𝕋ℂ𝕊-𝕋ℝ-𝔹-**09-6**

𝕋ℂ𝕊 Technical Report

Master's Thesis: The Classification Problem in Relational Property Testing

by

Charles Harold Jordan

Division of Computer Science

Report Series B

April 27, 2009

Hokkaido University
Graduate School of
Information Science and Technology

Email: skip@ist.hokudai.ac.jp

Phone: +81-011-706-7675

Fax: +81-011-706-7675

The Classification Problem in Relational Property Testing

Charles Harold Jordan
Division of Computer Science, Graduate School of Information Science and Technology
Hokkaido University, N-14, W-9, Sapporo 060-0814, Japan

Summary

In property testing, we desire to distinguish between objects that have a given property and objects that are *far* from the property by examining only a small, randomly selected portion of the objects. Property testing arose in the study of formal verification, however much of the recent work has been focused on testing *graph* properties.

In this thesis we introduce a generalization of property testing which we call *relational property testing.* Because property testers examine only a very small portion of their "inputs," there are potential applications to efficiently testing properties of massive structures. Relational databases provide perhaps the most obvious example of such massive structures, and our framework is a natural way to characterize this problem. We introduce a number of variations of our generalization and prove the relationships between them.

The second major topic of this thesis is the *classification problem for testability.* Using the general framework developed in previous chapters, we consider the testability of various syntactic fragments of first-order logic. This problem is inspired by the classical problem for decidability. We compare the current classification for testability with early results in the classification for testability, and then prove an additional class to be testable.

Contents

Chapter 1

Introduction

Property testing is an application of induction. Given a large object, for example a graph or database, we wish to state some conclusion about the entire structure after examining only a small, randomly selected sample. Lovász [31] has described it as the "third reincarnation" of this approach, after statistics and machine learning.

Property testers, which we formally define in Chapter 2, are probabilistic approximation algorithms that examine only a small part of their input. Our goal is always to distinguish inputs that have some desired property from inputs that are *far* from having it. We are especially interested in *classification*, i.e., considering the testability of large classes of properties.

There is an enormous amount of recent work in property testing. In the following section we introduce the history of the field, focusing particularly on results that influence our approach. Then, in Section 1.2 we summarize the results and structure of this thesis.

1.1 History of Property Testing

We begin with a brief history and overview of property testing. There are also a number of surveys of property testing, see for example Fischer [15] or Ron [39].

Property testing is a form of approximation where we trade accuracy for efficiency. Probabilistic machines appear to have been first formalized by de Leeuw *et al.* [30], who showed that such machines cannot compute deterministically uncomputable properties under reasonable assumptions. However, they explicitly mention the possibility that probabilistic machines could be more *efficient* than deterministic machines. An early example of such a result is the matrix multiplication checker of Freivalds [17].

Property testing itself is generally considered to have arisen from program verification (see Blum *et al.* [10] and Rubinfeld and Sudan [40]). Here we have some program $P(x)$ that purports to compute a function $f(x)$ and we wish to quickly verify that P is correct with high probability. Rubinfeld and Sudan [40] define the distance between two functions with the same domain and arity to be the fraction of the domain that they assign different outputs. They then define a *tester* for a set of functions to be a program that accepts

P if P is a program for some f in the set and rejects P with high probability if P is far from being a program for all f in the set.

We can consider a graph to be represented by a binary function $e(x, y)$ that is 1 if there is an edge from x to y and 0 otherwise, which is essentially an adjacency matrix. Then, any set of such functions is a graph property, allowing us to consider testable and untestable graph properties. This approach was first considered by Goldreich *et al.* [20], where they show the existence of testable NP-complete properties among many other results. However, if we are interested in properties of *bounded-degree* graphs, the adjacency matrix encoding is wasteful. A related approach using incidence lists to represent bounded-degree graphs has been studied by Goldreich and Ron [19]. Parnas and Ron [36] generalized this approach and attempted to move away from the functional representation of structures.

It is possible to consider properties of other structures, such as strings. Alon *et al.* [4] showed that although all regular languages are testable, there exist untestable context-free languages (see Theorem 4 below). Chockler and Kupferman [13] extended the positive result to the ω-regular languages. However, much of the recent work has been focused on graph property testing. Alon and Shapira [6] have written a survey of some of the recent results in graph testing.

If we consider graph property testing, Alon *et al.* [2] took the first step towards a *logical* characterization of the testable properties. They showed that all graph properties expressible by first-order sentences of the form "$\exists\forall$" are testable and that there exists a property expressible in the form "$\forall\exists$" that is not testable. This leads naturally to the classification problem for testability, which we consider in Chapter 4. Their positive result was obtained by showing that all such properties are essentially instances of a colorability problem, all of which they then showed to be testable. Fischer [16] showed various generalizations of this kind of colorability problem to also be testable.

Later, Alon and Shapira [5] gave a (near) characterization of the graph properties testable with one-sided error by algorithms unaware of the input size, a result that was generalized to hypergraphs by Rödl and Schacht [38]. Alon *et al.* [3] obtained an exact combinatorial characterization of the graph properties testable with a constant number of queries.

1.2 Thesis Overview

In this thesis we particularly focus on two issues in property testing. First, much of the recent work in testing has been focused on graph properties. In contrast, we seek a more general framework, which we call *relational property testing.* We introduce definitions and notation in Chapter 2 and then show a number of basic results in Chapter 3.

There are several possible variations of our framework. We prove the relationships between several such variations in Section 3.7. It is also possible to consider applications of property testing. In particular, we could be interested in efficiently testing properties of massive structures such as relational databases, a problem which is characterized by our framework. Properties of databases are generally defined in formal query languages such as SQL and so it is natural to consider the testability of such languages.

The second major topic in this thesis is the *classification problem for testability*, which we consider in Chapter 4. The objective here is to provide a classification of exactly which classes of first-order logic are entirely testable and which classes are not, which we consider in the framework of relational property testing developed in earlier chapters. We provide an overview of the currently known results for this classification and compare them with the classical results in the classification for decidability. We also show the testability of Ackermann's class with equality, providing an additional parallel to the classical case.

Chapter 2

Preliminaries

The notations and definitions that we require are divided into several topics. We consider property testing in a very general setting instead of restricting ourselves to some particular type of structure such as graphs.

In order to discuss properties of graphs or of strings, it is necessary to first define these types of structures. We therefore define fundamental notions such as vocabularies (types of structures) and structures in Section 2.1. This provides a sufficient basis to formalize our definitions of property testing, which we do in Section 2.2. This thesis is particularly focused on questions of formal logic in property testing, and so we give definitions related to logic in Section 2.3 and those used for discussing the classification problem in Section 2.4.

Before proceeding further, we recall fundamental definitions and introduce notation for familiar objects such as natural numbers, sets and strings. Our definitions are standard and readers familiar with this material can safely skip to Section 2.1.

The natural numbers are denoted by $\mathbb{N}$ and are the set of non-negative integers. We denote the set of real numbers by $\mathbb{R}$, although these are generally used for probabilities and so we usually use only real numbers $p \in [0, 1]$. We use bold characters to denote vectors, for example $\mathbf{x} \in \mathbb{R}^3$. Vectors are row vectors unless otherwise noted, we denote the *transpose* of a vector by $\mathbf{x}^T$. If $\mathbf{x} = (x_1, \ldots, x_a)$ is a vector, we call x_i the i-th *component* of $\mathbf{x}$.

The empty set is denoted by $\emptyset$. If A and B are sets, then the union of A and B is $A \cup B := \{x \mid x \in A \text{ or } x \in B\}$, the intersection of A and B is $A \cap B := \{x \mid x \in A \text{ and } x \in B\}$, and the set difference of A and B is $A \backslash B := \{x \mid x \in A \text{ and } x \notin B\}$. We generalize the union and intersection in the usual way, $\bigcup_{i \geq 0} A_i := A_0 \cup A_1 \cup \ldots$ and $\bigcap_{i \geq 0} A_i := A_0 \cap A_1 \cap \ldots$ respectively.

Set A is a subset of set B, written $A \subseteq B$ if $A \backslash B = \emptyset$. Set A is a *proper* subset of set B, written $A \subset B$ if $A \subseteq B$ and $B \backslash A \neq \emptyset$. The cardinality of a set A is the number of elements in the set, written $|A|$.

The product of sets A and B is the set of ordered pairs, $A \times B := \{(a, b) \mid a \in A \text{ and } b \in B\}$. The set of n-tuples of set A, written A^n is defined inductively as follows.

First, $A^1 = A$. Then, $A^{n+1} = A^n \times A$. We will always omit the extra parentheses, and so $(1, 2, 3)$ denotes $((1, 2), 3)$. The number of elements in the tuple is the *arity* n. A *predicate* P with arity n of set A is any subset of A^n. If $\mathbf{x} \in A^n$, we will generally abbreviate the proposition $\mathbf{x} \in P$ with $P(\mathbf{x})$.

An alphabet Σ is a set of symbols, and a string w over Σ is some sequence of the symbols in Σ. The empty string is denoted by λ. For example, $\{0, 1\}$ is the alphabet of binary strings and 0100 is an example of such a string. We number the positions in a string w from left to right with $0, 1, \ldots, n-1$ where n is the length of the string. Of course, the empty string λ has length 0. As usual, Σ^* is the free monoid of Σ and any subset $L \subseteq \Sigma^*$ of it is a *language*.

Let w be a string over the alphabet Σ. The *concatenation* of strings u and v is uv, while the *product* of two sets of strings L_1 and L_2 is $L_1L_2 := \{uv \mid u \in L_1 \text{ and } v \in L_2\}$. The *reversal* of w is written $\overleftarrow{w}$. Position i of $\overleftarrow{w}$ corresponds to position $n - 1 - i$ of w. Formally, $\overleftarrow{\lambda} = \lambda$ and $\overleftarrow{aw} = \overleftarrow{w}a$ for $a \in \Sigma$.

We mention a number of well-known classes of languages, for example the classes of regular and context-free languages. Hopcroft and Ullman [26] is a well-known introduction to these classes.

It is natural to represent a binary string $w \in \{0, 1\}^*$ as a pair $\{U, \mathcal{S}\}$ where U is the finite set of bit positions $0, \ldots, n - 1$[1] and $\mathcal{S} \subseteq A$ is a monadic *predicate*. We will define $\mathcal{S}(i)$ to mean that "bit position i of w is 1."

Graphs provide another natural example and allow for representation as a pair, $(V, \mathcal{E})$. Here V is the set of vertices and the edge set $\mathcal{E} \subseteq V^2$, a set of ordered pairs of V. The "names" of the vertices are not interesting to us, and we will identify them as $0, \ldots, n-1$ where n is the number of vertices. It is therefore natural to represent a graph as a pair $\{V, \mathcal{E}\}$ where $\mathcal{E}$ is a binary predicate over V.

We will formalize these notions more exactly in the following section. In particular, one of our goals is a generalized notion of property testing instead of restricting ourselves to fixed kinds of structures such as graphs and binary strings. The definitions in the following section are therefore necessarily abstractions of the ideas above.

2.1 Basic Definitions

Instead of restricting our attention to, for example, graphs, we focus on property testing in a general setting. We begin by defining vocabularies, which will also be the basis for most of our logical definitions. A *predicate symbol* is simply a syntactic character which is used to refer to predicates. Likewise, the arities in the following definition are simply positive integers that are later interpreted as arities.

Definition 1. *A* vocabulary τ *is a tuple of distinct predicate symbols* R_i *together with their arities* a_i*,*

$$\tau := (R_1^{a_1}, \ldots, R_s^{a_s}).$$

[1]The universe is the empty set if w is the empty string.

When we prove general theorems for all vocabularies, we will always use vocabularies τ as in Definition 1. The predicate symbols in these theorems will therefore always be named R_i for $1 \leq i \leq s$ and have arities a_i. Two examples of vocabularies are $\tau_G := \{E^2\}$, the vocabulary of directed *graphs* and $\tau_S := \{S^1\}$, the vocabulary of *binary strings*.

In order to define structures we must consider universes, which are sets. Our universes are always finite and we generally refer to the cardinality of a universe with n.

Definition 2. *An* (algebraic) structure A of type τ *is an* $(s+1)$*-tuple*

$$A := (U, \mathcal{R}_1^A, \ldots, \mathcal{R}_s^A)$$

consisting of a finite universe U and where each $\mathcal{R}_i^A \subseteq U^{a_i}$ is a predicate corresponding to the predicate symbol R_i.

In the following, we omit "(algebraic)" and refer to such structures simply as *structures*. We also omit the vocabulary when it is understood. If the universe U is the empty set, then $n = 0$ and the structure is the unique *empty structure of type* τ. Such structures are not very interesting in terms of testing but it is worth noting their existence.

For convenience we will always identify the elements of U with the non-negative integers $\{0, \ldots, n-1\}$ and use $n = \#(A)$ for the size of the universe of a structure A. As a notational convenience, we will use $U_n := \{0, 1, \ldots, n-1\}$ to refer to "the" universe of n elements. We use calligraphic characters to denote the predicates defined by a structure, however our logical definitions will provide a more convenient way (using the predicate symbols) to state propositions regarding the predicates.

If we consider a binary string, the universe U is the set of bit positions, which we will identify as $\{0, \ldots, n-1\}$ from left to right. For $i \in U$, we interpret the meaning of $i \in \mathcal{S}$ as "bit i of the string is 1." Likewise, for a graph G, the universe is the set of vertices, which we again identify as $\{0, \ldots, n-1\}$. For $x, y \in U$, we interpret $(x, y) \in \mathcal{E}$ as "there is an edge from x to y in the graph." Our graphs are therefore directed and possibly contain loops.

We define $STRUC^n(\tau)$ to be the set of all structures with vocabulary τ and universe size n. We then define $STRUC(\tau) := \bigcup_{0 \leq n} STRUC^n(\tau)$ to be the set of all structures of type τ.

A *property P of structures of type* τ is a set of structures of type τ, and so $P \subseteq STRUC(\tau)$. We do not consider properties of structures of mixed types, although it would be possible to do so. For all $A \in P$, we say that structure A *has* property P. That is, in order to avoid unwieldy language, *having* a property is always defined as membership in the set of structures defining the property. To reflect the conventional terminology from formal language theory, we use *language* instead of "property" to refer to sets of strings.

An example of a property is that of being a binary palindrome. A binary string is a palindrome if it is equivalent to its reversal. The language of such binary strings can be defined as

$$L_P := \{w \mid w \in STRUC(\tau_S) \text{ and } \mid \forall i \in U : i \in \mathcal{S} \text{ iff } (\#(w) - 1 - i) \in \mathcal{S}\}.$$

We generally use L, P and Q to denote properties and A and B to refer to structures. However, we refer to strings as u,v and w to reflect more common notation.

2.2 Property Testing Definitions

In property testing we wish to distinguish, with high probability, between inputs that have some desired property and inputs that are *far* from having the property. We begin by defining a distance measure between structures. The symbol $\oplus$ denotes exclusive-or. We recall that vocabulary $\tau = \{R_1^{a_1}, \dots, R_s^{a_s}\}$ and that the universe of a structure is denoted by U (cf. Definition 1).

Definition 3. *Let $A, B \in STRUC(\tau)$ be structures such that $\#(A) = \#(B) = n$. The* distance *between structures A and B is*

$$\mathrm{dist}(A, B) := \frac{\sum_{1 \le i \le s} |\{\mathbf{x} \mid \mathbf{x} \in U^{a_i} \text{ and } \mathcal{R}_i^A(\mathbf{x}) \oplus \mathcal{R}_i^B(\mathbf{x})\}|}{\sum_{i=1}^{s} n^{a_i}}.$$

That is, the distance is defined as the number of tuples that are assigned different truth values for the same predicate symbol in A and B, divided by the total number of tuples. It is the fraction of assignments on which the two structures disagree. For structures that are binary strings, the above definition is equivalent to their *edit distance*.

In the case of graphs, Definition 3 is equivalent to the "adjacency matrix" model introduced by Goldreich *et al.* [20]. This approach is particularly suited to dense graphs, and an alternative approach for *bounded degree* graphs which are represented using bounded incidence lists has been developed by Goldreich and Ron [19].

Definition 4. *Let P be a property of structures with vocabulary τ and A be such a structure with a universe of size n. Then, A is said to be ε-far from P if every structure B with universe of size n and vocabulary τ that has P satisfies* $\mathrm{dist}(A, B) \ge \varepsilon$.

Structure A is far from having a property if it is far from all structures that have the property and are the same size as A. Our goal is to differentiate between structures that have a desired property and those that are far from having the property. As usual, we must have this property in mind before constructing algorithms for it. We are especially interested in extremely efficient probabilistic approximation algorithms that examine only a very small portion of the structure.

