THE UNIVERSITY OF CHICAGO

SYMMETRY AND EQUIVALENCE RELATIONS IN CLASSICAL AND GEOMETRIC COMPLEXITY THEORY

A DISSERTATION SUBMITTED TO
THE FACULTY OF THE DIVISION OF THE PHYSICAL SCIENCES
IN CANDIDACY FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE

BY
JOSHUA ABRAHAM GROCHOW

CHICAGO, ILLINOIS
JUNE 2012

To my parents, Jerrold Marvin Grochow and Louise Barnett Grochow

ABSTRACT

This thesis studies some of the ways in which symmetries and equivalence relations arise in classical and geometric complexity theory. The Geometric Complexity Theory program is aimed at resolving central questions in complexity such as **P** versus **NP** using techniques from algebraic geometry and representation theory. The equivalence relations we study are mostly algebraic in nature and we heavily use algebraic techniques to reason about the computational properties of these problems. We first provide a tutorial and survey on Geometric Complexity Theory to provide perspective and motivate the other problems we study.

One equivalence relation we study is MATRIX ISOMORPHISM OF MATRIX LIE ALGEBRAS, which is a problem that arises naturally in Geometric Complexity Theory. For certain cases of MATRIX ISOMORPHISM OF LIE ALGEBRAS we provide polynomial-time algorithms, and for other cases we show that the problem is as hard as GRAPH ISOMORPHISM. To our knowledge, this is the first time GRAPH ISOMORPHISM has appeared in connection with any lower bounds program.

Finally, we study algorithms for equivalence relations more generally (joint work with Lance Fortnow). Two techniques are often employed for algorithmically deciding equivalence relations: 1) finding a complete set of easily computable invariants, or 2) finding an algorithm which will compute a canonical form for each equivalence class. Some equivalence relations in the literature have been solved efficiently by other means as well. We ask whether these three conditions—having an efficient solution, having an efficiently computable complete invariant, and having an efficiently computable canonical form—are equivalent. We show that this question requires non-relativizing techniques to resolve, and provide new connections between this question and factoring integers, probabilistic algorithms, and quantum computation.

ACKNOWLEDGMENTS

First I thank my advisors. I thank Lance Fortnow for his advice, support, guidance, and collaboration on Chapter 5 and several other projects which have yet to bear fruit. I thank Ketan Mulmuley for his advice and support, as well as countless hours of discussion throughout the course of my graduate career. Without these discussions, it would not have been possible to seriously work on problems related to Geometric Complexity Theory—such as MATRIX ISOMORPHISM OF LIE ALGEBRAS (Chapter 4)—let alone to write a survey on it (Chapter 3). I thank Benson Farb for his advice, guidance, and many fruitful discussions, even when he was not officially my advisor.

I thank my thesis committee for their continued advice, prodding, and editorial support.

I thank Anne Rogers for her support and for the countless decisions regarding my career trajectory, both large and small, she helped me understand how to make.

I thank Sasha Razborov for many interesting discussions, and for enforcing a much-needed kick-in-the-pants in the middle of my graduate career that ensured I graduated in a timely fashion. I cannot imagine this duty was much fun for him, but it was a tremendous help to me.

I thank anonymous reviewers for feedback that improved the quality, clarity, and presentation of the works on which Chapters 4 and 5 are based. In particular, one of the reviewers pointed out the importance of the complexity of factoring polynomials for Chapter 4. One of the reviewers suggested that we define some sort of hybrid notion of Cohen and transitive genericity, as well as suggested the notion of **UP**-transitive genericity that are used in Chapter 5. I also thank Lane Hemaspaandra—who was our editor for the corresponding paper—and Paolo Codenotti for useful comments on a draft of Chapter 5. I thank Laci Babai for useful comments on a draft of Chapter 4, as well as pointing me to several results [20, 62] related to that chapter, and suggesting that I consider the corresponding questions for associative algebras.

I thank Stuart Kurtz and Laci Babai for several useful discussions regarding Chapter 5. In particular, Stuart suggested the use of the equivalence relation R_L, which led to Theorem

5.3.3, and Laci pointed out the canonical form for subgroup equality of permutation groups [23]. I thank Scott Aaronson for the observations leading to Section 5.3.1. I thank Andreas Blass for pointing me to the original two papers he co-authored with Gurevich [56, 57].

I thank my collaborators, on projects both finished and in progress: Lance Fortnow, László Babai, Paolo Codenotti, Youming "Jimmy" Qiao, Jonah Blasiak, and Thomas Church. It was and continues to be a pleasure to work with them. In particular, Chapter 5 is based on joint work with Lance, and Jonah helped me clarify my thoughts on MATRIX ISOMORPHISM OF LIE ALGEBRAS and together realize the equivalence with GRAPH ISOMORPHISM in Chapter 4.

I find it incredibly useful, rewarding, and fun to talk through mathematics with others, and it is my great pleasure and honor to thank Lance Fortnow, Ketan Mulmuley, Benson Farb, Thomas Church, Ian Shipman, Spencer Dowdall, Anna Marie Bohmann, Daniel Studenmund, Vipul Naik, Paolo Codenotti, Youming "Jimmy" Qiao, Chris Umans, J. M. Landsberg, Jerzy Weyman, Shrawan Kumar, Neeraj Kayal, Arkadev Chatthopadhyay, Pascal Koiran, Gerald J. Sussman, and Jonah Blasiak for not only useful and interesting discussions, but also for their infectious enthusiasm. Many discussions regarding GCT and MATRIX ISOMORPHISM OF LIE ALGEBRAS took place at the Brown-ICERM Workshop on Mathematical Aspects of **P** vs. **NP** and its Variants in August 2011, for which I would like to thank ICERM and the organizers of the workshop—J. M. Landsberg, Saugata Basu, and J. Maurice Rojas—for the invitation and support to attend the workshop.

I would especially like to thank Stuart Kurtz and Gerald J. Sussman for sharing with me some small portion of their incredible breadth of knowledge and depth of philosophy. They have both made my research career and my life more interesting.

This thesis was partially supported by K. Mulmuley's NSF Grant CCF-1017760, L. Fortnow *et al.*'s NSF Grant DMS-0652521 and fellowships from the University Chicago Department of Computer Science.

I would like to thank the members of the University of Chicago Department of Computer Science Techstaff. They've setup such a great system and were so helpful that I barely noticed all the technology I was using: I could do what I wanted, how I wanted, when I wanted. I think this is the mark of a truly great technical staff. I would also like to thank

the staff of the University of Chicago Library, especially those in Eckhart Library: I am likely one of their most frequent patrons.

Finally, I thank my family and extended family. My extended family, who were also my roommates at various points throughout my graduate career: Spencer Dowdall, Ian Shipman, Ann Herbert, Rebecca Lordan, and (honorary roommate) Thomas Church; it's not so much that they made graduate school worth the time and effort, but that they made it worthwhile at least ten times over. I especially thank my grandparents Samuel and Frances Grochow, and Marvin and Hazel Barnett, my parents Jerrold and Louise Grochow, and my sister, Rebecca Grochow, for all their love and support in so many ways over the years. Last but by no means least, I thank my fiancé Nikki Pfarr. I thank her for her patience, support, and partnership; for her humor; for her love, romance, and companionship; for her wisdom, wit, humor, and intelligence; and for her smile.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
INTRODUCTION

This thesis is about symmetry, equivalence relations, and computational complexity. In this introduction we discuss what these terms mean and how they relate to one another. In the remainder of the thesis we study several relations among these topics in more depth: how symmetries of and equivalence relations on algorithms and algorithmic problems may improve our understanding of computational complexity, via Geometric Complexity Theory (Chapter 3); the computational complexity of a particular equivalence relation that arises in Geometric Complexity Theory (MATRIX ISOMORPHISM OF MATRIX LIE ALGEBRAS, Chapter 4); and finally, how computational complexity sheds light on algorithmic problems associated with equivalence relations in general (Chapter 5). In the concluding Chapter 6 we discuss other relationships between symmetry and computational complexity, and speculate on the future role of representation theory—the use of linear algebra to understand symmetry—in computational complexity.

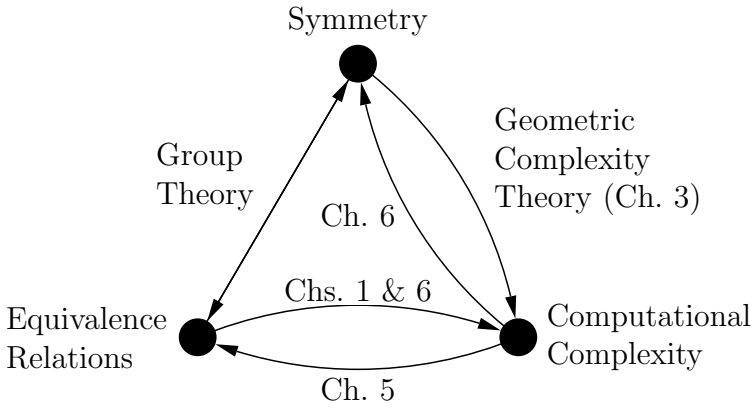


Figure 1.1: Some relationships between symmetry, equivalence relations, and computational complexity

In the remainder of this introduction, we define computational complexity, symmetry, and equivalence relations, and in the final Section 1.4 we introduce the main theme of this thesis:

how symmetry and equivalence relations may help shed light on computational complexity and vice versa.

1.1 Computational complexity

1.1.1 Computational problems and complexity measures

Computational complexity is the study of the difficulty, or *complexity*, of computing various functions or relations, often referred to generally as "problems." Definitionally, a *relational problem* may have more than one answer for each instance, while a *function (problem)* has at most one answer for each instance. For example, given a road map and two cities, finding any route between the two cities is a relational problem—there may be many such routes; finding the length of the shortest route between two cities is a function problem. In both these cases, a road map with two marked cities is an *instance* of the problem.

There are many measures of the computational complexity of a problem. Two of the most prominent complexity measures are the *time* needed to solve a problem, and the memory or *space* needed to solve a problem. "Time" is measured by the number of steps taken by an idealized computer called a Turing machine [266]. The *time complexity* of a problem is the least amount of time needed by *any* algorithm that solves the problem on this idealized computer. Of course, the actual time to solve a problem depends on the algorithm used and the computer it is run on. Measuring time on a Turing machine allows us to eliminate the issue of picking any particular hardware or software. Throughout the rest of this section we will focus on time complexity for concreteness, but what we say will also be valid for essentially any reasonable complexity measure.

For a given problem, there are typically many instances, even infinitely many, so how do we assign a quantitative measure to the time taken to solve a problem P? In the example above of road maps, we would expect any algorithm, even the fastest possible algorithm, to take more time solving the problem on a road map with 1000 cities than on a road map with 10 cities. Thus the time taken by an algorithm will be some function of the input size; in any given problem there is usually a natural notion of the *size* of an input, but generally speaking the size of an input should reflect the number of bits, or the amount of information, needed to encode the input in some reasonable fashion. In this thesis we will only be concerned with

so-called *worst-case* complexity, meaning that we consider the *maximum* amount of time taken by A over all instances of a given size.

But now we run into a problem: what do we mean by the *least* amount of time needed? By hard-coding the answers for certain instances into an algorithm, we can make the time to solve those instances essentially as small as we like. Instead, we measure the time complexity of P by the minimum *growth rate* of the time taken by any algorithm solving P; for example, for inputs of size n, as n goes to infinity, does the amount of time taken by the best algorithm for P scale like n? n^2? n^3? 2^n? This better captures the time taken by general strategies for solving P, independent of hard-coding the answers to certain instances.

Even this definition has issues. First, in reality few problems actually have infinitely many instances, as there are only finitely many atoms in the universe. Second, there are problems for which there is no optimal algorithm [59], that is, the minimum growth rate does not exist: there is some problem P and infinitely many algorithms solving P such that each algorithm takes time whose asymptotic growth rate is less than that of the previous algorithm in the list. However, these problems seem to rarely cause trouble in practice, and the asymptotic growth rate of the time taken by algorithms has proved to be a robust and useful measure of the complexity of a problem, both in theory and in practice.

1.1.2 Degrees of complexity

In this thesis, we will mainly be concerned with the theory of *polynomial-time* complexity. An algorithm A is said to run in *polynomial time* if there are constants c_1, c_2, c_3, independent of n, so that for all n, the time taken by A on any instance of size n is at most $c_1 n^{c_2} + c_3$. Problems that can be solved in polynomial time are often said to be "efficiently solvable." Despite the fact that an algorithm which runs in time n^{100} is hardly efficient—even on inputs of size 10 the number of steps needed by such an algorithm is more than the number of atoms in the universe—history has shown that the discovery of a polynomial-time algorithm for a problem often leads to the discovery of a truly efficient algorithm, in the real-world, practical sense.

One of the original motivations for the definition of polynomial time was to formally show that an algorithm is better than brute force [100, 218]. If the possible solutions to instances of size n consist of n-bit strings, then a naive brute-force strategy might consider

all 2^n strings of n bits and thus take exponential time. The question of whether all problems that have brute force solutions with short answers can be solved in polynomial time is the famous "**P** versus **NP**" question [88, 265, 209, 124], for which there is a million-dollar prize [80].

A useful technique in complexity theory is to relate problems to one another by saying that P_1 is at most as hard as P_2. This enables us to make statements like "problems P_1 and P_2 have the same complexity," without having to know what that complexity actually is. If in the future the complexity of P_2 is determined, then that of P_1 would be automatically determined as well.

Informally, we say that "P_1 reduces to P_2" if any algorithm for P_2 yields an algorithm of similar (polynomially related) complexity for P_1. Such a reduction tells us that P_1 is at most as hard as P_2. If the reverse holds as well—that is, if P_2 also reduces to P_1—then we say that P_1 and P_2 have the same *degree of complexity*. In order to formally capture the idea that an algorithm for P_2 yields an algorithm for P_1, Turing introduced the notion of "oracle machines." An algorithm with an "oracle for P_2" is an algorithm A that calls an algorithm for P_2 as a black-box subroutine: that is, A may write down instances of P_2 and then expect to receive answers back from this subroutine. Aside from the fact that the subroutine solves P_2, its exact nature is unimportant; in some sense it doesn't even matter if P_2 is solvable at all, in which case A is treating the subroutine as an "oracle" that solves P_2. Note, however, that this oracle may be replaced by any algorithm for P_2, and then the oracle algorithm would turn into a complete, down-to-earth, oracle-free algorithm.

The task of determining the exact complexity of any given problem has turned out to be *incredibly* difficult, and for most problems this question has resisted 40 years of intense research (for example, see the survey by Fortnow [108]). But in those four decades, thousands of reductions *between* problems have been discovered. Through these reductions, complexity theorists have grouped myriad problems, some of theoretical interest but most coming from practical needs, together into a very few degrees of complexity. Algorithmic problems, algorithms, and degrees of complexity are the principal objects of study in computational complexity theory.

1.2 Equivalence relations

Finding a good notion of equivalence can be an important step in figuring out how to show that two mathematical objects, such as complexity classes, are distinct from one another. This idea is borne out in the histories of almost every branch of math: algebra, analysis, topology, geometry, combinatorics, etc. At a more basic level, complexity degrees themselves are examples of equivalence relations, and other equivalence relations arise in complexity theory in fundamental ways, discussed throughout this thesis.

The most natural equivalences arise because we are forced to use symbols to write something down, but those symbols are not essential to the thing itself. For example, whether we write integers in base 2 or base 10, we are still dealing with "the same" numbers. What we mean by an integer is an abstract notion, independent of the way it is written down, that depends only on its relationship with other integers.

The next example, graph isomorphism, is used throughout this introduction and appears prominently in Chapter 4. A *graph* consists of a set V of vertices and a set E of edges, where each edge consists of a pair of vertices. In the following diagrams, dots represent vertices and lines represent edges; intersections are artifacts of the drawing. Each of the two graphs in Figure 1.2 have 10 vertices and 15 edges, but are they "the same?"

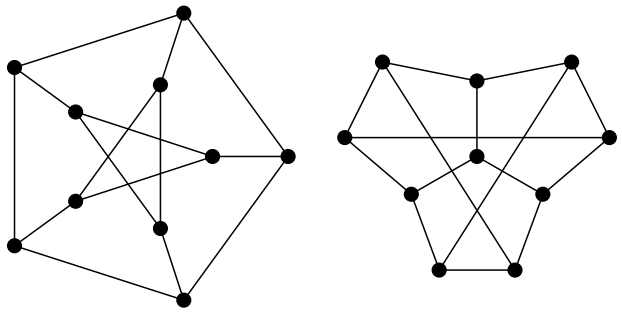


Figure 1.2: Are these two graphs "the same?"

The drawings as you see them do not look the same. Nevertheless, the two graphs are "the same" in that the manner in which vertices are related to one another by edges in the left graph is the same as the manner in which vertices are related to one another by edges in the right graph; if we choose a particular association between the vertices in the left graph with the vertices in the right graph then this can be easily checked. For example, if we label the vertices by the letters $A, \ldots, J$ as in Figure 1.3, then the set of edges of each graph becomes

the same set of 15 unordered pairs $\{\{A,B\},\{A,E\},\{A,F\},\{B,G\},\{B,C\},\{C,D\},\{C,H\},\{D,E\},\{D,I\},\{E,J\},\{F,H\},\{F,I\},\{G,I\},\{G,J\},\{H,J\}\}$.

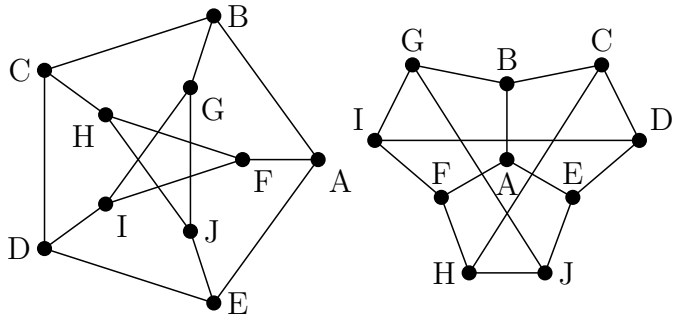


Figure 1.3: A labeling of the vertices

This does *not* follow automatically from the fact that the two graphs have the same number of vertices and edges, as Figure 1.4 shows.

Figure 1.4: Two graphs with the same number of vertices and edges that are *not* the same.

Had we been given two graphs that were not the same, how we would have known? We could start to look for other ways in which graphs might be equivalent, and rule out two graphs being the same by showing that they are not equivalent according to one of these notions. For example, the number of vertices, number of edges, number of paths of a given length, number of vertices with a given number of adjacent edges, etc. are all criteria which might be used to show that two graphs are not the same.

In addition to distinguishing mathematical objects, equivalence relations have historically been used to define fields of mathematical inquiry. It is often the case that mathematicians begin with some intuitive idea of their subject matter, say, algorithms, or geometry, and only later formalize this into a definition of what a geometry "is." This process of defining is often closely tied with an equivalence relation, which defines when two geometries are "the same" and discards other possible features of geometries that may have been relevant but ultimately were deemed irrelevant. In particular, artifacts of the symbols used to write something down are often discarded by such equivalence relations. This is actually a very

revisionist view of the history of mathematics: very rarely are such first definitions phrased in terms of equivalence relations. However, the notion of equivalence relation *formalizes* and *reifies* this process of finding the right definitions, and by studying equivalence relations as (meta-)mathematical objects in their own right we might gain insight into what kinds of equivalence relations are useful for various mathematical pursuits.

A philosophical premise which might be said to underlie Geometric Complexity Theory is that the most useful equivalence relations for giving good definitions in complexity theory are those that in some sense have low complexity (discussed in Chapter 5), and those that are based on symmetry, which we discuss next.

1.3 Symmetry

Equivalence relations—hence mathematical definitions, as in the previous section—based on symmetry are in some sense "better" than others. Such symmetry-based equivalence relations often provide more tools to distinguish mathematical objects, particularly through the theory of symmetry itself, *group theory*. In this section we define what we mean by symmetry and explain how symmetries can give rise to equivalence relations. In the next section we begin to see in what sense these symmetry-based equivalence relations are "better," and what this might tell us about complexity.

In this regard, we highly recommend the book *Symmetry* by Hermann Weyl [275], both for novices and experts: it is wonderfully written and takes the reader on a path from symmetry in art and nature to its formalization in group theory. Here we will take a more direct route, which although less scenic, has the virtue of being only a few pages instead of a few dozen.

In what sense is a circle "more symmetric" than an equilateral triangle? Or an equilateral triangle more symmetric than an isosceles triangle (two sides equal), or an isosceles triangle more symmetric than a general triangle (see Figure 1.5)?

We say that *a symmetry* of an object is a *transformation* under which, at the end of the transformation, the object *appears* exactly as it did before the transformation. For example, rather than saying that an isosceles triangle has left-right symmetry, we say that it is symmetric under the reflection through its vertical axis. If a transformation is a symmetry of an object, we say that the object is *invariant* under the transformation.

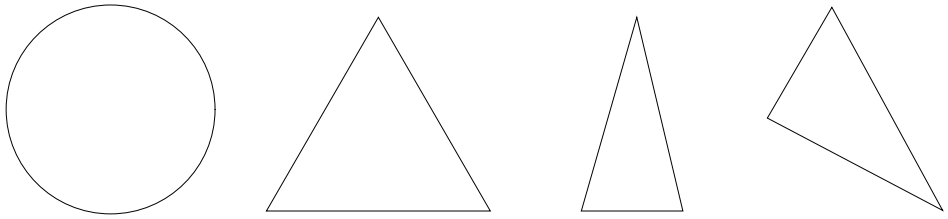


Figure 1.5: Some shapes with varying degrees of symmetry: a circle, an equilateral triangle, an isosceles triangle, a general triangle.

A general triangle has no symmetries. Actually, it will be convenient to regard the null or identity transformation ("do nothing") as a symmetry of any object. When we say an object has no symmetries, we mean it has no symmetries other than the identity transformation.

An equilateral triangle has six symmetries: the identity, rotation by 120 or 240 degrees, and reflection through any of its three axes. Had we forgotten the rotation by 240 degrees, we could have deduced its existence: rotation by 240 degrees is the consequence of rotating by 120 degrees twice. If we perform one transformation followed by another, we get a third transformation, whose result is simply the result of applying the two transformations in succession. If two transformations are both symmetries of an object, then the transformation we get by applying them in succession is again a symmetry. This generalizes the idea of applying the rotation by 120 degrees *twice*. For an equilateral triangle, if we had only discovered the rotation by 120 degrees and a single reflection through an axis, we could have deduced the rest of the triangle's symmetries by this method of composing transformations.

An isosceles triangle has the symmetry given by reflection through its axis, as well as the identity symmetry. Note that composing the reflection with itself results in the identity transformation.

Moreover, if a transformation is a symmetry of some object, then the transformation's *inverse*—undoing the transformation, or doing the transformation in reverse—is also a symmetry of that object. Any collection of transformations that is closed under composition and under taking inverses is called a *group* (of transformations, or symmetries). Group theory is the formal study of symmetry.

1.3.1 Continuous symmetries and Lie algebras

The circle has infinitely many symmetries: it is invariant under all rotations about its center, as well as under any reflection through any line that passes through the circle's center. Not only does the circle have infinitely many symmetries, but "continuously many." We say its symmetries form a "continuous group." What we mean by this should be intuitive, but to help clarify we give a non-example.

Consider the geometric figure which consists of infinitely many points, one placed at each integer (Figure 1.6). This figure has infinitely many symmetries: shift left or right by n, for

Figure 1.6: A geometric figure with an infinite but discrete group of symmetries.

any integer n, reflect 180 degrees around any of the points of the figure, or reflect 180 degrees around any point that is halfway between two points of the figure. Despite having infinitely many symmetries, the symmetries of this figure form a discrete collection (group), rather than a continuous one as in the case of the circle.

Continuous groups of symmetries are called Lie groups (or more generally topological groups), after their inventor Sophus Lie; Lie algebras, the main subject of Chapter 4, are a key tool in the study of Lie groups. A Lie algebra is an "infinitesimal approximation" of a Lie group in the same way that the terms of a Taylor series are approximations of a function. In fact, in exactly the same way: the Lie algebra consists of the first-order approximations of the transformations in a Lie group, where we think of each transformation as a function to be approximated by a Taylor series. The infinitesimal approximation afforded by Lie algebras allows the use of plain linear algebra to understand these continuous groups of symmetries. Both continuous and finite groups of symmetries appear throughout this thesis.

1.3.2 Symmetry-based equivalence relations

Symmetries naturally lead to equivalence relations. For example, in the isosceles triangle, the two corners of its base are equivalent, but they are not equivalent to the corner at the

top of the triangle. In a general triangle, all three corners are not equivalent to one another; in an equilateral triangle, all three corners are equivalent to one another. However, a corner is not equivalent to a point on the side. Moreover, most (but not all!) points on the sides of an equilateral triangle are not equivalent to one another.

Given a group G of symmetries acting on some set—in the above examples, the set in question is the set of points of the figure—two points of the set are *(G-)equivalent* if one of them can be taken to the other by some transformation in the group G. In general, G need not be the group of *all* symmetries of the set; it may be a subgroup, but it must still be closed under composition and inversion.

In the equilateral triangle, we have already mentioned that the three corners form one equivalence class under the group of symmetries of the equilateral triangle. Figure 1.7 shows two other equivalence classes of points.

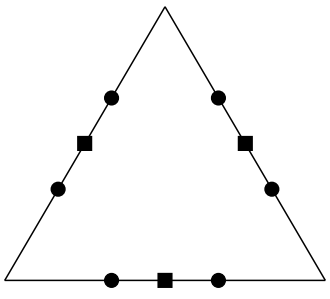


Figure 1.7: Under the full symmetry group of an equilateral triangle, the points marked by circles are all equivalent to one another. The midpoints of the sides, marked by squares, are all equivalent to one another, but are not equivalent to the points marked by circles.

In a similar manner, any group of symmetries leads to an equivalence relation. In Section 3.3.1 we discuss further how the very notion of group was in some sense designed to lead to equivalence relations; see especially Proposition 3.3.1 and the subsequent discussion.

There are also quite general situations in which the reverse connection holds: any equivalence relation with certain natural properties arises from *some* group in the above manner. We discuss one of these, the Feldman–Moore Theorem, in Footnote 1 on page 13.

Symmetries apply to more than just geometrical figures. For example, we would naturally say that the expression $x^2 + y^2 + z^2$ is symmetric in x, y, and z. Sticking with the formalism of transformation groups, we would say that this expression is invariant under any transformation that permutes the variables x, y, and z. Similarly, $x^2 + y^2 + z^3$ is symmetric in x and y, but not in z. That is, x and y are equivalent in this expression, but they are not equivalent to z.

The symmetries of the plane consist of its rigid motions: translations, rotations, and reflections. They form a continuous group of symmetries. Felix Klein, in his famous Erlangen Program, first introduced the idea that a *geometry*, such as the Euclidean geometry of the plane, or more general geometries including non-Euclidean geometries, is fully determined by its group of symmetries. In other words, geometric statements about the plane are exactly those statements that are invariant under the symmetry group of the plane. As a starting point, note that the distance between two points is unchanged if they are both simultaneously translated, reflected about an axis, or rotated about any third point. By specifying a symmetry group, one specifies which geometry one is interested in. This is the symmetry-based version of the idea from the previous section that an equivalence relation specifies the aspects of a mathematical object that define a field of inquiry.

Now we return to the more interesting example of graph isomorphism. Recall that graph isomorphism is an equivalence relation in which two graphs are equivalent if the vertices of one can be matched with the vertices of the other to make the graphs identical. In this case, the underlying set of the equivalence relation is the set of all graphs on n vertices, which we denote $\mathcal{G}_n$. For definiteness, let us label these vertices by the numbers $1, \ldots, n$. Any permutation of the numbers $1, \ldots, n$ induces a transformation on the set of all graphs on n vertices; the set of all such permutations is the symmetry group of $\mathcal{G}_n$. The equivalence classes under this symmetry group are exactly the isomorphism classes of graphs.

Note that the symmetries of a single graph form a subgroup of the symmetries of $\mathcal{G}_n$. For example, the group of symmetries of $\mathcal{G}_4$ is the group of all permutations of $\{1, \ldots, 4\}$. There are $4! = 24$ such permutations. The group of symmetries of the square (see Figure 1.4) form a subgroup consisting of only 8 transformations. Chester [76] gives a nice discussion of this phenomenon in the world of physics: the symmetries of $\mathcal{G}_n$ are analogous to the symmetries of a physical law—such as conservation of angular momentum—while the symmetries of

the square are analogous to the symmetries of a given physical system—such as a spinning asteroid—which may have fewer symmetries than the fundamental physical law.

1.4 Symmetry and equivalence relations in complexity

When an equivalence relation arises from a group of symmetries, as with graph isomorphism, geometry, and the other examples in the previous section, then the tools of group theory may be used in its study. In this manner, groups have risen to a central place in mathematics, second only perhaps to numbers and sets. More than that, when an equivalence relation arises from a group, we often have a better grasp of its meaning. For example, equivalence relations arising from groups are so central in physics that Chester titled his paper "Is symmetry identity?" [76]; he suggests that groups of symmetries *are* notions of identity or equivalence. In the end he recognizes there are other equivalence relations and hence other notions of identity, but the thrust of the paper is that the equivalence relations that matter are those coming from groups. All of this suggests that an equivalence relation without an underlying group often yields a somewhat unsatisfying notion of sameness. In my view, this is certainly the case in computational complexity, though Geometric Complexity Theory offers an intriguing possibility for more satisfying equivalence relations in complexity theory.

In computational complexity, equivalence relations arise at two very different levels: at the level of computational problems to be solved, such as the graph isomorphism problem—given two graphs, decide whether they are isomorphic—and at the meta level of describing computational problems and algorithms, as in the notion of degree of complexity. In the former, group theory still often plays a central role. For example, the best known algorithms to solve the graph isomorphism problem heavily use group theory [35, 33]. Also, the ability to efficiently compute the determinant of a matrix is closely related to the group of symmetries of the determinant (see Proposition 3.4.3). In Chapter 5 we study the complexity of deciding equivalence relations in general: that is, for any fixed equivalence relation $\sim$, what is the complexity of the problem "given x and y, decide whether $x \sim y$." Since quantum mechanics is so intimately related to symmetry, the connection between equivalence relations and symmetry allows us to show a new connection between quantum computing and the (non-quantum) computational complexity of equivalence relations.

However, at the meta level in computational complexity, the primary equivalence relation of interest is that of degree of complexity. It is not clear how we might employ group theory in the study of degrees of complexity[1]. We still do not have a really good notion of what it means for two computational problems or two algorithms to be equivalent, in the sense that we have yet to find a notion, symmetry-based or otherwise, that allows us to distinguish complexity classes from one another.

Algebraic complexity offers the possibility of a more satisfying notion of equivalence naturally arising from a group of symmetries for computational problems and algorithms. The Geometric Complexity Theory Program [207] (see also Chapter 3), in turn, suggests a method of exploiting group theory to resolve the fundamental questions of computational complexity that have eluded the community for more than 40 years, such as **P** versus **NP**. Geometric Complexity Theory also offers a way to extend these techniques from algebraic complexity to traditional Turing-machine-based computational complexity (see Section 3.3.3).

In algebraic complexity, the primary concern is not the number of steps of a Turing machine, but the number of arithmetic operations—addition, multiplication, subtraction, and division—needed to compute a function. If $f(x_1, \ldots, x_n)$ is a function, and A is an $n \times n$ matrix, then we define the function $A \cdot f$ by first applying A to the inputs, and then applying f:

$$(A \cdot f)(x_1, \ldots, x_n) := f((x_1, \ldots, x_n)A),$$

where we treat $(x_1, \ldots, x_n)$ as a row vector. The algebraic complexity of $A \cdot f$ is at most that of f plus n^2, the number of operations needed to compute the vector-matrix multiplication $(x_1, \ldots, x_n)A$. If A is invertible, then the reverse also holds, since we may use A^{-1} in place of A. In particular, if the complexity of f is greater than n^2—or if we are wondering whether f can be computed in polynomially many arithmetic operations at all—then it is equivalent to study f or $A \cdot f$, when A is invertible.

1. It is known, from the very general Feldman–Moore Theorem [103], that the equivalence relation "has the same complexity degree," which we'll write as $P_1 \equiv P_2$, does in fact arise from a group of symmetries. However, the Feldman–Moore Theorem merely shows the *existence* of such a group; there are infinitely many possibilities for the group, and the Feldman–Moore Theorem constructs one. The groups given by the Feldman–Moore Theorem are non-canonical, in that they are generally not related to the underlying equivalence relation in a natural way; it is not clear how to pick a group that is naturally associated to $\equiv$. Thus, despite the Feldman–Moore Theorem and the existence of a group yielding $\equiv$, it is still unclear how really to use group theory to study $\equiv$.

For example, consider the function $f(x, y) = x^2 + y^2$. f is equivalent to the function $(A \cdot f)(x, y) = (\alpha x + \beta y)^2 + (\gamma x + \delta y)^2$, whenever the matrix $A = \begin{pmatrix} \alpha & \beta \\ \gamma & \delta \end{pmatrix}$ is invertible. The function $x^2 + y^2$ can be computed in three arithmetic operations—one addition and two multiplications—and $(A \cdot f)(x, y)$ can be computed in nine operations (though perhaps one can do better!).

Furthermore, these transformations of the variables give us a notion of equivalent *algorithms*, as follows. Consider the following program for computing $x^2 + y^2$:

1. Let $a_1 := x$.
2. Let $a_2 := y$.
3. Compute $a_3 := a_1^2$.
4. Compute $a_4 := a_2^2$.
5. Compute and output $a_3 + a_4$.

We may get an equivalent program by first rotating the vector (x, y) by any angle θ: since $x^2 + y^2$ is the square of the length of the vector (x, y), and this length is invariant under the rotation $\begin{pmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{pmatrix}$, when A is of this form, we have not only that $A \cdot f$ and f are equivalent, but that they are *equal*. Thus the following program, which obviously computes $A \cdot f$, in fact computes f, and we may consider it equivalent to the previous program under the rotation A:

1. Compute $a_1 := (\cos(\theta)x + \sin(\theta)y)$.
2. Compute $a_2 := (-\sin(\theta)x + \cos(\theta)y)$.
3. Compute $a_3 := a_1^2$.
4. Compute $a_4 := a_2^2$.
5. Compute and output $a_3 + a_4$.

Here we used the fact that the symmetry group of $f(x, y) = x^2 + y^2$ contains all rotations around the origin; it also contains all reflections about lines through the origin. That this is the same as the symmetry group of the circle should perhaps not be a surprise, since the unit circle is given by the equation $x^2 + y^2 = 1$. In particular, this is a continuous Lie group. In Chapter 4 we use the Lie group of symmetries of a function to study the complexity of

the question of when two given functions are equivalent. This leads us to a natural question on Lie algebras, whose complexity we ultimately relate to that of the graph isomorphism problem. We also show that certain cases of this problem on Lie algebras can be solved in polynomial time. Thus, in the algebraic setting the meta-relation of equivalence between problems is closely related to a concrete computational problem on Lie algebras.

The above notion of equivalence is very similar to complexity degrees, but is based on the group of all invertible $n \times n$ matrices (this is indeed a group: by definition every such matrix has an inverse, and composing the linear transformations of matrices is the same as matrix multiplication). This gives a symmetry-based notion of "equivalent complexity" for computational problems and algorithms in the algebraic setting. There are certain natural problems, such as the multiplication of 2×2 matrices [99] or the multiplication of two polynomials modulo a third [19], for which the optimal algorithm is *unique*, in the sense that any two optimal algorithms are equivalent under this group. We are unaware of any such statement for *any* problem in the Turing machine model. This phenomenon is just one benefit of the prominence of symmetries in algebraic complexity. The symmetry-based nature of equivalence relations on problems and algorithms in algebraic complexity, and their use in Geometric Complexity Theory, suggests that they may enable further understanding of complexity in general.

1.5 Organization

Chapter 2 introduces needed formalisms and background material. In Chapter 3 we give a tutorial and survey of the Geometric Complexity Theory program, including some new observations of our own. In Chapter 4 we discuss and present the matrix isomorphism problem for matrix Lie algebras and present our results on that problem, as well as its application to the affine equivalence problem for polynomials and the isomorphism problem for abstract Lie algebras. In Chapter 5 we discuss the algorithmic question of solving equivalence relations more generally, and show that various approaches to solving equivalence relations are likely to be of different computational powers. We also relate the complexity of equivalence relations to probabilistic and quantum computation. In Chapter 6 we conclude with some open questions and remarks for future work. Each chapter has its own introduction detailing its contents and organization.

CHAPTER 2
BACKGROUND

This section serves to introduce standard concepts, and fix notation and conventions. We recommend that the reader proceed directly to the chapter they are interested in, and refer to this chapter only as needed.

2.1 Complexity Theory

We assume the reader is familiar with standard (uniform) models of computation as in the books by Sipser [247] or Arora and Barak [14]. We use the multi-tape Turing machine with read-only input tape and write-only output tape as our standard model of computation, and make no further mention of the model except where it is relevant. Oracle Turing machines have a separate oracle tape and oracle query state. When the machine enters the query state, it transitions to one of two specified states depending on whether the string on the oracle tape is in the oracle. An oracle Turing machine with unspecified oracle is denoted $M^{\square}$ for emphasis.

Alphabet and strings Throughout, Σ denotes a finite set, called the *alphabet*, and is usually taken to be $\{0,1\}$. We often use the term "bit" rather than the more general "symbol" because of this convention. The set of strings of length exactly k over Σ is denoted Σ^k. The empty string is denoted ε. The notation $\Sigma^{\leq k}$ is used to denote $\bigcup_{n=0}^{k} \Sigma^n$, and Σ^* is used to denote the set of all finite strings. The length of a string is denoted by absolute value: thus $|x| = k$ if and only if $x \in \Sigma^k$.

Lexicographic order. When Σ is an initial segment of the natural numbers, it is equipped with the usual ordering, but even otherwise we may think of Σ as having an ordering $<_\Sigma$. The lexicographic ordering on Σ^* is given by $x <_{lex} y$ if $|x| < |y|$ or $|x| = |y|$, and if j is the leftmost position at which x and y differ, then $x_j <_\Sigma y_j$, where x_j denotes the j-th bit of x.

There is a bijective correspondence between Σ^* and $\mathbb{N}$, given by the lexicographic ordering on Σ^*, and we use this correspondence freely, referring to elements of Σ^* as "numbers" and speaking of the "length of the number n." Note that the length of the number n is $\lceil \log_{|\Sigma|}(n) \rceil$. We use log to denote $\log_2$.

Tuples. Ordered tuples are written with parentheses, such as $(u_0, \ldots, u_k)$. When needed, an ordered tuple is encoded into a single string by the iterated application of an easily computable and easily invertible bijective pairing function $\langle \cdot, \cdot \rangle \colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ such as $\langle x, y \rangle = \frac{1}{2}(x+y)(x+y+1)+y$. The iteration is performed as follows: $\langle u_0, \ldots, u_k \rangle = \langle u_0, \langle u_1, \ldots, u_k \rangle \rangle$.

2.1.1 Computational problems

A subset $L \subseteq \Sigma^*$ is called a *language*. The complement of L is denoted $\overline{L} = \Sigma^* \backslash L$. The *decision problem* for a language L is: given $x \in \Sigma^*$, decide whether or not $x \in L$. Many computational problems can be stated as decision problems, or are computationally equivalent to decision problems.

However, some problems are more naturally stated as *search problems*. A search problem is: given $x \in \Sigma^*$, find some y such that (x, y) satisfies some condition. For example, given an (encoding of) a graph G, find a Hamiltonian path in G if one exists. A solution to a search problem is a function f such that $(x, f(x))$ satisfies the desired condition, or $f(x) = \perp$ if there is no string y such that (x, y) satisfies the desired condition. Hence the computational complexity of search problems is closely related to the computational complexity of functions.

The *indicator function* of a language L is the function

$$L(x) = \begin{cases} 1 & \text{if } x \in L \\ 0 & \text{if } x \notin L \end{cases}$$

It is standard to abuse notation and use the same letter for both the language and its indicator function. Algorithmically solving the decision problem L is the same as computing the function L.

2.1.2 Reductions

A *Turing reduction* from language A to language B is an oracle Turing machine $M^{\square}$ such that $A(x) = M^B(x)$ for all $x \in \Sigma^*$. We write $M\colon A \leq_T B$. (The function-like notation "$M\colon A \leq_T B$" is not standard, but is a natural combination of standard function notation "$f\colon X \to Y$" and the standard reduction notation "$A \leq_T B$.")

A *many-one reduction* or m-reduction from A to B is a (computable) function $f\colon \Sigma^* \to \Sigma^*$ such that $x \in A \iff f(x) \in B$. We write $f\colon A \leq_m B$.

For any notion of reduction r, $A \equiv_r B$ denotes that $A \leq_r B$ and $B \leq_r A$. If $\mathcal{C}$ is a class of machines, then $\leq_r^{\mathcal{C}}$ denotes that the reducing machine lies in $\mathcal{C}$. Many complexity classes are naturally associated with a class of machines, and when this is the case we use the name of the complexity class. For example, although **P** is a collection of languages, in reductions we use **P** to denote the class of deterministic polynomial-time Turing machines. In particular, the polynomial-time-bounded versions of the above reductions are denoted $\leq_T^P$ and $\leq_m^P$, respectively.

Polynomial-time Turing reductions are known as *Cook reductions* and polynomial-time many-one reductions are known as *Karp reductions*, since these were the types of reductions originally used by their respective namesakes to define **NP**-completeness [88, 156].

A class (collection) of languages $\mathcal{C}$ is said to be *closed under r reductions* if $B \in \mathcal{C}$ and $A \leq_r B$ implies $A \in \mathcal{C}$.

2.1.3 Complexity classes

Polynomial time. The class of languages decidable in deterministic polynomial time is denoted **P**.

The class of languages decidable in *nondeterministic* polynomial time is denoted **NP**. Equivalently, $A \in \mathbf{NP}$ if there is a set $B \in \mathbf{P}$ such that

$$x \in A \iff (\exists^p w)[(x, w) \in B]$$

where the right hand side is taken to mean "there exists a polynomial q such that $|w| \leq q(|x|)$ and $(x, w) \in B$." Such a string w is said to *witness* that $x \in A$, and is called a witness for x.

If $\mathcal{C}$ is a class of languages, then $\mathbf{co}\mathcal{C} = \{L : \overline{L} \in \mathcal{C}\}$. For example, $A \in \mathbf{coNP}$ if and only if there is a set $B \in \mathbf{P}$ such that

$$x \in A \iff (\forall^p w)[(x, w) \in B]$$

where $\forall^p$ has the obvious meaning. Note that we can use $(x, w) \in B$ or $(x, w) \notin B$ in the above characterization, since $\mathbf{P}$ is closed under complementation, i. e., $\mathbf{P} = \mathbf{coP}$.

The following basic questions (and many more) are open: $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$, $\mathbf{NP} \stackrel{?}{=} \mathbf{coNP}$, $\mathbf{P} \stackrel{?}{=} \mathbf{NP} \cap \mathbf{coNP}$.

Hardness and completeness. If $\mathcal{C}$ is a class of languages and r is a notion of reduction, a language L is said to be *hard for $\mathcal{C}$ under r reductions* if $X \leq_r L$ for every $X \in \mathcal{C}$. If, furthermore $L \in \mathcal{C}$, then L is said to be *r-complete for $\mathcal{C}$*.

In many cases, a standard notion of reduction is used. For example, a language L is said to be **NP**-hard if it is hard for **NP** under Karp ($\leq_m^P$) reductions.

Logarithmic space. The class of languages decidable in deterministic logarithmic space is denoted **L**. Here the input is read-only and the space of the input is not counted towards the space bound. The class of languages decidable in nondeterministic logarithmic space is denoted **NL**. Unlike the situation for **NP**, it is known that $\mathbf{NL} = \mathbf{coNL}$ [144, 258].

Polynomial space. The class of languages decidable in polynomial space is denoted **PSPACE**. The nondeterministic analogue, **NPSPACE** is often mentioned only up to the point of Savitch's Theorem, which implies $\mathbf{PSPACE} = \mathbf{NPSPACE}$ [231]. We make no further mention of **NPSPACE**.

Relativizing complexity classes. For a language A, and a class $\mathcal{M}$ of oracle Turing machines, we can define the relativized class $\mathcal{M}^A$ as the class of languages that are Turing-reducible to A by some machine in $\mathcal{M}$. For a class of machines $\mathcal{M}$ and a class of languages $\mathcal{C}$, we define $\mathcal{M}^{\mathcal{C}} = \bigcup_{L \in \mathcal{C}} \mathcal{M}^L$.

It is standard to abuse this terminology and use classes of languages instead of classes of machines for the base of the oracle, but the meaning is as expected. For example, $\mathbf{P}^A$ is the set of all languages that are polynomial-time Turing-reducible to A.

The polynomial hierarchy. Relativizing to a language L is essentially the same as relativizing to its complement $\overline{L}$. Hence, for example $\mathbf{P}^{\mathbf{NP}}$ contains both **NP** and **coNP**. Based on this observation, we may define the *polynomial hierarchy*, originally introduced by

Stockmeyer and Meyer [255, 254] in analogy with the arithmetic hierarchy from computability theory:

$$\begin{aligned} \mathbf{\Sigma_0 P} &= \mathbf{P} \\ \mathbf{\Sigma_1 P} &= \mathbf{NP} \\ \mathbf{\Sigma_{k+1} P} &= \mathbf{NP}^{\mathbf{\Sigma_k P}} \\ \mathbf{\Delta_{k+1} P} &= \mathbf{P}^{\mathbf{\Sigma_k P}}. \end{aligned}$$

From these, we define $\mathbf{\Pi_k P} = \mathbf{co\Sigma_k P}$; for example, $\mathbf{\Pi_1 P} = \mathbf{coNP}$. Thus $\mathbf{\Sigma_0 P} = \mathbf{\Pi_0 P} = \mathbf{\Delta_0 P} = \mathbf{\Delta_1 P} = \mathbf{P}$. Note that $\mathbf{\Sigma_{k+1} P} = \mathbf{\Sigma_k P}^{\mathbf{NP}}$.

It is clear that $\mathbf{\Sigma_k P} \cup \mathbf{\Pi_k P} \subseteq \mathbf{\Delta_{k+1} P} \subseteq \mathbf{\Sigma_{k+1} P} \cap \mathbf{\Pi_{k+1} P}$. The polynomial hierarchy is the union $\mathbf{PH} = \bigcup_{k=0}^{\infty} \mathbf{\Sigma_k P} = \bigcup_{k=0}^{\infty} \mathbf{\Pi_k P} = \bigcup_{k=0}^{\infty} \mathbf{\Delta_k P}$.

The following are equivalent: (1) $\mathbf{\Sigma_k P} = \mathbf{\Pi_k P}$, (2) $\mathbf{\Sigma_j P} = \mathbf{\Sigma_k P}$ for some $j \geq k$, and (3) $\mathbf{PH} = \mathbf{\Sigma_k P}$. If any (and hence all) of these conditions holds, we say the hierarchy *collapses to the k-th level*. If this does not hold for any level k, we say that **PH** is infinite. It is widely believed that **PH** is infinite.

Complexity class operators. We now define the operators $\forall \cdot$ and $\exists \cdot$ on complexity classes. If $\mathcal{C}$ is a complexity class, then $\forall \cdot \mathcal{C}$ consists of those languages L for which there is a language $L' \in \mathcal{C}$ such that

$$x \in L \iff (\forall^p y)[(x, y) \in L'].$$

The $\exists \cdot$ operator is defined similarly. It is clear from our definitions that $\mathbf{NP} = \exists \cdot \mathbf{P}$ and $\mathbf{coNP} = \forall \cdot \mathbf{P}$. Indeed, it holds generally that $\mathbf{co}\exists \cdot \mathcal{C} = \forall \cdot \mathbf{co}\mathcal{C}$.

It is a standard exercise to show that

$$\forall \cdot \mathbf{\Sigma_k P} = \mathbf{\Pi_{k+1} P} \text{ and } \exists \cdot \mathbf{\Pi_k P} = \mathbf{\Sigma_{k+1} P}.$$

Hence we may consider $\mathbf{\Sigma_k}$ as the operator $\exists \cdot \forall \cdot \cdots Q_k \cdot$ where there are k operators total and Q_k is $\forall$ or $\exists$ depending on whether k is even or odd, respectively. Similarly, we may consider $\mathbf{\Pi_k}$ to be the operator $\forall \cdot \exists \cdot \cdots Q'_k \cdot$.

Randomness. Several complexity classes have been defined to capture various notions of randomized computation. Bounded-error probabilistic polynomial time, denoted **BPP**, consists of those languages L for which there is a language $L' \in \mathbf{P}$ and a polynomial p such that, for all x of length n:

$$\Pr_{r \in \Sigma^{p(n)}}[L'(x,r) = L(x)] \geq 2/3$$

Here, $2/3$ can be replaced by any function of n that is bounded below by $1/2 + \varepsilon$ for some constant $\varepsilon > 0$. By running an algorithm for L' several times with independent random bits r and taking the majority vote, the probability of correctness can be increased to $1 - 2^{q(n)}$ for any polynomial q. Note that **BPP** allows *two-sided error*: L' can err on strings $x \in L$ and on strings $x \notin L$. **BPP** algorithms are sometimes referred to as *polynomial-time Monte Carlo* algorithms.

The classes **RP** and **coRP** are the one-sided error version of **BPP**. The class **RP** consists of those languages L for which there is a language $L' \in \mathbf{P}$ and a polynomial p such that

$$\begin{aligned} x \in L &\implies \Pr_{r \in \Sigma^{p(|x|)}}[L'(x,r) = 1] > 1/2 \\ x \notin L &\implies \Pr_{r \in \Sigma^{p(|x|)}}[L'(x,r) = 1] = 0 \end{aligned}$$

Probabilistic classes can also be defined in terms of nondeterministic Turing machines. A probabilistic Turing machine is a nondeterministic Turing machine where each binary nondeterministic choice, referred to as a "coin flip," is assigned a probability of $1/2$. The probability of any given branch of the computation is the product of the probabilities of the coin flips that occur on that branch. From this viewpoint, it is clear that $\mathbf{RP} \subseteq \mathbf{NP}$.

The class **ZPP**, or zero-error probabilistic polynomial time, consists of those languages for which there is a randomized algorithm that never errs, and runs in *expected* polynomial time, the expectation being taken over the random coin flips. It is an easy exercise to show that $\mathbf{ZPP} = \mathbf{RP} \cap \mathbf{coRP}$. **ZPP** algorithms are sometimes referred to as *polynomial-time Las Vegas* algorithms [21].

The relationship between **BPP** and **NP** is unknown. Today it is an easy exercise to show that if $\mathbf{NP} \subseteq \mathbf{BPP}$ then $\mathbf{NP} = \mathbf{RP}$, though this was originally proved by Ko [168]. Sipser [245], with help from Gács, and Lautemann [179] showed that $\mathbf{BPP} \subseteq \mathbf{\Sigma_2 P} \cap \mathbf{\Pi_2 P}$.

Similar to the $\forall\cdot$ and $\exists\cdot$ operators, we can define the $\mathbf{BP}\cdot$ operator. The class $\mathbf{BP}\cdot\mathcal{C}$ consists of those languages L for which there is a language $L' \in \mathcal{C}$ and a polynomial p such that

$$\Pr_{r\in\Sigma^{p(|x|)}}[L'(x,r) = L(x)] \geq 2/3.$$

It is clear that $\mathbf{BP}\cdot\mathbf{P} = \mathbf{BPP}$.

Mixing randomness and nondeterminisim. Arthur–Merlin games, and their corresponding complexity class **AM**, were introduced by Babai [22]. The basic idea is that the mere mortal King Arthur (with access to random coins) wishes the all-powerful wizard Merlin to prove a fact to him. For our purposes, it is simplest to define the class of Arthur-Merlin games as:

$$\mathbf{AM} = \mathbf{BP}\cdot\mathbf{NP}.$$

Babai showed [22] that for any fixed number of alternations greater than 2 of the operators **BP** and $\exists\cdot$, $\mathbf{BP}\cdot\exists\cdot\mathbf{BP}\cdot\exists\cdot\cdots\cdot\mathbf{P} = \mathbf{AM}$. Often such extensions are denoted, for example, $\mathbf{MAM} = \exists\cdot\mathbf{BP}\cdot\exists\cdot\mathbf{P}$.

Note that in this definition, Arthur's coins are public. At the same time, Goldwasser, Micali, and Rackoff [126] defined a similar class in which the coin tosses are all private—they need not be revealed to the verifier. Both of these models are known as *interactive proofs*. Subsequently, Goldwasser and Sipser [127] showed that "public coins are as good as private coins," that is, the class of languages with constant-round interactive proofs is exactly **AM**.

Subsequently it was shown that $\mathsf{GI} \in \mathbf{coAM}$ [125, 127].

Although it will not be relevant, we feel we should mention one of the crowning achievements of complexity theory in the 1990s. The class **IP** consists of those languages that have interactive proofs with a polynomially bounded number of rounds, that is, the number of rounds can grow as a polynomial of the size of the input. One of the two non-relativizing proof techniques currently known—arithmetization—was developed in the course of proving that $\mathbf{IP} = \mathbf{PSPACE}$ [187, 238] (see also [31, 29, 4] for related work).

Quantum complexity. The class **BQP** consists of those languages that can be decided on a quantum computer in polynomial time with error strictly bounded away from 1/2, as in the definition of **BPP**. For more details on quantum computing, we recommend the book by Nielson and Chuang [211].

Function classes

Complexity-bounded function classes are defined in terms of *Turing transducers*: Turing machines with an additional write-only output tape. A transducer only outputs a value if it enters an accepting state. In general, then, a nondeterministic transducer can be partial and/or multi-valued. Whenever we say "partial" or "multivalued," we mean "potentially partial" and "potentially multivalued." For such a function f, we write

$$\textit{set-}f(x) = \{y : \text{ some accepting computation of } f \text{ outputs } y\}$$

The *domain* of a partial multi-valued function is the set $\mathrm{dom}(f) = \{x : \textit{set-}f(x) \neq \emptyset\}$. The *graph* of a partial multi-valued function is the set $\mathrm{graph}(f) = \{(x, y) : y \in \textit{set-}f(x)\}$.

The class **FP** is the class of all total functions computable in polynomial time. The class **PF** is the class of all partial functions computable in polynomial time. Note that machines computing a **PF** function must halt in polynomial time even when they make no output.

Logarithmic-space functions. The class **FL** is the class of all (single-valued, total) functions computable by a logspace transducer. Note that neither the input tape nor the output tape is counted in the space usage.

Nondeterministic functions. The class **NPSV** consists of all single-valued partial functions computable by a nondeterministic polynomial-time transducer. Note that multiple branches of an **NPSV** transducer may accept, but they must all have the same output.

The class **NPMV** consists of all multi-valued partial functions computable by a nondeterministic polynomial-time transducer.

The classes $\mathbf{NPSV_t}$ and $\mathbf{NPMV_t}$ are the subclasses of **NPSV** and **NPMV**, respectively, consisting of the total functions in those classes.

The classes $\mathbf{NPSV_g}$ and $\mathbf{NPMV_g}$ are the subclasses of **NPSV** and **NPMV**, respectively, whose graphs are in **P**.

A *refinement* of a multi-valued partial function f is a multi-valued partial function g such that $\mathrm{dom}(g) = \mathrm{dom}(f)$ and $\textit{set-}g(x) \subseteq \textit{set-}f(x)$ for all x. In particular, if $\textit{set-}f(x)$ is nonempty then so is $\textit{set-}g(x)$.

If $\mathcal{F}_1$ and $\mathcal{F}_2$ are two classes of partial multi-valued functions, then we write $\mathcal{F}_1 \subseteq_c \mathcal{F}_2$ to indicate that every function in $\mathcal{F}_1$ has a refinement in $\mathcal{F}_2$.

It is known that $\mathbf{NPMV} \subseteq_c \mathbf{PF}$ if and only if $\mathbf{P} = \mathbf{NP}$ [235] if and only if $\mathbf{NPSV} \subseteq \mathbf{PF}$ [237]. Selman [236] is one of the classic works in this area, and gives many more results regarding these function classes.

The following theorem is our main formal evidence for believing that $\mathbf{NPMV} \not\subseteq_c \mathbf{NPSV}$:

Theorem 2.1.1 (Hemaspaandra, Naik, Ogihara and Selman [138])**.** *The following conditions are equivalent:*

1. *There is a function $f \in \mathbf{NPSV}$ such that, for any formula φ, $f(\varphi)$ is a satisfying assignment of φ, if one exists, or $\perp$ otherwise;*
2. $\mathbf{NPMV} \subseteq_c \mathbf{NPSV}$*;*
3. $\mathbf{NPMV_g} \subseteq_c \mathbf{NPSV}$ *[236].*

If any, and hence all, of the above conditions hold, then $\mathbf{PH} = \mathbf{\Sigma_2 P}$.

In fact, they showed that the conditions of the above theorem imply $\mathsf{SAT} \in (\mathbf{NP} \cap \mathbf{coNP})/poly$ [138]. At the time, this was only known to imply $\mathbf{PH} = \mathbf{\Sigma_2 P}$, but shortly thereafter the collapse was improved to $\mathbf{PH} = \mathbf{ZPP^{NP}}$ [170].

We note that $\mathbf{NPMV_g} \subseteq_c \mathbf{NPSV_g}$ obviously implies $\mathbf{NPMV_g} \subseteq_c \mathbf{NPSV}$, and hence that the conditions of the above theorem hold, but that the converse, namely the implication $\mathbf{NPMV} \subseteq_c \mathbf{NPSV} \implies \mathbf{NPMV_g} \subseteq_c \mathbf{NPSV_g}$, is not known to hold.

It is not difficult to show that $\mathbf{NPMV_g} \subseteq_c \mathbf{NPSV_g}$ implies $\mathbf{NP} = \mathbf{UP}$; we review a proof of this fact in the proof of Corollary 5.3.2. Again, the converse is not known to hold. We also note that it is still an open question as to whether $\mathbf{NP} = \mathbf{UP}$ implies any collapse of $\mathbf{PH}$ at all.

The following diagram summarizes these implications:

$$\begin{array}{ccccc}
\mathbf{NPMV_g} \subseteq_c \mathbf{NPSV_g} & \Longrightarrow & \mathbf{NPMV_g} \subseteq_c \mathbf{NPSV} & \overset{[138]}{\Longrightarrow} & \mathsf{SAT} \in (\mathbf{NP} \cap \mathbf{coNP})/poly \\
\Downarrow & & \Updownarrow [236] & & \Downarrow \\
\mathbf{NP} = \mathbf{UP} & & \mathbf{NPMV} \subseteq_c \mathbf{NPSV} & & \mathbf{PH} = \mathbf{\Sigma_2 P}\ [138] \\
 & & & & \mathbf{PH} = \mathbf{ZPP^{NP}}\ [170]
\end{array}$$

Any implication not present in the above diagram is not known to hold, nor are there oracles known to settle these non-implications either way.

Counting classes

The function class $\#\mathbf{P}$ is defined as the class of functions f such that there is a nondeterministic Turing machine M such that $f(x)$ is the number of accepting paths of $M(x)$. The class $\#\mathbf{L}$ has the same relationship to $\mathbf{NL}$ as $\#\mathbf{P}$ does to $\mathbf{NP}$.

The class $\mathbf{GapP}$ is the set of differences of $\#\mathbf{P}$ functions, that is, $\mathbf{GapP} = \{f - g : f, g \in \#\mathbf{P}\}$. One may similarly define $\mathbf{GapL}$. $\mathbf{GapP}$ may also be defined as $\{f - g : f \in \#\mathbf{P}, g \in \mathbf{FP}\}$. In particular, $\mathbf{P}^{\#\mathbf{P}} = \mathbf{P}^{\mathbf{GapP}}$, even if we restrict the reducing machines to make only a single query to the $\#\mathbf{P}$ oracle. Note that all functions in $\#\mathbf{P}$ are nonnegative, whereas $\mathbf{GapP}$ contains functions with both positive and negative values. However, it is not known if every nonnegative function in $\mathbf{GapP}$ is in $\#\mathbf{P}$. For further discussion of this and other properties of these counting classes, see the survey by Fortnow [107].

Counting solutions seems to be much more powerful than detecting the existence of a solution. For example, although $\#\mathbf{P}$ is a counting analog of $\mathbf{NP}$, its power seems to be much greater than that of $\mathbf{NP}$: Toda's Theorem [261] states that $\mathbf{PH} \subseteq \mathbf{P}^{\#\mathbf{P}}$.

The permanent of a matrix is defined like the determinant, but without signs:

$$\operatorname{perm}(X) = \sum_{\pi \in S_n} x_{1,\pi(1)} \cdots x_{n,\pi(n)}.$$

Valiant [269] showed that computing the permanent of matrices with $0,1$-entries is $\#\mathbf{P}$-complete (equivalently, $\mathbf{GapP}$-complete). Similarly, computing the determinant of integer matrices is $\mathbf{GapL}$-complete [93, 262, 273].

