Contents

Chapter 0

Introduction

In the beginning, there were two measures of computational complexity: time and space. From an engineering standpoint, these were very natural measures, quantifying the amount of physical resources needed to perform a computation. From a mathematical viewpoint, time and space were somewhat less satisfying, since neither appeared to be tied to the inherent mathematical complexity of the computational problem.

In 1974, Ron Fagin changed this. He showed that the complexity class NP — those problems computable in nondeterministic polynomial time — is exactly the set of problems describable in second-order existential logic. This was a remarkable insight, for it demonstrated that the computational complexity of a problem can be understood as the richness of a language needed to specify the problem. Time and space are not model-dependent engineering concepts, they are more fundamental.

Although few programmers consider their work in this way, a computer program is a completely precise description of a mapping from inputs to outputs. In this book we follow database terminology and call such a map a *query* from input structures to output structures. Typically a program describes a precise sequence of steps that compute a given query. However, we may choose to describe the query in some other precise way. For example, we may describe queries in variants of first- and second-order mathematical logic.

Fagin's Theorem gave the first such connection. Using first-order languages, this approach, commonly called descriptive complexity, demonstrated that virtually all measures of complexity can be mirrored in logic. Furthermore, as we will see, the most important classes have especially elegant and clean descriptive characterizations.

Descriptive complexity provided the insight behind a proof of the Immerman-Szelepcsényi Theorem, which states that nondeterministic space classes are closed under complementation. This settled a question that had been open for twenty-five years; indeed, almost everyone had conjectured the negation of this theorem.

Descriptive complexity has long had applications to database theory. A relational database is a finite logical structure, and commonly used query languages are small extensions of first-order logic. Thus, descriptive complexity provides a natural foundation for database theory, and many questions concerning the expressibility of query languages and the efficiency of their evaluation have been settled using the methods of descriptive complexity. Another prime application area of descriptive complexity is to the problems of Computer Aided Verification.

Since the inception of complexity theory, a fundamental question that has bedeviled theorists is the P versus NP question. Despite almost three decades of work, the problem of proving P different from NP remains. As we will see, P versus NP is just a famous and dramatic example of the many open problems that remain. Our inability to ascertain relationships between complexity classes is pervasive. We can prove that more of a given resource, e.g., time, space, nondeterministic time, etc., allows us to compute strictly more queries. However, the relationship between different resources remains virtually unknown.

We believe that descriptive complexity will be useful in these and many related problems of computational complexity. Descriptive complexity is a rich edifice from which to attack the tantalizing problems of complexity. It gives a mathematical structure with which to view and set to work on what had previously been engineering questions. It establishes a strong connection between mathematics and computer science, thus enabling researchers of both backgrounds to use their various skills to set upon the open questions. It has already led to significant successes.

The Case for Finite Models

A fundamental philosophical decision taken by the practitioners of descriptive complexity is that computation is inherently finite. The relevant objects — inputs, databases, programs, specifications — are all finite objects that can be conveniently modeled as finite logical structures. Most mathematical theories study infinite objects. These are considered more relevant, general, and important to the typical mathematician. Furthermore, infinite objects are often simpler and better behaved than their finite cousins. A typical example is the set of natural numbers, $\mathbf{N} = \{0, 1, 2, \ldots\}$. Clearly this has a simpler and more elegant theory than the set of natural numbers representable in 64-bit computer words. However, there is a significant danger in taking the infinite approach. Namely, the models are often wrong! Properties that we can prove about **N** are often false or irrelevant if we try to apply them to the objects that computers have and hold. We find that the subject of finite models is quite different in many respects. Different theorems hold and different techniques apply.

Living in the world of finite structures may seem odd at first. Descriptive complexity requires a new way of thinking for those readers who have been brought up on infinite fare. Finite model theory is different and more combinatorial than general model theory. In Descriptive complexity, we use finite model theory to understand computation. We expect that the reader, after some initial effort and doubt, will agree that the theory of computation that we develop has significant advantages. We believe that it is more accurate and more relevant in the study of computation.

I hope the reader has as much pleasure in discovering and using the tools of Descriptive complexity as I have had. I look forward to new contributions in the modeling and understanding of computation to be made by some of the readers of this book.