Instead of directly providing these algorithms with the structures as input, we provide them with access to an oracle. We will assume that the algorithm is for testing property P of type τ, and that we wish to run this on the "input" $A \in STRUC(\tau)$. The algorithm is allowed to query the oracle for desired bits of the input. We formalize the queries as being of the form $R_i(\mathbf{x})$, where R_i is a predicate symbol in the vocabulary τ determined by the property and $\mathbf{x} \in \mathbb{N}^{a_i}$ is some tuple with the appropriate arity. The oracle then returns "1" if the tuple is in the predicate $\mathcal{R}_i^A$ and "0" if it is not.

Once the structure A is fixed, it is of some fixed universe size n. A truly random query is overwhelmingly likely to be of a tuple $x \notin U_n$. It is therefore essential to provide the algorithm with some means of making "meaningful" queries. In our model we allow an additional, special query: The algorithm may ask the oracle for n, the size of the universe.

We discuss multiple variations of the above after formalizing our definition in the following way. In particular, the special query for the universe is not allowed in the so-called "oblivious" model. There, it is common for the algorithm to give the oracle a natural number m, in response to which the oracle returns a uniformly random induced substructure of A with size m or an error if no such substructure exists.

Definition 5. *An ε-*tester *for property P is a randomized algorithm that is given an oracle which answers queries for the universe size and truth values of relations on desired tuples in a structure A. The ε-tester must accept A with probability at least* 2/3 *if A has P and must reject A with probability at least* 2/3 *if A is ε-far from P.*

The choice of 2/3 in Definition 5 is both traditional and arbitrary. Any probability strictly greater than 1/2 can be chosen and the resulting testers iterated a constant number of times and the majority taken to achieve any desired accuracy strictly less than one, see e.g., Hromkovič [27]. The tester is not designed according to any particular structure A and so different "inputs" can be tested using different oracles. However, the tester *may* be designed using the approximation parameter ε, which we discuss shortly.

Our definition of testers allows them to know the size of the input, have two-sided error and make adaptive queries based on the results of previous queries. Testers that are not given the size of their input are known as *oblivious* testers. These testers generally make queries by requesting a random sample of the input which is of a certain size. Such a sample is returned if it exists, otherwise the query fails. It is easy to construct properties that are testable by our definition but not by oblivious testers; for example, the property that the size of the universe is odd. In fact, several of the testers in this thesis will examine the size of the universe.

Many of those testers will examine the size of the universe only to determine if it is sufficiently large, where sufficiently large is defined as greater than some function of ε. It is possible to construct an oblivious tester in such cases, by having the tester request a sample that is larger than this function. The input is sufficiently large iff such a sample is available.

Alon and Shapira [7] have provided an exact characterization of the graph properties testable by oblivious testers with one-sided error. Goldreich and Trevisan [21] have shown that every graph property testable with q adaptive queries can be tested with $O(q^2)$ non-adaptive queries and Gonen and Ron [22] have shown that this gap exists.

We have mentioned above the notion of *testable* properties, and also implied the existence of *untestable* properties. It is therefore necessary to define testability, which we do as follows.

Definition 6. *Property P is called* testable *if for every $\varepsilon > 0$ there exists an ε-tester that makes a number of queries which can be upper-bounded by a function depending only on ε.*

It is interesting to note we allow different ε-testers for each $\varepsilon > 0$ and natural to ask why a single algorithm does not suffice. The situation is similar to that familiar in circuit complexity (cf. Straubing [45]), where we have uniform and non-uniform cases.

Definition 6 is non-uniform in the sense that the ε-testers may not be constructible given ε. It is very natural to require the ε-testers for P to be computable given ε, an additional condition (equivalent to requiring the ε-testers to be a single algorithm) that results in *uniform testability.* We [28] have shown that there exist undecidable properties that are testable iff we use a non-uniform definition of testability while Alon and Shapira [8] have also shown the same separation between uniform and non-uniform testability using a decidable property. We discuss the role of uniformity in Section 3.6.

Alon and Shapira [8] also emphasize the importance of the function in Definition 6 being only an *upper bound* on the number of queries. This is because query complexities such as $1/\varepsilon + (-1)^n$ are acceptable, albeit problematic for oblivious testers.

2.3 Logical Definitions

The definitions in this section are used especially in formulating and discussing classification problems in Chapter 4. Our logic is a pure predicate logic with equality that does not contain function symbols. There are no ordering symbols such as $\leq$ nor are there arithmetic relations such as PLUS or BIT. Enderton [14] provides a more comprehensive introduction to logic and Börger *et al.* [11] is an excellent reference for classification problems.

We begin by defining the underlying language. There exist countably infinite variable symbols, which we generally name with (possibly subscripted) x, y and z. We use lower case letters to distinguish these variable symbols from predicate symbols and let VAR be the set of variable symbols.

The equality symbol ($=$) is special. Although $=$ is a predicate, we do not allow structures to redefine it and insist that it is always interpreted as true equality. We do not allow function symbols or constant symbols (nullary function symbols).

Given a vocabulary τ as defined in Definition 1, the first-order logic of τ is defined as follows. Our logic does not contain constant symbols, and so the *atomic terms* are the variable symbols, x. Our language does not contain function symbols, and so the *terms* are exactly the atomic terms.

The *atomic formulas* are $x = y$ for terms x and y and $R_i(x_1, \ldots, x_{a_i})$ where the x_j are terms and R_i is a predicate symbol of τ.

The formulas are defined inductively. If φ and ψ are formulas, then $(\varphi \vee \psi)$ and $(\neg\varphi)$ are formulas. If x is a variable, then $(\exists x : \varphi)$ and $(\forall x : \varphi)$ are also formulas. A well-formed formula is a formula in which no variable occurs free. The first-order logic of τ

is exactly the set of well-formed formulas. We have no further use for formulas with free variables and will now refer to well-formed formulas simply as formulas.

Additional connectives including $\wedge$, $\rightarrow$ and $\leftrightarrow$ are allowed but are formally considered abbreviations. The parentheses required by the definition are omitted where the intended meaning is clear.

We say that a structure A of type τ and universe U_n *models* (or satisfies) a formula φ, written $A \models \varphi$, if φ is true when interpreted in the context of A. We define this more formally in the following way.

An *interpretation* is a function from the variable symbols to the universe,

$$I : VAR \mapsto U_n.$$

In addition, the function f maps the predicate symbols of τ to their corresponding predicates in A, that is, $f(R_i) = \mathcal{R}_i^A$. Recalling that vocabularies and structures are tuples, the function f maps the i-th element of τ to the $(i+1)$-th element of A.

As a notational convenience, if I is a function, we will write $I[x\backslash\backslash a]$ to mean

$$I[x\backslash\backslash a](y) = \begin{cases} I(y), & \text{if } y \neq x; \\ a, & \text{if } y = x. \end{cases}$$

That is, $I[x\backslash\backslash a](y)$ is the function $I(y)$ except that the value $I(x)$ has been replaced with a.

The definition of truth is inductive and follows our definition of formulas. We only write $A \models \varphi$ for well-formed formulas φ and so the initial interpretation is irrelevant, but formulas which are not well-formed may appear during the inductive steps. We say that $A \models \varphi$ if there is an interpretation I such that $(A, I, f) \models \varphi$. Inductively,

1. $(A, I, f) \models (x = y)$ if $I(x) = I(y)$. Note $=$ is always interpreted as equality on U_n;
2. $(A, I, f) \models R_i(x_1, \ldots, x_{a_i})$ if the tuple $(I(x_1), \ldots, I(x_{a_i})) \in f(R_i)$;
3. $(A, I, f) \models (\psi \vee \gamma)$ if it is true that $(A, I, f) \models \psi$ or $(A, I, f) \models \gamma$;
4. $(A, I, f) \models (\neg\psi)$ if $(A, I, f) \models \psi$ is not true,
5. $(A, I, f) \models (\exists x : \psi)$ if there exists an $a \in U_n$ such that $(A, I[x\backslash\backslash a], f) \models \psi$;
6. $(A, I, f) \models (\forall x : \psi)$ if for every $a \in U_n$, it is true that $(A, I[x\backslash\backslash a], f) \models \psi$.

This definition of truth is generally attributed to Tarski [47] (see [48] for an English translation).

We say that formula φ defines property $P := \{A \mid A \models \varphi\}$ and so $A \models \varphi$ is equivalent to saying that A has property P. On one hand, the expressive power of our language is quite weak. First-order logic with ordering $(\leq)$ and arithmetic relations PLUS and TIMES, or equivalently BIT, is a characterization of DLOGTIME-uniform AC^0, see e.g. Barrington *et al.* [9]. It therefore cannot express the property PARITY, which is true only for binary strings containing an odd number of 1s, see Furst *et al.* [18]. Our language does not contain ordering or arithmetic relations and so it is even weaker.

On the other hand, it is rich enough to express both testable and untestable properties. In some sense (see *indistinguishability* in Section 3.4) it is more powerful in the context of testing than classically. It is useful that our language is the same as the pure predicate logic considered in the traditional classification problem, allowing us to make several comparisons in Subsection 4.1.2. Finally, properties that are closed under isomorphisms are most natural in property testing and many possible additions to our language would force us to focus on properties that are not closed under isomorphisms.

We use lower-case Greek letters, especially φ, ψ and γ, for first-order formulas and x, y and z for first-order variables. We refer to members of the universes of structures with a, b and u when it is necessary to distinguish between variables and the underlying members of the universe that they are bound to.

2.4 Classification Definitions

Recall that every first-order formula has an equivalent formula in *prenex normal form.* That is, for any φ there exists a logically equivalent φ' that is of the form

$$\varphi' = \pi_1 x_1 \ldots \pi_a x_a : \psi,$$

where ψ is quantifier-free and the π_i are either $\forall$ or $\exists$. For example, the first-order formula $\exists x : (S(x) \wedge \exists x : (\neg S(x)))$ is equivalent to $\exists x \exists y : (S(x) \wedge \neg S(y))$.

This example can be expressed with two existential quantifiers ($\exists^2$), a single monadic predicate and does not require equality. There are a number of common ways to classify first-order formulas, including the number of distinct variables and the number of quantifiers. However, the most traditional methods have been to classify formulas in prenex normal form based on the *pattern* of quantifiers and the vocabulary τ defining the language. In particular, there are a number of interesting relationships between the testability of properties and the patterns of quantifiers that can be used to express them.

Our definitions for classification are very similar to those used by Börger *et al.* [11], however one notable difference is that we restrict ourselves to pure predicate logics rather than also considering logics with function symbols. The similar notation allows us to easily compare what is currently known regarding the classification for testability with the traditional classification for decidability, which we do in Section 4.1.2.

Definition 7. *A* prefix vocabulary class *is specified as*

$$[\Pi, p]_e.$$

Here, Π *is a string over the four-character alphabet* $\{\exists, \forall, \exists^*, \forall^*\}$*,* p *is either the special phrase 'all' or a sequence over* $\mathbb{N}$ *and the first infinite ordinal* ω*, and e is either '='* *or* λ*.*

Note that we have only defined a syntactic object; it is essentially a triple that also contains two brackets. In general, p is an infinite sequence although we will consider normal forms shortly. We will use these triples to define classes of first-order formulas, and so we now define their meaning.

The first-order sentence

$$\varphi := \pi_1 x_1 \pi_2 x_2 \dots \pi_r x_r : \psi$$

in prenex normal form, with quantifiers π_i and quantifier-free ψ, is a member of the prefix vocabulary class given by $[\Pi, (p_1, p_2, \dots)]_e$, where $p_i \in \mathbb{N} \cup \{\omega\}$ iff

1. The string $\pi_1 \pi_2 \dots \pi_r$ is contained in the language specified by Π when Π is interpreted as a regular expression[2].

2. If p is not *all*, at most p_i distinct predicate symbols of arity $i \geq 1$ appear in ψ.

3. Equality ($=$) appears in ψ only if e is '$=$'.

That is, Π describes the pattern of quantifiers for sentences in the class, p gives the maximum number of predicate symbols of each arity and e determines whether the equality symbol is permitted. It is traditional to include an additional sequence f describing the permitted function symbols, but we do not allow function symbols and so omit f.

Our sentences are always finite in length. If a prefix class has $p_i = \omega$, then formulas may contain any finite number of i-ary relation symbols. If $p = all$, then formulas in the class may contain any finite number of relation symbols of any finite arities.

We mentioned above that p is, in general, an infinite sequence. However if we consider graphs as an example, it is tiresome to write the infinitely many trailing zeros in $(0, 1, 0, \dots)$. As a convention, we therefore suppress trailing zeros in p, and so $(0, 1)$ corresponds to the case of graphs.

If there are not infinitely many trailing zeros in p, then the sum $p_i + p_{i+1} + \cdots$ is infinite for all i. In this case, we can use some of these higher arity predicate symbols to "simulate" lower arity symbols, for example using $E(x, x)$ to simulate a monadic predicate $S(x)$. This implies that all sequences p have an equivalent sequence p' such that p' is either a finite sequence (omitting trailing zeros) or the special phrase *all*. This is part of the definition of a *standard* prefix class, see Börger *et al.* [11].

We are interested in the testability of prefix classes and so we say that a prefix class is *testable* if every formula in the class expresses a testable property in the context of every vocabulary in which it is possible to evaluate the formula. However, it is sufficient to consider only the minimal vocabulary needed to evaluate the formula. We formalize this in the following simple lemma, where an extension of a vocabulary τ is any vocabulary formed by adding a new (distinct) predicate symbol to those in τ.

[2]We slightly modify the usual semantics of regular expressions so that $\forall$ (resp. $\exists$) matches the empty string λ as well as $\forall$ (resp. $\exists$). This is because we wish to consider closed prefix vocabulary classes, see Section 2.3.3 of Börger *et al.* [11].

Lemma 1. *Let φ be a formula in the first-order logic of vocabulary τ and τ' be any extension of τ. If φ defines a property that is testable in the context of τ, then the property of type τ' defined by φ is also testable.*

Proof. Assume φ defines property P of type τ when interpreted as a formula of type τ and property P' of type τ' when interpreted as a formula of type τ'. Additionally assume that the "new" predicate symbol in τ' is N which has arity a.

Let T^{τ}_{ε} be a ε-tester for P. We will show that this is also a ε-tester for P'. Let A be a structure with type τ' and assume $A \in P'$. Removing the N predicate, the corresponding $A' \in \mathit{STRUC}(\tau)$ has property P and so the tester will accept with probability at least 2/3, as desired.

Assume that $\operatorname{dist}(A, P') \geq \varepsilon$ and again let A' be the structure of type τ formed by removing the N predicate from A. By the definition of distance,

$$\operatorname{dist}(A', P) = \min_{B \in P} \frac{\sum_{1 \leq i \leq s} |\{\mathbf{x} \mid \mathbf{x} \in U^{a_i} \text{ and } \mathcal{R}_i^A(\mathbf{x}) \oplus \mathcal{R}_i^B(\mathbf{x})\}|}{\sum_{i=1}^{s} n^{a_i}} \geq$$

$$\operatorname{dist}(A, P') = \min_{B \in P} \frac{\sum_{1 \leq i \leq s} |\{\mathbf{x} \mid \mathbf{x} \in U^{a_i} \text{ and } \mathcal{R}_i^A(\mathbf{x}) \oplus \mathcal{R}_i^B(\mathbf{x})\}|}{n^a + \sum_{i=1}^{s} n^{a_i}} \geq \varepsilon.$$

The tester will therefore reject such A with probability at least 2/3 as desired. □

Testable properties therefore remain testable when the vocabulary is extended and so a property is testable in every relevant vocabulary if and only if it is testable in the minimal relevant vocabulary. A prefix class is *untestable* if it is not testable, that is, if it includes some property that is untestable.

Chapter 3

Basic Results

3.1 Testable Properties

Perhaps the most basic result in property testing is the existence of the objects being studied, testable properties. The testability of a property depends only on the property itself and not on which of the various equivalent definitions (formulas) expresses it. It is therefore easy to see that trivial properties such as *always true* and *always false* are testable, but it is worthwhile to prove an example explicitly. We begin by considering palindromes of even length. It is easy to extend this to all palindromes, however it is useful to maintain uniformity with later results (cf. Theorem 4) from the literature.

Theorem 1. *The property of being an even-length palindrome, $L_P = \{u\overleftarrow{u}\}$ over the alphabet $\{0, 1\}$, is testable with query complexity $O(1/\varepsilon)$.*

Proof. The following algorithm is an ε-tester for the property.

1. Query the oracle for the number of characters, n. Reject if n is odd.
2. Choose $2/\varepsilon$ integers uniformly at random from $[0, n/2 - 1]$.
3. For each of these integers i, query the oracle for $S(i)$ and $S(n - 1 - i)$.
4. Reject if any of these pairs differ, otherwise accept.

To show that this is an ε-tester for even-length palindromes, we are required to show that it accepts palindromes with probability greater than 2/3, and rejects when the oracle uses a string w where $\text{dist}(w, L_P) \geq \varepsilon$, again with probability greater than 2/3. The first part is easy; if the input is a palindrome, the bits we compare will be equal by definition and so we will accept with zero error.

