \log n)$ many adaptive conditional samples from μ to output a sample.

Proof. The number of samples is inherited from Algorithm 6.6 as no other samples are taken. The algorithm switches the return value to "0" whenever $i \in B$ (as defined in the proof of Lemma 6.5.2), and otherwise returns "0" exactly according to the corresponding conditional probability difference for i between $\tilde{\mu}$ (as in the definition of a reconstituted distribution) and $\overline{\mu}$ (as in the definition of the corresponding trimmed distribution). Finally, whenever the returned sample is $i > 0$ the algorithm clearly returns the corresponding j_i (see Definition 6.15). □

We are now ready to present the algorithm providing Theorem 6.2. It is given conditional sample access to a distribution μ.

Algorithm 6.8 Distribution Approximation

Input: Distance parameter ϵ and confidence parameter δ.

Output: An index $i \in [n]$ and a real number.

```
1: Set $s = 2^{12}\epsilon^{-4}\log^2(n)\log(\delta^{-1})$, and $k = \frac{\log n \log(12\epsilon^{-1})}{\log^2(1+\epsilon/12)}5$.
2: Take $s$ samples through the $(\epsilon/12, \delta/2, s)$-Trimming Sampler.
3: Denote by $s_0$ the number of times that the sampler returned "0", and for $1 \leq j \leq k$ denote
   by $s_j$ the number of times that the sampler returned $(i, j)$ for any $i$.
4: Let $m'_0, \ldots, m'_k$ be the bucketization of $\alpha_0 = \frac{s_0}{s}, \ldots, \alpha_k = \frac{s_k}{s}$.
5: return the tentative distribution according to $m'_0, \ldots, m'_k$.
```

Lemma 6.6.2. *The Distribution Approximation algorithm (Algorithm 6.8) will with probability at least $1 - \delta$ return a distribution that is ϵ-close to a permutation of μ. This is performed using at most $\tilde{O}(\epsilon^{-8}\log^7 n \log^2(\delta^{-1}))$ conditional samples.*

Proof. The number of samples is immediate from the algorithm statement and Observation 6.9.

By Observation 6.9, with probability at least $1-\delta/2$ all samples of the Trimming Sampler will be from one $\epsilon/12$-trimming of some $\epsilon/12$-fine distribution $\overline{\mu}$. Set $m_0 = |\{1 \leq i \leq n : \overline{\mu}(i) = i\}|$ and for $1 \leq j \leq k$ set $m_j = |\{i : \overline{\mu}(i) = \frac{(1+\epsilon)^{j-1}}{n}\epsilon\}|$. Recall that the $\epsilon/12$-renormalized distribution corresponding to $\overline{\mu}$ is in fact the tentative distribution according to $m_0, \ldots, m_k$. By Lemma 6.5.3, this distribution is $\epsilon/2$-close to μ.

Note now that for every $1 \leq j \leq k$ the expectation of α_j is exactly $m_j \frac{(1+\epsilon/12)^{j-1}}{n}\epsilon/12$. By virtue of a Chernoff bound and the union bound, our choice of s implies that with probability $1-\delta/2$ (conditioned on the previous event) we get values such that $|m_j - \alpha_j| \frac{(1+\epsilon/12)^{j-1}}{n}\epsilon/12 < \frac{\epsilon/12}{2k}$ for every $1 \leq j \leq k$. This satisfies the assertions of Lemma 6.6.1, and thus the tentative distribution according to $m'_0, \ldots, m'_k$ will be $\epsilon/2$-close to the tentative distribution according to $m_0, \ldots, m_k$, and hence will be ϵ-close to μ. □

Note that if we were to use this algorithm for testing purposes, the dependence on δ^{-1} can be made logarithmic by setting it to $1/3$ and repeating the algorithm $\log(\delta^{-1})$ times, taking the majority vote (but this may not be possible if we are interested in $\overline{\mu}$ itself).

6.7 Lower bounds for label invariant properties

In this section we prove two sample complexity lower bounds for testing label-invariant distribution properties in our model. The first is for testing uniformity, and applies to non-adaptive algorithms. The second bound is for testing whether a distribution is uniform over some subset $U \subseteq \{1, \ldots, n\}$ of size exatcly 2^{2k} for some k, and applies to general (adaptive) algorithms.

The analysis as it is written relies on the particular behavior of our model when conditioning on a set of probability zero, but this can be done away with: Instead of a distribution μ with probabilities $p_1, \ldots, p_n$ over $[n]$, we can replace it with the $o(1)$-close distribution $\hat{\mu}$ with probabilities $\hat{p_1}, \ldots, \hat{p_i}$ where $\hat{p_i} = \frac{1}{n^2} + (1 - \frac{1}{n})p_i$. The same analysis of why an algorithm will fail to correctly respond to μ will pass on to $\hat{\mu}$, which has no zero probability sets.

6.7.1 Preliminary definitions

We start with some definitions that are common to both lower bounds.

First, an informal reminder of Yao's method for proving impossibility results for general randomized algorithms: Suppose that there is a fixed distribution over "positive" inputs (inputs that should be accepted) and a distribution over "negative" inputs, so that no deterministic algorithm of the prescribed type can distinguish between the two distributions. That is, suppose that for every such algorithm, the difference in the acceptance probability over both input distributions is $o(1)$. This will mean that no randomized algorithm can distinguish between these distributions as well, and hence for every possible randomized algorithm there is a positive instance and a negative instance so that it cannot be correct for both of them.

In our case an "input" is a distribution μ over $\{1, \ldots, n\}$, and so a "distribution over inputs" is in fact a distribution over distributions. To see why a distribution over distributions cannot be replaced with just a single "averaged distribution", consider the following example. Assume that an algorithm takes two independent samples from a distribution μ over $\{1,2\}$. If μ is with probability $\frac{1}{2}$ the distribution always giving 1, and with probability $\frac{1}{2}$ the distribution always giving 2, then the two samples will be either $(1,1)$ or $(2,2)$, each with probability $\frac{1}{2}$. This can never be the case if we had used a fixed distribution for μ, rather than a distribution over distributions.

What it means to be a deterministic version of our testers will be defined below; as with other settings, these result from fixing in advance the results of the internal coin tosses of the randomized testers. The following are the two distributions over distributions that we will use to prove lower bounds (a third one will simply be "pick the uniform distribution over $\{1,\dots,n\}$ with probability 1").

Definition 6.10. Given a set $U \subseteq \{1,\dots,n\}$, we define the *$U$-distribution* to be the uniform distribution over U, that is we set $p_i = 1/|U|$ if $i \in U$ and $p_i = 0$ otherwise.

The *even uniblock distribution* over distributions is defined by the following:

1. Uniformly choose an integer k such that $\frac{1}{8}\log n \le k \le \frac{3}{8}\log n$.
2. Uniformly (from all possible such sets) pick a set $U \subseteq \{1,\dots,n\}$ of size exactly 2^{2k}.
3. The output distribution μ over $\{1,\dots,n\}$ is the U-distribution (as defined above).

The *odd uniblock distribution* over distributions is defined by the following:

1. Uniformly choose an integer k such that $\frac{1}{8}\log n \le k \le \frac{3}{8}\log n$.
2. Uniformly (from all possible such sets) pick a set $U' \subseteq \{1,\dots,n\}$ of size exactly 2^{2k+1}.
3. The output distribution μ over $\{1,\dots,n\}$ is the U'-distribution.

Finally, we also identify the *uniform distribution* with the distribution over distributions that picks with probability 1 the uniform distribution over $\{1,\dots,n\}$.

For these to be useful for Yao arguments, we first note their distance properties.

Observation 6.11. Any distribution over $\{1,\dots,n\}$ that may result from the even uniblock distribution over distributions is $\frac{1}{2}$-far from the uniform distribution over $\{1,\dots,n\}$, as well as $\frac{1}{2}$-far from any distribution that may result from the odd uniblock distribution over distributions.

Proof. This follows directly from a variation distance calculation. Specifically, the variation distance between a uniform distribution over U and (a permutation of) a uniform distribution over V with $|V| \ge |U|$ (which is minimized when we make the permutation such that $U \subseteq V$) is at least $(|V| - |U|)/|V|$. In our case we always have $|V| \ge 2|U|$, and hence the lower bound. □

All throughout this section we consider properties that are label-invariant (such as the properties of being in the support of the distributions defined above). This allows us to simplify the analysis of our algorithms.

First, some technical definitions.

Definition 6.12. Given $A_1, \ldots, A_r \subseteq \{1, \ldots, n\}$, the *atoms generated by* $A_1, \ldots, A_r$ are all non-empty sets of the type $\bigcap_{j=1}^{r} C_j$ where every C_j is one of A_j or $\{1, \ldots, n\} \setminus A_j$. In other words, these are the minimal (by containment) non-empty sets that can be created by boolean operations over $A_1, \ldots, A_r$. The family of all such atoms is called the *partition* generated by $A_1, \ldots, A_r$; when $r = 0$ that partition includes the one set $\{1, \ldots, n\}$.

Given $A_1, \ldots, A_r$ and $j_1, \ldots, j_r$ where $j_i \in A_i$ for all i, the *r-configuration* of $j_1, \ldots, j_r$ is the information for any $1 \leq l, k \leq r$ of whether $j_k \in A_l$ (or equivalently, which is the atom that contains j_k) and whether $j_k = j_l$.

The label-invariance of all properties discussed in this section will allow us to "simplify" our algorithms prior to proving lower bounds. We next define a simplified version of a non-adaptive algorithm.

Definition 6.13. A *core non-adaptive distribution tester* is a non-adaptive distribution tester, that in its last phase bases its decision to accept or reject only on the $t(\epsilon)$-configuration of its received samples and on its internal coin tosses.

For a core non-adaptive tester, fixing the values of the internal "coins" in advance gives a very simple deterministic counterpart (for use in Yao arguments): The algorithm now consists of a sequence of fixed sets $A_1, \ldots, A_{t(\epsilon)}$, followed by a function assigning to every possible $t(\epsilon)$-configuration a decision to accept or reject.

We note that indeed in the non-adaptive setting we only need to analyze core algorithms:

Observation 6.14. A non-adaptive testing algorithm for a label-invariant property can be converted to a corresponding core algorithm with the same sample complexity.