Advice

Karp and Lipton [157] introduced the notion of Turing machines that take advice. Advice comes in the form of a number of bits that depend only on the input length n, and not on anything else about the input. This is the only requirement on the advice bits—they need not be generated by a Turing machine, for example. In other words, any function $f\colon \mathbb{N} \to \Sigma^*$ is a legal advice function.

If $\ell\colon \mathbb{N} \to \mathbb{N}$ is any function and $\mathcal{C}$ is a complexity class (technically, a class of machines), then $\mathcal{C}/\ell$ is the class of languages decidable by a machine in $\mathcal{C}$ with advice of length at most

ℓ. The most frequently used class is $\mathbf{P/poly}$, which is the class of languages decidable in polynomial time with polynomial-length advice. We will see in the next section that advice is closely related to circuit complexity.

As an example of the utility of advice, suppose a language $L \subseteq \Sigma^*$ is *sparse*: the number of strings in L up to length n is bounded by a polynomial in n. Then $L \in \mathbf{P/poly}$. Simply take the advice at length n to be the concatenation of all the strings in L of length n. On input x, a polynomial-time machine can check whether x is in the list provided by the advice, thus deciding correctly whether or not $x \in L$. In fact, Meyer observed (according to Berman and Hartmanis [52]) that $\mathbf{P/poly}$ is exactly the class of languages that are polynomial-time Turing reducible to a sparse language.

2.1.4 Circuit complexity

Boolean circuits are used to model the complexity of problems in a way that is not necessarily uniform in the input size n; in other words, circuits provide a formal model of computational complexity of a problem where each input length n can have its own algorithm. From the perspective of algorithms, this may not seem particularly useful, but this view has dominated most lower bounds research in the last 30 years. For example, to show that $\mathbf{P} \neq \mathbf{NP}$, it suffices to show the stronger statement that $\mathbf{NP}$ does not have (non-uniform) polynomial-size circuits.

A *Boolean circuit* is a computational device computing a function on a fixed number of input bits, say n. It is defined by a directed acyclic graph, in which there are exactly n nodes with no in-edges, called "input nodes," that are labeled by the input variables $x_1, \ldots, x_n$. All non-input nodes are called "gates," and are labeled by a computational operation such as AND, OR, or NOT. Each gate performs the specified computational operation, taking as input the values coming along its in-edges, and putting its output on all of its out-edges. A gate with no out-edges is called an "output node" of the circuit. The output nodes are ordered. If we call them $y_1, \ldots, y_m$, then a circuit C on input x computes the function $C(x) = y$. As it should never cause confusion, we often use C to denote both the circuit and the function it computes.

Just as there are several measures of complexity of an algorithm, there are several measures of the complexity of a circuit. The two most frequently used complexity measures are

the *size* of a circuit—the number of edges—and the *depth* of a circuit—the length of the longest directed path from any input node to any output node.

The *fan-in* of a circuit is the maximum in-degree of any node, and the *fan-out* is the maximum out-degree. A circuit of fan-out 1 is called a *formula*.

A family of circuits $(C_n)_{n=1}^{\infty}$ computes a function $\Sigma^* \to \Sigma^*$ by applying C_n to strings of length n. Note that, unlike Turing machines, the circuits C_n and C_m need not be related to one another when $n \neq m$.

The *circuit-size complexity* of a function $f\colon \Sigma^n \to \Sigma^m$ is the minimum size of a circuit computing f. When no further specification is made, we mean circuits with AND, OR, and NOT gates. The circuit-size complexity of a function $f\colon \Sigma^* \to \Sigma^*$ is the function $n \mapsto \text{circuit-size}(f|_{\Sigma^n})$. The circuit-size complexity of a language $L \subseteq \Sigma^*$ is the circuit-size complexity of its characteristic function. One similarly defines circuit-depth complexity.

The class of languages decidable by polynomial-size circuit families, where the size of C_n is bounded by a polynomial in n, coincides with the advice class $\mathbf{P/poly}$. In one direction, the circuit C_n can be encoded into the advice for length n; in the other direction, the advice for length n can be hard-coded into C_n.

A circuit family (C_n) is said to be an $\mathbf{AC}^i$ circuit family if the size of C_n is bounded by a polynomial in n, and the depth of C_n is bounded by $\log^i n$. Here the superscript denotes exponentiation, as in $(\log n)^i$ (rather than iteration $\log\log\log\cdots\log n$). A circuit family is said to be an $\mathbf{NC}^i$ circuit family if it is an $\mathbf{AC}^i$ circuit family that in addition has bounded fan-in; that is, there is a universal constant c such that the fan-in of all C_n is at most c. The complexity class $\mathbf{AC}^i$ consists of all languages computed by $\mathbf{AC}^i$ circuit families, and similarly for $\mathbf{NC}^i$.

A *threshold (or majority) gate* is a gate with k inputs whose output is 1 if and only if at least $\lfloor k/2 \rfloor + 1$ of its inputs are 1. The complexity class $\mathbf{TC}^i$ is defined just like $\mathbf{AC}^i$, except that $\mathbf{TC}^i$ circuits may, in addition, have threshold gates. Typically only $\mathbf{TC}^0$ is studied.

We define the hierarchies $\mathbf{AC} = \bigcup_i \mathbf{AC}^i$ and similarly for $\mathbf{NC}$ and $\mathbf{TC}$. As $\mathbf{AC}^i \subseteq \mathbf{NC}^{i+1} \subseteq \mathbf{AC}^{i+1}$, we have $\mathbf{AC} = \mathbf{NC}$. Moreover, $\mathbf{AC}^i \subseteq \mathbf{TC}^i \subseteq \mathbf{AC}^{i+1}$, so again we have $\mathbf{TC} = \mathbf{NC}$. Typically this whole hierarchy is only referred to as $\mathbf{NC}$, though for specific i the classes $\mathbf{AC}^i, \mathbf{NC}^i, \mathbf{TC}^i$ are usually referred to individually.

One of the first great achievements in circuit complexity was the lower bound the the PARITY function—what is the parity of the number of 1s in the input x—is not in $\mathbf{AC}^0$, thus showing that $\mathbf{AC}^0 \neq \mathbf{TC}^0$ [120].

Uniformity conditions. Because of the non-uniformity in n in the above models, even "small" circuit classes such as $\mathbf{AC}^0$ contain *uncomputable* languages in them. Comparing such classes to uniform classes like $\mathbf{L}$ and $\mathbf{P}$ thus makes little sense without addition restrictions.

A *uniformity condition* on a circuit class is the requirement that the function $1^n \mapsto C_n$ be computable in a particular class. For example, $\mathbf{P}$-uniform $\mathbf{NC}^i$ is the class of languages decided by $\mathbf{NC}^i$ circuit families (C_n) where the function $1^n \mapsto C_n$ can be computed in poly(n) time. Other often-used uniformity conditions are $\mathbf{L}$-uniform and **DLOGTIME**-uniform. This last deserves some discussion, since deterministic logarithmic time does not even allow enough time to read the entire input. The idea here is that questions about the circuit C_n, such as "is gate i an input to gate j?" can be answered in $O(\log n)$ time. There is a consensus that for really small circuit classes like $\mathbf{AC}^0$ and $\mathbf{TC}^0$, this is the "right" uniformity condition. It is sometimes referred to as "fully uniform." See Barrington, Immerman, and Straubing [41] and Ruzzo [228].

With full-uniformity, we have the following inclusions:

$$\mathbf{NC}^1 \subseteq \mathbf{L} \subseteq \mathbf{NL} \subseteq \mathbf{DET} \subseteq \mathbf{NC}^2 \subseteq \mathbf{NC} \subseteq \mathbf{P}$$

where **DET** is the class of decision problems that are logspace Turing reducible to computing the determinant of an integer matrix. Note that **DET**, although a class of languages, has essentially the same power as the function class **GapL**.

2.1.5 Algebraic complexity

In algebraic complexity, we are typically interested in the number of *arithmetic* operations—typically just additions, subtractions, multiplications, and divisions, but occasionally other operations such as extracting square roots, etc.—needed to perform an algebraic computation. Algebraic complexity can be studied over any ring R, or even other algebraic structures

such as the $(\min, +)$ semi-ring. In this thesis we will only be considering algebraic questions over fields $\mathbb{F}$.

Because we are only concerned with the number of arithmetic operations, field elements are often considered "indivisible units" rather than being represented in any particular way over a finite alphabet. This enables us to talk about algebraic complexity over uncountable fields such as the real and complex numbers, even though these cannot be specified exactly on a Boolean computer.

There are three related but distinct computational models that are most popularly used in algebraic complexity, which we now briefly review.

An *algebraic circuit* is defined just as in Boolean circuits, except instead of the gates being labeled by logical operations, they are labeled by arithmetic operations—addition, multiplication, subtraction. When we wish to allow division as well we will explicitly mention it. However, a result of Strassen [256] says that, for the computation of polynomials, division gates do not substantially help. Each input can be any field element and a single wire carries a field element on it, rather than just a single bit as in the Boolean model. The notions of circuit size and depth are defined exactly as in the Boolean model. As with Boolean circuits, this is a non-uniform model of computation, in that if (C_n) is a family of algebraic circuits where C_n has n inputs, there need not be any relation between C_n and C_m when $m \neq n$.

The Blum–Shub–Smale (or BSS) model [58] essentially generalizes the "real RAM model," and provides a nice uniform counterpart to the algebraic circuit model. They start with the definition of Turing machine, but each tape cell is now allowed to contain an element from the base ring or field of computation, and arithmetic operations can be performed in a single step. Constants from the field may also be built into the control of the Turing machine. Defining analogs of complexity classes such as $\mathbf{P}$ and $\mathbf{NP}$ is often quite easy in the BSS model, because the definitions of these classes in the Boolean model are so closely tied to Turing machines. With some classes there are some technical hurdles, but we will not encounter any of these intricacies in this thesis. The corresponding complexity classes in the BSS model over a ring R are denoted $\mathbf{P}_R$ and $\mathbf{NP}_R$, for example, $\mathbf{P}_{\mathbb{C}}$.

Valiant's model is somewhat closer to the algebraic circuit model, but with a few differences. Call a family of polynomials (f_n) a *p-family* if the degree and number of variables of f_n are bounded by a polynomial in n. The class $\mathbf{VP}$ ("V" for "Valiant") consists of

those p-families that are computable by polynomial-size arithmetic circuits. Such families are called *p-computable*.

As with the other models, there is a notion of **VP** for each field $\mathbb{F}$. Technically this can work over other algebraic structures as rings, but typically Valiant's classes are only discussed over fields. When we wish to emphasize the field, we may write $\mathbf{VP}_{\mathbb{F}}$.

The class **VNP** consists of those p-families (f_n) such that there is a p-family (g_n) satisfying:

$$f_n(x_1, \ldots, x_{p(n)}) = \sum_{e \in \{0,1\}^{q(n)}} g_n(e_0, \ldots, e_{q(n)}, x_1, \ldots, x_{p(n)})$$

where $p(n)$ and $q(n)$ are both polynomials. Such a family (f_n) is called *p-definable*. This is essentially equivalent to the coefficients of f_n being p-computable. For other equivalent definitions, see the books by Bürgisser [69] and Bürgisser, Clausen and Shokrollahi [70].

The notion of reduction used in Valiant's theory is that of projections: we say a polynomial f is a projection of a polynomial g if f can be gotten from g by replacing the variables of g with constants or (possibly other) variables. A family (f_n) is a *p-projection* of a family (g_n) if there is a polynomial $p(n)$ such that f_n is a projection of $g_{p(n)}$. Using this notion of reduction one defines the notion of hardness and completeness in the usual manner.

Valiant [269] showed that the permanent is **VNP**-complete. Thus, although **VNP** is in some sense an algebraic analog of the Boolean class **NP**, it is much closer to the Boolean counting class #**P**.

In the Boolean world, the determinant of integer matrices is **GapL**-complete. It is perhaps surprising then that in the algebraic world—where the determinant was first defined and naturally lives—the place of the determinant in terms of algebraic complexity is somewhat tricky. For example, there are Valiant analogs of the circuit classes $\mathbf{NC}^i$; however, although the determinant is very close to $\mathbf{NC}^2$ in the Boolean world, in Valiant's theory $\mathbf{VNC}^2 = \mathbf{VP}$, as follows from a classical depth-reduction result for circuits [267].

In complexity, the word *quasipolynomial* is taken to mean any function of the form $2^{O(\log^k n)}$ for fixed k. One defines quasipolynomial projections, or qp-projections, just as p-projections, and defines **VQP** as the quasipolynomial analog of **VP**. Then the determinant is **VQP**-complete under qp-projections. However, although this is an exact characterization

of the determinant in Valiant's model, it is again somewhat unsatisfying since we know that the determinant is in **VP**.

Another characterization was given by Malod and Portier [191]. They define a restricted kind of circuit that they call "weakly skew," and show that the determinant is complete under p-projections for the weakly skew analog of **VP**, denoted $\mathbf{VP}_{ws}$. The permanent versus determinant question in the algebraic world is thus the same as $\mathbf{VP}_{ws}$ versus **VNP** question.

2.1.6 Barriers: relativization, algebrization, and natural proofs

Baker, Gill, and Solovay [38] showed that there are oracles A and B relative to which $\mathbf{P}^A = \mathbf{NP}^A$ and $\mathbf{P}^B \neq \mathbf{NP}^B$. Thus any proof technique which would work just as well in the presence of oracles—in other words, that *relativizes* (to any oracle)—cannot be used to resolve the **P** versus **NP** question. In particular, this rules out the powerful techniques of simulation and diagonalization. This result can be taken somewhat in the spirit of the independence results of Cohen [84, 85]. In some sense, this shows that **P** versus **NP** cannot be resolved by an argument based "solely on the axioms" for a Turing machine, but must somehow use some specific properties of a specific Turing machine or specific language.

The first non-relativizing result was **IP** = **PSPACE** [187, 238]. This was shown unconditionally, after Fortnow and Sipser [111] had given an oracle A relative to which $\mathbf{coNP}^A \not\subseteq \mathbf{IP}^A$. Using the technique of arithmetization [30], a whole new series of results arose and led to new directions for complexity theory: interactive and probabilistically checkable proofs, together with approximation algorithms and inapproximability results. In the wake of these results, people naturally wondered what it was that made the proof techniques not relativize.

Arora, Impagliazzo, and Vazirani [15] attempted to formalize this notion by giving an axiom system based on "local checkability," such that theorems proved in this axiom system are exactly the relativizing theorems. In particular, they showed that the aforementioned non-relativizing results could not be proved within their axiom system. However, their axiom system is somewhat controversial; Fortnow [106], argues that their axiom system, when given a natural oracle access mechanism, does in fact relativize, and that the crucial property of

the non-relativizing results was their algebraic nature. To argue for this thesis, he showed that the **IP** = **PSPACE** "relativizes with a low-degree polynomial extension of an oracle."

This latter idea was eventually formalized and strengthened to the notion of *algebraic relativization* or *algebrization* for short [4]. Aaronson and Wigderson [4] showed that algebrization captures very closely the limits of our current techniques, as well as providing an interesting connection with communication complexity. Impagliazzo, Kabanets, and Kolokolova [145] extended the ideas of Arora–Impagliazzo–Vazirani to algebraic relativization.

We remark that relativization results have somewhat fallen out of favor in the community, though we still firmly believe in their relevance. We highly recommend Fortnow [106] and Aaronson [2, Section 1.3] for discussions of the importance and utility of oracle results.

Razborov and Rudich [221] showed that a different class of proof techniques, that they call "natural proofs," cannot be used to show $\mathbf{NP} \not\subseteq \mathbf{P/poly}$. A proof or proof technique is said to "naturalize" if it can be converted into a natural proof; this conversion process is not always obvious, as they demonstrate in their paper. A "natural proof" actually consists of a key useful property; $(C_n)_{n=1}^{\infty}$ is a useful property if each C_n is a subset of the n-variable Boolean functions with a subset C_n^* such that

1. (Constructivity) Determining whether an n-variable Boolean function is in C_n^* can be done in polynomial time in the size of the truth table of the function;

2. (Largeness) $|C_n^*| \geq 2^{-O(n)}2^{2^n}$, where 2^{2^n} is the number of Boolean functions on n variables;

3. (Usefulness) The circuit-size of any sequence of functions $f_1, f_2, \ldots, f_n, \ldots$ where $f_n \in C_n$ is super-polynomial. That is, for any k there is a sufficiently large n such that the circuit-size of f_n is greater than n^k.

Razborov and Rudich show that if such a natural property exists, then 2^{n^ε}-hard pseudorandom generators do not exist. In particular, if such a natural property exists then factoring integers can be done in time 2^{n^ε} for any $\varepsilon > 0$, which is significantly better than the current state of the art, which is $2^{O(n^{1/3}(\log n)^{2/3})}$ for n-bit numbers [89].

2.2 Algebra

2.2.1 Equivalence relations

A binary relation R on a set X is a subset of $X \times X$. When we speak of relations as algorithmic problems, we will take $X = \Sigma^*$, the set of all strings. If R is an equivalence relation and $(x, y) \in R$, we write $x \sim_R y$. An *equivalence relation* is

1. reflexive: $x \sim_R x$ for all x;
2. symmetric $x \sim_R y \iff y \sim_R x$ for all x, y;
3. transitive: if $x \sim_R y$ and $y \sim_R z$ then $x \sim_R z$.

If f is any function, then the *kernel* of f is

$$\mathrm{Ker}(f) = \{(x, y) : f(x) = f(y)\}.$$

It is clear that $\mathrm{Ker}(f)$ is an equivalence relation. If $R = \mathrm{Ker}(f)$ then f is said to be a *complete invariant* for R. A *canonical form* for an equivalence relation R is a function g such that $x \sim_R g(x)$ for all x, and $x \sim_R y$ if and only if $g(x) = g(y)$. Note that if g is a canonical form for R, then $\mathrm{Ker}(g) = R$ and g is idempotent, that is, $g \circ g = g$. Indeed, g is a canonical form for R if and only if g is an idempotent complete invariant for R.

If R is an equivalence relation, then the *equivalence class* of the string x is $[x]_R = \{y : x \sim_R y\}$. The equivalence classes of R partition X.

The *trivial relation* is all of $X \times X$, that is, all elements of X are equivalent under the trivial relation, or equivalently $[x] = X$ for all x. The *discrete relation* is the relation of equality, that is, each element is equivalent only to itself under the discrete relation. Equivalently, the discrete relation is defined by $[x] = \{x\}$ for every x.

2.2.2 Groups

A group is a set G together with a binary operation $\cdot : G \times G \to G$, usually written in infix notation as $a \cdot b$ or sometimes omitted altogether, as in ab, satisfying three axioms:

1. Associativity: $a(bc) = (ab)c$ for all $a, b, c \in G$

2. Identity: there is an element $1 \in G$ such that $1a = a1 = a$ for all $a \in G$;

3. Inverses: for each a in G there is an element $b \in G$ such that $ab = 1$.

A group is called *abelian*, after Niels Henrik Abel, if it also satisfies the axiom of commutativity: $ab = ba$ for all $a, b \in G$. It is sometimes customary to denote the operation in an abelian group using additive, rather than multiplicative, notation, so $a + b$ rather than ab, $-a$ for the inverse of a, and 0 for the identity. We also speak of the sum of elements rather than product, and denote by na the n-fold sum of a with itself (analogous to the multiplicative notation a^n above). If we use additive notation for a group we will explicitly mention it, with the exception of the additive group of a ring, module, or vector space (see below).

If G is a group, then a *subgroup* of G is a subset $H \subseteq G$ such that $1 \in H$, and H is closed under the group operation and under taking inverses. In other words, $a \in H$ implies $a^{-1} \in H$, and $a, b \in H$ implies $ab \in H$. If H is a subgroup of G, we write $H \leq G$.

If $A, B \subseteq G$, then we write AB for the set of products $\{ab : a \in A, b \in B\}$. If either set is a singleton, we denote it by a single element without braces, as in aB, for some $a \in G$. We use this notation iteratively, so $ABA = \{a_1 b a_2 : a_1, a_2 \in A, b \in B\}$ and $aBa^{-1} = \{aba^{-1} : b \in B\}$. We also write $A^{-1} := \{a^{-1} : a \in A\}$.

The *order* or *size* of G is its cardinality as a set, and is denoted $|G|$.

Proposition 2.2.1 (Lagrange's Theorem; definition of index)**.** *If $H \leq G$, then $|H|$ divides $|G|$. The number $|G|/|H|$ is called the* index *of H in G and is denoted $[G : H]$. Hence $|G| = |H|[G : H]$.*

Proof sketch. For $H \leq G$, we call aH a *(left) coset* of H in G. The result follows from the claim that the cosets of H are all of equal size and partition the group G. A key step is to note that $aH = bH$ if and only if $b^{-1}a \in H$. □

The collection of (left) cosets of $H \leq G$ is denoted $G/H = \{aH : a \in G\}$ and has cardinality $[G : H]$.

A map $\varphi\colon G_1 \to G_2$ is a *group homomorphism* if $\varphi(ab) = \varphi(a)\varphi(b)$ for all $a, b \in G_1$, where on the left-hand side the multiplication ab takes place in G_1 and on the right-hand side the multiplication takes place in G_2. The *kernel* of a homomorphism φ is $\ker(\varphi) := \{g \in$

$G_1 : \varphi(g) = 1\}$, and the image is the image of φ as a map of sets. Note that in algebra the use of the word kernel is slightly different than the more general usage for equivalence relations, above. We use the lowercase $\ker(\varphi)$ for the algebraic meaning, and the uppercase $\mathrm{Ker}(\varphi)$ for the equivalence relation. The two are related, however: if $H = \ker(\varphi)$ (algebraic sense), then $\mathrm{Ker}(\varphi)$ (equivalence relation) is the equivalence relation $\{(a, b) : \varphi(a) = \varphi(b)\}$, which is the same as $\{(a, b) : aH = bH\}$, so the group kernel completely determines the equivalence relation kernel. Unless otherwise specified, the "kernel of a group homomorphism" is the subgroup just introduced, and *not* the equivalence relation kernel.

Any vector space V, together with its addition operation, forms an abelian group. A linear map between vector spaces is, in particular, a group homomorphism. The kernel of a linear map in the sense of linear algebra—the set of vectors mapped to zero—is identical to the kernel of the linear map when considered as a homomorphism of abelian groups.

A homomorphism is surjective (onto), injective (one-to-one), or bijective if it is so as a map of sets. An *isomorphism* is a bijective homomorphism. Two groups G_1, G_2 are *isomorphic* if there exists an isomorphism between them, and we denote this by $G_1 \cong G_2$. The set-theoretic inverse $\varphi^{-1} \colon G_2 \to G_1$ of a group isomorphism is also a group isomorphism. Isomorphic groups have "the same" group structure, and are essentially different labelings or namings of the same group.

An *automorphism* is an isomorphism from a group to itself. If $\varphi_1 \colon G_1 \to G_2$ and $\varphi_2 \colon G_2 \to G_3$ are group homomorphisms, then the composition $\varphi_2 \circ \varphi_1 \colon G_1 \to G_3$ is a group homomorphism. In particular, if we take $G_1 = G_2 = G_3 = G$, then since every automorphism has an inverse, the collection of automorphisms of a fixed group G is itself a group, with the group operation being composition of maps. This group is denoted $\mathrm{Aut}(G)$.

The kernel and image of a group homomorphism are always groups. In fact, the kernel of a homomorphism satisfies the stronger property of being a normal subgroup. A subgroup $H \leq G$ is *normal* if $gHg^{-1} = H$ for every $g \in G$. The property of being a normal subgroup depends on both the subgroup and its ambient group. That is, it is possible for $H \leq G \leq K$ with H normal in G but H not normal in K. We denote normal subgroups by $H \trianglelefteq G$.

If $a, b \in G$, then the *conjugate* of a by b is bab^{-1}. Hence H is a normal subgroup of G if and only if every conjugate of H in G is equal to H.

The kernel of a homomorphism is a normal subgroup. Conversely, every normal subgroup is the kernel of some homomorphism. Namely, if $N \trianglelefteq G$, then we may define the *quotient group* by giving the group operation on the coset space G/N. The group operation is given by $(aN)(bN) = abN$; that this is well-defined follows from the equality of sets $aNbN = abNb^{-1}bN = abNN = abN$. By viewing the cosets as sets, rather than elements in their own right, the axioms of a group are easily verified for G/N.

Proposition 2.2.2 (First Isomorphism Theorem)**.** *If* $\varphi\colon G_1 \to G_2$ *is a group homomorphism, then* $\operatorname{im}(\varphi) \cong G/\ker(\varphi)$.

Given any set $S \subseteq G$, the subgroup *generated by* S is the smallest subgroup of G containing S, and is denoted $\langle S \rangle$. For finite groups it is plain to see that such a subgroup exists; for infinite groups one uses Zorn's Lemma. If $\langle S \rangle = G$, then S is said to be a generating set for G. Note that every finite group has a generating set of size at most $\log_2 |G|$: indeed, suppose $a_1, \ldots, a_k$ is a sequence of group elements, let A_i denote $\langle a_1, \ldots, a_i \rangle$, and suppose $a_i \notin A_{i-1}$ for all i. Then as $A_{i-1} \lneq A_i$ for all i, and the index $[A_i : A_{i-1}]$ is an integer, the index must be at least two. Hence $|A_i| \geq 2|A_{i-1}|$. Hence, by the time $k \geq \log_2 |G|$, we must have $A_k = G$.

Important Classes and Examples of Groups

We now cover some of the most important examples of groups. A group is called *cyclic* if it is generated by a single element. A cyclic group is determined up to isomorphism by its order, so we may speak unambiguously of *the* cyclic group of order n, denoted C_n or sometimes $\mathbb{Z}/n\mathbb{Z}$—the integers modulo n with addition as the group operation.

The *symmetric group* S_n is the group of all permutations of the set $\{1, \ldots, n\}$. We may similarly define the symmetric group on any set X, in which case it is denoted $\operatorname{Sym}(X)$. The *alternating group* A_n is the unique index 2 subgroup of S_n; A_n consists of all even permutations; a permutation is even if the number of two-element swaps it takes to implement the permutation is even.

The *general linear group* of degree n over a field $\mathbb{F}$ is the set of all invertible $n \times n$ matrices over $\mathbb{F}$, and is denoted $\mathrm{GL}_n(\mathbb{F})$. When we wish to leave the field unspecified or it is clear from context, we may write GL_n. If V is any vector space over a field, $\mathrm{GL}(V)$, the general

linear group "on V" is the group of all invertible linear maps $V \to V$. If $\dim V = n$, then picking a basis for V yields an isomorphism $\mathrm{GL}(V) \cong \mathrm{GL}_n$.

A group is *simple* if it has no proper non-trivial normal subgroups. Equivalently, G is simple if and only if, whenever $\varphi\colon G \to H$ is a homomorphism, either φ is trivial ($\mathrm{im}(\varphi) = 1$) or φ is an isomorphism onto its image ($\ker(\varphi) = 1$). Taking quotient groups forgets certain information about the original group, thereby simplifying it and perhaps facilitating additional understanding. This tool is, by definition, unavailable for simple groups. The finite simple groups have been completely classified in what is perhaps the largest single theorem proven to date, spanning tens of thousands of pages by hundreds of authors (see, for example, the accounts by Solomon [249] and Aschbacher [17] and references therein). The finite simple groups fall into a few infinite families, plus 26 "sporadic" groups that do not fall into these families. The infinite families are: the cyclic groups of prime order (the only abelian finite simple groups), the alternating groups, and the groups of Lie type. The groups of Lie type are closely related to the simple classical Lie algebras over finite fields, which are the finite fields analogues of simple Lie algebras over $\mathbb{C}$ (see Section 2.2.4).

The *free group on n generators* F_n is defined as follows. It is an infinite group with n generators $X = \{x_1, \ldots, x_n\}$, sometimes denoted F_X. A *word* in the generators is a string over the alphabet $X \cup X^{-1} := \{x_1, \ldots, x_n, x_1^{-1}, \ldots, x_n^{-1}\}$. A word $a_1 \ldots a_k$ with each $a_i \in X \cup X^{-1}$ is called reduced if it is never the case that two adjacent letters are of the form $x_i x_i^{-1}$ or $x_i^{-1} x_i$. The elements of F_n are the reduced words over $X \cup X^{-1}$; words are multiplied by juxtaposition, and then reducing—eliminating all adjacent pairs $x_i x_i^{-1}$ and $x_i^{-1} x_i$ recursively until arriving at a reduced word. The group is called "free" because its generators are not constrained by any relations between them other than those required by the group axioms.

Free groups are characterized by the property that any map $X \to G$ from a set X to a group G extends uniquely to a group homomorphism $F_X \to G$.

A group can be defined by giving a *presentation* of the group. This consists of giving a generating set, say $x_1, \ldots, x_n$, and a set of relations that are required to hold of these generators. These are often written $\langle x_1, \ldots, x_n | r_1, \ldots, r_k \rangle$ where each r_i is a word over $X \cup X^{-1}$, called a *relator*. The group is defined as F_X / R, where R is the smallest normal subgroup of F_X containing the words $r_1, \ldots, r_k$. Such a group presentation satisfies a universal property

similar to a free group: any map $X \to G$ from a set X to a group G such that the words r_i become trivial in G extends uniquely to a group homomorphism $F_X/R \to G$. In general, group presentations need not be finite—they may contain infinitely many generators and relators. Group presentations are useful tools, but they can be difficult to work with; for example, it is uncomputable—there is no algorithm, regardless of efficiency—to tell from a group presentation whether or not the group is isomorphic to the trivial, one-element group [212, 60].

2.2.3 Rings, fields, and modules

Rings generalize the algebraic structure of the integers, the rational numbers, and $n \times n$ matrices. These are all examples of rings, and are good to keep in mind when reading the definition. A *ring* R is a set together with two binary operations $\cdot\colon R \times R \to R$ and $+\colon R \times R \to R$ satisfying the following axioms:

1. $(R, +)$ is an abelian group, with additive notation. In particular, its identity is denoted by 0.
2. $\cdot$ is an associative operation with two-sided identity denoted by 1.
3. Distributivity: $a(b + c) = ab + ac$ and $(b + c)a = ba + ca$ for all $a, b, c \in R$.

The following facts are easily derived from these axioms:

- The additive identity 0 is unique. Moreover, $0r = r0 = 0$ for all $r \in R$.
- The multiplicative identity 1 is unique, and is distinct from the additive identity, $1 \neq 0$, unless the ring consists of only a single element.
- If a ring element $r \in R$ has a (left) multiplicative inverse $s \in R$, that is $rs = 1$, then $sr = 1$, and s is the unique such inverse, which is thus denoted r^{-1}.

A ring is called *commutative* if its multiplication operation is commutative: $rs = sr$ for all $r, s \in R$. The integers $\mathbb{Z}$ and rationals $\mathbb{Q}$ are commutative rings; the set of $n \times n$ matrices is not commutative whenever $n \geq 2$.

A map $\varphi\colon R \to S$ is a *ring homomorphism* if $\varphi(1_R) = 1_S$, and φ preserves both the addition and multiplication, that is, $\varphi(r_1 + r_2) = \varphi(r_1) + \varphi(r_2)$ and $\varphi(r_1 r_2) = \varphi(r_1)\varphi(r_2)$. An *isomorphism* is a bijective homomorphism; an *automorphism* is an isomorphism from a ring to itself. Under composition of maps, the set of automorphisms of a ring R becomes a group, denoted $\mathrm{Aut}(R)$.

An *ideal* in a ring R is a subset $I \subseteq R$ such that I is closed under addition and under multiplication by arbitrary elements of R. That is, for all $i_1, i_2 \in I$ and $r \in R$, $i_1 + i_2 \in I$ and $ri_1, i_1 r \in I$. As with the notion of normal subgroup, the notion of ideal depends both on I and on the parent ring R. That is, it is possible to have an ideal I in R, R a subring of S, but I is not an ideal in S.

The kernel of a ring homomorphism $\varphi\colon R \to S$ is $\ker(\varphi) = \{r \in R : \varphi(r) = 0\}$. The kernel is an ideal of R, and the image of φ is a subring of S. Conversely, given any ideal $I \subseteq R$, we may define the quotient ring R/I. The additive group of R/I is just the additive quotient group (note that, under addition, I is a subgroup of $(R, +)$). The product is defined by $(a + I)(b + I) = ab + I$; it is easily verified that, since I is an ideal, this product is well-defined and gives a ring structure to R/I. As with groups, the first isomorphism theorem holds for rings: if $\varphi\colon R \to S$ is a ring homomorphism, then $\mathrm{im}(\varphi) \cong R/\ker(\varphi)$.

A *field* is a commutative ring in which every nonzero element has a multiplicative inverse. The rationals $\mathbb{Q}$, reals $\mathbb{R}$, and complex numbers $\mathbb{C}$ are fields. For every prime p and natural number n, there is a finite field of order p^n that is unique up to isomorphism, and is denoted $\mathbb{F}_{p^n}$. These are the only finite fields. These fields are nested, in that $\mathbb{F}_{p^n}$ is a subfield of $\mathbb{F}_{p^m}$ whenever n divides m.

If a ring R contains a field $\mathbb{F}$ as a subring and the elements of $\mathbb{F}$ commute with every element in R, then R is said to be an $\mathbb{F}$-algebra. Note that, in this case, the additive structure of R and the multiplication between $\mathbb{F}$ and R give R the structure of a (possibly infinite-dimensional) vector space over $\mathbb{F}$.

A *skew field* or *division ring* is like a field but need not be commutative. The quaternions $\mathbb{H}$ are a skew field: $\mathbb{H}$ is the four-dimensional $\mathbb{R}$-algebra with $\mathbb{R}$-basis $\{1, i, j, k\}$ and multiplication defined by the multiplication in $\mathbb{R}$ and the rules $i^2 = j^2 = k^2 = ijk = 1$. As a consequence $ij = k$, $ji = -k$, $jk = i$, $kj = -i$, $ki = j$, $ik = -j$. All finite division rings are in fact fields (a result originally due to Wedderburn and Dickson, see Parshall [215]).

The *characteristic* of a ring is the least positive integer n such that $1 + 1 + \cdots + 1 = 0$ (n times). If this never holds, we say the ring has characteristic zero. The characteristic of a field is necessarily either 0 or a prime number. In particular, the finite field $\mathbb{F}_{p^n}$ has characteristic p, as it contains $\mathbb{F}_p \cong \mathbb{Z}/p\mathbb{Z}$ as a subfield.

A field is *algebraically closed* if every polynomial with coefficients in the field has at least one root in the field. If $\mathbb{F}$ is any field, then its *algebraic closure* is denoted $\overline{\mathbb{F}}$, and is defined to be the smallest algebraically closed field containing $\mathbb{F}$; every field has a unique algebraic closure, up to isomorphism. $\mathbb{C}$ is algebraically closed, and is the algebraic closure of $\mathbb{R}$. The algebraic closure of $\mathbb{Q}$ is distinct from $\mathbb{C}$ and, following the above convention, is denoted $\overline{\mathbb{Q}}$. The algebraic closure of a finite field $\mathbb{F}_{p^n}$ is the "union" (this notion can be made precise) of $\mathbb{F}_{p^m}$ for all m. In particular, $\overline{\mathbb{F}}_{p^n} = \overline{\mathbb{F}}_p$.

If one field $\mathbb{F}$ contains another field $\mathbb{F}'$ as a subfield, then $\mathbb{F}$ is a vector space over $\mathbb{F}'$. The dimension of this vector space is said to be the *degree* of the field extension.

A field is *perfect* if either it has characteristic zero, or it has characteristic p and every element is a p-th power. In particular, all fields of characteristic zero, all finite fields, and all subfields of the algebraic closures $\overline{\mathbb{F}}_p$ of finite fields are perfect. There is a more satisfying, internal definition of perfectness, but for this thesis this equivalent definition suffices.

2.2.4 Lie algebras

For the purposes of this thesis, we highly recommend the book of De Graaf [97]. We summarize the necessary highlights here. For more general background on Lie algebras we recommend any of several standard books [116, 150, 143, 166]. For Lie algebras over non-algebraically closed fields or in positive characteristic, we recommend the books by Jacobson [150] and especially Seligman [234], though it is a good idea to have seen the case of Lie algebras over $\mathbb{C}$ before wading into modular Lie algebras [234].

A *Lie algebra* is a vector space $\mathcal{L}$ over a field $\mathbb{F}$ together with a bilinear operation, referred to as the *Lie bracket* and written $[\cdot, \cdot] \colon \mathcal{L} \times \mathcal{L} \to \mathcal{L}$ satisfying:

1. Skew-symmetry: $[v, v] = 0$ for all $v \in \mathcal{L}$ (or equivalently in characteristic $\neq 2$, $[v_1, v_2] = -[v_2, v_1]$)

2. Bi-linearity: $[\alpha v + \beta w, u] = \alpha[v, u] + \beta[w, u]$, and similarly for the second coordinate.

3. The Jacobi identity: $[u,[v,w]]+[w,[u,v]]+[v,[w,u]]=0$. This is the "Lie algebra" version of associativity, and can be thought of as "the derivative of the associative law."

A *matrix Lie algebra* is a set of matrices where taking $[A,B] := AB - BA$ makes the set into a Lie algebra. In particular, the collection $M_n(\mathbb{F})$ of all $n \times n$ matrices over $\mathbb{F}$ is a matrix Lie algebra.

A *homomorphism* between Lie algebras $\mathcal{L}_1, \mathcal{L}_2$ is a linear map $\rho\colon \mathcal{L}_1 \to \mathcal{L}_2$ that preserves the brackets, that is, where $\rho([u,v]_{\mathcal{L}_1}) = [\rho(u),\rho(v)]_{\mathcal{L}_2}$. An *isomorphism* is a bijective homomorphism; an *automorphism* is an isomorphism of $\mathcal{L}$ with itself.

Note that conjugate matrix Lie algebras are isomorphic as abstract Lie algebras, since $g[M_1,M_2]g^{-1} = [gM_1g^{-1}, gM_2g^{-1}]$, that is, conjugation by g is a Lie algebra homomorphism whose inverse is conjugation by g^{-1}.

Structure theory of Lie algebras

Given any two Lie algebras $\mathcal{L}_1, \mathcal{L}_2$, their *direct sum* is the Lie algebra $\mathcal{L}_1 \oplus \mathcal{L}_2$ whose underlying vector space is the direct sum of the underlying vector spaces of the $\mathcal{L}_i$. The bracket $[v_1, v_2]$ for any elements $v_1 \in \mathcal{L}_1$ and $v_2 \in \mathcal{L}_2$ is defined to be zero.

An *ideal* in a Lie algebra is a subspace $I \subseteq \mathcal{L}$ such that $[u,v] \in I$ for any $u \in \mathcal{L}$ and $v \in I$. Ideals are the Lie-algebraic analogue of normal subgroups of groups and ideals in rings. Given any ideal, one can form the quotient Lie algebra $\mathcal{L}/I$ whose elements are additive cosets of I, that is, of the form $v + I$; conversely, given any homomorphism of Lie algebras its kernel is an ideal.

A Lie algebra is *abelian* if $[u,v] = 0$ for all $u, v \in \mathcal{L}$. Any vector space can thus be given the structure of an abelian Lie algebra. Every subspace of an abelian Lie algebra is an ideal.

0 is the trivial ideal. An ideal is proper if it is not the whole Lie algebra. In a direct sum $\mathcal{L} = \mathcal{L}_1 \oplus \mathcal{L}_2$, each $\mathcal{L}_i$ is a proper ideal of $\mathcal{L}$. A Lie algebra is *simple* if it contains no proper non-trivial ideals, and is non-abelian. (This last condition excludes, for convenience, the 1-dimensional abelian Lie algebra.)

Given two ideals $A, B \subseteq \mathcal{L}$, their commutator is defined as $[A,B] := \mathrm{Span}\{[a,b] : a \in A, b \in B\}$; the commutator of two ideals is again an ideal (this is an exercise in the Jacobi

identity). The *derived series* of $\mathcal{L}$ is defined as follows: $\mathcal{L}^{(0)} := \mathcal{L}$, $\mathcal{L}^{(i+1)} := [\mathcal{L}^{(i)}, \mathcal{L}^{(i)}]$. $\mathcal{L}^{(1)} = [\mathcal{L}, \mathcal{L}]$ is called the *derived or commutator subalgebra*.

Definition 2.2.3. A Lie algebra $\mathcal{L}$ is *solvable* if the derived series terminates at $\mathcal{L}^{(k)} = 0$ for some k.

Each step in the derived series, $\mathcal{L}^{(i)}/\mathcal{L}^{(i+1)}$ is abelian, so solvable Lie algebras are "iterated extensions of abelian Lie algebras."

Every Lie algebra $\mathcal{L}$ has a unique maximal solvable ideal, called the *(solvable) radical* and denoted $\operatorname{Rad}\mathcal{L}$; this follows from the lemma that the sum (not necessarily direct) of two solvable ideals is again solvable. A Lie algebra is *semisimple* if $\operatorname{Rad}\mathcal{L} = 0$, or equivalently if $\mathcal{L}$ contains no abelian ideals. $\mathcal{L}/\operatorname{Rad}\mathcal{L}$ is always semisimple.

In order to state one of the main structural theorems of Lie algebras in characteristic zero, we define semidirect products and derivations. A *derivation* on a Lie algebra $\mathcal{L}$ is a linear map $d\colon \mathcal{L} \to \mathcal{L}$ such that $d([u,v]) = [u, d(v)] + [d(u), v]$. Note the similarity with the product rule for differentiation. Since a derivation is a linear map, we may compose two derivations as linear maps; then defining $[d_1, d_2] := d_1 \circ d_2 - d_2 \circ d_1$ makes the collection of derivations of $\mathcal{L}$ into a Lie algebra denoted $\operatorname{Der}(\mathcal{L})$.

Given two Lie algebras $\mathcal{L}_1, \mathcal{L}_2$ and a homomorphism $\varphi\colon \mathcal{L}_2 \to \operatorname{Der}(\mathcal{L}_1)$, we define the semi-direct product $\mathcal{L}_1 \rtimes_\varphi \mathcal{L}_2$ as follows. The underlying vector space is the direct sum of $\mathcal{L}_1$ and $\mathcal{L}_2$. On each of these subspaces, the Lie bracket is defined as it was originally. If $v \in \mathcal{L}_1$ and $d \in \mathcal{L}_2$ we define

$$[v, d] := \varphi(d)(v).$$

Extending by linearity and skew-symmetry, we find

$$[v_1 + d_1, v_2 + d_2] = [v_1, v_2] + \varphi(d_2)(v_1) - \varphi(d_1)(v_2) + [d_1, d_2]$$

where $v_i \in \mathcal{L}_1$ and $d_i \in \mathcal{L}_2$.

Conversely, if a Lie algebra $\mathcal{L}$ has an ideal I and a subalgebra $\mathcal{L}'$ such that $\mathcal{L}$ is the direct sum of I and $\mathcal{L}'$ as vector spaces, then $\mathcal{L} = I \rtimes \mathcal{L}'$, where the map $\mathcal{L}' \to \operatorname{Der} I$ is given by the Lie bracket in $\mathcal{L}$.

The following two theorems are quite strong structural theorems. For example, nothing even close to these holds in the case of finite groups, despite the similarity in the definitions of all the notions (nilpotent, solvable, semidirect product). See Section 4.9.2 for more.

Theorem 2.2.4 (Levi's Theorem, cf. §III.9, p. 91 of Jacobson [150])**.** *Every Lie algebra in characteristic zero is the semidirect product of a solvable Lie algebra by a semisimple one. (That is, the semisimple one acts as derivations on the solvable one.)*

The *lower central series* is defined by $\mathcal{L}_0 := \mathcal{L}$ and $\mathcal{L}_{i+1} := [\mathcal{L}, \mathcal{L}_i]$. Note that here we take the commutator of $\mathcal{L}_i$ with the whole of $\mathcal{L}$, rather than just with $\mathcal{L}_i$ (as in the derived series). Hence the lower central series decreases more slowly than the derived series.

Definition 2.2.5. A Lie algebra $\mathcal{L}$ is *nilpotent* if the lower central series terminates at $\mathcal{L}_k = 0$ for some k.

Theorem 2.2.6 (see Corollary II.7.1 on p. 51 of Jacobson [150])**.** *A Lie algebra in characteristic zero is solvable if and only if its derived subalgebra is nilpotent.*

Remark 2.2.7. Since solvable Lie algebras are iterated extensions of abelian ones (see above), and considering Theorem 2.2.4, we may say that abelian and simple Lie algebras form the "building blocks" of all Lie algebras in characteristic zero. More generally, abelian and *semisimple* Lie algebras form the building blocks, over any field.

Over any algebraically closed field of characteristic zero, such as $\overline{\mathbb{Q}}$ or $\mathbb{C}$, semisimple Lie algebras—those with no abelian ideals—are the same as direct sums of simple Lie algebras. Furthermore, over algebraically closed fields of characteristic zero, the simple Lie algebras have been completely classified for nearly a century. They fall into four infinite families, referred to as type A_n ($\mathfrak{sl}_{n+1}$, consisting of all trace zero $(n+1) \times (n+1)$ matrices), B_n ($\mathfrak{so}_{2n+1}$, consisting of all $(2n+1) \times (2n+1)$ skew-symmetric matrices $M = -M^T$), C_n ($\mathfrak{sp}_{2n}$ consisting of all $2n \times 2n$ matrices M satisfying $JM = -M^T J$ where $J = \begin{pmatrix} 0 & I_n \\ -I_n & 0 \end{pmatrix}$), and D_n ($\mathfrak{so}_{2n}$), and there are five exceptional simple Lie algebras, known as $\mathfrak{e}_6$, $\mathfrak{e}_7$, $\mathfrak{e}_8$, $\mathfrak{f}_4$, and $\mathfrak{g}_2$. This "A,B,C,D,E,F,G" classification is sometimes referred to as the Cartan–Killing classification.

Cartan subalgebras

Cartan subalgebras are a key tool in understanding (non-nilpotent) Lie algebras. The *normalizer* of a subalgebra $\mathcal{H} \subseteq \mathcal{L}$ is the largest subalgebra of $\mathcal{L}$ in which $\mathcal{H}$ is an ideal; it is denoted $N_{\mathcal{L}}(\mathcal{H})$. Equivalently, the normalizer of $\mathcal{H}$ is the set of elements that send $\mathcal{H}$ into itself: $N_{\mathcal{L}}(\mathcal{H}) = \{x \in \mathcal{L} : (\forall h \in \mathcal{H})[[x, h] \in \mathcal{H}]\}$. A *Cartan subalgebra* of a Lie algebra $\mathcal{L}$ is a subalgebra $\mathcal{H}$ that is nilpotent and self-normalizing, that is, $N_{\mathcal{L}}(\mathcal{H}) = \mathcal{H}$. In particular, this implies that $\mathcal{H}$ is not an ideal of $\mathcal{L}$ unless $\mathcal{L}$ is nilpotent, in which case $\mathcal{H} = \mathcal{L}$. Cartan subalgebras are rarely unique.

The *adjoint representation* $\operatorname{ad}_{\mathcal{L}} x \colon \mathcal{L} \to \mathcal{L}$ is defined by $(\operatorname{ad}_{\mathcal{L}} x)(y) = [x, y]$. As $\operatorname{ad} x$ is a linear map, it makes sense to takes its powers $(\operatorname{ad} x)^k(y) = [x, [x, \ldots, [x, y]]]$ (k times).

A Cartan subalgebra is *split* (over $\mathbb{F}$) if all the eigenvalues of $\operatorname{ad}_{\mathcal{L}}(h)$ lie in $\mathbb{F}$ for every $h \in \mathcal{H}$. Having a split Cartan subalgebra is the generalization to Lie algebras of the property of a matrix being diagonalizable over $\mathbb{F}$. It is possible for a Lie algebra to have some Cartan algebras that are split and others that are not; we discuss this issue further when it becomes relevant, in Section 4.9.1.

Representations

A *representation of a Lie algebra* $\mathcal{L}$ is a homomorphism $\rho \colon \mathcal{L} \to M_n$ for some n. A representation is *faithful* if this homomorphism is injective. Two representations $\rho_1, \rho_2 \colon \mathcal{L} \to M_n$ are *equivalent* if there is an invertible $n \times n$ matrix g such that $\rho_1(v) = g\rho_2(v)g^{-1}$ for all $v \in \mathcal{L}$.

Equivalence of representations is similar to, but not the same as, conjugacy of matrix Lie algebras. Given two representations $\rho_1, \rho_2 \colon \mathcal{L} \to M_n$, their images $\mathcal{L}_i := \operatorname{im}(\rho_i)$ are matrix Lie algebras. The representations ρ_i are equivalent if they are conjugate *as maps*, whereas the matrix Lie algebras forget the maps and only care about their images. In fact, Lemma 4.4.3 shows that $\mathcal{L}_1$ and $\mathcal{L}_2$ are conjugate matrix Lie algebras if and only if ρ_1 and ρ_2 are equivalent up to an automorphism of $\mathcal{L}$, that is, ρ_1 is equivalent to $\rho_2 \circ \alpha$ for some automorphism $\alpha \colon \mathcal{L} \to \mathcal{L}$. These automorphisms are what cause all the computational difficulties, and allow the equivalences with GRAPH ISOMORPHISM and CODE EQUIVALENCE.

Given two representations $\rho_i \colon \mathcal{L} \to M_{n_i}$ for $i = 1, 2$, their *direct sum* $\rho_1 \oplus \rho_2 \colon \mathcal{L} \to M_{n_1+n_2}$ is defined by the block-matrix:

$$(\rho_1 \oplus \rho_2)(v) = \begin{pmatrix} \rho_1(v) & 0 \\ 0 & \rho_2(v) \end{pmatrix}.$$

A representation is called *decomposable* if it is (equivalent to) a non-trivial direct sum; otherwise it is called *indecomposable*.

The set M_n of $n \times n$ matrices acts on the vector space $\mathbb{F}^n$ by the usual matrix-vector multiplication. Given a subset $S \subseteq M_n$, if $V \subseteq \mathbb{F}^n$ is a subspace such that $S \cdot V \subseteq V$, then V is called an *S-invariant subspace*. The 0 subspace and the whole space $\mathbb{F}^n$ are S-invariant for any S.

A representation $\rho \colon \mathcal{L} \to M_n$ is called *irreducible* if 0 and $\mathbb{F}^n$ are the only $\mathrm{im}(\rho)$-invariant subspaces. Otherwise a representation is called *reducible*. Note that a decomposable representation is reducible, but the converse need not be true, as illustrated by the example:

$$\left\{ \begin{pmatrix} 1 & x \\ 0 & 1 \end{pmatrix} : x \in \mathbb{F} \right\}.$$

A representation is *completely reducible* if it can be decomposed into a direct sum of irreducible representations. Every representation can be decomposed into indecomposable representations; in a completely reducible representation these indecomposables must also be irreducible.

A matrix Lie algebra $\mathcal{L} \subseteq M_n$, can be viewed as the image of a faithful representation of $\mathcal{L}$, namely, take $\rho \colon \mathcal{L} \to M_n$ to be the inclusion (i. e., identity) map. Via this identification, we also apply the terms (in)decomposable and (ir)reducible to matrix Lie algebras. If $\mathcal{L}$ is a completely reducible matrix Lie algebra, then it is equivalent (conjugate) to a matrix Lie algebra consisting of block-diagonal matrices, where the restriction to each block is irreducible.

Theorem 2.2.8 (see Theorem III.10 on p. 81 of Jacobson [150])**.** *A matrix Lie algebra $\mathcal{L}$ over a field of characteristic zero is completely reducible if and only if $\mathcal{L}$ is isomorphic to the direct sum of an abelian, diagonalizable Lie algebra and a semisimple Lie algebra.*

Inner and Outer Automorphisms

The collection of automorphisms of a Lie algebra $\mathcal{L}$ form a group $\mathrm{Aut}(\mathcal{L})$ under composition of maps. Given a Lie algebra $\mathcal{L}$ and $v \in \mathcal{L}$, the Jacobi identity implies that the map $\mathrm{ad}_v \colon \mathcal{L} \to \mathcal{L}$ defined by $\mathrm{ad}_v(u) := [v, u]$ is a homomorphism of Lie algebras. If $\mathrm{ad}_v^k := \mathrm{ad}_v \circ \cdots \circ \mathrm{ad}_v$ is the zero map for k sufficiently large, then $\exp(\mathrm{ad}_v) := I + \mathrm{ad}_v + \frac{1}{2}\mathrm{ad}_v^2 + \cdots + \frac{1}{(k-1)!}\mathrm{ad}_v^{k-1}$ is an automorphism of $\mathcal{L}$ in characteristic zero (since we need to divide by $k-1$). Automorphisms arising in this way are called *inner automorphisms*. The inner automorphisms form a normal subgroup $\mathrm{Inn}(\mathcal{L}) \leq \mathrm{Aut}(\mathcal{L})$. The quotient group $\mathrm{Aut}(\mathcal{L})/\mathrm{Inn}(\mathcal{L})$ is called the *outer automorphism* group and is denoted $\mathrm{Out}(\mathcal{L})$.

The outer automorphism groups of the simple Lie algebras are completely known:

$$
\begin{array}{rrcl@{\qquad}rrcl}
 & \mathrm{Out}(\mathfrak{sl}_n) & = & S_2 & & \mathrm{Out}(\mathfrak{sp}_{2n}) & = & 1 \\
(n \neq 4) & \mathrm{Out}(\mathfrak{so}_{2n}) & = & S_2 & & \mathrm{Out}(\mathfrak{so}_{2n+1}) & = & 1 \\
 & \mathrm{Out}(\mathfrak{so}_8) & = & S_3 & & \mathrm{Out}(\mathfrak{e}_7) & = & 1 \\
 & \mathrm{Out}(\mathfrak{e}_6) & = & S_2 & & \mathrm{Out}(\mathfrak{e}_8) & = & 1 \\
 & & & & & \mathrm{Out}(\mathfrak{f}_4) & = & 1 \\
 & & & & & \mathrm{Out}(\mathfrak{g}_2) & = & 1
\end{array}
$$

The action of $\mathrm{Out}(\mathfrak{sl}_2)$ on the representations of $\mathfrak{sl}_2$ is trivial, despite the fact that $\mathrm{Out}(\mathfrak{sl}_2) \cong S_2$. The action technically takes a representation to its dual, but for $\mathfrak{sl}_2$, the dual of a representation is equivalent to that representation.

Twisting representations by automorphisms

Given an automorphism $\alpha \colon \mathcal{L} \to \mathcal{L}$ and a representation $\rho \colon \mathcal{L} \to M_n$, we get another representation $\rho \circ \alpha \colon \mathcal{L} \xrightarrow{\alpha} \mathcal{L} \xrightarrow{\rho} M_n$, given by $(\rho \circ \alpha)(v) = \rho(\alpha(v))$. Since α is an automorphism, it is, in particular, onto, so $\mathrm{im}(\rho \circ \alpha) = \mathrm{im}(\rho)$. However, $\rho \circ \alpha$ and ρ need not be equivalent as representations, despite having the same image. We call $\rho \circ \alpha$ the *twist* of the representation ρ by the automorphism α.

In many cases, twisting by inner automorphisms in fact leads to equivalent representations. For example, abelian Lie algebras have no inner automorphisms, since $\mathrm{ad}\, v = 0$ for all v. Additionally, when the characteristic is zero, twisting a representation of a semisimple

Lie algebra by an inner automorphism leads to an equivalent representation (see De Graaf [97, Lemma 8.5.1.]). In particular, direct sums of abelian and semisimple Lie algebras have this property, including all completely reducible algebras in characteristic zero.

In these cases, we find that the outer automorphism group $\mathrm{Out}(\mathcal{L})$ acts on the set of representations-up-to-equivalence. If $\alpha \in \mathrm{Out}(\mathcal{L})$, we denote the image of ρ under the action of α by ρ^α. Equivalently, let $\alpha_* \in \mathrm{Aut}(\mathcal{L})$ be a representative of $\alpha \in \mathrm{Out}(\mathcal{L})$; then ρ^α is the equivalence class of $\rho \circ \alpha_*$; this equivalence class is independent of the choice of representative α_*, since twisting by inner automorphisms in this situations does not change ρ up to equivalence.

Tensor products of representations

Given an $n \times m$ matrix A and $p \times q$ matrix B, their *tensor, or Kronecker, product* $A \otimes B$ has size $np \times mq$, and its entries are all possible products of an entry from A and an entry from B. If we treat the rows of $A \otimes B$ as indexed by $[n] \times [p]$ and similarly its columns indexed by $[m] \times [q]$, then we have

$$(A \otimes B)_{(r_A, r_B), (c_A, c_B)} = A_{r_A, c_A} B_{r_B, c_B}.$$

As with the direct sum, we may extend this operation from single matrices to representations. For associative algebras this is straightforward (the matrix of a tensor product of representations is the tensor product of the representations); in order for the tensor product of representations of a Lie algebra to still be a representation of the Lie algebra, we must instead use the "derivative" of the most obvious (non-)definition, which we now describe.

If $\rho_1 \colon \mathcal{L} \to M_{n_1}$ and $\rho_2 \colon \mathcal{L} \to M_{n_2}$ are two representations of a Lie algebra $\mathcal{L}$, then their *(internal) tensor product* $\rho_1 \otimes \rho_2 \colon \mathcal{L} \to M_{n_1 n_2}$ is defined by

$$(\rho_1 \otimes \rho_2)(x) = \rho_1(x) \otimes I_{n_2} + I_{n_1} \otimes \rho_2(x),$$

where I is the identity matrix. It is easily verified that with this definition, the tensor product of representations of $\mathcal{L}$ is again a representation of $\mathcal{L}$.

Finally, if $\rho_i \colon \mathcal{L}_i \to M_{n_i}$ are two representations of two Lie algebras $\mathcal{L}_1, \mathcal{L}_2$, then their *(external) tensor product* $\rho_1 \otimes \rho_2 \colon \mathcal{L}_1 \oplus \mathcal{L}_2 \to M_{n_1 n_2}$ is defined by

$$(\rho_1 \otimes \rho_2)(x_1 + x_2) = \rho_1(x_1) \otimes I_{n_2} + I_{n_1} \otimes \rho_2(x_2)$$

where $x_i \in \mathcal{L}_i$ for each $i = 1, 2$. For future reference we record the following standard proposition.

Proposition 2.2.9. *Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be two semisimple Lie algebras over any algebraically closed field of characteristic zero. Then the irreducible representations of the direct sum $\mathcal{L}_1 \oplus \mathcal{L}_2$ are exactly the (external) tensor products of the irreducible representations of $\mathcal{L}_1$ with the irreducible representations of $\mathcal{L}_2$.*

Proof. This follows from the complete reducibility of representations of semisimple Lie algebras and their weight theory—as in any of the standard references mentioned at the beginning of this section—and the discussion of tensor products of weight spaces as in Chapter III of Jacobson [150]. □

CHAPTER 3
A TUTORIAL AND SURVEY OF GEOMETRIC COMPLEXITY THEORY

This chapter is intended to be a self-contained introduction to the Geometric Complexity Theory program (initiated in 2001 by Mulmuley and Sohoni [207]). The only prerequisites are some familiarity with complexity theory and a very basic familiarity with groups, rings, and vector spaces, as in Chapter 2; all other material will be covered as needed.

3.1 Introduction

Geometric Complexity Theory (GCT) was developed by Ketan Mulmuley and Milind Sohoni [207] as an approach to fundamental lower bounds questions in complexity theory, such as **P** versus **NP**, using algebraic geometry and representation theory. The GCT program has attracted an increasing amount of attention over the last several years, and researchers have started making progress on some of the subproblems of and problems related to GCT.

Despite the deep and beautiful mathematics behind GCT, we believe it is possible to understand the structure of the GCT program, the flavor of the mathematics involved, and what it may tell us about complexity without first learning algebraic geometry and representation theory at the research level. We aim to present this material while still conveying some of the technical aspects of GCT, which necessitates establishing some mathematical terminology along the way. This mathematics is established in clearly marked "Background" sections. For readers already familiar with the necessary mathematics, we hope the remainder of this chapter serves as a useful overview.

Given the pace of progress and the large quantity of sometimes-overlapping papers on GCT, finding where to start, identifying the key ideas, and grabbing hold of the narrative of GCT can be a daunting task, especially in the face of the required background knowledge of complexity theory on the one hand and algebraic geometry and representation theory on the

other hand. The goal of this survey is to provide a single source with a consistent narrative that highlights the main points of GCT.

While we would like to include only proven facts and crisp conjectures, we feel it is important to also sometimes include philosophical or more heuristic motivation. GCT is a *program* towards lower bounds; very few, if any, researchers believe that super-polynomial lower bounds will come out of GCT any time soon. As such, it is very difficult if not impossible to tell only from proven facts what the way forward is. As always in mathematics, philosophical justification and analogies are important tools that may help guide us.