Next, assume that $\text{dist}(w, L_P) \geq \varepsilon$. Therefore, there are at least εn bits in w that must be modified to reach a palindrome. For each of the random bit pairs that we examine, the probability that we have failed to find one of these bits of evidence is at most $(1 - \varepsilon)$. The total error probability is then at most $(1 - \varepsilon)^{2/\varepsilon}$, which is uniformly upper-bounded by $\mathbf{e}^{-2\varepsilon/\varepsilon} = \mathbf{e}^{-2} \approx 0.135$ for all ε in $(0, 1]$, where $\mathbf{e}$ is the Euler number. The tester therefore rejects such inputs with probability at least $1 - \mathbf{e}^{-2} > 2/3$, as desired. □

The language of palindromes (with even length) is a well-known example of a context-free language that is not regular. The property of being an empty graph is also testable, and the proof is nearly identical. However, we will use both results a number of times, and so we show both explicitly.

Theorem 2. *The property of being an empty graph,*

$$P_E := \{G \mid G \in STRUC(\tau_G) \text{ and } G \text{ has no edges}\},$$

is testable.

Proof. The following is an ε-tester for the property.

1. Query the oracle for the number of vertices, n.
2. Choose $2/\varepsilon$ pairs of vertices (x, y) uniformly at random.
3. For each of these pairs, query the oracle for $E(x, y)$.
4. Reject if we see any edges, otherwise accept.

Of course, we will not see any edges if the input is empty and so we will accept with zero error. For G such that $\text{dist}(G, P_E) \geq \varepsilon$, graph G contains at least εn^2 edges. For each of the random pairs that we query, the probability that we have failed to find an edge is at at most $(1 - \varepsilon)$. As above, the total error probability is at most $(1 - \varepsilon)^{2/\varepsilon}$ which is strictly less than $1/3$ for all $\varepsilon \in (0, 1]$. Therefore, we will find an edge and reject such G with probability at least $2/3$, as desired. □

3.2 Untestable Properties

In Section 3.1, we proved that the language of (even-length) palindromes is testable by providing a correct tester that makes $O(1/\varepsilon)$ queries. Here we wish to show that there exists an untestable property, i.e., a property which cannot be tested with $o(n)$ queries. The example and proof that we use is due to Alon *et al.* [4].

The specific example given here allows a number of interesting corollaries, for example, that the testable properties are not closed under complement (Corollary 3). We begin by reviewing and proving the tool that we will use to prove non-testability.

3.2.1 Yao's Principle

The proof relies on a variation (Principle 1) of Yao's principle, a tool for proving lower bounds in the context of randomized algorithms. Yao's principle is an interpretation by Yao [49] of von Neumann's [34] minimax theorem for randomized computation. We prove only the direction of the minimax theorem that is required for our purposes; for a survey of minimax theorems and their proofs see Simons [42].

We begin by providing the definitions required to state Principle 1.

Definition 8. *A* deterministic tester *is a binary tree where each internal node is labeled with a non-negative integer, each leaf is labeled "accept" or "reject," and the two edges from a node are labeled* 0 *and* 1*.*

Given an input, we execute the tester as follows. Beginning at the root, we interpret the labels on internal nodes as the bit position of the input that will be queried. If the result of the query is 0, we follow the 0 edge and otherwise the 1 edge. When we reach a leaf we output the decision on the label. Note that it is equivalent to label the internal nodes with atomic formulas such as $E(0,1)$ as these are equivalent to specific bits of the input, the positions of which can be easily computed.

Definition 9. *The* complexity *of a deterministic tester is the number of internal nodes, including the root unless it is a leaf, on the longest path from the root to a leaf.*

The complexity of a deterministic tester is then the maximum number of queries that it makes. Our interest is limited to testers of finite complexity, i.e., those that output a decision in finite time. Without loss of generality, we can restrict our attention to balanced binary trees, by making extra, useless queries on the shorter paths in order to "pad" their lengths.

Principle 1 (Yao's Principle)**.** *Let τ be a vocabulary. If there exists an $\varepsilon \in (0,1)$ and a distribution over $STRUC^n(\tau)$ such that all deterministic testers with complexity c have an error-rate greater than $1/3$ for property P, then P is not testable with complexity c.*

The definition of "testable" is of course our usual one involving random testers. In general, our goal will be to show that it is impossible to test P using randomized testers whose complexity does not depend on the input size. That is, for sufficiently large n and some increasing function $c := c(n)$ of n, we wish to find a distribution of inputs such that all deterministic testers with complexity c have error-rates greater than $1/3$.

For testability, we are concerned only with the error-rate on inputs that either have our desired property or are ε-far from having the property. Therefore, we define the error-rate of a tester to be non-zero only on such examples. Because of this, it suffices to restrict our attention to distributions that give zero probability to the remaining "possible" inputs (those that do not have the property in question, but are also not ε-far from it).

We begin by proving the following direction of the minimax theorem. We say that a vector $\mathbf{x} \in \mathbb{R}^{|\mathbf{x}|}$ is a *probability vector* if each of its components is a non-negative real number and the sum of its components is 1. For $n > 0$, we let $\mathbb{P}^n$ be the set of probability vectors with n components,

$$\mathbb{P}^n := \left\{ \mathbf{x} \mid \mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n, x_i \geq 0 \text{ for all } 1 \leq i \leq n, \text{ and } \sum_{i=1}^{n} x_i = 1 \right\}.$$

It is well-known that equality holds in the following, however we restrict ourselves to stating and proving only the direction required for Principle 1, as mentioned above.

Theorem 3 (Minimax Theorem). *Let M be an $a \times b$ matrix of non-negative reals and $X = \mathbb{P}^a$ and $Y = \mathbb{P}^b$ be the sets of all probability vectors with a and b components, respectively. Then,*

$$\max_{\mathbf{y}\in Y} \min_{\mathbf{x}\in X} \mathbf{x}M\mathbf{y}^T \le \min_{\mathbf{x}\in X} \max_{\mathbf{y}\in Y} \mathbf{x}M\mathbf{y}^T. \tag{3.1}$$

Proof. Let $\mathbf{y}^*$ be any of the arg max on the left in (3.1),

$$\mathbf{y}^* := \arg\max_{\mathbf{y}\in Y} \min_{\mathbf{x}\in X} \mathbf{x}M\mathbf{y}^T,$$

and $\mathbf{x}^*$ be any of the arg min on the left in (3.1),

$$\mathbf{x}^* := \arg\min_{\mathbf{x}\in X} \mathbf{x}M\mathbf{y}^{*T}.$$

Then, for any probability vector $\mathbf{x} \in X$, by the definition of minimum,

$$\mathbf{x}M\mathbf{y}^{*T} \ge \mathbf{x}^*M\mathbf{y}^{*T}. \tag{3.2}$$

Let $\mathbf{x}^+$ and $\mathbf{y}^+$ be any of the arg min and arg max on the right of (3.1), that is

$$\mathbf{x}^+ := \arg\min_{\mathbf{x}\in X} \max_{\mathbf{y}\in Y} \mathbf{x}M\mathbf{y}^T \text{ and } \mathbf{y}^+ := \arg\max_{\mathbf{y}\in Y} \mathbf{x}^+M\mathbf{y}^T.$$

Then, by the definition of maximum, $\mathbf{x}^+M\mathbf{y}^{+T} \ge \mathbf{x}^+M\mathbf{y}^{*T}$. Therefore, by (3.2),

$$\min_{\mathbf{x}\in X} \max_{\mathbf{y}\in Y} \mathbf{x}M\mathbf{y}^T = \mathbf{x}^+M\mathbf{y}^{+T} \ge \mathbf{x}^+M\mathbf{y}^{*T} \ge \mathbf{x}^*M\mathbf{y}^{*T} = \max_{\mathbf{y}\in Y} \min_{\mathbf{x}\in X} \mathbf{x}M\mathbf{y}^T.$$

□

Given Theorem 3, it is easy to show Principle 1. In (3.1), we call $\mathbf{x}$ on the left and $\mathbf{y}$ on the right "inner vectors." This is because we can think of them as being chosen after the "outer vectors" ($\mathbf{y}$ on the left and $\mathbf{x}$ on the right) are fixed. For the "inner" vectors, it suffices to consider unit vectors $\mathbf{e}_i$, where component i is 1 and all other elements are 0. This is because once the outer vector is fixed, denoting the i-th component of a vector $\mathbf{z}$ by $[\mathbf{z}]_i$,

$$\min_{\mathbf{x}\in X} \mathbf{x}M\mathbf{y}^T = \mathbf{e}_{\arg\min_i [M\mathbf{y}^T]_i} M\mathbf{y}^T$$

and likewise for the maximum on the right of (3.1). This proves the following simple corollary of Theorem 3.

Corollary 1. *Let M be an $a \times b$ matrix of non-negative reals and $X = \mathbb{P}^a$ and $Y = \mathbb{P}^b$ be the sets of all probability vectors with a and b components, respectively. Then,*

$$\max_{\mathbf{y}\in Y} \min_{\mathbf{e}_i\in X} \mathbf{e}_iM\mathbf{y}^T \le \min_{\mathbf{x}\in X} \max_{\mathbf{e}_j\in Y} \mathbf{x}M\mathbf{e}_j^T. \tag{3.3}$$

Proof (Principle 1). Principle 1 is an interpretation of (3.3) in the context of testing. We let a be the number of deterministic testers with complexity c whose queries are evaluable in structures of type τ that have n elements. We assume that there is an enumeration of these testers. Then, a randomized tester with complexity c for structures of n elements is given by a probability vector $\mathbf{x} \in X$, where $[\mathbf{x}]_i$ is interpreted as the probability that the randomized tester behaves like the i-th deterministic tester. A unit vector $\mathbf{e}_i \in X$ specifies the i-th deterministic tester.

Likewise, we assume there is an enumeration of structures of type τ that have n elements and let b be the number of such structures. Then, a probability vector $\mathbf{y} \in Y$ is a distribution over these structures and a unit vector $\mathbf{e}_j$ specifies the j-th structure.

For matrix M, we let M_{ij} be 1 if the i-th deterministic tester is incorrect on the j-th input and this input either has the desired property or is ε-far from it. Otherwise, we let M_{ij} be 0.

We now have have an $a \times b$ matrix M and a meaning for a and b component probability vectors and so we can interpret the meaning of Corollary 1. On the left, $\mathbf{e}_i M \mathbf{y}^T$ is the average-error of the i-th deterministic tester on a structure chosen according to distribution $\mathbf{y}$. Likewise, on the right, $\mathbf{x} M \mathbf{e}_j^T$ is the error-rate of the randomized tester specified by $\mathbf{x}$ on the j-th structure.

Therefore, the left side of (3.3) is the average-error of the "best" deterministic tester on the "worst" distribution of inputs, when we define "best" as the lowest average-error. If we find some distribution $\mathbf{y}^+$ of inputs such that all deterministic testers have an error-rate greater than 1/3, then 1/3 is a lower bound on the left side, and therefore the right side, of (3.3).

The right side of (3.3) is the error-rate of the "best" randomized tester on the "worst" input structure, when "best" is defined as the best worst-case. If the "best" randomized tester with complexity c has an error-rate greater than 1/3 on an input, we can conclude that the property in question is not testable with complexity c. □

3.2.2 An Untestable Property

We will now prove that there exists an untestable property. The specific example that we use is $L := \{u\overleftarrow{u}v\overleftarrow{v}\}$, where the underlying alphabet is $\{0, 1\}$. This is clearly a context-free language and so Theorem 4 will also imply that there exist untestable context-free languages. The proof here is due to Alon *et al.* [4].

In Theorem 1, we showed that $L_P = \{u\overleftarrow{u}\}$ is testable. We might hope to use this tester twice in an attempt to test $L = \{u\overleftarrow{u}v\overleftarrow{v}\}$, possibly iterating and taking the majority decision in order to increase the success probability above 2/3. However, this assumes that we know the "divider" between $u\overleftarrow{u}$ and $v\overleftarrow{v}$ in advance, which is not true. Our attempt to use the tester from Theorem 1 is therefore unsuccessful, as we would be forced to search for the divider by using the previous tester a linear number of times and make too many queries. In the following theorem, we will show that it is not possible to test L using any algorithm that makes $o(\sqrt(n))$ queries. This is our first example of a property that is *untestable.*

Theorem 4 (Alon *et al.* [4]). *The language $L = \{u\overleftarrow{u}v\overleftarrow{v}\}$, where u and v are strings over $\{0,1\}$ is not testable with complexity $o(\sqrt{n})$.*

Proof. The proof is an application of Principle 1, and so we begin by defining a distribution D of strings. Our goal is to show that D is such that *all* deterministic testers that make $c = o(\sqrt{n})$ queries have error rates greater than 1/3 for L. Principle 1 will then imply that L is not testable with $o(\sqrt{n})$ queries, as desired. In the proof we will assume that n is sufficiently large, although it is of course always finite. This will conveniently allow us to assume that n is divisible by 6 (if it is not, use the first multiple of 6 that is larger than n), avoiding the use of floors.

Distribution D draws from a distribution D^+ of strings in L or from a distribution D^- of strings that are ε-far from L with probability 1/2 in each case. Distribution D^- selects a $w \in \{0,1\}^n$ uniformly at random from the set of w such that $\text{dist}(w, L) \geq \varepsilon$. Distribution D^+ first selects a positive integer $k \in \{1, \ldots, n/3\}$, and then selects strings $u \in \{0,1\}^k$ and $v \in \{0,1\}^{(n-2k)/2}$, uniformly at random in each case. The result is a string $w = u\overleftarrow{u}v\overleftarrow{v} \in L \cap \{0,1\}^n$. We can think of D^+ as selecting a pair (k, w) and some w can be chosen with multiple values of k.

We now consider an arbitrary deterministic tester A with complexity $c = o(\sqrt{n})$. As mentioned above, we can consider A to be a balanced binary tree and it therefore has 2^c leaves. We let T_0 be the set of leaves labeled “reject” and T_1 be the set of leaves labeled “accept.” Additionally, we associate to each leaf t a sequence of c pairs $Q_t := ((p_1^t, r_1^t), \ldots, (p_c^t, r_c^t))$ corresponding to the bit positions queried ($p_i^t \in [0, n-1]$) and the results ($r_i^t \in \{0,1\}$) on the path from the root of A to t. Furthermore, we define $q(Q, w)$ for $w \in \{0,1\}^n$ to be true if w agrees with Q, or more formally, if bit p_i in w is r_i for all $1 \leq i \leq c$, and false otherwise.

We also define $E^+(Q)$ (resp. $E^-(Q)$) to be the set of strings with length n that agree with Q and are positive (ε-far) instances for L. More formally,

$$E^+(Q) := \{w \mid w \in L \cap \{0,1\}^n \text{ and } q(Q,w) \text{ is true}\} \text{ and}$$

$$E^-(Q) := \{w \mid w \in \{0,1\}^n, \text{dist}(w, L) \geq \varepsilon \text{ and } q(Q,w) \text{ is true}\}.$$

If we consider some fixed accepting leaf t, then $E^-(Q_t)$ is the set of strings that A makes an error on when ending in t and similarly for a rejecting leaf t and $E^+(Q_t)$. We define the probability of a set to be the sum of the probabilities of the members of the set, and therefore the total error of A is

$$\sum_{t \in T_0} \Pr_D(E^+(Q_t)) + \sum_{t \in T_1} \Pr_D(E^-(Q_t)).$$

We will show the following two lemmata for any sequence Q of $o(\sqrt{n})$ elements.

Lemma 2. $\Pr_D(E^+(Q)) = (1/2 - o(1))2^{-|Q|}$.

Lemma 3. $\Pr_D(E^-(Q)) = (1/2 - o(1))2^{-|Q|}$.

Every leaf of A must be labeled with exactly one decision, and so $|T_0| + |T_1| = 2^{|Q|}$. Using the two lemmata, the total error of A is then

$$|T_0|(1/2 - o(1))2^{-|Q|} + |T_1|(1/2 - o(1))2^{-|Q|} = \frac{|T_0| + |T_1|}{2^{|Q|}}(1/2 - o(1)) = (1/2 - o(1)).$$

The $o(1)$ term approaches zero as n grows and so the error is greater than 1/3 for sufficiently large n. Therefore, by Principle 1, language L cannot be tested with complexity $o(\sqrt{n})$ and is therefore untestable. □

We must now show the two lemmata used by the theorem. We will use the (loose) upper bound $|L \cap \{0,1\}^n| \leq 2^{n/2}n/2$, which follows from the observation that we first choose the string uv and then choose the length of u to determine a $w \in L$. In addition, if the length of u is fixed at k, there are exactly $2^{n/2}$ corresponding w, which will be useful given the definition of distribution P.

Lemma 2. *If* $|Q| = o(\sqrt{n})$, *then* $\Pr_D(E^+(Q)) = (1/2 - o(1))2^{-|Q|}$.