Proof. We start with the original algorithm, but choose a uniformly random permutation σ of $\{1, \ldots, n\}$ and have the algorithm act on the correspondingly permuted input distribution, rather than the original one. That is, every set A_i that the algorithm conditions on is converted to $\{\sigma(k) : k \in A_i\}$, while instead of j_i the algorithm receives $\sigma^{-1}(j_i)$. This clearly preserves the guaranteed bounds on the confidence parameter if the property is label-invariant.

To conclude, note that due to the random permutation, all outcomes for $j_1, \ldots, j_t$ that satisfy a given configuration are equally likely, and hence can be simulated using internal coin tosses once the configuration itself is made known to the algorithm. □

For an adaptive algorithm, the definition will be more complex. In fact we will need to set aside some "external" coin tosses, so that also the "deterministic" counterpart will have a probabilistic element, but it will be a manageable one.

Definition 6.15. A *core adaptive distribution tester* is an adaptive distribution tester, that acts as follows.

- In the i'th phase, based only on the internal coin tosses and the configuration of the sets $A_1, \ldots, A_{i-1}$ and $j_1, \ldots, j_{i-1}$, the algorithm assigns a number k_A for every atom A that is generated by $A_1, \ldots, A_{i-1}$, between 0 and $|A \setminus \{j_1, \ldots, j_{i-1}\}|$. If all provided k_A are 0 then K_i may not be empty. Additionally the algorithm provides $K_i \subseteq \{1, \ldots, i-1\}$.

- A set $B_i \subseteq \{1,\ldots,n\} \setminus \{j_1,\ldots,j_{i-1}\}$ is drawn uniformly among all such sets whose intersection with every atom A as above is of size k_A, and A_i is set to $B_i \cup \{j_k : k \in K_i\}$. The random draw is done independently of prior draws and the algorithm's own internal coins, and A_i is not revealed to the algorithm (however, the algorithm will be able to calculate the sizes of the atoms in the partition generated by $A_1 \ldots, A_i$ using the $i-1$-configuration, and the numbers provided based on it and the internal coin tosses).
- A sample j_i is drawn according to μ conditioned over A_i, independently of all other draws. j_i is not revealed to the algorithm, but the new i-configuration is revealed (in other words, the new information that the algorithm receives is whether $j_i \in A_k$ and whether $j_i = j_k$ for each $k < i$).
- After $t(\epsilon)$ such phases, the algorithm bases its decision to accept or reject only on the t-configuration of its received samples and on its internal coin tosses.

Note that also a "deterministic" version of the above algorithm acts randomly, but only in a somewhat "oblivious" manner. The sets A_i will still be drawn at random, but the decisions that the algorithm is allowed to make about them (through the k_A numbers and the K_i sets) as well as the final decision whether to accept or reject will all be deterministic. This is since a deterministic version fixes the algorithm's internal coins and only them.

Also for adaptive algorithms we need to analyze only the respective core algorithms.

Observation 6.16. An adaptive testing algorithm for a label-invariant property can be converted to a corresponding core algorithm with the same sample complexity.

Proof. Again we use a uniformly random permutation σ of $\{1,\ldots,n\}$. Regardless of how the original set A_i was chosen, now it will be chosen uniformly at random among all sets satisfying the same intersection sizes with the atoms of the partition generated by $A_1,\ldots,A_{i-1}$ and the same membership relations with $j_1,\ldots,j_{i-1}$. Hence the use of a uniformly drawn set based on the k_A numbers and K_i is justified, and since σ is not revealed to the algorithm, the particular resulting set A_i is not revealed.

Also, the probability for a particular value of j_i now can depend only on the resulting i-configuration, and hence it is sufficient to reveal only the configuration to the algorithm – the algorithm can then use internal coin tosses to simulate the actual value of j_i (uniformly drawing it from all values satisfying the same configuration). The same goes for the decision whether to accept or reject in the end.

To further illustrate the last point, note that the analysis does not change even if we assume that at every phase, after choosing A_i we also draw a new random permutation, chosen uniformly at random among all those that preserve $j_1,\ldots,j_{i-1}$ and the atoms of $A_1,\ldots,A_i$ (but can "reshuffle" each atom internally). Then the "position inside its atom" of j_i will be completely uniform among those of the same configuration (if the configuration makes it equal to a previous j_k then there is only one choice for j_i anyway). □

6.7.2 Uniformity has no constant sample non-adaptive test

Theorem 6.17. *Testing uniformity requires at least $\Omega(\log\log n)$ non-adaptive conditional samples (for some fixed ϵ).*

Proof. This follows from Lemma 6.7.5 below. □

To prove this lower bound, we show that for any fixed t and large enough n, no deterministic non-adaptive algorithm can distinguish with probability $\frac{1}{3}$ between the case where the input distribution is the uniform one (with probability 1), and the case where the input distribution is drawn according to the even uniblock distribution over distributions. Recall that such a deterministic algorithm is in fact given by fixed sets $A_1, \ldots, A_t \subseteq \{1, \ldots, n\}$ and a fixed acceptance criteria based on the t-configuration of the obtained samples (to see this, take a core non-adaptive testing algorithm and arbitrarily fix its internal coins).

We now analyze the performance of a deterministic non-adaptive tester against the even uniblock distribution. Asymptotic expressions are for an increasing n.

Definition 6.18. We call a set $A \subseteq \{1, \ldots, n\}$ *large* if $|A| > n2^{\sqrt{\log n}}/|U|$, where U is the set chosen in the construction of the even uniblock distribution. We call A *small* if $|A| < n2^{-\sqrt{\log n}}/|U|$.

Lemma 6.7.1. *With probability at least $1 - \frac{2^{t+2}}{\sqrt{\log n}}$ over the choice of U, all atoms in the partition generated by $A_1, \ldots, A_t$ are either large or small.*

Proof. There are at most 2^t atoms. An atom A is neither large nor small if two inequalities both hold: $n2^{-\sqrt{\log n}} \leq |A||U| \leq n2^{\sqrt{\log n}}$. Recall that $|U| = 2^{2k}$ where $\frac{1}{8}\log n \leq k \leq \frac{3}{8}\log n$ is chosen uniformly. Therefore, for a fixed A, there are at most $\sqrt{\log n}$ values of k which will make it neither large nor small. Since the range of k is of size $\frac{1}{4}\log n$, we get that with probability at most $\frac{4}{\sqrt{\log n}}$ the atom A is neither large nor small. Taking the union bound over all atoms gives the statement of the lemma. □

Lemma 6.7.2. *With probability at least $1 - 2^{t-\sqrt{\log n}}$, no small atom intersects U.*

Proof. Given a fixed k, for any small set A the probability of it intersecting U is clearly bounded by $2^{-\sqrt{\log n}}$. We can now conclude the proof by union-bounding over all small atoms, whose number is bounded by 2^t. □

Lemma 6.7.3. *With probability $1 - \exp\left(t - t^2 \cdot 2^{\sqrt{\log n}/2-1}\right)$, for every large atom A, we have $|A \cap U| = \left(1 \pm \frac{t}{2^{\sqrt{\log n}/4}}\right)|A| \cdot |U|/n$.*

Proof. This is by a large deviation inequality followed by a union bound over all atoms. Note first that if instead of U we had a uniformly random sequence $u_1, \ldots, u_{2^{2k}}$ (chosen with possible repetitions), then this would have been covered by Lemma 2.0.1. However, U is a random set of fixed size instead. For this we appeal to Section 6 of [Hoe63], where it is proved that moving from a Binomial to a Hypergeometric distribution (which corresponds to choosing the set U with the fixed size) only makes the distribution more concentrated. The rest follows by the fact that A is large. □

Now we can take $t \leq \frac{1}{4}\log\log n$ and put forth the following lemma, which implies that the even uniblock distribution over distributions is indeed indistinguishable from the uniform distribution by a deterministic non-adaptive core algorithm using only t samples.

Lemma 6.7.4. *For $t \leq \frac{1}{4}\log\log n$, with probability $1 - o(1)$, the distribution over $\{1, \ldots, n\}$ obtained from the even uniblock distribution over distributions, is such that the resulting distribution over the configurations of $j_1, \ldots, j_t$ is $o(1)$-close in the variation distance to the distribution over configurations resulting from the uniform distribution over $\{1, \ldots, n\}$.*

Proof. With probability $1 - o(1)$ all of the events in Lemmas 6.7.1, 6.7.2 and 6.7.3 occur. We prove that in this case the two distributions over configurations are $o(1)$-close. Recall that the uniform distribution over the set U (resulting from the uniblock distribution) is called the *U-distribution.* The lemma follows from the following:

- A sample taken from a set A_i that contains only small atoms will be uniform from this set (and independent of all others), both for the uniform distribution and the U-distribution. For the U-distribution it follows from U not intersecting A_i at all (recall that in our model, a conditional sample from a set of zero probability returns a uniformly random element from that set).

- A sample taken from a set A_i that contains some large atom will not be identical to any other sample with probability $1 - o(1)$ for both distributions. This follows from the birthday paradox: Setting A to be the large atom contained in A_i, recall that we have $|A \cap U| = \left(1 \pm \frac{\log\log n/4}{2^{\sqrt{\log n/4}}}\right)|A| \cdot |U|/n$. This quantity is $\omega\left((\log\log n)^2\right)$. Thus for a fixed i the probability for a collision with any other j is $o(1/\log\log n)$ (regardless of whether A_j contains a large atom), and hence with probability $1 - o(1)$ there will be no collision for any i for which A_i contains a large atom.

- For a set A_i containing a large atom, the distribution over the algebra of the events $j_i \in A_k$ (which corresponds to the distribution over which atom in the partition generated by $A_1, \ldots, A_t$ contains j_i) are $o(1/\log\log(n))$ close for both distributions. To show this we analyze every atom A generated by $A_1, \ldots, A_t$ that is contained in A_i separately. If A is small, then for the uniform distribution, j_i will not be in it with probability $1 - o(1/\log\log(n))$ (a small atom is in particular of size $o(|A_i|/\log\log(n))$ since A_i contains a large atom as well), while for the U-distribution this is with probability 1 (recall that we conditioned on the event of U not intersecting any small atom). If A is large, then we have $|A \cap U| = \left(1 \pm \frac{\log\log(n)/4}{2^{\sqrt{\log n/4}}}\right)|A| \cdot |U|/n$, implying that the probabilities for $j_i \in A$ for the U-distribution and the uniform one are only $o(1/(\log\log(n))^2)$ apart, implying an $o(1/\log\log(n))$ difference in the distributions over the identity of A.