This survey is organized in a perhaps slightly unusual manner. Each section views GCT from a different "height": first we give the 1,000-foot view, then the 100-foot view, then the 10-foot view, and finally the view from the ground. It is our hope that readers may descend as far as they like towards the details and still gain profitably from the reading. Additional background sections are included as we get further into the details, though we remark that for this survey none of the background is particularly onerous to anyone who has at least a passing familiarity with groups, vector spaces, and polynomials. Current or very recent results related to GCT are included where appropriate.

3.1.1 Outline

From 1,000 feet, we can see the general plan of attack of GCT, and are able to discuss to what extent algebraic geometry and representation theory may be necessary for lower bounds. From 100 feet, we can see the complete details of the translation from complexity theory to algebraic geometry, though even at that level we do not actually need any notions from algebraic geometry! At such a close distance we are also able to distinguish between algebraic and Boolean complexity, and we discuss the relationship between the two and how GCT proposes to move from its current algebraic setting to the final Boolean setting. From 10 feet away we can really see the details of how symmetry will be used, and discussion the fundamental phenomenon of *characterization by symmetries*. We show how, if symmetry-characterization plays as crucial a role in GCT as is hypothesized, then symmetry-characterization alone is enough to avoid the Razborov–Rudich natural proofs barrier. We also show how symmetry-characterization is already known to have an effect on complexity: Kayal [159] has shown that a certain problem on polynomials that is **NP**-hard in general

can be done in randomized polynomial time for the permanent and determinant, by taking advantage of their symmetry-characterization. It is this algorithm that we generalize in our study of MATRIX ISOMORPHISM OF LIE ALGEBRAS in Chapter 4.

Finally, once we're on the ground we can see more about the (algebraic) geometry of the algebraic varieties arising in GCT, even still without actually using really any algebraic geometry. We discuss essentially the only known example of a function known to lie in the boundary of the orbit closure of determinant [177], and show that from the complexity point of view that problem is essentially equivalent to the determinant. We also show that the Mulmuley–Sohoni Conjecture 3.3.4, which is *a priori* a strengthening of the original permanent versus determinant conjecture, is formally much closer to the original permanent versus determinant conjecture than it appears on the surface. These latter two results, while not difficult, have not appeared before.

We also discuss a concrete connection between the varieties arising in GCT and the down-to-earth linear algebra problem of understanding the linear subspaces sitting inside the collection of matrices with zero determinant (or permanent). This is a very concrete problem which, although it has been studied sporadically in the last 100 or so years by the techniques of linear algebra and algebraic geometry, does not necessarily *need* algebraic geometry for its study, and may be an approachable and enjoyable problem for some complexity theorists to try their hand at.

3.2 The 1,000-foot view

Here we outline the basic GCT plan of attack. Although the machinery of GCT is currently most well-developed in the algebraic setting, from 1,000 feet up we can barely see the distinction between algebraic and Boolean complexity; we will return to this issue in more detail Section 3.3.3. The only prerequisite for this section is some familiarity with the landscape of complexity.

3.2.1 The plan of attack

There are two major phases to the GCT approach: the first, which was already completed in the first paper on GCT [207], is to translate complexity questions to questions in algebraic

geometry and representation theory. This step already suggests the exciting possibility of new lower bounds methods, incorporating techniques and tools from algebra, algebraic and differential geometry, representation theory, and algebraic combinatorics.

The second phase is a suggestion of how to go about resolving the representation-theoretic conjectures that arise from the first phase. To describe the approach without going into representation theory, we use an analogy. The very high-level view from this analogy is as follows—be warned that this is such a high level as to make it almost sound naive, but we hope it helps ground the more detailed plan of GCT. To prove that $\mathbf{NP} \not\subseteq \mathbf{P/poly}$, first construct a nontrivial algorithm A such that $A(0^n)$ computes the circuit-size complexity of SAT up to length n, which we denote $\mathsf{SAT}_{\leq n}$, then by analyzing the properties of A, show that $A(0^n)$ grows faster than any polynomial in n. Rather than applying this idea to **NP** versus **P/poly** directly, an analogous strategy is applied to various problems in representation theory. Of course GCT goes much deeper than this, and the depth and details are what turn this naive-sounding plan into something that could rightfully be called a *program* towards lower bounds. We will spend the rest of the chapter on these details.

We emphasize here the role of the *algorithm* and its *nontriviality*. Continuing with the above analogy to **NP** versus **P/poly**, if instead of an algorithm we had merely asked for a function, then this would be nothing more than a restatement of $\mathbf{NP} \not\subseteq \mathbf{P/poly}$, which is what we set out to prove in the first place. And if we omit the word "nontrivial," then there is an obvious brute-force algorithm A that computes the circuit-size complexity of $\mathsf{SAT}_{\leq n}$ as above. However, the brute-force nature of such an algorithm suggests that analyzing it is, again, little more than analyzing a thinly veiled reformulation of the original problem. We need the algorithm A to reflect deeper properties of **NP** and circuit-size than merely the fact that circuit-size of a function on a finite domain is finite[1].

Instead of finding an algorithm which computes circuit-size directly, GCT is currently aimed at finding an algorithm that will produce witnesses that $\mathsf{SAT}_{\leq n}$ is not computed by circuits of size n^k. Finding such an algorithm is currently broken down into three steps:

1. Translate complexity questions to questions in algebraic geometry;

1. One formalization of this need is the relativization barrier (see Section 2.1.6) since both the **NP**-completeness of SAT and the brute-force algorithm for computing circuit-size complexity of $\mathsf{SAT}_{\leq n}$ relativize.

2. Translate algebro-geometric questions to questions in representation theory;

3. Find an algorithm that computes the answers to the representation-theoretic questions, and then by analyzing this algorithm finally resolve the representation-theoretic questions.

The first step was achieved by Mulmuley and Sohoni [207]. The second step is classical mathematics, applied to the algebro-geometric questions raised by Mulmuley and Sohoni. The third step is where the work remains.

We will talk more about the details of these steps in a moment, but even at this level of description we can provide some commentary.

3.2.2 On the necessity of algebraic geometry, representation theory, and algorithms

For questions of algebraic complexity, such as permanent versus determinant, the translation to algebraic geometry is seamless and provides necessary and sufficient conditions for the complexity-theoretic conjectures to hold. It may of course be possible to resolve these algebro-geometric questions by non-algebro-geometric methods, such as algebraic or differential topology, or even possibly combinatorics, but to resolve the complexity questions is completely equivalent to resolving the algebro-geometric questions.

When we move from algebraic complexity to Boolean complexity, the necessity of algebraic geometry is a more subtle question, which we return to in Section 3.3.3.

In the setting of algebraic complexity, though, it may be possible to resolve complexity questions by resolving algebro-geometric questions alone, without recourse to representation theory. However, there is some formal evidence that representation theory may indeed be necessary for these particular algebro-geometric questions.

The way in which Mulmuley and Sohoni propose to use representation theory can be used to show a separation of complexity classes, but we do not know how—and indeed it may not be possible—to use the representation theory to show an inclusion instead. The GCT program in its narrowest sense, which attempts to use representation theory, thus seems aimed at resolving lower bounds questions only in one direction. However, because of the equivalence of complexity questions and certain questions in algebraic geometry, the

study of these algebraic varieties—which might rightfully be called geometric complexity theory (rather than the Geometric Complexity Theory Program)—does allow the possibility of resolving complexity questions in either direction.

It may also be possible to resolve the representation-theoretic questions directly, without finding an algorithm for them and then analyzing it. I view the suggestion to look for an algorithm more as a way to guide our thinking than as a necessary proof technique. I think searching for algorithms may be a useful guide for several reasons: viewing a problem algorithmically provides an additional concrete target to aim for, beyond simply gaining understanding until we can solve the problem completely. Additionally, that concrete target can serve as a useful intermediate goal; finding a nontrivial algorithm can often be a good indicator of progress on the mathematical problem, even before the problem is completely resolved.

Finding nontrivial algorithms for a mathematical problem often requires understanding the purely mathematical structure of the problem better. Put another way, mathematical properties that can be exploited algorithmically can often also be exploited mathematically; by searching for better algorithms, we thus help focus our attention on properties that may ultimately be useful to prove the sought-after conjectures.

Furthermore, finding an algorithm for these questions can be viewed as an attempt to reverse-engineer the structure of an analogous but more well-understood representation-theoretic problem, the Littlewood–Richardson problem. For the Little-wood–Richardson problem, its mathematical structure was discovered first, and then later that structure was exploited algorithmically—in particular, to show that a certain integer linear programming problem associated with the Littlewood–Richardson problem could in fact be solved by its rational relaxation. The hope is that there is analogous, but most likely more difficult, structure in the representation-theoretic problems arising in GCT, and that by first finding an algorithm we will be putting ourselves on the right track.

Finally, as a sanity check we may ask whether such an algorithm should exist at all. If we ask specifically about a polynomial-time algorithm for the representation-theoretic problems arising in GCT, the answer is unclear. However, one consequence of such an algorithm is known to hold, assuming that the complexity-theoretic conjectures we are attempting to prove in fact hold:

Theorem 3.2.1 (Mulmuley [206]). *If the permanent of an $n \times n$ matrix over a field $\mathbb{F}$ cannot be computed by* poly(n)*-size arithmetic circuits over $\mathbb{F}$, then there is a probabilistic polynomial-time algorithm A over $\mathbb{F}$ that outputs a single set of counterexamples against* all *polynomial-size circuits. In other words, $A(0^n)$ outputs a set of matrices $\{M_1, \ldots, M_k\}$ that, with high probability over the randomness of A, if C is any arithmetic circuit over $\mathbb{F}$ of polynomial size, then there is some i such that $C(M_i) \neq \text{perm}(M_i)$.*

Furthermore, if arithmetic circuit identity testing can be black-box derandomized, then A can be made to run in deterministic polynomial time.

A similar theorem holds for an analogous problem capturing **NP** that we discuss in Section 3.3.3. This can be taken as evidence that it is at least plausible that polynomial-time algorithms exist for the representation-theoretic questions arising in GCT. Even from a more skeptical point of view, had this theorem not held it would have been a strong indication that efficient algorithms for the relevant representation theory problems should also not exist.

The current fastest algorithms for these representation-theoretic problem use the natural reduction to Gröbner bases. Computing Gröbner bases is in general **EXPSPACE**-complete [193, 195] (see [194] for a survey)—that is, complete for exponential *space*, whose best known time bound is *doubly* exponential $2^{2^{O(n^k)}}$—so **EXPSPACE** is the current best upper bound for the complexity of these representation-theoretic problems. A polynomial-time algorithm for the representation-theoretic problems—which, themselves, we do not expect to be **EXPSPACE**-complete—would certainly be nontrivial. Indeed, one of the motivations for the original definitions of polynomial-time was to be able to formally show that some algorithm is "better" than brute force.

Mulmuley refers to the above result as the "Flip Theorem" [206]; its proof for the permanent is closely related to downward self-reducibility, and a generalization that has arisen in GCT called "characterization by symmetries," which allows a flip theorem to be proven for **NP** as well. We discuss characterization by symmetries in Section 3.4.

In summary, there is some evidence that the representation-theoretic techniques suggested by Mulmuley and Sohoni may be necessary, there is some evidence that nontrivial, polynomial-time algorithms for these representation-theoretic problems should exist, and finding such algorithms, although not strictly necessary, may be a good guide for progress

in the area. As with any guide, we should make sure that the search for algorithms does not blind us to other possibilities for fruitful angles of attack, but it may nonetheless be productive to follow the guide while we can.

3.2.3 The plan of attack II: a few details

Here we give some of the details of the above steps are implemented. Without actually delving into algebraic geometry and representation theory, we still hope to give the general flavor of these translations. In Section 3.3 we give the full details of the translation from complexity theory to algebraic geometry—still without much need for any actual algebraic geometry.

Step 1: translate complexity questions to algebraic geometry. Suppose we wish to show that $\mathbf{P} \neq \mathbf{NP}$. The GCT approach constructs, for each complexity class, and for each input length n, a set of points in some high-dimensional vector space—typically the dimension is exponential in the input size—which happens to be an algebraic variety. We will refer to these sets as varieties, but for now the technical definition of algebraic variety is irrelevant. Call these varieties $X_{\mathbf{P},n}$ and $Y_{\mathbf{NP},n}$. These varieties will be constructed in such a way that $\mathbf{NP} \subseteq \mathbf{P}$ if and only if $Y_{\mathbf{NP},n} \subseteq X_{\mathbf{P},n^k}$ for some k and all sufficiently large n. In a very real sense, the points of these varieties correspond to other functions in their respective complexity classes. In Section 3.3 we will see what this means in more detail. It is important to note that these varieties are constructed in such a way that an inclusion of the varieties is both necessary and sufficient for an inclusion between the complexity classes, so an (algebraic) complexity question is *completely equivalent* to an algebraic geometry question.

Step 2: translate algebro-geometric questions to representation theory. The natural way in which these varieties are constructed leads to the varieties being symmetric under an action of the general linear group GL_n of all $n \times n$ invertible (complex) matrices. Because of this large group of symmetries, tools from group theory—most notably representation theory—can be applied to help understand these varieties. Although the representation theory does not obviously capture all the structure of these varieties and their inclusion relationships, there is a purely representation-theoretic statement which implies, for example, that $Y_{\mathbf{NP},n^k}$ is not contained in $X_{\mathbf{P},n}$ for each k and infinitely many n, and hence that $\mathbf{P} \neq \mathbf{NP}$ (in an algebraic setting, see Section 3.3.3).

3.3 The 100-foot view: from computational reductions to orbit closures

In this section we construct the algebraic varieties associated to the permanent (i.e. #**P** or **VNP**) and determinant (i.e. **GapL**, $\mathbf{VP}_{ws}$, or **VQP**). Although the sets we are constructing are indeed algebraic varieties—and hence algebraic geometry is indispensable for their study—this section can be completely understood without any algebraic geometry. The only additional prerequisite for this section, beyond that required for the 1,000-foot view, is a basic familiarity with group actions with the notion of the closure of a subset of $\mathbb{C}^n$. We give a brief review in the first subsection.

3.3.1 Background: group actions and orbits

Groups very often arise as the symmetries of some other mathematical object, such as a set, relation, graph, topological space, vector space, or other algebra (including other groups!), in which case we say that the group *acts* on the set (graph, vector space, etc.). In fact, this is their *raison d'être*; historically groups first arose this way, and it took nearly a century before the abstract notion of a group as a set with a binary operation satisfying certain axioms was put forth (see, e. g. [278]).

For example, consider the symmetry group of an equilateral triangle: this consists of all rigid motions of the plane—generated by rotations, translations, and reflections—which send the triangle to itself. There are six such motions: the identity, which we denote 1, rotation by $\pi/3$ around the center of the triangle, rotation by $2\pi/3$, and reflections over any of the three main axes of the triangle. Let's denote rotation by $\pi/3$ by ρ and reflection over the y-axis by σ. As an abstract group, this is simply the dihedral group of order 6, which can be expressed as the group generated by two formal elements, r and s, subject to the constraints $r^3 = s^2 = 1$ and $srs = r^{-1}$. When we want this abstract group to act on the triangle, we must specify, for each group element, how it acts: we must associate to r the rotation ρ by $\pi/3$, and to s the reflection σ.

Here is another example, perhaps more familiar to complexity theorists: let $\mathcal{G}_n$ denote the set of all graphs on n vertices labeled $1, \ldots, n$. Any permutation of $\{1, \ldots, n\}$ induces a permutation on the set of graphs, by relabeling vertices. For example, let G_1 be the

undirected graph on 3 vertices with a single edge $\{1, 2\}$ (see Figure 3.1); let G_2 be the undirected graph on three vertices with a single edge $\{2, 3\}$. These are distinct elements of $\mathcal{G}_3$; the permutation $1 \mapsto 2 \mapsto 3 \mapsto 1$, denoted (123), sends G_1 to G_2. Although G_1 and G_2 are distinct labeled graphs, they are clearly isomorphic, and (123) is one isomorphism between them. The symmetric group S_n thus acts on $\mathcal{G}_n$ by taking a graph to an isomorphic, but possibly distinct, copy.

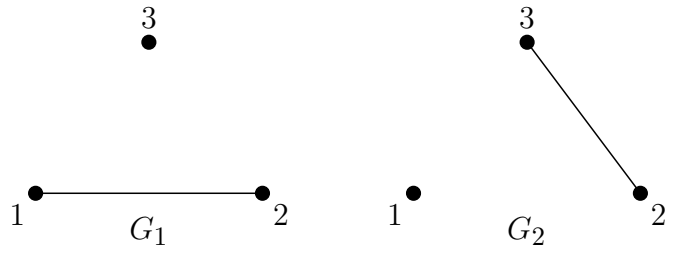


Figure 3.1: The action of S_n on n-vertex graphs is by isomorphisms

An action of a group G on a set (graph, vector space, etc.) X generalizes and formalizes the above examples: an *action* is a group homomorphism $\alpha\colon G \to \mathrm{Aut}(X)$, where by $\mathrm{Aut}(X)$ we mean the automorphism group of X, whatever type of structure X may be. For example, if X is simply a set, then we may take $\mathrm{Aut}(X)$ to be the group of all permutations of X, sometimes denoted $\mathrm{Sym}(X)$ and referred to as "the symmetric group on X." If X is a graph, then $\mathrm{Aut}(X)$ consists of the graph automorphisms of X; if X is a group, then $\mathrm{Aut}(X)$ consists of the group automorphisms of X. If X is a vector space, then $\mathrm{Aut}(X)$ consists of the invertible linear maps from X to X, sometimes denoted $\mathrm{GL}(X)$ and referred to as "the general linear group on X."

If $\alpha\colon G \to \mathrm{Aut}(X)$ is an action of a group G on a set (graph, vector space, etc.) X, and α is understood from context, then we may write $g(x)$, thinking of g as an automorphism on X via α, or more simply $g \cdot x$ or gx.

The fact that α is a homomorphism is equivalent to the following two conditions: 1) the identity acts trivially—$1x = x$ for all $x \in X$—and 2) the action of a product gh is the composition of the actions of g and h—$(gh) \cdot x = g \cdot (h \cdot x)$ for all $g, h \in G$ and $x \in X$. Some authors present group actions as maps $G \times X \to X$ satisfying axioms corresponding

to conditions (1) and (2); the two viewpoints are equivalent, and often used interchangeably when convenient.

As another example, let $G = \{1, \tau\}$ be the cyclic group of order two, with $\tau^2 = 1$, and let $X = \{a, b, c\}$ be a set with three elements. There are $\binom{3}{2} = 3$ distinct ways G can act nontrivially on X. Since the identity must act trivially, we need only specify the action of τ. One action is given by $\tau \mapsto (ab)$, that is, $\tau(a) = b$ and $\tau(b) = a$. Another action is given by $\tau \mapsto (bc)$, and yet another is given by $\tau \mapsto (ac)$. Finally, if τ acts trivially as well, fixing each element of X, then all of G acts trivially, and the action is called "trivial."

One of the most important concepts in group actions is that of the orbit of an element of the set X being acted on. The *orbit* of $x \in X$ is the set of all images of x under all group elements; it is often denoted $G \cdot x := \{g \cdot x \colon g \in G\}$. In the previous example, when $\tau \mapsto (ab)$, the orbit of a is $\{a, b\}$, the orbit of b is $\{a, b\}$, and the orbit of c is the singleton $\{c\}$. In the example of the equilateral triangle above, the orbit of a corner of the triangle consists of all three corners of the triangle; even if we only used rotations this would be the case. The orbit of the center of an edge of the triangle consists of the centers of all three edges. The orbit of a point on an edge that is neither a corner nor the center of the edge consists of six points (see Figure 3.2).

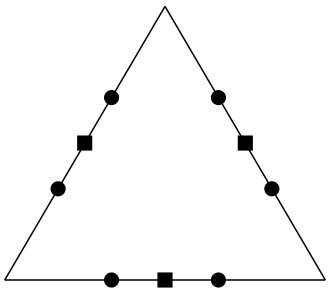


Figure 3.2: Orbits of points on an equilateral triangle under the action of the dihedral group. Each shape (square or circle) corresponds to a single orbit.

In the example of S_n acting on the set of all graphs on n vertices, the orbit of a single graph consists of all graphs isomorphic to it.

One feature to note from these examples is that the relation of "being in the same orbit" is an equivalence relation; in the case of S_n acting on $\mathcal{G}_n$ this is the familiar fact that isomorphism is an equivalence relation. This is a general feature of the orbits of group actions:

Proposition 3.3.1. *For any group action on a set X, the relation of "being in the same orbit" is an equivalence relation on X.*

Proof. Suppose a group G acts on a set X. First, every point lies in its own orbit: $x \in Gx$, since G has an identity element and the identity must act trivially. Next we show symmetry: if $x \in Gy$ then $x = gy$ for some $g \in G$. But then $g^{-1}x = g^{-1}gy = 1y = y$, so $y \in Gx$. Finally we show transitivity: suppose $x \in Gy$ and $y \in Gz$; then $x = g_1 y$ and $y = g_2 z$ for some $g_1, g_2 \in G$. Then $x = g_1 y = g_1 g_2 z$, so $x \in Gz$. □

We see from the proof of this proposition that the three axioms for groups—identity, inverse, and associativity—exactly mirror the three axioms for equivalence relations—reflexivity, symmetry, and transitivity. This reflects the fact that group actions are a fundamental part of the nature and origin of groups.

One feature we have not yet seen is the idea of an orbit closure. When a finite group acts on a topological space in which each individual point is a closed set, then every orbit is closed, since every orbit is a finite set. But when an arbitrary group acts on a topological space, its orbits need not be closed. For example, consider the group $\mathbb{C}^*$ of nonzero complex numbers with multiplication as the group operation. $\mathbb{C}^*$ acts on $\mathbb{C}$ by complex multiplication. Since $\mathbb{C}$ is a field, the orbit of any nonzero $z \in \mathbb{C}$ consists of all nonzero complex numbers. But the set of all nonzero complex numbers is not a closed subset of $\mathbb{C}$: its closure also includes zero. This is perhaps the simplest example of a non-closed orbit; we will encounter many more such examples in Geometric Complexity Theory, and the ones arising in complexity tend not to be so simple. In fact, as we will see in Sections 3.4 and 3.5, the core of the geometric approach to complexity theory lies in understanding the relationship between certain orbits and their closures.

3.3.2 Equivalence of lower bounds and orbit closure containment

In this section we discuss how questions in algebraic complexity theory translate faithfully to questions regarding orbits and their closures, that is, the questions from the two seemingly different areas are equivalent. In the next section we discuss how this applies to questions of Boolean complexity. We discuss the permanent versus determinant problem as a model problem, though the treatment applies to many other problems.

Here is the high-level dictionary of the translation from classical complexity to orbit closures. The dictionary will be explained in detail throughout the remainder of this section, but we present it here for reference and to give its flavor.

Classical Complexity		**Geometric Complexity**
Function f to be computed	$\leftrightarrow$	point in the space of functions
Equivalent functions $f \sim g$	$\leftrightarrow$	points in the same orbit
Reduction between equivalent functions	$\leftrightarrow$	action of group element
Reduction between arbitrary functions	$\leftrightarrow$	action of *limits* of group elements
$f \leq g$	$\leftrightarrow$	f lies in the orbit closure of g

Valiant's work [269] suggests the permanent versus determinant problem as an algebraic analog of **P** versus **NP**; we discuss the exact nature of this analogy in the next section, but for now it suffices that this is a central open question in algebraic complexity. The permanent versus determinant problem asks whether there is a map from $n \times n$ matrices X to $m \times m$ matrices Y such that the entries of Y are constants or linear combinations of the entries of X, making $\mathrm{perm}(X) = \det(Y)$ and with m polynomially bounded in n. If this holds, we say the permanent is a p-projection of the determinant. By completeness results of Valiant [269] and Malod and Portier [191], the permanent versus determinant question is equivalent to the question of whether $\mathbf{VNP} \subseteq \mathbf{VP}_{ws}$.

We will rephrase the permanent versus determinant question in terms of orbits of a certain group action on a vector space, namely, the vector space of degree m homogeneous polynomials in m^2 variables. The function $\det_m$ is a single point of this vector space. Inconveniently, perm_n is not, since it has lower degree and fewer variables. To remedy this problem we instead consider without loss of generality, as we'll explain shortly, the "padded permanent."

Definition 3.3.2. (Padded[2]) Let $f(X)$ be a homogeneous polynomial of degree n in the variables $X = (x_1, \ldots, x_k)$. Then the *m-padded* version of f is f times a new variable to the $m-n$ power. In particular, the m-padded version of f is homogeneous of degree m, and has the form $z^{m-n}f(X)$, where z is a new variable, independent from those in X. When m is clear from context, we refer to the padded version of f, without specifying m.

If f is any polynomial of degree n, not necessarily homogeneous, then the m-padded version of f is the sum of the m-padded version of the homogeneous components of f. That is, if $f = f_n + f_{n-1} + \cdots + f_0$, where each f_i is either zero or homogeneous of degree i, then the m-padded version of f is $z^{m-n}f_n + z^{m-(n-1)}f_{n-1} + \cdots + z^m f_0$.

Lemma 3.3.3. *Let f be any polynomial of degree n, and let g be a homogeneous polynomial of degree $m \geq n$. Then f is an affine projection of g if and only if the m-padded version of f is a projection of g.*

A polynomial f is an *affine projection* of g if $f(X) = g(A(X))$ where $X = (x_1, \ldots, x_k)$, and the entries of $A(X)$ are affine linear combinations of the entries of X, that is, $A(X)_j$ is of the form $c_j + \sum_i c_{ij}x_i$. This is a natural generalization of the notion of projection defined above. Note that if f and g are both homogeneous, as in the case of permanent and determinant, then f is an affine projection of g if and only if f is a projection of g: any term c_j as above only contributes to terms in $g(A(X))$ of strictly lower degree than $\deg f$, and since f is homogeneous these terms must cancel, so we might as well have left them out in the first place.

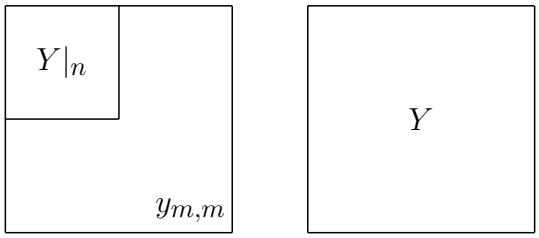
Proof of Lemma 3.3.3. Suppose $f(X) = g(A(X))$, where the entries of $A(X)$ are affine linear combinations of the variables $X = (x_1, \ldots, x_k)$. Let $\tilde{A}(X)$ be the same as $A(X)$, except every constant term is multiplied by the new variable z. Then since g is homogeneous of degree m, and the entries of $\tilde{A}(X)$ are now homogeneous linear combinations of variables, $g(\tilde{A}(X))$

2. This definition is fairly obvious and not particularly deep. We nonetheless give a formal definition because we use the term frequently and want the reader to have an easy reference for its meaning. This term was introduced by Kadish and Landsberg [154]. Prior to their paper, the term "blasted" was coming into use, especially during the Brown-ICERM Workshop on "Mathematical Aspects of **P** vs. **NP** and its Variants," August, 2011 [130]. When applied to the permanent, "blasted" had a double-meaning: the one given to "padding" in the definition here, and an indication of the difficulty or frustration of working with the permanent, as in "those blasted meddling kids!" Nonetheless, we prefer "padded" as it is more descriptive.

is homogeneous of degree m. It can then be verified that $g(\tilde{A}(X)) = z^{m-n} f_n(X) + \cdots + z^m f_0(X)$, where $f_d(X)$ is the homogeneous component of f of degree d, by direct computation.

Conversely, if $z^{m-n} f_n(X) + \cdots + z^m f_0(X)$ is a projection of g, then substituting $z = 1$ shows that $f(X)$ is an affine projection of g. □

When discussing permanent versus determinant we use a specific convention for the variable z used in the padding. There is nothing special in this choice of parameter—any choice of padding parameter works for any function, as long as it is not one of the original variables of the function—but this convention will simplify some of the ensuing discussion. Let Y be an $m \times m$ matrix of variables, and let $Y|_n$ be the $n \times n$ upper-left sub-matrix (see Figure 3.3). Then we take the padded permanent to be $\operatorname{perm}_n(Y|_n)$ times the lower-right variable $y_{m,m}$ to the $m - n$ power. As $y_{m,m}$ is not in the upper-left submatrix $Y|_n$, $y_{m,m}$ is a valid choice for the padding parameter z.



Is $y_{m,m}^{m-n} \operatorname{perm}_n(Y|_n)$ a projection of $\det_m(Y)$?

Figure 3.3: The padded permanent.

We denote the m^2-dimensional vector space of $m \times m$ matrices by $M_m(\mathbb{C})$, and we denote the space of degree m homogeneous polynomials of m^2 variables by $\operatorname{Poly}^m(M_m(\mathbb{C}))$ for mnemonic reasons, the m^2 variables corresponding to the entries of $m \times m$ matrices (in the literature this space is often denoted $\operatorname{Sym}^m(M_m(\mathbb{C})^*)$ or $S^m(M_m(\mathbb{C})^*)$, using notation that is more standard from multilinear algebra and representation theory). Both the determinant and the padded permanent are points of $\operatorname{Poly}^m(M_m(\mathbb{C}))$.

Moreover, by the above argument, the permanent is a p-projection of the determinant if and only if there is a linear map $A\colon M_m(\mathbb{C}) \to M_m(\mathbb{C})$ such that

$$y_{m,m}^{m-n} \operatorname{perm}_n(Y|_n) = \det_m(A(Y)).$$

We denote the space of all linear maps $M_m(\mathbb{C}) \to M_m(\mathbb{C})$ by $\mathrm{End}(M_m(\mathbb{C}))$, the endomorphisms of $M_m(\mathbb{C})$. Although $\mathrm{End}(M_m(\mathbb{C}))$ is not a group, since it contains non-invertible linear maps, it is still closed under composition of maps, and has an action on $\mathrm{Poly}^m(M_m(\mathbb{C}))$; we have already been using this action implicitly: if $A \in \mathrm{End}(M_m(\mathbb{C}))$ and $f(X) \in \mathrm{Poly}^m(M_m(\mathbb{C}))$ then $(A \cdot f)(X) = f(A^T(X))$. Again, although End is not a group, the notion of orbit still makes sense, and we use the same notation as for group actions. Be warned, however, that since End contains non-invertible elements, the End-orbits no longer necessarily form a partition of the space. However, each End-orbit is a union of group orbits, as we discuss below. The permanent is thus a p-projection of the determinant if and only if the padded permanent lies in the endomorphism-orbit of the determinant, or in symbols: $y_{m,m}^{m-n}\,\mathrm{perm}_n(Y|_n) \in \mathrm{End}(M_m(\mathbb{C}))\cdot\det_m(Y)$. Since the endomorphism orbit, as with a group orbit, is closed under multiplication by elements of $\mathrm{End}(M_m(\mathbb{C}))$, the padded permanent is in the endomorphism orbit of determinant if and only if the entire *endomorphism orbit* of the padded permanent is contained in that of the determinant. Thus the permanent versus determinant question is exactly equivalent to this containment question about endomorphism orbits.

Next we move from the endomorphism orbit to the orbit under an actual group, enabling the use of group-theoretic techniques in the study of the permanent versus determinant question. The same operation as above gives an action of the group $\mathrm{GL}(M_m(\mathbb{C}))$—all invertible elements of $\mathrm{End}(M_m(\mathbb{C}))$—on $\mathrm{Poly}^m(M_m(\mathbb{C}))$. Recall that $\mathrm{GL}(M_m(\mathbb{C}))$ is dense in $\mathrm{End}(M_m(\mathbb{C}))$: for any $A \in \mathrm{End}(M_m(\mathbb{C}))$ that is not invertible, there is a sequence $\{A_i\}_{i=1}^{\infty}$ of invertible $m^2 \times m^2$ matrices—elements of $\mathrm{GL}(M_m(\mathbb{C}))$—such that $\lim_{i\to\infty} A_i = A$. As a consequence, the group orbit $\mathrm{GL}(M_m(\mathbb{C})) \cdot \det_m$ is a dense subset of the endomorphism orbit $\mathrm{End}(M_m(\mathbb{C})) \cdot \det_m$. Understanding this group orbit is thus an important first step in understanding the permanent versus determinant problem.

Finally, we come to the principal objects of study in Geometric Complexity Theory: the *closure* of the orbit $\mathrm{GL}(M_m(\mathbb{C})) \cdot \det_m$ in the usual topology on the vector space $\mathrm{Poly}^m(M_m(\mathbb{C}))$. Following notation introduced by Landsberg [176, Chapter 13], we denote this closure by

$$\mathcal{D}et_m := \overline{\mathrm{GL}(M_m(\mathbb{C})) \cdot \det_m} = \overline{\mathrm{End}(M_m(\mathbb{C})) \cdot \det_m}.$$

Since the group orbit is dense in the endomorphism orbit, the endomorphism orbit is con-

tained in $\mathcal{D}et_m$. The orbit closure $\mathcal{D}et_m$ is in fact strictly larger than the endomorphism orbit of $\det_m$—that is, the endomorphism orbit is not a closed subset of $\mathrm{Poly}^m(M_m(\mathbb{C}))$—though at this point in the exposition it is probably not clear why. We discuss techniques for understanding the boundary of $\mathcal{D}et_m$, the points in the orbit closure that are not in the orbit, in Section 3.5.

If we denote $\mathcal{P}erm_m^n$ the orbit closure of the m-padded permanent of $n \times n$ matrices, then one of the conjectures currently focused on in GCT is:

Conjecture 3.3.4 (Mulmuley and Sohoni [207]). *$\mathcal{P}erm_m^n \not\subseteq \mathcal{D}et_m$ when m is polynomially bounded in n. In other words, the padded permanent is not the limit of a sequence of points in the $\mathrm{GL}(M_m(\mathbb{C}))$-orbit of $\det_m$—that is, points linearly equivalent to $\det_m$—when m is polynomial in n.*

This is a strengthening of the original permanent versus determinant conjecture, which, as we have shown above, was about the containment of endomorphism orbits, rather than orbit closures. The notion of the padded permanent being in the orbit closure of determinant, and hence being a limit of functions in the orbit of determinant, is a notion of approximation; although this kind of approximation appears to be different from other kinds studied in complexity theory, it nonetheless provides a complexity-theoretic interpretation of Conjecture 3.3.4.

Every point in the orbit closure is, by definition, a limit of points in the orbit. There is a well-defined sense of "how accurately" a sequence of points in the orbit approaches its limit. In Section 3.5 we make this notion of accuracy precise, and show that if the padded permanent lies in the orbit closure of the determinant, and is approached by its limiting sequence at polynomial accuracy, then the permanent is actually a projection of a polynomially larger determinant. In particular, showing that the permanent does not lie in the "polynomial-accuracy" portion of the orbit closure $\mathcal{D}et$ is equivalent to the original permanent versus determinant conjecture.

The main advantage of using the orbit closures instead of the endomorphism orbits is that the orbit closures are, by construction, closed subsets of $\mathrm{Poly}^m(M_m(\mathbb{C}))$. In particular, they are algebraic varieties, so they may be studied using the powerful techniques of algebraic geometry and representation theory.

A similar situation has long been studied in the context of matrix multiplication. Without going into too much detail, the notion of "tensor rank"—within a constant factor of the number of essential multiplications needed to compute matrix multiplication—roughly corresponds to an orbit containment question, while the notion of "border rank" exactly corresponds to an orbit *closure* containment question. In the history of matrix multiplication orbit closures, in the guise of border rank, have played an important role in both upper and lower bounds (see the survey [175] and references therein). This suggests orbit closures and the correspondent notion of approximation as useful, fundamental measures of complexity.

We now discuss how this construction completes the analogy we mentioned at the outset. Any function that lies in the orbit of the determinant has the same complexity as the determinant, since it only differs from the determinant by a linear change of variables. By the argument earlier, we saw that a function f is a p-projection of the determinant if and only if the padded version of f lies in the endomorphism orbit, or more generally the "polynomial-accuracy" portion of the orbit closure, of the determinant. Thus we see that the group action in GCT plays the role of reductions in classical complexity. The group orbit of the determinant corresponds to functions equivalent to determinant—roughly, to the $\mathbf{VP}_{ws}$-complete functions. The endomorphism orbit corresponds to functions reducible to determinant, as does the "polynomial-accuracy" portion of the orbit closure, and the full orbit closure corresponds to functions approximable by the determinant.

3.3.3 Algebraic versus Boolean complexity

Much of the current work in GCT and most of the currently available expositions we are aware of are about algebraic analogs of **P** versus **NP**, rather than **P** versus **NP** itself. In this section we discuss the nature and utility of these analogies, as well as how GCT can also approach the usual, Boolean **P** versus **NP** problem. One outgrowth of this discussion, mentioned below, suggests why algebraic geometry may be a good toolkit to use in complexity theory, both algebraic and Boolean.

General considerations

Broadly speaking, algebraic complexity is the study of the complexity of algorithmic problems with the restriction that the algorithms treat algebraic objects as single, indivisible units.

For example, arithmetic circuits over a field $\mathbb{F}$—such as the real numbers, complex numbers, or a finite field—are only allowed to use the field operations: addition and multiplication. They cannot "look inside" the field elements and extract their bits for free. Many algorithmic problems of interest in computer science are algebraic in nature, and most natural approaches to such problems are algebraic in nature. Finding lower bounds against algebraic algorithms would tell us that most natural approaches to these problems will not work efficiently. It is also hoped that lower bounds for algebraic algorithms or circuits would give us insight into techniques that could be used for lower bounds in the usual Boolean model of computation.

There are also several more concrete connections between algebraic and Boolean complexity; two of the most prominent connections are:

- Certain algebraic separations are consequences of Boolean ones, so the algebraic separations are natural and necessary first targets. In particular, $\mathbf{P/poly} \neq \mathbf{NP/poly}$ implies $\mathbf{P}_{\mathbb{C}} \neq \mathbf{NP}_{\mathbb{C}}$ in the BSS model [90, Corollary 4], and also, assuming the Generalized Riemann Hypothesis, $\mathbf{VP}_{\mathbb{C}} \neq \mathbf{VNP}_{\mathbb{C}}$ in Valiant's theory [69, Chapter 4]. Bürgisser [69] also showed that, without GRH, $\mathbf{NC/poly} \neq \mathbf{P/poly}$ implies $\mathbf{VP}_{\mathbb{C}} \neq \mathbf{VNP}_{\mathbb{C}}$.

- A celebrated result of Kabanets and Impagliazzo [153] shows that the following three facts cannot be simultaneously true: arithmetic circuit identity testing is in $\mathbf{P}$, $\mathbf{NEXP} \subseteq \mathbf{P/poly}$, and the permanent can be computed by polynomial-size arithmetic circuits. Typically this is phrased more positively, in that derandomizing efficient algorithms for arithmetic circuit identity testing implies the Boolean separation $\mathbf{NEXP} \not\subseteq \mathbf{P/poly}$ or the algebraic separation that the permanent does not have polynomial-size arithmetic circuits.

The permanent versus determinant problem, or equivalently the $\mathbf{VNP}$ versus $\mathbf{VP}_{ws}$ problem, is often suggested as an algebraic analog of $\mathbf{P}$ versus $\mathbf{NP}$. This analogy is strengthened by Malod and Portier's [191] characterization of the determinant as complete for $\mathbf{VP}_{ws}$—polynomial-size, polynomial-degree, weakly-skew arithmetic circuits—and Venkateswaran's characterization of $\mathbf{P}$ by skew circuits [272].

However, the permanent of integer matrices is complete for the counting class $\#\mathbf{P}$, which is at least as hard as the entire polynomial hierarchy [261], and hence is thought to be much

harder than **NP**. Also, the determinant of integer matrices is **GapL**-complete [273, 93, 262][3], hence lies in $\mathbf{NC}^2$, which is thought to be much smaller than all of **P**. In this sense, the permanent versus determinant problem is closer to $\mathbf{NC}^2$ versus **PH** or **GapL** versus **GapP**—a counting analog of **NL** versus **NP**—than **P** versus **NP**. Since $\mathbf{GapL} \subseteq \mathbf{FNC}^2 \subseteq \mathbf{FP} \leq \mathbf{NP} \subseteq \mathbf{PH} \leq \mathbf{GapP}$, the permanent versus determinant conjecture is a natural and necessary consequence of $\mathbf{P} \neq \mathbf{NP}$, and so may be a good stepping stone to **P** versus **NP**, but does not seem to tightly capture **P** versus **NP** itself.

P *versus* **NP** *in Geometric Complexity Theory*

However, GCT may provide more than stepping stones and analogies in the move from algebraic to Boolean complexity. First, instead of permanent and determinant, Mulmuley and Sohoni [207] construct functions that more closely capture **P** and **NP**, in that separating these functions (over finite fields) would show $\mathbf{NP} \not\subseteq \mathbf{P/poly}$.

Second, the algebraic nature of algebraic geometry means that its techniques often transfer from topological fields like $\mathbb{R}$ and $\mathbb{C}$ to arbitrary, and in particular finite, fields. Many known results in algebraic complexity use heavily the topological nature of $\mathbb{R}$ and $\mathbb{C}$, for example, seminal lower bounds on sorting in the algebraic decision tree model [253, 48], and lower bounds on approximating roots of polynomials in the BSS model [248, 197, 198, 199]. One distinguishing feature of GCT is that it is attempting to use more purely algebraic techniques, which have a better chance of transferring to finite fields and hence to the Boolean case.

Even though the techniques of algebraic geometry more easily transfer from $\mathbb{C}$ to finite fields, the representation theory involved admittedly becomes more complicated and less well-understood. In a still-unpublished manuscript, Mulmuley [205] shows which mathematical properties of the algebraic varieties and which techniques should be useful over finite fields, beyond just representation theory.

We now describe the functions Mulmuley and Sohoni construct [207] for **P/poly** and **NP**. These functions are often left out of expositions of GCT not because they are particularly

3. The typical sources cited for this can be difficult to acquire. Other proofs of **GapL**-completeness of the determinant appear in Bürgisser [69, Chapter 2] and Mahajan and Vinay [189].

complicated, but because the permanent and determinant are more familiar, and the overall structure of the program is the same whichever pair of functions is used, so long as the functions have certain nice properties. However, to give a feel for how natural these functions are, and to discuss how they might aid in moving from algebraic to Boolean complexity, we describe them here.

For **P/poly**, consider a layered arithmetic circuit with n inputs and n levels, with n gates on each level except the last, which has a single output gate. Each gate on one level has every gate in the previous level as an input. The gates on the first level are the input gates, with values given by the variables $x_1, \ldots, x_n$. The function computed at any other gate k is defined by $f_k := \sum x_{i,j}^{(k)} f_i f_j$, where the sum is over all pairs of gates (i, j) in the level preceding the level of k, and each $x_{i,j}^{(k)}$ is an independent variable. Let $H_n(X)$ denote the function computed by the output node, where $X = \{x_1, \ldots, x_n\} \cup \{x_{i,j}^{(k)}\}_{i,j,k}$. We may think of the variables $x_{i,j}^{(k)}$ as weights on the wires of the circuit; by setting the weights to appropriate constants we can control which circuit H is really computing. If $(f_n)_{n=1}^{\infty}$ is a family of functions computed by polynomial-size arithmetic circuits, then (f_n) is a p-projection of H_n, by setting the weights $x_{i,j}^{(k)}$ of H_n appropriately. Thus the function family (H_n) is complete for the class of polynomial-size arithmetic circuits over an arbitrary field, including the Boolean field $\mathbb{F}_2$ in which case this class is exactly **P/poly**.

The function constructed by Mulmuley and Sohoni [207] captures **NP** in some ways better than the permanent does, though it does not capture **NP** as exactly as H_n captures **P/poly**. The input to this function will be two $n \times n$ matrices X_0, X_1. For a binary string s of length n, let X_s denote the matrix whose i-th column is that of X_0 if $s_i = 0$ and is that of X_1 if $s_i = 1$, for each $1 \leq i \leq n$. Then Mulmuley and Sohoni suggest the study of $E(X) := \prod_{s \in \{0,1\}^n} \det(X_s)$. Although this is an exponential product of determinants, and so might seem closer to **NEXP** than **NP**, we discuss why this is a potentially good function to capture **NP**.

The function $E(X)$ shares many nice properties with the permanent and determinant—such as characterization by symmetries (see Section 3.4)—but more closely captures **NP** than the permanent does. First, Gurvits [136, pp. 450–451] showed that, if X_0 and X_1 are integer matrices, then testing whether $E(X)$ is zero is **NP**-complete (Boolean **NP**), by a reduction from the subset-sum problem. For the permanent, testing zero or nonzero is in **P** on *positive*

integer matrices; however, testing whether the permanent of an *arbitrary* integer matrix is zero is not even known to be in **NP**. Although this may seem like a slight difference, positivity often a very powerful restriction (see, e. g., [268, 135, 134], [130, Section 3] and references therein). Second, over a finite field of order q, $E(X)^q$ is $\{0,1\}$-valued, and computing $E(X)^q$ is in **NP**. In contrast, over finite fields computing the permanent (or its q-th power, to make it $\{0,1\}$-valued) is complete for $\mathbf{Mod_k P}$, which by Toda's Theorem [261] is as hard as the entire polynomial hierarchy. It is still an open question whether computing $E(X)^q$ over $\mathbb{F}_q$ is **NP**-complete, but at least it is known to be in **NP**. In particular, if one could show that it was not a p-projection of H_n, and hence not computable by polynomial-size arithmetic circuits, this would show $\mathbf{NP} \not\subseteq \mathbf{P/poly}$. No such statement is known to follow from showing that the permanent over a finite field is not computable by polynomial-size arithmetic circuits.

Despite not being **NP**-complete, computing $E(X)$ is closely related to another problem thought to be difficult, namely the problem of telling whether a system of polynomial equations has a common root. Even over $\mathbb{F}_2$ and when the equations all have degree at most 3, the problem is **NP**-complete by reduction from 3SAT. If we restrict to homogeneous polynomials, the problem is not known to be **NP**-complete, but it would be very surprising if the homogeneous case were in **P**. Mulmuley and Sohoni [207, p. 525] show that $E(X)$ is exactly the resultant of a system of homogeneous polynomial equations, so $E(X)$ vanishes if and only if the system of homogeneous equations has a solution. Hence, even over finite fields where we do not know that $E(X)$ is **NP**-complete, deciding the vanishing of $E(X)$ is as hard as solving systems of homogeneous polynomials, which is not expected to be in **P**.

3.4 The 10-foot view: characterization by symmetries

Up to this point, we could have used any pair of functions complete for two complexity classes and asked about the inclusion relationship between their orbit closures. If that were all there were to the GCT program, it would be little more than a restatement of the original complexity questions in an algebraic setting. Before we get to the real meat of the GCT program—the algebraic geometry and representation theory that really make GCT a program, rather than just a restatement—we can highlight one more important

feature of GCT without having to wade through any deep mathematics: characterization by symmetries.

Symmetry-characterization is a conjecturally crucial part of the GCT program. It is this property that may enable us to better understand certain functions—like permanent and determinant—and their orbit closures more than other ones of similar complexity. In particular, in Section 3.4.3, we show that if symmetry-characterization ends up playing as fundamental a role in GCT as it is conjectured to, it will automatically avoid the Razborov–Rudich natural proofs barrier. Symmetry-characterization also has other complexity-theoretic consequences that we discuss in the next few sections.

3.4.1 Background: stabilizers in group actions

Suppose G is a group acting on a set (graph, vector space, etc.) X (see Section 3.3.1). For any point $x \in X$, its *stabilizer* or *symmetry group* (sometimes in the literature also *isotropy group*) is $\{g \in G : gx = x\}$; this is often denoted $\mathrm{Stab}_G(x)$ or $\mathrm{Stab}(x)$ when G is clear from context, or G_x. It is an easy exercise to see that this is a group—if you are not familiar with this fact it is also a helpful and possibly illuminating exercise.

If G acts on a set X, we sometimes refer to X as a G-set. When X is a vector space and G acts by linear transformations, we call X a (linear) representation of G. If X and Y are both G-sets, then a map $f\colon X \to Y$ is called *G-equivariant* if f commutes with the action of G; in other words, it doesn't matter if we first act by G on X and then apply f, or first apply f and then act by G on Y. In symbols, for each $g \in G$ and each $x \in X$, $f(g \cdot x) = g \cdot f(x)$, where on the left hand side $g \cdot x$ is the action of G on X and on the right hand side $g \cdot f(x)$ is the action of G on Y. Two G-sets are said to be equivalent or isomorphic *as G-sets* if there is a G-equivariant isomorphism between X and Y. For example, if X and Y are sets, this is just a G-equivariant bijection; if X and Y are vector spaces, this is a G-equivariant bijective linear map, etc.

If X is a G-set and $x \in X$, then the orbit $Gx \subset X$ is also a G-set, in fact, it is the smallest G-subset of X containing x (or, if you like, the G-set "generated by" x).

If H is a subgroup of G, then the coset space of H in G is the set of cosets $gH = \{gh : h \in H\}$ for $g \in G$, and is denoted $G/H := \{gH : g \in G\}$. The coset space G/H has a natural G-action on it, namely the action by left multiplication: $g' \cdot gH = (g'g)H$. Since the

trivial coset H is a point of G/H, and every coset is of the form gH for some $g \in G$, it is clear that G/H consists of a single orbit under the natural action of G.

Since group actions are perhaps some of the most important aspects of groups, it should not be surprising that one of the most important, if not the most important, elementary theorems of group theory relates the orbits of a group action to its stabilizers.

Proposition 3.4.1 (Orbit–Stabilizer Theorem)**.** *Suppose $\rho\colon G \to \mathrm{Aut}(X)$ is an action of the group G on the set (graph, vector space, etc.) X, and let $x \in X$. Then the orbit Gx is equivalent as a G-set to the coset space $G/\operatorname{Stab}_G(x)$. In particular, $|Gx||\operatorname{Stab}_G(x)| = |G|$.*

Another way to phrase the Orbit–Stabilizer Theorem is that every single-orbit G-set is (equivalent to) a coset space. Since every G-set is partitioned into G-orbits, this says that the possible actions of G are completely determined internally to G, by its subgroups and their coset spaces. Or, perhaps more poetically but less accurately, the only group actions are actions on themselves.

Proof. We claim that the G-equivalence between Gx and $G/\operatorname{Stab}_G(x)$ is given by

$$gx \mapsto g \operatorname{Stab}_G(x)$$

First, we show that this map is well-defined. Suppose $g_1x = g_2x$ for some $g_1, g_2 \in G$. Then $g_2^{-1}g_1x = x$, so $g_2^{-1}g_1 \in \operatorname{Stab}_G(x)$. But then $g_2^{-1}g_1 \operatorname{Stab}(x) = \operatorname{Stab}(x)$, so $g_1 \operatorname{Stab}(x) = g_2 \operatorname{Stab}(x)$.

Next, we show that this map is G-equivariant. Let $g' \in G$. Then $g'(gx) = (g'g)x \leftrightarrow (g'g)\operatorname{Stab}(x) = g'(g\operatorname{Stab}(x))$.

Finally, we show that the map is bijective. Surjectivity follows from the fact that the map is G-equivariant and that both Gx and $G/\operatorname{Stab}(x)$ consist of single G-orbits. To show injectivity, suppose $g_1 \operatorname{Stab}(x) = g_2 \operatorname{Stab}(x)$. Then $g_2^{-1}g_1 \in \operatorname{Stab}(x)$, so $g_2^{-1}g_1x = x$, hence $g_1x = g_2x$. □

There is also, as one might expect, a relationship between the stabilizer of a point $x \in X$ and the stabilizer of another point gx in its orbit:

Proposition 3.4.2 (Definition of orbit types)**.** *Let X be a G-set. Then $\operatorname{Stab}_G(gx) = g \operatorname{Stab}_G(x) g^{-1}$.*

If H and K are two subgroups of G, then G/H and G/K are equivalent as G-sets if and only if H and K are conjugate within G. Conjugacy classes of subgroups may thus be referred to as the "orbit types" of G.

Proof. The first statement follows from the observation that if $s \in \mathrm{Stab}(x)$, then $gsg^{-1}(gx) = gs(g^{-1}g)x = gsx = gx$, where the final equality follows from the definition of $\mathrm{Stab}(x)$.

For the second statement, suppose that $f\colon G/H \to G/K$ is a G-equivalence. Then $f(H) = gK$ for some $g \in G$. Since the two cosets spaces consist of single orbits and f is G-equivariant by assumption, the value of $f(H)$ completely determines f, since $f(gH) = gf(H)$. Suppose $a \in H$. Then $aH = H$, so $f(aH) = f(H) = gK$. But also we must have $f(aH) = af(H) = agK$. So we have $agK = gK$. Multiplying on the right by g^{-1}, we get $agKg^{-1} = gKg^{-1}$. Since gKg^{-1} is a subgroup of G, this means that $a \in gKg^{-1}$. Every step in this chain of reasoning is reversible, so we get that $H = gKg^{-1}$.

Conversely, if $H = gKg^{-1}$, then similar reasoning to the above shows that defining $f(aH) = agK$ gives a well-defined G-equivalence between G/H and G/K. □

3.4.2 Symmetry-characterization and self-reduction: the Flip Theorem

Characterization by symmetries

There are several possible definitions of symmetry-characterization, some of which have so far proven more useful than others, but before we get to the possible definitions we motivate them with the example of the determinant.

It follows from basic linear algebra that $\det(X^T) = \det(X)$ and $\det(AXB) = \det(X)$ whenever $\det(A) = 1/\det(B)$. Thus we refer to the transformations $X \mapsto X^T$ and $X \mapsto AXB$ with $\det(A) = 1/\det(B)$ as symmetries of the determinant. It has also long been known [114] that these are the only symmetries of the determinant, and moreover, the determinant is the *only* function with these symmetries (up to scalar multiples, which is unavoidable since $\alpha \det(X)$ obviously has the same symmetries as $\det(X)$ for any $\alpha \in \mathbb{C}\backslash\{0\}$). What this means is that if $f(X)$ is any other homogeneous polynomial of degree n in the n^2 variables of the matrix $X = (x_{ij})_{i,j=1}^n$ such that $f(X^T) = f(X)$ and $f(AXB) = f(X)$ whenever $\det(A) = 1/\det(B)$, then f must be a scalar multiple of the determinant. In this sense, the determinant is characterized by its symmetries.

In fact, the symmetry-characterization of the determinant is essentially equivalent to the existence of the Gaussian elimination algorithm for computing the determinant. To illustrate this we give the full proof:

Proposition 3.4.3 (Symmetry-characterization of the determinant)**.** *If $f(X)$ is any degree n homogeneous polynomial in the n^2 variables of the matrix $X = (x_{ij})$ such that $f(AXB) = f(X)$ for all (A, B) such that $\det(A) = 1/\det(B)$, then $f(X) = \alpha \det(X)$ for some nonzero $\alpha \in \mathbb{C}$.*

Note that we do not need the symmetry $f(X^T) = f(X)$ in the statement of the proposition: it follows for free. In fact, our proof will show that this result holds over any algebraically closed field.

Proof by Gaussian elimination. Suppose f is as described in the statement of the proposition. Let $\alpha = f(I)$, I the $n \times n$ identity matrix. Note that, since f is homogeneous of degree n, $f(\beta I) = \beta^n \alpha$, so that on scalar matrices βI, f already agrees with $\alpha \det$.

We will evaluate f at $X = C$ for some matrix C of complex numbers by performing Gaussian elimination on C. There are three types of steps in the Gaussian elimination algorithm, each of which corresponds to an elementary matrix:

- Multiply the i-th row by a nonzero constant β. The corresponding elementary matrix is $M_{i,\beta}$ which is the identity matrix except that its i-th diagonal entry is replaced by β.

- Swap rows i and j. The corresponding elementary (permutation) matrix is P_{ij}, which is the identity matrix except rows i and j have been swapped.

- Add β times row j to row i ($j \neq i$). The corresponding elementary matrix $A_{ij,\beta} = I + \beta E_{ij}$ where E_{ij} is the matrix with a 1 in the (i, j) position and zeroes everywhere else.

Let $C_0 = C$, and let C_s be C after the s-th step of Gaussian elimination. We will define a new sequence of matrices C'_s such that $C'_0 = C_0 = C$, $f(C'_{s+1}) = f(C'_s)$, $\det(C'_{s+1}) = \det(C'_s)$, and the final step $C'_{s_{max}}$ is a scalar matrix assuming C was invertible.

Before defining the sequence C'_s, let us show how the result follows. Suppose C is invertible. Then we have $C'_{s_{max}} = \beta I$ for some β, so we have $f(C'_{s_{max}}) = f(\beta I) = \alpha\beta^n = \alpha \det(C'_{s_{max}})$. But then from the equalities $f(C'_{s+1}) = f(C'_s)$ and $\det(C'_{s+1}) = \det(C'_s)$ it follows by induction that

$$f(C) = f(C'_0) = \alpha \det(C'_0) = \alpha \det(C).$$

Finally, since f and det are polynomials, they are in particular continuous functions. The invertible matrices are dense in the set of all matrices, and since the identity $f(C) = \alpha \det(C)$ holds for all invertible matrices, it follows by continuity that this identity holds for all matrices.

To finish the proof, we define the sequence C'_s and show that is has the desired properties. Let $C'_0 = C_0 = C$.

- If $C_{s+1} = M_{i,\beta}C_s$, then let $C'_{s+1} := M_{i,\beta}C'_s(\beta^{-1/n}I)$. Note that $\det(\beta^{-1/n}I) = \beta^{-1} = 1/\det(M_{i,\beta})$, so by assumption we have $f(C'_{s+1}) = f(M_{i,\beta}C'_s\beta^{-1/n}I) = f(C'_s)$. By the same reasoning, we have $\det(C'_{s+1}) = \det(C'_s)$.

- If $C_{s+1} = P_{ij}C_s$, then let $C'_{s+1} := P_{ij}C'_s\beta I$ where $\det(\beta I) = -1$ (we may take $\beta = -1$ if n is odd, and $\beta = \sqrt{-1}$ if n is even). As before, we have $\det(\beta I) = -1 = 1/\det(P_{ij})$, so $f(C'_{s+1}) = f(C'_s)$ and $\det(C'_{s+1}) = \det(C'_s)$.

- If $C_{s+1} = A_{ij,\beta}C_s$, then let $C'_{s+1} := A_{ij,\beta}C'_s$. Since $\det(A_{ij,\beta}) = 1$, we can multiply by I on the right to see, as in the previous two cases, that $f(C'_{s+1}) = f(C'_s)$ and $\det(C'_{s+1}) = \det(C'_s)$.

Finally, if C was invertible, then $C_{s_{max}}$ is the identity matrix. It is easily verified that at each step C_s differs from C'_s only by multiplication by a scalar matrix, so that $C'_{s_{max}}$ is scalar, as promised. □

The only special facts we used about $\mathbb{C}$ were the argument by continuity—which can be replaced by a degree argument over a general field $\mathbb{F}$ so long as $|\mathbb{F}| > n$—and the use of $\beta^{-1/n}$ and $\sqrt{-1}$. Hence, the proof goes through over any algebraically closed field.

Now we come to the definition of symmetry-characterization. Looking back at the determinant, it may seem somewhat unfair or asymmetric that we only consider those $f(X)$

such that $f(X) = f(X^T)$ and $f(AXB) = f(X)$ whenever $\det(A) = 1/\det(B)$—that is, we seem to be "cheating" by building the determinant into its own definition of symmetry-characterization. Why aren't we instead considering those f such that $f(X) = f(X^T)$ and $f(AXB) = f(X)$ whenever $f(A) = 1/f(B)$?

The answer comes from the viewpoint of group actions. Recall the action of $\mathrm{GL}(M_n(\mathbb{C}))$ on $\mathrm{Poly}^n(M_n(\mathbb{C}))$ from Section 3.3.2. The symmetries of the determinant form a subgroup of $\mathrm{GL}(M_n(\mathbb{C}))$; for example, the symmetry $X \mapsto X^T$ is given by an $n^2 \times n^2$ permutation matrix that permutes the basis of $M_n(\mathbb{C})$ by $E_{ij} \mapsto E_{ji}$ (where E_{ij} is the matrix with a 1 in the (i, j) position and zeroes everywhere else). Similarly, for each pair of $n \times n$ invertible matrices A and B, the map $X \mapsto AXB$ can be realized as an element of $\mathrm{GL}(M_n(\mathbb{C}))$, that is, an $n^2 \times n^2$ invertible matrix.