Proof. First, for any $w \in \{0,1\}^n \cap L$, by the definition of D,

$$\Pr_D(w) = 1/2 \Pr_{D^+}(w) = \frac{\sum_{k=1}^{n/3} |\{u \mid \exists v : w = u\overleftarrow{u}v\overleftarrow{v}, |u| = k\}|}{2^{n/2}\frac{2n}{3}}.$$

Next, it follows from the definition of $E^+(Q)$ that $\Pr_D(E^+(Q)) = \frac{1}{2}\Pr_{D^+}(E^+(Q)) =$

$$\left(\frac{1}{2^{n/2}2n/3}\right)\sum_{k=1}^{n/3} |\{w \mid \exists u, v : w = u\overleftarrow{u}v\overleftarrow{v}, |u| = k \text{ and } q(Q, w)\}|.$$

If the sequence Q contains a pair of queries that are symmetric about k or $n/2 + k$, then the pair queries the same bit in u and $\overleftarrow{u}$ or in v and $\overleftarrow{v}$. If the results of these queries must be different, then it is impossible for such a w to exist. However, Q contains $\binom{|Q|}{2}$ pairs, and so there is no such pair of queries for at least $n/3 - 2\binom{|Q|}{2}$ choices of k. For each of these k,

$$|\{w \mid \exists u, v : w = u\overleftarrow{u}v\overleftarrow{v}, |u| = k \text{ and } q(Q, w)\}| = 2^{n/2-|Q|}.$$

Therefore, as an upper bound,

$$\Pr_D(E^+(Q)) \leq \left(\frac{1}{2^{n/2}2n/3}\right)\left(2^{n/2-|Q|}\right)\left(n/3 - 2\binom{|Q|}{2}\right) =$$

$$\left(1/2 - \frac{2\binom{|Q|}{2}}{n/3}\right)2^{-|Q|},$$

which is $(1/2 - o(1))2^{-|Q|}$ for $|Q| = o(\sqrt{n})$, as desired. □

Lemma 3. *If* $|Q| = o(\sqrt{n})$, *then* $\Pr_D(E^-(Q)) = (1/2 - o(1))2^{-|Q|}$.

Proof. We noted above that the number of strings in L with length n is at most $2^{n/2}n/2$ and so the number of strings of length n that are not ε-far from L satisfies

$$|\{w \mid \operatorname{dist}(w, L) \leq \varepsilon, |w| = n\}| \leq \left(2^{n/2}n/2\right) \sum_{i=0}^{\varepsilon n} \binom{n}{i}.$$

This is upper-bounded by $2^{n/2+2\varepsilon n \log 1/\varepsilon}$. Recall that $E^-(Q)$ is the set of $w \in \{0,1\}^n$ such that $\operatorname{dist}(w, L) \geq \varepsilon$ and w agrees with Q. Then, the size of $E^-(Q)$ must satisfy

$$\left|E^-(Q)\right| \geq 2^{n-|Q|} - 2^{n/2+2\varepsilon \log 1/\varepsilon n}.$$

Then, by the definition of distribution D,

$$\Pr_D(E^-(Q)) = 1/2Pr_{D^-}(E^-(Q)) \geq \frac{|E^-(Q)|}{2^n}.$$

For $\varepsilon < 1/16$, this is

$$(1/2 - 2^{-\Theta(n)+|Q|})2^{-|Q|} = (1/2 - o(1))2^{-|Q|},$$

where the last equality holds for $|Q| = o(n)$. Our assumption is stricter, $|Q| = o(\sqrt{n})$. □

Theorem 4 allows a number of interesting corollaries. In particular, now that we have seen that both testable and untestable properties exist, in the next section we will examine closure properties of the set of testable properties.

3.3 Closure Properties

The language in Theorem 4 is the set of strings that are the concatenation of two even-length palindromes. That is, it is the product $L_P L_P$, where L_P is the language of binary palindromes with even length proven testable in Theorem 1. We therefore immediately get the following corollary.

Corollary 2. *The testable properties are not closed under product.*

In addition, the definition of testability is asymmetric in the sense that it requires us to accept all positive instances with high probability while requiring us to reject, again with high probability, only those negative instances that are also far from having the property.

Corollary 3. *The testable properties are not closed under complement.*

Proof. We use the complement of the language L from Theorem 4, $\overline{L} := \{w \mid w \notin L\}$. We begin by showing that for sufficiently large n, there is no $w \in \{0,1\}^n$ such that $\operatorname{dist}(w, \overline{L}) \geq \varepsilon$.

Lemma 4. *Given $\varepsilon > 0$, there is no $w \in \{0,1\}^n$ such that* $\operatorname{dist}(w, \overline{L}) \geq \varepsilon$ *for* $n > 1/\varepsilon$.

Proof. Let $w \in \{0,1\}^n$ be arbitrary. There are two cases. If $w \in \overline{L}$, then $\operatorname{dist}(w, \overline{L}) = 0 < \varepsilon$. Therefore, assume that $w \in L$ and $n > 0$. It is trivial to see that all strings in L have an even number of 1s and an even number of 0s. We change the first character of w to its opposite, and call the resulting string w'. The number of 1s in w was even, and therefore the number of 1s in w' is odd (we have either removed one or added one). Therefore, w' cannot be in L and so it is by definition in $\overline{L}$. We have only modified one bit, and so $\operatorname{dist}(w, \overline{L}) = 1/n < \varepsilon$, where the inequality holds for $n > 1/\varepsilon$. □Lemma 4

A tester for $\overline{L}$ can therefore simply verify that the input string is sufficiently long, i.e., that $n > 1/\varepsilon$, and then accept. Membership for inputs that are not sufficiently long can be computed exactly after querying all $n \leq 1/\varepsilon$ bits. This tester clearly accepts $w \in \overline{L}$ with no error. When $\varepsilon > 0$ is fixed, if $\operatorname{dist}(w, \overline{L}) \geq \varepsilon$, then $n \leq 1/\varepsilon$, by Lemma 4. The tester therefore queries every bit and rejects without error. The tester makes at most $1/\varepsilon$ queries, which does not depend on n, and therefore $\overline{L}$ is testable.

However, by the definition of complement, $\overline{\overline{L}} = L$, which is not testable by Theorem 4 and so the testable properties are not closed under complement. □

The above results are both negative. However, if we consider closure under union, we obtain the following.

Theorem 5. *The testable properties are closed under finite unions.*

Proof. Recall that for all $\alpha, \beta \in (0, 1/2)$, if we have a tester which is correct with probability at least $1/2 + \alpha$, we can construct a tester which is correct with probability at least $1/2 + \beta$ by running the first tester a constant number of times and taking the majority output (see, e.g., Section 11.2 in Papadimitriou [35]). The number of iterations for fixed reals α and β does not depend on n and so the number of queries in the new tester is a constant multiple of the original number of queries.

Let P_1 and P_2 be two testable properties. Then, for every $\varepsilon > 0$, there exist testers T_1^ε (T_2^ε) accepting inputs S that have property P_1 (resp. P_2) and rejecting inputs S where $\operatorname{dist}(S, P_1) \geq \varepsilon$ (resp. $\operatorname{dist}(S, P_2) \geq \varepsilon$). The discussion above allows us to assume that the testers are correct in each case with probability at least $\sqrt{2/3}$. It is then simple to construct the following tester T^ε for $P_1 \cup P_2$, for input I.

1. Run T_1^ε on I. If T_1^ε accepts, halt and accept.

2. Next, run T_2^ε on I and output its decision.

Clearly, T^ε accepts inputs I that have property P_1 or P_2 with probability greater than $2/3$. An input I that is ε-far from the property $P_1 \cup P_2$ is ε-far from both P_1 and from P_2. This is because, if I is ε-far from $P_1 \cup P_2$, then there is no $I' \in P_1 \cup P_2$ such that $\operatorname{dist}(I, I') \leq \varepsilon$. By the definition of union this implies that there is no $I' \in P_1$ such that $\operatorname{dist}(I, I') \leq \varepsilon$, and likewise for P_2.

We therefore reject inputs which are ε-far from $P_1 \cup P_2$ with probability at least $\sqrt{2/3}^2 = 2/3$ and so T^ε is an ε-tester for $P_1 \cup P_2$, as desired. □

3.4 Indistinguishability

When proving the testable properties are not closed under complement (Corollary 3), we showed that all sufficiently long strings were "close" to the language $\overline{L}$ we were considering. That is, for all ε there is an upper bound on the length of strings w such that $\mathrm{dist}(w, \overline{L}) \geq \varepsilon$ (Lemma 4).

This was our first example of a concept called *indistinguishability*, which was defined for graph properties by Alon *et al.* [2]. Roughly speaking, if we have two languages (properties) L_1 and L_2 such that all elements of L_1 are *close* to L_2 and all elements of L_2 are *close* to L_1, then we say the two languages (properties) are indistinguishable. Because property testers only examine a very small portion of their inputs, the probability that these testers will find evidence that a $w \in L_1$ is not actually in L_2 is small. This will allow us to show the most important result regarding indistinguishability, that it preserves testability (Theorem 6).

Unfortunately, formally defining "close enough" is not trivial in our generalized setting. Alon *et al.* [2] defined it for loop-free graphs, but we do not have the luxury of such restrictions. Two structures are "close enough" if there is no difference that can distinguish them with high probability, and so simply using our definition of distance may not suffice.

To see this, we consider a vocabulary with two relations, H of high arity and L of low arity. The low arity relation contributes an asymptotically insignificant amount to the distance of Definition 3 because the number of possible high arity tuples dominates any difference in L assignments. Consider two large structures A and B with this vocabulary, where A and B have identical H assignments and opposite L assignments. For $\varepsilon > 0$, if these structures are sufficiently large, then $\mathrm{dist}(A, B) < \varepsilon$. However, although these structures are very close, a tester could distinguish between them by examining only a small number (one) of the L assignments.

We can use loops in graphs in a similar way, a graph possibly containing loops is equivalent to an loop-free graph with a monadic color relation. We therefore begin by defining a *subtype* of a relation.

Definition 10. *Let R be a relation with arity a. Then, a* subtype *of R is a set of at most a elements, each of which is a set containing integers from $\{1, \ldots, a\}$. Each of these integers must appear in exactly one of the elements of the subtype.*

For example, $\{\{1\}, \{2\}\}$ is a subtype of the edge predicate E for graphs. This corresponds to the set of pairs of E for which the element in position 1 of the pair occurs only in position 1 and the element in position 2 occurs only in position 2. That is, this subtype is the set of edges that are not loops. The subtype $\{\{1, 2\}\}$ then corresponds to the set of edges that are loops. This can be more formally defined as follows.

Definition 11. *Let R be a relation with arity a and S be a subtype of R,*

$$S = \left\{ \{t_1^1, \ldots, t_{b_1}^1\}, \ldots, \{t_1^{|S|}, \ldots, t_{b_{|S|}}^{|S|}\} \right\}.$$

A tuple $(x_1, \ldots, x_a)$ belongs to *S if for all t_1^i, it is the case that $x_{t_1^i} = x_{t_j^i}$ for all j and, if $x_u = x_v$ for some u, v then u and v are both contained in the same element of S.*

It is harmless but useless for the same i to appear multiple times in the same element of a subtype. We now define the $S-$distance between structures, where S is a subtype of relation R_i.

Definition 12. *Let $A, B \in STRUC^n(\tau)$ be structures with universe U and vocabulary τ, and let S be a subtype of relation $R_i \in \tau$. Then, the S-distance between A and B is*

$$S\text{-dist}(A, B) := \frac{|\{\mathbf{x} \mid \mathbf{x} \in U^{a_i}, \mathbf{x} \text{ belongs to } S \text{ and } R_i^A(\mathbf{x}) \oplus R_i^B(\mathbf{x})\}|}{n^{|S|}}.$$

The S-distance is the fraction of tuples belonging to S that are given different assignments by R_i in A and B. We define the R_i-distance to be the maximum S-distance for subtypes S of R_i. If the maximum R_i-distance between two structures is very small, all of the subtypes of all the relations are very similar, and there is no special query that has a high probability of finding a difference.

Let $SUB(R)$ be the set of subtypes of relation R. Then, the maximum R_i-distance is the following.

Definition 13. *Let $A, B \in STRUC^n(\tau)$ be structures with universe U and vocabulary τ. Then, the maximum R_i-distance between A and B is*

$$\max_{1 \le i \le s} R_i\text{-dist} := \max_{1 \le i \le s} \max_{S \in SUB(R_i)} S\text{-dist}(A, B).$$

When defining indistinguishability, we use the maximum R_i-distance. Note that $\text{dist}(A, B) \le \max_i R_i\text{-dist}(A, B)$, which we show as part of Theorem 9 in Section 3.7. The R_i-distance is equal to the usual distance for vocabularies with exactly one predicate symbol if it has one subtype, such as loop-free graphs or binary strings.

We generalize our definition of R_i-distance in the following way, where P is a property.

Definition 14. *Let $A \in STRUC^n(\tau)$ be a structure and P be a property of type τ. Then, the* R_i-distance between A and P *is*

$$R_i\text{-}dist(A, P) := \min_{\{B \in P \mid \#(B) = n\}} R_i\text{-}dist(A, B).$$

If the minimum is taken over an empty set, we define the distance to be infinite.

Properties that are closed under isomorphisms are most natural in property testing, and we now define this idea.

Definition 15. *Let $A, B \in STRUC^n(\tau)$ be structures with vocabulary τ. We call A* isomorphic *to B if there is a mapping $\phi : \{0, \ldots, n-1\} \mapsto \{0, \ldots, n-1\}$ such that*

1. *The mapping ϕ is bijective;*
2. *For all $R_i \in \tau$ and all $(x_1, \ldots, x_{a_i})$, it is true that $R_i^A(x_1, \ldots, x_{a_i})$ if and only if $R_i^B(\phi(x_1), \ldots, \phi(x_{a_i}))$.*

The mapping ϕ is called an *isomorphism*. We define properties to be closed under isomorphisms in the following way.

Definition 16. *Let P be a property of type τ. We say that P is* closed under isomorphisms *if for all $n \in \mathbb{N}$ and all $A, B \in STRUC^n(\tau)$, if A and B are isomorphic then $A \in P$ if and only if $B \in P$.*

We note that given the lack of ordering, all properties that can be expressed by formulas in our logic are closed under isomorphisms, although we do not use this assertion here. We are now finally ready to define indistinguishability. In the case of loop-free graphs it is equivalent to the definition given by Alon *et al.* [2].

Definition 17. *Two properties P and Q of structures with vocabulary τ are said to be* indistinguishable *if they are closed under isomorphisms and for every $\varepsilon > 0$ there exists an N_ε such that for any structure A with vocabulary τ and universe of size $n \geq N_\varepsilon$, if A has P then $\max_i R_i$-dist$(A, Q) \leq \varepsilon$ and if A has Q then $\max_i R_i$-dist$(A, P) \leq \varepsilon$.*

It is worthwhile to note that indistinguishability is an equivalence relation. It is clearly symmetric and reflexive, and transitivity is also simple to show. Assume that P is indistinguishable from P_2, which is in turn indistinguishable from P_3. For sufficiently large A, if A has P_1 then there is a B that has P_2 such that $\max_i R_i\text{-dist}(A, B) \leq \varepsilon/2$ and likewise for B and a C that has P_3. Then, $\max_i R_i\text{-dist}(A, P_3) \leq \max_i R_i\text{-dist}(A, C) \leq \varepsilon$, as desired.

The importance of indistinguishability is expressed by the following theorem, which was shown by Alon *et al.* [2] for the case of graphs. Their proof works nearly verbatim in our setting.

Theorem 6. *If P and Q are indistinguishable, then P is testable if and only if Q is testable.*

Proof. We assume without loss of generality that P is testable and provide an ε-tester for Q. By assumption, P is testable and so there exists an $\varepsilon/2$-tester $T^{\varepsilon/2}$ for P that makes at most $c(\varepsilon/2)$ queries. The tester accepts structures A that have P and rejects those that are $\varepsilon/2$-far with probability at least $2/3$ in both cases.

Let N be such that for structures A of size $n \geq N$, if A has Q then $\max_i R_i\text{-dist}(A, P) \leq \min\left(\varepsilon/2, \frac{2}{81c}\right)$. The $2/(81c)$ appears because we would like to use a trick similar to that in Theorem 1, and this will cancel nicely because $81 = 3 \cdot 27$.

We will test input A for Q in the following way.

1. If $\#(A) \leq N$, query all of A and output an exact decision.

2. Otherwise, run $T^{\varepsilon/2}$ three times on A, using random permutations of the labels of A each time. Output the majority decision.

The restriction that the properties are closed under isomorphisms allows us to use the random permutations in Step 2. This tester makes at most $3c := 3c(\varepsilon/2)$ queries and has a success probability of at least 2/3. Assume that $n = \#(A)$ is sufficiently large. If A is ε-far from Q, it is also $\varepsilon/2$-far from P. Therefore, $T^{\varepsilon/2}$ will reject A with probability at least 2/3 and our tester correctly rejects with probability at least $20/27 > 2/3$.