The items above allow us to conclude the proof. They mean that for both the $|U|$-distribution (conditioned on the events in Lemmas 6.7.1, 6.7.2 and 6.7.3) and the uniform distribution, the resulting distributions over configurations are $o(1)$-close to the one resulting by setting the following:

1. First for every i for which A_i contains only small atoms, uniformly pick $j_i \in A_i$ independently of all other random choices; write down the equalities between these samples and the atoms to which these samples belong.

2. Then, for every i for which A_i contains a large atom, write j_i as having no collisions with any other sample; then pick the atom containing j_i from all atoms contained in A_i according to their relative sizes, in a manner independent of all other random choices. □

Lemma 6.7.4 allows us to conclude the argument by Yao's method.

Lemma 6.7.5. *All non-adaptive algorithms taking $t \le \frac{1}{4}\log\log n$ conditional samples will fail to distinguish the uniform distribution from the even uniblock distribution over distributions (which are all $\frac{1}{2}$-far from uniform) with any probability more than $o(1)$.*

Proof. By Observation 6.14 it is enough to consider core non-adaptive algorithms, and by Yao's argument it is enough to consider deterministic ones.

For any deterministic non-adaptive core algorithm (characterized by $A_1, \dots, A_t$ and a function assigning a decision to every possible configuration), by Lemma 6.7.4 the even uniblock distribution with probability $1 - o(1)$ will choose a U-distribution, which in turn will induce a distribution over configurations that is $o(1)$-close to that induced by the uniform distribution over $\{1, \dots, n\}$. This means that if we look at the distribution over configurations caused by the even uniblock distribution over distributions, it will also be $o(1)$-close to the one induced by the uniform distribution. Therefore the acceptance probabilities of the algorithm for both distributions over distributions are $o(1)$-close. □

6.7.3 A label-invariant property with no constant sample adaptive test

Theorem 6.19. *There exists a label invariant property such that any adaptive testing algorithm for it must use at least $\Omega(\sqrt{\log\log n})$ conditional samples (for some ϵ).*

Proof. This follows from Lemma 6.7.9 below. □

The property will be that of the distribution being the possible result of the even uniblock distribution over distributions. In other words, it is the property of being equal to the U-distribution over some set U of size 2^{2k} for some $\frac{1}{8}\log n \le k \le \frac{3}{8}\log n$.

We show that no "deterministic" adaptive core algorithm can distinguish between the even and odd uniblock distributions using $o(\sqrt{\log\log n})$ samples, while by Observation 6.11 a proper $\frac{1}{2}$-test must distinguish between these. Considering such algorithms, we first note that they can be represented by decision trees, where each node of height i corresponds to an $i-1$-configuration of the samples made so far. An internal node describes a new sample, through the numbers k_A provided for every atom A of $A_1, \dots, A_i$ (where the atoms are labeled by their operations, as the A_i themselves are not revealed to the algorithm), and the set K_i. All these parameters can be different for different nodes of height i. A leaf is labeled with an accept or reject decision.

The basic ideas of the analysis are similar to those of the previous subsection, but the analysis itself is more complex because we have to consider the "partition generated by the samples so far" in every step of the algorithm. The first thing to note is that there are not too many nodes in the decision tree.

Observation 6.20. The number of nodes in a decision tree corresponding to a t-sample algorithm is less than $t2^{2t^2}$.

Proof. A configuration can be described by assigning each of the i samples with a vector of length $2i$, indicating which sets do they belong to and which of the other samples are they equal to. This gives an $i \times 2i$ binary matrix, where every possible i-configuration for i samples corresponds to some such matrix. That gives us at most 2^{2i^2} possible i-configurations. Summing for all $i \le t$ gives the bound in the statement. □

From now on we will always assume that n is larger than an appropriate fixed constant. For the analysis, we consider two input distributions as being drawn at once, one according to the even uniblock distribution and the other according to the odd uniblock distribution. We first choose $\frac{1}{8}\log n \leq k \leq \frac{3}{8}\log n$ uniformly at random, and then uniformly choose a set U of size 2^{2k} and a set U' of size 2^{2k+1}. We then set μ to be the U-distribution and μ' to be the U'-distribution.

We will now show that the fixed decision tree accepts with almost the same probability when given either μ or μ', which will allow us to conclude the proof using Yao's argument. We start with a notion of "large" and "small" similar to the one used for non-adaptive algorithms, only here we need it for the numbers themselves.

Definition 6.21. We call a number b *large* with respect to U if $b > n2^{\sqrt{\log n}}/|U|$. We call b *small* with respect to U if $b < n2^{-\sqrt{\log n}}/|U|$. We make the analogous definitions with respect to U'.

Lemma 6.7.6. *With probability at least $1 - \frac{t2^{3t^2+2}}{\sqrt{\log n}}$, all "$k_A$" numbers appearing in the decision tree are either small with respect to both U and U', or large with respect to both U and U'.*

Proof. By Observation 6.20 the total of different "k_A" numbers is no more than $t2^{3t^2}$ (the number of nodes times 2^t – the bound on the size of the partition generated by $A_1, \ldots, A_i$ in every node). We can conclude similarly to the proof of Lemma 6.7.1 that since $|U|$ and $|U'|$ differ by a factor of 2, there are at most $\sqrt{\log n}$ values of k for which some fixed number k_A will not be either large with respect to both or small with respect to both. The bound in the statement then follows by union bound. □

From now on we assume that the event of Lemma 6.7.6 has occurred, and fix k (that is, the following will hold not only for the entire distributions, but also for the conditioning on every specific k for which the event of Lemma 6.7.6 is satisfied). The following lemma is analogous to the non-adaptive counterparts Lemma 6.7.2 and Lemma 6.7.3, but here it is proved by induction for every node that is reached while running the decision tree over the distribution drawn according to either μ or μ', where the inductive argument requires both statements to hold. This lemma will essentially be used as a warm-up, since the final proof will refer to the proof and not just the statement of the lemma.

Lemma 6.7.7. *Assuming that $t \leq \sqrt{\frac{1}{32}\log\log n}$, and conditioned on that the events of Lemma 6.7.6 have occurred, for every $1 \leq i \leq t$, with probability at least $1 - \frac{2^{t+1}}{\sqrt{\log n}}$, the following occur.*

- *All small atoms in the partition generated by $A_1, \ldots, A_i$ contain no members of either U or U' outside (possibly) $\{j_1, \ldots, j_{i-1}\}$.*
- *For every large atom B in the partition generated by the sets $A_1, \ldots, A_i$, we have both $|B \cap U| = \left(1 \pm \frac{i}{2^{\sqrt{\log n}/4}}\right)|B| \cdot |U|/n$ and $|B \cap U'| = \left(1 \pm \frac{i}{2^{\sqrt{\log n}/4}}\right)|B| \cdot |U'|/n$.*

Proof. We shall prove the lemma not only conditioned on the event of Lemma 6.7.6, but also conditioned on any fixed $|U|$ (and $|U'| = 2|U|$) for which Lemma 6.7.6 is satisfied. We assume by induction that this occurs for the atoms in the partition generated by $A_1, \ldots, A_{i-1}$ with probability at least $1 - \frac{2^i}{\sqrt{\log n}}$, and prove it for $A_1, \ldots, A_i$ with probability at least $1 - \frac{2^{i+1}}{\sqrt{\log n}}$. Recall that the way A_i is generated, the algorithm in fact specifies how many members of it will appear in $A \setminus \{j_1, \ldots, j_{i-1}\}$ for every atom A of the partition generated by $A_1, \ldots, A_{i-1}$ (while specifying exactly which of $j_1, \ldots, j_{i-1}$ will appear in it), and then the actual set is drawn uniformly at random from those that satisfy the specifications.

We show the conclusion of the lemma to hold even if U and U' are held fixed (as long as they satisfy the induction hypothesis and their sizes satisfy the assertion of Lemma 6.7.6). Let B be an atom of $A_1, \dots, A_i$ and let A be the atom of $A_1, \dots, A_{i-1}$ so that $B \subseteq A$. We have several cases to consider, conditioned on the fact that the event in the statement does occur for $i-1$.

- If A is small, then so is B. By the induction hypothesis $A \setminus \{j_1, \dots, j_{i-1}\}$ has no members of U or U', and hence so does B. This happens with (conditional) probability 1.
- If the atom A is large but the atom B is small, by the induction hypothesis we have that both $|A \cap U| = \left(1 \pm \frac{(i-1)}{2^{\sqrt{\log n}/4}}\right)|A| \cdot |U|/n$ and $|A \cap U'| = \left(1 \pm \frac{(i-1)}{2^{\sqrt{\log n}/4}}\right)|A| \cdot |U'|/n$. When this happens, as $B \setminus \{j_1, \dots, j_{i-1}\}$ is in fact chosen uniformly from all subsets of $A \setminus \{j_1, \dots, j_{i-1}\}$ of the same size (either k_A or $|A \setminus \{j_1, \dots, j_{i-1}\}| - k_A$), and since B is small, we can use a union bound to see that no member of either U or U' is taken into B, with probability at least $1 - 2^{1-\sqrt{\log n}}$.
- If the atom B is large (and hence so is A), then again by the induction hypothesis both $|A \cap U| = \left(1 \pm \frac{(i-1)}{2^{\sqrt{\log n}/4}}\right)|A| \cdot |U|/n$ and $|A \cap U'| = \left(1 \pm \frac{(i-1)}{2^{\sqrt{\log n}/4}}\right)|A| \cdot |U'|/n$. We also note that since B is large we have in particular $t \leq \frac{1/2}{2^{\sqrt{\log n}/4}}|B|$. We can now use a large deviation inequality (as in Lemma 6.7.3) to conclude the bounds for $|B \cap U|$ and $|B \cap U'|$ with probability $1 - 2\exp(-2^{\sqrt{\log n}/2-2})$.