We denote the symmetry group of the determinant within $\mathrm{GL}(M_n(\mathbb{C}))$ by $\mathrm{Stab}_{\mathrm{GL}(M_n(\mathbb{C}))}(\det_n)$ (see Section 3.4.1 for more on this notation). What we have really said above is that if f is any other point in $\mathrm{Poly}^n(M_n(\mathbb{C}))$ that is also fixed by every element of $\mathrm{Stab}_{\mathrm{GL}(M_n(\mathbb{C}))}(\det_n)$, then f is a scalar multiple of $\det_n$. Since $\mathrm{GL}(M_n(\mathbb{C}))$ acts on $\mathrm{Poly}^n(M_n(\mathbb{C}))$ by linear transformations, this scalar multiple is the best we could hope for. The fact that $\mathrm{Stab}_{\mathrm{GL}(M_n(\mathbb{C}))}(\det_n)$ involves the determinant in its description—since it includes all maps $X \mapsto AXB$ such that $\det(A) = 1/\det(B)$— is a property special to the determinant, but is not inherent to the notion of symmetry-characterization that we are interested in.

Definition 3.4.4 (Symmetry-characterization)**.** A homogeneous polynomial f of degree d on n variables is *symmetry-characterized* if it is the only such homogeneous polynomial that is fixed by f's symmetries, up to scalar multiplication. In other words, if $g \in \mathrm{Poly}^d(\mathbb{C}^n)$ is such that $A \cdot g = g$ for all $A \in \mathrm{Stab}_{\mathrm{GL}(\mathbb{C}^n)}(f)$, then $g = \alpha f$ for some scalar α.

For example, since the permanent and determinant are degree n on n^2 variables, we work in $\mathrm{Poly}^n(\mathbb{C}^{n^2}) = \mathrm{Poly}^n(M_n(\mathbb{C}))$.

There is another possible definition of symmetry-characterization which is sometimes confused for this one, and although the other possibility is natural, it is not what we are interested in. This other possibility, which is weaker than the notion defined above, is to say that a function f is symmetry-characterized if it is the only function with its stabilizer; in other words, any function g that is not a scalar multiple of f must have $\mathrm{Stab}(g) \neq \mathrm{Stab}(f)$.

In contrast, Definition 3.4.4 says that not only are there no other functions g with $\mathrm{Stab}(g) = \mathrm{Stab}(f)$, but there aren't even other functions g with $\mathrm{Stab}(g) \supseteq \mathrm{Stab}(f)$. This is a distinction which, in specific cases seems to cause no trouble, but some conjectures regarding symmetry-characterized functions may not apply to the weaker notion of symmetry-characterization mentioned in this paragraph.

A function f that is symmetry-characterized also has the property that its orbit is the unique orbit (as always, up to scalar multiplication) with its orbit type (see Proposition/Definition 3.4.2). Saying that a function f has the property that its orbit is the unique orbit with its orbit type is again a different possible definition of symmetry-characterization. However, it differs from Definition 3.4.4 in two respects. First, it has the same issue as the possible alternative definition discussed above; in particular Definition 3.4.4 implies not only that Gf is the unique orbit with its orbit type, but that no other orbit type is a supergroup of the orbit type of f. Second, it is conceivable that Gf is the unique orbit of its type, but that the orbit Gf contains more than one function that is fixed by $\mathrm{Stab}_G(f)$. Despite being weaker than Definition 3.4.4, this consequence regarding orbit types may still be useful to consider.

It is easy to generalize the notion of symmetry-characterization to any group action—when the group action is not linear, we can avoid the caveats about scalar multiples—including actions on spaces other than $\mathrm{Poly}^d(\mathbb{C}^n)$, such as the space of all polynomials of degree $\leq d$. In this thesis we will only have occasion to consider the linear notion of symmetry-characterization, though we will occasionally use it within spaces other than $\mathrm{Poly}^d(\mathbb{C}^n)$.

The proof of the symmetry-characterization of the determinant is of a very special form. To give the flavor of other symmetry-characterization results, we now discuss the symmetry-characterization of the permanent and $E(X)$ (from Section 3.3.3 and Mulmuley–Sohoni [207]).

Proposition 3.4.5. *The permanent of $n \times n$ matrices is symmetry-characterized over any field $\mathbb{F}$ with $|\mathbb{F}| > n$.*

Proof. The symmetries of the permanent we need in this proof are:

1. $\mathrm{perm}(AXB) = \mathrm{perm}(X)$ whenever A and B are both permutation matrices

2. $\operatorname{perm}(AXB) = \operatorname{perm}(X)$ whenever A and B are diagonal matrices and the product of the nonzero entries of A with the nonzero entries of B is 1. (Since A and B are diagonal, this product happens to be equal to $\operatorname{perm}(AB)$ and $\det(AB)$.)

These transformations, together with $\operatorname{perm}(X) = \operatorname{perm}(X^T)$, in fact generate the full symmetry group of the permanent [192], but for this proof all we need to know is that the above transformations are symmetries of the permanent. The remaining symmetries of the permanent, including the transpose, will come for free.

Suppose $f(X)$ is a homogeneous polynomial of degree n having the symmetries listed above. Let $M_{i,\beta}$ be the diagonal matrix whose i-th diagonal entry is β and all other diagonal entries are 1. Then $f(M_{i,\beta}XM_{j,\beta^{-1}}) = f(X)$ for all i, j and all $\beta \neq 0$. Suppose a term contains the variable $x_{k,j}$; consider the preceding identity for any $i \neq k$. This identity implies that the term must also contain at least one entry from the i-th row. Since this is true for every $i \neq k$, and the term already contains a variable from the k-th row, the term, and hence every term, must contain at least one variable from each row. Since f is homogeneous of degree n, every term contains of *exactly* one variable from each row. Similarly, every term of f must involve exactly one entry from each column. That is, every monomial of f must look like $x_{1,\pi(1)} \cdots x_{n,\pi(n)}$ for some permutation π. In particular, the monomials of f are a subset of the monomials of the permanent.

If f is zero we are done, so we assume f is nonzero. Then f contains some nonzero term $\alpha x_{1,\pi(1)} \cdots x_{n,\pi(n)}$ for some permutation π. Let σ be any other permutation and let P_σ be the corresponding permutation matrix. Then $f(X) = f(XP_\sigma)$ implies that f also contains the monomial $x_{1,\sigma(\pi(1))} \cdots x_{n,\sigma(\pi(n))}$ with the same coefficient, α. Since this is true for every permutation, every such monomial appears, and they all have the same coefficient α. Hence $f(X) = \alpha \operatorname{perm}(X)$. □

Proposition 3.4.6 (Theorem 5.1 of Mulmuley [206])**.** *The function $E(X^{(0)}, X^{(1)})$ (see Section 3.3.3) is symmetry-characterized within the space of homogeneous polynomials of degree $n2^n$ divisible by $\det(X^{(0)}) \det(X^{(1)})$.*

For symmetry-characterization here, the space of homogeneous polynomials of fixed degree $d \geq 2n$ divisible by $\det(X^{(0)}X^{(1)})$ is indeed a vector space: it consists of all functions $\det(X^{(0)}X^{(1)})f(X)$ with $\deg f = d - 2n$. The ambient group action we consider is the

natural action of $\mathrm{Stab}_{\mathrm{GL}(\mathbb{C}^{n\times 2n})}(\det(X^{(0)}X^{(1)}))$ on this space. This stabilizer is exactly $\mathrm{Stab}_{\mathrm{GL}(M_n(\mathbb{C}))}(\det(X^{(0)})) \times \mathrm{Stab}_{\mathrm{GL}(M_n(\mathbb{C}))}(\det(X^{(1)})) \rtimes \mathbb{Z}/2\mathbb{Z}$, where the $\mathbb{Z}/2\mathbb{Z}$ acts by swapping $X^{(0)}$ and $X^{(1)}$. (Because of the symmetries of $E(X)$, we could have equally well considered the space of homogeneous polynomials of degree $n2^n$ on $n \times 2n$ variables divisible by $\det(X^{(0)})$. However, then the ambient group, namely $\mathrm{Stab}_{\mathrm{GL}(\mathbb{C}^{n\times 2n})}(\det(X^{(0)}))$, would not have been reductive, which would introduce additional complications in the ensuing representation theory.)

The proof of symmetry-characterization of $E(X)$ involves some basic algebraic geometry that we do not want to go into here. A key ingredient is that $E(X^{(0)}, X^{(1)})$ is symmetric under $E(AX^{(0)}, AX^{(1)}) = E(X^{(0)}, X^{(1)})$, whenever $\det(A) = 1$. Other symmetries of $E(X)$ include $E(X^{(1)}, X^{(0)}) = E(X^{(0)}, X^{(1)})$ and $E(X^{(0)}P_\pi, X^{(1)}P_\pi) = E(X^{(0)}, X^{(1)})$ for any permutation matrix P_π.

The Flip Theorem

Fortnow, Pavan, and Sengupta [110], building off of a learning algorithm of Bshouty *et al.* [64] showed that if $\mathbf{NP} \not\subseteq \mathbf{P/poly}$, then for every exponent k and every length n, there is a set of at most $\mathrm{poly}(n)$ formulas $\{\varphi_1, \ldots, \varphi_{\mathrm{poly}(n)}\}$ such that every circuit of size $\leq n^k$ differs from SAT on at least one of these formulas. In other words, for every small circuit C there is some i such that either φ_i is satisfiable and $C(\varphi_i) = 0$, or φ_i is unsatisfiable and $C(\varphi_i) = 1$. This set of formulas can be found in $\mathbf{ZPP}^{\mathbf{NP}}$, which is the complexity of Bshouty *et al.*'s learning algorithm.

Atserias [18] gave a similar result also based on Bshouty *et al.*'s learning algorithm: Atserias removes the **NP** oracle from the complexity of constructing the set of formulas, but at the cost that the algorithm to do the constructing now depends on the circuit C. More precisely, if $\mathbf{NP} \not\subseteq \mathbf{P/poly}$ then for each $L \in \mathbf{P/poly}$, there is a probabilistic polynomial-time oracle algorithm A such that $A^L(0^n)$ outputs a small set of formulas $\{\varphi_1, \ldots, \varphi_k\}$ with $k \leq \mathrm{poly}(n)$ such that L differs from SAT on some φ_i.

Generally speaking, these results say that if $\mathbf{NP} \not\subseteq \mathbf{P/poly}$, then this complexity separation can be *effectively* witnessed. We refer to such results in general as "flip theorems:" they say that if some function is hard—e. g., SAT does not have polynomial-size circuits—

then another function must be easy—e. g., a function that witnesses the hardness of SAT by constructing counterexamples against small circuits.

We note that the non-computability of the halting problem furnishes another such flip theorem: there is an algorithm D such that if M is any machine purporting to solve the halting problem, then $D(M)$ outputs the description of a machine M' for which $M(M')$ incorrectly computes whether or not M' halts. Note that the "effectiveness" of a flip theorem is desirably of the same level as the complexity separation being witnessed: in the case of the halting problem, showing that there is a set in **CE** that is not computable, the witnessing procedure D is computable.

Nothing quite so strong is known for **NP** versus **P/poly**. In particular, although Atserias's algorithm runs in probabilistic polynomial time, and hence within **FP/poly**, the algorithm depends on the language $L \in \mathbf{P/poly}$ being diagonalized against. In contrast, Fortnow, Pavan, and Sengupta's algorithm is independent of the language being diagonalized against, but has the higher complexity $\mathbf{ZPP}^{\mathbf{NP}}$.

In geometric complexity theory, we get the stronger result that the witnesses *do not depend on* L and the algorithm constructing the witnesses has low complexity. For example, recall Theorem 3.2.1, which we now give two proofs of:

Proof of Theorem 3.2.1 via downward self-reducibility. The downward self-reducibility of the permanent follows from its Laplace expansion:

$$\operatorname{perm}(X) = \sum_{i=1}^{n} x_{n,i} \operatorname{perm}(X(n|i))$$

where by $X(n|i)$ we mean the square $(n-1) \times (n-1)$ submatrix of X that results from removing the n-th row and the i-th column of X. In particular, if C is an arithmetic circuit computing $\operatorname{perm}_n(X)$, then by plugging in 1 for $x_{n,n}$ and 0 for all other $x_{i,n}$ and $x_{n,i}$ $(i \neq n)$, we get a circuit computing $\operatorname{perm}_{n-1}(X(n|n))$. For any circuit C with n^2 variables, call the resulting circuit $C|_{n-1}$.

Given a circuit C, it computes the permanent if and only if, $C|_1(X) = x_{1,1}$ and

$$C|_k(X) = \sum_{i=1}^{k} x_{k,i} C|_{k-1}(X) \qquad \text{for all } 2 \leq k \leq n.$$

By the Schwarz–Zippel Lemma [233, 280], these n identities may be tested by choosing sufficiently many random matrices for X. These random matrices provide the set of counterexamples promised in the theorem. □

Proof of Theorem 3.2.1 via symmetry-characterization. By the symmetry-character-ization of the permanent, an arithmetic circuit C computes the permanent if and only if C computes a homogeneous degree n polynomial and C has the same symmetries as the permanent. These two conditions can be tested by verifying the following circuit identities:

$$\begin{aligned} C(\alpha X) &= \alpha^n C(X) \\ C(M_{i,\alpha} X) &= \alpha C(X) \\ C(X) &= C(X^T) \\ C(P_\pi X) &= C(X) \qquad \text{for } \pi = (12) \text{ and } \pi = (12\cdots n) \end{aligned}$$

In the above identities, α is to be treated as a new variable, $M_{i,\alpha}$ is the identity matrix except its (i,i) entry is replaced by α, and P_π is a permutation matrix. The identities involving P_π need only be tested for permutations π from a generating set for S_n. As in the previous proof, the random matrices used to verify these circuit identities serve as the set of universal counterexamples. □

The proof via symmetry-characterization has the advantage that it also works for the function $E(X)$ designed to capture the complexity class **NP**. In contrast, using, for example, the downward self-reducibility of SAT, the best flip theorems that are known for **NP** are the ones mentioned above due to Atserias [18] and Fortnow, Pavan, and Sengupta [110]. The only sense in which the Flip Theorem for $E(X)$ may be weaker than these results is that $E(X)$ is not known to be **NP**-complete, so the Flip Theorem for $E(X)$ does not follow simply from the assumption that $\mathbf{NP} \not\subseteq \mathbf{P/poly}$, but rather from the *a priori* slightly stronger assumption that $E(X)$ is not in algebraic $\mathbf{P/poly}$.

Incidentally, the proof via symmetry-characterization also reduces the size of the set of counterexamples—and the number of polynomial identity tests needed—from $O(n)$ to $O(1)$, but we do not yet know of any immediately useful consequences of this reduction.

To get a flip theorem for $E(X)$, one also needs the following two facts. First, testing if an arithmetic circuit C computes a homogeneous function of degree $n2^n$ can be verified by the

circuit identity $C(\alpha X) = \alpha^{n2^n} C(X)$, where α is a new variable. Note that $\alpha \mapsto \alpha^{n2^n}$ can be computed by an arithmetic circuit of linear size by repeated squaring. Second, testing if an arithmetic circuit $C(X)$ computes a function that is divisible by $\det(Y)$, where Y is any subset of the variables X, can also be verified by picking random values for Y such that $\det(Y) = 0$. If Y is an $n \times n$ matrix, this is easily achieved by picking completely random values for the first $n-1$ rows of Y—call these rows $\vec{r}_1, \ldots, \vec{r}_{n-1}$—and then picking n additional random values $a_1, \ldots, a_{n-1}$ and making the last row of Y equal to $\sum_{i=1}^{n-1} a_i \vec{r}_i$. Since this is no longer strictly an arithmetic circuit identity testing problem, a slightly stronger derandomization assumption is needed in the flip theorem for $E(X)$, but the randomized result is the same as that for the permanent.

Both proofs of Theorem 3.2.1 only depend on the hard function being characterized by some circuit identities, or slight generalizations of circuit identities in the case of $E(X)$. Mulmuley formalizes this notion in the following definition:

Definition 3.4.7 (Characterization by circuit identities, Definition 11.1 of Mulmuley [206])**.** A family of polynomials (f_n) is *characterized by circuit identities* if each f_n is the only nonzero polynomial over $\mathbb{Q}$, up to scaling, that satisfies a $\text{poly}(n)$ number of polynomial identities with integral coefficients, each having a specification of $\text{poly}(n)$ many bits and containing $O(1)$ terms. Each identity here is of the form $g(f(X_1), \ldots, f(X_k)) = 0$, where $g(u_1, \ldots, u_k)$ is a polynomial computable by a constant-bit-size circuit over $\mathbb{Z}$, and each X_i can be computed from X by a $\mathbb{Z}$-circuit of $\text{poly}(n)$ bit-size.

If we only require that g be computable by $\text{poly}(n)$-bit-size $\mathbb{Z}$-circuits, we say f is *weakly characterized by circuit identities*.

Mulmuley calls this "characterization by symmetries," but we reserve that term for Definition 3.4.4, since that is more closely associated with actual symmetries, as opposed to more general properties that can be characterized by circuit identities.

Under suitable conditions, such as those that hold for the determinant, permanent, and $E(X)$, symmetry-characterization in the sense of Definition 3.4.4 implies characterization by circuit identities. Discussing these conditions in detail would take us too far afield into the theory of algebraic groups, but we can at least mention two main ingredients: connected algebraic subgroups of GL_n, including the identity component of $\text{Stab}(f)$, are generated by

at most n^2 many one-parameter subgroups, and finite groups of order N are generated by at most $\log_2 N$ elements, so as long as $|G| \leq 2^{\text{poly}(n)}$, G is generated by poly(n) many elements.

Although characterization by circuit identities, and not necessarily symmetry-characterization, is all that is needed for a flip theorem, it is not enough for other useful representation-theoretic consequences It thus seems to us that symmetry-characterization is the more fundamental property for GCT.

3.4.3 Symmetry-characterization avoids the Razborov–Rudich barrier

Intuitively, the known barriers to complexity separations—relativization, algebraic relativization, and Razborov–Rudich natural proofs (see Section 2.1.6)—all have a similar message: any significant separation of complexity classes must really use properties of problems that are very specific to those problems. These properties cannot relativize, even algebraically, and they must either be non-constructive or not apply to very many functions. From the intuitive point of view, it seems clear that any crucial use of symmetry-characterization will, *a priori*, avoid these barriers.

For the Razborov–Rudich barrier, this intuition can be made precise: very few functions are symmetry-characterized, so symmetry-characterization violates the largeness criterion. We suspect that symmetry-characterization also violates the constructivity criterion, but we leave that as an open question:

Open Question 3.4.8. Given a homogeneous degree d polynomial—for example, over a finite field with the function given as a complete table, or over an arbitrary field as a list of the coefficients attached to all monomials of degree d—what is the complexity of testing if that function is symmetry-characterized?

Although it remains open whether there is a purely algebraic version of the Razborov–Rudich barrier, we nonetheless present:

Proposition 3.4.9. *Over $\mathbb{C}$, the set of symmetry-characterized points in* $\text{Poly}^d(\mathbb{C}^n)$ *has measure* 0 *for $d \geq 2$.*

Proof. This follows almost immediately from Theorem A of Richardson [224], which says that there is a subset $S \subseteq \text{Poly}^d(\mathbb{C}^n)$ such that every point in S has the same orbit type (recall

Proposition/Definition 3.4.2 and the discussion after Definition 3.4.4) and the complement of S has measure zero. Recall that the orbit of a symmetry-characterized point is the only orbit of its orbit type, up to scaling. The dimension of a $\mathrm{GL}(\mathbb{C}^n)$ orbit is at most n^2, so the dimension of an orbit and all its scalings together is at most $n^2 + 1$. Thus, when $d > 2$, no single orbit in the action of $\mathrm{GL}(\mathbb{C}^n)$ on $\mathrm{Poly}^d(\mathbb{C}^n)$, together with its scalings, is large enough for its complement to have measure zero, so no symmetry-characterized point belongs to S. □

The result of Richardson [224] also applies to the setting of $E(X)$ namely, the action of $\mathrm{Stab}(\det(X^{(0)}X^{(1)}))$ on the space of degree $n2^n$ homogeneous polynomials divisible by $\det(X^{(0)}X^{(1)})$.

Over an arbitrary field—including finite fields, where the Razborov–Rudich natural proofs barrier applies—we use the notion of characterization by circuit identities due to Mulmuley [206, Definition 11.1]:

Proposition 3.4.10 (Mulmuley [206])**.** *The number of functions on* $\mathrm{poly}(n)$ *variables over a field* $\mathbb{F}$*, weakly characterized by circuit identities (see Definition 3.4.7) is at most* $2^{\mathrm{poly}(n)}$*.*

Proof. This result is, in some sense, built into Mulmuley's definition of characterization by circuit identities. Definition 3.4.7 bounds the size of the circuit identities *in terms of bit-length* by $\mathrm{poly}(n)$, so there are only $2^{\mathrm{poly}(n)}$ possibilities for the circuit identities. (Note, however, that not all sets of identities uniquely specify a single function.) Hence there are at most this many functions characterized by circuit identities. □

3.4.4 An algorithmic consequence of symmetry-characterization

Agrawal and Saxena [7] showed that testing whether two homogeneous polynomials lie in the same GL-orbit is as hard as graph isomorphism; combined with a result of Kayal and Saxena [161], this problem is also as hard as factoring integers. In contrast to this general situation, using the symmetry-characterization of the permanent and determinant, Kayal [159] showed that testing whether a function lies in the orbit of determinant or in the orbit of permanent can be done in probabilistic polynomial time.

The basic idea is as follows; we give it in more detail in Section 4.6. Given a homogeneous polynomial f of degree n on n^2 variables, compute its symmetry group $\mathrm{Stab}_{\mathrm{GL}(M_n(\mathbb{C}))}(f)$.

By symmetry-characterization, f lies in the $\mathrm{GL}(M_n(\mathbb{C}))$ orbit of $\det_n$ if and only if $\mathrm{Stab}(f)$ is conjugate to $\mathrm{Stab}(\det_n)$. Then Kayal uses properties specific to the stabilizers of the permanent and determinant to complete this final step.

Computing the symmetry group and testing whether two symmetry groups are conjugate are *a priori* hard computational problems in algebraic geometry. However, by using the theory of Lie algebras these problems may be reduced to simpler, though still nontrivial, problems in linear algebra. In Chapter 4 we generalize Kayal's result by tackling the problem of conjugacy of Lie algebras of subgroups of GL directly. We show that in general this problem is as hard as graph isomorphism, but that for a fairly wide range of cases, including that of the Lie algebra of $\mathrm{Stab}(\det_n)$, the problem can be solved in polynomial time.

Although algorithmically testing if a point lies in the orbit of determinant and permanent does not lead immediately to any lower bounds, Kayal [159, Section 5.2] points out an interesting connection between affine-invariant properties of polynomials and lower bounds.

Finally, we mention that algorithmically determining if a point lies in the orbit of determinant and permanent can be seen as a first step towards Mulmuley's conjecture that it can be determined in $\mathrm{poly}(n, m)$ time whether the padded permanent of an $n \times n$ matrix lies in the orbit closure of the $m \times m$ determinant.

3.5 The view from the ground

Although understanding the $\mathrm{GL}(M_n(\mathbb{C}))$-orbit of the permanent and determinant is an important first step, the objects we are really interested in are the endomorphism orbits and the orbit closures. Recall that in order to place a smaller permanent perm_n into the same space as a larger determinant $\det_m$, we had to multiply the permanent by a trivial factor to get the "padded permanent" $z^{m-n}\,\mathrm{perm}_n$. However, the padded permanent does not lie in the GL-orbit of the determinant: the determinant is an irreducible polynomial—it cannot be factored as the product of two non-constant polynomials—and the property of being irreducible is preserved by the action of GL. In contrast, the padded permanent is easily seen to be the product of z^{m-n} and perm_n, hence it does not lie in the GL-orbit of the determinant. This suggests a strong need for understanding the endomorphism orbits and the orbit *closures*.

In this section we will discuss an elementary approach to understanding orbit closures.

3.5.1 Using the zeroes of a function to understand its orbit closure

Suppose we want to understand the orbit closure of the determinant (or the permanent, $E(X)$, matrix multiplication, etc.—almost nothing we say in this section will be specific to the determinant). A first step is to understand its orbit; representation theory directly suggests a method to understand the orbit. The crucial second step is to understand the boundary of the orbit closure: those functions f that are the limit of functions equivalent to the determinant, but such that f is not itself equivalent to the determinant. In this section, our goal is to show:

> *understanding the boundary of the orbit closure of a function is essentially equivalent to understanding the linear subspaces contained in the set of zeroes of that function.*

In particular, until recently almost all the known lower bounds [274, 201, 74, 72, 200] on permanent versus determinant were essentially achieved by studying the zero set of the determinant and permanent[4], which is a subset of $M_n(\mathbb{C})$. In contrast, in GCT we are studying the orbit closures of these functions, which are subsets of the much larger space $\mathrm{Poly}^n(M_n(\mathbb{C}))$. But the message of this section is that there are further properties of the zero loci of these functions—again, living in the much smaller space—which can give us a wealth of information about the much higher-dimensional orbit closures.

Since we will be referring so frequently to the orbit and orbit closure of det here, we give them shorter names: $\mathcal{O} = \mathrm{GL}(M_n(\mathbb{C})) \cdot \det_n$ for the orbit, and $\overline{\mathcal{O}}$ for the orbit closure.

We also introduce here the projective point of view. We saw that the symmetry-characterization of the determinant is only up to a constant scalar multiple, since the action of $\mathrm{GL}(M_n(\mathbb{C}))$ on $\mathrm{Poly}^n(M_n(\mathbb{C}))$ is linear. Most interesting properties of homogeneous polynomials are preserved under (non-zero) scalar multiplication—their zero sets, their complexity,

4. We are only aware of one exception, namely Babai and Seress [36] proved a $\sqrt{2}n - 6\sqrt{n}$ lower bound by purely combinatorial methods. Around the same time, von zur Gathen had achieved $1.06n - 1$ by geometric methods, and then Cai achieved the then-best bound of $\sqrt{2}n$ by geometric methods. Meshulam achieved the same bound as Cai by considering subspaces of matrices all of which have rank at least k; in other words, (affine) linear subspaces of the *complement* of the zero locus of $k \times k$ minors. Such zero loci are called *determinantal varieties*. Although this is not strictly speaking the zero locus of the determinant, it has a distinctly similar geometric flavor.

etc. In some sense we only care about polynomials up to scalar multiplication. We refer to this as the "projective" viewpoint.

Suppose $f \in \overline{\mathcal{O}}$. Then by definition there is a sequence of points $f_k \in \mathcal{O}$ such that $\lim_{k\to\infty} f_k = f$. Since $\mathcal{O}$ is the orbit of the determinant, there must be a corresponding sequence of elements $A_k \in \mathrm{GL}(M_n(\mathbb{C}))$ such that $f_k = A_k \cdot \det$ (recall the action of $\mathrm{GL}(M_n(\mathbb{C}))$ on $\mathrm{Poly}^n(M_n(\mathbb{C}))$ from Section 3.3). It is not hard to imagine that we may replace the sequence A_k ($k \in \mathbb{N}$) with a continuous family $A_t \in \mathrm{GL}(M_n(\mathbb{C}))$ ($t \in \mathbb{C}\backslash\{0\}$) such that $\lim_{t\to 0} A_t \cdot \det = f$. Note that now we take the limit as $t \to 0$ instead of $t \to \infty$; this is achieved simply by replacing t with $1/t$. We would like to expand the entries of A_t as Taylor series in t.

A classical result from algebraic geometry[5] says that we may do much more: we may take such a family A_t where the entries of A_t are polynomials in t. (We need to allow that $A_t \notin \mathrm{GL}(M_n(\mathbb{C}))$ for finitely many values of t, but this does not affect taking the limit as $t \to 0$.) So we may write $A_t = A^{(0)} + tA^{(1)} + \cdots + t^d A^{(d)}$ with $A^{(0)} \neq 0$, where each $A^{(i)}$ is a linear map $A^{(i)} \colon M_n(\mathbb{C}) \to M_n(\mathbb{C})$.

We now want to expand $f_t(X) = \det(A_t(X))$ in terms of t. We may write $f_t(X) = \det(A^{(0)}(X)) + tf^{(1)}(X) + \cdots + t^D f^{(D)}(X)$, where D may be as large as nd, since the determinant has degree n. If $\det(A^{(0)}(X)) \neq 0$, then $f(X) = \lim_{t\to 0} f_t(X) = \det(A^{(0)}(X))$ lies in the endomorphism orbit of determinant.

If $\det(A^{(0)}(X)) = 0$, then f does not lie in the endomorphism orbit. It is tempting to think that in this situation f must be equal to zero, since we now have $f = \lim_{t\to 0} t(f^{(1)}(X) + tf^{(2)}(X) + \cdots + t^{D-1} f^{(D)}(X))$. However, for any fixed t, we may divide this expression by t to instead get $f^{(1)}(X) + tf^{(2)}(X) + \cdots + t^{D-1} f^{(D)}(X)$, and we get a projectively equivalent function. Thus we get a point f that is (projectively) in the orbit of determinant but not in its endomorphism orbit.

Before proceeding further, we discuss one of the few known examples of such functions, due to Landsberg, Manivel, and Ressayre [177]:

Proposition 3.5.1 (Proposition 3.5.1 of Landsberg, Manivel, and Ressayre [177])**.** *Let* $A(X) = \frac{1}{2}(X - X^T)$ *and* $S(X) = \frac{1}{2}(X + X^T)$ *be the antisymmetric and symmetric parts,*

5. This is essentially part of the proof that the Zariski-closure of a Zariski-open set is the same as its closure in the usual complex topology. See, for example, Theorem 2.33 of Mumford [208].

respectively, of the variable matrix X. Let Pf *denote the Pfaffian of a matrix, and let* $\mathrm{Pf}_i(Y)$ *denote the Pfaffian of the matrix Y after removing the i-th row and column of Y. Then for n odd, the function*

$$P_\Lambda(X) := \sum_{i,j=1}^{n} S(X)_{i,j} \operatorname{Pf}_i(A(X)) \operatorname{Pf}_j(A(X))$$

lies in the orbit closure of $\det_n(X)$*, but not in its endomorphism orbit.*

The function P_Λ above is the projective limit $\lim_{t\to 0} \det_n(A_t(X))$ for $A_t(X) = A(X) + tS(X)$, where A and S are as defined in the proposition. In fact, it is exactly the limit as t goes to 0 of $\frac{1}{t}\det_n(A(X) + tS(X))$. Showing that this limit yields the expression in the proposition is an exercise in derivatives of the determinant, specifically the Taylor series of in t of $\det(A_t(X))$, and the relationship between the determinant and the Pfaffian; this exercise may also help give a better feel for what it means for a function to lie in the orbit closure of determinant. Note that the constant term vanishes: $\det(A(X)) = 0$ since the determinant of an odd-dimensional antisymmetric matrix is always zero.

Here we see that the analogy put forth in Section 3.3.2 breaks down somewhat, though we still believe the large-scale picture suggested by the analogy is accurate. The analogy would suggest that, since P_Λ lies in the boundary of the orbit closure of determinant, but not in its orbit, that P_Λ should have strictly lower complexity than the determinant. But the next proposition suggests that their complexities are essentially the same.

This crack in the analogy was to be expected, however: $\mathcal{D}et_n$ only deals with a single determinant, whereas the notion of reduction used in complexity uses heavily the asymptotics associated with the *family* $(\det_n)_{n=1}^{\infty}$. The reductions in this next proposition reduce $\det_n$ to P_Λ on inputs essentially twice as large; the fact that P_Λ is not in the orbit of determinant suggests that such a reduction is not possible without blowing up the size of the input by a constant factor bounded away from 1. Also, here we are not using any of the complexity aspects afforded by padding.

Proposition 3.5.2. *Over any field, the function* $\det(X)^2$ *is a p-projection of P_Λ. In the Boolean model, the determinant of an integer matrix reduces to P_Λ in* **DLOGTIME***-uniform* $\mathbf{TC}^0$.

The exact weakness of the reduction used here is not critical, but for the statement to be nontrivial the reduction should be significantly weaker than the determinant and P_Λ. Since the determinant is **NL**-hard—and so is P_Λ, by the proposition—**L** reductions would suffice for a nontrivial result. Note that **DLOGTIME**-uniform $\mathbf{TC}^0$ is contained in uniform $\mathbf{NC}^1$ which is contained in **L**.

We leave the following as an open question:

Open Question 3.5.3. Is the determinant a p-projection of P_Λ?

Proof of Proposition 3.5.2. For the first claim, let X be an $n \times n$ matrix, and define

$$M(X) = \begin{pmatrix} 1 & & \\ & 0 & X \\ & -X^T & 0 \end{pmatrix}.$$

We claim that $\det(X)^2 = P_\Lambda(M(X))$. The symmetric and antisymmetric part of $M(X)$ are:

$$S(M(X)) = \frac{1}{2}(M(X) + M(X)^T) = \begin{pmatrix} 1 & & \\ & 0 & 0 \\ & 0 & 0 \end{pmatrix}$$

$$A(M(X)) = \frac{1}{2}(M(X) - M(X)^T) = \begin{pmatrix} 0 & & \\ & 0 & X \\ & -X^T & 0 \end{pmatrix}.$$

Thus

$$\begin{aligned} P_\Lambda(M(X)) &= \sum_{i,j} S(M(X))_{ij} \operatorname{Pf}_i(A(M(X))) \operatorname{Pf}_j(A(M(X))) \\ &= 1 \operatorname{Pf}_1(A(M(X))) \operatorname{Pf}_1(A(M(X))) \\ &= \left(\operatorname{Pf} \begin{pmatrix} 0 & X \\ -X^T & 0 \end{pmatrix}\right)^2 \\ &= \left((-1)^{n(n-1)/2} \det(X)\right)^2 \\ &= \det(X)^2 \end{aligned}$$

The penultimate equality is a standard relationship between the Pfaffian and the determinant.

For the second claim, note that over integer inputs, the map $X \mapsto M(X)$ is a reduction from $\det(X)^2$ to P_Λ and each bit of $M(X)$ can clearly be computed from X in **DLOGTIME**-uniform $\mathbf{AC}^0$. All that remains to reduce det to P_Λ over the integers is to compute integer square roots. Hesse, Allender, and Barrington showed that computing integer roots can be done in **DLOGTIME**-uniform $\mathbf{TC}^0$ [139, Corollary 6.5]. □

For n odd, the space of antisymmetric matrices is a large (in fact, maximal) linear subspace of $M_n(\mathbb{C})$ on which the determinant vanishes. Landsberg, Manivel, and Ressayre's example proceeds by taking this linear subspace of the zeroes of the determinant, and then picking a way of approaching this subspace, namely $A(X) + tS(X)$ as $t \to 0$. The discussion preceding Proposition 3.5.1 shows that every point in the orbit closure of determinant either lies in the endomorphism orbit of determinant, or it arises by a similar construction: picking a linear subspace of $M_n(\mathbb{C})$ contained in the zeroes of determinant and picking a way of approaching that subspace.

Note that, although every point in the orbit closure of determinant arises this way, the correspondence between subspaces of the zeroes of determinant and points in the orbit closure is most likely many-to-many.

3.5.2 The relationship between the Mulmuley–Sohoni Conjecture and permanent versus determinant

In this section we use the ideas developed in the previous section to show that the Mulmuley–Sohoni Conjecture 3.3.4 is actually much closer to the original permanent versus determinant conjecture than it may at first appear. The main result is Proposition 3.5.4, which to our knowledge is new.

In approaching a subspace of the zeroes of the determinant, there are two parameters of interest: the order of vanishing—that is, the smallest v such that $\det(A^{(v)}(X))$ is not identically zero—and the order of approximation—that is, the largest D such that $A^{(D)} \colon M_n(\mathbb{C}) \to M_n(\mathbb{C})$ is not the zero map.

The order of vanishing is a notion of how quickly a sequence approaches its limit. If A_t is a sequence of invertible linear maps $M_n(\mathbb{C}) \to M_n(\mathbb{C})$, then $f = \lim_{t\to 0} \det(A_t(X))$ is in the endomorphism orbit if and only if the order of vanishing is 0. Moreover, if the order of vanishing is v, then the corresponding point in the orbit closure is determined entirely by $A^{(0)}, A^{(1)}, \ldots, A^{(v)}$, but not the higher-order terms. That is, two sequence A_t and A'_t yield the same point in the orbit closure if they have the same order of vanishing v and their expansions agree up to the v-th term. However, the higher-order terms may nonetheless be necessary to ensure that A_t is invertible for all t (except possibly finitely many). The number of higher-order terms needed is captured by the other parameter, the order of approximation.

The order of approximation is the notion of "how accurately" a sequence approaches the point in the boundary of the orbit closure, mentioned in Section 3.3.2, or how much accuracy is needed. In the context of matrix multiplication, Bini [53, Corollary 4.1] showed that if a sequence of algorithms approximating matrix multiplication uses k essential multiplications and approaches matrix multiplication with order of approximation $d/3$ then $n \times n$ matrix multiplication can be computed exactly with $(1 + d)k$ essential multiplications. We prove a similar result for the determinant. This result brings the Mulmuley–Sohoni Conjecture 3.3.4 much closer to the original permanent versus determinant conjecture than had previously been known:

Proposition 3.5.4. *The permanent is a p-projection of the determinant if and only if the padded $n \times n$ permanent lies in the orbit closure of $\det_{\mathrm{poly}(n)}$ with order of approximation $\mathrm{poly}(n)$. This holds over any infinite field.*

Before giving the proof, we need an auxiliary definition for the key lemma.

Definition 3.5.5. A family of functions $(f_n(X))$ is *linearly closed* if any linear combination $\sum_{i=1}^{n} \lambda_i f_{n_i}(X^{(i)})$ is a projection of some f_m with $m \leq \mathrm{poly}(n, \max_i n_i)$. Here the $X^{(i)}$ represent independent sets of variables and the λ_i are constants.

Lemma 3.5.6. *Suppose (f_n) is a linearly closed family of polynomials; let p be the polynomial bound in the definition of linear closure for (f_n).*

If g lies in the orbit closure of f_n, $\deg f_n = d$, and g can be approximated by a sequence $g(X) = \lim_{t\to 0} f_n(A_t(X))$ with order of approximation D, then g is a projection of $f_{p(Dd,n)}$ so long as the ground field $\mathbb{F}$ has size strictly greater than Dd.

The proof applies Bini's use of interpolation to any linearly closed family instead of to tensor rank. Tensor rank is not quite linearly closed in the above sense, but see the discussion following the proof.

Proof. We have that $g(X) + te_t(X) = f_n(A_t(X))$ where $A_t(X) = A^{(0)}(X) + tA^{(1)}(X) + \cdots + t^D A^{(D)}(X)$, and $e_t(X)$ is a polynomial in t and the entries of X of degree at most $Dd - 1$. Let $\alpha_1, \alpha_2, \ldots, \alpha_{Dd+1}$ be distinct elements of $\mathbb{F}$. Then the system

$$\begin{pmatrix} 1 & 1 & \cdots & 1 \\ \alpha_1 & \alpha_2 & \ldots & \alpha_{Dd+1} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_1^{Dd} & \alpha_2^{Dd} & \ldots & \alpha_{Dd+1}^{Dd} \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_{nd} \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

has a solution $y_i = \beta_i$ in $\mathbb{F}$, since the determinant of the matrix is the classic Vandermonde determinant, which is equal to $\prod_{i<j}(\alpha_i - \alpha_j)$. By evaluating both sides of $g(X) + te_t(X) = f(A_t(X))$ at $t = \alpha_i$ for each $i = 1, \ldots, Dd$ and summing with weights β_i, we then get

$$\begin{aligned} \sum_{i=1}^{Dd} \beta_i g(X) &+ \sum_{i=1}^{Dd} \beta_i \alpha_i e_{\alpha_i}(X) &&= \sum_{i=1}^{Dd} \beta_i f_n(A_{\alpha_i}(X)) \\ g(X) &+ 0 &&= \sum_{i=1}^{Dd} \beta_i f_n(A_{\alpha_i}(X)). \end{aligned}$$

The right hand side is a projection of $\sum_{i=1}^{Dd} \beta_i f_n(X^{(i)})$. Since f is linearly closed, this function is in turn a projection of $f_{p(Dd,n)}$. □

Proof of Proposition 3.5.4. This theorem in fact holds for any function family in place of the permanent. Malod and Portier showed that the determinant is linearly closed [191, Proposition 7]. Apply Lemma 3.5.6 to $f_n = \det_n$. Then we have $d = \deg \det_n = n$. By hypothesis, the order of approximation D is also polynomial in n; then the theorem follows directly from the lemma. □

We can generalize the notion of linearly closed families to families of *sets* of functions, rather than just families of functions.

Definition 3.5.7. Let $\mathcal{F}_n$ be a set of functions for each n. The family $(\mathcal{F}_n)_{n=1}^{\infty}$ is *linearly closed* if any linear combination $\sum_{i=1}^{n} \lambda_i f_{n_i}$ with $f_{n_i} \in \mathcal{F}_{n_i}$ is a projection of some $f_m \in \mathcal{F}_m$ with $m \leq \text{poly}(n, \max_i n_i)$.

It is clear that if we let $\mathcal{F}_n$ be the set of bilinear functions on n variables with tensor rank at most n, then $\mathcal{F}_n$ is linearly closed in this generalized sense. It is not difficult to see that Lemma 3.5.6 also applies to this generalized notion of linear closure. Hence, with this definition, Lemma 3.5.6 shows that Bini's technique actually applies in a strictly more general setting that includes both Bini's original result on matrix multiplication and our result on permanent versus determinant.

In the case of matrix multiplication, it is known that the exponent of matrix multiplication can be defined either as the rate of polynomial growth of the tensor rank of $n \times n$ matrix multiplication or, equivalently, by using border rank. One might hope that similar techniques could be used to show that the Mulmuley–Sohoni Conjecture 3.3.4 is exactly equivalent to the permanent versus determinant conjecture. In the case of matrix multiplication the result on border rank uses the tensor power trick. Unfortunately, because there is as yet no analog to the tensor power trick for the permanent and determinant, Proposition 3.5.4 is the best we have been able to prove.

CHAPTER 4
MATRIX ISOMORPHISM OF MATRIX LIE ALGEBRAS

This chapter is based on the author's conference paper [129]. The following are notable additions that did not appear there: in Section 4.6 we show how our results have implications for LINEAR EQUIVALENCE to matrix multiplication and iterated matrix multiplication, in addition to determinant (the latter already being in [129]); in Section 4.8 we show that TWISTED CODE EQUIVALENCE Karp-reduces to GRAPH ISOMORPHISM, which we had previously speculated [129]; in Section 4.7 we show how our results resolve the complexity of (abstract) LIE ALGEBRA ISOMORPHISM for a certain class of Lie algebras, as well as discussing the relationship between LIE ALGEBRA ISOMORPHISM and MATRIX ISOMORPHISM OF MATRIX LIE ALGEBRAS more generally; in Section 4.9.1 we discuss the issues involved and ideas for extending our results to other fields; and finally, in Section 4.9.2, we discuss connections between algorithmic problems on Lie algebras and on finite groups.

4.1 Introduction

A *matrix Lie algebra* over a field $\mathbb{F}$ is a set of $n \times n$ matrices closed under the following operations: multiplication by scalars from $\mathbb{F}$, matrix addition, and a multiplication-like operation denoted $[A, B] := AB - BA$. Lie algebras are an important tool in areas as diverse as differential equations [213, 252], particle physics [122], group theory [116, 77, 214], and the Geometric Complexity Theory program [207].

In complexity theory, Kayal [159] has recently used Lie algebras in the so-called affine equivalence problem, which arises in many areas of complexity: factoring integers, permanent versus determinant, matrix multiplication, lower bounds for depth-three circuits, and several more (see [159, §1.1]). Kayal essentially used MATRIX ISOMORPHISM OF LIE ALGEBRAS to give a randomized polynomial-time algorithm to decide when a function can be gotten from the determinant by an invertible linear change of variables. This is the affine equivalence problem for the determinant.

The following are examples of Lie algebras, which should help give their flavor, and introduces some of those Lie algebras on which we prove results, namely abelian, diagonalizable, and (semi-)simple (see Section 2.2.4 for definitions):

1. The collection of all $n \times n$ matrices.

2. The collection of all diagonal $n \times n$ matrices is a Lie algebra of dimension n. Any two diagonal matrices D_1, D_2 commute. Since $D_1 D_2 - D_2 D_1 = 0$ this is a Lie algebra. Any Lie algebra in which all matrices commute is called *abelian*.

3. In fact, *any* collection of diagonal matrices that is closed under taking linear combinations is a Lie algebra, for the same reason as above. Furthermore, if $\mathcal{D}$ is such a Lie algebra, then $A\mathcal{D}A^{-1}$ is as well, since conjugating by A preserves the fact that all the matrices in $\mathcal{D}$ commute. Any Lie algebra conjugate to a set of diagonal matrices is called *diagonalizable*.

4. The collection of all $n \times n$ matrices with trace zero. Since $\operatorname{tr}(AB - BA) = 0$ for any A, B, this is also a Lie algebra. This is an example of a *simple* Lie algebra.

5. The collection of all $2n \times 2n$ matrices of the form $\begin{pmatrix} C & 0 \\ 0 & D \end{pmatrix}$ where C, D are $n \times n$ matrices and $\operatorname{tr} C + \operatorname{tr} D = 0$.

Two matrix Lie algebra $\mathcal{L}_1, \mathcal{L}_2$ are *matrix isomorphic* if there is an invertible matrix A such that $\mathcal{L}_1 = A\mathcal{L}_2 A^{-1}$. Although it was not phrased this way, a randomized reduction from AFFINE EQUIVALENCE for the determinant to MATRIX ISOMORPHISM OF LIE ALGEBRAS that are abstractly isomorphic to example (5) is implicit in Kayal [159]. However, where he uses properties very specific to the Lie algebras associated to permanent and determinant that can be computed using randomization, and does not use the full strength of MATRIX ISOMORPHISM, we are able to instead use a deterministic approach to the more general problem of MATRIX ISOMORPHISM OF LIE ALGEBRAS.

4.1.1 Results

We show that certain cases of MATRIX ISOMORPHISM OF LIE ALGEBRAS are solvable in polynomial time. We also show that extending these cases is difficult, as such an extension is

equivalent to GRAPH ISOMORPHISM in one case and at least as hard as GRAPH ISOMORPHISM in the other case. One of these cases is strong enough to mostly derandomize Kayal's result [159] on AFFINE EQUIVALENCE for the determinant (see Section 4.6 for details).

Our results hold in a computational model with field operations at unit cost. Some of our algorithms and reductions also use an oracle for factoring single-variable polynomials, and these are referred to as f-algorithms and f-reductions. See Section 4.1.2 for details, as well as the implications of f-algorithms for usual algorithms.

For simplicity we state all our results over algebraically closed fields of characteristic zero, though many of our results may extend to finite fields of sufficiently large characteristic under additional natural assumptions; see Section 4.9.1 for an outline of these extensions.

We now give the formal definition of MATRIX ISOMORPHISM OF LIE ALGEBRAS. Since Lie algebras are closed under taking linear combinations, we can give them as input to algorithms by providing a linear basis (see Section 4.3 for more details).

> **Problem:** MATRIX ISOMORPHISM OF MATRIX LIE ALGEBRAS (MATISOLIE)
> **Input:** Two Lie algebras $\mathcal{L}_1$ and $\mathcal{L}_2$ of $n \times n$ matrices, given by basis elements.
> **Output:** An invertible $n \times n$ matrix A such that $A\mathcal{L}_1 A^{-1} = \mathcal{L}_2$, if such A exists, otherwise "the Lie algebras are not matrix isomorphic."

Our main result is an equivalence between MATRIX ISOMORPHISM OF SEMISIMPLE LIE ALGEBRAS and GRAPH ISOMORPHISM, which also yields an efficient algorithm for certain cases of MATRIX ISOMORPHISM OF SEMISIMPLE LIE ALGEBRAS because of the structure of the reduction. In characteristic zero, a Lie algebra is semisimple if and only if it is a direct sum of simple Lie algebras. Simple Lie algebras are one of the two main building blocks of all Lie algebras, and are analogous to simple groups. Example (4) above is a simple Lie algebra; see Section 2.2.4 for the full definition, and the discussion leading up to Remark 2.2.7 for what we mean by "building blocks."

Theorem 4.4.1. *Over algebraically closed fields of characteristic zero,* GRAPH ISOMORPHISM $\leq_m$ SEMISIMPLE MATISOLIE $\leq_m^f$ GRAPH ISOMORPHISM. *In particular, the two problems are f-Karp-equivalent.*

Theorem 4.4.7. *Over algebraically closed fields of characteristic zero,* SEMISIMPLE MATISOLIE *of $n \times n$ matrices can be solved by a* poly(n)*-time f-algorithm*[1] *when the Lie algebras have only $O(\log n)$ simple direct summands.*

Despite the $O(\log n)$ restriction, Theorem 4.4.7 is already strong enough to mostly derandomize Kayal's result (Corollary 4.6.4 below). Also, note that even a single simple Lie algebra can have unbounded dimension, as in example (4), let alone a semisimple one with $O(\log n)$ simple summands. Theorem 4.4.8 gives another class on which MATISOLIE is solvable in polynomial time.

Abelian Lie algebras are the remaining building blocks of all Lie algebras, together with the simple Lie algebras. Recall the definitions of abelian and diagonalizable from examples (2) and (3) above, respectively. Abelian Lie algebras are considered fairly trivial from the Lie-theoretic perspective, so our next observation, that MATRIX ISOMORPHISM OF ABELIAN DIAGONALIZABLE LIE ALGEBRAS is equivalent to LINEAR CODE EQUIVALENCE, may be somewhat surprising at first sight.

A d-dimensional *linear code* of length n over a field $\mathbb{F}$ is a d-dimensional subspace of $\mathbb{F}^n$. Linear codes are represented algorithmically by giving bases for them as subspaces. The symmetric group S_n acts on $\mathbb{F}^n$ by permutation of coordinates: for $\pi \in S_n$ and $\vec{\alpha} = (\alpha_1, \ldots, \alpha_n) \in \mathbb{F}^n$, $\pi \cdot \vec{\alpha} = (\alpha_{\pi(1)}, \ldots, \alpha_{\pi(n)})$. S_n then acts on subspaces $V \subseteq \mathbb{F}^n$ by $\pi \cdot V = \{\pi \cdot v : v \in V\}$. The *linear code equivalence problem* is: given two linear codes C_1, C_2, determine whether there is a permutation $\pi \in S_n$ such that $\pi \cdot C_1 = C_2$.

Proposition 4.2.1. *Over any field,* LINEAR CODE EQUIVALENCE $\leq_m$ ABELIAN DIAGONALIZABLE MATISOLIE $\leq_m^f$ LINEAR CODE EQUIVALENCE. *Lie algebras of $n \times n$ matrices of dimension d are reduced to d-dimensional codes of length n, and vice versa. In particular,* GRAPH ISOMORPHISM $\leq_m$ ABELIAN DIAGONALIZABLE MATISOLIE.

For codes over $\mathbb{F}_2$, LINEAR CODE EQUIVALENCE was known to be at least as hard as GRAPH ISOMORPHISM [216]; we extend their proof to show that LINEAR CODE EQUIVALENCE over *any field* is at least as hard as GRAPH ISOMORPHISM. Combining this proposition with an algorithm of Babai [27, Theorem 7.1], we get the following algorithmic corollary:

1. See Section 4.1.2.

Corollary 4.2.2. *Over any field,* ABELIAN DIAGONALIZABLE MATISOLIE *of $n \times n$ matrices can be solved by a* poly(n)*-time f-algorithm when the Lie algebras have dimension* $O(1)$*.*

We also combine the abelian diagonalizable and semisimple cases together, to show results on MATISOLIE when the Lie algebras are the direct sum of an abelian diagonalizable Lie algebra and a semisimple one:

Corollary 4.5.2. MATISOLIE *of $n \times n$ matrices can be determined in* poly(n) *time when the Lie algebras are a direct sum of an $O(1)$-dimensional abelian diagonalizable Lie algebra and a semisimple Lie algebra with $O(\log n)$ simple direct summands.*

Since abelian is a special case of abelian-plus-semisimple, this more general case is at least as hard as LINEAR CODE EQUIVALENCE, and hence GRAPH ISOMORPHISM, when we drop the quantitative restrictions of the above theorem.

Finally, we mention that there are significant relationships between Lie algebras and associative algebras. A fair amount of work has been done on algorithms for associative algebras (see, for example, Friedl and Rónyai [113], Ivanyos and Rónyai [147], the survey by Rónyai [226] and references therein), though we believe MATRIX ISOMORPHISM OF ASSOCIATIVE ALGEBRAS *per se* has not been previously studied.

4.1.2 A note on finding roots of single-variable polynomials

The reductions *from* GRAPH ISOMORPHISM and LINEAR CODE EQUIVALENCE to various subcases of MATRIX ISOMORPHISM OF LIE ALGEBRAS require no additional assumptions. However, the reductions in the opposite direction seem to require diagonalizing matrices. Diagonalizing a matrix is equivalent to factoring a single-variable polynomial, under polynomial-time many-one reductions over $\mathbb{F}$ (say, using the Blum–Shub–Smale notion of Turing machines over a field [58]). The problem of factoring univariate polynomials over fields is well-studied; its complexity depends on the ground field, which we discuss here. See the survey by Kaltofen [155] for more details and history.

In the settings in which we need to diagonalize a matrix, it will always be under the explicitly stated assumption that the eigenvalues lie in the ground field $\mathbb{F}$. This assumption can be weakened somewhat—to, say, a polynomial-degree extension field of $\mathbb{F}$—but we include this assumption for simplicity. This assumption seems necessary for the use of factoring

polynomials, as a generic polynomial of degree n has a splitting field of degree $n!$ over the ground field.

Matrices over $\mathbb{F}$ whose diagonalizations require a degree $n!$ extension are easily seen to arise in ABELIAN DIAGONALIZABLE MATISOLIE whenever such an extension of $\mathbb{F}$ exists. In contrast, we are not aware of whether such cases can arise in semisimple matrix Lie algebras.

Throughout this paper, an *f-algorithm* over a field $\mathbb{F}$ is an algorithm over $\mathbb{F}$ (in the sense of Blum–Shub–Smale [58]) with an oracle for factoring univariate polynomials over $\mathbb{F}$. Thus once the coefficients of a degree n polynomial have been written down, in n steps the coefficients may be replaced by the roots. We similarly define *f-reductions*. We denote polynomial-time BSS many-one reductions by $\leq_m$ and polynomial-time BSS many-one f-reductions by $\leq_m^f$.

We now discuss known results on the complexity of factoring polynomials.

Although we will not be considering such fields, there are fields where the arithmetic operations are computable but finding roots of polynomials is not computable [271] (see also [115]). Van der Waerden constructed a field in which the problem "Is there an n such that $E(n)$?" for any "effectively computable" predicate $E(n)$ of the natural numbers reduced to the problem of factoring polynomials in that field. A remarkable aspect of this result is that it was published in 1930, before the notion of algorithm or reduction had been formalized by any of Church, Turing, Herbrand, or Gödel.

Over the finite field $\mathbb{F}_q$, polynomials of degree n can be factored deterministically in $\text{poly}(n, q)$ many field operations [50]. However, the Boolean input size is $\text{poly}(n, \log q)$ as field elements may be represented with $\log q$ bits; much work has gone into finding deterministic algorithms that are polynomial in the bit-size of the input, but the best known is still a randomized (Las Vegas-type) algorithm taking an expected $\text{poly}(n, \log q)$ field operations. For example, see Berlekamp [51] and Bach and Shoup [37].

Over $\mathbb{Q}$ or algebraic number fields, polynomials can be factored in polynomial time. The key algorithm for factoring polynomials over $\mathbb{Q}$ is due to Lenstra, Lenstra, and Lovász [181]. The reduction from algebraic number fields to $\mathbb{Q}$ was essentially known to Kronecker [172], with an improvement due to Trager [264] that is presented along with timing analysis in Landau [174]. Similar results were found independently by Chistov and Grigoryev [79] and A. Lenstra [180].

Over fields such as $\mathbb{R}$ and $\mathbb{C}$, roots of course cannot be computed exactly, but efficient approximation algorithms are known [210]. Specifically, given a polynomial f of degree n with complex coefficients whose magnitudes are bounded by 2^m, and an approximation parameter ℓ, one can find by **NC** arithmetic circuits over $\mathbb{C}$ (in terms of $\ell + n + m$) a set of n numbers $z_1, \ldots, z_n$ such that each root of f is within $2^{-\ell}$ of one of the z_k. A similar result can be achieved in the Boolean model, where the z_i are now "rational" complex numbers, that is, of the form $a + ib$ with $a, b \in \mathbb{Q}$, and the cost of the algorithm is measured in terms of bits.

For the purposes for which we use f-algorithms, approximation is good enough as long as distinct roots can be distinguished. This requires that the distance between distinct roots be at least 2^{-n^c} for some polynomial n^c. While we have not been able to guarantee this in general—so the complexity of our f-algorithms over $\mathbb{C}$, once the factoring oracle is removed, remains open—we suspect that the structure of matrix Lie algebras can be leveraged so that the matrices we need to diagonalize always satisfy this property.

To summarize, with the current state of the art, polynomial-time f-algorithms have the following complexities:

- Over $\mathbb{F}_q$, $\text{poly}(n, q)$ deterministic time, or $\text{poly}(n, \log q)$ randomized time.
- Over $\mathbb{Q}$ and algebraic number fields, polynomial time in the bit-length.
- Over $\mathbb{R}$ and $\mathbb{C}$, approximation algorithms exist in both the algebraic and the Boolean models, but the exact complexity of our f-algorithms over $\mathbb{C}$ remains unknown, due to the possibility of distinct roots that are exponentially close together. See Open Question 4.9.3.

4.1.3 Outline

Although it is independent of our main result, we begin in Section 4.2 with the abelian diagonalizable case, as a warm-up that can be understood without any background on Lie algebras. There we also introduce the notion of *weights*, which generalize the notion of eigenvalue from a single matrix to an entire algebra of matrices.

Our main results require more knowledge of Lie algebras; any necessary background that is not presented here can be found in Section 2.2.4. In Section 4.4 we prove our

Main Theorem 4.4.1 and its Corollaries 4.4.7 and 4.4.8 on SEMISIMPLE MATISOLIE over algebraically closed fields of characteristic zero. In Section 4.5 we prove Corollary 4.5.2 on direct sums of abelian diagonalizable and semisimple Lie algebras.

In Section 4.6 we show how to use the above machinery to essentially derandomize Kayal's result on AFFINE EQUIVALENCE for the determinant. We also show how to get similar algorithms for affine equivalence to matrix multiplication and iterated matrix multiplication; we note that although the determinant is equivalent to iterated matrix multiplication from a complexity point of view, they are not affinely equivalent, so results on AFFINE EQUIVALENCE for the two problems are not immediately interchangeable.

In Section 4.8 we show that TWISTED CODE EQUIVALENCE Karp-reduces to GRAPH ISOMORPHISM. This result is not strictly needed for the application to Lie algebras, but it is a natural generalization of the ideas used in a lemma needed for Theorem 4.4.1.

In Section 4.9.1 we discuss how our results might be extended to other fields. In Section 4.9.2 we discuss connections between problems on Lie algebras and problems on finite groups. In the final section, we discuss how close the abelian-plus-semisimple case is to the general case, directions toward the general case, and other future work, including potential ways to solve important special cases of the affine equivalence problem efficiently without having to efficiently solve GRAPH ISOMORPHISM.

4.2 Warm-up: diagonalizable Lie algebras and LINEAR CODE EQUIVALENCE

In this section we show that MATRIX ISOMORPHISM OF ABELIAN DIAGONALIZABLE LIE ALGEBRAS is at least as hard as LINEAR CODE EQUIVALENCE under many-one reductions, and is equivalent to LINEAR CODE EQUIVALENCE under many-one f-reductions. In particular, DIAGONALIZABLE MATRIX ISOMORPHISM OF LIE ALGEBRAS is at least as hard as GRAPH ISOMORPHISM. The reduction to LINEAR CODE EQUIVALENCE allows us to solve ABELIAN DIAGONALIZABLE MATRIX ISOMORPHISM for constant-dimensional Lie algebras by a polynomial-time f-algorithm, using an algorithm of Babai [27, Theorem 7.1] for LINEAR CODE EQUIVALENCE.

We defined linear codes and their equivalences in Section 4.1 just prior to the statement of Proposition 4.2.1 there.

Proposition 4.2.1. *Over any field,* LINEAR CODE EQUIVALENCE $\leq_m$ MATRIX ISOMORPHISM OF ABELIAN DIAGONALIZABLE LIE ALGEBRAS $\leq_m^f$ LINEAR CODE EQUIVALENCE. *Lie algebras of $n \times n$ matrices of dimension d are reduced to d-dimensional codes of length n, and vice versa.*

Before proving this theorem we give some of its consequences.