Now, assume that A has Q. Then, there exists an A' that has P such that A and A' differ in no more than a $\frac{2}{81c}$ fraction of the possible assignments, for every subtype and every relation. The probability that any particular query will see one of these assignments is at most $\frac{2}{81c}$ and so the probability that any of the $3c$ queries will see one of these bits is at most $1 - \left(1 - \frac{2}{81c}\right)^{3c} \leq \mathbf{e}^{-2/27} \leq 2/27$. The probability of success is then at least $20/27 - 2/27 = 2/3$ and so Q is testable, as desired. □

Indistinguishability is an equivalence relation on properties that preserves testability. It therefore partitions the set of properties into equivalence classes such that each class contains only testable properties or only untestable properties. However, in the general case only non-uniform testability is preserved, as we will see in Section 3.6. Alon *et al.* [2] used indistinguishability as a tool for proving classes of properties to be testable, as we will, but it is likely to be of more general interest.

3.5 Testable Properties that are Hard to Decide

We have already seen that there exist context-free languages that are testable (Theorem 1) and also some that are not testable (Theorem 4). However, the relationship of the testable properties with the traditional complexity hierarchy is worthy of more investigation. Goldreich *et al.* [20] have shown that there exist testable NP-complete properties. Assuming P $\neq$ NP, this means that there exists properties for which it is very easy to decide if the input nearly satisfies the property but quite hard to decide the problem exactly. Here we will show that this gap is much larger: There exist properties for which it is very easy to decide if the input nearly satisfies the property but extremely difficult to decide exactly.

In this section we show the existence of testable properties that are arbitrarily hard to decide exactly. The testable properties therefore extend arbitrarily high in the traditional hierarchy, but, by Theorem 4, do not contain even the context-free languages.

We define "arbitrarily hard" properties as meaning, for every computable $f(n)$, containing properties that are not decidable in DTIME$(f(n))$[1].

It suffices to consider loop-free graph properties, where the maximum R_i-distance is equal to the usual distance. The same result can also be shown for other vocabularies.

[1]The choice of DTIME here is arbitrary and other complexity measures such as DSPACE result in equivalent definitions.

Theorem 7. *There are arbitrarily-hard testable properties.*

Proof. Let P be an arbitrarily-hard property. We define a property Q such that P reduces to Q. Let $p(x)$ be the characteristic function for P on some reasonable encoding of the input. We define Q to be true for graph G of size n iff the number of edges in G is equal to $p(n)$.

We can obviously reduce P to Q by, on input x, computing the encoding x and outputting a graph on x vertices with exactly one edge. We can therefore construct arbitrarily hard properties Q.

It is worth noting that this reduction increases the input length by an exponential factor. Because we are only interested in ***arbitrarily hard*** properties and by the time hierarchy theorem (see Hartmanis and Stearns [25]), we can simply choose P such that it is not computable in DTIME$(2^{f(n)})$ to construct a property Q that is not computable in DTIME$(f(n))$.

Property Q is indistinguishable from the property of being the empty graph, which we showed to be testable in Theorem 2. If a graph G has property Q, it is either empty or it has one edge, and so the maximum R_i-distance is at most $1/n^2$. The converse is the same, an empty graph either has Q or the distance is at most $1/n^2$. Obviously, $1/n^2 < \varepsilon$ for $n > \sqrt{1/\varepsilon}$ and so the properties are indistinguishable.

We showed in Theorem 2 that being the empty graph is testable, and therefore, by Theorem 6, so is Q. $\square$

3.6 Uniformity Conditions

We encoded the characteristic function for an arbitrary property while proving Theorem 7. There was no particular requirement for the property to be decidable. The existence of undecidable properties that are testable warrants additional consideration.

We noted that our definition of testability (Definition 6) is non-uniform in the sense that it does not require the various ε-testers to be computable given ε. It is reasonable to expect a situation similar to that of circuit complexity classes such as AC^0, where the non-uniform versions contain undecidable properties and the uniform versions do not (see, for example, Straubing [45]). We will see that this is the case, thereby showing that the non-uniformly testable properties strictly contain the uniformly testable properties. Alon and Shapira [8] have shown the same separation by using a decidable graph property.

That probabilistic machines do not compute properties that are deterministically uncomputable was first shown by de Leeuw *et al.* [30]. We have defined ε-testers to be randomized *algorithms*, and so we assume that the probability of a given choice being made by an ε-tester is a computable real number. Machines that make choices with noncomputable probabilities have additional power, see, e.g., de Leeuw [30].

We again consider loop-free graph properties and so the maximum R_i-distance is the usual distance.

Theorem 8. *There exist undecidable properties that are (non-uniformly) testable.*

Proof. We define a graph property P that is not decidable but is testable. As shown in Hopcroft and Ullman [26], there is an enumeration of all and only Turing machines. Let M_i refer to the i-th machine in this enumeration. A graph G with size n has property P iff either G is empty and M_i does not halt on the empty string, or G has exactly one edge and M_i does halt on the empty string.

First, P is clearly undecidable. Given i, the problem of deciding whether machine M_i halts on the empty string is a canonical RE-complete property, and it is simple to reduce this to P. On input i, output a graph with i vertices and exactly one edge.

P is indistinguishable from the empty graph in exactly the same way as in the proof to Theorem 7 and so it is testable. □

The non-uniformness of these testers can be seen by observing that for $n < \sqrt{1/\varepsilon}$ we are required to decide, correctly with high probability, whether machine M_i halts on the empty string. Once ε is fixed, there are only a finite number of machines for which we must decide this question, and so their behavior can be given in a finite list. However, in the case of uniform testability, we must be able to construct these finite lists for every value of ε, which would contradict the undecidability of P.

We show this more formally in the following. All uniform testers are probabilistic machines and, by choosing $\varepsilon > 0$ as a function of n, we can remove the "approximation" in "probabilistic approximation algorithm." Of course we then make a number of queries that depends on the input size.

Proposition 1. *All uniformly testable properties can be decided by a probabilistic Turing machine with success probability at least* 2/3.

Proof. Assume property P of structures with vocabulary τ is testable. On input A with universe size n, choose ε such that $\varepsilon < 1/\sum_i n^{a_i}$, for example, $\varepsilon = \frac{1}{1+\sum_i n^{a_i}}$. Run the ε-test on A and output the result. The tester must differentiate between inputs having P and inputs being ε-far from having P. For A, a structure with universe size n, it must therefore distinguish between those that have the property and those that do not, with probability 2/3, as desired. □

Corollary 4. *All uniformly testable properties are decidable.*

Proof. Convert the probabilistic machine of Proposition 1 using the following generic construction.

All probabilistic machines can be modified such that their randomness is taken from a special binary "random tape" that is randomly fixed when the machine is started, in which each digit is 0 or 1 with equal probability.

All halting probabilistic machines must eventually halt, regardless of the random choices made. We can then simulate the machine over all initial segments of increasing lengths, keeping track of "accepting," "rejecting" and "still running" states. Once any given segment has halted, all random strings beginning with that initial segment must also halt. Therefore, the percentage of halting paths is increasing, and we shall eventually reach a length such that at least 7/10 of the paths have halted. Our error probability is at most 1/3, strictly less than half of 7/10 and so we can output the decision of the majority of the halting paths. □

Theorem 8 and Corollary 4 immediately yield the following separation. A different proof using a decidable property has been given by Alon and Shapira [8].

Corollary 5. *There exist properties that are non-uniformly testable but not uniformly testable.*

3.7 Alternate Definitions of Distance

Definition 3 defines the distance between structures to be the fraction of assignments on which they disagree. As we saw in Section 3.4, this means that any possible difference in the assignments of low-arity relations is dominated by the number of high-arity tuples. In some sense this is a natural definition, as the number of low-arity tuples is asymptotically insignificant and so two structures are close if most of their assignments agree.

However, there are situations where this definition may not be ideal. Vertex-colored graphs are graphs in which each vertex is assigned one of some (usually constant) number of colors. We say that the coloring is *admissible* if for all $(x, y) \in E$, vertices x and y have different colors. If we consider 3-colored graphs with the vocabulary $\tau_C := \{E^2, R^1, G^1, B^1\}$ we might be interested in testing whether the given coloring is admissible or not. If we use our usual definition of coloring, this is equivalent to testing whether the graph is 3-colorable and ignores (for sufficiently large graphs) the existing coloring. If we wish to test whether the given coloring is nearly admissible, we need a slightly different model.

In this section we present a number of alternate definitions for the distance between structures. In testing we wish to distinguish structures that have a desired property and those that are far from the property, and so modifying the definition of distance changes the task of testing. We will also show the relationships between the sets of testable properties that arise from each of our definitions. As in Definition 3, the symbol $\oplus$ denotes exclusive-or.

Definition 18. *Let A,$B \in STRUC(\tau)$ be structures such that $\#(A) = \#(B) = n$. The* r-distance *between structures A and B is*

$$\mathrm{rdist}(A, B) := \max_{1 \leq i \leq s} \frac{|\{\mathbf{x} \mid \mathbf{x} \in U^{a_i} \text{ and } R_i^A(\mathbf{x}) \oplus R_i^B(\mathbf{x})\}|}{n^{a_i}}.$$

That is, the r-distance is the maximum over the relations R_i of the fraction of R_i-assignments differing between A and B. While Definition 3 gave equal weight to each *tuple*, regardless of its arity, this definition gives equal weight to each *relation.* Tuples with lower arity can then be considered to have greater weight than those with higher arity. We will also consider the $\max_i R_i$-distance (which we will shorten to $\mathrm{mrdist}(A, B)$), which was defined using subtypes in Section 3.4 (cf. Definition 13).

We begin by showing the following simple relationship between these distances.

Theorem 9. *Let τ be a vocabulary and $A, B \in STRUC(\tau)$ be structures such that $\#(A) = \#(B)$. Then,*

$$\mathrm{dist}(A, B) \leq \mathrm{rdist}(A, B) \leq \mathrm{mrdist}(A, B).$$

Proof. We begin by showing $\mathrm{dist}(A, B) \leq \mathrm{rdist}(A, B)$. Essentially, if $\mathrm{dist}(A, B) = \varepsilon$ then an ε-fraction of the total assignments differs. If we then partition the total assignments, there must exist a partition such that at least an ε-fraction of the assignments differs in that partition.

Let $\mathrm{dist}(A, B) = \varepsilon$ and let α_i be the fraction of R_i-assignments that differs between the structures, i.e.,

$$\alpha_i := \frac{|\{\mathbf{x} \mid \mathbf{x} \in U^{a_i} \text{ and } R_i^A(\mathbf{x}) \oplus R_i^B(\mathbf{x})\}|}{n^{a_i}}.$$

Then, $\mathrm{rdist}(A, B) = \max_i \alpha_i$ and we can write $\mathrm{dist}(A, B)$ in terms of the α_i:

$$\mathrm{dist}(A, B) = \frac{\sum_i \alpha_i n^{a_i}}{\sum_i n^{a_i}} = \varepsilon.$$

This implies that $\sum_i \alpha_i n^{a_i} = \varepsilon \sum_i n^{a_i}$. We do not know the values a_i, but if equality holds there must be at least one α_i satisfying $\alpha_i \geq \varepsilon$. This implies that $\mathrm{dist}(A, B) \leq \mathrm{rdist}(A, B)$, as desired.

Next, we show that $\mathrm{rdist}(A, B) \leq \mathrm{mrdist}(A, B)$. The proof is nearly identical to the above. If $\mathrm{rdist}(A, B) = \varepsilon$ then there is an R_i such that an ε-fraction of the R_i-assignments differs between the structures. If we partition the R_i-assignments into the subtypes of R_i (which are disjoint), then there must be some partition such that at least an ε-fraction of the assignments in that partition differ. As above we let $\mathrm{rdist}(A, B) = \varepsilon$. Let R_i be a relation such that the R_i-distance between A and B is ε and let α_j be the fraction of assignments in subtype S_j of R_i that differ between the structures. Using the notation of Section 3.4, $i := \arg\max_k R_k\text{-}\mathrm{dist}(A, B)$ and $\alpha_j = S_j\text{-}\mathrm{dist}(A, B)$ for subtypes S_j of R_i.

Then, $\mathrm{mrdist}(A, B) = \max_j \alpha_j$ and

$$\mathrm{rdist}(A, B) = \frac{\sum_j \alpha_j |S_j|}{\sum_j |S_j|} = \varepsilon.$$

By exactly the same argument as above, there must exist an $\alpha_j \geq \varepsilon$ and so $\mathrm{rdist}(A, B) \leq \mathrm{mrdist}(A, B)$. $\square$

There exist vocabularies such as $\tau_S = \{S^1\}$ where equality is attained in each of the above inequalities and also vocabularies where equality is not attained.

Theorem 9 immediately admits the following nice corollary, where we let T be the set of testable properties with our usual definitions, T_r be the set of testable properties using the rdist definition for distance and T_{mr} be the set of testable properties using the mrdist definition for distance.

Corollary 6. *$T_{mr} \subseteq T_r \subseteq T$.*

Proof. Assume we have a T_{mr} ε-tester for property P. Then, it distinguishes with probability at least $2/3$ between structures with P and structures A such that mrdist $\geq \varepsilon$. We will show that this is also a T_r ε-tester for P. Let A denote the input structure. If A has property P then our tester accepts with probability at least $2/3$, as desired. We saw above that $\text{rdist}(A, B) \leq \text{mrdist}(A, B)$ and so if $\text{rdist}(A, B) \geq \varepsilon$ then $\text{mrdist}(A, B) \geq \varepsilon$ and the tester rejects with probability at least $2/3$, as desired.

The proof of $T_r \subseteq T$ is analogous to the above. □

In fact, we can show both of these inclusions are strict and so it is strictly easier to test using Definition 3 than the others. Of course, it is possible for equality to hold if we restrict ourselves to a fixed vocabulary. We have noted above that the definitions are all equivalent in the case of binary strings. In the case of graphs or other structures with exactly one predicate symbol which is not monadic, $\text{rdist}(G, G') = \text{dist}(G, G')$ and so testing these properties with the dist definition is equivalent to testing with the rdist definition for distance. The proof of Theorem 10 will show that testing such properties with the mrdist definition for distance is not equivalent.

Theorem 10. *$T_{mr} \subset T_r \subset T$.*

Proof. The inclusions are shown in Corollary 6 and so it suffices to show the separations. We begin by showing that $T \backslash T_r$ is not empty. To do this we consider the vocabulary $\tau_C := \{E^2, S^1\}$ which we can interpret as the vocabulary of (not necessarily admissible) one-colored graphs.

We will show that $P_1 \in T \backslash T_r$, where P_1 is the set of structures A with vocabulary τ_C such that the S assignments encode the language $L = \{u\overleftarrow{u}v\overleftarrow{v}\}$ from Theorem 4. That is, A has P_1 if there is some $0 \leq k \leq n/2$ such that for all $0 \leq i < k$, $S(i)$ is true iff $S(2k - 1 - i)$ is true and for all $0 \leq j < (n - 2k)/2$, $S(2k + j)$ is true iff $S(n - 1 - j)$ is true. The property is decided only by the low-arity relation S; the E relation exists only to provide "padding" so that the property is testable using the dist definition for distance.

We begin by showing that P_1 is testable using Definition 3. An input structure A with a universe of odd size cannot have P_1. With our definitions, a tester can begin by checking the parity of n and then reject if it is odd. We can therefore assume in the following that the size of the universe is even. It is also possible to define P slightly differently to avoid this problem, for example requiring the S assignments to be of the form $u\overleftarrow{u}v\overleftarrow{v}$ *or* $1u\overleftarrow{u}v\overleftarrow{v}$. However, it is simpler for us to be consistent with Theorem 4.

Lemma 5. *Property P_1 is testable under the dist definition for distance.*

Proof. The proof is similar to that of Corollary 3. We again begin by showing that for sufficiently large n, there is no structure A with universe size n such that $\text{dist}(A, P_1) \geq \varepsilon$.

For any (even) n, there exist possible $w \in \{0,1\}^n$ that are of the form $u\overleftarrow{u}v\overleftarrow{v}$, such as 1^n. Given A we create A' by changing all $S(i)$ assignments to be true. This involves at most n modifications and so $\text{dist}(A, P_1) \leq \text{dist}(A, A') = O(n)/\Theta(n^2) < \varepsilon$, where the final inequality holds for sufficiently large n. Let $N(\varepsilon)$ be the smallest value of n for which the inequality holds. Then, the following is an ε-tester for P_1, where the input has universe size n.

1. If $n < N(\varepsilon)$, query all assignments and output whether the input has P_1.

2. Otherwise, accept.

If the input A has P_1, we will accept with zero error. If $\text{dist}(A, P_1) \geq \varepsilon$, then $n < N(\varepsilon)$ by the above. In this case we query all assignments and will reject with zero error.