Thus in all cases the statement will not hold with probability at most $\frac{1}{\sqrt{\log n}}$ for n large enough. By taking the union bound over all possibilities for B (up to 2^i events in total) we get that with probability at least $1 - \frac{2^i}{\sqrt{\log n}}$ the statement of the lemma holds for $A_1, \dots, A_i$, conditioned on the event occurring for $A_1, \dots, A_{i-1}$. A union bound with the event of the induction hypothesis happening for $A_1, \dots, A_{i-1}$ gives the required probability bound. □

We now prove the lemma showing the indistinguishability of the distrbituion μ from μ' whenever $t \leq \sqrt{\frac{1}{32}\log\log n}$, conditioned on the event of Lemma 6.7.6. We assume without loss of generality that the decision tree of the algorithm is full and balanced, which means that the algorithm will always take t samples even if its output was already determined before they were taken.

Lemma 6.7.8. *Assuming that $t \leq \sqrt{\frac{1}{32}\log\log n}$ and that the event of Lemma 6.7.6 has occurred, consider the resulting distributions of which of the leaves of the algorithm was reached. These two distributions, under μ compared to under μ', are at most $\frac{2^{3t+1}}{\sqrt{\log n}}$ apart from each other.*

Proof. The proof is reminiscent of the proof of Lemma 6.7.4 above, but requires more cases to be considered, as well as induction over the height of the nodes under consideration. Denoting this height by i, we shall prove by induction that the distributions over which of the height i nodes was reached, under μ compared to μ', are only at most $1 - \frac{2^{3i+1}}{\sqrt{\log n}}$ apart from each other.

We shall use the induction hypothesis that the corresponding distributions of the node of height $i-1$ (the parent of the node that we consider now) are at most $1 - \frac{2^{3i-2}}{\sqrt{\log n}}$ apart, and then show that the variation distance between the distributions determining the transition from a particular parent to a child node is no more than $\frac{2^{3i}}{\sqrt{\log n}}$, which when added to the difference in the distributions over the parent nodes gives required bound.

The full induction hypothesis will include not only the bound on the distributions of the parent nodes, but also a host of other assumptions, that we prove along to occur with probability at least $1 - \frac{2^{3i+1}}{\sqrt{\log n}}$. In particular, instead of using the statement of Lemma 6.7.7, we essentially re-prove it here. Hence the induction hypothesis also includes that all of the events proved during the inductive proof of Lemma 6.7.7 hold here with respect to $A_1, \ldots, A_{i-1}$. Also, as in the proof of Lemma 6.7.7, the conditional probability of them not holding for $A_1, \ldots, A_i$ is at most $\frac{2^i}{\sqrt{\log n}}$ (by the union bound done there for every atom generated by $A_1, \ldots, A_i$ of the event of the hypothesis failing for any single atom A). Therefore, we assume that additionally the inductive hypothesis used in the proof of Lemma 6.7.7 has occurred for $A_1, \ldots, A_i$, and prove that with probability at least $1 - \frac{2^{2i}}{\sqrt{\log n}}$ all other assertions of the inductive hypothesis occur as well as that the variation distance between the distributions over the choice of the child node is at most $\frac{2^{2i}}{\sqrt{\log n}}$. By a union bound argument (and for the variation distance, a "common large probability event" argument), this will give us the $1 - \frac{2^{3i}}{\sqrt{\log n}}$ bound that we need for the induction. Recall that the choice of child node depends deterministically on the question of which atom of $A_1, \ldots, A_i$ contains the obtained sample j_i, so in fact we will bound the distance between the distributions of the atom in which j_i has landed.

Additionally, we define by induction over i the following notion: An index i is called *smallish* if all the "k_A" numbers relating to it are small, and additionally K_i contains only smallish indexes (recall that $K_i \subseteq \{1, \ldots, i-1\}$). A final addition to our induction hypothesis is that with probability at least $1 - \frac{2^{3i-2}}{\sqrt{\log n}}$, in addition to all our other assertions, the following occur for every $i' < i$.

- The sample $j_{i'}$ is in U or respectively U' if and only if i' is not smallish (note that the assignment of smallish indexes depends on the parent node).
- If i' is not smallish but all its corresponding "k_A" numbers are small, then $j_{i'}$ is equal to some j_l where l is a non-smallish index smaller than i'.
- If there exists a large "k_A" number for i', then $j_{i'}$ is not equal to j_l for any $l < i'$, and additionally $j_{i'}$ lies in some atom A' for which the corresponding $k_{A'}$ is not small (it is allowed that $A' = A$).

We now work for every possible parent node of height $i - 1$ separately. Note that we restrict our attention to nodes whose corresponding $(i-1)$-configurations satisfy the induction hypothesis. Recall that we assume that the induction hypothesis in the proof of Lemma 6.7.7 has occurred for $A_1, \ldots, A_i$, and aim for a $\frac{2^{2i}}{\sqrt{\log n}}$ "failure probability" bound. We separate to cases according to the nature of $A_1, \ldots, A_i$.

- A sample taken from a set A_i, where i is smallish, will be uniform and independent of other samples, for both the U-distribution and the U'-distribution. Moreover, this j_i in itself will not be a member of U or respectively U'. This is since $A_i \setminus \{j_k : k \in K_i\}$ does not intersect U or U', together with the induction hypothesis for $\{j_k : k \in K_i\}$ (so also A_i does not intersect U or U'). So conditioned on the entire induction hypothesis for $i - 1$ and the hypothesis in the proof of Lemma 6.7.7 for $A_1, \ldots, A_i$, all assertions for i will occur with probability 1, and the distributions for selecting the height i node given this particular parent node are identical under either μ or μ'.
- A sample taken from a set A_i, where the k_A numbers are all small but i is not smallish, will be a member of U or respectively U', chosen uniformly (and independently) from $\{j_k : k \in K'_i\}$, where K'_i denotes the (non-empty) set of all non-smallish indexes in K_i. This is since $\{j_k : k \in K'_i\}$ is exactly the set of members of U or respectively of U' in A_i

(by the hypothesis for $A_1, \ldots, A_i$ there will be no member of U or U' in $A_i \setminus \{j_k : k \in K_i\}$, and the rest follows from the induction hypothesis concerning smallish indexes). Again the assertions for i follow with probability 1 (conditioned on the above hypotheses), and the distributions for selecting the height i node are identical.

- If a sample is taken from A_i where at least one of the k_A numbers is not small, then the following occur.
  - Since A_i in particular contains A, and both $|A \cap U| = \left(1 \pm \frac{i}{2\sqrt{\log n}/4}\right) |A| \cdot |U|/n$ and $|A \cap U'| = \left(1 \pm \frac{i}{2\sqrt{\log n}/4}\right) |A| \cdot |U'|/n$ by the assertion over $A_1, \ldots, A_i$ relating to Lemma 6.7.7, we note that in particular $i = o(\frac{1}{\sqrt{\log n}}|A_i \cap U|)$ and $i = o(\frac{1}{\sqrt{\log n}}|A_i \cap U'|)$, so with probability less than $\frac{1}{\sqrt{\log n}}$ (for n larger than some constant) we will get under either μ or μ' a sample that is identical to a prior one.
  - By the assertion over $A_1, \ldots, A_i$, an atom B inside A_i for which the corresponding k_B is small will not contain a member of U or U', and so j_i will not be in such an atom (in the preceding item we have already established that there are members of U and respectively U' in A_i).
  - By the assertion over $A_1, \ldots, A_i$, for every large atom B inside A_i we have both $|B \cap U| = \left(1 \pm \frac{i}{2\sqrt{\log n}/4}\right) |B| \cdot |U|/n$ and $|B \cap U'| = \left(1 \pm \frac{i}{2\sqrt{\log n}/4}\right) |B| \cdot |U'|/n$, implying that $\frac{|B \cap U|}{|U|} = \left(1 \pm \frac{i}{2\sqrt{\log n}/5}\right) \frac{|B \cap U'|}{|U'|}$ (for large enough n). Also, every small atom C inside A_i contains no members of U or U', so summing over all atoms of A_i we obtain $\frac{|A_i \cap U|}{|U|} = \left(1 \pm \frac{i}{2\sqrt{\log n}/5}\right) \frac{|A_i \cap U'|}{|U'|}$, and thus for every atom B of A_i (large or small) we finally have $\frac{|B \cap U|}{|A_i \cap U|} = \left(1 \pm \frac{i}{2\sqrt{\log n}/6}\right) \frac{|B \cap U'|}{|A_i \cap U'|}$ (for small atoms both sides are zero).

    Te final thing to note is that $\frac{|B \cap U|}{|A_i \cap U|}$ and respectively $\frac{|B \cap U'|}{|A_i \cap U'|}$ equal the probabilities of obtaining a sample from B under μ and respectively μ'. Summing over all atoms contained in A_i (of which there are 2^{i-1}) we obtain a difference over these distributions that is bounded by $\frac{2^i}{\sqrt{\log n}}$, which satisfies the requirements (also after conditioning on that the events related to the rest of the induction hypothesis have occurred).

Having covered all cases, this completes the proof that the inductive hypothesis follows to i, and thus the proof of the lemma is complete. □

Now we can conclude the argument by Yao's method to prove the following lemma that implies the theorem.

Lemma 6.7.9. *All adaptive algorithms taking $t \leq \sqrt{\frac{1}{32}\log\log n}$ conditional samples will fail to distinguish the even uniblock distribution over distributions from the odd one (whose outcomes are always $\frac{1}{2}$-far from those of the even distribution) with any probability larger than $o(1)$.*

Proof. By Observation 6.16 it is enough to consider only core adaptive algorithms, and then by Yao's argument it is enough to consider "deterministic" ones (the quote marks are because the external coin tosses are retained as per the definitions above). We now consider the decision tree of such an algorithm, and feed to it either μ or μ' that are drawn as per the definition above. With probability at least $1 - \frac{t2^{3t^2+2}}{\sqrt{\log n}} = 1 - o(1)$ the event of Lemma 6.7.6 has occurred, and conditioned on this event (or even if we condition on particular U and U'), Lemma 6.7.8

provides that the variation distance between the resulting distributions over the leafs is at most $\frac{2^{3t+1}}{\sqrt{\log n}} = o(1)$. In particular this bounds the difference between the (conditional) probabilities of the event of reaching an accepting leaf of the algorithm.

Since we have an $o(1)$ difference when conditioned on a $1 - o(1)$ probability event, we also have an $o(1)$ difference on the unconditional probability of reaching an accepting leaf under μ compared to μ'. This means that the algorithm cannot distinguish between the two corresponding distributions over distributions. $\square$

6.8 A lower bound for testing general properties of distributions

For properties that are not required to be label-invariant, near-maximal non-testability could happen also when conditional samples are allowed.