Corollary 4.2.2. *Over any field,* ABELIAN DIAGONALIZABLE MATISOLIE *of $n \times n$ matrices can be solved by a* $\mathrm{poly}(n)$*-time f-algorithm when the Lie algebras have dimension $O(1)$.*

Proof. Babai (see [27, Theorem 7.1]) showed that, over any field $\mathbb{F}$, equivalence of d-dimensional linear codes of length n reduces to $\binom{n}{d}$ instances of $d \times (n-d)$ EDGE-COLORED BIPARTITE GRAPH ISOMORPHISM. Each such instance can be solved in $\mathrm{poly}(n) \cdot \min\{d!, (n-d)!\}$ time, so when $d = O(1)$ LINEAR CODE EQUIVALENCE can be solved in polynomial time. Using the f-reduction of Proposition 4.2.1, d-dimensional DIAGONALIZABLE MATISOLIE can be solved by an f-algorithm in polynomial time when $d = O(1)$. □

Corollary 4.2.3. GRAPH ISOMORPHISM $\leq_m$ MATRIX ISOMORPHISM OF ABELIAN DIAGONALIZABLE LIE ALGEBRAS.

Proof. Petrank and Roth [216] showed that GRAPH ISOMORPHISM polynomial-time many-one reduces to LINEAR CODE EQUIVALENCE over $\mathbb{F}_2$. Over an arbitrary field we use the same reduction, but an extension of their proof is required, which we give in Lemma 4.2.4 below. Proposition 4.2.1 then shows that MATRIX ISOMORPHISM OF ABELIAN DIAGONALIZABLE LIE ALGEBRAS is at least as hard as GRAPH ISOMORPHISM, under many-one reductions. □

Lemma 4.2.4. GRAPH ISOMORPHISM *polynomial-time many-one reduces to* LINEAR CODE EQUIVALENCE *over any field* $\mathbb{F}$.

Proof. Given a graph G, we construct the generator matrix for a code over $\mathbb{F}$ such that two graphs are isomorphic if and only if the codes are equivalent. Let $M(G) = [I_m | I_m | I_m | D]$

where $m = |E(G)|$, I_m is the $m \times m$ identity matrix, and D is the incidence matrix of G:

$$D_{e,v} = \begin{cases} 1 & \text{if } v \in e \\ 0 & \text{otherwise} \end{cases}$$

The *Hamming weight* of a vector over $\mathbb{F}$ is the number of non-zero entries. The following claim is essentially the crux of Petrank and Roth's argument, but generalized so as to apply over any field.

Claim: up to permutation and scaling of the rows, $M(G)$ is the unique generator matrix of its code which satisfies the following properties:

1. it is a $|E| \times (3|E| + |V|)$ generator matrix;
2. each row has Hamming weight ≤ 5;
3. any nondegenerate linear combination of two or more rows has Hamming weight ≥ 6

A linear combination of k rows is nondegenerate if all k of its coefficients are nonzero.

Proof of claim: First, $M(G)$ satisfies (1)–(3). The only part to check is (3): in the first $3m$ columns, any nondegenerate linear combination of $k \geq 2$ rows will have $3k \geq 6$ nonzero entries. Next, let C denote the code generated by the rows of $M(G)$. By (2) and (3) the rows of $M(G)$ are the *unique* vectors in C (up to scaling) of Hamming weight ≤ 5. Hence if M' is any other generator matrix of C satisfying (1)–(3), its rows must be scaled versions of the rows of $M(G)$ in some order. This proves the claim.

Now, suppose that $M(G_1)$ and $M(G_2)$ generate equivalent codes. Then there is a nonsingular matrix S and a permutation matrix P such that $M(G_1) = SM(G_2)P$. By the claim, $S = \Delta S'$ where Δ is diagonal and S' is a permutation matrix. However, since the first $3|E|$ columns of $M(G_1)$ and $M(G_2)$ only contain $0, 1$-entries, $\Delta = I$. The rest of the proof of the reduction, including the other direction, proceeds exactly as in Petrank and Roth [216]. □

Proof of Proposition 4.2.1. Let $(A_1, \ldots, A_d), (B_1, \ldots, B_d)$ be two spanning sets of diagonalizable Lie algebras over $\mathbb{F}$. Using the root-finding oracle and standard techniques in linear algebra, the A_i can be simultaneously diagonalized in polynomial time, so we may now assume that the A_i are in fact diagonal, rather than merely diagonalizable. Similarly for the B_i. Let $\mathcal{A}$, resp. $\mathcal{B}$, denote the Lie algebras spanned by the A_i, resp. B_i.

Claim: If $\mathcal{A}$ and $\mathcal{B}$ are diagonal, then they are matrix isomorphic if and only if they are matrix isomorphic via a permutation matrix.

Here is how the result follows from the claim. By "flattening out" the entries of the diagonal matrices into "row" vectors the claim shows that d-dimensional MATRIX ISOMORPHISM for *diagonal* Lie algebras of $n \times n$ matrices is polynomial-time many-one equivalent to d-dimensional CODE EQUIVALENCE for codes of length n. The root-finding oracle is only needed for one direction, to diagonalize the Lie algebras. Thus the claim will complete the proof of the theorem.

Proof of claim: Suppose $C\mathcal{A}C^{-1} = \mathcal{B}$. Since $\mathcal{A}$ and $\mathcal{B}$ are both diagonal, C must preserve the eigenspaces of every matrix in $\mathcal{A}$. The formalization of this notion will allow us to prove our claim. Let $\lambda_i \colon \mathcal{A} \to \mathbb{F}$ be the linear function $\lambda_i(A) = A_{ii}$. We can think of λ_i as a "simultaneous eigenvalue for the space $\mathcal{A}$ of matrices," generalizing the notion of an eigenvalue of a single matrix. Such functions are called *weights* in Lie theory, and they will play a significant role here and in the case of semisimple Lie algebras. Analogous to an eigenspace corresponding to an eigenvalue, there are *weight spaces* corresponding to weights. Namely, if $\lambda \colon \mathcal{A} \to \mathbb{F}$ is a weight, the corresponding weight space is

$$V_\lambda(\mathcal{A}) := \{v \in \mathbb{F}^n : Av = \lambda(A)v \text{ for all } A \in \mathcal{A}\}$$

It is these weight spaces that C must preserve in order for $C\mathcal{A}C^{-1}$ to be diagonal. For example, if every weight space is 1-dimensional—or equivalently, if for every pair of indices $1 \le i < j \le n$ there is some matrix $A \in \mathcal{A}$ with $A_{ii} \neq A_{jj}$—then C must be the product of a permutation matrix and a diagonal matrix. Since diagonal matrices commute, conjugating $\mathcal{A}$ by a diagonal matrix has no effect, and we may discard it; hence C may be taken to be a permutation matrix.

More generally, C may send $v \in V_{\lambda_1}$ into V_{λ_2} if and only if $CV_{\lambda_1} = V_{\lambda_2}$. Within each weight space, C may act in an arbitrary invertible manner. In other words, C is composed of invertible blocks of dimension $\dim V_{\lambda_i}$, the pattern in which these blocks appear is a permutation, and that permutation may send $i \mapsto j$ if and only if $\dim V_{\lambda_i} = \dim V_{\lambda_j}$. However, if C' has the same permutation pattern as C but all the blocks in C' are the identity, then $C\mathcal{A}C^{-1} = C'\mathcal{A}C'^{-1}$. Hence, without loss of generality, we may take C to be a permutation matrix, proving the claim. □

4.3 Basic algorithms for Lie algebras and their representations

In this section we describe the basic algorithms for working with Lie algebras and their representations, over algebraically closed fields of characteristic zero. Many of these algorithms are present in De Graaf's book [97], which is an invaluable resource in this regard. However, De Graaf explicitly [97, p. vi] omits any complexity analysis, so we re-present the necessary algorithms here in order to analyze their complexity, except when this has already been done in the literature. We refer to the theorems in De Graaf [97] for the proofs of correctness, as to do otherwise would result in reproducing much of the content of De Graaf's book.

4.3.1 Describing Lie algebras and representations as input to algorithms

An abstract Lie algebra is specified in an algorithm by giving a basis for it as a vector space, say $v_1, \ldots, v_d$, and its *structure constants* $c_{ij}^{(k)}$:

$$[v_i, v_j] = \sum_{k=1}^{n} c_{ij}^{(k)} v_k.$$

Because of the bilinearity of the bracket, the structure constants are enough to determine the value of the bracket on any elements of the Lie algebra: $[\sum \alpha_i v_i, \sum \beta_j v_j] = \sum_{ijk} \alpha_i \beta_j c_{ij}^{(k)} v_k$. Each of the axioms of a Lie algebra translates into a condition on the structure constants, for example, skew-symmetry is equivalent to $c_{ij}^{(k)} = -c_{ji}^{(k)}$ for all i, j, k.

A linear map between Lie algebras $\varphi\colon \mathcal{L} \to \mathcal{L}'$ is specified by giving, for each basis element v_i of $\mathcal{L}'$, the coefficients of the image of v_i in the basis $\{v'_1, \ldots, v'_d\}$ of $\mathcal{L}'$. In other words, if $\varphi(v_i) = \sum_j a_{ij} v'_j$, then φ is specified by the constants a_{ij}, as is typical for linear maps in general. In particular, a representation $\rho\colon \mathcal{L} \to M_n(\mathbb{F})$ is a linear map between Lie algebras. If we take the standard basis for $M_n(\mathbb{F})$, then the above corresponds to defining ρ by giving, for each basis element v_i of $\mathcal{L}$, a matrix $A_i \in M_n(\mathbb{F})$ such that $\rho(v_i) = A_i$.

Given a matrix Lie algebra $\mathcal{L} \subseteq M_n(\mathbb{F})$, we may compute its structure constants and thus get an abstract isomorphic copy $\mathcal{L}'$ of $\mathcal{L}$, with $\mathcal{L}'$ specified by structure constants, as follows. Suppose the matrix Lie algebra is given to us by a basis of matrices $\{A_1, \ldots, A_d\}$. For each pair $i \neq j$, compute the matrix $[A_i, A_j] = A_i A_j - A_j A_i$ and write it as a linear combination of the A_k. This involves multiplying matrices and then solving a system of linear equations.

Then the structure constants of $\mathcal{L}$ are given by the $c_{ij}^{(k)}$ defined by $[A_i, A_j] = \sum_k c_{ij}^{(k)} A_k$. Now, the abstract copy $\mathcal{L}'$ of $\mathcal{L}$ can be defined simply as a d-dimensional vector space with standard basis $v_1 = (1, 0, \ldots, 0), v_2 = (0, 1, 0, \ldots, 0), \ldots, v_d = (0, 0, \ldots, 1)$, and structure coefficients $c_{ij}^{(k)}$. The isomorphism between $\mathcal{L}$ and $\mathcal{L}'$ is then given by $A_i \leftrightarrow v_i$.

4.3.2 Abstract isomorphism of semisimple Lie algebras

In this section we consider semisimple Lie algebras over algebraically closed fields of characteristic zero.

Theorem 4.3.1 (De Graaf [97])**.** *Let $\mathcal{L}$ be a semisimple Lie algebra over an algebraically closed field of characteristic zero, given by structure constants.*

Then there is a polynomial-time f-algorithm that decomposes $\mathcal{L}$ into its direct sum decomposition $\mathcal{L} = \bigoplus_i \mathcal{L}_i$ where each $\mathcal{L}_i$ is simple, and identifies the type (in the Cartan–Killing classification) of each $\mathcal{L}_i$.

In particular, given two such Lie algebras by structure constants, there is a polynomial-time f-algorithm that determines whether they are isomorphic, and finds an isomorphism if one exists.

Proof. The proof proceeds as in De Graaf [97].

Phase I: compute a split Cartan subalgebra. De Graaf, Ivanyos, and Rónyai [94] show that a Cartan subalgebra can be computed in polynomial time in the bit-length over algebraic number fields (see also the algorithm "CartanSubalgebraBigField" in De Graaf [97, p. 67]). Their algorithm is arithmetic in nature, and clearly works in polynomially many arithmetic steps over any field of characteristic zero. Since we have assumed the ground field is algebraically closed, all Cartan subalgebras are split (recall the definition of split Cartan subalgebra from Section 2.2.4).

Phase II: decompose $\mathcal{L}$ into root spaces. Suppose $\mathcal{H}$ is the split Cartan subalgebra computed in Phase I above. Let $\{h_1, \ldots, h_r\}$ be a basis for $\mathcal{H}$. Using the root-finding oracle, find a basis of $\{h'_1, \ldots, h'_r, x'_1, \ldots, x'_k\}$ for which $\{h'_1, \ldots, h'_r\}$ is a basis for $\mathcal{H}$ and $\mathrm{ad}_{\mathcal{L}} h'_1$ is diagonal in this basis. If $\mathrm{ad}_{\mathcal{L}} h_1$ had any repeated eigenvalues, it is possible that the other $\mathrm{ad}_{\mathcal{L}} h_i$ are not yet diagonal within each eigenspace of $\mathrm{ad}_{\mathcal{L}} h_1$. Repeat this procedure recursively on each $\mathrm{ad}_{\mathcal{L}} h_i$ in turn until they are all diagonal. Then we have the Cartan

decomposition $\mathcal{L} = \mathcal{H} \bigoplus_\alpha \mathcal{L}_\alpha$ where each $\mathcal{L}_\alpha$ is the root space relative to $\mathcal{H}$ with root α. That is, $x \in \mathcal{L}_\alpha$ if and only if $[h, x] = \alpha(h)x$ for all $h \in \mathcal{H}$.

Phase III: decompose $\mathcal{L}$ into direct summands. This follows the algorithm "DirectSumDecomposition" from De Graaf [97, p. 129], which in turn is taken from a paper of De Graaf's [95]. Remarks 8 and 9 of that paper show that the algorithm works for semisimple Lie algebras in characteristic zero, and that the algorithm runs in polynomial time with a root-finding oracle.

Phase IV: compute the type of each simple summand. De Graaf [95, Section 4] shows how to compute the type, in the Cartan–Killing classification, of a simple Lie algebra. In De Graaf [95], a very nice method is used in order to give a polynomial-time algorithm—in the bit-length, Boolean model—for isomorphism of semisimple Lie algebras over $\overline{\mathbb{Q}}$ with structure constants in $\mathbb{Q}$. There, he reduces coefficients modulo p for a prime p with certain properties, in order to avoid the blow-up that may occur from root-finding. However, in the algebraic model with a root-finding oracle we don't need to go to such lengths, and indeed if the structure constants are not in $\mathbb{Q}$ then such an algorithm does not even seem to make sense (what does it mean to reduce a complex number modulo p?). Instead, once we have our Cartan decomposition of $\mathcal{L}$, it is straightforward linear algebra to compute the Cartan matrix, the root system, and canonical generators.

Finally, the algorithm "CanonicalGenerators" of De Graaf [97, p. 182] constructs a canonical generating set for a semisimple Lie algebra, by solving a few linear equations, once the Cartan decomposition has been found. The isomorphism between two isomorphic semisimple Lie algebras is given by mapping the canonical generating set of one to the canonical generating set of the other. □

4.3.3 Equivalence and decomposition of representations

The results in this section are surely not new; although they are straightforward algorithmic applications of known results on the representations of semisimple Lie algebras, we have been unable to find explicit references to these algorithmic consequences, so we include their proofs here. Later, we use Proposition 4.3.2 to solve the decision version of MATRIX ISOMORPHISM OF LIE ALGEBRAS for semisimple and completely reducible Lie algebras.

Proposition 4.3.2. *Let $\mathbb{F}$ be an algebraically closed field of characteristic zero.*

Given a semisimple matrix Lie algebra $\mathcal{L} \subseteq M_n(\mathbb{F})$, there is a polynomial-time $BSS_{\mathbb{F}}$ f-algorithm to determine the types—equivalently, highest weights—of the representations appearing in $\mathcal{L}$ together with their multiplicities.

Proof. $\mathcal{L}$ is given to us by a spanning set of matrices $\{X_1, \ldots, X_d\}$. Compute a Cartan subalgebra $\mathcal{H}$ of $\mathcal{L}$, compute a matrix A_0 simultaneously diagonalizing every $h \in H$, and compute a canonical set of generators of $\mathcal{L}$, using the algorithms of De Graaf, as in Theorem 4.3.1. Conjugate $\mathcal{L}$ by A_0, so that $\mathcal{H}$ becomes diagonal, which we assume hereafter.

Compute a set of positive simple roots, with their corresponding matrices in $\mathcal{L}$. Let W be the intersection of the kernels of these root matrices.

By the standard theory of highest weights (see, for example, Jacobson [150] or Fulton and Harris [116]), W consist of all the highest weight vectors in $\mathbb{F}^n$. As $\mathcal{H}$ is diagonal, we can choose a basis of W consisting of weight vectors for $\mathcal{H}$. To do this, intersect W with each weight space V_λ in turn, for all weights λ that are present. Call the resulting space W_λ, so that $W = \bigoplus_\lambda W_\lambda$. Then the irreducible representation λ appears in $\mathcal{L}$ with multiplicity $\dim W_\lambda$. □

4.4 Semisimple Lie algebras and GRAPH ISOMORPHISM

Theorem 4.4.1. *Over algebraically closed fields of characteristic zero,* GRAPH ISOMORPHISM $\leq_m$ MATRIX ISOMORPHISM OF SEMISIMPLE LIE ALGEBRAS $\leq_m^f$ GRAPH ISOMORPHISM.

Proof. We break the proof into four lemmas. By Lemma 4.4.3, MATRIX ISOMORPHISM of semisimple Lie algebras is equivalent to deciding whether two completely reducible representations of a semisimple Lie algebra are equivalent up to outer automorphism. By Lemma 4.4.4, the latter problem reduces to a special case of TWISTED CODE EQUIVALENCE WITH MULTIPLICITIES, which we refer to as PROBLEM A. Finally, Lemma 4.4.5 reduces PROBLEM A to GRAPH ISOMORPHISM, and Lemma 4.4.6 reduces GRAPH ISOMORPHISM to MATRIX ISOMORPHISM OF SEMISIMPLE LIE ALGEBRAS.

The polynomial factorization oracle is only used in two places, both times to diagonalize a Cartan subalgebra:

- in Lemma 4.4.3, to find an abstract isomorphism between semisimple Lie algebras in order to reduce to OUTER EQUIVALENCE OF REPRESENTATIONS; and
- in Lemma 4.4.4, via Proposition 4.3.2, to find the highest weight spaces of a representation.

In both of these cases, if we were given a Chevalley basis of the Lie algebra and its representation we would not need to solve polynomial factorization. A Chevalley basis is a particularly nice basis; for our purposes the properties we need are that it consists of a basis of a Cartan subalgebra, together with a basis of the rest of the Lie algebra such that the Cartan subalgebra is diagonal in this basis in the adjoint action. Similarly, the relevant property of a Chevalley basis of the representation is that it is a basis of $\mathbb{F}^n$ such that the matrices in the given Cartan subalgebra act diagonally on that basis. □

Lemma 4.4.2. *Let $\mathcal{C}$ be a class of Lie algebras over any field, and let* LIEISO($\mathcal{C}$) *denote the problem of finding isomorphism between abstract Lie algebras in $\mathcal{C}$. Then* MATRIX ISOMORPHISM OF LIE ALGEBRAS *for Lie algebras in $\mathcal{C}$ is $\mathbf{P}^{LieIso(\mathcal{C})}$-equivalent[2] to—nearly just a restatement of—the following problem:*

> ***Problem:*** *Equivalence of representations up to automorphism*
> ***Input:*** *Two faithful representations $\rho_1, \rho_2 \colon \mathcal{L} \to M_n$ of an abstract Lie algebra $\mathcal{L}$ from the class $\mathcal{C}$.*
> ***Output:*** *An automorphism $\alpha \in \mathrm{Aut}(\mathcal{L})$ such that ρ_1^α is equivalent to ρ_2, or "the two representations are not equivalent up to automorphism."*

Proof. Suppose $\mathcal{L}_1, \mathcal{L}_2 \subseteq M_n(\mathbb{F})$ are two matrix Lie algebras over any field $\mathbb{F}$. Their structure constants are easily computed as in Section 4.3.1, and then, by assumption, we can determine whether $\mathcal{L}_1$ and $\mathcal{L}_2$ are isomorphic as abstract Lie algebras, and if so, find an isomorphism $f \colon \mathcal{L}_1 \to \mathcal{L}_2$. If not, they are not matrix isomorphic. By computing the structure constants

2. The reduction from EQUIVALENCE OF REPRESENTATIONS UP TO AUTOMORPHISM to MATRIX ISOMORPHISM is polynomial-time many-one, and polynomial-time in the bit-length whenever that notion makes sense, for example over finite fields, $\mathbb{Q}$, and number fields. The reduction in the opposite direction is as efficient as finding isomorphisms between abstract Lie algebras in $\mathcal{C}$. More precisely, the reduction from MATRIX ISOMORPHISM to EQUIVALENCE OF REPRESENTATIONS UP TO AUTOMORPHISM is polynomial-time many-one with an oracle for finding isomorphisms between abstract Lie algebras in $\mathcal{C}$.

as in Section 4.3.1, we get an abstract Lie algebra $\mathcal{L}$ together with an isomorphism $\rho_1 \colon \mathcal{L} \to \mathcal{L}_1$. Let $\rho_2 = \rho_1 \circ f$; then ρ_2 is an isomorphism $\mathcal{L} \to \mathcal{L}_2$, as abstract Lie algebras. Since ρ_1 and ρ_2 are abstract isomorphisms they are injective, and since $\mathcal{L}_i \subseteq M_n(\mathbb{F})$, each ρ_i is in fact a faithful representation of $\mathcal{L}$. We claim that the ρ_i are equivalent up to an automorphism of $\mathcal{L}$ if and only if the $\mathcal{L}_i$ are matrix isomorphic.

Suppose $\mathcal{L}_2 = A\mathcal{L}_1 A^{-1}$ for some invertible matrix A. Let $c_A \colon M_n(\mathbb{F}) \to M_n(\mathbb{F})$ be defined by $c_A(X) = AXA^{-1}$. Then $\alpha = \rho_2^{-1} \circ c_A \circ \rho_1$ is a map from $\mathcal{L}$ to $\mathcal{L}$ (see Figure 4.1). Since the ρ_i are isomorphisms, and $c_A|_{\mathcal{L}_1} \colon \mathcal{L}_1 \to \mathcal{L}_2$ is an isomorphism, the composition α

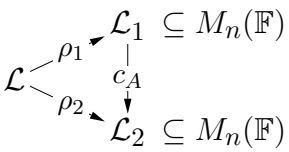


Figure 4.1: Two matrix isomorphic faithful representations of a Lie algebra $\mathcal{L}$ yield an automorphism of $\mathcal{L}$ by going around the triangle clockwise: $\rho_2^{-1} \circ c_A \circ \rho_1$.

is an automorphism of $\mathcal{L}$. Then $\rho_2 \circ \alpha = \rho_2 \circ \rho_2^{-1} \circ c_A \circ \rho_1 = c_A \circ \rho_1$, which is by definition equivalent to ρ_1.

Conversely, suppose $\rho_1 \circ \alpha$ is equivalent to ρ_2 for some automorphism α. Then there is an invertible matrix A such that $\rho_2 = c_A \circ \rho_1 \circ \alpha$. Then we have

$$\mathcal{L}_2 = \mathrm{im}(\rho_2) = \mathrm{im}(c_A \circ \rho_1 \circ \alpha) = c_A(\mathrm{im}(\rho_1 \circ \alpha)).$$

Since α is an automorphism it is onto, so $\mathrm{im}(\rho_1 \circ \alpha) = \mathrm{im}(\rho_1) = \mathcal{L}_1$, and we have $\mathcal{L}_2 = A\mathcal{L}_1 A^{-1}$.

The preceding argument gives a reduction from MATRIX ISOMORPHISM OF LIE ALGEBRAS to EQUIVALENCE OF REPRESENTATIONS UP TO AUTOMORPHISM. The reduction in the other direction is as follows: suppose $\mathcal{L}$ is a Lie algebra and $\rho_1, \rho_2 \colon \mathcal{L} \to M_n(\mathbb{F})$ are two faithful representations. We reduce this to the instance of MATRIX ISOMORPHISM OF LIE ALGEBRAS given by $\mathcal{L}_i = \mathrm{im}(\rho_i)$ $(i = 1, 2)$. The proof that this is a reduction is identical to the proof above. □

Lemma 4.4.3 (De Graaf[3]). *Let $\mathbb{F}$ be an algebraically closed field of characteristic zero. Then* MATRIX ISOMORPHISM OF SEMISIMPLE LIE ALGEBRAS *over $\mathbb{F}$ is f-equivalent to—nearly just a restatement of—the following problem:*

> ***Problem:*** *Outer equivalence of Lie algebra representations*
> ***Input:*** *Two faithful representations $\rho_1, \rho_2 \colon \mathcal{L} \to M_n(\mathbb{F})$ of a semisimple (abstract) Lie algebra $\mathcal{L}$.*
> ***Output:*** *An outer automorphism $\alpha \in \mathrm{Out}(\mathcal{L})$ such that ρ_1^α is equivalent to ρ_2, or "the two representations are not equivalent up to automorphism."*

Proof. For semisimple Lie algebras, abstract isomorphisms can be found by a polynomial-time f-algorithm (see Theorem 4.3.1). By the previous lemma, we only have to reduce the problem of EQUIVALENCE OF REPRESENTATIONS UP TO AUTOMORPHISM to the above problem, which deals explicitly with *outer* automorphisms. The same reductions from the previous lemma apply, but by using more structural results we can show that only outer automorphisms need to be considered.

Suppose $\mathcal{L}_2 = A\mathcal{L}_1A^{-1}$ for some invertible matrix A. Let $c_A \colon M_n \to M_n$ be defined by $c_A(X) = AXA^{-1}$. As in the previous lemma $\alpha = \rho_2^{-1} \circ c_A \circ \rho_1$ is an automorphism of $\mathcal{L}$ such that $\rho_2 \circ \alpha$ is equivalent to ρ_1. By a standard result (see, for example, De Graaf [97, Lemma 8.5.1]), inner automorphisms of a semisimple Lie algebra extend to equivalenes of its representations. In particular, $\rho_2 \circ \alpha$ only depends on the image of α in the outer automorphism group $\overline{\alpha}$. Thus $\rho_2^{\overline{\alpha}}$ is equivalent to ρ_1.

Conversely, suppose $\rho_1^{\overline{\alpha}}$ is equivalent to ρ_2 for some outer automorphism $\overline{\alpha}$. Let $\alpha \in \mathrm{Aut}(\mathcal{L})$ be a representative of $\overline{\alpha}$; the action of α on ρ_2 is well-defined, by the result mentioned previously (see De Graaf [97, Lemma 8.5.1]). As in the previous lemma, there is an invertible matrix A such that $\rho_2 = c_A \circ \rho_1 \circ \alpha$, and then $\mathcal{L}_2 = A\mathcal{L}_1A^{-1}$.

The reduction in the opposite direction requires nothing further than in the previous lemma. □

3. This lemma is essentially present in De Graaf's book [97], especially the content leading up to the discussion at the end of his Section 8.5. However, De Graaf's discussion is presented in terms of weights and the choice of Cartan subalgebra, whereas the aspect we wish to highlight requires no mention of these topics, and can be explained by completely elementary means.

Lemma 4.4.4. *Over an algebraically closed field of characteristic zero, outer equivalence of semisimple Lie algebra representations f-reduces to the following problem:*

> ***Problem:*** *Problem A*
>
> ***Input:*** *Two $r \times s$ integer matrices M_1, M_2; a partition of the columns into consecutive ranges $[1, \ldots, k_1], [k_1 + 1, \ldots, k_1 + k_2], \ldots [k_1 + \cdots + k_{t-1} + 1, \ldots, s]$; for each range, a group G_ℓ acting on the integers appearing in the corresponding columns, where each G_ℓ is abstractly isomorphic to one of: 1, S_2, or S_3.*
>
> ***Output:*** *A permutation $\pi \in S_r$, a permutation $\sigma \in S_{k_1} \times S_{k_2} \times \cdots \times S_{k_t}$, and for each column an element g_j in the group G_ℓ associated to that column range, such that for all i, j, $M_1(i, j) = g_j(M_2(\pi(i), \sigma(j)))$, or "the matrices are not equivalent." In other words, after applying π to the rows, σ to the columns, and each g_j to the values of the entries in the j-th column, M_1 and M_2 become equal.*

Proof idea. The columns of M_i correspond to the simple summands of the matrix Lie algebra $\mathcal{L}_i$ $(i = 1, 2)$. The rows correspond to the irreducible representations appearing in $\mathcal{L}_i$. Each irreducible representation of $\mathcal{L}_i$ is a tensor product of irreducible representations of the simple summands of $\mathcal{L}_i$ (Proposition 2.2.9). Each entry of M_i encodes the corresponding irreducible representation of a simple summand of $\mathcal{L}_i$. The irreducible representations of a simple Lie algebra can be described by combinatorial objects, which in turn we encode as integers. The columns of M_i are partitioned according to the abstract isomorphism type of the simple summands. The groups acting on the integers in M_i are in fact the outer automorphism groups of the simple summands, acting on the irreducible representations of the simple summands. For example, M_1 might look like:

	$\mathcal{L}_{1,1}\oplus$	$\cdots$	$\oplus\mathcal{L}_{1,k_1}\oplus$	$\mathcal{L}_{2,1}\oplus$	$\cdots$	$\oplus\mathcal{L}_{2,k_2}\oplus$	$\cdots$	$\oplus\mathcal{L}_{t,k_t}$
$\rho_{1,1}$	0	$\cdots$	7	3	$\cdots$	0	$\cdots$	1
$\rho_{1,2}$	1	$\cdots$	0	4	$\cdots$	12	$\cdots$	0
$\vdots$	$\vdots$		$\vdots$	$\vdots$		$\vdots$		$\vdots$
$\rho_{1,r}$	0	$\cdots$	1	15	$\cdots$	6	$\cdots$	8

where each $\mathcal{L}_{i,j}$ is simple, and $\mathcal{L}_{i,j}$ and $\mathcal{L}_{i',j'}$ are abstractly isomorphic if and only if $i = i'$. □

Proof. Let $\mathcal{L}$ be a semisimple Lie algebra, and let $\rho_1, \rho_2 \colon \mathcal{L} \to M_n(\mathbb{F})$ be two faithful representations of $\mathcal{L}$. Compute the direct sum decomposition of $\mathcal{L}$ into simple ideals, as in Theorem 4.3.1; suppose it is $\mathcal{L} = \mathcal{L}_{1,1} \oplus \cdots \oplus \mathcal{L}_{1,k_1} \oplus \mathcal{L}_{2,1} \oplus \cdots \oplus \mathcal{L}_{2,k_2} \oplus \cdots \oplus \mathcal{L}_{t,k_t}$ where each $\mathcal{L}_{i,j}$ is a simple summand of $\mathcal{L}$, and the $\mathcal{L}_{i,j}$ are grouped by isomorphism type, so that $\mathcal{L}_{i_1,j_1}$ and $\mathcal{L}_{i_2,j_2}$ are isomorphic if and only if $i_1 = i_2$. For each i, let $\mathcal{L}_i$ be a simple Lie algebra isomorphic to $\mathcal{L}_{i,j}$ for all j.

To each ρ_i we will associate a matrix M_i, as well as the other data necessary for Problem A. The columns correspond to the direct summands $\mathcal{L}_{i,j}$, and the column partition is along the isomorphism types of the summands.

Next, we define the permutation groups G_ℓ. To each simple type $\mathcal{L}_\ell$, we fix once and for all an encoding of its representations as integers; both the encoding and decoding should be polynomial-time. That this can be done follows from the standard description of the representations of the simple Lie algebras. The integer 0 will always stand for the (trivial) zero representation. The permutation action of $\mathrm{Out}(\mathcal{L}_\ell)$ on the representations of $\mathcal{L}_\ell$, encoded as integers, can be easily computed, as follows. Given $\overline{\alpha} \in \mathrm{Out}(\mathcal{L}_\ell)$ and an integer, convert it to the corresponding representation as above. This representation is a linear map $\mathcal{L}_\ell \to M_n(\mathbb{F})$ for some n. Pre-compose this map with a representative $\alpha \in \mathrm{Aut}(\mathcal{L}_\ell)$ of $\overline{\alpha}$; this can be done because the outer automorphisms of all simple Lie algebras are known explicitly and are easy to compute (see Section 2.2.4). For example, the unique outer automorphisms of $\mathfrak{sl}_n$, the trace zero matrices, is given by the map $A \mapsto -A^T$. The outer automorphism groups of simple Lie algebras are all trivial, S_2, or S_3. Finally, convert this new, "twisted-by-α" representation back to an integer. The group G_ℓ associated to the ℓ-th isomorphism type (=ℓ-th column grouping) is then $\mathrm{Out}(\mathcal{L}_\ell)$, and the action on the integers is the action just described.

Finally, we describe the rows and the entries of the matrices M_i. Decompose the representations ρ_i into their direct sum decompositions $\rho_i = \rho_{i,1} \oplus \cdots \oplus \rho_{i,r}$, where each $\rho_{i,r}$ is an irreducible representation of $\mathcal{L}$. The q-th row of M_i corresponds to the irreducible representations $\rho_{i,q}$. An irreducible representation of a direct sum of Lie algebras is the tensor product of the restrictions to each summand (see Proposition 2.2.9). Hence, the representation $\rho_{i,q}$ is specified by a representation of each summand $\mathcal{L}_{\cdot,\cdot}$, that is, an integer in each column.

Since the outer automorphism group of $\mathcal{L}$ is

$$\prod_{i=1}^{t} \mathrm{Out}(\mathcal{L}_i) \wr S_{k_i} = \left(\prod_{i=1}^{t} \mathrm{Out}(\mathcal{L}_i)^{k_i} \right) \rtimes \left(\prod_{i=1}^{t} S_{k_i} \right),$$

the representations ρ_1, ρ_2 are equivalent up to an outer automorphism if and only if there is a permutation of the columns (=direct summands of the Lie algebra), for each column an element $g_\ell \in G_\ell$ (=an outer automorphism of each direct summand), and a permutation of the rows (=irreducible constituents of ρ_i) that will make M_1 equal to M_2. Conversely, any such equivalence of M_1 and M_2 according to Problem A corresponds to an outer automorphism of $\mathcal{L}$ that makes ρ_1 and ρ_2 equivalent. □

Lemma 4.4.5. Problem A $\leq_m$ graph isomorphism.

Proof of Lemma 4.4.5. First, if the permutation groups G_ℓ are all trivial, then we can take each M_i as the bipartite adjacency matrix of a vertex-colored and edge-colored bipartite graph. The vertices corresponding to the columns are colored according to their part in the column partition; we refer to these vertices as column-vertices. The edges are colored by the integer entries of each M_i. It is clear that the M_i are equivalent if and only if the corresponding vertex- and edge-colored bipartite graphs are bipartite-color-isomorphic, that is, isomorphic by an isomorphism which preserves the two parts of the bipartition and preserves each color class of vertices and each color class of edges.

To handle the permutation groups G_i we make one additional step in the reduction. Since there is one G_i for each column i, we must encode its action on the edge-labels incident to the column-vertex i. To do this we add a "color palette" gadget for each column-vertex, which will encode both the edge-labels, as well as enforcing the action of G_i on these labels. That is, the color palette will be such that the way automorphisms of the resulting graph act on the encoding of the edge-labels is exactly the same as G_i acts on them.

To encode the edge-labels with the color palette, we divide each edge by a new vertex, and attach this new vertex to the vertex of the color palette which encodes the appropriate edge color. We only need color palettes capable of encoding permutation actions of the trivial group, S_2, and S_3.

G_i trivial. If some G_i is trivial, the corresponding color palette is simply a line of vertices with a marked vertex at the end. The marked vertex prevents reversing the order of the line,

and the different vertices in the line encode the different edge labels on the edges incident to column-vertex i.

$G_i \cong S_2$. S_2 has two possible orbit types (=transitive actions): a single fixed point, or an orbit of size two. The color palette is the disjoint union of two graphs corresponding to the two possible orbit types. Each of these graphs has its own marked vertex at the end. One of these two graphs is simply a line as in the previous case: the vertices of this line correspond to those edge-labels that are fixed by the action of S_2. The other graph is the disjoint union of two lines, each of which is joined at the end to the marked vertex. The action of S_2 swaps the i-th vertex of one of these lines with the i-th vertex of the other. This enforces that the action of the edge group G_i either swaps all of the edge labels (that is, via the nontrivial element of S_2) or none of them.

For the sake of the next case, it is useful to think of this color palette as gluing together in a line multiple copies of the "color gadget" consisting of two disconnected vertices.

$G_i \cong S_3$. S_3 has four orbit types: 1) the trivial action, 2) the action on two points by which odd permutations swap the points and even permutations fix them, 3) the natural action of S_3 on three points, and 4) the regular action of S_3 on itself (6 points). However, these last three orbit types must be linked, since if an element of S_3 swaps two points according to (2), it must also have some action according to (3) and (4). Thus the color palette in this case is the disjoint union of two palettes: the trivial, line palette as before, and a more complicated palette encoding the actions (2)–(4).

This more complicated palette is given by a "color gadget," multiple copies of which are glued together in a line, as in all the other cases. The color gadget is as in Figure 4.2.

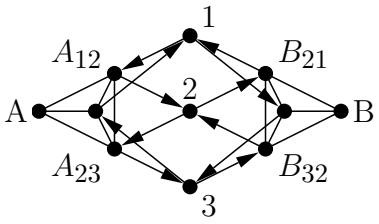


Figure 4.2: Color gadget encoding the action of the groups acting on the columns. In TWISTED CODE EQUIVALENCE these are the twisting groups; from the Lie algebra point of view these are the outer automorphism groups of the simple direct summands.

Multiple copies of this color gadget are glued together along three lines, one connecting the "1" vertices, one connecting the "2" vertices, and one connecting the "3" vertices. At one end of these lines, every vertex in the color gadget is connected to a new marked vertex, to prevent the line from being swapped end-to-end.

A set of edge colors corresponding to an orbit of type (2) is encoded by the A and B vertices.

A set of edge colors corresponding to an orbit of type (3) is encoded by the vertices 1, 2, and 3.

A set of edge colors corresponding to an orbit of type (4) is encoded by the vertices A_{12}, A_{23}, A_{31}, B_{21}, B_{32}, and B_{13}. A_{31} and B_{13} are not labeled in the diagram due to space, but there are directed edges $3 \to A_{31} \to 1$ and $1 \to B_{13} \to 3$.

It remains to show that this color gadget really works as desired. Let us examine the automorphisms of the color gadget. We claim that the automorphism group is S_3, that it acts on the vertices 1, 2, 3 in its natural action (3), it acts on the A_{ij}'s and B_{ji}'s together in its regular action (4), and it acts on A, B in its odd-even action (2).

1, 2, and 3 are the only vertices with in-degree and out-degree 1, so at most they can be swapped among each other. Hence the automorphism group is at most S_3. To show the above claim, it suffices to show that the generating set (123) and (12) of S_3 provides automorphisms of the color gadget that act as described.

Consider first (123). It acts on 1, 2, and 3 as described by the cycle notation: $1 \mapsto 2 \mapsto 3 \mapsto 1$. To be an automorphism, it is then forced to send $A_{12} \mapsto A_{23} \mapsto A_{31} \mapsto A_{12}$ and $B_{21} \mapsto B_{32} \mapsto B_{13} \mapsto B_{21}$. Note that (123) cannot possibly swap the A_{ij}'s and the B_{ji}'s, since the directed edges determine an orientation that is preserved by (123). Moreover, its action on these vertices is exactly the action of (123) by right multiplication on S_3 itself. This implies that (123), and hence all the even permutations, fix the vertices A and B.

Next, consider (12). Since (12) reverses the orientation determined by the directed edges, it must swaps the A_{ij}'s and B_{ji}'s, as follows: $A_{12} \leftrightarrow B_{21}$, $A_{23} \leftrightarrow B_{13}$, and $A_{31} \leftrightarrow B_{32}$. This also implies that $A \leftrightarrow B$. Hence odd permutations swap A and B. Finally, the action of (12) on the A_{ij}'s and B_{ji}'s is in accordance with the right regular action of (12) on S_3,

compatible with that of (123) above. We can put this together through the correspondence:

$$\begin{aligned} () &\sim A_{12} \\ (123) &\sim A_{23} \\ (132) &\sim A_{31} \\ (12) &\sim B_{21} \\ (13) &\sim B_{32} \\ (23) &\sim B_{13} \end{aligned}$$

□

In Section 4.8 we show how the above color palettes can be replaced in a black-box fashion by a construction due independently to Bouwer [62] and Babai [20]. We presented the above construction anyways, as we came to it independently and it helps give the flavor of the color palettes. We also use their construction to extend the above result to show that the general problem of TWISTED CODE EQUIVALENCE, as defined in Codenotti [83], Karp-reduces to GRAPH ISOMORPHISM.

Lemma 4.4.6. *Over any field of characteristic zero,* GRAPH ISOMORPHISM $\leq_m$ MATRIX ISOMORPHISM OF SEMISIMPLE LIE ALGEBRAS.

The key issue in the following proof is keeping the dimensions of the matrix Lie algebras polynomially bounded. For example, a natural approach using the bipartite adjacency matrix fails in this regard: having k nonzero entries in a single row of the adjacency matrix leads to a representation of dimension at least 2^k, and since the general bipartite graph has vertices of degree more than $O(\log n)$, this would not be polynomial. We instead use the incidence matrix of a graph, which has only two nonzero entries per row.

Proof. Let G_1, G_2 be two graphs with no isolated vertices, and let D_i be the 0-1 incidence matrix of G_i, where the rows correspond to edges and the columns correspond to vertices. The G_i are isomorphic if and only if there is a permutation of the rows and the columns that makes the D_i equal. Then (D_1, D_2) is an instance of PROBLEM A where the column partition is trivial, and the column groups G_ℓ are also trivial. We show how to reduce such

an instance of PROBLEM A to OUTER EQUIVALENCE OF LIE ALGEBRA REPRESENTATIONS, and hence to SEMISIMPLE MATISOLIE. Note that the reduction from OUTER EQUIVALENCE to MATISOLIE is deterministic, as it simply takes each representation to its image.

Given an instance of PROBLEM A as above—in particular, it only contains the entries 0 and 1, it contains exactly two non-zero entries per row, every column contains a non-zero entry (=no isolated vertices in the graphs), and the column partition and column groups are all trivial—let $\mathcal{L} = \mathfrak{sl}_2^{\oplus n}$, where n is the number of vertices of the G_i (=columns of the matrices) and $\mathfrak{sl}_2$ is the simple Lie algebra of 2×2 trace zero matrices. Let a 1 in the matrix D_i correspond to the standard representation of $\mathfrak{sl}_2$, which is faithful and has dimension 2. We note that which representation we chose for "1" is immaterial since all nontrivial representations of $\mathfrak{sl}_2$ are faithful, and each row is the tensor product of only constantly many—in fact, 2—representations. Then, by reversing the reduction in Lemma 4.4.4, we get an instance of OUTER EQUIVALENCE OF LIE ALGEBRA REPRESENTATIONS. Here we are implicitly using the fact that an irreducible representation of a direct sum of semisimple Lie algebras, such as $\mathfrak{sl}_2^{\oplus n}$, is the tensor product of its representations on the summands (see Proposition 2.2.9), in this case the n copies of $\mathfrak{sl}_2$.

Since each column contains a non-zero entry—equivalently, the G_i have no isolated vertices—these representations are faithful. Since each row contains exacty two 1's, the corresponding irreducible representation has dimension $3^2 = 9$, hence the representations we get are matrices of dimension $9m \times 9m$, where m is the number of edges of G_i. Since $\mathfrak{sl}_2^{\oplus n}$ is generated by $3n$ elements, the representations can be specified by $3n \times (9m)^2$ numbers—actually fairly small integers, bounded in absolute value by 4—which is polynomial in the size of the original graphs.

Finally, $\mathrm{Out}(\mathfrak{sl}_2)$ acts trivially on the representations of $\mathfrak{sl}_2$ up to equivalence, so the corresponding column groups are trivial, as desired. □

Theorem 4.4.7. *Over an algebraically closed field of characteristic zero,* MATRIX ISOMORPHISM OF SEMISIMPLE LIE ALGEBRAS *of $n \times n$ matrices can be solved by an f-algorithm in polynomial time, when the Lie algebras consist of $O(\log n)$ simple direct summands.*

Proof. If there are only $O(\log n / \log\log n)$ simple summands, then an elementary brute-force approach to PROBLEM A works in $\mathrm{poly}(n)$ time, since the number of outer automorphisms is at most $\left(\frac{c \log n}{\log\log n}\right)! \cdot 6^{O(\log n/\log\log n)} \leq \mathrm{poly}(n)$. However, when there are $O(\log n)$ simple

summands, the number of outer automorphisms can be as large as $n^{O(\log n)}$, so we instead use a more sophisticated approach to TWISTED CODE EQUIVALENCE, due to Babai, Codenotti, and Qiao [28] (see Codenotti [83, Theorem 4.2.1]). PROBLEM A is in fact a special case of TWISTED CODE EQUIVALENCE WITH MULTIPLICITIES; on the instance of PROBLEM A corresponding to semisimple Lie algebras with $O(\log n)$ simple summands, their algorithm runs in poly(n) time.

Translating between their terminology and ours, the size of the code is the number of rows of the M_i, which is the number irreducible representations of the $\mathcal{L}_i$, which is at most n, the size of the original matrices. The total length of the code is the number of simple summands, which is at most $O(\log n)$ by assumption. The code alphabets that they call Γ_i are the integers labelling the irreducible representations of the simple summands $\mathcal{L}_i$. The twisting groups (=column groups) are the outer automorphism groups of each simple summand $\mathcal{L}_i$. As our twisting groups all have bounded size, the size of the code is at most n, and the total length of the code is at most $O(\log n)$, their algorithm runs in poly(n) time. □

Theorem 4.4.8. *Over an algebraically closed field of characteristic zero,* MATRIX ISOMORPHISMS OF SEMISIMPLE LIE ALGEBRAS *of $n \times n$ matrices can be solved by a polynomial-time f-algorithm, when the Lie algebras consist of $O(\log n / \log\log n)$ irreducible representations, and unboundedly many simple direct summands, at most $O(\log(n))$ of which have nontrivial outer automorphism actions on the representations appearing.*

In Section 2.2.4 we list the simple Lie algebras and their outer automorphism groups, and mention which have trivial actions on *all* of their representations. Two of the four infinite families of simple Lie algebras have this property, as well as four of the five exceptional simple Lie algebras. However, even if the outer automorphism group acts nontrivially on representations, it may nonetheless fix the particular representations appearing in the given matrix Lie algebra; we take advantage of this possibility in Corollary 4.6.3.

Proof. In this case, the M_i in the instance of PROBLEM A have $O(\log n / \log\log n)$ rows. Although the size of the automorphism group may be more than polynomial, there are only polynomially many row permutations, so we only have to handle the outer automorphisms exhaustively *in each column*, but not the permutations between the columns. Specifically, try each combination of outer automorphisms of each column; since there are at most $O(\log n)$

columns with nontrivial outer automorphism actions, and the outer automorphism group of a simple Lie algebra has size at most 6, there are only poly(n) possibilities. For each such possibility, try each of the $\left(\frac{\log n}{\log\log n}\right)! \leq \text{poly}(n)$ many permutations of the rows, and for each check whether the set of columns of M_1 is equal to the set of columns of M_2. □

4.5 Completely reducible Lie algebras

In this section, we describe how the algorithms and reductions for the abelian diagonalizable and semisimple cases fit into a single general framework and can be combined to handle the case of a direct sum of an abelian diagonalizable matrix Lie algebra with a semisimple matrix Lie algebra. In characteristic zero, this class of matrix Lie algebras is exactly the class of *completely reducible* matrix Lie algebras. We have already used heavily the assumption that all representations that we work with are completely reducible, that is, that they can be written as a direct sum of irreducible representations (see Section 2.2.4). In characteristic zero, the class we study in this section is the largest class of Lie algebras with this property (Theorem 2.2.8).

Lemma 4.5.1. *Lemma 4.4.3 applies to the class of abelian Lie algebras and the class of Lie algebras that are a direct sum of an abelian Lie algebra and a semisimple Lie algebra.*

Proof. The proof of Lemma 4.4.3 only required two ingredients: that the isomorphism problem for abstract Lie algebras of the class under consideration be efficiently solvable, and that twisting a representation by an inner automorphism leads to an equivalent representation. Both of these ingredients hold for abelian Lie algebras: two abelian Lie algebras are abstractly isomorphic if and only if they have the same dimension, and abelian Lie algebras have no non-trivial inner automorphisms.

Similarly, let $\mathcal{L}$ be a direct sum $\mathcal{A} \oplus \mathcal{S}$ where $\mathcal{A}$ is abelian diagonalizable and $\mathcal{S}$ is semisimple. The isomorphism problem for this class of Lie algebras is solvable efficiently by Theorem 4.3.1. Finally, $\text{Inn}(\mathcal{L}) = \text{Inn}(\mathcal{A}) \times \text{Inn}(\mathcal{S}) \cong \text{Inn}(\mathcal{S})$ since abelian Lie algebras have no non-trivial inner automorphisms. Hence twisting a representation by an inner automorphism leads to an equivalent representation. □

Although it was not phrased this way above, we can now see that the algorithms and equivalences for MATRIX ISOMORPHISM OF DIAGONALIZABLE LIE ALGEBRAS in fact follow

the same lines as those for semisimple Lie algebras. The main difference is that the outer automorphism group of a d-dimensional abelian Lie algebra is the full general linear group GL_d of $d \times d$ invertible matrices—leading to LINEAR CODE EQUIVALENCE—whereas the outer automorphism group of a semisimple Lie algebra is close to S_n—leading to GRAPH ISOMORPHISM.

Furthermore, we can view Babai's reduction (see [27, Theorem 7.1]) from LINEAR CODE EQUIVALENCE to GRAPH ISOMORPHISM as a sort of "list normal form" algorithm for the action of GL_d by automorphisms. Since GL_d acts by change of basis, we would like reduced row echelon form to be a normal form for this action. However, since one may permute the coordinates in LINEAR CODE EQUIVALENCE, computing reduced row echelon form requires first picking the pivots. Babai's algorithm picks these pivots in all $\binom{n}{d}$ possible ways, reduces to row echelon form, and then uses GRAPH ISOMORPHISM to handle the permutation action on the remaining (non-pivot) coordinates of the code.

Combining these techniques together yields:

Corollary 4.5.2. *Over an algebraically closed field of characteristic zero,* MATRIX ISOMORPHISM *of a direct sum of an abelian diagonalizable a-dimensional Lie algebra with a semisimple Lie algebra of s simple direct summands, and where the matrix Lie algebra has r irreducible representation constituents reduces to $\binom{r}{a}$ instances of* PROBLEM A *of size $r \times (s+a)$. In particular,* MATRIX ISOMORPHISM *of such Lie algebras of $n \times n$ matrices can be solved in* $\mathrm{poly}(n)$ *time under any of the following conditions:*

- *$r = O(\log n / \log\log n)$, $a = O(\log n / \log\log n)$, s unbounded, and the number of simple summands with non-trivial outer automorphism action is at most $O(\log n)$;*
- *$r = O(\log n)$, $a = O(\log n / \log\log n)$, and $s = O(\log n)$;*
- *r unbounded, a or $r - a$ is $O(1)$, $s = O(\log n)$.*

Proof. In this case, the abelian summand $\mathcal{A}$ of the Lie algebra is exactly the center, which may be computed by solving a linear system of equations, and the semisimple part $\mathcal{S}$ is a Levi complement, which may be computed in polynomial time as in Proposition 4.7.1. Diagonalize $\mathcal{A}$. Determine the highest weight space W for $\mathcal{S}$, as in Theorem 4.3.2. Consider the restriction of $\mathcal{A}$ to W; as the restriction of $\mathcal{A}$ to any irreducible representation consists

of scalar matrices, the restriction of $\mathcal{A}$ to the highest weight of an irreducible representation completely determines $\mathcal{A}$ on that representation. In particular, the restriction of $\mathcal{A}$ to W is a linear code of dimension a and length r.

For each of the $\binom{r}{a}$ choice of pivots for the restriction of $\mathcal{A}$ to W, put that portion of $\mathcal{A}$ into reduced row echelon form, as discussed prior to the theorem. Finally, we get an instance of PROBLEM A as in Lemma 4.4.4. In addition to one column for each simple summand of $\mathcal{S}$, there is also a column for each dimension of $\mathcal{A}$, and the weight we put into that column is the eigenvalue from the reduced row echelon form of $\mathcal{A}$ on the corresponding irreducible representation of $\mathcal{S}$.

For the first set of parameters in the statement of the Corollary, we use the brute-force approach on the rows and outer automorphisms within each column, as in Theorem 4.4.8, noting that $\binom{r}{a} \leq \text{poly}(n)$ when both r and a are $O(\log n / \log\log n)$. For the remaining two sets of parameters, we use the algorithm for TWISTED CODE EQUIVALENCE [28] as before. This algorithm runs in time $2^{O(s+a)}\,\text{poly}(r, s, a)$, which is $\text{poly}(n)$ for the settings of parameters described, and we use it $\binom{r}{a}$ times, which is again $\text{poly}(n)$ for the parameters described. □

4.6 Application to equivalence of polynomials

In this section, we show how our results on completely reducible matrix Lie algebras may be applied to the linear equivalence problem for matrix multiplication—which is new—and the determinant—for which we mostly derandomize a result of Kayal's.

We say that a function f is *characterized by the continuous part of its stabilizer* if any function g that is fixed by $\text{Stab}(f)_0$ is a scalar multiple of f, where $\text{Stab}(f)_0$ denotes the connected component of the identity of the symmetry group of f.

Lemma 4.6.1. *Over an algebraically closed field of characteristic zero, let f be a polynomial that is characterized by the continuous part of its stabilizer. Then for any function g, g is linearly equivalent to αf for some scalar α if and only if* $\text{Lie}(\text{Stab}(g))$ *and* $\text{Lie}(\text{Stab}(f))$ *are matrix isomorphic.*

Proof. By Proposition 3.4.2, if g is linearly equivalent to αf then there is an invertible matrix A such that $A\,\text{Stab}(g)A^{-1} = \text{Stab}(\alpha f) = \text{Stab}(f)$; in particular, the same holds of

the identity components of the stabilizers: $A\,\mathrm{Stab}(g)_0 A^{-1} = \mathrm{Stab}(f)_0$. Conversely, since f is characterized by the continuous part of its stabilizer, if $A\,\mathrm{Stab}(g)_0 A^{-1} = \mathrm{Stab}(f)_0$, then $\mathrm{Stab}(Ag)_0 = \mathrm{Stab}(f)_0$, so by symmetry-characterization $Ag = \alpha f$ for some scalar α.

We are implicitly working in some larger matrix Lie group G acting on the relevant space of polynomials. Any connected Lie subgroup of G is completely determined by its matrix Lie algebra, so $A\,\mathrm{Stab}(g)_0 A^{-1} = \mathrm{Stab}(f)_0$ if and only if $A\,\mathrm{Lie}(\mathrm{Stab}(g)_0)A^{-1} = \mathrm{Lie}(\mathrm{Stab}(f)_0)$, that is, if and only if $\mathrm{Lie}(\mathrm{Stab}(g)_0)$ and $\mathrm{Lie}(\mathrm{Stab}(f)_0)$ are matrix isomorphic. Finally, since the Lie algebra only depends on the connected component of the identity, $\mathrm{Lie}(\mathrm{Stab}(f)) = \mathrm{Lie}(\mathrm{Stab}(f)_0)$, and similarly for g. □

Corollary 4.6.2. *Given the Lie algebra of the symmetry group of a bilinear function f from $\mathbb{C}^{n^2} \times \mathbb{C}^{n^2}$ to $\mathbb{C}^{n^2}$, one can determine whether f is bilinearly equivalent to matrix multiplication in* $\mathrm{poly}(n)$ *time with an oracle for finding the roots of univariate polynomials.*

Consequently, given f as a black box, there is a $\mathrm{poly}(n)$*-time Las Vegas f-algorithm to determine whether f is bilinearly equivalent to matrix multiplication.*

For bilinear functions $f_1, f_2 \colon V \times U \to W$, f_1 and f_2 are *bilinearly equivalent* if there are invertible matrices $A \in \mathrm{GL}(V), B \in \mathrm{GL}(U), C \in \mathrm{GL}(W)$ such that $Cf_1(Ax, By) = f_2(x, y)$ for all $(x, y) \in V \times U$. In the case of matrix multiplication, we take $V \cong U \cong W \cong \mathbb{C}^{n \times n}$.

Proof. The Lie algebra of the stabilizer of $n \times n$ matrix multiplication is completely reducible, connected, and is abstractly isomorphic to $\mathbb{F}^2 \oplus \mathfrak{sl}_n \oplus \mathfrak{sl}_n \oplus \mathfrak{sl}_n$. Moreover, matrix multiplication is characterized by the continuous part of its stabilizer. This all follows immediately from the viewpoint whereby the matrix multiplication tensor is essentially $I_n \otimes I_n \otimes I_n$, where I_n is the $n \times n$ identity matrix (see, for example, Bürgisser and Ikenmeyer [71]). By Lemma 4.6.1, any f is equivalent to matrix multiplication if and only if the Lie algebra of its stabilizer is matrix isomorphic to that of matrix multiplication. By Corollary 4.5.2 we can test if the Lie algebra of the symmetry group of f is matrix isomorphic to that of matrix multiplication.

The final consequence follows from Kayal's randomized algorithm for computing the Lie algebra of the stabilizer of a polynomial [159]. □

A similar result holds for iterated matrix multiplication, but we must use much more of the structure of the Lie algebra of its stabilizer. Iterated matrix multiplication is the function

that takes m $k \times k$ $A_1, \ldots, A_m$ to the trace of their product $\operatorname{tr}(A_1 A_2 \cdots A_m)$; we denote this function IMM_m^k. Ben-Or and Cleve showed that IMM_n^3 is complete for polynomial formula-size, or equivalently algebraic $\mathbf{NC}^1$ [49].

Corollary 4.6.3. *Given the Lie algebra of the stabilizer of a multilinear function $f\colon \mathbb{C}^{k^2} \times \cdots \times \mathbb{C}^{k^2} \to \mathbb{C}$ on m inputs of $k \times k$ matrices, one can determine whether f is multilinearly equivalent to IMM_m^k in $\mathrm{poly}(m, k)$ time with an oracle for finding the roots of univariate polynomials.*

Consequently, given f as a black box, there is a $\mathrm{poly}(n)$-time Las Vegas f-algorithm to determine whether f is equivalent to IMM_m^k.

Proof. As with matrix multiplication, IMM_m^k is characterized by the continuous part of its stabilizer, which is the completely reducible Lie group $\mathrm{GL}_k \times \cdots \times \mathrm{GL}_k / \mathbb{F}^*$, so Lemma 4.6.1 applies. The Lie algebra of its stabilizer is abstractly isomorphic to $\mathfrak{sl}_k^{\oplus m} \oplus \mathbb{F}^{m-1}$. In particular, our results do not apply directly, both because there are $\Theta(m)$ many simple summands, each of which has a nontrivial action of its outer automorphism group on its representations, and because the abelian part has dimension $\Theta(m)$. However, the techniques used in our results still apply, but require further analysis of the representations appearing in this matrix Lie algebra.

Towards this end, we describe the stabilizer and its Lie algebra more completely. The stabilizer is given by all transformations of the form $(A_1, A_2, \ldots, A_m) \mapsto (X_1^{-1} A_1 X_2, X_2^{-1} A_2 X_3^{-1}, \ldots, X_m^{-1} A_m X_1)$. It is clear that the internal X_i's all cancel when we take the product. The outer X_1^{-1} and X_1 do not cancel, but they preserve IMM_m^k because the trace is invariant under conjugation, which is exactly what X_1 is doing. However, if the X_i's are scalar matrices whose product is the identity, then the above map not only fixed IMM_m^k, but was in fact the trivial map. This explains the single factor of $\mathbb{F}^*$ that we mod out by in the stabilizer.

The matrix Lie algebra of the stabilizer of IMM_m^k then consists of m summands of $\mathfrak{sl}_k$, where the i-th summand acts in its natural action on the columns of A_{i-1}, which we treat as $\mathbb{F}^k$, and in its dual action (by $-X^T$) on the rows of A_i, which we treat as the dual of $\mathbb{F}^k$.

Each $\mathbb{F}$ summand of the Lie algebra of the stabilizer corresponds to when some X_i is scalar. Since X_i acts on both A_{i-1} and A_i, the instance of Problem A we get looks like this, where we use 1 for the standard representation on $\mathbb{F}^k$, and -1 for its dual:

	$\mathfrak{sl}_k\oplus$	$\mathfrak{sl}_k\oplus$	$\mathfrak{sl}_k\oplus$	$\cdots$	$\oplus\mathfrak{sl}_k\oplus$	$\mathbb{F}\oplus$	$\mathbb{F}\oplus$	$\cdots$	$\oplus\mathbb{F}$
A_1	-1	1	0	$\cdots$	0	1	0	$\cdots$	0
A_2	0	-1	1	$\cdots$	0	-1	1	$\cdots$	0
A_3	0	0	-1	$\cdots$	0	0	-1	$\cdots$	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$	$\vdots$		$\vdots$
A_{m-1}	0	0	0	$\cdots$	1	0	0	$\cdots$	1
A_m	1	0	0	$\cdots$	-1	0	0	$\cdots$	-1

Here we have abused terminology slightly and used the input matrices A_i to label the irreducible representations; each A_i is the standard representation of one $\mathfrak{sl}_k$ summand tensored with the dual representation of the next $\mathfrak{sl}_k$ summand.