□ Lemma 5

It remains to show that P_1 is not testable when using the rdist definition for distance. We do this by showing that if it were testable under the rdist definition, we would be able to construct testers for $\{u\overleftarrow{u}v\overleftarrow{v}\}$, contradicting Theorem 4.

Lemma 6. *Property P_1 is not testable under the rdist definition for distance.*

Proof. Assume that P_1 is testable under the rdist definition for distance. Then, there exist ε-testers T^ε using this definition for all $\varepsilon > 0$. We will show that the following is an ε-tester using *Definition 3* for the language L of Theorem 4. We denote the input with w and note that w is a binary string with vocabulary $\tau_S = \{S^1\}$.

1. Run T^ε and intercept all queries.

2. When a query is made for $S(i)$, return the value of $S(i)$ in w.

3. When a query is made for $E(i, j)$, return 0.

4. Output the decision of T^ε.

That is, given a string w, we simulate T^ε on the structure $A \in STRUC(\tau_C)$ that has the same universe size as w, agrees with w on the S assignments and where all E assignments are false.

Assume that $w \in L$ and let n be the size of the universe. Then, any $A \in STRUC(\tau_C)$ with universe size n that satisfies $\forall i : S^w(i) \leftrightarrow S^A(i)$ has property P_1. That is, if w and A agree on the S assignments, $w \in L$ implies $A \in P_1$. In this case, T^ε will accept A with probability at least 2/3. Therefore, our tester will also accept w with probability at least 2/3, as desired.

Now, assume that $\mathrm{dist}(w, L) \geq \varepsilon$. In this case, $\mathrm{rdist}(A, P_1) = \mathrm{dist}(w, L) \geq \varepsilon$ and so T^ε will reject A with probability at least 2/3. Our tester will therefore reject such w with probability at least 2/3 as desired.

We have constructed ε-testers for the language L shown to be untestable in Theorem 4, and therefore P_1 must be untestable under the rdist definition of distance. □Lemma 6

Lemmata 5 and 6, together with Corollary 6 show $T_r \subset T$. We now show the separation $T_{mr} \subset T_r$. The proof is similar: We construct a property where there is sufficient "padding" to make T_r testing simple but T_{mr} testing would contradict Theorem 4.

We use the following property P_2 of graphs ($\tau_G = \{E^2\}$). A graph G has P_2 if the "loops" $E(i, i)$ encode the language $\{u\overleftarrow{u}v\overleftarrow{v}\}$ from Theorem 4. That is, P_2 is determined only by the linear number of loops and the remaining E assignments can be arbitrary. More formally, a graph G with universe size n has P_2 if there exists some $0 \leq k \leq n/2$ such that for all $0 \leq i < k$, $E(i, i)$ is true iff $E(2k - 1 - i, 2k - 1 - i)$ is true and for all $0 \leq j < (n - 2k)/2$, $E(2k + j, 2k + j)$ is true iff $E(n - 1 - j, n - 1 - j)$ is true.

As before, we first show that P_2 is testable using the rdist definition for distance. The tester can again begin by checking the parity of the universe, and so we assume that n is even.

Lemma 7. *Property P_2 is testable under the rdist definition for distance.*

Proof. As before, we begin by showing that for sufficiently large n, there is no graph G with universe size n such that $\mathrm{rdist}(G, P_2) \geq \varepsilon$.

Recall that for all (even) n there exist possible $w \in \{0, 1\}^n$ that are of the form $u\overleftarrow{u}v\overleftarrow{v}$, for example 1^n. Given G we create G' by changing all $E(i, i)$ assignments to be true. This involves at most n modifications. There is only one relation symbol, and so $\mathrm{rdist}(G, P_2) = \mathrm{dist}(G, P_2) \leq \mathrm{dist}(G, G')$. There are n^2 possible E assignments, and so this is $O(n)/n^2 < \varepsilon$ where the inequality holds for sufficiently large n. Letting $N(\varepsilon)$ be the smallest value of n for which the inequality holds, we can easily construct a tester as in Lemma 5 by replacing "P_1" with "P_2". □Lemma 7

It remains to show that P_2 is not testable under the mrdist definition for distance. The proof is nearly identical to that of Lemma 6; given a T_{mr} style tester for P_2, we construct testers using *Definition 3* for the language $\{u\overleftarrow{u}v\overleftarrow{v}\}$ that was proven untestable in Theorem 4.

Lemma 8. *Property P_2 is not testable under the mrdist definition for distance.*

Proof. Assume that P_2 is testable under the rdist definition for distance. Then, there exist ε-testers T^ε using this definition for all $\varepsilon > 0$. We will show that the following is an ε-tester using *Definition 3* for the language L of Theorem 4. We denote the input by w and note that it is a binary string with vocabulary $\tau_S = \{S^1\}$.

1. Run T^ε and intercept all queries.

2. When a query is made for $E(i, i)$, return the value of $S(i)$ in w.
3. When a query is made for $E(i, j)$ and $i \neq j$, return 0.
4. Output the decision of T^ε.

Given a string w, we simulate T^ε on the graph G with "loops" in the graph corresponding to bits of w and all other edges absent. Note that G has the same universe size as w and that G has P_2 iff $w \in L$. Therefore, if $w \in L$, our tester will accept with probability at least 2/3, as desired.

Now, assume that $\text{dist}(w, L) \geq \varepsilon$. There are exactly two subtypes of a binary relation: the loops and the non-loops. The S-distance (see Definition 12) for the subtype S of non-loops is zero, and so $\text{mrdist}(G, P_2) = S_L\text{-dist}(G, P_2)$ where S_L is the subtype of loops $(\{\{1, 2\}\})$. This in turn is equal to $\text{dist}(w, L)$ by the definition of G and so $\text{mrdist}(G, P_2) \geq \varepsilon$. The tester T^ε will then reject G with probability at least 2/3, and so our tester will reject w with probability at least 2/3. □Lemma 8

Lemmata 7 and 8, together with Corollary 6 yield the desired $T_{mr} \subset T_r$. We have already shown $T_r \subset T$ and so the proof is complete. □Theorem 10

Chapter 4

The Classification Problem for Testability

In this section we consider the classification problem of first-order logic for testability. That is, our goal is to reach a complete classification of the prefix classes of first-order logic into testable classes (those that contain only testable properties) and untestable classes (those that contain an untestable property). This program is inspired by the traditional classification problem, that of decidable and undecidable classes, and also by results in property testing related to the testability of certain logical classes such as those by Alon *et al.* [2].

We begin by briefly describing the traditional classification problem, and in particular focus on the results that allow us to draw parallels to recent results in property testing. Next, we introduce the previous results that begin a classification for testability. Finally, we prove an additional class, traditionally known as Ackermann's class with equality, to be a testable class and briefly mention a few open problems.

4.1 History of the Classification Problem

Here we outline some of the history of the traditional classification problem, focusing especially on results where we can draw an explicit comparison with testability results. Börger *et al.* [11] provide the complete classification and its history, as well as proofs and an overview of related topics.

The classical decision problem was considered the central problem of logic in the 20th century. In this problem, we are given a sentence of first-order logic and wish to decide whether or not it is satisfiable (or, roughly equivalently, valid). Although Church and Turing showed the problem to be undecidable in general, Löwenheim [32] had shown in 1915 that the monadic case is decidable. There was therefore an enormous effort to understand exactly which classes of formulas are decidable and which classes are not.

The traditional classification problem is now considered to be essentially resolved and a complete classification has been obtained. By *complete* we mean that for any prefix class of first-order logic, either the class is contained in a class that is classified as decidable or it contains a class that is classified as undecidable. That such a complete and *finite* classification exists seems to be quite fortunate.

4.1.1 Gurevich's Classifiability Theorem

That a complete and finite classification of the prefix vocabulary classes of first-order logic exists is explained by Gurevich's Classifiability Theorem (see Section 2.3 of Börger *et al.* [11]). Fortunately, this phenomenon is not restricted to decidability and the theorem gives sufficient criteria for such a classification to exist. A complete statement and proof can be found in Börger *et al.* [11] while Gurevich [24] gives a nice introduction to the theorem and its history.

We can summarize Gurevich's Classifiability Theorem in the following way. Let D refer to the set of prefix vocabulary classes that have the property in question (in our case, D is the set of testable prefix vocabulary classes) and U refer to its negation (the untestable prefix vocabulary classes). Then, if D is closed downward and under finite unions, there exists a finite set M of the minimal closed classes of U such that U is the upwards closure of M and all members of M are standard[1].

The testable classes are closed downwards and we have shown (Theorem 5) that they are closed under finite unions. Therefore, there exists a finite set M of standard classes such that every prefix vocabulary class that contains an untestable property also contains one of the classes in M. Written in a table, M is a finite way to give a complete classification of the prefix vocabulary classes for testability.

Our goal of eventually finding a complete characterization of the testable prefix classes is then obtainable: there exists a finite table providing such a classification. In the following section we state some of the classical results for the classification for decidability and compare them with what is known regarding the classification for testability. It is worth noting that we do not necessarily state the optimal classical results, but instead focus on those that closely parallel the known results for testability.

4.1.2 Classification Similarities

Monadic first-order logic[2] was the first fragment of first-order logic to be proven decidable. We begin with this result, which is due to Löwenheim [32]. This subsection is an overview of the similarities and does not contain proofs. Later results in Subsection 4.3 depend on the testability results for monadic first-order logic, which we review more formally in Section 4.2.

Monadic logics are very well studied in the literature. Of particular relevance to us are the results connecting these logics to formal languages. Perhaps the most well-known is

[1]Standard classes are those with "nice" representations in our notation, see Börger *et al.* [11].

[2]Monadic first-order logic is the set of formulas in which all predicate symbols are monadic.

Büchi's [12] result that monadic *second*-order logic characterizes the regular languages. We are focused on (a very restricted) first-order logic, but we can at least conclude that all monadic first-order formulas express regular properties, given that monadic first-order logic is a subset of monadic second-order logic.

More immediately relevant is the result by McNaughton and Papert [33] that monadic first-order logic characterizes the *star-free* regular languages. Their definition of first-order logic includes an ordering predicate $\leq$ as well as some additional arithmetic that is not present in our logic, and so the properties expressible in our first-order logic are a strict subset of the star-free regular languages.

In any case, the star-free regular languages are a subset of the regular languages. Together with the result by Alon *et al.* [4], this immediately implies that all properties expressible in monadic first-order logic are testable. Using an extension of our notation from Section 2.4, we can say that $[all, (\omega)]_=$ is testable. This is the class proven decidable by Löwenheim [32], providing the first similarity between the classification for decidability and that for testability.

However, we can say more than this. As previously mentioned, Büchi [12] showed that monadic *second*-order logic characterizes the regular languages. As Alon *et al.* [4] themselves note, the combination of their result and Büchi's implies that all monadic second-order formulas express testable properties. This is not part of the classification problem for first-order logic, but it is worth making a comparison with Skolem's [43] extension to second-order logic of Löwenheim's result. That is, monadic second-order logic is another example of a class that is both testable and decidable.

If we return to first-order logic, Skolem [44] showed a prefix vocabulary class to be a *reduction class*. As a result of developing Skolem Normal Form, he showed the class of formulas where all universal quantifiers precede all existential quantifiers, $[\forall^*\exists^*, all]$ is a reduction class. Although it was of course unknown at the time, this implies that the class is undecidable. This class is not a minimal undecidable class and Skolem's result was later improved.

Alon *et al.* [2] considered testing *graph properties* and showed there exists an untestable property (an encoding of graph isomorphism) that is expressible in $[\forall^*\exists^*, (0,1)]_=$. This is not identical to the undecidable class considered by Skolem [44], but it is close enough to be interesting. Among the several improvements of Skolem's result is, for example, Surányi's [46] result that $[\forall^3\exists, (0,\omega)]$ is a reduction class. The encoding of graph isomorphism shown to be untestable by Alon *et al.* [2] can be expressed as a formula with twelve universal quantifiers and five existential quantifiers[3] while Theorem 12 below implies that one universal quantifier is not enough to express any untestable graph property. It would be worthwhile to attempt to find the minimum number of quantifiers needed to express such a property, and also to look at other encodings of isomorphisms.

Alon *et al.* [2] also showed a positive result: that $[\exists^*\forall^*, (0,1)]_=$ is testable. The restriction to graphs here is more unfortunate. The class $[\exists^*\forall^*, all]_=$ for pure predicate logic is

[3]That is, by a formula in $[\forall^{12}\exists^5, (0,1)]_=$.

very well-known and generally called Ramsey's class. This is a *maximal* decidable class whose decidability was first shown by Ramsey [37]. A generalization of the testability result to arbitrary vocabularies would be very desirable.

4.2 Testability of Monadic First-Order Logic

In Subsection 4.1.2 we mentioned that we can combine the result by McNaughton and Papert [33] that monadic first-order logic[4] characterizes the star-free regular languages with that of Alon *et al.* [4] showing the regular languages are testable. This allows us to conclude that all properties expressible in monadic first-order logic are testable. However, we will use this result in our proof for the testability of Ackermann's class with equality in Section 4.3, and so we consider this combination more formally.

Our goal is to prove that monadic first-order logic, *as we have defined it*, expresses only testable properties. Our definition of first-order logic does not contain ordering or symbols for successor and predecessor, unlike the definitions used by McNaughton and Papert [33].

The proof is structured in the following way. First, in Subsection 4.2.2 we show that all properties expressible by monadic first-order sentences can be expressed in a certain way which we can easily translate to a regular expression. Regular expressions characterize the regular languages and so all of our properties are regular. This part of the proof is based on that given by McNaughton and Papert [33] although our version is simplified due to the absence of the ordering and other symbols mentioned above.

Next, Alon *et al.* [4] have shown that all regular languages are testable. The proof uses the characterization of regular languages as finite automata. We omit the detailed proof here for space considerations. Basic results concerning the regular languages, such as the equivalence of regular expressions and finite automata, can be found in Hopcroft and Ullman [26].

4.2.1 Basics

Our proofs will use the vocabulary $\tau_M := \{R_1^1, \ldots, R_s^1\}$, and so there are s monadic predicate symbols named R_i. As always, if the universe of such a structure has size n, we label it with the non-negative integers $\{0, 1, \ldots, n-1\}$ and map these to the character positions of n-character strings from left to right. That is, universe member 0 corresponds to the leftmost character position in the string and $n-1$ to the rightmost.

Regular languages are defined over alphabets, traditionally denoted by Σ. The alphabet corresponding to τ_M will have 2^s symbols, which we will name with binary strings of length s in the obvious way, $\Sigma := \{00\cdots00, \ldots, 11\cdots11\}$. These denote the possible assignments $R_i(x)$ for some fixed character position x. For example, the leftmost character

[4]Their definition of monadic first-order logic allows ordering and other symbols that are not present in ours.

in a string is $00\cdots00$ iff $R_i(0)$ is false for all $0 \le i \le s$. Likewise, the rightmost character is $00\cdots01$ iff $R_i(n-1)$ is false for all $0 \le i < s$ and $R_s(n-1)$ is true. Any reasonable encoding will work provided that the character at position i can be determined using a constant number of queries. We will use c to denote characters in Σ.

The basic idea is to treat formulas $\exists x : \psi$ with the regular expression $\Sigma^* R \Sigma^*$, where R is a union over the characters for which ψ holds true. More complicated formulas are handled largely by using the closure properties of regular languages.

4.2.2 Monadic First-Order Logic is Regular

The goal of this subsection is to show the following theorem. This is a weaker result than that shown by McNaughton and Papert [33].

Theorem 11. *All properties expressible in monadic first-order logic are regular languages.*

Proof. We are given a monadic first-order formula φ of type $\tau_M = \{R_1^1, \ldots, R_s^1\}$. Without loss of generality, we can assume that φ is in prenex normal form and by renaming the variables, that no variable is bound by more than one quantifier. We rename the variables to be $x_1, \ldots, x_a$ in order, so that φ is of the form

$$\varphi = \pi_1 x_1 \pi_2 x_2 \cdots \pi_a x_a : \psi$$

where each π_j is either $\exists$ or $\forall$ and ψ is quantifier free.

Regular expressions are inherently ordered in the sense that they are written from left to right. However, we do not know if $x_1 < x_2$ in formulas such as $\exists x_1 \exists x_2 : \psi$. This will be handled by breaking the formula into a disjunction of three formulas, each enforcing an order. One formula will correspond to $x_1 < x_2$, one to $x_1 > x_2$ and one to $x_1 = x_2$. We do this by using restricted quantifiers.

Definition 19. *Quantifiers of the form $\exists_{\tau_1}^{\tau_2} x : \psi$ or $\forall_{\tau_1}^{\tau_2} x : \psi$ are* restricted quantifiers. *These quantifiers are interpreted in the following way. The formula $\exists_{\tau_1}^{\tau_2} x : \psi$ is equivalent to $\exists x : ((\tau_1 \le x \le \tau_2) \wedge \psi)$ and the formula $\forall_{\tau_1}^{\tau_2} x : \psi$ is equivalent to $\forall x : ((\tau_1 \le x \le \tau_2) \rightarrow \psi)$.*

The *range* of the quantified variable is restricted by τ_1 and τ_2. Note that these quantifiers are *not* part of our definition of monadic first-order logic. However, we will show that all formulas in our logic can be expressed in a special form using these quantifiers.