Theorem 6.1. *Some properties of distributions over $[n]$ require $\Omega(n)$ conditional samples to test (adaptively or not).*

Proof. We assume that n is even. We reduce the problem of testing general $n/2$-bit binary string properties $P \subseteq \{0,1\}^{n/2}$ to the problem of testing properties of distributions over $[n]$ using conditional samples, through Lemma 6.8.1 below. Then the lower bound of the theorem follows by the existence of hard properties $P \subseteq \{0,1\}^{n/2}$ that require $\Omega(n)$ queries to test, such as the original one of [GGR98] or the one of [BSHR05]. $\square$

The reduction proved in Lemma 6.8.1 is probabilistic in nature, succeeding with probability $1 - o(1)$ (which is sufficient for the hardness arguments to work), and only incurs an additional $O(1)$ factor in the query complexity. This means that every conditional sample made by the distribution tester is translated into (expected) $O(1)$ queries to the input binary string $x \in \{0,1\}^{n/2}$. The rest of this section is devoted to its proof.

6.8.1 The reduction lemma

We start with a few definitions. A string $y \in \{0,1\}^n$ is *balanced* if it has the same number of 0s and 1s (in particular we assume here that n is even). For $x \in \{0,1\}^{n/2}$, let $b(x) \in \{0,1\}^n$ be the string obtained by concatenating x with its bitwise complement (in which every original bit of x is flipped). Clearly $b(x)$ is balanced for all x.

For a property $P \subseteq \{0,1\}^{n/2}$, define $b(P) \subseteq \{0,1\}^n$ as $b(P) \triangleq \{b(x) : x \in P\}$.

Observation 6.2. For all $x, y \in \{0,1\}^{n/2}$, $d_{TV}(x,y) = d_{TV}(b(x), b(y))$.

Proof. Follows from the fact that if x and y differ in $d_{TV}(x,y) \cdot \frac{n}{2}$ entries, then $b(x)$ and $b(y)$ differ in $d_{TV}(x,y) \cdot n$ entries. $\square$

Observation 6.3. For all P and $\epsilon > 0$, ϵ-testing $b(P)$ requires at least as many queries as ϵ-testing P.

$\square$

Next, for every balanced string $x \in \{0,1\}^n$ we define a distribution μ_x on $[n]$ as follows:

- If $x_i = 0$ then $\mu_x(i) = \frac{1}{2n}$;
- if $x_i = 1$ then $\mu_x(i) = \frac{3}{2n}$.

Note that since x is balanced μ_x is indeed a distribution as $\sum_{i=1}^{n} \mu_x(i) = 1$.

Extending this definition further, for every property $P \subseteq \{0,1\}^{n/2}$ we define a property $\mathcal{P}_P$ of distributions over $[n]$ as follows: $\mathcal{P}_P \triangleq \{\mu_x : x \in b(P)\}$.

Observation 6.4. For all $x, y \in \{0,1\}^{n/2}$, $d_{TV}(b(x), b(y)) = 2 \cdot d_{TV}(\mu_{b(x)}, \mu_{b(y)})$, where the first distance refers to the normalized Hamming distance between binary strings, and the second is the variation distance between distributions.

□

Lemma 6.8.1. *For all P and $\epsilon > 0$, if ϵ-testing P with success probability $3/5$ requires at least q queries, then $\epsilon/2$-testing $\mathcal{P}_P$ with success probability $2/3$ requires at least $q/100$ conditional samples.*

Proof. By Observation 6.4, for all $x \in \{0,1\}^{n/2}$, if $x \in P$ then $\mu_{b(x)} \in \mathcal{P}_P$, and if $d_{TV}(x, P) > \epsilon$ then $d_{TV}(\mu_{b(x)}, \mathcal{P}_P) > \epsilon/2$. Now we show how to reduce the task of testing P to testing $\mathcal{P}_P$. Let T be a tester for $\mathcal{P}_P$ making at most $q/100$ conditional samples. Given an oracle access to the input string $x \in \{0,1\}^{n/2}$, which is to be tested for membership in P, we simulate each conditional sample $\emptyset \neq Q \subseteq [n]$ to $\mu_{b(x)}$ made by T as follows:

Algorithm 6.9 Sampler

```
Input: Set $\emptyset \neq Q \subseteq [n]$.
Output: An index $i \in Q$.
1: Pick $i \in Q$ uniformly at random. If $i \leq n/2$ query $x_i$ and set $v_i \leftarrow x_i$. Else, query $x_{i-n/2}$
   and set $v_i \leftarrow 1 - x_{i-n/2}$.
2: if $v_i = 1$ then
3:     return $i$
4: else
5:     With probability $1/3$ output $i$, and with the remaining probability go to Step 1.
```

It is clear that whenever Sampler outputs i with $v_i = 1$, then i is distributed uniformly among all indices $\{j \in Q : v_j = 1\}$. The same is true for i such that $v_i = 0$. So, to show that Sampler simulates conditional samples correctly, it remains to prove that the ratio between the probability of outputting i with $v_i = 1$ and the probability of outputting i with $v_i = 0$ is correct.

Let $q_1 \triangleq |\{i \in Q : v_i = 1\}|$ and $q_0 \triangleq |\{i \in Q : v_i = 0\}|$. According to our distribution $\mu_b(x)$, the distribution of indices in Q corresponding to the conditional sample is as follows:

- $\Pr[i] = \frac{3}{3q_1 + q_0}$ if $v_i = 1$.
- $\Pr[i] = \frac{1}{3q_1 + q_0}$ if $v_i = 0$.

In particular, the probability of selecting i such that $v_i = 1$ is $3q_1/q_0$ times the probability of selecting i with $v_i = 0$.

Let us now analyze what is the probability with which Sampler outputs (eventually) an index $i \in Q$ with $v_i = 1$, and with $v_i = 0$, respectively. At every round, an index i with $v_i = 1$ is output with probability $\frac{q_1}{q_1+q_0}$, and an index i with $v_i = 0$ is output with probability $\frac{q_0}{3(q_1+q_0)}$. With the remaining probability (of $\frac{2q_0}{3(q_1+q_0)}$) no index is output, and the process repeats independently of all previous rounds. Hence the ratio of the probability of outputting i such that $v_i = 1$ to the probability of outputting i with $v_i = 0$ is $3q_1/q_0$, as required. Note also that the expected number of rounds (and so queries to x) per one execution of Sampler is $(1 - \frac{2q_0}{3(q_1+q_0)})^{-1} \leq 3$.

The last ingredient in the reduction is a total-query counter, that makes sure that the number of queries to x does not exceed q (the lower bound). If so, the reduction fails. Since Sampler is called at most $q/100$ times (the query complexity of T), a $3/100 < 1/15$ bound on the failure probability follows by Markov's inequality, and we are done (the bound on the success probability follows even if we assume that the distribution tester "magically" guesses the correct answer whenever the reduction to the string property fails). $\square$

Chapter 7

Open Questions

In this thesis we have covered a range of results, but also introduced new models and avenues for research, many of them yet untapped. In this chapter we will suggest several intriguing open problems and paths for further research.

7.1 Testing formula satisfaction

In Section 3.7, we proved a lower bound against testing of formulas with non-boolean alphabets. This begs the following question:

Open Problem 7.1. Is there a sublinear upper bound for testing any fixed alphabet bounded arity formula?

Another interesting path is extending the techniques in Section 3.5, where we give a quasipolynomial upper bound for and-or formulas.

Open Problem 7.2. Is there a polynomial upper bound for testing formulas containing only and/or gates?

Open Problem 7.3. Can the techniques of Section 3.5 be extended to formulas containing also parity gates? what about 3-majority gates?

7.2 Partial property testing

The model of partial testing is gaining much interest, as evident by the independent work of Gur and Rothblum [GR15], and the follow up work of Goldreich, Gur and Rothblum [GGR15]. There is still a wealth of open problems left. One important issue is whether good paritions imply testability (to which we gave a positive answer for the case of proximity oblivious testable subproperties in Sections 5.6 and 5.7):

Open Problem 7.4. Let P be a property that can be partitioned into subsets $P_1, \ldots, P_k$ such that P is P_i-partially testable using q queries for every $1 \leq i \leq k$, what upper bound can we give on the query complexity for testing P (in terms of n, k and q)?

There is also the converse direction, of whether testability imply the existence of a good partition:

Open Problem 7.5. Let P be a property testable with r queries. Is it true that we can partition P into subsets $P_1, \dots, P_k$ such that P is P_i-partially testable using $O(1)$ queries for every $1 \leq i \leq k$ and k is bounded by some moderate function of k and r?

Theorem 5.1 implies that for some "hard" properties, a decomposition may have to be as large as $2^{\Theta(n/q)}$. It is not clear whether this value of k can always be obtained. The trivial upper bound for every property is by partitioning into 2^{n-q} subsets of size 2^q. Are there properties for which this is required?

Open Problem 7.6. Does there exist a "natural" property P such that for every $P' \subseteq P$ where P is P'-partially testable with q queries we also have $|P'| \leq |P| 2^{\Theta(q)-\Theta(n)}$?

In [GR15] there is a non-constructive proof (by counting the number of possible algorithms) that there is a property that is not partitionable to less than $2^{\Theta(n)-\Theta(q)}$ properties admitting partial tests with q queries, but the question of a constructive proof is still open.

7.3 Distribution testing with conditional samples

This model also saw an independent introduction by Cannone et. al. [CRS14] and several follow-up works [RT14, ACK14, Can15], yet there are many basic questions left unanswered.

The first order of business is closing the gaps between upper and lower bounds in Chapter 6.

Open Problem 7.7. Close the gaps between the upper and lower bounds for non-adaptive uniformity testing and adaptive testing of label invariant properties.

Another interesting direction is that of demanding certain structure for the conditioned upon sets. Cannone et. al. [CRS14] considered the case where the subsets must be intervals, and our upper bound for label invariant properties also only demands conditioning upon dyadic intervals.

Open Problem 7.8. What other results can be obtained where the conditioned upon sets must have a certain structure? Possible structures are intervals in an order, affine subspaces of a vector space, subgroups of a group, etc.