We show that the structure of the corresponding graph of the semisimple part combined with the structure of the linear code of the abelian part are simple enough that we can solve MATRIX ISOMORPHISM directly.

The graph corresponding to the semisimple part is an even-length cycle, where the vertices are alternately labeled by A_i or $\mathfrak{sl}_k$. The edges alternate between having the label 1 and -1.

Suppose that $\mathcal{L}$ is any other matrix Lie algebra abstractly isomorphic to the one above. Then the semisimple part of $\mathcal{L}$ is matrix isomorphic to the semisimple part of $\mathrm{Lie}(\mathrm{Stab}(\mathrm{IMM}_m^k))$ if and only if the corresponding instance of PROBLEM A yields a cyclic bipartite graph, where each $\mathfrak{sl}_k$ summand has exactly one 1 and one -1 in its column. This is because, given any even-length cycle where alternating vertices are labeled by $\mathfrak{sl}_k$, and each $\mathfrak{sl}_k$ vertex is incident to one edge labeled 1 and one edge labeled -1, there is a choice of outer automorphisms of the $\mathfrak{sl}_k$ summands that will make the labels 1 and -1 alternate all the way around the cycle.

Once we have put the $\mathfrak{sl}_k$ summands of $\mathcal{L}$ and the irreducible representations of $\mathcal{L}$ into this order, we then proceed to work on the abelian part. The abelian part consists of an S_n-invariant linear code, as it is exactly the space of vectors in $\mathbb{F}^m$ the sum of whose coordinates is zero. So the only remaining question is whether the two codes—one from $Z(\mathcal{L})$ and one from $Z(\mathrm{Lie}(\mathrm{Stab}(\mathrm{IMM}_m^k)))$—are *equal*. This is easily checked by basic linear algebra. □

Corollary 4.6.4. *Given the Lie algebra of the symmetry group of a polynomial f on n^2 variables, one can determine whether f is linearly equivalent to* $\det_n$ *in* deterministic $\mathrm{poly}(n)$

time with an oracle for finding roots of univariate polynomials.

Remark 4.6.5. Over algebraically closed fields of characteristic zero, this derandomizes Kayal's result [159] essentially as much as is currently possible. We will now show that computing the Lie algebra of the symmetry group of a polynomial is in fact *equivalent*, under deterministic polynomial-time many-one reductions, to POLYNOMIAL IDENTITY TESTING, and hence cannot be derandomized without proving significant lower bounds [153]. Kayal [159, Lemma 26] shows how to compute the Lie algebra of the symmetry group of a polynomial given as a black-box, using the algorithm from [160] for computing the linear dependencies between a set of polynomials. Kayal [160] noted that computing such linear dependencies reduces to the search version of POLYNOMIAL IDENTITY TESTING. The search and decision versions of POLYNOMIAL IDENTITY TESTING are equivalent for low-degree functions. Conversely, a polynomial is constant if and only if its symmetry group consists of all invertible transformations of the variables. This holds if and only if the Lie algebra of its symmetry group consists of all linear transformations of the variables. Once this has been determined, evaluating the polynomial at any single point will determine whether it is zero or a non-zero constant.

In some sense, we have thus derandomized Kayal's algorithm as far as is possible in the black-box setting without derandomizing POLYNOMIAL IDENTITY TESTING. Kayal uses randomization at several points, not just in the computation of the Lie algebra of the symmetry group, and we derandomize those using our deterministic algorithm for MATRIX ISOMORPHISM OF SEMISIMPLE LIE ALGEBRAS from Theorem 4.4.7.

Kayal's algorithm [159] also uses the ability to diagonalize matrices as a subroutine, which currently uses randomness over finite fields. We doubt that one can do without this basic primitive. We thus have removed as much randomness as currently seems possible from Kayal's algorithm.

We also note that, even in the dense, non-black-box setting this represents an improvement from $2^{O(n^2)}$ to $2^{O(n \log n)}$. The latter is essentially optimal, since a generic function that is equivalent to $\det_n$ will include nearly all monomials of degree n in n^2 variables, of which there are $2^{\Theta(n \log n)}$. By the dense setting we mean the setting in which f is given by a list of coefficients of all monomials of degree n in n^2 variables (without loss of generality, f is homogeneous of degree n). Naive derandomization of Kayal's algorithm takes time $2^{O(n^2)}$,

since step (ii) of his Section 6.2.1 guesses a random element of a space of dimension $\Theta(n^2)$. Similarly, testing affine equivalence to the determinant can be solved using quantifier elimination (see, e. g., Basu, Pollack, and Roy [42, Ch. 14]), again in time $2^{O(n^2)}$, because the witness to equivalence is an $n \times n$ matrix together with an n-dimensional vector. However, computing the Lie algebra of the symmetry group of f only requires solving a linear system of n^2 equations in a number of variables equal to the number of monomials possible, which is roughly $\binom{n^2}{n} \leq n^{2n} = 2^{O(n \log n)}$.

Proof of Corollary 4.6.4. The Lie algebra of the symmetry group of $\det_n$ is $\mathbb{F} \oplus \mathfrak{sl}_n \oplus \mathfrak{sl}_n$, which has only two simple factors plus a center of scalar matrices. By Corollary 4.5.2 we can test if the Lie algebra of the symmetry group of f is matrix isomorphic to that of $\det_n$. If it is, then act on f by the conjugating matrix so that the Lie algebra is now equal to that of $\det_n$. One might expect to then have to check whether $f(X) = f(X^T)$, since this is also part of the symmetry group of $\det_n$, however, this is not necessary, as the determinant is characterized by the continuous part of its stabilizer (see Proposition 3.4.3). Note that we have combined here all three main steps of Kayal's algorithm into a single reduction to MATRIX ISOMORPHISM OF LIE ALGEBRAS: Kayal uses only the Cartan subalgebra of the Lie algebra to reduce to permutational and scaling equivalence, then solves permutational equivalence and scaling equivalence separately. □

4.7 Application to abstract isomorphism of Lie algebras

In this section, we discuss the relationship between MATRIX ISOMORPHISM OF MATRIX LIE ALGEBRAS and abstract LIE ALGEBRA ISOMORPHISM.

Generally speaking, if for any class $\mathcal{C}$ of Lie algebras one can find a canonical faithful representation of any Lie algebra in $\mathcal{C}$ efficiently, then the abstract isomorphism problem reduces to the matrix isomorphism problem.

In particular, for Lie algebras with zero center, including semisimple Lie algebras, the adjoint representation is a canonical faithful representation that is easily computable. Hence abstract LIE ALGEBRA ISOMORPHISM Karp-reduces to MATRIX ISOMORPHISM OF LIE ALGEBRAS for centerless Lie algebras, including semisimple.

The famous theorems of Ado [5] (characteristic zero) and Iwasawa [148] (positive characteristic) state that every Lie algebra has a faithful finite-dimensional representation (see, for example, Jacobson [150, Chapter VI] or De Graaf [97, Chapter 6]). De Graaf [96] shows how to effectivize the Ado's Theorem, so that the faithful representation is constructible. However, the bounds on the dimension of the faithful representation are not yet polynomial, so this does not give a polynomial-time reduction from LIE ALGEBRA ISOMORPHISM to MATRIX ISOMORPHISM OF LIE ALGEBRAS. The best bound we are aware of is due to Burde and Moens, and is exponential in the dimension of the solvable radical [68, Proposition 4.3]. For nilpotent Lie algebras of dimension n and nilpotency class c, De Graaf [96] gives the constructive bound $\binom{n+c}{c}$; in general c may be as large as $n-1$ so this is still not polynomial in general, but for fixed nilpotency class it is polynomial. However, even if the bound were polynomial, it is not clear that De Graaf's construction algorithm is "canonical" in the sense that it may yield non-matrix-isomorphic representations of abstractly isomorphic Lie algebras. See also Burde, Eick, and De Graaf [66] and Burde and Moens [67].

Finally, we note that, because of Levi's Theorem, for a certain class of Lie algebras we have already determined the complexity of abstract LIE ALGEBRA ISOMORPHISM. Recall the definition of semidirect product of Lie algebras from Section 2.2.4. Levi's Theorem says that in characteristic zero, every Lie algebra $\mathcal{L}$ is the semidirect product of its radical $\mathcal{R}$ by a semisimple subalgebra $\mathcal{S} \subseteq \mathcal{L}$. Any such semisimple subalgebra is called a *Levi subalgebra*. Any two Levi subalgebras of $\mathcal{L}$ differ from one another only by an automorphism of $\mathcal{L}$ (see, for example, Jacobson [150, Theorem III.9 on p. 92]). Using algorithms for computing the radical and Levi subalgebra of a Lie algebra, we observe:

Proposition 4.7.1. *Over any field of characteristic zero,* MATRIX ISOMORPHISM OF SEMISIMPLE LIE ALGEBRAS *is Cook-equivalent to (abstract)* LIE ALGEBRA ISOMORPHISM *for Lie algebras with abelian radical.*

The above equivalence uses only two queries. If we restrict attention to abstract Lie algebras with abelian radical with the additional property that the action of a Levi complement on the radical is faithful, the Cook-equivalence becomes a Karp-equivalence.

Remark 4.7.2. If the ground field is algebraically closed of characteristic zero, the second query of the Cook reduction above may be replaced by a polynomial-time f-algorithm, and we get an f-Karp-equivalence.

The most natural and obvious equivalence is between MATRIX ISOMORPHISM OF SEMISIMPLE MATRIX LIE ALGEBRAS and LIE ALGEBRA ISOMORPHISM of Lie algebras with abelian radical *on which a Levi complement acts faithfully*; this is because matrix Lie algebras by nature correspond to their own *faithful* representations. By using a little extra linear algebra and an additional query to MATRIX ISOMORPHISM, we get the equivalence stated in the proposition.

Proof. Suppose $\mathcal{L}_1, \mathcal{L}_2 \subseteq M_n(\mathbb{F})$ are two semisimple matrix Lie algebras. Let $V = \mathbb{F}^n$ denote the vector space on which the $\mathcal{L}_i$ naturally act. Then we may treat V as an (abstract) abelian Lie algebra. Since the Lie bracket on V is always zero, any linear map $V \to V$ is a derivation, that is, $\mathrm{Der}(V) = \mathrm{End}(V)$. Since each matrix in $\mathcal{L}_i$ induces (or, depending on your viewpoint, *is*) a linear map from $V \to V$, we get a natural homomorphism from $\mathcal{L}_i$ to $\mathrm{End}(V) = \mathrm{Der}(V)$; call this homomorphism φ_i. We then construct the abstract Lie algebras $\mathcal{L}_i \ltimes_{\varphi_i} V$. Since the radical is characteristic—that is, invariant under all automorphisms—it is a standard fact about semidirect products that $\mathcal{L}_1 \ltimes_{\varphi_1} V$ and $\mathcal{L}_2 \ltimes_{\varphi_2} V$ are abstractly isomorphic if and only if there are abstract isomorphisms $f\colon \mathcal{L}_1 \to \mathcal{L}_2$ and $g\colon V \to V$ such that this pair of isomorphisms makes the actions φ_1 and φ_2 equal, that is

$$\varphi_1(x_1)(v) = g^{-1}(\varphi_2(f(x_1))(g(v))) \tag{4.1}$$

for all $x_1 \in \mathcal{L}_1$, $v_1 \in V$. If we treat $\varphi_i(x)$ as a matrix, then (4.1) says exactly that $g\colon V \to V$ is a matrix isomorphism between $\mathcal{L}_1$ and $\mathcal{L}_2$; the isomorphism f here does not affect the *sets* of matrices corresponding to $\mathcal{L}_i$, only their choice of basis. (This is the same observation from before: that twisting a representation by an automorphism does not change its image.)

Conversely, suppose $\mathcal{L}_1, \mathcal{L}_2$ are (abstract) Lie algebras whose radicals are abelian. For each $\mathcal{L}_i$ we will construct a pair of semisimple matrix Lie algebras $(\mathcal{M}_i, \mathcal{N}_i)$ such that $\mathcal{L}_1$ and $\mathcal{L}_2$ are abstractly isomorphic if and only if $\mathcal{M}_1$ is matrix isomorphic to $\mathcal{M}_2$ and $\mathcal{N}_1$ is matrix isomorphic to $\mathcal{N}_2$. If the action of a Levi complement of $\mathcal{L}_i$ on its radical is faithful, then $\mathcal{M}_i$ will be zero, yielding the second claim of the proposition. For simplicity, we drop the subscripts $1, 2$ and construct from an abstract Lie algebra $\mathcal{L}$ a pair of matrix Lie algebras $(\mathcal{M}, \mathcal{N})$.

Here we will use several algorithms presented in De Graaf [97]. We present them here to demonstrate that, under the stated structural assumptions on the Lie algebras, they indeed involve nothing more than solving linear systems of equations, and hence can be done in polynomial time.

First we compute the solvable radical use Cartan's criterion [44] (described in De Graaf [97, pp. 52–53]): compute a basis $\{y_1, \ldots, y_k\}$ for the derived subalgebra $[\mathcal{L}, \mathcal{L}]$, and then intersect the kernels of the linear maps $x \mapsto \operatorname{tr}(\operatorname{ad}_{\mathcal{L}} x \operatorname{ad}_{\mathcal{L}} y_j)$ over all $1 \leq j \leq k$. This intersection is the radical $\mathcal{R}$ of $\mathcal{L}$.

Next we compute a Levi subalgebra [220, 98]. When the radical is abelian, as in our case, the algorithm as presented in De Graaf is quite straightforward, and proceeds as follows [97, p. 131]. Compute a basis $\{x_1, \ldots, x_s\}$ for any complement to $\mathcal{R}$ in $\mathcal{L}$ as a vector space. Compute the structure constants $\gamma_{i,j}^{(k)}$ of $\mathcal{L}/\mathcal{R}$ by computing $[x_i, x_j] = \sum_k \gamma_{i,j}^{(k)} x_k + r_{ij}$, where r_{ij} are arbitrary elements of $\mathcal{R}$. Now let r_i be a variable representing an element of $\mathcal{R}$, and write $z_i = x_i + r_i$. Then the z_i are a basis for a Levi complement if and only if $[z_i, z_j] = \sum_k \gamma_{i,j}^{(k)} z_k$, this time with exact equality, rather than with a residue in $\mathcal{R}$. Each such equation becomes a linear equation in the r_i:

$$[x_i + r_i, x_j + r_j] = \sum_k \gamma_{i,j}^{(k)} (x_k + r_k)$$

becomes

$$[x_i, r_j] + [r_i, x_j] - \sum_k \gamma_{i,j}^{(k)} r_k = -[x_i, x_j] + \sum_k \gamma_{i,j}^{(k)} x_k.$$

Let $\mathcal{S}$ denote this Levi complement; in particular, $\mathcal{L} = \mathcal{R} \rtimes \mathcal{S}$.

Finally, let $\varphi \colon \mathcal{S} \to \operatorname{Der}(\mathcal{R}) = \operatorname{End}(\mathcal{R})$ be the representation of $\mathcal{S}$ on $\mathcal{R}$ induced by the Lie bracket in $\mathcal{L}$. Compute the kernel of this representation, and denote it $\mathcal{M} := \ker(\varphi)$.

On a semisimple Lie algebra, the bilinear form defined by $(x, y) := \operatorname{tr}(\operatorname{ad} x \operatorname{ad} y)$ is called the *Killing form*. It is nondegenerate, and if the Lie algebra is a direct sum of simple algebras, then its Killing form is the direct sum of the Killing forms of its simple factors (see, for example, De Graaf [97, Lemmas 4.3.2 and 4.3.3]). In particular, this means that $\mathcal{S} = \mathcal{M} \oplus \mathcal{N}$ where $\mathcal{N}$ is the orthogonal complement of $\mathcal{M}$ relative the Killing form; that is, $\mathcal{N} = \{x : \operatorname{tr}(\operatorname{ad} x, \operatorname{ad} y) = 0 \text{ for all } y \in \mathcal{M}\}$ is a subalgebra. This complement $\mathcal{N}$ is simply

computed as the intersection of the kernels of the maps $x \mapsto \operatorname{tr}(\operatorname{ad} x, \operatorname{ad} z_i)$ for each z_i in a basis of $\mathcal{M}$.

Now we have that $\mathcal{L} = (\mathcal{R} \rtimes \mathcal{N}) \oplus \mathcal{M}$ where $\mathcal{N}$ acts faithfully on $\mathcal{R}$. We treat $\mathcal{N}$ as a matrix Lie algebra via its faithful representation on $\mathcal{R}$. We treat $\mathcal{M}$ as a matrix Lie algebra via its adjoint representation, as in the discussion prior to this proposition.

As in the first part of this proof, $\mathcal{L}_1$ and $\mathcal{L}_2$ are abstractly isomorphic if and only if $\mathcal{S}_1$ and $\mathcal{S}_2$ are matrix isomorphic in their representations on $\mathcal{R}_1$ and $\mathcal{R}_2$, respectively. The representation of $\mathcal{S}_i$ on $\mathcal{R}_i$ is completely determined by its kernel $\mathcal{M}_i$ and by the matrix representation of the complement of the kernel, namely $\mathcal{N}_i$. Hence $\mathcal{L}_1$ and $\mathcal{L}_2$ are abstractly isomorphic if and only if the $\mathcal{M}_i$ are abstractly isomorphic and the $\mathcal{N}_i$ are matrix isomorphic in their representations on the $\mathcal{R}_i$. Since the $\mathcal{M}_i$ are semisimple, and in particular centerless, they are abstractly isomorphic if and only if they are matrix isomorphic in their adjoint representations.

Finally, if the Levi complement $\mathcal{S}$ acts faithfully on $\mathcal{R}$, then $\mathcal{M} = 0$ and we get a Karp reduction instead of a Cook reduction with two queries. □

The remark after the proposition follows from the fact that we only use the second query to MATRIX ISOMORPHISM OF SEMISIMPLE LIE ALGEBRAS to determine abstract isomorphism of the semisimple Lie algebras $\mathcal{M}_i$, which can be performed by a polynomial-time f-algorithm when the field is algebraically closed of characteristic zero [95].

Proposition 4.7.1 provides more evidence for the pattern that MATRIX ISOMORPHISM is, in some sense, a "step up" from abstract LIE ALGEBRA ISOMORPHISM; see Table 4.1.

Lie algebras	ABSTRACT ISOMORPHISM	MATRIX ISOMORPHISM
abelian	trivial	LINEAR CODE EQUIVALENCE
semisimple	polynomial time	GRAPH ISOMORPHISM
abelian radical	GRAPH ISOMORPHISM	Open Question 4.9.1

Table 4.1: The complexity of abstract LIE ALGEBRA ISOMORPHISM and MATRIX ISOMORPHISM OF LIE ALGEBRAS. This table suggests that the latter is "one step up" from the former.

Not only is the complexity of MATRIX ISOMORPHISM OF LIE ALGEBRAS for matrix Lie algebras with abelian radicals open, but we suspect it is significantly more complicated than the semisimple case, since the representations of Lie algebras with abelian radical are not

in general completely reducible. The theory of non-completely-reducible representations is significantly less well-developed than that of completely reducible ones—see Chapter 6 for more—the latter being essentially completely understood for our purposes.

4.8 Twisted code equivalence reduces to graph isomorphism

The proof of Lemma 4.4.5 suggests that a more general reduction between TWISTED CODE EQUIVALENCE, as defined in Codenotti [83], and GRAPH ISOMORPHISM is possible, which we give here, even though we do not see any further connection with Lie algebras. PROBLEM A is a special case of TWISTED CODE EQUIVALENCE WITH MULTIPLICITES. Since it is not used elsewhere, we give here the definition of TWISTED CODE EQUIVALENCE.

A *code* (not necessarily linear!) of length n over an alphabet Γ is a subset of Γ^n. As in the case of LINEAR CODE EQUIVALENCE, the symmetric group acts on codewords and codes by permuting the coordinates: $\pi \cdot (x_1, \ldots, x_n) = (x_{\pi(1)}, \ldots, x_{\pi(n)})$, where each $x_i \in \Gamma$. A permutation π acts on $C \subseteq \Gamma^n$ by acting on each of its codewords individually: $\pi \cdot C = \{\pi \cdot x : x \in C\}$. Two codes $C_1, C_2 \subseteq \Gamma^n$ are *equivalent* if there is a permutation $\pi \in S_n$ such that $\pi \cdot C_1 = C_2$.

For TWISTED CODE EQUIVALENCE, as in PROBLEM A, we will need an extension of these notions to multiple alphabets. Let $\Gamma_1, \ldots, \Gamma_r$ be multiple alphabets. Then a code of length $(k_1, \ldots, k_r)$ over $(\Gamma_1, \ldots, \Gamma_r)$ is a subset of $\prod_{i=1}^{r} \Gamma_i^{k_i}$. A permutation $\pi_i \in S_{k_i}$ naturally acts on the coordinates of $\Gamma_i^{k_i}$, as before. Given a list $\pi = (\pi_1, \ldots, \pi_r)$, where $\pi_i \in S_{k_i}$, π naturally acts on codewords, namely, π_i permutes the coordinates of the $\Gamma_i^{k_i}$ part of the code. Equivalently, we may think of π as defining a permutation in $S_{k_1} \times \cdots \times S_{k_r} \subseteq S_{k_1 + \cdots + k_r}$, and this permutation acts on the codewords as before. Two such multi-alphabet codes C_1, C_2 are *equivalent* if there exists such a list $\pi = (\pi_1, \ldots, \pi_r)$, $\pi_i \in S_{k_i}$, such that $\pi \cdot C_1 = C_2$.

Finally, we come to the twisting. Given groups G_i acting on Γ_i, two multi-alphabet codes are $\mathcal{G}$-equivalent ($\mathcal{G} = (G_1, \ldots, G_r)$) if there is a permutation $\pi \in S_{k_1} \times \cdots \times S_{k_r}$ of the coordinates, and for each coordinate in $j \in [k_i]$ an element $g_j \in G_i$ such that, after applying g_j to the letter in the j-th coordinate, and applying the permutation π, the two codes become equal.

An instance of TWISTED CODE EQUIVALENCE consists of: a list of alphabets $\Gamma_1, \ldots, \Gamma_r$, for each of two codes $C_1, C_2 \subseteq \prod_{i=1}^{r} \Gamma_i^{k_i}$ a list of every element of each code, and for each $i = 1, \ldots, r$ a permutation group $G_i \leq \mathrm{Sym}(\Gamma_i)$.

Proposition 4.8.1. TWISTED CODE EQUIVALENCE $\leq_m$ GRAPH ISOMORPHISM.

Here we assume the twisting groups G_i are given by listing all of their elements as permutations, so the size of the input is at least $\sum_{i=1}^{r} |G_i||\Gamma_i| \log |\Gamma_i|$. We leave it as Open Question 4.8.2 whether a similar reduction exists that does not depend on the order of the groups.

Proof. Bouwer [62] and Babai [20] independently showed that for any permutation group $G \leq S_n$, there is a graph X of size roughly $n^3|G|$ such that G is abstractly isomorphic to $\mathrm{Aut}(X)$ and there is a subset Y of the vertices of X such that $\mathrm{Aut}(X) \cdot Y = Y$ and $\mathrm{Aut}(X)$ permutes the vertices of Y in the same manner that G permutes the set $\{1, \ldots, n\}$. In other words, the restriction of $\mathrm{Aut}(X)$ to the vertices in Y is permutationally isomorphic to G. This is exactly the property we needed of the "color palettes" in Lemma 4.4.5. Moreover, from the description in Babai [20] it is clear that this graph can be constructed in time essentially linear in the size of the graph, which is polynomial in the size of the code and twisting groups.

For each twisting group G_i, we use the above graphs as the color palettes, and the rest of the proof proceeds as in Lemma 4.4.5, only slightly simpler. In Lemma 4.4.5 each color palette consisted of multiple copies of a color gadget, because of the possibility that the twisting group would have multiple orbits of the same type. Here, *all* the orbits of the action of the twisting group G_i on Γ_i are known ahead of time, and are built into the color pallete directly via the construction of Babai and Bouwer. □

Open Question 4.8.2. Is there a Karp-reduction from TWISTED CODE EQUIVALENCE to GRAPH ISOMORPHISM that does not depend on the order of the groups G_i?

The above is at least plausible, as each G_i might be given by a generating set of permutations. Since the degree of each G_i is $|\Gamma_i|$, G_i has a generating set of size at most $O(|\Gamma_i| \log |\Gamma_i|)$, so the running time might only need to depend on the code length $\sum k_i$ and the alphabet sizes $|\Gamma_i|$.

However, the above proof does not work in this stronger setting, even if the sizes of the constructions of Babai [20] and Bouwer [62] could be improved. This follows from the fact that the alternating group A_n can be generated by only two elements, but there is a constant $c > 1$ such that any graph G with $\mathrm{Aut}(G)$ isomorphic to A_n must have at least c^n elements [182].

4.9 Future work

MATRIX ISOMORPHISM OF MATRIX LIE ALGEBRAS arises in Geometric Complexity Theory and AFFINE EQUIVALENCE OF POLYNOMIALS. We solved MATRIX ISOMORPHISM OF LIE ALGEBRAS over algebraically closed fields of characteristic zero in polynomial time for several important classes of Lie algebras—namely abelian diagonalizable, semisimple, and completely reducible (=abelian diagonalizable $\oplus$ semisimple)—under various quantitative constraints, using an oracle for finding roots of single-variable polynomials. We showed that without these quantitative constraints, these cases of MATRIX ISOMORPHISM OF MATRIX LIE ALGEBRAS all become at least as hard as GRAPH ISOMORPHISM. In Section 4.9.1 we discuss the possibility of and difficulties in extending our results to other fields. In Section 4.9.2 we discuss the relationship between LIE ALGEBRA ISOMORPHISM and FINITE GROUP ISOMORPHISM. Finally, in Section 4.9.3 we present additional open questions.

4.9.1 Other fields

Here we discuss the main difficulty in extending our results to other fields. Extending our results to finite fields will be the subject of future work; in this section we outline what we believe are the necessary ingredients.

The main Lie-theoretic ingredients of our results were:

1. Efficiently finding abstract isomorphisms of (semisimple) Lie algebras

2. An efficient version of highest weight theory for representations

In the setting we are working in—semisimple Lie algebras over an algebraically closed field of characteristic zero—the two ingredients above are essentially equivalent to being able to construct and diagonalize Cartan subalgebras of matrix Lie algebras. Over other fields,

we can make additional natural assumptions, discussed in the next few sections, so that (1) and (2) are still equivalent to finding and diagonalizing Cartan subalgebras. The latter task then becomes the main bottleneck, as discussed below.

We note that all of the assumptions we introduce over finite fields are satisfied by the matrix Lie algebras arising in the instances of LINEAR EQUIVALENCE OF POLYNOMIALS to which we apply our results in Section 4.6.

Semisimple versus classical

The definition of semisimplicity—containing no abelian ideals—implies the Cartan–Killing classification in characteristic zero. Even over non-algebraically closed fields of characteristic zero this remains true: if $\mathcal{L}$ is a semisimple Lie algebra over any field $\mathbb{F}$ of characteristic zero, then $\mathcal{L} \otimes_{\mathbb{F}} \overline{\mathbb{F}}$ is $\overline{\mathbb{F}}$-isomorphic to one of the Lie algebras in the Cartan–Killing classification. For a given Cartan–Killing type—A_n, B_n, etc.—the problem of determining all semisimple Lie algebras over $\mathbb{F}$ that are of that type over $\overline{\mathbb{F}}$ is a major open problem called "the problem of ($\mathbb{F}$-)forms." We discuss one aspect of this in more detail in the next section.

However, in positive characteristic semisimplicity does not imply the Cartan–Killing classification. To recover the Cartan–Killing classification in positive characteristic, we must restrict our attention to the so-called "classical" Lie algebras, in the sense of Mills and Seligman [204] (see also Seligman [234]). We note that many authors, for example Fulton and Harris [116] use the term "classical" in a more colloquial and confined sense, to mean those Lie algebras of type A,B,C,D ($\mathfrak{sl}$, $\mathfrak{so}$, and $\mathfrak{sp}$); these algebras are "classical" in the sense of "of or pertaining to the classics," whereas the meaning we use here, due to Mills and Seligman, is a term of art. The Mills–Seligman definition works over any field of characteristic $\neq 2, 3$, and yields especially nice results over *perfect* fields of characteristic $\neq 2, 3$. Recall that a field is *perfect* if either it has characteristic zero, or it has characteristic p and every element is a p-th power. In particular, all finite fields are perfect.

It bears repeating that over any finite field $\mathbb{F}_q$ of characteristic $\neq 2, 3$, the simple classical Lie algebras are exactly classified by the Cartan–Killing "A,B,C,D,E,F,G" classification, and every classical Lie algebra is a direct sum of simple classical ones. The classical Lie algebras are constructed in the same manner as they are over an algebraically closed field of characteristic zero, and they have the same outer automorphism groups. In particular,

classical Lie algebras contain split Cartan subalgebras. See Seligman [234] for details. We expect any immediate extension of our results to finite fields to only work in the classical case.

The main bottleneck: finding split *Cartan subalgebras*

As we mentioned at the beginning of Section 4.9.1, under appropriate assumptions finding and diagonalizing a Cartan subalgebra is the algorithmic keystone needed for our results. Here we discuss how this presents a difficulty over non-algebraically closed fields, even for classical Lie algebras.

Recall that a Cartan subalgebra is *split* if it can be diagonalized over the ground field. If we are working in an abstract Lie algebra, then a Cartan subalgebra is split if and only if it can be diagonalized over the ground field in the adjoint representation.

For readers acquainted with the the algebraically closed case, the fact that finding a *split* Cartan subalgebra is an issue may at first seem surprising, due to the following two facts: 1) in every classical abstract Lie algebra there is a basis, called a Chevalley basis, such that, in the adjoint representation, the Cartan subalgebra has a basis of diagonal matrices over $\mathbb{Z}$ in characteristic zero, or $\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z}$ in characteristic p (regardless of the ground field $\mathbb{F}$!); and 2) any two Cartan subalgebras are conjugate by an inner automorphism over $\overline{\mathbb{F}}$. Now, if a matrix A over $\mathbb{F}$ is conjugate over $\overline{\mathbb{F}}$ to a diagonal matrix over $\mathbb{Z}$ (respectively, $\mathbb{Z}/p\mathbb{Z}$), then A's eigenvalues are integral, and in particular lie in $\mathbb{F}$. But then A's eigenvectors are defined over $\mathbb{F}$ as well, so A is in fact diagonalizable over the ground field $\mathbb{F}$. It thus seems that if we find any Cartan subalgebra of $\mathcal{L}$, it is $\overline{\mathbb{F}}$-conjugate to a Cartan subalgebra defined over $\mathbb{Z}$, so should have weights over $\mathbb{Z}$, and so should in fact be diagonalizable over $\mathbb{F}$.

What went wrong? The issue is that, except in very small cases, the Cartan subalgebra is *not* spanned by a single matrix! A Cartan subalgebra $\mathcal{H}$ over $\mathbb{F}$ may have the property that *no* $\mathbb{F}$*-basis* of $\mathcal{H}$ is $\overline{\mathbb{F}}$-conjugate to a set of integral diagonal matrices. That is, in order to get a basis of $\mathcal{H} \otimes_{\mathbb{F}} \overline{\mathbb{F}}$ that is $\overline{\mathbb{F}}$-conjugate to integral diagonal matrices, one must take linear combinations of some $\mathbb{F}$-basis of $\mathcal{H}$ *using coefficients from the algebraic closure* $\overline{\mathbb{F}}$. This phenomenon is closely related to the so-called "issue of forms" mentioned at the beginning of this section.

In particular, although De Graaf, Ivanyos, and Rónyai [94] gave an algorithm to compute a Cartan subalgebra over any field of characteristic zero using only polynomially many arithmetic operations (see our discussion in Phase I of the proof of Theorem 4.3.1), this does not immediately allow our results to go through for non-algebraically closed fields of characteristic zero. Moreover, Babai has suggested [24] that over $\mathbb{Q}$ finding matrix isomorphisms of simple classical matrix Lie algebras may be at least as hard as factoring integers; this is by analogy with a result of Rónyai, which says that determining isomorphism of 4-dimensional central simple associative algebras over $\mathbb{Q}$ is at least as hard as factoring squarefree integers under Las Vegas polynomial-time many-one reductions, assuming the Generalized Riemann Hypothesis [225].

Over finite fields, however, there is a Las Vegas polynomial-time algorithm for finding a *split* Cartan subalgebra of an abstract Lie algebra, due to Ryba [229]. We note that even as an f-algorithm, Ryba's algorithm is a Las Vegas polynomial-time f-algorithm; that is, it uses more randomness than just that needed for factoring polynomials over finite fields.

Ingredients for finite fields

In this section we outline the other ingredients we believe are needed to extend our results to the finite field case. We have already mentioned that the term "semisimple" must be replaced by "classical" and the term "simple" by "simple classical."

Next, because Ryba's algorithm is a Las Vegas polynomial-time f-algorithm, wherever our results use an f-algorithm or f-reduction, over finite fields they would at best turn into Las Vegas f-algorithms and Las Vegas f-reductions, respectively.

We also need an analog of the highest weight theory for classical Lie algebras over finite fields. This is furnished by theorems of Curtis [91], but only for so-called "restricted" representations of "restricted" Lie algebras. All classical Lie algebras are restricted (see Jacobson [150, Corollary on p. 191]), so that is no further assumption for us, but assuming that the representation is restricted may be a necessary additional assumption.

Although we will not go into further details, we give the definition of restricted here. A Lie algebra $\mathcal{L}$ over a field of positive characteristic p is *restricted* if for every $x \in \mathcal{L}$, there is a $y \in \mathcal{L}$ such that $(\operatorname{ad} x)^p = \operatorname{ad} y$. Given a restricted Lie algebra, we may define a "p-th power operation" by $x \mapsto x^{[p]} = y$, with x and y as before. A *restricted representation* of a

restricted Lie algebra is then a representation $\rho\colon \mathcal{L} \to M_n$ such that $\rho(x^{[p]}) = \rho(x)^p$ for all $x \in \mathcal{L}$, where the latter notation means taking the p-th power of $\rho(x)$ as a matrix.

Curtis [91] showed that the irreducible restricted representations of a classical Lie algebra in positive characteristic are in bijective correspondence with their highest weights, which are now linear maps from a split Cartan subalgebra to the prime field $\mathbb{F}_p$, where p is the characteristic of the ground field $\mathbb{F}$.

However, it seems like we need to add yet another assumption: that the representations we are dealing with are completely reducible. In characteristic zero this follows for free from semisimplicity, but in positive characteristic it does not. Indeed, Jacobson [149, 150] showed that in positive characteristic every Lie algebra has a faithful representation that is completely reducible as well as a faithful representation that is *not* completely reducible. Furthermore, a restricted Lie algebra has the property that all of its restricted representations are completely reducible if and only if the Lie algebra is abelian and the p-th power mapping is injective [140].

Finally, we may need to assume that the characteristic p is greater than both the dimension of the Lie algebra and the size n of the matrices. This allows Lemma 8.5.1 of De Graaf [97]—which says that inner automorphsims extend to representations, and allows us to move from automorphisms to outer automorphisms, as in Lemma 4.4.3 above—to be extended to the finite field case, since then the matrix exponential of a nilpotent $n \times n$ matrix is well-defined and has the usual properties. That a nilpotent element of a Lie algebra corresponds to a nilpotent matrix in any representation of that Lie algebra comes from the Jordan decomposition for restricted Lie algebras in positive characteristic (see, for example, Seligman [234, Theorem V.7.2]).

We believe that the above ingredients are enough to extend our results to Las Vegas polynomial-time f-algorithms and f-reductions over finite fields, but have not checked this carefully and so do not state any theorems. To summarize, we have introduced the following assumptions:

1. Classical d-dimensional Lie algebras of $n \times n$ matrices, possibly direct sum with abelian diagonalizable matrix Lie algebras (as in Corollary 4.5.2)...

2. ...over finite fields of characteristic $p > \max\{d, n\}$, and $p \neq 2, 3$...

3. ...whose corresponding representation is completely reducible...

4. ...and $\mathcal{L}$ is closed under taking the p-th power of any of its matrices, that is, the corresponding representation is restricted.

4.9.2 Connections with FINITE GROUP ISOMORPHISM

The group isomorphism problem is: given two groups by their multiplication tables, determine whether they are isomorphic. For MATRIX ISOMORPHISM OF MATRIX LIE ALGEBRAS we used, among other things, algorithms of Babai [27, Theorem 7.1] and Babai, Codenotti, and Qiao [28] for LINEAR CODE EQUIVALENCE and TWISTED CODE EQUIVALENCE, respectively. These algorithms were developed in the course of investigations into GROUP ISOMORPHISM for finite groups with no abelian normal subgroups; this is the analog of the definition of semisimple Lie algebras. It is striking that the same algorithms appear useful in both cases. In this section we provide some further commentary and speculations on this connection.

We begin with the similarities. To avoid repetition, we use the term "object" to refer to either a group or a Lie algebra, and the term "normal subobject" to refer to "normal subgroup" in the group case and "ideal" in the Lie algebra case. In the setting of Lie algebras, abelian Lie algebras are excluded from being simple by definitional fiat; in order to make the analogy with groups cleaner, in the following we nonetheless refer to 1-dimensional abelian Lie algebras as "abelian simple."

In both Lie algebras and finite groups...

- The definitions of direct sum, abelian, nilpotent, nilpotency class, upper/lower central series, solvable, derived series, simple, direct sum or product, and semidirect product are exactly analogous;

- There is a unique maximal solvable normal subobject called the solvable radical. Moreover, the quotient of an object by its solvable radical contains no solvable normal subobjects. Equivalently, this quotient contains no abelian normal subobjects;

- The direct summand decomposition of a direct sum of non-abelian simple objects is unique "on the nose:" the summands are determined uniquely as sets, not only up to isomorphism;

- The direct summand decomposition of a direct sum of abelian simple objects is not unique. Indeed, in both cases such direct sums are exactly vector spaces, and every vector subspace is a normal subobject;

- There is a feeling that class 2 nilpotent objects are somehow universal for all nilpotent objects. In the case of FINITE GROUP ISOMORPHISM this is folklore; in the case of Lie algebras, we may take the so-called "wildness" of abstract LIE ALGEBRA ISOMORPHISM for nilpotent Lie algebras of class 2 [47, 46] as evidence of the universality of class 2.

- Finally, representations of nilpotent objects in the natural characteristic are not in general completely reducible. In other words, in this setting, there are representations that cannot be decomposed as a direct sum, but that are not irreducible, that is, they do contain a nontrivial proper sub-representation.

Here, by the "natural characteristic" of a Lie algebra over $\mathbb{F}$ we mean the characteristic of $\mathbb{F}$; the "natural characteristic" of a finite p-group is p. Every finite nilpotent group is the direct product of its maximal (that is, Sylow) groups of prime power order; by the "natural characteristic" of such a nilpotent group we mean any prime dividing the order of the group, or equivalently, the natural characteristic of any subgroup of prime power order.

However, in the case of Lie algebras, there are nilpotent Lie algebras of characteristic zero, whereas there are no finite nilpotent groups of "characteristic zero." Lie algebras in characteristic zero have several strong structural theorems that are known *not* to hold for finite groups:

- Levi's Theorem (see Theorem 2.2.4) says that every Lie algebra in characteristic zero is not just an extension of a solvable one by one with no abelian ideals, but that it is in fact a *semidirect product* (also called a split extension) of the solvable radical by the (semisimple) quotient.

- In characteristic zero, a Lie algebra with no abelian ideals is a direct sum of (non-abelian) simple Lie algebras. (In contrast, a finite group with no abelian normal subgroups is the extension of a direct sum of non-abelian simple groups by a solvable group of derived length at most 3 and a permutation group of logarithmic degree. See, for example, Babai, Codenotti, Grochow, and Qiao [27].)

- The classification of simple Lie algebras in characteristic zero is significantly simpler than the classification of finite simple groups. Indeed, via Chevalley bases and tensoring with finite fields, one can get from simple Lie algebras the so-called Chevalley finite simple groups, yet there are many other finite simple groups that greatly complicate their classification.

- In characteristic zero, a Lie algebra is solvable if and only if its derived subalgebra is nilpotent.

- In characteristic zero, solvable Lie algebras are iterated semi-direct products of abelian Lie algebras (see, for example, De Graaf [96]). In particular, every Lie algebra in characteristic zero is an iterated semidirect product of abelian Lie algebras and simple Lie algebras.

Lie algebras of characteristic zero thus provide a potentially fertile testing ground where the issues of positive characteristic and nilpotency may be teased apart.

There is of course the danger that Lie algebras in characteristic zero, despite being very nice in the above ways, may present other difficulties that do not arise in the case of finite groups. Over non-algebraically closed fields, the problem of forms rears its ugly head. For example, even if we understood Lie algebras over, say, $\overline{\mathbb{Q}}$, to our satisfaction, this does not automatically yield the same level of understanding of Lie algebras over $\mathbb{Q}$, as two Lie algebras over $\mathbb{Q}$ might be non-isomorphic despite being isomorphic after tensoring with $\overline{\mathbb{Q}}$. We saw this problem already for semisimple Lie algebras in Section 4.9.1. In addition to the problem of forms, it is possible that by working over an infinite field, or especially a topological field such as $\mathbb{C}$, that there is simply a richer variety of behaviors possible because there is more "room" for them.

There is also a basic difference in terms of computational complexity between LIE ALGEBRA ISOMORPHISM and GROUP ISOMORPHISM, but this difference actually works in our favor. The difference to which we refer is that Lie algebras are given by bases and structure constants, whereas in GROUP ISOMORPHISM the groups are given by listing all of their elements and the entire multiplication table. Of course, over an infinite field the latter is impossible for a Lie algebra, but over finite fields it is not. Hence, polynomial-time algorithms for Lie algebras given by bases are in fact much stronger than polynomial-time algorithms

for groups given by multiplication tables. For a Lie algebra of dimension n over a finite field $\mathbb{F}_q$, even a simply-exponential-time algorithm, that is, $2^{O(n)}$, becomes polynomial-time if the Lie algebra is given by listing all its elements, as a group would be given. For example, Babai's simply-exponential time algorithm for LINEAR CODE EQUIVALENCE becomes polynomial-time if the codes have dimension $\Omega(n)$ and are given by listing their elements rather than by bases.

Thus, it should perhaps not be disappointing that for Lie algebras given by bases, the MATRIX ISOMORPHISM problem is as hard as GRAPH ISOMORPHISM. Indeed, the equivalence between MATRIX ISOMORPHISM OF SEMISIMPLE LIE ALGEBRAS and GRAPH ISOMORPHISM in Theorem 4.4.1, combined with the best-known algorithm for GRAPH ISOMORPHISM [35] (see also [33])—which takes time $2^{O(\sqrt{n}(\log n)^{3/2})}$—immediately yields an $2^{O(n(\log n)^{3/2})}$ algorithm for MATRIX ISOMORPHISM OF SEMISIMPLE LIE ALGEBRAS of $n \times n$ matrices. If our results extend to finite fields, as suggested in Section 4.9.1, this would yield a $N^{(\log\log N)^{3/2}}$ algorithm for MATRIX ISOMORPHISM OF SEMISIMPLE LIE ALGEBRAS over a finite field given by listing their elements, where N is the *size*, rather than dimension, of the Lie algebra. This would be better than the best known bound on the running time of algorithms for FINITE GROUP ISOMORPHISM; in the case of finite groups, the "almost-trivial" algorithm of trying all possible irredundant generating sets take $N^{\log N + O(1)}$ time, an observation which is credited to Tarjan (see Miller [203]). This all suggests that, from the point of view of the analogy with FINITE GROUP ISOMORPHISM, Theorem 4.4.1 should in fact be taken as a statement of how *easy* MATRIX ISOMORPHISM OF SEMISIMPLE LIE ALGEBRAS is, rather than how hard.

This "easiness" agrees with the situation in finite groups, where Babai, Codenotti, and Qiao [28], building off of joint work with the author [27], use their algorithm for TWISTED CODE EQUIVALENCE, amongst other techniques, to give a polynomial-time algorithm for GROUP ISOMORPHISM of groups with no abelian normal subgroups.

In our work on MATRIX ISOMORPHISM OF LIE ALGEBRAS, we were naturally led to the problem of TWISTED CODE EQUIVALENCE (more precisely, PROBLEM A, which is a special case), and to Babai's algorithm for LINEAR CODE EQUIVALENCE (see the unifying viewpoint in Section 4.5, particularly the discussion preceding Corollary 4.5.2). Moreover, the instances of TWISTED CODE EQUIVALENCE appearing in MATRIX ISOMORPHISM are somewhat simpler than those appearing in FINITE GROUP ISOMORPHISM: in the Lie algebra case, the twisting

groups are the outer automorphism groups of simple Lie algebras, which are all trivial, S_2, or S_3, whereas in the finite groups case, the twisting groups are permutation groups of logarithmic degree in the order of the group.

In summary, we believe that

1. the structural similarities, simpler in the case of Lie algebras,
2. the linear-algebraic nature of Lie algebras,
3. the algorithmic similarities already encountered,
4. and the existence of strong structural theorems for Lie algebras in characteristic zero, particularly the ability to tease apart nilpotency from positive characteristic in the case of Lie algebras,

all suggest that LIE ALGEBRA ISOMORPHISM and MATRIX ISOMORPHISM OF MATRIX LIE ALGEBRAS may provide a gentler slope to climb than tackling FINITE GROUP ISOMORPHISM directly, and that techniques discovered for Lie algebras may be of use for finite groups.

Finally, we would be remiss if we did not mention some of the more direct connections that are known between groups and Lie algebras. First, Lie algebras originally arose in the study of Lie groups, where they have proven to be an incredibly useful tool for understanding the structure and representation theory of Lie groups (see, for example, Fulton and Harris [116]). But even for groups that are not Lie groups, Lie algebras play an important role. We have already mentioned the finite Chevalley groups, but see Alperin [10] for a discussion of the use of Lie methods in finite group theory in general. Next, although there is no such thing as a finite group "of characteristic zero," there are infinite nilpotent groups that are essentially of "characteristic zero." Namely, every finitely generated torsion-free (having no elements of finite order—in some sense as far from being finite as possible) nilpotent group is the lattice of integer points in a nilpotent Lie group over $\mathbb{R}$, the so-called "Mal′cev completion" of the group [190]. These groups may also be a good starting point for understanding the algorithmic aspects of nilpotency while avoiding positive characteristic. It is known that the isomorphism problem for finitely presented nilpotent groups is decidable by an algorithm [131], though the current algorithms seem far from efficient. Finally, for finite p-groups, there are several correspondences with various kinds of algebras, including Lie algebras. For

the Mal′cev and Lazard Correspondences, we refer the reader to the excellent books by Khukro [163, 164] and the lecture notes of Mazza [196]. For correspondences with Jordan algebras, we refer to the recent work by James Wilson [276, 277].

4.9.3 Open Questions

In characteristic zero, the completely reducible case is not far from the general case, though significant obstacles remain. Levi's Theorem says that every Lie algebra is the semi-direct product of a solvable Lie algebra by a semisimple one; a solvable Lie algebra is an iterated extension of abelian Lie algebras (see Section 2.2.4 for definitions). The completely reducible case, which we resolved, restricts the solvable part to be abelian, and restricts the semidirect product to be direct. The complexity of MATRIX ISOMORPHISM OF LIE ALGEBRAS in general remains open, but we believe the following is an achievable next target:

Open Question 4.9.1. What is the complexity of MATRIX ISOMORPHISM OF LIE ALGEBRAS that are *semi*direct products of abelian by semisimple? Or even direct products, but in which the abelian part is not diagonalizable?

As discussed in Section 4.9.1, the main obstacle to extending our results to other fields is the ability to find a split Cartan subalgebra; Ryba [229] gives a Las Vegas polynomial-time algorithm for this over finite fields. Can we do away with the additional randomness Ryba uses, and can this be done at all over non-algebraically closed fields of characteristic zero?

Open Question 4.9.2. What is the exact complexity of MATRIX ISOMORPHISM OF LIE ALGEBRAS over $\mathbb{R}$, $\mathbb{Q}$, number fields, or finite fields? In particular, in a classical Lie algebra over these fields, is it possible to find a *split* Cartan subalgebra in deterministic polynomial time with a root-finding oracle?

We essentially derandomized Kayal's algorithm for testing equivalence to the determinant, except for a part of the algorithm that is equivalent to POLYNOMIAL IDENTITY TESTING and the use of a root-finding oracle. In characteristic zero root-finding can be done deterministically, but there is an issue of accuracy, as discussed in Section 4.1.2:

Open Question 4.9.3. Is there a polynomial-time algebraic algorithm that will find a basis of a Cartan subalgebra of a semisimple matrix Lie algebra over $\mathbb{C}$ or $\overline{\mathbb{Q}}$ such that

the eigenvalues of each basis element are separated by at least 2^{-n^c} for some polynomial n^c? If so, this would yield truly polynomial-time algorithms for MATRIX ISOMORPHISM OF SEMISIMPLE MATRIX LIE ALGEBRAS over $\overline{\mathbb{Q}}$ or $\mathbb{C}$ under the same assumptions used in Theorems 4.4.7, 4.4.8 and 4.5.2.

It would also be nice to know whether there is a way around POLYNOMIAL IDENTITY TESTING and root-finding, though we suspect there is not:

Open Question 4.9.4. Show that testing equivalence to the determinant is as hard as POLYNOMIAL IDENTITY TESTING, or give a deterministic polynomial-time algorithm for it in the black-box setting. In the dense setting, can equivalence to the determinant be tested in time poly(t) where t is the number of non-zero monomials of the input function?

Open Question 4.9.5. Show that MATRIX ISOMORPHISM OF LIE ALGEBRAS of the classes discussed in this paper is as hard as root-finding.

Finally, there are two avenues for further progress on AFFINE EQUIVALENCE to symmetry-characterized functions using MATRIX ISOMORPHISM OF LIE ALGEBRAS. First, although GRAPH ISO-MORPHISM-hardness may seem to be the "final" word in the short term, in the application to AFFINE EQUIVALENCE we may be able to avoid GRAPH ISOMORPHISM altogether. MATRIX ISOMORPHISM OF LIE ALGEBRAS is most directly useful for testing affine equivalence to symmetry-characterized functions such as the determinant and matrix multiplication. Not every Lie algebra can arise as the Lie algebra of the symmetries of a symmetry-characterized function. It is possible that the properties of such Lie algebras are strong enough to avoid the full difficulty of GRAPH ISOMORPHISM, as we did in Corollary 4.6.3, despite the fact that the Lie algebra in that case was certainly large enough to support the general reduction from GRAPH ISOMORPHISM. We are not aware of any polynomials characterized by the continuous part of their stabilizers whose stabilizers are large enough to support the reduction from general GRAPH ISOMORPHISM, aside from IMM_m^k, which we have already shown how to handle efficiently. Second, in addition to the Lie algebra of the symmetries of a function, a function may have a finite group of symmetries "sitting on top of" the Lie algebra.

Open Question 4.9.6. What is the complexity of testing conjugacy of finite groups of symmetries, arising from symmetry-characterized functions?

Appendix on terminology

An original version of this work was titled "Lie algebra conjugacy." In this appendix we explain the change in terminology.

We chose the term "MATRIX ISOMORPHISM OF MATRIX LIE ALGEBRAS" for several reasons: it is concrete; it distinguishes the problem being studied from other closely related problems; it is consistent with other similar usages in complexity theory; and finally, it makes it clear that the problem being studied falls squarely into the same camp as other isomorphism problems so well-known to complexity theorists. This latter reason is partially justified *a posteriori* by the results in this chapter.

We start with the analogy of permutation groups. A permutation group is typically defined as a group G of permutations, or as a subgroup $G \leq S_d$ for some d (for the moment we restrict attention to finite groups). Taken literally, this terminology as stated might seem redundant, as *every* finite group is a permutation group, by Cayley's Theorem. However, implicit in the usage of this definition is that, if we were to consider the abstract structure of a permutation group, it is actually a group G *together with* an injective group homomorphism $G \hookrightarrow S_d$ for some d. This also fits well with the computational paradigm, where the manner in which the group is presented can greatly affect the algorithms available for use on it and their complexities. Thus there are many algorithmic papers considering various *abstract* group problems, such as isomorphism, for permutation groups—that is, groups given as input to the algorithms by permutations.

There are two possible notions of isomorphism of permutation groups: isomorphism as abstract groups, or isomorphism in a way that preserves the permutation structure. The latter is given by a bijection $\pi\colon \{1,\ldots,d\} \to \{1,\ldots,d\}$ such that $\pi G_1 \pi^{-1} = G_2$. This is referred to as "permutational isomorphism." See, for example Codenotti [83].

Similarly, we have defined "matrix isomorphisms" of "matrix Lie algebras." Thus, "MATRIX ISOMORPHISM OF MATRIX LIE ALGEBRAS" is the problem considered in this chapter, whereas "ISOMORPHISM OF MATRIX LIE ALGEBRAS" would be the problem of determining whether two matrix Lie algebras, given as matrices, were *abstractly* isomorphic.

We considered alternatives using the word "equivalence" but ultimately rejected them because of its usage in representation theory. The term "equivalent representations" has been completely standard in the literature for nearly 100 years. Additionally, the equivalence of

representations is also defined in terms of conjugacy, so there was some potential confusion with this term. Indeed, Lemma 4.4.3 shows that MATRIX ISOMORPHISM is the same as determining whether two representations of a Lie algebra are equivalent *up to automorphisms*.

Finally, we note that, analogous to the case of permutation groups, the Ado–Iwasawa Theorem states that every abstract finite-dimensional Lie algebra is abstractly isomorphic to a matrix Lie algebra [5, 148] (see Section 4.7 for more details). The Ado–Iwasawa Theorem is, however, significantly more difficult than Cayley's Theorem on finite groups. In particular, Cayley's Theorem simply uses the left regular action of a group G on itself by left multiplication. In contrast, in the case of Lie algebras, any x in the center acts trivially, so the left adjoint representation of a Lie algebra is not in general faithful. The adjoint is the Lie algebra analogue of the representation by conjugation in groups, where the center also acts trivially; the point is that for Lie algebras in general there is no "regular" representation, and the adjoint is the closest we get.

CHAPTER 5
THE COMPLEXITY OF EQUIVALENCE RELATIONS

This chapter is based on the author's joint paper with Fortnow [109]. We have a few small updates to report since the paper first appeared, in the form of new (to us) references: the group membership problem in $\mathrm{PSL}_2(\mathbb{Z})$ can be solved in polynomial time [133], and hence subgroup equality in $\mathrm{PSL}_2(\mathbb{Z})$ might lie in $\mathbf{PEq}\backslash\mathbf{Ker}(\mathbf{FP})$; computably enumerable equivalence relations are actively being studied [121, 13]; and there are individual groups whose word equality problems are **NP**-complete [230, 54, 162], thus furnishing examples of natural **NP**-complete equivalence relations, which was an open question in our original paper.

5.1 Introduction

Equivalence relations and their associated algorithmic problems arise throughout mathematics and computer science. Examples run the gamut from trivial—decide whether two lists contain the same set of elements—to undecidable—decide whether two finitely presented groups are isomorphic [212, 60]. Some examples are of great mathematical importance, and some are of great interest to complexity theorists, such as GRAPH ISOMORPHISM (GRAPHISO).

Complete invariants are a common tool for finding algorithmic solutions to equivalence problems. Normal or canonical forms—where a unique representative is chosen from each equivalence class as the invariant of that class—are also quite common, particularly in algorithms for GRAPHISO and its variants [141, 142, 35, 118, 202, 32]. More recently, Agrawal and Thierauf [8, 260] used a randomized canonical form to show that BOOLEAN FORMULA NON-ISOMORPHISM ($\overline{\mathrm{FI}}$) is in $\mathbf{AM}^{\mathbf{NP}}$. The monograph by Thierauf [260] gives an excellent overview of equivalence and isomorphism problems in complexity theory more generally.

Many efficient algorithms for special cases of GRAPHISO have been upgraded to canonical forms or complete invariants. Are these techniques necessary for an efficient algorithm? Are

these techniques distinct? Gary Miller [202] pointed out that GraphIso has a polynomial-time complete invariant if and only if it has a polynomial-time canonical form (see also [132]). The general form of this question is central both in Blass and Gurevich [56, 57] and here: are canonical forms or complete invariants necessary for the efficient solution of equivalence problems?

In 1984, Blass and Gurevich [56, 57] introduced complexity classes to study these algorithmic approaches to equivalence problems. Although we came to the same definitions and many of the same results independently, this work can be viewed partially as an update and a follow-up to their papers in light of the intervening 25 years of complexity theory. The classes **UP**, **RP**, and **BQP**, the function classes **NPMV** (multi-valued functions computed by **NP** machines) and **NPSV** (single-valued functions computed by **NP** machines), and generic oracle (forcing) methods feature prominently in this work.

Blass and Gurevich [56, 57] introduced the following four problems and the associated complexity classes. Where they use "normal form" we say "canonical form," though the terms are synonymous and the choice is immaterial. We also introduce new notation for these complexity classes that makes the distinction between language classes and function classes more explicit. For an equivalence relation $R \subseteq \Sigma^* \times \Sigma^*$, they defined:

The *recognition problem*: given $x, y \in \Sigma^*$, decide whether $x \sim_R y$.

The *invariant problem*: for $x \in \Sigma^*$, calculate a complete invariant $f(x) \in \Sigma^*$ for R, that is, a function such that $x \sim_R y$ if and only if $f(x) = f(y)$.

The *canonical form problem*: for $x \in \Sigma^*$ calculate a canonical form $f(x) \in \Sigma^*$ for R, that is, a function such that $x \sim_R f(x)$ for all $x \in \Sigma^*$, and $x \sim_R y$ implies $f(x) = f(y)$.

The *first canonical form problem*: for $x \in \Sigma^*$, calculate the first $y \in \Sigma^*$ such that $y \sim_R x$. Here, "first" refers to the standard length-lexicographic ordering on Σ^*, though any ordering that can be computed easily enough would suffice.

The corresponding polynomial-time complexity classes are defined as follows:

Definition 5.1.1. **PEq** consists of those equivalence relations whose recognition problem has a polynomial-time solution. **Ker**(**FP**) consists of those equivalence relations that have a polynomial-time computable complete invariant. **CF**(**FP**) consists of those equivalence relations that have a polynomial-time canonical form. **LexEqFP** consists of those equivalence relations whose first canonical form is computable in polynomial time.

We occasionally omit the "**FP**" from the latter three classes. It is obvious that

$$\mathbf{LexEq} \subseteq \mathbf{CF} \subseteq \mathbf{Ker} \subseteq \mathbf{PEq},$$

and our first guiding question is: which of these inclusions is tight?

5.1.1 Examples

To get a better feel for these complexity classes and help motivate them, we begin with several examples, especially including those that potentially witness the separation of these classes. Some of these will be discussed in more depth in Section 5.3.2. We also rephrase some of the examples we have already mentioned using these classes.

Example 5.1.2. GRAPH ISOMORPHISM is in **NPEq**—equivalence problems decidable in **NP**—and is in **Ker**(**FP**) if and only if it is in **CF**(**FP**) [202] (see also [132]). In fact, this result also holds for any function class that is closed under **FP** reductions such as $\mathbf{FP}^{\mathbf{NP} \cap \mathbf{coNP}}$.

Example 5.1.3. BOOLEAN FORMULA EQUIVALENCE—do two Boolean formulae compute the same function—is in **coNPEq**, and is **coNP**-complete: to check if φ is a tautology, see if it is equivalent to the constant-true formula 1.

Example 5.1.4. Sorting a list is a first canonical form for set equality. Set equality is thus in **LexEqFP**.

Example 5.1.5. The characteristic polynomial is a polynomial-time complete invariant for GRAPH COSPECTRALITY. No polynomial-time canonical form is known for this problem, so GRAPH COSPECTRALITY is a potential witness to $\mathbf{CF} \neq \mathbf{Ker}$.

Example 5.1.6. The *subgroup equality problem* is: given two subsets of a group G determine if they generate the same subgroup. For permutation groups on $\{1, \ldots, n\}$, this problem lies in **CF**(**FP**), via a simple modification [23] of the classic techniques of Sims [242, 243], whose analysis was completed by Furst, Hopcroft, and Luks [119] and Knuth [167]. The subgroup equality problem can also be solved in **P** for (finitely generated) subgroups of $\mathrm{PSL}_2(\mathbb{Z})$ [133]. Subgroup equality problems are a potential source of witnesses to $\mathbf{Ker} \neq \mathbf{PEq}$.