First, we replace all universal quantifiers $\forall x_i : \psi$ with duals of existential quantifiers $\neg\exists x_i : \neg\psi$. Next, we will convert all quantifiers to restricted quantifiers and ensure that the current formula is always equivalent to the original.

We begin with x_1. There are no variables in the ordering yet and so we replace $\exists x_1 : \psi$ with $\exists_0^{n-1} x_1 : \psi$, keeping any initial negation. Inductively, we consider x_{i+1} and have existing orderings over the i variables that have already been handled.

Let the current formula be $\varphi = \alpha(\exists x_{i+1} : \psi)$, where α denotes the part of the formula preceding the quantifier on x_{i+1}. We replace $\gamma := (\exists x_{i+1} : \psi)$ with an equivalent disjunction $(\gamma_1 \vee \cdots \vee \gamma_{2i+1})$ of at most $2i + 1$ formulas that correspond to the possible new orderings, creating

$$\varphi' = \alpha(\gamma_1 \vee \cdots \vee \gamma_{2i+1}),$$

which is logically equivalent to φ.

There are at most i cases where x_{i+1} is equal to one of the previous variables. These are handled by letting γ_j be the formula γ with the quantifier $\exists x_{i+1}$ removed and all occurrences of x_{i+1} replaced with x_j, for $1 \leq j \leq i$. Cases where more than two variables are equal to each other can be easily handled in several ways, for example by renaming variables or "skipping" removed quantifiers.

By induction, in φ there is a fixed ordering on the variables that have already been handled. For each adjacent pair (α, β) in this ordering we construct a γ_j that is identical to the original formula γ except that $\exists x_{i+1}$ is replaced with $\exists_{\alpha+1}^{\beta-1} x_{i+1}$. In addition, we must add formulas where x_{i+1} is the smallest variable as well as the largest. These formulas are constructed by replacing the quantifier with $\exists_0^{\alpha-1} x_{i+1}$ and $\exists_{\beta+1}^{n-1} x_{i+1}$, where α is the minimum and β the maximum element in the ordering. This is at most an additional $i + 1$ formulas and the resulting φ' is equivalent to the original formula.

We continue applying the above process until all quantifiers have been replaced by restricted quantifiers. At this point we have a (considerably larger) formula such that there is always an ordering of the variables. It is now simple to determine whether equality holds between two variables and so we can replace all instances of $x_i = x_j$ with logical truth, written $\top$, if $i = j$ and with logical falsehood, written $\bot$, otherwise. We can then make the obvious simplifications if desired. The resulting formula satisfies the previous conditions and no longer contains equality symbols, which we have replaced with restricted quantifiers and case distinctions.

All predicate symbols are monadic. We move all $R_i(x)$ predicates to their minimum "depth," by which we mean that $R_i(x)$ may occur in the scope of the quantifier on x but must not occur in the scope of any variable whose quantifier occurs in the scope of that on x. This is done using the following general procedure, which is borrowed from McNaughton and Papert [33].

Let $R_i(x_j)$ be a relation violating the condition that occurs in the formula. Assume the formula φ is $\gamma\exists_{\tau_1}^{\tau_2} x_j : \psi$. We replace this with

$$\gamma\exists_{\tau_1}^{\tau_2} x_j : ((R_i(x_j) \wedge \psi_1) \vee (\neg R_i(x_j) \wedge \psi_2)) .$$

Here, ψ_1 is the result of replacing all occurrences of $R_i(x_j)$ in ψ with $\top$, and ψ_2 is the result of replacing these occurrences with $\bot$.

We also want to ensure that all quantified formulas are at their minimum depth. This is because we will use the quantifier $\exists_{\tau_1}^{\tau_2} x : \psi$ to construct a regular expression for the part of the string corresponding to the interval $[\tau_1, \tau_2]$, and quantifiers that are not at their minimum depth can "escape" this interval. We use essentially the same process that we used to move predicate symbols to their minimum depths.

We will consider the quantifier $\exists_{\tau_1}^{\tau_2} x : \psi$. By construction, there is only one case where both τ_1 and τ_2 contain no quantified variables. This is where $\tau_1 = 0$ and $\tau_2 = n - 1$, and it only occurs at the outermost level. Without loss of generality, we will assume that y is the quantified variable in $\tau_1 = y + 1$ and z is the quantified variable in τ_2. We will also assume that the quantifier on y is in the scope of the quantifier on z. Cases where $\tau_1 = 0$ *or* $\tau_2 = n - 1$ are similar although only one of these quantified variables exists, and so the minimum depth is immediately within the scope of that variable.

The formula φ is

$$\alpha \exists_{\tau_3}^{\tau^4} z \beta \exists_{\tau_5}^{\tau_6} y \gamma (\exists_{y+1}^{z-1} x : \psi) \zeta.$$

We move $(\exists_{y+1}^{z-1} x : \psi)$ to its minimum depth in the same way we moved predicate symbols, replacing φ with

$$\varphi' := \alpha \exists_{\tau_3}^{\tau^4} z \beta \exists_{\tau_5}^{\tau_6} y : \left[\Big((\exists_{y+1}^{z-1} x : \psi) \wedge (\gamma \top \zeta) \Big) \vee \Big(\neg (\exists_{y+1}^{z-1} x : \psi) \wedge (\gamma \bot \zeta) \Big) \right].$$

The quantifier on x was the innermost quantifier that did not satisfy the new condition, and so all quantifiers within ψ are at their minimum depths. There quantifiers immediately within ψ are therefore of the form $\exists_{x+1}^{z-1} x_1$ or $\exists_{y+1}^{x-1} x_2$. This is because our construction has ensured that quantifiers always range over the interval spanned by adjacent pairs in the ordering. In addition, all variables occurring within ψ are either x or they are bound by quantifiers that occur inside ψ. This is because we have moved all predicates to their minimum depth. Inductively, ψ depends only on the interval $[y + 1, z - 1]$ and so φ' is logically equivalent to φ. If the input is the empty string, then $\exists_{y+1}^{z-1} x : \psi$ is false by definition and the formula evaluates as it would have. We repeat the process as necessary.

Finally, we will ensure that in all quantifications $\varphi = \exists_{\tau_1}^{\tau_2} x : \psi$, the formula ψ is a conjunction over atomic formulas, negated atomic formulas, quantified formulas and negated quantified formulas. This is simple to do. We begin with the innermost quantification that does not satisfy the condition. If we treat quantified formulas as units, we begin by converting φ into disjunctive normal form,

$$\varphi_1 = \exists_{\tau_1}^{\tau_2} x : (\psi_1 \vee \cdots \vee \psi_c),$$

where c is the number of clauses. This is equivalent to

$$\varphi' = \left(\exists_{\tau_1}^{\tau_2} x : \psi_1 \right) \vee \cdots \vee \left(\exists_{\tau_1}^{\tau_2} x : \psi_c \right).$$

Each quantified formula is a conjunction and the process is iterated as necessary.

The result we will call a *basic formula*. These formulas satisfy the following conditions, which will help us construct regular expressions from them.

1. All quantifiers are existential.

2. All quantified formulas have a fixed ordering of the variables and no two variables are equal to each other.

3. Equality does not occur.

4. All predicate symbols occur at "minimum" depth. Predicate symbols are the only way for a variable to appear in these formulas, and so all variables x occur only at minimum depth, omitting their appearance on the restricted quantifiers.

5. All quantifiers are at their minimum depth.

6. All quantifiers are over conjunctions of atomic symbols, negated atomic symbols, quantified formulas and negated quantified formulas.

We will now show that a regular expression corresponds to each of these formulas. Recall that the regular expressions are closed under complement, union and intersection.

First, the formula $\top$ corresponds to the regular expression Σ^* and $\bot$ to $\emptyset$. The regular expressions are closed under complement, and so the regular expression for $\neg\phi$ is the complement of the expression for ϕ.

Next, all quantified formulas are of the form

$$\phi = \exists_{\tau_1+1}^{\tau_2-1} x : (\alpha \wedge \beta \wedge \gamma),$$

where α is a conjunction of monadic predicate symbols (and possibly $\top$ and $\bot$) whose atomic parts are x, β is a (possibly empty) conjunction of quantified formulas ranging from $\tau_1 + 1$ to $x - 1$, and γ is a (possibly empty) conjunction of quantified formulas ranging from $x + 1$ to $\tau_2 - 1$.

There is a (possibly empty) set of characters that correspond to assignments of the $R_i(x)$ that model α. If the set is empty, then ϕ is equivalent to $\bot$ and so the regular expression $\emptyset$ corresponds to it. Otherwise, we let C be regular expression formed by the union of these characters.

By induction we have regular expressions for each of the quantified formulas appearing in β and γ. The regular expression L corresponding to β is formed by taking the intersection of the regular expressions for each of the quantified formulas appearing in it. The regular expressions are closed under intersection, and so L exists. Likewise, the regular expression R corresponding to γ is formed by taking the intersection of the regular expressions for each of the quantified formulas appearing in it.

The regular expression corresponding to ϕ is then the concatenation LCR. This corresponds to the part of the string from positions $\tau_1 + 1$ to $\tau_2 - 1$.

Finally, disjunctions may appear outside of quantified formulas at the outermost level. These are disjunctions of quantified formulas ranging from 0 to $n - 1$ and so the regular expression corresponding to the disjunction is the union of the expressions for the formulas.

The regular languages are closed under intersection, union and complement, and so we have shown that monadic first-order logic characterizes a subset of the regular languages.

□

4.3 Ackermann's Class with Equality

In Section 4.1 we reviewed a small portion of the history of the classification problem for decidability. We found several similarities between the classification for decidability and the currently known classification for testability. In this section we consider an additional class, generally referred to as Ackermann's class with equality, and show this class to be testable.

In our notation (see Section 2.4), we denote Ackermann's class with equality 2.4) by $[\exists^*\forall\exists^*, all]_=$. It is the set of sentences containing at most one universal quantifier. Equality and any number of predicate symbols of any arities may occur, but we consider only pure predicate logic and so there are no function symbols.

Ackermann's class, without equality, was first shown to be decidable and to have the finite model property by Ackermann [1]. If we also allow equality and one unary function symbol, the resulting class is Shelah's class. The decidability of this class was proven by Shelah [41], however it does not have the finite model property.

In Section 4.1 we described similarities between the traditional classification for decidability and the currently known classification for testability. However, all of the decidable classes mentioned there have the finite model property and so all of the similarities we found also hold between the classification for the finite model property and the classification for testability. One interesting open problem is therefore to determine whether Shelah's class is testable. This would require a further generalization of our definitions to allow function symbols.

Returning to the history of Ackermann's class with equality, Kolaitis and Vardi [29] showed that the satisfiability problem for the Ackermann class with equality is complete for NEXPTIME. They also showed that a 0-1 law holds for sentences of existential second-order logic where the first-order part belongs to Ackermann's class with equality. In the absence of equality, Grädel [23] showed that satisfiability for Ackermann's class is decidable in deterministic exponential time. The main result of Chapter 4 is the testability of Ackermann's class with equality, which we show in the next section.

4.3.1 Testability of Ackermann's Class with Equality

We would like to assume that the the formulas we consider contain at least one predicate symbol with arity at least two. Therefore, we must treat the remaining cases specially, which we do now. For formulas that contain no predicate symbols, we can simply compute φ with at most one query. These are therefore trivially testable. First-order formulas that contain only monadic predicate symbols characterize a subset of the regular languages and are therefore testable, see Section 4.2. All remaining cases contain at least one predicate symbol with arity at least two.

The proof is similar to many of the proofs in Chapter 3. For an arbitrary formula in Ackermann's class with equality, we show that either there are only finitely many structures that are ε-far from having the property defined by the formula, or there are only finitely many structures that are models of the formula.

Theorem 12. *All properties expressible by a formula in* $[\exists^*\forall\exists^*, all]_=$ *are testable.*

Proof. Let P be the property expressed by an arbitrary formula φ in $[\exists^*\forall\exists^*, all]_=$. Then, we can rename the variables in φ so that it takes the following form, where ψ is quantifier-free,

$$\varphi := \exists x_1 \dots \exists x_a \forall y \exists z_1 \dots \exists z_b : \psi.$$

We can further assume that P is a property of structures with vocabulary τ, and we therefore have s relation symbols R_i of arity a_i. We let $m := \max_i(a_i)$ be the maximum arity. The structures in the proof are all implicitly of vocabulary τ.

Either property P holds for at most finitely many structures or it holds for infinitely many. We can trivially test any property that holds for at most finitely many structures by making the constant number of queries required to determine if the input is exactly one of the models. Therefore, it remains only to consider the case where φ has infinitely many models.

We will show that in this case, there are only finitely many structures $A \in STRUC(\tau)$ that satisfy $\text{dist}(A, P) \geq \varepsilon$ (cf. Lemma 9). There is therefore a function $N(\varepsilon)$ such that for any $A \in STRUC(\tau)$, if $\#(A) > N(\varepsilon)$ then the structure A is not ε-far from P. Therefore, the following is an ε-tester for P on input A with size $\#(A) = n$.

1. If $n \leq N(\varepsilon)$, query all assignments in A and decide exactly whether A has P.
2. Otherwise, accept.

It is worth noting that in the non-uniform case, the $N(\varepsilon)$ is simply a constant. It does depend on the vocabulary, but $N(\varepsilon)$ is computable and so P is also uniformly testable. The proof is complete conditional on Lemma 9, which we show next. □

Lemma 9. *Let P be the property with vocabulary τ expressed by*

$$\varphi := \exists x_1 \dots \exists x_a \forall y \exists z_1 \dots \exists z_b : \psi$$

where ψ is quantifier free. If φ has infinitely many models, then there exists an $N(\varepsilon)$ for every $\varepsilon > 0$ satisfying the following. If $A \in STRUC(\tau)$ has universe size $n > N(\varepsilon)$, then $\text{dist}(A, P) < \varepsilon$.

Proof. Let $A \in STRUC(\tau)$ be arbitrary and assume it is sufficiently large. We will show that there exists an A' such that $A' \models \varphi$ (and so A' has P) and $\text{dist}(A, A') < \varepsilon$.

We begin by showing in Lemma 10 that there must exist a model, A_1, of φ such that the size of the universe of A_1 satisfies $a + 1 \leq \#(A_1) \leq \kappa_1$, where

$$\kappa_1 := a + 3b\left(a + 2^{\sum_{i=1}^{s}\sum_{j=1}^{a_i}\binom{a_i}{j}a^{a_i-j}}\right).$$

The constant κ_1 is determined only by φ and does not depend on the input structure. It is possible to prove tighter bounds on $\#(A_1)$ but this suffices for our purposes.

Next, we will show in Lemma 11 that we can easily add additional elements to the universe of A_1 and construct larger models of φ by making only a small number of modifications. We will then use an inductive argument to show that, for sufficiently large input structures A, we can construct $A' \models \varphi$ where A and A' have the same universe size and $\operatorname{dist}(A, A') < \varepsilon$.

We now prove the two necessary lemmata.

Lemma 10. *If $\varphi := \exists x_1 \dots \exists x_a \forall y \exists z_1 \dots \exists z_b \psi$ has infinitely many models with vocabulary τ, then it has a model A_1 such that the cardinality of the universe in A_1 satisfies*

$$a + 1 \leq \#(A_1) \leq \kappa_1.$$

Proof. It suffices to consider cases where φ has infinitely many models with vocabulary τ. Let B be the smallest[5] model of φ such that $\#(B) \geq a + 1$. We are guaranteed that B exists because there are infinitely many models. The proof is by contradiction; assume $\#(B) > \kappa_1$. We will show that we can construct a smaller[6] model of φ, violating the requirement that B is the smallest. Therefore, the smallest model of φ larger than $a + 1$ must be of size at most κ_1.

We have chosen B such that it models φ, and therefore there exists a tuple of a elements $(u_1, \dots, u_a)$ such that φ is satisfied when the existential quantifiers bind these elements to $(x_1, \dots, x_a)$. If there are multiple possible choices for the x_i, we choose one arbitrarily. We now consider the x_i and the substructure induced by them to be fixed. We refer to this substructure as A_x.

There are at most $\kappa_2 := a + 2^{\sum_{i=1}^{s} \sum_{j=1}^{a_i} \binom{a_i}{j} a^{a_i - j}}$ many *distinct* structures constructed by adding an element labeled y to A_x when we include the structures where the label y is simply placed on one of the x_i. We let $v \leq \kappa_2$ be the number of such structures that occur in B and assume there is an enumeration of them.