Chapter 6 deals with sampling oracles that are guaranteed to give independent samples. It would be interesting to see if anything can be discerned when this is no longer guaranteed.

Open Problem 7.9. What results can be obtained for distribution sampling with non-i.i.d. oracles? One possible model can be where the sampling is the current state of a markov chain.

Bibliography

[ABI86] Noga Alon, László Babai, and Alon Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *Journal of Algorithms*, 7(4):567–583, 1986.

[ACK14] Jayadev Acharya, Clément Canonne, and Gautam Kamath. A chasm between identity and equivalence testing with conditional queries. *arXiv preprint arXiv:1411.7346*, 2014.

[AFKS00] Noga Alon, Eldar Fischer, Michael Krivelevich, and Mario Szegedy. Efficient testing of large graphs. *Combinatorica*, 20(4):451–476, 2000.

[AFNS09] Noga Alon, Eldar Fischer, Ilan Newman, and Asaf Shapira. A combinatorial characterization of the testable graph properties: it's all about regularity. *SIAM Journal on Computing*, 39(1):143–167, 2009.

[AKNS00] Noga Alon, Michael Krivelevich, Ilan Newman, and Mario Szegedy. Regular languages are testable with a constant number of queries. *SIAM Journal on Computing*, 30(6):1842–1862, 2000.

[AS08] Noga Alon and Joel H. Spencer. *The probabilistic method.* Wiley-Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons Inc., Hoboken, NJ, third edition, 2008.

[BBM12] Eric Blais, Joshua Brody, and Kevin Matulef. Property testing lower bounds via communication complexity. *Computational Complexity*, 21(2):311–358, 2012.

[BCL$^+$06] Christian Borgs, Jennifer Chayes, László Lovász, Vera T Sós, Balázs Szegedy, and Katalin Vesztergombi. Graph limits and parameter testing. In *Proceedings of the 38th Annual Symposium on Theory of Computing*, pages 261–270. ACM, 2006.

[BDKR05] Tugkan Batu, Sanjoy Dasgupta, Ravi Kumar, and Ronitt Rubinfeld. The complexity of approximating the entropy. *SIAM Journal on Computing*, 35(1):132–150, 2005.

[BFH$^+$13] Arnab Bhattacharyya, Eldar Fischer, Hamed Hatami, Pooya Hatami, and Shachar Lovett. Every locally characterized affine-invariant property is testable. In *Proceedings of the 45th Annual Symposium on Theory of Computing*, pages 429–436. ACM, 2013.

[BFR$^+$10] Tugkan Batu, Lance Fortnow, Ronitt Rubinfeld, Warren D. Smith, and Patrick White. Testing closeness of discrete distributions. *CoRR*, abs/1009.5397, 2010. Extended abstract appeared in the proceedings of the 41nd Annual Symposium on Foundations of Computer Science, pages 259–269. ACM, 2000.

[BGS13] Arnab Bhattacharyya, Elena Grigorescu, and Asaf Shapira. A unified framework for testing linear-invariant properties. *Random Structures & Algorithms*, 2013.

[BKR04] Tugkan Batu, Ravi Kumar, and Ronitt Rubinfeld. Sublinear algorithms for testing monotone and unimodal distributions. In *Proceedings of the 36th Annual Symposium on Theory of Computing*, pages 381–390. ACM, 2004.

[BLR93] Manuel Blum, Michael Luby, and Ronitt Rubinfeld. Self-testing/correcting with applications to numerical problems. *Journal of Computer and System Sciences*, 47(3):549–595, 1993.

[BSGH$^+$06] Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil P. Vadhan. Robust PCPs of proximity, shorter PCPs, and applications to coding. *SIAM Journal on Computing*, 36(4):889–974, 2006.

[BSHR05] Eli Ben-Sasson, Prahladh Harsha, and Sofya Raskhodnikova. Some 3CNF properties are hard to test. *SIAM Journal on Computing*, 35(1):1–21, 2005.

[Can15] Clément Canonne. Big data on the rise: Testing monotonicity of distributions. *arXiv preprint arXiv:1501.06783*, 2015.

[CFL$^+$07] Sourav Chakraborty, Eldar Fischer, Oded Lachish, Arie Matsliah, and Ilan Newman. Testing st-connectivity. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 380–394. Springer, 2007.

[CGR$^+$14] Artur Czumaj, Oded Goldreich, Dana Ron, C. Seshadhri, Asaf Shapira, and Christian Sohler. Finding cycles and trees in sublinear time. *Random Struct. Algorithms*, 45(2):139–184, 2014.

[CRS14] Clement Cannone, Dana Ron, and Rocco Servedio. Testing equivalence between distributions using conditional samples. In *ACM-SIAM Symposium on Discrete Algorithms*, 2014. Extended version published as ECCC TR12-155.

[CS10] Artur Czumaj and Christian Sohler. Testing expansion in bounded-degree graphs. *Combinatorics, Probability and Computing*, 19(5-6):693–709, 2010.

[CT00] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley-Interscience, New York, USA, 2000.

[DG08] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[Edm65] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of mathematics*, 17(3):449–467, 1965.

[ER60] Paul Erdős and Richard Rado. Intersection theorems for systems of sets. *Journal of the London Mathematical Society*, 35:85–90, 1960.

[Fis04a] E. Fischer. The art of uninformed decisions:A primer to property testing. *Current Trends in Theoretical Computer Science: The Challenge of the New Century*, I:229–264, 2004.

[Fis04b] Eldar Fischer. The art of uninformed decisions: A primer to property testing. *Current Trends in Theoretical Computer Science: The Challenge of the New Century*, I:229–264, 2004.

[Fis05] Eldar Fischer. The difficulty of testing for isomorphism against a graph that is given in advance. *SIAM Journal on Computing*, 34(5):1147–1158, 2005.

[FLM$^+$12] Eldar Fischer, Oded Lachish, Arie Matsliah, Ilan Newman, and Orly Yahalom. On the query complexity of testing orientations for being eulerian. *ACM Transactions on Algorithms*, 8(2):15, 2012.

[FLN$^+$02] Eldar Fischer, Eric Lehman, Ilan Newman, Sofya Raskhodnikova, Ronitt Rubinfeld, and Alex Samorodnitsky. Monotonicity testing over general poset domains. In *Proceedings of the 34th Annual Symposium on Theory of Computing*, pages 474–483. ACM, 2002.

[FM08] Eldar Fischer and Arie Matsliah. Testing graph isomorphism. *SIAM Journal on Computing*, 38(1):207–225, 2008.

[FNS04] Eldar Fischer, Ilan Newman, and Jiri Sgall. Functions that have read-twice constant width branching programs are not necessarily testable. *Random Structures & Algorithms*, 24(2):175–193, 2004.

[FY11] Eldar Fischer and Orly Yahalom. Testing convexity properties of tree colorings. *Algorithmica*, 60(4):766–805, 2011.

[GGR98] Oded Goldreich, Shafi Goldwasser, and Dana Ron. Property testing and its connection to learning and approximation. *Journal of the ACM*, 45(4):653–750, 1998.

[GGR15] Oded Goldreich, Tom Gur, and Ron D. Rothblum. Proofs of proximity for context-free languages and read-once branching programs. *Electronic Colloquium on Computational Complexity (ECCC)*, 22:24, 2015.

[Gil52] Edgar N. Gilbert. A comparison of signalling alphabets. *The Bell System Technical Journal*, 31(3):504–522, May 1952.

[Gmaa] Gmail now has 425 million users, google apps used by 5 million businesses and 66 of the top 100 universities. `http://techcrunch.com/2012/06/28/gmail-now-has-425-million-users-google-apps-used-by-5-million-businesses-and-66-of-the-top-100-universities/`. Accessed: 2014-12-8.

[Gmab] See your storage limit. `https://support.google.com/mail/answer/6558?hl=en`. Accessed: 2014-12-8.

[GMV09] Sudipto Guha, Andrew McGregor, and Suresh Venkatasubramanian. Sublinear estimation of entropy and information distances. *ACM Transactions on Algorithms*, 5(4), 2009.

[Gol10a] Oded Goldreich. A brief introduction to property testing. In Oded Goldreich, editor, *Property Testing*, pages 1–5. Springer-Verlag, 2010.

[Gol10b] Oded Goldreich, editor. *Property Testing - Current Research and Surveys [outgrow of a workshop at the Institute for Computer Science (ITCS) at Tsinghua University, January 2010]*, volume 6390 of *Lecture Notes in Computer Science*. Springer, 2010.

[GR97] Oded Goldreich and Dana Ron. Property testing in bounded degree graphs. In *Proceedings of the 29th Annual Symposium on Theory of Computing*, pages 406–415. ACM, 1997.

[GR99] Oded Goldreich and Dana Ron. A sublinear bipartiteness tester for bounded degree graphs. *Combinatorica*, 19(3):335–373, 1999.

[GR11] Oded Goldreich and Dana Ron. On testing expansion in bounded-degree graphs. In Oded Goldreich, editor, *Studies in Complexity and Cryptography*, volume 6650 of *Lecture Notes in Computer Science*, pages 68–75. Springer, 2011.

[GR13] Oded Goldreich and Dana Ron. On sample-based testers. *Electronic Colloquium on Computational Complexity*, (109), 2013. ECCC TR13-109.

[GR15] Tom Gur and Ron D. Rothblum. Non-interactive proofs of proximity. In Tim Roughgarden, editor, *Proceedings of the 6th Innovations in Theoretical Computer Science Conference*, pages 133–142. ACM, 2015.

[GS14] Oded Goldreich and Igor Shinkar. Two-sided error proximity oblivious testing. *Random Structures & Algorithms*, 2014.

[Hås01] Johan Håstad. Some optimal inapproximability results. *Journal of the ACM*, 48(4):798–859, 2001.

[HLNT07] Shirley Halevy, Oded Lachish, Ilan Newman, and Dekel Tsur. Testing properties of constraint-graphs. In *Proceedings of the 22nd Conference on Computational Complexity*, pages 264–277. IEEE, 2007.

[Hoe63] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58:13–30, 1963.

[KL51] Solomon Kullback and Richard A. Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, March 1951.

[KM93] P. Kilpelainen and H. Mannila. Retrieval from hierarchical texts by partial patterns. In *ACM SIGIR*, Structured Text, pages 214–222, 1993.