Although factoring integers is not an equivalence problem, its hardness would imply **CF** $\neq$ **Ker**, as the next proposition shows. In Section 5.3.2, we show a similar result based on the hardness of any collision-free hash function that can be computed deterministically. The proof of this proposition highlights what seems to be an essential difference between **CF** and **Ker**.

Proposition 5.1.7. *If* **CF** = **Ker** *then integers can be factored in probabilistic polynomial time.*

Proof. Suppose we wish to factor an integer N. We may assume N is not prime, since primality can be determined in polynomial time [6], but even much weaker machinery lets us do so in probabilistic polynomial time [251, 219], which is sufficient here. By hypothesis, the kernel of the Rabin function $x \mapsto x^2 \pmod N$:

$$R_N = \{(x, y) : x^2 \equiv y^2 \pmod N\}$$

has a canonical form $f \in \mathbf{FP}$.

Randomly choose $x \in \mathbb{Z}/N\mathbb{Z}$ and let $y = f(x)$. Then $x^2 \equiv y^2 \pmod N$; equivalently, $(x - y)(x + y) \equiv 0 \pmod N$. If $y \not\equiv \pm x \pmod N$, then since neither $x - y$ nor $x + y$ is $\equiv 0 \pmod N$, $\gcd(N, x - y)$ is a nontrivial factor z of N. Let $r(N)$ be the least number of distinct square roots modulo N. Then $\Pr_x[y \not\equiv \pm x] \geq 1 - \frac{2}{r(N)}$. Since N is composite and odd without loss of generality, $r(N) \geq 4$. Thus $\Pr_x[y \not\equiv \pm x] = \Pr_x[\text{the algorithm finds a factor of } N] \geq \frac{1}{2}$. Recursively call the algorithm on N/z. □

5.1.2 Main results

Blass and Gurevich showed that none of the four problems above polynomial-time Turing-reduces (Cook-reduces) to the next in line. We extend their results using generic oracles, and we also give further complexity-theoretic evidence for the separation of these classes, giving new connections to probabilistic and quantum computing. Our main results in this regard are:

Proposition 5.1.7. *If* **CF** = **Ker** *then integers can be factored in probabilistic polynomial time.*

Proposition 5.3.12. *If* $\mathbf{CF} = \mathbf{Ker}$ *then collision-free hash functions that can be evaluated in deterministic polynomial time do not exist.*

Theorem 5.3.3. *If* $\mathbf{Ker} = \mathbf{PEq}$ *then* $\mathbf{UP} \subseteq \mathbf{BQP}$. *If* $\mathbf{CF} = \mathbf{PEq}$ *then* $\mathbf{UP} \subseteq \mathbf{RP}$.

Theorem 5.3.6. *If* $\mathbf{PromiseKer} = \mathbf{PromisePEq}$ *then* $\mathbf{NP} \subseteq \mathbf{BQP} \cap \mathbf{SZK}$*, and in particular* $\mathbf{PH} = \mathbf{AM}$.

We give the definitions of **PromisePEq** and **PromiseKer** in Section 5.3.1. We also show the following two related results:

Corollary 5.3.2. *If* $\mathbf{CF} = \mathbf{Ker}$ *then* $\mathbf{NP} = \mathbf{UP}$ *and* $\mathbf{PH} \subseteq \mathbf{S_2}[\mathbf{NP} \cap \mathbf{coNP}] \subseteq \mathbf{ZPP}^{\mathbf{NP}}$.

Corollary 5.3.4. *If* $\mathbf{CF} = \mathbf{PEq}$ *then* $\mathbf{NP} = \mathbf{UP} = \mathbf{RP}$ *and in particular,* $\mathbf{PH} = \mathbf{BPP}$.

Corollary 5.3.2 follows from the slightly stronger Theorem 5.3.1, but we do not give the statement here as it requires further definitions.

5.1.3 Organization

The remainder of this chapter is organized as follows. In Section 5.2 we review the original results of Blass and Gurevich [56, 57]. We also combine their results with other results that have appeared in the past 25 years to yield some immediate extensions. In Section 5.3.1 we prove new results connecting these classes with probabilistic and quantum computation. In Section 5.3.1 we introduce the promise versions of **PEq** and **Ker** and prove Theorem 5.3.6. We also introduce a group-like condition on the witness sets of **NP**-complete problems that would allow us to extend the first half of Theorem 5.3.3 from **UP** to **NP**, giving much stronger evidence that $\mathbf{Ker} \neq \mathbf{PEq}$. We believe the question of whether any **NP**-complete sets have this property is of independent interest: a positive answer would provide nontrivial quantum algorithms for **NP** problems, and a negative answer would provide further concrete evidence for the lack of structure in **NP**-complete problems. In Section 5.3.2 we discuss collision-free hash functions, the subgroup equality problem and Boolean function congruence (not isomorphism) as potential witnesses to the separation of these classes. We also introduce a notion of reduction between equivalence relations and the corresponding notion of completeness. In Section 5.4, we update and extend some of the oracle results of

Blass and Gurevich [56, 57] using generic oracles. In the final section we mention several directions for further research, in addition to the several open questions scattered throughout the paper.

5.2 Previous Results

Here we recall the previous results most relevant to our work. Most of the results in this section are from Blass and Gurevich [56, 57]. We are not aware of any other prior work in this area. However, results in other areas of computational complexity that have been obtained since 1984 can be used as black boxes to extend their results, which we do here.

We mention that analogues of these classes for finite-state machines have been studied, and nearly all their interrelationships completely determined [152]. For the class of computable functions or the class of primitive recursive functions, Blass and Gurevich [56] already noted that all four classes of equivalence relations are equal. However, for the class of computably *enumerable* equivalence relations, a rich theory is developing [121, 13].

If $R \in \mathbf{PEq}$, then the language $R' = \{(x, y) : (\exists z)[z \leq_{lex} y \text{ and } (x, z) \in R]\}$ is in **NP**, and can be used to perform a binary search for the first canonical form for R. Hence, $\mathbf{PEq} \subseteq \mathbf{LexEqFP^{NP}}$. The first result shows that this containment is tight:

Theorem 5.2.1 ([56] Theorem 1)**.** *There is an equivalence relation $R \in \mathbf{CF}$ whose first canonical form problem is essentially $\mathbf{\Delta_2 P}$-complete, that is, it is in $\mathbf{FP^{NP}} = \mathbf{F\Delta_2 P}$ and is $\mathbf{\Delta_2 P}$-hard.*

Note that the above proof that $\mathbf{PEq} \subseteq \mathbf{LexEqFP^{NP}}$ relativizes, so all four polynomial-time classes of equivalence relations are equal in any world where $\mathbf{P} = \mathbf{NP}$, in particular, relative to any **PSPACE**-complete oracle. The next result gives relativized worlds in which $\mathbf{Ker} \neq \mathbf{PEq}$, $\mathbf{CF} \neq \mathbf{Ker}$, and $\mathbf{LexEq} \neq \mathbf{CF}$, though these worlds cannot obviously be combined.

Theorem 5.2.2 (Blass & Gurevich [56] Theorem 2)**.** *Of the four equivalence problems defined above, none is Cook reducible to the next in line. In particular:*

a. There is an equivalence relation $R \notin \mathbf{Ker}(\mathbf{FP}^R)$, i. e., $\mathbf{Ker}(\mathbf{FP}^R) \neq \mathbf{P}^R\mathbf{Eq}$.

b. *There is a function f such that* $\mathrm{Ker}(f) \notin \mathbf{CF}(\mathbf{FP}^f)$*, i. e.,* $\mathbf{CF}(\mathbf{FP}^f) \neq \mathbf{Ker}(\mathbf{FP}^f)$.

c. *There is an idempotent function f such that* $\mathrm{Ker}(f) \notin \mathbf{LexEq}(\mathbf{FP}^f)$*, i. e.,* $\mathbf{LexEq}(\mathbf{FP}^f) \neq \mathbf{CF}(\mathbf{FP}^f)$.

Furthermore, there is an equivalence relation $R \notin \mathbf{Ker}(\mathbf{NPSV_t}^R)$*, i. e.,* $\mathbf{P}^R\mathbf{Eq} \not\subseteq \mathbf{Ker}(\mathbf{NPSV_t}^R)$ *[57, Theorem 5].*

In addition to several extensions of these results, Blass and Gurevich [56, 57] also show that collapses between certain classes of equivalence problems are equivalent to more standard complexity-theoretic hypotheses. Here we collect some of their main results:

Theorem 5.2.3. *1.* $\mathbf{CF}(\mathbf{FP}) \subseteq \mathbf{LexEqNPSV_t} \iff \mathbf{NPEq} \subseteq \mathbf{coNPEq} \iff \mathbf{coNPEq} \subseteq \mathbf{NPEq} \iff \mathbf{NP} = \mathbf{coNP}$ *[57, Thm. 1].*

2. $\mathbf{LexEqNPSV_t} \subseteq \mathbf{PEq} \iff \mathbf{P} = \mathbf{NP} \cap \mathbf{coNP}$ *[57, Thm. 2].*

Note that **NPEq** consists of those equivalence relations decidable in **NP**, and is distinct from $\mathbf{P^{NP}Eq}$ assuming $\mathbf{NP} \neq \mathbf{P^{NP}}$. This follows from the observation that, for any set A there is an equivalence relation R that is polynomial-time equivalent to A, namely the equivalence relation generated by $\{(0x, 1x) : x \in A\}$ (if A is neither empty nor Σ^*, then $A \equiv^p_m R$; in any case, $A \equiv^p_{1-tt} R$).

We think the following result is one of their most surprising:

Theorem 5.2.4 (Blass & Gurevich [57] Theorem 3)**.** *The following statements are equivalent:*

1. $\mathbf{Ker}(\mathbf{FP})_= \subseteq \mathbf{CF}(\mathbf{NPSV_t})$.

2. **NP** *has the shrinking property (see Glaßer, Reitwießner, and Selivanov [123]): if* $A, B \in \mathbf{NP}$*, then there are disjoint* $A', B' \in \mathbf{NP}$ *such that* $A' \subseteq A$*,* $B' \subseteq B$*, and* $A \cup B = A' \cup B'$.

3. $\mathbf{NPMV} \subseteq_c \mathbf{NPSV}$*, i. e., the uniformization principle holds for* **NP**.

Hemaspaandra, Naik, Ogihara, and Selman [138] showed that if $\mathbf{NPMV} \subseteq_c \mathbf{NPSV}$ then $\mathbf{SAT} \in (\mathbf{NP} \cap \mathbf{coNP})/poly$. At the time, the strongest known consequence of $\mathbf{SAT} \in (\mathbf{NP} \cap \mathbf{coNP})/poly$ was $\mathbf{PH} = \mathbf{\Sigma_2 P}$ [157]. Shortly thereafter Köbler and Watanabe [170] improved the collapse to $\mathbf{PH} = \mathbf{ZPP^{NP}}$, and in the early 2000's Cai, Chakaravarthy, Hemaspaandra, and Ogihara [73] further improved the collapse to $\mathbf{PH} = \mathbf{S_2}[\mathbf{NP} \cap \mathbf{coNP}]$. Combined with Theorem 5.2.4, this immediately implies a result that has not been announced previously:

Corollary 5.2.5. *If* $\mathbf{CF} = \mathbf{Ker}$ *then* $\mathbf{PH} \subseteq \mathbf{S_2}[\mathbf{NP} \cap \mathbf{coNP}] \subseteq \mathbf{ZPP^{NP}}$. □

5.3 Evidence for Separation

5.3.1 New Collapses

Blass and Gurevich's [57] proof that $\mathbf{Ker}(\mathbf{FP})_= \subseteq \mathbf{CF}(\mathbf{NPSV_t}) \implies \mathbf{NPMV} \subseteq_c \mathbf{NPSV}$ essentially shows the following slightly stronger result:

Theorem 5.3.1. *If* $\mathbf{CF} = \mathbf{Ker}$ *then* $\mathbf{NPMV_g} \subseteq_c \mathbf{NPSV_g}$.

However, as $\mathbf{NPMV} \subseteq_c \mathbf{NPSV}$ is not known to imply $\mathbf{NPMV_g} \subseteq_c \mathbf{NPSV_g}$, our result does not directly follow from their *result*, but only from its proof, the core of which is reproduced here.

Proof. Let $f \in \mathbf{NPMV_g}$, let M be a nondeterministic polynomial-time transducer computing f, and let V be a polynomial-time decider for graph(f). If $\mathbf{CF} = \mathbf{Ker}$, then the equivalence relation

$$\{((x,y),(x,y')) : V(x,y) = V(x,y')\} = \mathrm{Ker}((x,y) \mapsto (x, V(x,y)))$$

has a canonical form $c \in \mathbf{FP}$. Then the following algorithm computes a refinement of f in $\mathbf{NPSV_g}$: simulate $M(x)$. On each branch, if the output would be y, accept if and only if $c(x,y) = (x,y)$. Hence $f \in_c \mathbf{NPSV_g}$. □

Similar to the original result [57], we can weaken the assumption of this theorem to $\mathbf{Ker}_p \subseteq \mathbf{CF}$, without modifying the proof. By padding, we can further weaken the assumption to $\mathbf{Ker}_= \subseteq \mathbf{CF}$.

Corollary 5.3.2. *If* $\mathbf{CF} = \mathbf{Ker}$ *then* $\mathbf{NP} = \mathbf{UP}$ *and* $\mathbf{PH} \subseteq \mathbf{S_2}[\mathbf{NP} \cap \mathbf{coNP}] \subseteq \mathbf{ZPP}^{\mathbf{NP}}$. □

Note that Corollary 5.2.5 alone does not imply Corollary 5.3.2, as neither of the statements $\mathbf{PH} = \mathbf{S_2}[\mathbf{NP} \cap \mathbf{coNP}]$ and $\mathbf{NP} = \mathbf{UP}$ is known to imply the other. Indeed, it is still an open question as to whether $\mathbf{NP} = \mathbf{UP}$ implies any collapse of $\mathbf{PH}$ whatsoever. See Section 2.1.3, especially the discussion after Theorem 2.1.1.

Our next result gives a new connection between complexity classes of equivalence problems and quantum and probabilistic computation:

Theorem 5.3.3. *If* $\mathbf{Ker} = \mathbf{PEq}$ *then* $\mathbf{UP} \subseteq \mathbf{BQP}$. *If* $\mathbf{CF} = \mathbf{PEq}$ *then* $\mathbf{UP} \subseteq \mathbf{RP}$.

Proof. Suppose $\mathbf{Ker} = \mathbf{PEq}$. Let L be a language in $\mathbf{UP}$, let V be a $\mathbf{UP}$ verifier for L, let p be a polynomial bounding the size of V-witnesses for L. Consider the relation

$$R_L = \{((a,x),(a,y)) : x = y \text{ or } |x| = |y| \text{ and } V(a, x \oplus y) = 1\}$$

where $\oplus$ denotes bit-wise exclusive-or. Clearly $R_L \in \mathbf{PEq}$, so by hypothesis R_L has a complete invariant $f \in \mathbf{FP}$. Since $L \in \mathbf{UP}$, for each $a \in L$ there is a unique string w_a such that $V(a, w_a) = 1$. Define $f_a(x) = f(a,x)$. Then for all distinct x and x', $f_a(x) = f_a(x')$ if and only if $x \oplus x' = w_a$. Given a and f_a, and the promise that f_a is either injective or two-to-one in the manner described, finding w_a or determining that there is no such string is exactly Daniel Simon's problem, which is in $\mathbf{BQP}$ [240].

Now suppose further that $\mathbf{CF} = \mathbf{PEq}$. Then we may take f to be not only a complete invariant but further a canonical form for R_L. On input a, the following algorithm decides L in polynomial time with bounded error: for each length $\ell \leq p(|a|)$, pick a string x of length ℓ at random, compute $f((a,x)) = (a,y)$, and compute $V(a, x \oplus y)$. If $V(a, x \oplus y) = 1$ for any length ℓ, output 1. Otherwise, output 0. If $a \notin L$ then this algorithm always returns 0. If $a \in L$ and 0^ℓ is a's witness, then the algorithm always returns 1. If $a \in L$ and 0^ℓ is not a's witness, then with probability $1/2$, $y \neq x$ and hence the answer is correct. □

We would like to extend the first half of Theorem 5.3.3 from $\mathbf{UP}$ to $\mathbf{NP}$ to give stronger evidence that $\mathbf{Ker} \neq \mathbf{PEq}$, but the techniques do not obviously apply. We pose two approaches to this problem in *Promise classes* and *Groupy witnesses for* $\mathbf{NP}$ *problems*, below.

Corollary 5.3.4. *If* $\mathbf{CF} = \mathbf{PEq}$ *then* $\mathbf{NP} = \mathbf{UP} = \mathbf{RP}$ *and in particular,* $\mathbf{PH} = \mathbf{BPP}$.

Proof. If $\mathbf{CF} = \mathbf{PEq}$ then it follows directly from Theorems 5.3.1 and 5.3.3 that $\mathbf{NP} = \mathbf{UP} \subseteq \mathbf{RP}$. Thus $\mathbf{NP} = \mathbf{RP}$, since $\mathbf{RP} \subseteq \mathbf{NP}$ without any assumptions. Furthermore, it follows that $\mathbf{PH} \subseteq \mathbf{BPP}$ [279], and since $\mathbf{BPP} \subseteq \mathbf{PH}$ [179, 245], the two are equal. □

The collapse inferred here is stronger than that of Corollary 5.2.5, since $\mathbf{BPP} \subseteq \mathbf{S_2P} \subseteq \mathbf{S_2}[\mathbf{NP} \cap \mathbf{coNP}]$ [227, 75]. However, this result is incomparable to Corollary 5.2.5 since it also makes the stronger assumption $\mathbf{CF} = \mathbf{PEq}$, rather than only assuming $\mathbf{CF} = \mathbf{Ker}$.

Promise classes

One way to extend the first half of Theorem 5.3.3 from **UP** to **NP**, suggested to us by Scott Aaronson [3], involves promise versions of **PEq** and **Ker**.

Definition 5.3.5. A language R of triples is in **PromisePEq** if there is a polynomial-time algorithm A such that, whenever $R_a = \{(x, y) : (a, x, y) \in R\}$ is an equivalence relation, $A(a, x, y) = R(a, x, y)$ for all $x, y \in \Sigma^*$.

Similarly, R is in **PromiseKer** if there is a polynomial-time function f such that, whenever R_a is an equivalence relation, $f(a, x) = f(a, y) \iff (a, x, y) \in R$ for all $x, y \in \Sigma^*$. We call such f a *promise complete invariant* for R.

As usual for promise classes, if R_a is not an equivalence relation, we do not restrict the output of $A(a, x, y)$ or $f(a, x)$ in any way.

Theorem 5.3.6. *If* $\mathbf{PromiseKer} = \mathbf{PromisePEq}$ *then* $\mathbf{NP} \subseteq \mathbf{BQP} \cap \mathbf{SZK}$*, and in particular* $\mathbf{PH} = \mathbf{AM}$.

Proof. The first part of the proof follows that of Theorem 5.3.3, treating the promises with care. Suppose $\mathbf{PromiseKer} = \mathbf{PromisePEq}$. Let L be a language in **PromiseUP**, let V be a **PromiseUP** verifier for L, let p be a polynomial bounding the size of V-witnesses for L. That is, if $\#V(x) = \#\{y : V(x, y) = 1\} \leq 1$ then $x \in L \iff (\exists y)[|y| \leq p(|x|)$ and $V(x, y) = 1]$. Consider the relation

$$R_L = \{((a, x), (a, y)) : x = y \text{ or } |x| = |y| \text{ and } V(a, x \oplus y) = 1\}$$

(the same relation as in Theorem 5.3.3). Clearly $R_L \in \mathbf{PromisePEq}$, so by hypothesis R_L has a promise complete invariant $f \in \mathbf{FP}$. Since $L \in \mathbf{PromiseUP}$, for each $a \in L$ such that $\#V(x) \leq 1$, there is at most one string w_a such that $V(a, w_a) = 1$. Define $f_a(x) = f(a, x)$. Then for all distinct x and x', $f_a(x) = f_a(x')$ if and only if $x \oplus x' = w_a$, when such w_a exists, and f_a is injective otherwise. As in Theorem 5.3.3, given a and f_a, finding w_a or determining that there is no such string is exactly Simon's problem [240]. Here, of course, we have reduced to the *promise version* of Simon's problem, which is in **PromiseBQP**.

To show $\mathbf{NP} \subseteq \mathbf{BQP}$, we use the technique of Valiant and Vazirani [270]: given a Boolean formula φ, they randomly produce a formula φ' such that if φ is unsatisfiable, then so is φ', and if φ is satisfiable, then φ' has a *unique* satisfying assignment with probability at least $1/p(|\varphi|)$ for some polynomial p. In this case, $(\varphi', f_{\varphi'})$ satisfies the promise of Simon's problem, and the **BQP** algorithm for Simon's problem either finds the satisfying assignment to φ' or correctly reports that none exists. Since the initial randomized construction of φ' from φ can also be carried out in **BQP**, this whole algorithm puts $\mathsf{SAT} \in \mathbf{BQP}$.

Next we show $\mathbf{NP} \subseteq \mathbf{SZK}$. As above, we randomly transform a Boolean formula φ into a formula φ' which has at most one satisfying assignment, with probability at least $1/p(|\varphi|)$. Then we run the **SZK** protocol for Simon's problem on φ', which we reproduce here for completeness. If $\varphi'(00\cdots0) = 1$, then the verifier accepts immediately. Otherwise, the verifier randomly picks x and sends $f_{\varphi'}(x) = f(\varphi', x)$ to the prover; the prover must try to recover x. If φ' has no satisfying assignments, then $f_{\varphi'}$ is one-to-one, and the prover always succeeds. If φ' has a (unique, not-all-zero) satisfying assignment, then $f_{\varphi'}$ is two-to-one, and the prover fails with probability at least $1/2$. It is clear that this is an **SZK** protocol.

Since the construction of φ' from φ does not require any interaction between the prover and verifier, it can be prepended to the above protocol to give a statistical zero-knowledge protocol for SAT.

Finally, we have $\mathbf{SZK} \subseteq \mathbf{AM} \cap \mathbf{coAM}$ [105, 9], and $\mathbf{NP} \subseteq \mathbf{coAM}$ implies $\mathbf{PH} = \mathbf{AM}$ [22, 61]. □

The two conclusions of the above theorem (that is, "$\mathbf{NP} \subseteq \mathbf{BQP}$" and "$\mathbf{PH} = \mathbf{AM}$") are not known to be related by implication in either direction. Even $\mathbf{NP} \subseteq \mathbf{BQP}$ and $\mathbf{NP} \subseteq \mathbf{SZK}$ are not known to be related by implication. Indeed, there is an oracle relative

to which **SZK** is not contained in **BQP** [1], and there is an oracle relative to which **BQP** is not contained in **SZK** [78].

Groupy witnesses for **NP** *problems*

The technique of the first half of Theorem 5.3.3 does not apply to arbitrary problems in **NP**. However, if an **NP** problem's witnesses satisfy a certain group-like condition, then Theorem 5.3.3 may be extended to that problem.

Let $L \in \mathbf{NP}$ and let V be a polynomial-time verifier for L. By padding if necessary, we may suppose that for each $a \in L$, a's witnesses all have the same length. Suppose there is a polynomial-time length-restricted group structure on Σ^*, that is, a function $f \in \mathbf{FP}$ such that for each length n, Σ^n is given a group structure defined by $xy^{-1} \stackrel{def}{=} f(x, y)$. Then

$$R_L = \{((a, x), (a, y)) : x = y \text{ or } V(a, xy^{-1}) = 1\}$$

is an equivalence relation if and only if a's witnesses are a subgroup of this group structure, or a subgroup less the identity. The technique of Theorem 5.3.3 then reduces L to the hidden subgroup problem over the family of groups defined by f. In this case Σ^n would be a group of order 2^n; we will generalize this below so our groups need not fall into such a specific class.

The *hidden subgroup problem*, or HSP, for a group G is: given generators for G, an oracle computing the operation $(x, y) \mapsto xy^{-1}$, a set X, and a function $f\colon G \to X$ such that $\mathrm{Ker}(f)$ is the partition given by the right cosets of some subgroup $H \leq G$, find a generating set for H [165]. Hidden subgroup problems have played a central role in the study of quantum algorithms. Integer factoring and the discrete logarithm problem both easily reduce to abelian HSPs. The first polynomial-time quantum algorithm for these problems was discovered by Shor [239]; Kitaev [165] then noticed that Shor's algorithm in fact solves all abelian HSPs. The unique shortest vector problem for lattices reduces to the dihedral HSP [222], which is solvable in subexponential quantum time [173]. The graph isomorphism problem reduces to the HSP for the symmetric group [43] or the wreath product $S_n \wr S_2$ [102], but it is still unknown whether any nontrivial quantum algorithm exists for GI.

The proof of Theorem 5.3.3 showed that if **Ker** = **PEq** then every language in **UP** reduces to Daniel Simon's problem. We can now see that Simon's problem is in fact the HSP for $(\mathbb{Z}/2\mathbb{Z})^n$, where the hidden subgroup has order 2. Simon [240] gave a zero-error expected polynomial-time quantum algorithm for this problem, putting it in $\mathbf{ZQP} \subseteq \mathbf{BQP}$. This result was later improved by Brassard and Høyer [63] to a worst-case polynomial time quantum algorithm, that is, in the class **EQP** (sometimes referred to as just **QP**).

This discussion motivates the following definition, results, and open question:

Definition 5.3.7. Let $L \in \mathbf{NP}$. For each a let $W(a)$ denote the set of a's witnesses; without loss of generality, by padding if necessary, assume that $W(a) \subseteq \Sigma^n$ for some n. The language L has *groupy witnesses* if there are functions $\mathsf{mul}, \mathsf{gen}, \mathsf{dec} \in \mathbf{FP}$ such that for each $a \in L$:

1. let $G(a) = \{x \in \Sigma^n : \mathsf{dec}(a, x) = 1\}$; then for all $x, y \in G(a)$, defining $xy^{-1} \stackrel{def}{=} \mathsf{mul}(a, x, y)$ gives a group structure to $G(a)$;
2. $\mathsf{gen}(a) = (g_1, g_2, \ldots, g_k)$ is a generating set for $G(a)$; and
3. $W(a)$ is a subgroup of $G(a)$, or a subgroup less the identity.

The following results are corollaries to the proof, rather than to the result, of Theorem 5.3.3.

Corollary 5.3.8. *If* $\mathbf{Ker} = \mathbf{PEq}$ *and a language* $L \in \mathbf{NP}$ *has groupy witnesses in a family* $\mathcal{G}$ *of groups, then* L *Cook-reduces to the hidden subgroup problem for the family* $\mathcal{G}$*. Briefly:* $L \leq_T^P \mathsf{HSP}(\mathcal{G})$.

Proof. Let $L \in \mathbf{NP}$, let W, G, dec, mul, and gen be as in the definition of groupy witnesses, and let V be a polynomial-time verifier for L such that the witnesses accepted by V on input a are exactly the strings in $W(a)$. Then the equivalence relation

$$\begin{aligned} R_L = \{((a,x),(a,y)) : {} & x = y, \text{ or } \mathsf{dec}(a,x) = \mathsf{dec}(a,y) \\ & \text{and whenever } \mathsf{dec}(a,x) = 1 \text{ we have } V(a, xy^{-1}) = 1]\} \end{aligned}$$

is in **PEq**, since xy^{-1} can be computed by the polynomial-time algorithm mul guaranteed in the definition of groupy witnesses. By hypothesis, R_L has a complete invariant f. The function f, the function mul, and the generating set $\mathrm{gen}(a)$ are a valid instance of the hidden

subgroup problem. If $a \notin L$, then f is injective, and the hidden subgroup is trivial. If $a \in L$, then the hidden subgroup is $W(a)$. Conversely, if the hidden subgroup is trivial, then either $a \notin L$ or the identity of the group is a witness that $a \in L$, which can be easily checked. Hence L reduces to the hidden subgroup problem. □

Corollary 5.3.9. *If* $\mathbf{Ker} = \mathbf{PEq}$ *and the language* L *has abelian groupy witnesses, then* $L \in \mathbf{BQP}$. □

Remark 5.3.10. Every language in **UP** has abelian groupy witnesses. □

Open Question 5.3.11. Are there **NP**-complete problems with abelian groupy witnesses? Assuming $\mathbf{P} \neq \mathbf{NP}$, are there any problems in $\mathbf{NP} \backslash \mathbf{UP}$ with abelian groupy witnesses?

Our definition of having groupy witnesses is similar but not identical to Arvind and Vinodchandran's definition of group-definability [16]. If a set $A \in \mathbf{NP}$ has abelian groupy witnesses, then in general the function $a \mapsto |G(a)|$ is in $\#\mathbf{P}$. If it so happens that this function is in **FP**, then Arvind and Vinodchandran's techniques are sufficient to show that A is low for **PP**. This may or may not be taken as evidence that such an A is unlikely to be **NP**-complete: on the one hand, Beigel [45] gives an oracle relative to which **NP** is *not* low for **PP**, and hence A could not be **NP**-complete. On the other hand, Toda and Ogiwara [263] show that $\mathbf{PP}^{\mathbf{PH}} \subseteq \mathbf{BP} \cdot \mathbf{PP}$ (Tarui [259], independently but using similar methods, strengthens this to $\mathbf{ZP} \cdot \mathbf{PP}$). Hence, under a derandomization assumption, **NP** *is* in fact low for **PP**, and so the lowness of A for **PP** is no obstruction to its being **NP**-complete.

However, even if $|G(a)|$ is computable in polynomial time, it may yet be possible to use Corollary 5.3.8 to show that $\mathbf{Ker} = \mathbf{PEq} \implies \mathbf{NP} \subseteq \mathbf{BQP}$, as there are several classes of non-abelian, and even non-solvable, groups for which the HSP is known to be in **BQP** (see, e. g., [128, 112, 146]).

5.3.2 Hardness

Collision-free hash functions

Collision-free hash functions are a useful cryptographic primitive (see, e. g., [39]). Proposition 5.1.7 suggests a more general connection between the collapse $\mathbf{CF} = \mathbf{Ker}$ and the existence of collision-free hash functions.

A *collection of collision-free hash functions* is a collection of functions $\{h_i : i \in I\}$ for some $I \subseteq \Sigma^*$ where $h_i : \Sigma^{|i|+1} \to \Sigma^{|i|}$ are

1. Easily accessible: there is a probabilistic polynomial-time algorithm G such that $G(1^n) \in \Sigma^n \cap I$;

2. Easy to evaluate: there is a probabilistic polynomial-time algorithm E such that $E(i, w) = h_i(w)$; and

3. Collision-free: for all probabilistic polynomial-time algorithms A and all polynomials p there is a length N such that $n > N$ implies:

$$\Pr_{\substack{i=G(1^n)\\(x,y)=A(i)}} [x \neq y \text{ and } h_i(x) = h_i(y)] < \frac{1}{p(n)}.$$

It is not known whether collections of collision-free hash functions exist, though their existence is known to follow from other cryptographic assumptions (see, e. g., [92]). Many proposed collections of collision-free hash functions, such as MD5 or SHA, can be evaluated deterministically, that is, $E \in \mathbf{FP}$.

Proposition 5.3.12. *If* $\mathbf{CF} = \mathbf{Ker}$ *then collision-free hash functions that can be evaluated in deterministic polynomial time do not exist.*

Proof. The equivalence relation $\{((i, x), (i, y)) : E(i, x) = E(i, y)\}$ has a canonical form $f \in \mathbf{FP}$ by hypothesis. As in the proof of Proposition 5.1.7, the canonical form f can be used by a randomized algorithm to find collisions in h_i with non-negligible probability: choose x at random, and if $f(x) \neq x$ then a collision has been found.

Since h_i maps $\Sigma^{|i|+1} \to \Sigma^{|i|}$, there are at most $2^{|i|} - 1$ singleton classes in $R = \mathrm{Ker}(h_i)$. If x lies in an equivalence class of size at least 2, then $\Pr_x[f(x) \neq x | \#[x]_R \geq 2] \geq \frac{1}{2}$. Thus $\Pr_x[f(x) \neq x] = \Pr_x[f(x) \neq x | \#[x]_R \geq 2] \Pr_x[\#[x]_R \geq 2] \geq \frac{1}{2}\left(\frac{1}{2} + \frac{1}{2^{|i|+1}}\right) > \frac{1}{4}$. □

Subgroup equality

The *subgroup equality problem* is: given two subsets $\{g_1, \ldots, g_t\}$, $\{h_1, \ldots, h_s\}$ of a group G determine if they generate the same subgroup. The *group membership problem* is: given

a group G and group elements $g_1, \ldots, g_t, x$, determine whether or not $x \in \langle g_1, \ldots, g_t \rangle$. A solution to the group membership problem yields a solution to the subgroup equality problem, by determining whether each h_i lies in $\langle g_1, \ldots, g_t \rangle$ and vice versa. However, a solution to the group membership problem does *not* obviously yield a complete invariant for the subgroup equality problem. Thus subgroup equality problems are a potential source of candidates for problems in **PEq\Ker**.

Note that the complexity of these problems still makes sense for non-finite groups, so long as group elements can be specified by finite strings and the group operations are computable.

Fortunately or unfortunately, the subgroup equality problem for permutation groups on $\{1, \ldots, n\}$ has a polynomial-time canonical form, via a simple modification [23] of classical techniques [242, 243, 119, 167] (see Example 5.1.6 for more of the history).

The subgroup equality problem for $\mathrm{PSL}_2(\mathbb{Z})$ is also in **PEq**, via a polynomial-time algorithm for the group membership problem [133].

Boolean function congruence

Two Boolean functions f and g are *congruent* if the inputs to f can be permuted and possibly negated to make f equivalent to g. If f and g are given by formulae φ and ψ, respectively, deciding whether φ and ψ define congruent functions is Karp equivalent to FORMULA ISOMORPHISM. If f and g are given by their truth tables, however, Luks [186] gives a polynomial-time algorithm for deciding whether or not they are congruent. Yet no polynomial-time complete invariant for BOOLEAN FUNCTION CONGRUENCE is known. Hence BOOLEAN FUNCTION CONGRUENCE may be in **PEq\Ker**.

Complete problems?

Equivalence problems that are **P**-complete under **NC** or **L** reductions may lie in **PEq\Ker** due to their inherent difficulty. However, we currently have no reason to believe that **P**-completeness is related to complexity classes of equivalence problems. Towards this end, we introduce a natural notion of reduction for equivalence problems:

Definition 5.3.13. An equivalence relation R *kernel-reduces* to an equivalence relation S, denoted $R \leq^P_{ker} S$, if there is a function $f \in \mathbf{FP}$ such that

$$x \sim_R y \iff f(x) \sim_S f(y).$$

Most natural reductions between well-studied equivalence problems, for example GRAPH ISOMORPHISM and its variants, are of this form. Note that $R \in \mathbf{Ker}$ if and only if R kernel-reduces to the relation of equality. Also note that if $R \leq^P_{ker} S$ via f, then $R \leq^P_m S$ via $(x, y) \mapsto (f(x), f(y))$, leading to the question:

Open Question 5.3.14. Are kernel reduction and Karp reduction different? Are they different on **PEq**? In other words, are there two equivalence relations R and S (in **PEq**?) such that $R \leq^P_m S$ but $R \not\leq^P_{ker} S$?

An equivalence relation $R \in \mathbf{PEq}$ is **PEq**-*complete* if every $S \in \mathbf{PEq}$ kernel-reduces to R. For any **PEq**-complete R, $R \in \mathbf{Ker}$ if and only if $\mathbf{Ker} = \mathbf{PEq}$ if and only if the relation of equality is **PEq**-complete.

Unlike **NP**-completeness, however, the notion of **PEq**-completeness does not become trivial if $\mathbf{Ker} = \mathbf{PEq}$: the relation of equality does not kernel-reduce to the trivial relation simply because equality has infinitely many equivalence classes but the trivial relation has only one. In particular, if $\mathbf{P} = \mathbf{NP}$ then kernel reduction and Karp reduction are distinct on **PEq**, albeit in a rather trivial way. The question becomes more interesting if we ask for languages R and S in **PEq** *of the same densities* on which kernel reduction and Karp reduction differ.

Open Question 5.3.15. Are there **PEq**-complete equivalence problems?

5.4 Oracles

In order to combine the oracles from Blass and Gurevich [56] into a single oracle, as well as construct new oracles that simultaneously separate some classes of equivalence relations and collapse others, we introduce two notions of generic oracle. Generic oracles maintain some of the key advantages of random oracles, but allow us much greater flexibility—much

of the power of finite injury arguments—in their construction[1]. For example, it is often possible to show that some property (complexity class collapse or separation) holds relative to *every* generic oracle, so that it becomes much easier to construct oracles satisfying multiple properties at once. We begin with a review of generic oracle constructions; for a more in-depth discussion, see Fenner, Fortnow, Kurtz, and Li [104].

For those not interested in the technical details of generic oracles, the main result we will need from the next section is Lemma 5.4.4, but we have attempted to keep the technicalities to a minimum. We only use fairly restricted versions of genericity[2] and all the associated concepts in this paper, allowing us to greatly simplify their discussion. Much more general versions and their uses are presented in Fenner, Fortnow, Kurtz, and Li [104].

5.4.1 Preliminaries on Generic Oracles

Throughout this section we will use the first construction of an oracle separating **P** from **NP** [38] as a canonical example.

Many oracle constructions proceed by finite extensions: at each stage of the construction, some requirement is to be satisfied (e.g. "the i-th polynomial-time machine does not accept some fixed relativizable language L^{O}"), and we satisfy it by specifying the oracle on finitely many more strings, leaving those strings we have previously specified untouched. In this paper, a generic oracle is one built by finite extensions which also satisfies Murphy's law: "anything which can happen will happen." More prosaically, a generic oracle is built by interleaving all finite extension arguments that are "interleavable." In the remainder of this section we make these ideas precise.

A *condition* is a partial characteristic function whose domain is finite, that is, a partial function $\sigma\colon \Sigma^* \to \{0,1\}$ with $\mathrm{dom}(\sigma)$ finite. In more general discussions of genericity, such

1. Indeed, there is a notion of genericity $\mathcal{R}$ such that results regarding $\mathcal{R}$-generic oracles are completely equivalent to results regarding random oracles [250] (see also [104], the paragraph just prior to Section 3.2), so generic oracle constructions can be viewed as a generalization of random oracle constructions.

2. For the initiated: rather than treat conditions in general as perfect collections of oracles, we define a condition as a partial characteristic function with finite domain. We also require a strong form of basicness: the union of any two consistent $\mathcal{G}$-conditions (union as partial characteristic functions) must also be a $\mathcal{G}$-condition.

conditions are called *Cohen conditions*. We say that an oracle O *extends* σ if the characteristic function of O agrees with σ on $\mathrm{dom}(\sigma)$. Two conditions σ_1, σ_2 are *consistent* if for every $a \in \mathrm{dom}(\sigma_1) \cap \mathrm{dom}(\sigma_2)$ we have $\sigma_1(a) = \sigma_2(a)$.

Terminologically we treat a partial characteristic function as a partial oracle/set: we write $a \in \sigma$ and say "a is in σ" if $\sigma(a) = 1$, and similarly we write $a \notin \sigma$ and "a is not in σ" if $\sigma(a) = 0$. We are careful not to use either terminology if $a \notin \mathrm{dom}(\sigma)$.

Definition 5.4.1. A *notion of genericity* is a nonempty set $\mathcal{G}$ of conditions such that

0. (branching) for all $\sigma \in \mathcal{G}$, there are at least two distinct conditions $\tau_1, \tau_2 \in \mathcal{G}$ extending σ;

1. (generic) for all $\sigma \in \mathcal{G}$ and all $a \in \Sigma^* \setminus \mathrm{dom}(\sigma)$ there is a condition $\sigma' \in \mathcal{G}$ extending σ such that $a \in \mathrm{dom}(\sigma')$; and

2. (basic) if $\sigma_1, \sigma_2 \in \mathcal{G}$ are consistent, then $\sigma_1 \cup \sigma_2 \in \mathcal{G}$.

Note that the collection of all (Cohen) conditions is a notion of genericity, typically referred to as Cohen genericity. Less trivial is the notion of **UP**-genericity. A **UP** condition is a condition which has at most one string of each length, and only has strings at lengths $tower(k)$, where the *tower* function is defined by $tower(0) = 1$ and $tower(n+1) = 2^{tower(n)}$. The collection of all **UP** conditions yields the notion of **UP**-genericity.

A $\mathcal{G}$-generic oracle is simply one built by further and further specification by $\mathcal{G}$-conditions which satisfies an additional constraint, namely, the formal version of "Murphy's law" which we now present.

Throughout this section we fix a logical system that is strong enough to express all the sentences we care about; for example, Peano Arithmetic with an additional unary predicate X, corresponding to the oracle, will suffice. If φ is a sentence in such a system, then an oracle O satisfies φ if φ is true upon replacing the predicate X by the characteristic function for O. We assume, without loss of generality from the point of view of our constructions, that the logical system has only countably many sentences.

We say that a condition σ *forces* the truth of a sentence φ if φ is true of every oracle O extending σ. For example, φ might be the sentence

$$(\exists n)[M(1^n) = 0 \iff (\exists x)[|x| = n \text{ and } X(x)]]. \tag{5.1}$$

The classic argument of Baker, Gill, and Solovay [38] shows how to construct a Cohen condition forcing φ. That is, we only need to specify a finite amount of the oracle to ensure that φ is true, regardless of how we construct the rest of the oracle.

We say that a notion of genericity $\mathcal{G}$ is strong enough to force a sentence φ if φ can *always* eventually be forced, that is, for every $\mathcal{G}$-condition σ there is another $\mathcal{G}$-condition σ' extending σ such that σ' forces φ. We say, equivalently, that $\{\sigma \in \mathcal{G} : \sigma \text{ forces } \varphi\}$ is *dense* in $\mathcal{G}$. In fact Baker, Gill, and Solovay essentially showed that Cohen genericity is strong enough to force (5.1).

Finally, "Murphy's law," which we require of generic oracles, is that a $\mathcal{G}$-generic oracle must force every sentence φ that $\mathcal{G}$ is strong enough to force.

Definition 5.4.2 (Generic Oracle)**.** Let $\mathcal{G}$ be a notion of genericity. An oracle O is $\mathcal{G}$-generic if there is a consistent collection of $\mathcal{G}$-conditions $\{\sigma_1, \sigma_2, \dots\}$ such that O extends every σ_i, the σ_i fully specify O (that is, $\bigcup_i \operatorname{dom}(\sigma_i) = \Sigma^*$), and every sentence φ that $\mathcal{G}$ is strong enough to force is forced by some σ_i.

We see that this definition essentially captures the idea of simultaneously interleaving all constructions that "can be interleaved," that is, that $\mathcal{G}$ is strong enough to force.

Lemma 5.4.3 (Existence of $\mathcal{G}$-generic oracles)**.** *For every notion of genericity $\mathcal{G}$, $\mathcal{G}$-generic oracles exist. Furthermore, the $\mathcal{G}$-generics are dense in $\mathcal{G}$, that is, for every $\mathcal{G}$-condition σ there is a $\mathcal{G}$-generic oracle extending σ.*

Proof. This is essentially Lemma 3.12 of Fenner, Fortnow, Kurtz, and Li [104], and their proof goes through *mutatis mutandis*, despite our restricted definitions. □

Putting this all together, the way we construct generic oracles in practice is captured by the following lemma:

Lemma 5.4.4. *Let $\mathcal{G}$ be a notion of genericity and φ a sentence. If $\mathcal{G}$ is strong enough to force φ—that is, if every $\sigma \in \mathcal{G}$ can be extended to a $\sigma' \in \mathcal{G}$ forcing φ—then* every *$\mathcal{G}$-generic oracle satisfies φ.*

Finally, this entire discussion relativizes. When we relativize to an oracle A, our formal system includes a new unary predicate which is the characteristic function of A, in addition to the previous unary predicate X corresponding to the generic oracle. We then speak of $\mathcal{G}$-generics relative to A.

5.4.2 Oracles for **PEq**, **Ker**, and **CF**

In this section we introduce and use two new notions of genericity. A *one-sided transitive* condition is a (Cohen) condition τ such that

1. (Length restriction on the 1-side): $1\langle x, y\rangle \in \tau$ implies $|x| = |y|$, and
2. (Transitivity on the 1-side): $1\langle x, y\rangle \in \tau$ and $1\langle y, z\rangle \in \tau$ implies $1\langle x, z\rangle \in \tau$.

We refer to the set of strings starting with the bit b as "the b-side" of an oracle or condition. Note that in a one-sided transitive condition, all we require of the 0-side is that $\mathrm{dom}(\sigma)$ is finite there. It is easily verified that one-sided transitive conditions form a notion of genericity, so by Lemma 5.4.3, one-sided transitive generics exist, and furthermore Lemma 5.4.4 applies to them.

A **UP**-*transitive condition* is a condition τ such that

1. ("**UP**") For each length n, there is at most one string of length n in σ;
2. (gappy) σ is only nonempty at lengths $tower(k)$ for some k. The *tower* function is defined by $tower(0) = 1$ and $tower(n) = 2^{tower(n-1)}$;
3. (length-restricted) $\langle x, y\rangle \in \sigma$ implies $|x| = |y|$.

Note that transitivity—$\langle x, y\rangle \in \tau$ and $\langle y, z\rangle \in \tau$ implies $\langle x, z\rangle \in \tau$—follows from the **UP** restriction (1) and the length restriction (3). Again it is easily verified that **UP**-transitive conditions form a notion of genericity, so **UP**-transitive generics exist, and Lemma 5.4.4 applies to them.

Theorem 5.4.5. *There are oracles A and B relative to which* $\mathbf{P} \neq \mathbf{NP}$ *and*

$$\mathbf{CF}(\mathbf{FP}^A) \neq \mathbf{Ker}(\mathbf{FP}^A) \neq \mathbf{P}^A\mathbf{Eq}, \tag{5.1}$$

$$\mathbf{CF}(\mathbf{FP}^B)_p = \mathbf{Ker}(\mathbf{FP}^B)_p \text{ and } \mathbf{Ker}(\mathbf{FP}^B) \neq \mathbf{P}^B\mathbf{Eq}. \tag{5.2}$$

In fact, (5.1) holds relative to any one-sided transitive generic oracle and (5.2) holds relative to $O \oplus G$ whenever O is **PSPACE**-*complete and G is* **UP**-*transitive generic relative to O.*

We break most of the proof into three lemmas. The proofs of Lemmas 5.4.7 and 5.4.8 are adaptations of the proofs of Blass and Gurevich [56] to generic oracles. The proof of Lemma 5.4.9 is new.

We start by restating a useful combinatorial lemma:

Lemma 5.4.6 (Blass & Gurevich [56] Lemma 1)**.** *Let G be a directed graph on $2k$ vertices such that the out-degree of each vertex is strictly less than k. Then there are two nonadjacent vertices in G.*

Lemma 5.4.6 can be proved by a simple counting argument.

For **UP**-transitive conditions σ (or oracles O) we denote by $\sim_\sigma$ the corresponding equivalence relation, that is, the reflexive, symmetric closure of $\{(x,y) : \langle x,y\rangle \in \sigma\}$. If σ is only a partial function, we take care to only ever write $x \sim_\sigma y$ if $\langle x,y\rangle \in \mathrm{dom}(\sigma)$. For one-sided transitive conditions τ, we use the same notation $\sim_\tau$ to denote the equivalence relation corresponding to the 1-side, that is, the reflexive, symmetric closure of $\{(x,y) : 1\langle x,y\rangle \in \tau\}$.

Lemma 5.4.7. *Relative to any one-sided transitive generic oracle or any* **UP***-transitive generic oracle,* $\mathbf{Ker} \neq \mathbf{PEq}$.

Proof. The proofs for the two types of genericity are essentially identical. Let $\mathcal{G}$ be "one-sided transitive" or "**UP**-transitive" throughout. We give the proof for one-sided transitive genericity, in which all the diagonalization happens on the 1-side; for **UP**-transitive genericity, drop the prefixed 1's throughout and only add strings at lengths $n = tower(k)$ for some k.

For each polynomial-time oracle Turing machine M, let φ_M denote the sentence (often called a requirement):

$$\varphi_M \stackrel{def}{=} (\exists n)[\mathrm{Ker}(M^X) \neq\!\sim_X \text{ on strings of length } n]$$

By Lemma 5.4.4, it suffices to show that any $\mathcal{G}$-condition τ can be extended to a $\mathcal{G}$-condition τ' such that τ' forces φ_M. For then φ_M will hold for every $\mathcal{G}$-generic oracle and for every M, separating **Ker** from **PEq**.

Let M be a polynomial-time oracle transducer running in time $p(|x|)$. Let τ be any $\mathcal{G}$-condition. Let $\overline{\tau}$ denote the minimal (under inclusion) extension of τ to a complete characteristic function (i. e., oracle). We show how to extend τ to another $\mathcal{G}$-condition τ' that forces φ_M, i. e., such that $\mathrm{Ker}(M^O) \neq \sim_O$ for any O extending τ'.

Let n be a length such that $p(n) < 2^{n-1}$ and τ is not defined on $1\langle a, b\rangle$ for any strings a and b of length $\geq n$. Let τ' be the extension of τ to length $p(n)$ that is equal to $\overline{\tau}$ to length $p(n)$. If there are distinct strings x and y of length n such that $M^{\overline{\tau}}(x) = M^{\overline{\tau}}(y)$, then $x \not\sim_{\tau'} y$ but $M^{\tau'}(x) = M^{\tau'}(y)$, and this clearly holds for any O extending τ'.

Otherwise, $M^{\overline{\tau}}(x) \neq M^{\overline{\tau}}(y)$ for every two distinct strings x and y. Say that x *affects* y if M queries $\overline{\tau}$ about $1\langle x, y\rangle$ or $1\langle y, x\rangle$ in the computation of $M^{\overline{\tau}}(y)$. Let G be a digraph on the strings of length n, in which there is a directed edge from y to x if x affects y. The out-degree of each vertex is at most $p(n)$, which is strictly less than 2^{n-1} by the choice of n. Since there are 2^n vertices, Lemma 5.4.6 implies that there are two strings x and y of length n such that neither affects the other. Put $1\langle x, y\rangle$ into τ'. Then $M^{\tau'}(x) \neq M^{\tau'}(y)$ but $x \sim_{\tau'} y$, and this holds for any oracle O extending τ'.

Thus $\mathbf{Ker}^O \neq \mathbf{PEq}^O$ relative to any $\mathcal{G}$-generic oracle O, for $\mathcal{G}$ either "one-sided transitive" or "**UP**-transitive." □

Lemma 5.4.8. *Relative to any one-sided transitive generic oracle,* $\mathbf{CF} \neq \mathbf{Ker}$.

Proof. For this proof, all the diagonalization is performed on the 0-side.

We describe our oracles O and conditions τ with values in the alphabet $\{0, 1, 2\}$ for simplicity (that is, $\tau\colon \Sigma^* \to \{0, 1, 2\}$). Let $read^O\colon \Sigma^* \to \Sigma^*$ denote the oracle function

$$read^O(x) = O(0x01)O(0x011)\cdots O(0x01^{k-1})$$

where k is the least value such that $O(0x01^k) = 2$. Note that the bits used by $read^O$ on input x are disjoint from those used by $read^O$ on any input $y \neq x$. Also note that $read^O$ only queries the oracle regarding strings on the 0-side. Let $R^O = \mathrm{Ker}(read^O)$.

Let f be any polynomial-time oracle transducer, and define

$$\psi_f \stackrel{def}{=} (\exists n)[f^X \text{ is not a canonical form for } R^X \text{ on strings of length } n].$$

As in Lemma 5.4.7, it suffices to show that any one-sided transitive condition τ can be extended to a one-sided transitive condition τ' forcing ψ_f, by Lemma 5.4.4.

Let f be a polynomial-time oracle transducer running in time $p(|x|)$. Let τ be a one-sided transitive condition, and let $\overline{\tau}$ denote the oracle extending τ which has value 2 on strings of the form $0x$ that are not in $\mathrm{dom}(\tau)$ and value 0 on all other strings not in $\mathrm{dom}(\tau)$. We show how to extend τ to a one-sided transitive condition τ' such that f^O does not compute a canonical form for R^O for any O extending τ'.

Let n be a length such that $p(n) < 2^{n-1}$ and such that τ is not defined for any strings $0x$ with $|x| \geq n$. For a string x of length n, let τ_x denote the minimal extension of τ such that $read^{\overline{\tau_x}}$ is the identity on all strings of length n, except $read^{\overline{\tau_x}}(x) = 1^{n+1}$. Since the *read* function only queries strings on the 0-side, τ_x differs from τ only on the 0-side, and we do not need to worry about violating transitivity on the 1-side. Note that $read^{\overline{\tau_x}}$ is injective on strings of length n, so its kernel at length n is the relation of equality. In particular, any canonical form for $R^{\overline{\tau_x}}$ must be the identity on strings of length n.

If there is an x of length n such that $f^{\overline{\tau_x}}(x) \neq x$, then $f^{\overline{\tau_x}}(x)$ is not the identity on strings of length n, so $f^{\overline{\tau_x}}$ is not a canonical form for $R^{\overline{\tau_x}}$. Let the extension τ' be $\overline{\tau_x}$ up to length $p(n)$.

Otherwise, $f^{\overline{\tau_x}}(x) = x$ for all x of length n. We say that $f^O(x)$ queries the oracle about y if $f^O(x)$ queries any of the strings that $read^O(y)$ queries. Find x and y of length n such that $f^{\overline{\tau_x}}(x)$ does not query the oracle about y and $f^{\overline{\tau_y}}(y)$ does not query the oracle about x. This is possible by Lemma 5.4.6, as in the proof of Lemma 5.4.7. Let τ' be the minimal oracle extending τ such that $read^{\tau'}$ is the identity on strings of length n, except $read^{\tau'}(x) = read^{\tau'}(y) = 1^{n+1}$. Then τ' differs from $\overline{\tau_x}$ only on those strings in its domain queried by $read^{\tau'}(y)$ and τ' differs from $\overline{\tau_y}$ only on those strings in its domain queried by $read^{\tau'}(x)$. Since $f^{\overline{\tau_x}}(x)$ does not query the oracle about y we have $f^{\overline{\tau_x}}(x) = f^{\tau'}(x) = x$ and similarly $f^{\overline{\tau_y}}(y) = f^{\tau'}(y) = y$. So relative to any oracle O extending τ', we have $(x, y) \notin \mathrm{Ker}(f^O)$ but $read^O(x) = read^O(y) = 1^{n+1}$. Again, τ' forces that $f^{\tau'}$ is not a canonical form for $R^{\tau'}$.

Thus $\mathbf{CF}^O \neq \mathbf{Ker}^O$ relative to any one-sided transitive generic oracle O. □

Lemma 5.4.9. *If* $\mathbf{P} = \mathbf{PSPACE}$*, and* O *has at most one string of each length tower*(k) *and no other strings, then* $\mathbf{CF}(\mathbf{FP}^{\mathbf{O}})_p = \mathbf{Ker}(\mathbf{FP}^{\mathbf{O}})_p$*. Furthermore, this result relativizes.*

Proof. Let O have at most one string of each length $tower(k)$, and no other strings. Let f be an oracle transducer running in polynomial time $p(|x|)$, let $R = \mathrm{Ker}(f^O)$, and suppose that $\langle x, y\rangle \in R$ implies $|x| \leq q(|y|)$ for some polynomial q. For any input x of sufficient length, all elements of O except possibly one have length either $\leq \log p(|x|)$, in which case they can be found rapidly, or $> p(q(|x|))$ in which case they cannot be queried by f on any input $y \sim_R x$. Following a technique used in [65], we call this one element the "cookie" for this equivalence class.

For the remainder of this proof, "minimum," "least," etc. will be taken with respect to the standard length-lexicographic ordering.

We show how to efficiently compute a canonical form for R. Let R_y denote the inverse image of y under f^O, which is an R-equivalence class. Let

$$B_y \;=\; \{x : f^O(x) = y \text{ and } f^O(x) \text{ does not query the cookie}\},$$

$r_y = \min R_y$, and $b_y = \min B_y$. A canonical form for R is

$$g(x) = \begin{cases} b_y & \text{if } B_y \neq \emptyset \\ r_y & \text{otherwise,} \end{cases}$$

where $y = f^O(x)$. Now we show that g is in fact in $\mathbf{FP^O}$. On input x, the computation of g proceeds as follows:

1. Find all elements of O of length at most $\log p(|x|)$. Any further queries to O of length $\leq \log p(|x|)$ will be simulated without queries by using this data.

2. Compute $y = f^O(x)$.

3. If the cookie was queried, then *all* further queries to O will be simulated without queries using this data. Using the power of **PSPACE**, determine whether or not $B_y = \emptyset$. If $B_y = \emptyset$, find and output r_y. If $B_y \neq \emptyset$, find and output b_y.

4. If the cookie was not queried, then $x \in B_y$, so $B_y \neq \emptyset$. Use the power of **PSPACE** to find the least z such that $f(z) = y$, answering 0 to any queries made by f to strings of length ℓ between $\log p(|x|) < \ell \leq p(q(|x|))$.

5. Run $f^O(z)$. If $f^O(z)$ did not query the cookie, then $f^O(z) = f(z) = y$ and $z = b_y$, so output z. Otherwise, $f^O(z)$ queried the cookie, so no further oracle queries need be made. Using the power of **PSPACE**, find and output b_y.

□

Proof of Theorem 5.4.5. (**CF** $\neq$ **Ker** $\neq$ **PEq**) By Lemmas 5.4.7 and 5.4.8, **CF** $\neq$ **Ker** $\neq$ **PEq** relative to any one-sided transitive generic oracle.

(**CF$_\mathbf{p}$** = **Ker$_\mathbf{p}$** and **Ker** $\neq$ **PEq**) Relativize to any **PSPACE**-complete set C, let O be any **UP**-transitive generic oracle relative to C, and *re*relativize to O. Note that Lemma 5.4.7 relativizes, so relative to C and O combined, **Ker** $\neq$ **PEq**. Since **P** = **PSPACE** relative to C, and O has at most one string of each length $tower(k)$ and no other strings, and Lemma 5.4.9 relativizes, we also have **CF**$_p$ = **Ker**$_p$ relative to C and O combined. □

Open Question 5.4.10. Does **CF** = **Ker** imply **P** = **NP**? Or is there an oracle relative to which **CF** = **Ker** but nonetheless **P** $\neq$ **NP**? Further, is there an oracle relative to which **P** $\neq$ **NP** but **CF** = **Ker** = **PEq**?

Open Question 5.4.11. Is there an oracle relative to which **CF** $\neq$ **Ker** = **PEq**?

5.5 Future Work

Here we present several directions for future work, in addition to the open problems mentioned throughout the paper.

5.5.1 Logarithmic Space

It would also be interesting to study equivalence relations decidable in logarithmic space.

For example, it has been shown that the word equality problem (given two words in the generators of a group, do they represent the same group element?) for a finitely generated linear group is decidable in logarithmic space [184, 241]. (A group is linear if it is isomorphic to a group of matrices over some field.) In fact, implicit in the proofs is a log-space complete invariant: essentially the matrix corresponding to a word in the generators. But it seems unlikely that, in general, one can get from the matrix a corresponding canonical form, that is,

a canonical word in the group generators representing each group element. Hence the word equality problem in finitely generated linear groups is a potential witness to $\mathbf{Ker}(\mathbf{FL}) \neq \mathbf{CF}(\mathbf{FL})$. One open problem is to explicitly construct a linear group with no log-space canonical form for its word equality problem.

Analogues of many of the results in this paper for logarithmic space are intriguing open questions:

- Is **LEq** contained in $\mathbf{CF}(\mathbf{FL}^{\mathbf{NL}})$? Is it contained in $\mathbf{CF}(\mathbf{FP})$? In $\mathbf{Ker}(\mathbf{FP})$? We note that the straightforward binary search technique used to show $\mathbf{PEq} \subseteq \mathbf{LexEqFP}^{\mathbf{NP}}$ does not work in logarithmic space. Jenner and Torán [151] showed that the lexicographically minimal (or maximal—in this case the same technique works) solution of any **NL** search problem can be computed in $\mathbf{FL}^{\mathbf{NL}}$. However, the notion of an **NL** search problem is based on the following characterization of **NL** due to Lange [178]: a language A is in **NL** if and only if there is a a polynomial p and a log-space machine $M(x, \vec{y})$ that reads its second input in one direction only, indicated by "$\vec{y}$", such that

$$x \in A \iff (\exists y : |y| \leq p(|x|))[M(x, \vec{y}) = 1].$$

  Without the one-way restriction, this definition would give a characterization of **NP** rather than **NL**. An **NL** search problem is then: given such a machine M and input x, find a y such that $M(x, \vec{y}) = 1$. Any equivalence relation that can be decided by such a machine—that is, where $x \sim y$ if and only if $M(x, \vec{y}) = 1$—is in $\mathbf{LexEqFL}^{\mathbf{NL}}$, but it is not clear that this captures all of **LEq**.