We know that B models φ, and so for each of these v structures there exist b elements, which we call $w_1, \dots, w_b$, such that when we label w_i with z_i, the structure induced by $(x_1, \dots, x_a, y, z_1, \dots, z_b)$ models ψ. We construct $A_{i,j}$ for $1 \leq i \leq 3$ and $1 \leq j \leq v$ such that $A_{i,j}$ is a copy of the $w_1, \dots, w_b$ used for the j-th structure in our enumeration. In each of these cases, we connect the $A_{i,j}$ to A_x in the same way as in B, by modifying assignments on tuples $(A_x \cup A_{i,j})^{a_k}$.

For each w_h in $A_{i,j}$, we must consider the case where y is bound to w_h. By construction the substructure induced by $(x_1, \dots, x_a, y)$ occurs in B. We assume that it is the k-th structure in our enumeration. In this case we will use the elements of $A_{i+1 \bmod 3, k}$ to construct a structure satisfying ψ. Therefore, we modify the assignments as needed to create a structure identical to that in B satisfying ψ and claim that the resulting structure satisfies φ. Before this step we have not modified any assignments "spanning"

[5]If there are multiple "smallest" models, choose one arbitrarily.

[6]But still of size at least $a + 1$.

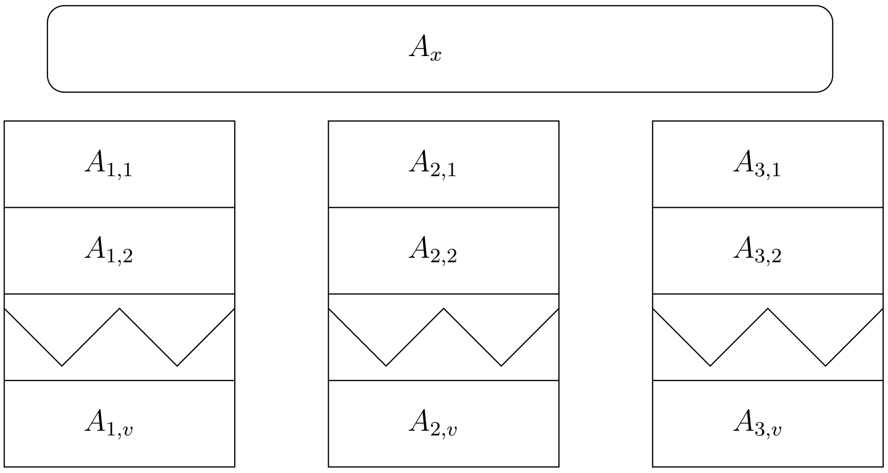

Figure 4.3.1: Structure A_1

the columns of A_1 (see Figure 4.3.1) and so there are no assignments that we modify more than once.

Counting the number of elements, we have at most $a + 3bv \leq a + 3b\kappa_2 = \kappa_1$. If $\#(A_1) < a + 1$, we can "grow" it by adding additional columns to A_1 in Figure 4.3.1 to construct a new A_1 satisfying the requirements of the Lemma. □Lemma 10

Our goal is to construct an A' that has property P by making only a small number of changes. We will make the substructure induced by a constant-sized part of A' equivalent to the A_1 shown to exist above. The size of A_1 is upper-bounded by a constant, and so this will require only constantly many modifications to assignments. However, we must also deal with the other elements of A. We will use the following lemma for these remaining elements.

Lemma 11. *Let $\varphi = \exists x_1 \ldots \exists x_a \forall y \exists z_1 \ldots \exists z_b \psi$ and assume there exists an $A \models \varphi$, such that $\#(A) \geq a + 1$. Then, for any structure A' containing A as an induced substructure where $\#(A') = \#(A) + 1$, we can construct a model of φ by modifying a constant number of assignments.*

Proof. A' contains an induced copy of A and one additional element, which we will denote by q. By assumption, A is a model of φ and therefore contains an a-tuple $(u_1, \ldots, u_a)$ such that the formula is satisfied when x_i is bound to u_i. In addition, the size of A is at least $a + 1$, and so it also contains at least one additional element, which we will call p. We will now make q equivalent to p.

We begin by modifying the assignments necessary to make the structure induced by $(x_1, \ldots, x_a, q)$ identical to that induced by $(x_1, \ldots, x_a, p)$. This requires at most $\sum_{i=1}^{s} \sum_{j=1}^{a_i} \binom{a_i}{j} a^{a_i - j} = O(1)$ modifications. There must exist $(v_1, \ldots, v_b)$ in A such that ψ is satisfied when z_i is bound to v_i and y to p. We now modify the assignments necessary to make the structure induced by $(q, v_1, \ldots, v_b)$ identical to that induced by $(p, v_1, \ldots, v_b)$[7]. This requires at most an additional $\sum_{i=1}^{s} \sum_{j=1}^{a_i} \binom{a_i}{j} b^{a_i - j} = O(1)$ modifications. The resulting structure has $\#(A) + 1$ elements, models φ and was constructed from A' by making only a constant number of modifications to assignments. $\square$Lemma 11

Now, given a sufficiently large structure A, we will construct A' by making only a linear[8] number of modifications, and ensure that $A' \models \varphi$.

We begin by selecting arbitrarily $\#(A_1)$ elements and making the induced substructure identical to the A_1 proven to exist in Lemma 10. The size of A_1 is constant and therefore there are at most a constant number of assignments to modify. The A_1 from Lemma 10 is of size at least $a + 1$ and so it satisfies the conditions of Lemma 11. We proceed inductively: select an element q of A that has not yet been selected. Using Lemma 10, make the substructure induced by the elements selected so far a model of φ. Each step requires $O(1)$ modifications and we require $\Theta(n)$ steps. By induction we can then construct A' from A such that $A' \models \varphi$ by making at most $O(n)$ modifications to the assignments of relations in A.

Then, by definition,

$$\text{dist}(A, A') = \frac{O(n)}{\Theta(n^m)} < \varepsilon.$$

The maximum arity is at least two, and so the inequality holds for n sufficiently large. This completes the proof of Lemma 9 and therefore that of Theorem 12. $\square$Lemma 9

[7]The case where $v_i = p$ can be handled either by assuming it does not occur or by replacing v_i with q in $(q, v_1, \ldots, v_b)$.

[8]Linear in $n = \#(A)$.

Chapter 5

Conclusion

In this thesis we have introduced a generalization of property testing which we call *relational property testing*. We gave a number of basic results in Chapter 3. In particular, in Section 3.7 we considered a number of natural alternative definitions of distance and showed the relationships between the resulting definitions of testability. The definitions form a strict hierarchy, and the "best" definition depends on the problem in question.

Relational databases are perhaps the most obvious example of massive structures where it would be promising to consider applications of property testing. Relational property testing is a natural way to characterize this problem. In addition, properties of databases are often given by queries written in formal languages such as SQL and so it is very natural to consider the testability of properties expressible in various syntactic restrictions of formal languages. Finally, a generalization of property testing such as ours is required if we wish to consider this kind of classification problem.

The second major topic of this thesis is the *classification problem for testability*, which we considered in Chapter 4. This problem is inspired by the classical problem for decidability. The major result of Chapter 4 is the testability of Ackermann's class with equality, another example of a similarity between the classification for testability and the classical one for decidability.

Acknowledgements

I am indebted to my advisor, Prof. Thomas Zeugmann, for his constant encouragement and advice, which has significantly improved both my research life and this thesis. I am very grateful for the support of the Japanese government via a Monbukagakusho scholarship. Finally, I will always be indebted to my family for their love and support, and for putting up with me being so far away for so long.

Bibliography

[1] Wilhelm Ackermann. Über die Erfüllbarkeit gewisser Zählausdrücke. *Math. Annalen*, 100:638–649, 1928.

[2] Noga Alon, Eldar Fischer, Michael Krivelevich, and Mario Szegedy. Efficient testing of large graphs. *Combinatorica*, 20(4):451–476, 2000.

[3] Noga Alon, Eldar Fischer, Ilan Newman, and Asaf Shapira. A combinatorial characterization of the testable graph properties: It's all about regularity. In *STOC '06: Proceedings of the 38th Annual ACM Symposium on Theory of Computing*, pages 251–260, New York, NY, USA, 2006. ACM.

[4] Noga Alon, Michael Krivelevich, Ilan Newman, and Mario Szegedy. Regular languages are testable with a constant number of queries. *SIAM J. Comput.*, 30(6):1842–1862, 2001.

[5] Noga Alon and Asaf Shapira. A characterization of the (natural) graph properties testable with one-sided error. In *Proceedings, 46th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2005*, pages 429–438, Washington, DC, USA, 2005. IEEE Computer Society.

[6] Noga Alon and Asaf Shapira. Homomorphisms in graph property testing. In M. Klazar, J. Kratochvíl, M. Loebl, J. Matoušek, R. Thomas, and P. Valtr, editors, *Topics in Discrete Mathematics*, volume 26 of *Algorithms and Combinatorics*, pages 281–313. Springer, 2006.

[7] Noga Alon and Asaf Shapira. A characterization of the (natural) graph properties testable with one-sided error. *SIAM J. Comput.*, 37(6):1703–1727, 2008.

[8] Noga Alon and Asaf Shapira. A separation theorem in property testing. *Combinatorica*, 28(3):261–281, 2008.

[9] David A. Mix Barrington, Neil Immerman, and Howard Straubing. On uniformity within NC1. *J. of Comput. Syst. Sci.*, 41(3):274–306, 1990.

[10] Manuel Blum, Michael Luby, and Ronitt Rubinfeld. Self-testing/correcting with applications to numerical problems. *J. of Comput. Syst. Sci.*, 47(3):549–595, 1993.

[11] Egon Börger, Erich Grädel, and Yuri Gurevich. *The Classical Decision Problem*. Springer-Verlag, 1997.

[12] J. Richard Büchi. Weak second-order arithmetic and finite-automata. *Z. Math. Logik Grundlagen Math.*, 6:66–92, 1960.

[13] Hana Chockler and Orna Kupferman. ω-regular languages are testable with a constant number of queries. *Theoret. Comput. Sci.*, 329(1-3):71–92, 2004.

[14] Herbert B. Enderton. *A Mathematical Introduction to Logic*. Academic Press, second edition, 2000.

[15] Eldar Fischer. The art of uninformed decisions. *Bulletin of the European Association for Theoretical Computer Science*, 75:97–126, October 2001. Columns: Computational Complexity.

[16] Eldar Fischer. Testing graphs for colorability properties. *Random Struct. Algorithms*, 26(3):289–309, 2005.

[17] Rūsiņš Freivalds. Fast probabilistic algorithms. In *Mathematical Foundations of Computer Science 1979, Proceedings, 8th Symposium, Olomouc, Czechoslovakia, September 3-7, 1979*, volume 74 of *Lecture Notes in Computer Science*, pages 57–69. Springer-Verlag, 1979.

[18] Merrick L. Furst, James B. Saxe, and Michael Sipser. Parity, circuits, and the polynomial-time hierarchy. *Mathematical Systems Theory*, 17(1):13–27, 1984.

[19] O. Goldreich and D. Ron. Property testing in bounded degree graphs. *Algorithmica*, 32:302–343, 2002.

[20] Oded Goldreich, Shafi Goldwasser, and Dana Ron. Property testing and its connection to learning and approximation. *J. ACM*, 45(4):653–750, 1998.

[21] Oded Goldreich and Luca Trevisan. Three theorems regarding testing graph properties. *Random Struct. Algorithms*, 23(1):23–57, 2003.

[22] Mira Gonen and Dana Ron. On the benefits of adaptivity in property testing of dense graphs. In Moses Charikar, Klaus Jansen, Omer Reingold, and José D.P. Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques. 10th International Workshop, APPROX 2007 and 11th International Workshop, RANDOM 2007. Princeton, NJ, USA, August 2007, Proceedings*, volume 4627 of *Lecture Notes in Computer Science*, pages 525–539. Springer, 2007.

[23] Erich Grädel. Satisfiability of formulae with one $\forall$ is decidable in exponential time. *Arch. Math. Logic*, 29:256–276, 1990.

[24] Yuri Gurevich. On the classical decision problem. *Bulletin of the European Association for Theoretical Computer Science*, pages 140–150, October 1990.

[25] J. Hartmanis and R. E. Stearns. On the computational complexity of algorithms. *Transactions of the American Mathematical Society*, 117:285–306, 1965.

[26] John E. Hopcroft and Jefferey D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Adison-Wesley Publishing Company, Reading, Massachusets, USA, 1979.

[27] Juraj Hromkovič. *Design and Analysis of Randomized Algorithms: Introduction to Design Paradigms.* Springer, 2005.

[28] Skip Jordan and Thomas Zeugmann. Indistinguishability and first-order logic. In Manindra Agrawal, Dingzhu Du, Zhenhua Duan, and Angsheng Li, editors, *Theory and Applications of Models of Computation, 5th International Conference, TAMC 2008, Xi'an, China, April 2008, Proceedings*, volume 4978 of *Lecture Notes in Computer Science*, pages 94–104. Springer, 2008.

[29] Phokion G. Kolaitis and Moshe Y. Vardi. 0-1 laws and decision problems for fragments of second-order logic. *Inf. Comput.*, 87(1-2):302–338, 1990.

[30] K. de Leeuw, E. F. Moore, C. E. Shannon, and N. Shapiro. Computability by probabilistic machines. In C.E. Shannon and J. McCarthy, editors, *Automata Studies*, pages 183–212. Princeton University Press, Princeton, NJ, 1956.

[31] László Lovász. Some mathematics behind graph property testing. In Yoav Freund, László Györfi, György Turán, and Thomas Zeugmann, editors, *Algorithmic Learning Theory, 19th International Conference, ALT 2008, Budapest, Hungary, October 2008, Proceedings*, volume 5254 of *Lecture Notes in Computer Science*, page 3. Springer, 2008.

[32] Leopold Löwenheim. Über Möglichkeiten im Relativkalkül. *Math. Annalen*, 76:447–470, 1915.

[33] Robert McNaughton and Seymour Papert. *Counter-Free Automata.* M.I.T. Press, 1971.

[34] J. v. Neumann. Zur Theorie der Gesellschaftsspiele. *Math. Annalen*, 100(1):295–320, 1928.

[35] Christos M. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

[36] Michal Parnas and Dana Ron. Testing the diameter of graphs. *Random Struct. Algorithms*, 20(2):165–183, 2002.

[37] F. P. Ramsey. On a problem of formal logic. *Proc. London Math. Soc. (2)*, 30:264–286, 1930.

[38] Vojtěch Rödl and Mathias Schacht. Property testing in hypergraphs and the removal lemma. In *STOC '07: Proceedings of the 39th Annual ACM Symposium on Theory of Computing*, pages 488–495, New York, NY, USA, 2007. ACM.

[39] Dana Ron. Property testing. In Sanguthevar Rajasekaran, Panos M. Pardalos, John H. Reif, and José Rolim, editors, *Handbook of Randomized Computing*, volume II, chapter 15, pages 597–649. Kluwer Academic Publishers, 2001.

[40] Ronitt Rubinfeld and Madhu Sudan. Robust characterizations of polynomials with applications to program testing. *SIAM J. Comput.*, 25(2):252–271, 1996.

[41] Saharon Shelah. Decidability of a portion of the predicate calculus. *Israel J. Math.*, 28(1-2):32–44, 1977.

[42] Stephen Simons. Minimax theorems and their proofs. In Ding-Zhu Du and Panos M. Pardalos, editors, *Minimax and Applications*, pages 1–23. Kluwer Academic Publishers, 1995.

[43] Th. Skolem. Untersuchungen über die Axiome des Klassenkalküls und über Produktations und Summationsprobleme, welche gewisse Klassen von Aussagen betreffen. *Videnskapsselskapets skrifter, I. Matematisk-naturvidenskabelig klasse*, (3):37–71, 1919.

[44] Th. Skolem. Logisch-kombinatorische Untersuchungen über die Erfüllbarkeit oder Beweisbarkeit mathematischer Sätze nebst einem Theorem über dichte Mengen. *Videnskapsselskapets skrifter, I. Matematisk-naturvidenskabelig klasse*, (4):1–26, 1920.

[45] Howard Straubing. *Finite Automata, Formal Logic, and Circuit Complexity.* Birkhäuser, 1994.

[46] János Surányi. Contributions to the reduction theory of the decision problem: Second paper: Three universal, one existential quantifiers. *Acta Mathematica Hungarica*, 1(2–4):261–271, 1950.

[47] Alfred Tarski. Der Wahrheitsbegriff in den formalisierten Sprachen. *Studia Philosophica*, 1:261–405, 1936.

[48] Alfred Tarski. The concept of truth in formalized languages. In *Logic, Semantics, Metamathematics*, pages 152–278. Oxford University Press, Oxford, 1956.

[49] Andrew Chi-Chih Yao. Probabilistic computations: Toward a unified measure of complexity. In *18th Annual Symposium on Foundations of Computer Science*, pages 222–227. IEEE Computer Society, 1977.