[KS08] Tali Kaufman and Madhu Sudan. Algebraic property testing: the role of invariance. In *Proceedings of the 40th Annual Aymposium on Theory of Computing*, pages 403–412. ACM, 2008.

[LRR10] Reut Levi, Dana Ron, and Ronitt Rubinfeld. Testing properties of collections of distributions. In Bernard Chazelle, editor, *Proceedings of the 1st Symposium on Innovations in Computer Science*, pages 179–194, Beijing, China, January 5–7 2010.

[MS77] F. Jessie MacWilliams and Neil J. A. Sloane. *The Theory of Error-Correcting Codes*, volume 16 of *North-Holland Mathematical Library*. North-Holland, 1977.

[MST72] F. Jessie MacWilliams, Neil J. A. Sloane, and John G. Thompson. Good self dual codes exist. *Discrete Math.*, 3:153–162, 1972.

[Mut05] S Muthukrishnan. *Data streams: Algorithms and applications*. Now Publishers Inc, 2005.

[New02] Ilan Newman. Testing membership in languages that have small width branching programs. *SIAM J. Comput.*, 31(5):1557–1570, 2002.

[New10] Ilan Newman. Property testing of massively parametrized problems - A survey. In Oded Goldreich, editor, *Property Testing*, volume 6390 of *Lecture Notes in Computer Science*, pages 142–157. Springer, 2010.

[NNL13] Antoine M. Ndione, Joachim Niehren, and Aurelien Lemay. Sublinear DTD Validity. Research report, March 2013.

[Pin64] Mark S. Pinsker. *Information and information stability of random variables and processes*. Holden-Day Inc., San Francisco, Calif., 1964. Translated and edited by Amiel Feinstein.

[Ron08] Dana Ron. Property testing: A learning theory perspective. *Foundations and Trends in Machine Learning*, 1(3):307–402, 2008.

[Ron09] Dana Ron. Algorithmic and analysis techniques in property testing. *Foundations and Trends in Theoretical Computer Science*, 5(2):73–205, 2009.

[RRSS09] Sofya Raskhodnikova, Dana Ron, Amir Shpilka, and Adam Smith. Strong lower bounds for approximating distribution support size and the distinct elements problem. *SIAM Journal on Computing*, 39(3):813–842, 2009.

[RS96] Ronitt Rubinfeld and Madhu Sudan. Robust characterizations of polynomials with applications to program testing. *SIAM Journal on Computing*, 25(2):252–271, 1996.

[RT14] Dana Ron and Gilad Tsur. The power of an example: Hidden set size approximation using group queries and conditional sampling. *arXiv preprint arXiv:1404.5568*, 2014.

[RVW13] Guy N. Rothblum, Salil Vadhan, and Avi Wigderson. Interactive proofs of proximity: delegating computation in sublinear time. In *Proceedings of the 45th Annual Symposium on Theory of Computing*, pages 793–802. ACM, 2013.

[RW09] Mark D. Reid and Robert C. Williamson. Generalised pinsker inequalities. In *Proceedings of the 22nd Annual Conference on Learning Theory*, 2009.

[SN00] Torsten Schlieder and Felix Naumann. Approximate tree embedding for querying XML data. In *SIGIR Workshop On XML and Information Retrieval*. ACM, 2000.

[Sud11] Madhu Sudan. Invariance in property testing. In *Property testing*, pages 211–227. Springer, 2011.

[Sze99] Mario Szegedy. Many-valued logics and holographic proofs. In *Proceedings of the 26th International Colloquium on Automata, Languages and Programming*, pages 676–686. Springer, 1999.

[Val05] Gabriel Valiente. Constrained tree inclusion. *Journal of Discrete Algorithms*, 3(2-4):431–447, 2005.

[Val11] Paul Valiant. Testing symmetric properties of distributions. *SIAM Journal on Computing*, 40(6):1927–1968, 2011.

[Var57] Rom R. Varshamov. Estimate of the number of signals in error correcting codes. In *Doklady Akademii Nauk SSSR*, volume 117, pages 739–741, 1957.

[Yah08] Orly Yahalom. *Topics in Property Testing over Massively Parameterized Models*. PhD thesis, Technion — Israel Institute of Technology, 2008. http://www.graduate.technion.ac.il/theses/Abstracts.asp?Id=24498.

[Yah12] Orly Yahalom. Testing for forbidden posets in ordered rooted forests. *International Journal of Foundations of Computer Science*, 23(6):1405, 2012.

אלגוריתמי בדיקת תכונות בעלי סיבוכיות שאילתות טובה, אך ניתן לפרק אותן למספר קטן יחסית של מקרים עבורם כן קיימים אלגוריתמי בדיקת תכונות בעלי סיבוכיות שאילתות נמוכה.

אנחנו מוכיחים כי זה אינו המקרה הכללי, כלומר קיימות בעיות שלא ניתן לפרק למספר קטן של מקרים עבורם קיים אלגוריתם בדיקת תכונות בעל סיבוכיות שאילתות נמוכה. לצורך כך אנחנו מפתחים טכניקות חדשות, מבוססות טיעוני אנטרופיה, להוכחת חסמים תחתונים על סיבוכיות שאילתות של בעיות בדיקת תכונות. בכך למעשה נוצר ציר נוסף של מדידת קושי של בעיות בדיקת תכונות. כפי שניתן לשאול מה סיבוכיות השאילתות של פתרון בעיה מסויימת, כעת ניתן גם לשאול כיצד משתנה סיבוכיות זו עם החלוקה לתתי תכונות.

עבור בעיות כלליות, אנחנו מראים כי עבור מחלקה מוגבלת של אלגוריתמי בדיקת תכונות, "אלגוריתמים האדישים למרחק", קיומו של פירוק למספר קטן של מקרים הניתנים לבדיקה ביעילות מאפשר לפתח אלגוריתם בדיקת תכונות בעל סיבוכיות שאילתות תת לינארית עבור כל המקרים. הדבר מבוסס על בניה של "בודק אוניברסלי", ומתוך כך עונה על שאלה פתוחה בדבר "בודקים מבוססי דגימות" שהועלתה על ידי גולדרייך ורון.

כמו כן, אנחנו מראים תכונה פשוטה שלא ניתנת לבדיקה יעילה, אבל ניתן לפרקה בקלות למעט תתי תכונות שאת כל אחת מהן ניתן לבדוק במספר שאילתות מועט.

חלק ד׳ - בדיקת תכונות של הסתברויות עם אוב לדגימות מותנות

סוג נוסף של בעיות בדיקת תכונות הוא בדיקת תכונות של התפלגויות. במקרה זה הקלט הוא התפלגות בלתי ידועה, ובמקום לקרוא ביטים מהקלט מתאפשר רק לבצע דגימות לפי ההתפלגות. קיימת ספרות ענפה בתחום החוקרת מגוון של תכונות כגון אחידות, קרבה להתפלגות ידועה ומונוטוניות, וכן מקרבת פרמטרים כגון אנטרופיה וגודל התומך. יש לציין שמכיוון שהבחירה היחידה של האלגוריתם ביחס לקלט היא מספר הדגימות שנלקחות ממנו, אין מקום לעושר אלגוריתמי.

אנחנו מציעים מודל עשיר יותר, בו ניתן לדגום לפי ההתפלגות כאשר היא מותנית על תת קבוצה של העולם לבחירת האלגוריתם. זהו מודל טבעי לבעיות רבות, לדוגמא בעת קיום סקרי דעת קהל הנדגמים תוך התניה על היותם בקבוצה דמוגרפית מסויימת. אנחנו מוכיחים כי במודל עם אוב לדגימות מותנות בעיות רבות שידועות כקשות במודל הרגיל הופכות לקלות, כגון בדיקת אחידות ובדיקת קרבה להתפלגות נתונה, וכן המשפחה הכללית של תכונות סימטריות של התפלגויות. מצד שני, אנחנו מוכיחים כי אף במודל זה ישנן בעיות קשות לפתרון שדורשות מספר דגימות לינארי.

לבסוף, במודל עם אוב לדגימות מותנות יש מקום להבדיל בין אלגוריתמים אדפטיביים לאילו שאינם אדפטיביים. אלגוריתם אדפטיבי בוחר את קבוצות הדגימה במהלך ריצתו, בעוד שאלגוריתם שאינו אדפטיבי בוחר את תת הקבוצות עליהן יותנו הדגימות לפני שקיבל כל תוצאה של דגימה. יש לציין שאין משמעות להבחנה כזאת באלגוריתמי בדיקת תכונות של התפלגויות במודל הרגיל. אנחנו מוכיחים כי המודל האדפטיבי חזק מהמודל הלא אדפטיבי, אך גם כי המודל הלא אדפטיבי חזק מהמודל חסר הדגימות המותנות.

תזה זאת מתמקדת במקרים בהם אין די בבחירה אקראית אחידה. במקרים בהם לקלט מבנה מורכב ומשתנה, לרוב נדרשת גם תובנה אלגוריתמית בבחינת הקלט, ואין מספיק בבחירה אחידה. התזה עוסקת במספר מקרים בהם אלגוריתמי בדיקת תכונות מתבססים על תובנות אלגוריתמיות כדי לקרוא רק חלקים זניחים מהקלט, במקרים בהם מודל עשיר יותר מאפשר נקודת מבט אלגוריתמית מתוחכמת יותר, ובאפשרויות של גישות אלגוריתמיות שונות לבדיקת תכונות.

אחד התחומים המפתים ביותר להפעלת שיטות אלגוריתמיות בבדיקת תכונות הוא בדיקת תכונות עם פרמטר מסיבי. בתחום זה התכונה נקבעת על בסיס פרמטר נתון, אך פרמטר זה הוא מסדר גודל של הקלט עצמו. כך, מכיוון שהתכונה נקבעת לפי הפרמטר, חזקה על האלגוריתם לבחון אותו. התלות המהותית במבנהו של הפרמטר שוללת אפשרות של דגימה אחידה.