- Does $\mathbf{CF}(\mathbf{FL}) = \mathbf{Ker}(\mathbf{FL})$ imply $\mathbf{NL} = \mathbf{UL}$? Note that $\mathbf{NL} = \mathbf{UL}$ if and only if $\mathbf{FL}^{\mathbf{NL}} \subseteq \#\mathbf{L}$ [11].

- Does $\mathbf{CF}(\mathbf{FL}) = \mathbf{LEq}$ imply $\mathbf{UL} \subseteq \mathbf{RL}$? A positive answer to this question and the previous one would give very strong evidence that $\mathbf{CF}(\mathbf{FL}) \neq \mathbf{LEq}$, as significant progress has been made towards showing $\mathbf{L} = \mathbf{RL}$ [223].

5.5.2 Additional Questions

In no particular order:

- Study expected polynomial-time canonical forms. If every $R \in \mathbf{Ker}(\mathbf{FP})$ has an expected polynomial-time canonical form, does **PH** collapse? An interesting example of an expected polynomial-time canonical form is that for GRAPH ISOMORPHISM [34].

- Find a class of groups for which the group membership problem is in **P** but no efficient complete invariant is known for the subgroup equality problem (see Section 5.3.2).

- If $\mathbf{Ker} = \mathbf{PEq}$, does **PH** collapse?

- $\mathbf{LexEqFP}^{\mathbf{\Sigma_i P}} \stackrel{?}{=} \mathbf{CF}(\mathbf{FP}^{\mathbf{\Sigma_i P}}) \stackrel{?}{=} \mathbf{Ker}(\mathbf{FP}^{\mathbf{\Sigma_i P}}) \stackrel{?}{=} \mathbf{P}^{\mathbf{\Sigma_i P}}\mathbf{Eq}$. If $\mathbf{Ker}(\mathbf{FP}^{\mathbf{\Sigma_i P}}) = \mathbf{P}^{\mathbf{\Sigma_i P}}\mathbf{Eq}$ does **PH** collapse?

- Study counting classes of equivalence relations. For an equivalence relation R, the associated counting function is $f(x) = \#\{y : y \sim_R x\}$.

- Preorders have been studied in the context of p-selectivity and semifeasible sets [169], and partial orders have been studied in the context of $\#\mathbf{P}$ and acceptance mechanisms for nondeterministic machines [137]. It would be interesting to develop these further, as well as to study complexity classes of lattices and total orders.

CHAPTER 6
CONCLUSION

In this thesis we highlighted just a few of the many ways that symmetry and equivalence relations play important roles in complexity theory, and we have taken a complexity-theoretic lens to the general question of equivalence relations. In this concluding chapter we elaborate on some of the ideas laid out in Chapter 1 in light of the intervening results and discussions.

In Chapter 1 we mentioned briefly the importance of finding a good equivalence relation on algorithms and algorithmic problems—where particularly nice equivalence relations are symmetry-based. Here we elaborate on this theme furhter. We begin with a quote from Weyl [275], as we could not put it better:

> To a certain degree this scheme is typical for all theoretic knowledge: We begin with some general but vague principle (symmetry in the [intuitive] sense), then find an important case where we can give that notion a concrete precise meaning (bilateral symmetry), and from that case we gradually rise again to generality, guided more by mathematical construction and abstraction than by the mirages of philosophy; and if we are lucky we end up with an idea no less universal than the one from which we started. Gone may be much of its emotional appeal, but it has the same or even greater unifying power in the realm of thought and is exact instead of vague. –Hermann Weyl, *Symmetry*, p. 6 [275]

In the case of complexity, we have an intuitive notion of equivalence between algorithms. This intuitive notion can be made precise in the case of algebraic complexity. But we have yet to rise again to full generality to get an equally precise and useful notion of equivalence between Turing machine algorithms.

However, in the case of complexity I am not even sure that we yet have the right intuitive notions. We certainly have some intuition for when two algorithms are the same. For example, in our intuitive notion it is immaterial what programming language an algorithm is written in. We may not all agree on when two algorithms are intuitively the same, but

this is part of the necessary vagueness of intuition. But our intuitive notion of equivalent algorithms have not yet given us intuitive notions that would let us separate complexity classes. This is in fact a good thing! For Blass, Dershowitz, and Gurevich [55] argue that there is no formal notion of two algorithms being "the same" that agrees with our intuitive notion of "equivalent algorithms." But we see that what we want is not a formalization of our intuitive notion of equivalent algorithms, but rather first we need a better intuition, that might help us settle the complexity of algorithmic problems.

We should also mention that there are in fact notions of equivalence between algorithms, but the ones we are aware of are either too fine or too coarse to be of use in understanding complexity. The two most natural notions are given by: on the one hand, saying that two algorithms are the same if they compute the same function, and on the other hand, saying that two algorithms are the same if they compute step-by-step in the same manner. The former is too coarse, since it completely ignores the resources used by the algorithm, and the latter is too fine, since there are algorithms of the same complexity, and indeed even algorithms that are intuitively the same, that are far from being step-by-step equivalent.

There are also notions of equivalence between algorithmic problems, but these only take us so far in understanding complexity. Certainly the various notions of reduction introduced by Post [217] and then taken up by the complexity community in the definition of **NP**-completeness and beyond, have been useful in our understanding of complexity. Indeed, in some sense these notions of reductions let us formalize what we even mean by two problems having the same complexity. But this notion of equivalence does not seem to be strong enough to point us towards the relevant properties of algorithmic problems that would let us settle their complexity once and for all. This remains true even of notions of reduction that are closely related to symmetries, such as p-isomorphism [52]. Somehow in all of these notions, the mathematical objects that get created—algorithmic problems up to Turing equivalence, or algorithmic problems up to p-isomorphism—are still too unwieldy to really understand.

This last statement is based on attempts by complexity theorists to treat complexity classes as individual objects of study and determine their most relevant properties, and the (so-far) failure of these attempts to actually separate complexity classes. For example, many researchers have tried to distinguish complexity classes by their properties, such as being closed under complement [144, 258], existence of complete sets, properties of their complete

sets [52, 12], or measure-like properties [188]. But determining these properties has generally turned out to be no easier than the original complexity class separation originally sought (e. g. Sipser's result [244] that resolving whether **BPP** has complete sets requires non-relativizing techniques). In the current state of the art, it is not clear how one might proceed with these approaches to make progress on unconditional complexity class separations. This reflects the fact that the equivalence relation of "having the same complexity" is very poorly understood, and does not (yet) lead to "nice" mathematical objects.

I believe that symmetry has a significant role to play in this regard in complexity. One of the reasons I find Geometric Complexity Theory exciting and promising is that it studies a notion of equivalence that is not only closely tied to complexity, but is also closely tied to other well-studied notions like symmetry, representation theory, and algebraic geometry. This gives us a toolkit with which to work to try to understand algorithms and problems up to this notion of equivalence. Furthermore, because algebraic varieties are "small" objects, we can ask about the complexity of problems on algebraic varieties, thus completing the circle by using complexity theory to study the methods by which we ultimately hope to understand complexity. We saw a similar phenomenon in Chapter 4 when we used an algorithmic problem on Lie algebras to understand an equivalence relation—namely, linear equivalence, especially in the case of determinant—where the equivalence relation is really a notion of complexity.

Although computation in general appears too complicated and messy for symmetry to play a role in it, little could be further from the truth. Beyond the connections between symmetry and complexity in this thesis, symmetry arises over and over again in complexity. The complexity of multiplying integers is naively $O(n^2)$, but using the fast Fourier transform—an idea ultimately based on symmetry—this complexity can be brought down to $O(n \log n \log \log n)$ [232], and more recently to $O(n \log n 2^{\log^* n})$ [117]. Cohn and Umans [87] have extended this idea to noncommutative groups to give algorithms for matrix multiplication [86]. Barrington's Theorem [40] characterizes $\mathbf{NC}^1$ in terms of a problem on the group of permutations of 5 elements. Babai, Beals, and Takácsi-Nagy [25], following Clote and Kranakis [82], show a close relationship between the circuit complexity of a Boolean function and structural properties of its symmetry group. Arvind and Vinodchandran [16] essentially show that any **NP** problem whose witnesses are invariant under a significant

group of symmetries is low for **PP**. Groups have played a crucial role in the best known algorithms for graph and hypergraph isomorphism [185, 26]. The hidden subgroup problem has played a central role in quantum algorithms and complexity [165, 173, 128, 146]. Recently, affine-invariant properties—properties of polynomials that are invariant under the group of affine linear transformations of the variables, including linearity and low-degreeness—have arisen in connection with locally correctable codes and property testing [158] (see also the survey by Sudan [257]). Surely there are more examples, but this list should suffice to make the point that symmetry and complexity are intimately related.

In recent years, representation theory has become more prevalent in complexity theory. It has long been known to be necessary in GROUP ISOMORPHISM, which is a key obstacle to polynomial-time algorithms for GRAPH ISOMORPHISM. Representation theory is obviously closely related to the isomorphism problems for abstract Lie algebras and matrix Lie algebras. And representation theory is used crucially in GCT. In addition to these uses, mostly presented in this thesis, representation theory has also been used in matrix multiplication [87, 86, 71], parametrized complexity [171], and coding theory [101].

A key roadblock in our current understanding of algorithms and representation theory is understanding indecomposable representations that are not completely reducible. Given the increasing uses of representation theory in complexity theory, we suspect that progress in complexity theory may benefit greatly from further understanding the symmetries inherent in algorithms and algorithmic problems, perhaps specifically with regards to non-completely reducible representations.

REFERENCES

[1] Scott Aaronson. Quantum lower bound for the collision problem. In *STOC '02: 34th Annual ACM Symposium on Theory of Computing*, pages 635–642. ACM, 2002.

[2] Scott Aaronson. **BQP** and the polynomial hierarchy. Technical Report TR09-104, Electronic Colloquium on Computational Complexity, 2009. Also available as arXiv e-print quant-ph/0910.4698.

[3] Scott Aaronson. Personal communication, November 2009.

[4] Scott Aaronson and Avi Wigderson. Algebrization: a new barrier in complexity theory. In *STOC '08: 40th Annual ACM Symposium on Theory of Computing*, pages 731–740. ACM, 2008.

[5] I. D. Ado. The representation of Lie algebras by matrices. *Uspehi Matem. Nauk (N.S.)*, 2(6(22)):159–173, 1947. English translation: The representation of Lie algebras by matrices, in American Mathematical Society Translations, Vol. 1949, No. 2, AMS, 1949, p. 21.

[6] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. PRIMES is in **P**. *Ann. of Math. (2)*, 160(2):781–793, 2004.

[7] Manindra Agrawal and Nitin Saxena. Equivalence of $\mathbb{F}$-algebras and cubic forms. In *STACS 2006*, volume 3884 of *Lecture Notes in Computer Science*, pages 115–126. Springer, Berlin, 2006.

[8] Manindra Agrawal and Thomas Thierauf. The formula isomorphism problem. *SIAM J. Comput.*, 30(3):990–1009, 2000.

[9] William Aiello and Johan Håstad. Statistical zero-knowledge languages can be recognized in two rounds. *J. Comput. System Sci.*, 42(3):327–345, 1991. FOCS '87: 28th Annual IEEE Symposium on Foundations of Computer Science.

[10] Jon L. Alperin. A Lie approach to finite groups. In *Groups—Canberra 1989*, volume 1456 of *Lecture Notes in Math.*, pages 1–9. Springer, Berlin, 1990.

[11] Carme Álvarez and Birgit Jenner. A very hard log-space counting class. *Theoret. Comput. Sci.*, 107(1):3–30, 1993.

[12] Klaus Ambos-Spies. **P**-mitotic sets. In *Logic and machines: decision problems and complexity (Münster, 1983)*, volume 171 of *Lecture Notes in Computer Science*, pages 1–23. Springer, Berlin, 1984.

[13] Uri Andrews, Steffen Lempp, Joseph S. Miller, Keng Meng Ng, Luca San Mauro, and Andrea Sorbi. Universal computably enumerable equivalence relations. Submitted; available at `http://www.math.wisc.edu/~jmiller/Papers/ceers.pdf`, 2012.

[14] Sanjeev Arora and Boaz Barak. *Computational complexity. A modern approach*. Cambridge University Press, Cambridge, 2009.

[15] Sanjeev Arora, Russell Impagliazzo, and Umesh Vazirani. Relativizing versus nonrelativizing techniques: the role of local checkability, 1988. Latest update 2007. Available at `http://www.cs.berkeley.edu/~vazirani/pubs/relativizing.ps` and `http://cseweb.ucsd.edu/~russell/ias.ps`.

[16] V. Arvind and N. V. Vinodchandran. The counting complexity of group-definable languages. *Theoret. Comput. Sci.*, 242(1-2):199–218, 2000.

[17] Michael Aschbacher. The status of the classification of the finite simple groups. *Notices Amer. Math. Soc.*, 51(7):736–740, 2004.

[18] Albert A. Atserias. Distinguishing SAT from polynomial-size circuits, through black-box queries. In *CCC '06: 21st IEEE Conference on Computational Complexity*, pages 88–95. IEEE, 2006. Also available as ECCC Tech. Report TR05-154.

[19] Amir Averbuch, Zvi Galil, and Shmuel Winograd. Classification of all the minimal bilinear algorithm for computing the coefficients of the product of two polynomials modulo a polynomial, part I: the algebra $G[u]/\langle Q(u)^l\rangle, l > 1$. *Theoret. Comput. Sci.*, 58:17–56, 1988.

[20] László Babai. Representation of permutation groups by graphs. In *Combinatorial theory and its applications, I (Proc. Colloq., Balatonfüred, 1969)*, pages 55–80. North-Holland, Amsterdam, 1970.

[21] László Babai. Monte Carlo algorithms in graph isomorphism testing. Technical Report DMS 79-10, Université de Montréal, 1979.

[22] László Babai. Trading group theory for randomness. In *STOC '85: 17th Annual ACM Symposium on Theory of Computing*, pages 421–429. ACM, 1985.

[23] László Babai. Personal communication, May 2008. Canonical generators for permutation groups.

[24] László Babai. Personal communication, April 2012.

[25] László Babai, Robert Beals, and Pál Takácsi-Nagy. Symmetry and complexity. In *STOC '92: 24th Annual ACM Symposium on Theory of Computing*, pages 438–449. ACM, 1992.

[26] László Babai and Paolo Codenotti. Isomorphism of hypergraphs of low rank in moderately exponential time. In *FOCS '08: 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 667–676. IEEE Computer Society, 2008.

[27] László Babai, Paolo Codenotti, Joshua A. Grochow, and Youming Qiao. Code equivalence and group isomorphism. In *SODA '11: ACM–SIAM Symposium on Discrete Algorithms*, 2011.

[28] László Babai, Paolo Codenotti, and Youming Qiao. Polynomial-time isomorphism test for groups with no abelian normal subgroups, 2012. To appear, ICALP '12.

[29] László Babai and Lance Fortnow. A characterization of #**P** by arithmetic straight line programs. In *FOCS '90: 31st Annual IEEE Symposium on Foundations of Computer Science*, pages 26–34. IEEE Comput. Soc. Press, 1990.

[30] László Babai and Lance Fortnow. Arithmetization: a new method in structural complexity theory. *Comput. Complexity*, 1(1):41–66, 1991.

[31] László Babai, Lance Fortnow, and Carsten Lund. Nondeterministic exponential time has two-prover interactive protocols. In *FOCS '90: 31st Annual IEEE Symposium on Foundations of Computer Science*, volume 1, pages 16–25. IEEE Computer Society, 1990.

[32] László Babai, D. Yu. Grigoryev, and David M. Mount. Isomorphism of graphs with bounded eigenvalue multiplicity. In *STOC '82: 14th Annual ACM Symposium on Theory of Computing*, pages 310–324. ACM, 1982.

[33] László Babai, William M. Kantor, and Eugene M. Luks. Computational complexity and the classification of finite simple groups. In *FOCS '83: 24th Annual IEEE Symposium on Foundations of Computer Science*, pages 162–171, Los Alamitos, CA, USA, 1983. IEEE Computer Society.

[34] László Babai and Ludik Kučera. Canonical labelling of graphs in linear average time. In *FOCS '79: 20th Annual IEEE Symposium on Foundations of Computer Science*, pages 39–46, 1979.

[35] László Babai and Eugene M. Luks. Canonical labeling of graphs. In *STOC '83: 15th Annual ACM Symposium on Theory of Computing*, pages 171–183. ACM, 1983.

[36] László Babai and Ákos Seress. Personal communication to J. von zur Gathen (cf. [274]), March 1987.

[37] Eric Bach and Victor Shoup. Factoring polynomials using fewer random bits. *J. Symbolic Comput.*, 9(3):229–239, 1990.

[38] Ted Baker, John Gill, and Robert Solovay. Relativizations of the **P** =? **NP** question. *SIAM J. Comput*, 4:431–442, 1975.

[39] S. Bakhtiari, R. Safavi-naini, and J. Pieprzyk. Cryptographic hash functions: a survey. Technical report, Department of Computer Science, University of Wollongong, 1995.

[40] David A. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in $\mathbf{NC}^1$. *J. Comput. System Sci.*, 38(1):150–164, 1989.

[41] David A. Mix Barrington, Neil Immerman, and Howard Straubing. On uniformity within $\mathbf{NC}^1$. *J. Comput. System Sci.*, 41(3):274–306, 1990.

[42] Saugata Basu, Richard Pollack, and Marie-Françoise Roy. *Algorithms in real algebraic geometry*, volume 10 of *Algorithms and Computation in Mathematics*. Springer-Verlag, Berlin, second edition, 2006.

[43] Robert Beals. Quantum computation of Fourier transforms over symmetric groups. In *STOC '97: 29th Annual ACM Symposium on Theory of Computing*, pages 48–53. ACM, 1997.

[44] Robert E. Beck, Bernard Kolman, and Ian N. Stewart. Computing the structure of a Lie algebra. In *Computers in nonassociative rings and algebras (Special session, 82nd Annual Meeting Amer. Math. Soc., San Antonio, Tex., 1976)*, pages 167–188. Academic Press, New York, 1977.

[45] Richard Beigel. Perceptrons, **PP**, and the polynomial hierarchy. *Comput. Complexity*, 4(4):339–349, 1994. Special issue on circuit complexity (Barbados, 1992).

[46] Genrich Belitskii, Andrii R. Dmytryshyn, Ruvim Lipyanski, Vladimir V. Sergeichuk, and Arkady Tsurkov. Problems of classifying associative or Lie algebras over a field of characteristic not two and finite metabelian groups are wild. *Electron. J. Linear Algebra*, 18:516–529, 2009.

[47] Genrich Belitskii, Ruvim Lipyanski, and Vladimir Sergeichuk. Problems of classifying associative or Lie algebras and triples of symmetric or skew-symmetric matrices are wild. *Linear Algebra Appl.*, 407:249–262, 2005.

[48] Michael Ben-Or. Lower bounds for algebraic computation trees. In *STOC '83: 15th Annual ACM Symposium on Theory of Computing*, pages 80–86. ACM, 1983.

[49] Michael Ben-Or and Richard Cleve. Computing algebraic formulas using a constant number of registers. *SIAM J. Comput*, 21(1):54–58, 1992.

[50] Elwyn R. Berlekamp. Factoring polynomials over finite fields. *Bell System Tech. J.*, 46:1853–1859, 1967.

[51] Elwyn R. Berlekamp. Factoring polynomials over large finite fields. *Math. Comp.*, 24:713–735, 1970.

[52] Leonard Berman and Juris Hartmanis. On isomorphisms and density of **NP** and other complete sets. *SIAM J. Comput*, 6(2):305–322, 1977.

[53] Dario Bini. Relations between exact and approximate bilinear algorithms. Applications. *Calcolo*, 17(1):87–97, 1980.

[54] Jean-Camille Birget, Alexander Yu. Ol′shanskii, Eliyahu Rips, and Mark V. Sapir. Isoperimetric functions of groups and computational complexity of the word problem. *Ann. of Math. (2)*, 156(2):467–518, 2002.

[55] Andreas Blass, Nachum Dershowitz, and Yuri Gurevich. When are two algorithms the same? *Bull. Symbolic Logic*, 15(2):145–168, 2009.

[56] Andreas Blass and Yuri Gurevich. Equivalence relations, invariants, and normal forms. *SIAM J. Comput*, 13(4):682–689, 1984.

[57] Andreas Blass and Yuri Gurevich. Equivalence relations, invariants, and normal forms, II. In *Logic and Machines: Decision Problems and Complexity*, volume 171 of *Lecture Notes in Computer Science*, pages 24–42. Springer, 1984.

[58] Lenore Blum, Mike Shub, and Steve Smale. On a theory of computation and complexity over the real numbers: **NP**-completeness, recursive functions and universal machines. *Bull. Amer. Math. Soc. (N.S.)*, 21(1):1–46, 1989.

[59] Manuel Blum. A machine-independent theory of the complexity of recursive functions. *J. Assoc. Comput. Mach.*, 14:322–336, 1967.

[60] William W. Boone. Certain simple, unsolvable problems of group theory. V, VI. *Nederl. Akad. Wetensch. Proc. Ser. A. 60 = Indag. Math.*, 19:22–27, 227–232, 1957.

[61] Ravi Boppana, Johan Håstad, and Stathis Zachos. Does **co-NP** have short interactive proofs? *Inform. Process. Lett.*, 25:27–32, 1987.

[62] I. Z. Bouwer. Section graphs for finite permutation groups. *J. Combinatorial Theory*, 6:378–386, 1969.

[63] Gilles Brassard and Peter Høyer. An exact quantum polynomial-time algorithm for Simon's problem. In *Proc. 5th Israeli Symp. on Theory of Computing Systems*, pages 12–23. IEEE Computer Society, 1997.

[64] Nader H. Bshouty, Richard Cleve, Sampath Kannan, and Christino Tamon. Oracles and queries that are sufficient for exact learning. *J. Comput. System Sci.*, 52:421–433, 1996.

[65] Harry Buhrman and Lance Fortnow. Two queries. *J. Comput. System Sci.*, 59(2):182–194, 1999. Special issue for selected papers from the 13th IEEE Conference on Computational Complexity.

[66] Dietrich Burde, Bettina Eick, and Willem de Graaf. Computing faithful representations for nilpotent Lie algebras. *J. Algebra*, 322(3):602–612, 2009.

[67] Dietrich Burde and Wolfgang Moens. Minimal faithful representations of reductive Lie algebras. *Arch. Math. (Basel)*, 89(6):513–523, 2007.

[68] Dietrich Burde and Wolfgang Alexander Moens. Faithful Lie algebra modules and quotients of the universal enveloping algebra. *J. Algebra*, 325:440–460, 2011.

[69] Peter Bürgisser. *Completeness and reduction in algebraic complexity theory*, volume 7 of *Algorithms and Computation in Mathematics*. Springer-Verlag, Berlin, 2000.

[70] Peter Bürgisser, Michael Clausen, and M. Amin Shokrollahi. *Algebraic complexity theory*, volume 315 of *Grundlehren der Mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences]*. Springer-Verlag, Berlin, 1997. With the collaboration of Thomas Lickteig.

[71] Peter Bürgisser and Christian Ikenmeyer. Geometric complexity theory and tensor rank. In *STOC '11: 43rd Annual ACM Symposium on Theory of Computing*, pages 509–518. ACM, 2011.

[72] Jin-Yi Cai. A note on the determinant and permanent problem. *Inform. and Comput.*, 84(1):119–127, 1990.

[73] Jin-Yi Cai, Venkatesan T. Chakaravarthy, Lane A. Hemaspaandra, and Mitsunori Ogihara. Competing provers yield improved Karp-Lipton collapse results. *Inform. and Comput.*, 198(1):1–23, 2005.

[74] Jin-Yi Cai, Xi Chen, and Dong Li. Quadratic lower bound for permanent vs. determinant in any characteristic. *Comput. Complexity*, 19(1):37–56, 2010.

[75] Ran Canetti. More on **BPP** and the polynomial-time hierarchy. *Inform. Process. Lett.*, 57(5):237–241, 1996.

[76] Marvin Chester. Is symmetry identity? arXiv e-print physics.hist-ph/1202.0292, 2012.

[77] Claude Chevalley. *Theory of Lie Groups. I.* Princeton Mathematical Series, vol. 8. Princeton University Press, Princeton, N. J., 1946.

[78] Andrew M. Childs, Richard Cleve, Enrico Deotto, Edward Farhi, Sam Gutmann, and Daniel A. Spielman. Exponential algorithmic speedup by a quantum walk. In *STOC '03: 35th Annual ACM Symposium on Theory of Computing*, pages 59–68. ACM, 2003.

[79] Alexander L. Chistov and Dima Yu. Girgoryev. Polynomial time factoring of the multivariable polynomials over a global field. LOMI Preprint E-5-82, 1982.

[80] Clay Mathematics Institute. Millenium prize problems, 2000.

[81] Peter Clote and Jan Krajíček, editors. *Arithmetic, proof theory, and computational complexity*, volume 23 of *Oxford Logic Guides*. The Clarendon Press Oxford University

Press, 1993. Papers from the conference held in Prague, July 2–5, 1991, Oxford Science Publications.

[82] Peter Clote and Evangelos Kranakis. Boolean functions, invariance groups, and parallel complexity. *SIAM J. Comput*, 20(3):553–590, 1991.

[83] Paolo Codenotti. *Testing isomorphism of combinatorial and algebraic structures*. PhD thesis, University of Chicago, Chicago, IL, 2011.

[84] Paul Cohen. The independence of the continuum hypothesis. *Proc. Nat. Acad. Sci. U.S.A.*, 50:1143–1148, 1963.

[85] Paul J. Cohen. The independence of the continuum hypothesis. II. *Proc. Nat. Acad. Sci. U.S.A.*, 51:105–110, 1964.

[86] Henry Cohn, Robert Kleinberg, Balazs Szegedy, and Christopher Umans. Group-theoretic algorithms for matrix multiplication. In *FOCS '05: 46th Annual IEEE Symposium on Foundations of Computer Science*, pages 379–388. IEEE Computer Society, 2005.

[87] Henry Cohn and Christopher Umans. A group-theoretic approach to fast matrix multiplication. In *FOCS '03: 44th Annual IEEE Symposium on Foundations of Computer Science*, pages 438–449. IEEE Computer Society, 2003.

[88] Stephen Cook. The complexity of theorem-proving procedures. In *STOC '71: 3rd Annual ACM Symposium on Theory of Computing*, pages 151–158, 1971.

[89] Don Coppersmith. Modifications to the number field sieve. *J. Cryptology*, 6(3):169–180, 1993.

[90] Felipe Cucker, Marek Karpinski, Pascal Koiran, Thomas Lickteig, and Kai Werther. On real Turing machines that toss coins. In *STOC '95: 27th Annual ACM Symposium on Theory of Computing*, pages 335–342. ACM, 1995.

[91] Charles W. Curtis. Representations of Lie algebras of classical type with applications to linear groups. *J. Math. Mech.*, 9:307–326, 1960.

[92] Ivan Damgård. Collision free hash functions and public key signature schemes. In *EuroCrypt87*, volume 304 of *Lecture Notes in Computer Science*, pages 203–216. Springer, 1988.

[93] C. Damm. $\mathbf{DET}=\mathbf{L}^{\#\mathbf{L}}$. Technical Report Informatik-Preprint 8, Fachbereich Informatik der Humboldt-Universität zu Berlin, 1991.

[94] Willem de Graaf, Gábor Ivanyos, and Lajos Rónyai. Computing Cartan subalgebras of Lie algebras. *Appl. Algebra Engrg. Comm. Comput.*, 7(5):339–349, 1996.

[95] Willem A. de Graaf. Calculating the structure of a semisimple Lie algebra. *J. Pure Appl. Algebra*, 117/118:319–329, 1997. Algorithms for algebra (Eindhoven, 1996).

[96] Willem A. de Graaf. Constructing faithful matrix representations of Lie algebras. In *ISSAC '97: International Symposium on Symbolic and Algebraic Computation*, pages 54–59. ACM, 1997.

[97] Willem A. de Graaf. *Lie algebras: theory and algorithms*, volume 56 of *North-Holland Mathematical Library*. North-Holland Publishing Co., Amsterdam, 2000.

[98] Willem A. de Graaf, Gábor Ivanyos, Alex Küronya, and Lajos Rónyai. Computing Levi decompositions in Lie algebras. *Appl. Algebra Engrg. Comm. Comput.*, 8(4):291–303, 1997.

[99] Hans F. de Groote. On varieties of optimal algorithms for the computation of bilinear mappings. II. Optimal algorithms for 2×2-matrix multiplication. *Theoret. Comput. Sci.*, 7(2):127–148, 1978.

[100] Jack Edmonds. Paths, trees, and flowers. *Canad. J. Math.*, 17:449–467, 1965.

[101] Klim Efremenko. From irreducible representations to locally decodable codes. Technical Report TR11-154, Electronic Colloquium on Computational Complexity, 2011.

[102] Mark Ettinger and Peter Høyer. A quantum observable for the graph isomorphism problem. arXiv e-print quant-ph/9901029, 1999.

[103] Jacob Feldman and Calvin C. Moore. Ergodic equivalence relations, cohomology, and von Neumann algebras. I. *Trans. Amer. Math. Soc.*, 234(2):289–324, 1977.

[104] Stephen A. Fenner, Lance Fortnow, Stuart A. Kurtz, and Lide Li. An oracle builder's toolkit. *Inform. and Comput.*, 182(2):95–136, 2003.

[105] Lance Fortnow. The complexity of perfect zero-knowledge. In *STOC '87: 19th Annual ACM Symposium on Theory of Computing*, pages 204–209. ACM, 1987.

[106] Lance Fortnow. The role of relativization in complexity theory. *Bull. Euro. Assoc. Theoret. Comput. Sci.*, 52:229–244, 1994. Computational complexity column.

[107] Lance Fortnow. Counting complexity. In *Complexity theory retrospective, II*, pages 81–107. Springer, New York, 1997.

[108] Lance Fortnow. The status of the **P** versus **NP** problem. *Comm. Assoc. Comput. Mach.*, 52(9):78–86, September 2009.

[109] Lance Fortnow and Joshua A. Grochow. Complexity classes of equivalence problems revisited. *Inform. and Comput.*, 209(4):748–763, 2011. Also available as arXiv e-print cs/0907.4775.

[110] Lance Fortnow, A. Pavan, and Samik Sengupta. Proving SAT does not have small circuits with an application to the two queries problem. *J. Comput. System Sci.*, 2006. Special issue for selected papers from the 18th IEEE Conference on Computational Complexity.

[111] Lance Fortnow and Michael Sipser. Are there interactive protocols for **co-NP** languages? *Inform. Process. Lett.*, 28(5):249–251, 1988.

[112] Katalin Friedl, Gábor Ivanyos, Frédéric Magniez, Miklos Santha, and Pranab Sen. Hidden translation and orbit coset in quantum computing. In *STOC '03: 35th Annual ACM Symposium on Theory of Computing*, pages 1–9. ACM, 2003.

[113] Katalin Friedl and Lajos Rónyai. Polynomial time solutions of some problems of computational algebra. In *STOC '85: 17th Annual ACM Symposium on Theory of Computing*, pages 153–162. ACM, 1985.

[114] F. Georg Frobenius. Uber die Darstellung der endlichen Gruppen durch lineare Substitutionen. *Sitzungsber Deutsch. Akad. Wiss. Berlin*, pages 994–1015, 1897.

[115] A. Fröhlich and J. C. Shepherdson. Effective procedures in field theory. *Philos. Trans. Roy. Soc. London. Ser. A.*, 248:407–432, 1956.

[116] William Fulton and Joe Harris. *Representation theory*, volume 129 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 1991. A first course, Readings in Mathematics.

[117] Martin Fürer. Faster integer multiplication. *SIAM J. Comput*, 39(3):979–1005, 2009.

[118] Martin Fürer, Walter Schnyder, and Ernst Specker. Normal forms for trivalent graphs and graphs of bounded valence. In *STOC '83: 15th Annual ACM Symposium on Theory of Computing*, pages 161–170. ACM, 1983.

[119] Merrick Furst, John Hopcroft, and Eugene Luks. Polynomial-time algorithms for permutation groups. In *FOCS '80: 21st Annual IEEE Symposium on Foundations of Computer Science*, pages 36–41. IEEE, 1980.

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

[121] Su Gao and Peter Gerdes. Computably enumerable equivalence relations. *Studia Logica*, 67(1):27–59, 2001.

[122] Howard Georgi. *Lie algebras in particle physics*, volume 54 of *Frontiers in Physics*. Benjamin/Cummings Publishing Co. Inc. Advanced Book Program, Reading, Mass., 1982. From isospin to unified theories, With an introduction by Sheldon L. Glashow.

[123] Christian Glaßer, Christian Reitwießner, and Victor Selivanov. The shrinking property for **NP** and **coNP**. *Theoret. Comput. Sci.*, 412(8-10):853–864, 2011.

[124] Kurt Gödel. Letter to J. von Neumann, 20 March 1956. English translations available in [246] [81, Preface], [183, `http://rjlipton.wordpress.com/the-gdel-letter/`].

[125] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity, and a methodology of cryptographic protocol design. In *FOCS '86: 27th Annual IEEE Symposium on Foundations of Computer Science*, pages 174–187, 1986.

[126] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput*, 18(1):186–208, 1989.

[127] Shafi Goldwasser and Michael Sipser. Private coins versus public coins in interactive proof systems. In *STOC '86: 18th Annual ACM Symposium on Theory of Computing*, pages 59–68. ACM, 1986.

[128] Michelangelo Grigni, Leonard J. Schulman, Monica Vazirani, and Umesh Vazirani. Quantum mechanical algorithms for the nonabelian hidden subgroup problem. *Combinatorica*, 24(1):137–154, 2004.

[129] Joshua A. Grochow. Matrix Lie algebra isomorphism. In *CCC '12: 27th IEEE Conference on Computational Complexity*. IEEE, 2012. Also available as arXiv e-print cs/1112.2012 and ECCC Tech. Report TR11-168.

[130] Joshua A. Grochow and Korben Rusek. Report on "Mathematical Aspects of **P** vs. **NP** and its Variants". arXiv e-print cs.CC/1203.2888, 2012. Workshop held at Brown-ICERM in August, 2011, organizers: Saugata Basu, J. M. Landsberg, and J. Maurice Rojas.

[131] Fritz Grunewald and Daniel Segal. Some general algorithms. II. Nilpotent groups. *Ann. of Math. (2)*, 112(3):585–617, 1980.

[132] Yuri Gurevich. From invariants to canonization. *Bulletin of the EATCS*, 63:115–119, 1997.

[133] Yuri Gurevich and Paul Schupp. Membership problem for the modular group. *SIAM J. Comput*, 37(2):425–459, 2007.

[134] Leonid Gurvits. Classical complexity and quantum entanglement. *J. Comput. System Sci.*, 69(3):448–484, 2004.

[135] Leonid Gurvits. Combinatorial and algorithmic aspects of hyperbolic polynomials. arXiv e-print math.CO/0404474, 2004.

[136] Leonid Gurvits. On the complexity of mixed discriminants and related problems. In *MFCS '05: Symposium on Mathematical Foundations of Computer Science*, volume 3618 of *Lecture Notes in Computer Science*, pages 447–458. Springer, Berlin, 2005.

[137] Lane A. Hemaspaandra, Christopher M. Homan, Sven Kosub, and Klaus W. Wagner. The complexity of computing the size of an interval. *SIAM J. Comput*, 36(5):1264–1300, 2006.

[138] Lane A. Hemaspaandra, Ashish V. Naik, Mitsunori Ogihara, and Alan L. Selman. Computing solutions uniquely collapses the polynomial hierarchy. *SIAM J. Comput*, 25(4):697–708, 1996. Also available as ECCC Tech. Report TR96-027; preliminary version in Springer LNCS vol. 834, 1994.

[139] William Hesse, Eric Allender, and David A. Mix Barrington. Uniform constant-depth threshold circuits for division and iterated multiplication. *J. Comput. System Sci.*, 65(4):695–716, 2002. Special issue for selected papers from the 16th IEEE Conference on Computational Complexity.

[140] Gerhard Hochschild. Representations of restricted Lie algebras of characteristic p. *Proc. Amer. Math. Soc.*, 5:603–605, 1954.

[141] John Hopcroft and J. K. Wong. Linear time algorithm for isomorphism of planar graphs (preliminary report). In *STOC '74: 6th Annual ACM Symposium on Theory of Computing*, pages 172–184. ACM, 1974.

[142] John E. Hopcroft and Robert E. Tarjan. Isomorphism of planar graphs. In *Complexity of computer computations (Proc. Sympos., IBM Thomas J. Watson Res. Center, Yorktown Heights, N. Y., 1972)*, pages 131–152, 187–212. Plenum, New York, 1972.

[143] James E. Humphreys. *Introduction to Lie algebras and representation theory*, volume 9 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 1978. Second printing, revised.

[144] Neil Immerman. Nondeterministic space is closed under complementation. *SIAM J. Comput*, 17(5):935–938, 1988.

[145] Russell Impagliazzo, Valentine Kabanets, and Antonina Kolokolova. An axiomatic approach to algebrization. In *STOC '09: 41st Annual ACM Symposium on Theory of Computing*, pages 695–704. ACM, 2009.

[146] Gábor Ivanyos, Frédéric Magniez, and Miklos Santha. Efficient quantum algorithms for some instances of the non-abelian hidden subgroup problem. *Internat. J. Found. Comput. Sci.*, 14(5):723–739, 2003.

[147] Gábor Ivanyos and Lajos Rónyai. Computations in associative and Lie algebras. In *Some tapas of computer algebra*, volume 4 of *Algorithms Comput. Math.*, pages 91–120. Springer, Berlin, 1999.

[148] Kenkichi Iwasawa. On the representation of Lie algebras. *Jap. J. Math.*, 19:405–426, 1948.

[149] Nathan Jacobson. A note on Lie algebras of characteristic p. *Amer. J. Math.*, 74:357–359, 1952.

[150] Nathan Jacobson. *Lie algebras*. Interscience Tracts in Pure and Applied Mathematics, No. 10. Interscience Publishers (a division of John Wiley & Sons), New York-London, 1962.

[151] Birgit Jenner and Jacobo Torán. The complexity of obtaining solutions for problems in **NP** and **NL**. In *Complexity Theory Retrospective, II*, pages 155–178. Springer, New York, 1997.

[152] J. Howard Johnson. Rational equivalence relations. In Laurent Kott, editor, *ICALP '86: 13th International Colloquium on Automata, Languages and Programming*, volume 226 of *Lecture Notes in Computer Science*, pages 167–176. Springer, 1986.

[153] Valentine Kabanets and Russell Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. *Comput. Complexity*, 13(1-2):1–46, 2004.

[154] Harlan Kadish and J. M. Landsberg. Padded polynomials, their cousins, and geometric complexity theory. arXiv e-print math.AG/1204.4693, 2012.

[155] Erich Kaltofen. Polynomial factorization 1987–1991. In Imre Simon, editor, *LATIN '92*, volume 583 of *Lecture Notes in Computer Science*, pages 294–313. Springer Berlin / Heidelberg, 1992.

[156] Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103, 1972.

[157] Richard M. Karp and Richard J. Lipton. Turing machines that take advice. *Enseign. Math. (2)*, 28(3-4):191–209, 1982.

[158] Tali Kaufman and Madhu Sudan. Algebraic property testing: the role of invariance. In *STOC '08: 40th Annual ACM Symposium on Theory of Computing*, pages 403–412. ACM, 2008.

[159] Neeraj Kayal. Affine projections of polynomials. Technical Report TR11-061, Electronic Colloquium on Computational Complexity, 2011.

[160] Neeraj Kayal. Efficient algorithms for some special cases of the polynomial equivalence problem. In *SODA '11: ACM–SIAM Symposium on Discrete Algorithms*, 2011.

[161] Neeraj Kayal and Nitin Saxena. Complexity of ring morphism problems. *Comput. Complexity*, 15(4):342–390, 2006.

[162] Olga Kharlampovich, Alexei Myasnikov, and Mark V. Sapir. Residually finite finitely presented solvable groups. arXiv e-print math.GR/1204.6506, 2012.

[163] Evgenii I. Khukhro. *Nilpotent groups and their automorphisms*, volume 8 of *de Gruyter Expositions in Mathematics*. Walter de Gruyter & Co., Berlin, 1993.

[164] Evgenii I. Khukhro. *p-automorphisms of finite p-groups*, volume 246 of *London Mathematical Society Lecture Note Series*. Cambridge University Press, Cambridge, 1998.

[165] Alexei Kitaev. Quantum measurements and the abelian stabilizer problem. arXiv e-print quant-ph/9511026, 1995.

[166] Anthony W. Knapp. *Lie groups beyond an introduction*, volume 140 of *Progress in Mathematics*. Birkhäuser Boston Inc., Boston, MA, second edition, 2002.

[167] Donald E. Knuth. Efficient representation of perm groups. *Combinatorica*, 11(1):33–43, 1991.

[168] Ker-I Ko. Some observations on the probabilistic algorithms and **NP**-hard problems. *Inform. Process. Lett.*, 14(1):39–43, 1982.

[169] Ker-I Ko. On self-reducibility and weak **P**-selectivity. *J. Comput. System Sci.*, 26(2):209–221, 1983.

[170] Johannes Köbler and Osamu Watanabe. New collapse consequences of **NP** having small circuits. *SIAM J. Comput*, 28(1):311–324, 1999.

[171] Ioannis Koutis and Ryan Williams. Limits and applications of group algebras for parameterized problems. In *Automata, languages and programming. Part I*, volume 5555 of *Lecture Notes in Computer Science*, pages 653–664. Springer, Berlin, 2009.

[172] Leopold Kronecker. Grundzüge einer arithmetischen Theorie der algebraischen Grössen. *J. reine angew. Math.*, 92:1–122, 1882.

[173] Greg Kuperberg. A subexponential-time quantum algorithm for the dihedral hidden subgroup problem. *SIAM J. Comput*, 35(1):170–188, 2005.

[174] Susan Landau. Factoring polynomials over algebraic number fields. *SIAM J. Comput*, 14(1):184–195, 1985.

[175] J. M. Landsberg. Geometry and the complexity of matrix multiplication. *Bull. Amer. Math. Soc. (N.S.)*, 45(2):247–284, 2008.

[176] J. M. Landsberg. *Tensors: geometry and applications*, volume 128 of *Graduate Studies in Mathematics*. American Mathematical Society, Providence, RI, 2012.

[177] J. M. Landsberg, Laurent Manivel, and Nicolas Ressayre. Hypersurfaces with degenerate duals and the Geometric Complexity Theory Program. arXiv e-print math.AG/1004.4802, 2010.

[178] Klaus-Jörn Lange. Two characterizations of the logarithmic alternation hierarchy. In *MFCS '86: 12th Symposium on Mathematical Foundations of Computer Science*, volume 233 of *Lecture Notes in Computer Science*, pages 518–526. Springer-Verlag, 1986.

[179] Clemens Lautemann. **BPP** and the polynomial hierarchy. *Inform. Process. Lett.*, 17(4):215–217, 1983.

[180] Arjen K. Lenstra. Factoring polynomials over algebraic number fields. In *Computer algebra (London, 1983)*, volume 162 of *Lecture Notes in Computer Science*, pages 245–254. Springer, Berlin, 1983.

[181] Arjen K. Lenstra, Hendrik W. Lenstra, Jr., and László Lovász. Factoring polynomials with rational coefficients. *Math. Ann.*, 261(4):515–534, 1982.

[182] Martin W. Liebeck. On graphs whose full automorphism group is an alternating group or a finite classical group. *Proc. London Math. Soc. (3)*, 47(2):337–362, 1983.

[183] Richard Lipton and Kenneth Regan. Gödel's lost letter and **P** = **NP**. `http://rjlipton.wordpress.com/`. A weblog.

[184] Richard J. Lipton and Yechezkel Zalcstein. Word problems solvable in logspace. *J. Assoc. Comput. Mach.*, 24(3):522–526, 1977.

[185] Eugene M. Luks. Isomorphism of graphs of bounded valence can be tested in polynomial time. *J. Comput. System Sci.*, 25(1):42–65, 1982.

[186] Eugene M. Luks. Hypergraph isomorphism and structural equivalence of Boolean functions. In *STOC '99: 31st Annual ACM Symposium on Theory of Computing*, pages 652–658. ACM, 1999.

[187] Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. In *FOCS '90: 31st Annual IEEE Symposium on Foundations of Computer Science*, pages 2–10 vol.1, Washington, DC, USA, 1990. IEEE Computer Society.

[188] Jack H. Lutz. Category and measure in complexity classes. *SIAM J. Comput*, 19(6):1100–1131, 1990.

[189] Meena Mahajan and V. Vinay. Determinant: combinatorics, algorithms, and complexity. *Chicago J. Theoret. Comput. Sci.*, pages Article 5, 26 pp. (electronic), 1997.

[190] Anatoliĭ I. Mal′cev. On a class of homogeneous spaces. *Izvestiya Akad. Nauk. SSSR. Ser. Mat.*, 13:9–32, 1949.

[191] Guillaume Malod and Natacha Portier. Characterizing Valiant's algebraic complexity classes. *J. Complexity*, 24(1):16–38, 2008.

[192] Marvin Marcus and F. C. May. The permanent function. *Canad. J. Math.*, 14:177–189, 1962.

[193] Ernst Mayr. Membership in polynomial ideals over $\mathbb{Q}$ is exponential space complete. In *STACS '89: 6th Annual Symposium on Theoretical Aspects of Computer Science*, volume 349 of *Lecture Notes in Computer Science*, pages 400–406. Springer, Berlin, 1989.

[194] Ernst W. Mayr. Some complexity results for polynomial ideals. *J. Complexity*, 13(3):303–325, 1997.

[195] Ernst W. Mayr and Albert R. Meyer. The complexity of the word problems for commutative semigroups and polynomial ideals. *Adv. in Math.*, 46(3):305–329, 1982.

[196] Nadia Mazza. Finite p-groups in representation theory. `http://sma.epfl.ch/~dietler/Site/Mazza.html` or `http://www.maths.lancs.ac.uk/~mazza/ln-p-gps.pdf`, June 2010. Lecture notes from the summer school of the doctoral program in mathematics, EPFL.

[197] Curt McMullen. *Families of rational maps and iterative root-finding algorithms (dynamics, complex analysis, Newton's method)*. PhD thesis, Harvard University, 1985.

[198] Curt McMullen. Families of rational maps and iterative root-finding algorithms. *Ann. of Math. (2)*, 125(3):467–493, 1987.

[199] Curt McMullen. Braiding of the attractor and the failure of iterative algorithms. *Invent. Math.*, 91(2):259–272, 1988.

[200] Roy Meshulam. On two extremal matrix problems. *Linear Algebra Appl.*, 114/115:261–271, 1989.

[201] Thierry Mignon and Nicolas Ressayre. A quadratic bound for the determinant and permanent problem. *Int. Math. Res. Not.*, (79):4241–4253, 2004.

[202] Gary Miller. Isomorphism testing for graphs of bounded genus. In *STOC '80: 12th Annual ACM Symposium on Theory of Computing*, pages 225–235. ACM, 1980.

[203] Gary L. Miller. On the $n^{\log n}$ isomorphism technique (a preliminary report). In *STOC '78: 10th Annual ACM Symposium on Theory of Computing*, pages 51–58. ACM Press, 1978.

[204] W. H. Mills and George B. Seligman. Lie algebras of classical type. *J. Math. Mech.*, 6:519–548, 1957.

[205] Ketan D. Mulmuley. Geometric complexity theory IX: on the flip over fields of positive characteristic. In preperation.

[206] Ketan D. Mulmuley. On **P** vs. **NP**, Geometric Complexity Theory, and the flip I: a high-level view. Technical Report TR-2007-13, Computer Science Department, The University of Chicago, 2007. Available at `http://ramakrishnadas.cs.uchicago.edu/gctflip1.ps`.

[207] Ketan D. Mulmuley and Milind Sohoni. Geometric complexity theory I: an approach to the **P** vs. **NP** and related problems. *SIAM J. Comput*, 31(2):496–526, 2001.

[208] David Mumford. *Algebraic geometry I. Complex projective varieties*. Number 221 in Grundlehren der Mathematischen Wissenschaften. Springer-Verlag, Berlin, 1976.

[209] John Nash. Letters to the USA National Security Agency concerning enciphering, 18 January 1955. Available at `http://www.nsa.gov/public_info/_files/nash_letters/nash_letters1.pdf`.

[210] C. Andrew Neff. Specified precision polynomial root isolation is in **NC**. In *FOCS '90: 31st Annual IEEE Symposium on Foundations of Computer Science*, pages 152–162. IEEE Comput. Soc. Press, 1990.

[211] Michael A. Nielson and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.

[212] P. S. Novikov. *Ob algoritmičeskoĭ nerazrešimosti problemy toždestva slov v teorii grupp*. Trudy Mat. Inst. im. Steklov. no. 44. Izdat. Akad. Nauk SSSR, Moscow, 1955. English translation: On the algorithmic insolvability of the word problem in group theory, in: American Mathematical Society Translations, Ser. 2, Vol. 9, AMS, 1958, pp. 1–122.

[213] Peter J. Olver. *Applications of Lie groups to differential equations*, volume 107 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, second edition, 1993.

[214] A. L. Onishchik and È. B. Vinberg. *Lie groups and algebraic groups*. Springer Series in Soviet Mathematics. Springer-Verlag, Berlin, 1990. Translated from the Russian and with a preface by D. A. Leites.

[215] Karen Hunger Parshall. In pursuit of the finite division algebra theorem and beyond: Joseph H. M. Wedderburn, Leonard E. Dickson, and Oswald Veblen. *Arch. Internat. Hist. Sci.*, 33(111):274–299 (1984), 1983.

[216] Erez Petrank and Ron M. Roth. Is code equivalence easy to decide? *IEEE Transactions on Information Theory*, 43:1602–1604, 1997.

[217] Emil L. Post. Recursively enumerable sets of positive integers and their decision problems. *Bull. Amer. Math. Soc.*, 50:284–316, 1944.

[218] Michael O. Rabin. Mathematical theory of automata. In *Proc. Sympos. Appl. Math., Vol. XIX*, pages 153–175. Amer. Math. Soc., 1967.

[219] Michael O. Rabin. Probabilistic algorithm for testing primality. *J. Number Theory*, 12(1):128–138, 1980.

[220] D. Rand, P. Winternitz, and H. Zassenhaus. On the identification of a Lie algebra given by its structure constants. I. Direct decompositions, Levi decompositions, and nilradicals. *Linear Algebra Appl.*, 109:197–246, 1988.

[221] Alexander A. Razborov and Steven Rudich. Natural proofs. *J. Comput. System Sci.*, 55(1, part 1):24–35, 1997.

[222] Oded Regev. Quantum computation and lattice problems. *SIAM J. Comput*, 33(3):738–760, 2004.

[223] Omer Reingold, Luca Trevisan, and Salil Vadhan. Pseudorandom walks on regular digraphs and the **RL** vs. **L** problem. In *STOC '06: 38th Annual ACM Symposium on Theory of Computing*, pages 457–466. ACM, 2006.

[224] R. W. Richardson, Jr. Principal orbit types for algebraic transformation spaces in characteristic zero. *Invent. Math.*, 16:6–14, 1972.

[225] Lajos Rónyai. Zero divisors in quaternion algebras. *J. Algorithms*, 9(4):494–506, 1988.

[226] Lajos Rónyai. Computations in associative algebras. In *Groups and computation (New Brunswick, NJ, 1991)*, volume 11 of *DIMACS Ser. Discrete Math. Theoret. Comput. Sci.*, pages 221–243. AMS, Providence, RI, 1993.

[227] Alexander Russell and Ravi Sundaram. Symmetric alternation captures **BPP**. *Comput. Complexity*, 2(7):152–162, 1995.

[228] Walter L. Ruzzo. On uniform circuit complexity. *J. Comput. System Sci.*, 22(3):365–383, 1981. Special issue dedicated to Michael Machtey.

[229] Alexander J. E. Ryba. Computer construction of split Cartan subalgebras. *J. Algebra*, 309(2):455–483, 2007.

[230] Mark V. Sapir, Jean-Camille Birget, and Eliyahu Rips. Isoperimetric and isodiametric functions of groups. *Ann. of Math. (2)*, 156(2):345–466, 2002.

[231] Walter J. Savitch. Deterministic simulation of non-deterministic Turing machines (detailed abstract). In *STOC '69: 1st Annual ACM Symposium on Theory of Computing*, pages 247–248, 1969.

[232] Arnold Schönhage and Volker Strassen. Schnelle Multiplikation grosser Zahlen. *Computing (Arch. Elektron. Rechnen)*, 7:281–292, 1971.

[233] J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. Assoc. Comput. Mach.*, 27(4):701–717, 1980.

[234] G. B. Seligman. *Modular Lie algebras*. Ergebnisse der Mathematik und ihrer Grenzgebiete, Band 40. Springer-Verlag New York, Inc., New York, 1967.

[235] Alan L. Selman. A survey of one-way functions in complexity theory. *Math. Systems Theory*, 25(3):203–221, 1992.

[236] Alan L. Selman. A taxonomy of complexity classes of functions. *J. Comput. System Sci.*, 48(2):357–381, 1994.

[237] Alan L. Selman, Mei Rui Xu, and Ronald V. Book. Positive relativizations of complexity classes. *SIAM J. Comput*, 12(3):565–579, 1983.

[238] Adi Shamir. **IP** = **PSPACE**. *J. Assoc. Comput. Mach.*, 39(4):869–877, 1992.

[239] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput*, 26(5):1484–1509, 1997.

[240] Daniel R. Simon. On the power of quantum computation. *SIAM J. Comput*, 26(5):1474–1483, October 1997.

[241] Hans-Ulrich Simon. Word problems for groups and contextfree recognition. In *Fundamentals of computation theory (Proc. Conf. Algebraic, Arith. and Categorical Methods in Comput. Theory, Berlin/Wendisch-Rietz, 1979)*, volume 2 of *Math. Res.*, pages 417–422. Akademie-Verlag, Berlin, 1979.

[242] Charles C. Sims. Computational methods in the study of permutation groups. In *Computational Problems in Abstract Algebra (Oxford, 1967)*, pages 169–183. Pergamon, Oxford, 1970.

[243] Charles C. Sims. Computation with permutation groups. In *SYMSAC '71: Proceedings of the Second ACM Symposium on Symbolic and Algebraic Manipulation*, pages 23–28. ACM, 1971.

[244] Michael Sipser. On relativization and the existence of complete sets. In *ICALP '82: International Colloquium on Automata, Languages and Programming*, volume 140 of *Lecture Notes in Computer Science*, pages 523–531. Springer, Berlin, 1982.

[245] Michael Sipser. A complexity theoretic approach to randomness. In *STOC '83: 15th Annual ACM Symposium on Theory of Computing*, pages 330–335. ACM, 1983.

[246] Michael Sipser. The history and status of the **P** versus **NP** question. In *STOC '92: 24th Annual ACM Symposium on Theory of Computing*, pages 603–618. ACM, 1992.

[247] Michael Sipser. *Introduction to the Theory of Computation*. Course Technology, second edition, 2005.

[248] Steve Smale. On the topology of algorithms. I. *J. Complexity*, 3(2):81–89, 1987.

[249] Ronald Solomon. A brief history of the classification of the finite simple groups. *Bull. Amer. Math. Soc. (N.S.)*, 38(3):315–352, 2001.

[250] Robert M. Solovay. A model of set-theory in which every set of reals is Lebesgue measurable. *Ann. of Math. (2)*, 92:1–56, 1970.

[251] Robert M. Solovay and Volker Strassen. A fast Monte-Carlo test for primality. *SIAM J. Comput*, 6(1):84–85, 1977.

[252] Willi-Hans Steeb. *Continuous symmetries, Lie algebras, differential equations and computer algebra*. World Scientific Publishing Co. Pte. Ltd., Hackensack, NJ, second edition, 2007.

[253] J. Michael Steele and Andrew C. Yao. Lower bounds for algebraic decision trees. *J. Algorithms*, 3(1):1–8, 1982.

[254] Larry J. Stockmeyer. The polynomial-time hierarchy. *Theoret. Comput. Sci.*, 3(1):1–22 (1977), 1976.

[255] Larry J. Stockmeyer and Albert R. Meyer. Word problems requiring exponential time (preliminary report). In *STOC '73: 5th Annual ACM Symposium on Theory of Computing*, pages 1–9. ACM, 1973.

[256] Volker Strassen. Vermeidung von Divisionen. *J. Reine Angew. Math.*, 264:184–202, 1973.

[257] Madhu Sudan. Invariance in property testing. In *Property Testing*, number 6390 in Lecture Notes in Computer Science, pages 211–227. Springer, 2011. Also available as ECCC Tech. Report TR10-051.

[258] Róbert Szelepcsényi. The method of forced enumeration for nondeterministic automata. *Acta Inf.*, 26(3):279–284, 1988.

[259] Jun Tarui. Randomized polynomials, threshold circuits, and the polynomial hierarchy. In *STACS '91: 8th Annual Symposium on Theoretical Aspects of Computer Science*, pages 238–250. Springer-Verlag, 1991.

[260] Thomas Thierauf. *The Computational Complexity of Equivalence and Isomorphism Problems*, volume 1852 of *Lecture Notes in Computer Science*. Springer, New York, 2000.

[261] Seinosuke Toda. On the computational power of **PP** and #**P**. In *FOCS '89: 30th Annual IEEE Symposium on Foundations of Computer Science*, pages 514–519, 1989.

[262] Seinosuke Toda. Counting problems computationally equivalent to the determinant, 1991. Manuscript.

[263] Seinosuke Toda and Mitsunori Ogiwara. Counting classes are at least as hard as the polynomial-time hierarchy. *SIAM J. Comput*, 21(2):316–328, 1992.

[264] Barry M. Trager. Algebraic factoring and rational function integration. In *SYMSAC '76: 3rd ACM Symposium on Symbolic and Algebraic Computation*, pages 219–226. ACM, 1976.

[265] Boris A. Trakhtenbrot. A survey of Russian approaches to perebor (brute-force searches) algorithms. *Annals of the History of Computing*, 6(4):384–400, 1984.

[266] Alan M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proc. of the London Math. Soc.*, s2-42(1):230–265, 1937.

[267] L. G. Valiant, S. Skyum, S. Berkowitz, and C. Rackoff. Fast parallel computation of polynomials using few processors. *SIAM J. Comput*, 12(4):641–644, 1983.

[268] Lesile G. Valiant. Negation can be exponentially powerful. *Theoret. Comput. Sci.*, 12(3):303–314, 1980.

[269] Leslie G. Valiant. Completeness classes in algebra. In *STOC '79: 11th Annual ACM Symposium on Theory of Computing*, pages 249–261. ACM, 1979.

[270] Leslie G. Valiant and Vijay V. Vazirani. **NP** is as easy as detecting unique solutions. *Theoret. Comput. Sci.*, 47(1):85–93, 1986.

[271] Bartel L. van der Waerden. Eine Bemerkung über die Unzerlegbarkeit von Polynomen. *Math. Ann.*, 102(1):738–739, 1930.

[272] H. Venkateswaran. Circuit definitions of nondeterministic complexity classes. *SIAM J. Comput*, 21(4):655–670, 1992.

[273] V. Vinay. Counting auxiliary pushdown automata and semi-unbounded arithmetic circuits. In *Proceedings of the 6th Structures in Complexity Theory Conference*, Lecture Notes in Computer Science, pages 270–284. Springer, 1991.

[274] Joachim von zur Gathen. Permanent and determinant. *Linear Algebra Appl.*, 96:87–100, 1987.

[275] Hermann Weyl. *Symmetry*. Princeton Science Library. Princeton University Press, Princeton, NJ, 1989. Reprint of the 1952 original.

[276] James B. Wilson. Decomposing p-groups via Jordan algebras. *J. Algebra*, 322:2642–2679, 2009.

[277] James B. Wilson. Finding central decompositions of p-groups. *J. Group Theory*, 12:813–830, 2009.

[278] Hans Wussing. *The genesis of the abstract group concept*. MIT Press, Cambridge, MA, 1984. A contribution to the history of the origin of abstract group theory, Translated from the German by Abe Shenitzer and Hardy Grant.

[279] Stathis Zachos. Probabilistic quantifiers and games. *J. Comput. System Sci.*, 36(3):433–451, 1988. Structure in Complexity Theory Conference 1986.

[280] Richard Zippel. Probabilistic algorithms for sparse polynomials. In *Symbolic and algebraic computation (EUROSAM '79, Internat. Sympos., Marseille, 1979)*, volume 72 of *Lecture Notes in Computer Science*, pages 216–226. Springer, Berlin, 1979.