חלק א׳ - בדיקת סיפוק נוסחאות

מקרה ראשון של בדיקת תכונות עם פרמטר מסיבי שנלמד בתזה זו הוא בדיקת סיפוק נוסחאות. בבעיה כזאת נתון פרמטר מסיבי שהוא נוסחה לוגית, המתוארת על ידי עץ שצמתיו הפנימיים הם שערים לוגיים, ועליו הם קלטים לנוסחה. נוסחה כזאת מגדירה כתכונה את הקלטים שיוביאו לפלט "אמת" בשורש העץ. מדובר במשפחה עשירה של בעיות המוגדרות לפי אופי השערים המותר וגודל הא"ב, וקשורה באופן טבעי למושגים מתורת הסיבוכיות.

אנחנו חוקרים את הבעיה במקרה של א"ב בינארי ושערים מכלליות עולה - שערי "או" ו-"גם", שערים מונוטוניים מרוחב חסום, וכן שערים כלליים מרוחב חסום, ומפתחים אלגוריתמים יעילים לכל המקרים הללו. כמו כן, אנחנו מוכיחים חסם תחתון כנגד בדיקת סיפוק נוסחאות מעל א"ב מגודל 4, וכן נוסחאות מעל א"ב מגודל 5 עם שערים מונוטוניים בלבד.

חלק ב׳ - בדיקת תכונות של צביעות עץ

מקרה נוסף של בדיקת תכונות עם פרמטר מסיבי בתזה זו היא בדיקת תכונות של צביעות עץ. תכונות כאלה מוגדרת על ידי עץ, ואילוץ מסויים על פונקציות צביעה עליו מאוסף צבעים מסויים. במקרה זה הפרמטר המסיבי הוא תיאור העץ, והקלט לאלגוריתם הוא צביעה של צמתיו. כך, תכונה הנבדקת היא תכונה של צביעות אפשריות של עץ נתון הנקבע מראש.

אנחנו מתרכזים בתכונות צביעות עץ המוגדרות על ידי מינורים טופולוגיים אסורים בצביעה, ומפתחים אלגוריתם לבדיקת תכונות כאלה. בפרט, אנחנו מפתחים אלגוריתם עבור בדיקת תכונות המוגדרות על ידי אוסף של מינורים טופולוגיים אסורים. יש בכך עניין מיוחד מעבר לתכונות של מינור טופולוגי אסור יחיד, שכן חיתוך של תכונות בדיקות אינו בהכרח בדיק. כמו כן, תוצאה זו גוררת תוצאה דומה עבור הבעיה של בדיקת תכונות המוגדרות על ידי אוסף של מינורים (כלליים) אסורים, שהועלתה בתיזת הדוקטורט של יהלום.

לבסוף, אנחנו מוכיחים חסם תחתון עבור מקרה קרוב של בעיות צביעת עצים, בהן האילוץ הוא של משפחת תתי עצים מושרים צבועים אסורים.

חלק ג׳ - פירוק של תכונות

אחת הגישות האלגוריתמיות הבסיסיות ביותר להתמודדות עם בעיה קשה היא ללמוד מקרים פרטיים שונים שלה, ולאחר מכן לאחד את כלל המקרים לכדי פתרון כולל. אנו חוקרים את האפשרות של גישה כזאת בבעיות בדיקת תכונות. הדבר מפתה במיוחד שכן קיימות בעיות בדיקת תכונות רבות מאוד שלא קיימים להם

תקציר

כאשר מודדים יעילות של חישוב, בדרך כלל נבחן את קצב גידולו של משאב הנדרש על ידי החישוב ביחס לגדילת הקלט. ברור כי ככל שקצב גידול זה נמוך יותר, כך שימוש החישוב במשאב הנידון יעיל יותר. כך, מטרת תורת האלגוריתמים היא פיתוח אלגוריתמים לבעיות חשובות ששימושן במשאבים הוא נמוך ככל האפשר. בין המשאבים הנידונים בדרך כלל נמנים זמן החישוב (סיבוכיות זמן), וכמות הזכרון הנדרשת עבורו (סיבוכיות מקום). בתורת הסיבוכיות של חישובים מקובל כי סיבוכיות זמן שגדלה פולינומיאלית בגודל הקלט היא יעילה.

בשנים האחרונות עולם המחשוב מתמודד עם כמויות מידע מסדרי גודל אדירים, גדולים בהרבה מאלה שנתקל בהם בעבר, וזאת באופן תדיר ויום־יומי בתחומים מגוונים החל מפרסום באינטרנט וכלה במחקר רפואי. עבור בעיות אלה חישוב בסיבוכיות זמן פולינומיאלית הופך לממושך באופן לא סביר, ואף חישוב בסיבוכיות זמן לינארית הופך לבלתי אפשרי. בעיות אלה הובילו את המחקר במדעי המחשב לבחון אלגוריתמים בסיבוכיות תת־לינארית, קרי, אלגוריתמים שקצב גידול זמן הריצה שלהם קצר מסדר גודלו של הקלט. אלגוריתמים כאלה מטבעם נדרשים להגיע להחלטה מבלי לקרוא את כל הקלט. אם אין כל הנחה נוספת על הקלט, ברור מכך שעל אלגוריתמים אלה להתבסס על אקראיות וכן להגיע להחלטות שהן מקורבות באופיין.

החל מתחילת שנות התשעים נלמדים במדעי המחשב אלגוריתמי בדיקת תכונות, אלגוריתמים אקראיים שמספר הביטים שהם קוראים קטן משמעותית מגודל הקלט. מכוון שאלגוריתמים כאלה לא קוראים את כל הקלט, הדרישה מהם היא מקורבת – אלגוריתם לבדיקת תכונה מסויימת נדרש לקבל בהסתברות טובה קלט המקיים את התכונה, ולדחות בהסתברות טובה קלט הרחוק מכל קלט המקיים את התכונה. על פי רוב הקלטים הם מחרוזות בינאריות או מבנים קומבינטוריים אחרים, והמרחק נמדד באחוז מתוך המבנה שיש לשנות על מנת שיקיים את התכונה. מספר הביטים הנדרש לאלגוריתם נקרא סיבוכיות השאילתות שלו.

השאיפה במחקר בתחום בדיקת התכונות היא לפתח אלגוריתמים עם סיבוכיות שאילתות קבועה. כלומר, בעלי מספר שאילתות התלוי אך ורק בפרמטר הקובע מה הוא המרחק המינימלי מעליו נדרש האלגוריתם לדחות, ובפרט אינו תלוי בגודל הקלט. אלגוריתם סביר בסיבוכיות שאילתות קבועה יהיה בדרך כלל גם בעל סיבוכיות זמן קבועה. עם זאת, כל סיבוכיות שאילתות הנמוכה מלינארית מספקת תובנה יקרת ערך, שכן משמעות הדבר שניתן להגיע להחלטה משמעותית מבלי לקרוא את הקלט כולו. כלומר, תוך קריאת חלק זניח מהקלט אנחנו לומדים דבר מה משמעותי על כללותו. החסכנות בקריאת הקלט חשובה לא רק עבור קלטים גדולים מאוד, אלא גם כאשר עצם הפקת הקלט היא קשה מאוד, מכיוון שהיא מתבססת על מחקר אנושי, דגימה יקרה או כל פריט מידע אחר שהשגתו קשה.

עם תחילת המחקר בתחום התחוור במהרה כי רבות מהתוצאות בו מסתמכות למעשה על כך שהקלטים הרחוקים מקיום התכונה יכילו הפרות רבות של איזו תבנית מקומית שלה. כלומר, אם קלט רחוק מקיום התכונה באופן גלובלי, ישנה תבנית הפרות מקומית שנפוצה בו מאוד. לדוגמא, אם פונקציה רחוקה מלהיות לינארית, קיימות גם שלות רבות מהצורה $x+y=z$ עליהן היא אינה מקיימת את תכונת הלינאריות. כך, רבים מהאלגוריתמים הם למעשה בחירה אקראית של תת מבנה קטן של הקלט, ובחינה של כל הביטים של אותו תת המבנה כנגד איזו תבנית אסורה, או אוסף של תבניות כאלה. הדבר נחקר ואף הוסבר כתופעה כללית בחלק מתחומי העיסוק של בדיקת תכונות, כגון תכונות גרפים במודל הצפוף.

המחקר בוצע בהנחייתו של פרופסור אלדר פישר, בפקולטה למדעי המחשב.

חלק מן התוצאות בחיבור זה פורסמו כמאמרים מאת המחבר ושותפיו למחקר בכנסים ובכתבי־עת במהלך תקופת מחקר הדוקטורט של המחבר, אשר גרסאותיהם העדכניות ביותר הינן:

Sourav Chakraborty, Eldar Fischer, Yonatan Goldhirsh, and Arie Matsliah. On the power of conditional samples in distribution testing. In Robert D. Kleinberg, editor, *ITCS*, pages 561–580. ACM, 2013.

Eldar Fischer, Yonatan Goldhirsh, and Oded Lachish. Testing formula satisfaction. In Fedor V. Fomin and Petteri Kaski, editors, *SWAT*, volume 7357 of *Lecture Notes in Computer Science*, pages 376–387. Springer, 2012.

Eldar Fischer, Yonatan Goldhirsh, and Oded Lachish. Partial tests, universal tests and decomposability. In Moni Naor, editor, *ITCS*, pages 483–500. ACM, 2014.

תודות

ברצוני להודות למנחה שלי בעבודת הדוקטורט, פרופ' אלדר פישר, על תמיכתו העקבית לאורך שנות ההשתלמות שלי. אלה היו שנים מסעירות ומתגמלות, ואני בר מזל שאלדר היה מורה הדרך שלי לעולם המחקר. אני אסיר תודה לעמי פז, ששיחק תפקיד מפתח בהנעתי להגיע לכתיבת העבודה הזו. עמי עודד אותי כשהייתי בשפל, ועזר לי להפוך את התזה ממחסום בלתי עביר לפרוייקט מוגמר.

עבודה זו מוקדשת לאישתי האהובה, שרית, שתמיד תומכת בי בזמנים הקשים, וחוגגת איתי את ההצלחות.

הכרת תודה מסורה לטכניון על מימון מחקר זה.

אלגוריתמים לבדיקת תכונות ובעיות נוספות

חיבור על מחקר

לשם מילוי חלקי של הדרישות לקבלת התואר
דוקטור לפילוסופיה

יונתן גולדהירש

הוגש לסנט הטכניון – מכון טכנולוגי לישראל
אייר התשע"ה חיפה אפריל 2015

אלגוריתמים לבדיקת תכונות ובעיות נוספות

יונתן גולדהירש