

Contents

0	Introduction	1
1	Background in Logic	5
1.1	Introduction and Preliminary Definitions	5
1.2	Ordering and Arithmetic	15
1.2.1	FO(BIT) = FO(PLUS, TIMES)	17
1.3	Isomorphism	20
1.4	First-Order Queries	21
2	Background in Complexity	29
2.1	Introduction	29
2.2	Preliminary Definitions	30
2.3	Reductions and Complete Problems	34
2.4	Alternation	43
2.5	Simultaneous Resource Classes	51
2.6	Summary	52
3	First-Order Reductions	57
3.1	$\text{FO} \subseteq \text{L}$	57
3.2	Dual of a First-Order Query	59
3.3	Complete problems for L and NL	64
3.4	Complete Problems for P	67

4	Inductive Definitions	73
4.1	Least Fixed Point	73
4.2	The Depth of Inductive Definitions	79
4.3	Iterating First-Order Formulas	81
5	Parallelism	89
5.1	Concurrent Random Access Machines	90
5.2	Inductive Depth Equals Parallel Time	93
5.3	Number of Variables versus Number of Processors	98
5.4	Circuit Complexity	102
5.5	Alternating Complexity	112
5.5.1	Alternation as Parallelism	114
6	Ehrenfeucht-Fraïssé Games	119
6.1	Definition of the Games	119
6.2	Methodology for First-Order Expressibility	129
6.3	First-Order Properties are Local	134
6.4	Bounded Variable Languages	135
6.5	Zero-One Laws	140
6.6	Ehrenfeucht-Fraïssé Games with Ordering	143
7	Second-Order Logic and Fagin's Theorem	147
7.1	Second-Order Logic	147
7.2	Proof of Fagin's Theorem	150
7.3	NP-Complete Problems	154
7.4	The Polynomial-Time Hierarchy	157
8	Second-Order Lower Bounds	161
8.1	Second-Order Games	161
8.2	$SO\exists$ (monadic) Lower Bound on Reachability	167
8.3	Lower Bounds Including Ordering	172

9	Complementation and Transitive Closure	177
9.1	Normal Form Theorem for FO(LFP)	177
9.2	Transitive Closure Operators	182
9.3	Normal Form for FO(TC)	184
9.4	Logspace is Primitive Recursive	189
9.5	$\text{NSPACE}[s(n)] = \text{co-NSPACE}[s(n)]$	191
9.6	Restrictions of SO	194
10	Polynomial Space	199
10.1	Complete Problems for PSPACE	199
10.2	Partial Fixed Points	203
10.3	$\text{DSPACE}[n^k] = \text{VAR}[k + 1]$	206
10.4	Using Second-Order Logic to Capture PSPACE	210
11	Uniformity and Precomputation	215
11.1	An Unbounded Number of Variables	216
11.1.1	Tradeoffs Between Variables and Quantifier Depth	217
11.2	First-Order Projections	218
11.3	Help Bits	224
11.4	Generalized Quantifiers	225
12	The Role of Ordering	229
12.1	Using Logic to Characterize Graphs	230
12.2	Characterizing Graphs Using \mathcal{L}^k	232
12.3	Adding Counting to First-Order Logic	234
12.4	Pebble Games for \mathcal{C}^k	237
12.5	Vertex Refinement Corresponds to \mathcal{C}^2	239
12.6	Abiteboul-Vianu and Otto Theorems	243
12.7	Toward a Language for Order-Independent P	252

13 Lower Bounds	257
13.1 Håstad's Switching Lemma	257
13.4 A Lower Bound for REACH_a	281
13.5 Lower Bound for Fixed Point and Counting	288
14 Applications	281
14.1 Databases	281
14.1.1 SQL	282
14.1.2 Datalog	285
14.2 Dynamic Complexity	287
14.2.1 Dynamic Complexity Classes	289
14.3 Model Checking	298
14.3.1 Temporal Logic	299
14.4 Summary	305
15 Conclusions and Future Directions	307
15.1 Languages That Capture Complexity Classes	307
15.1.1 Complexity on the Face of a Query	310
15.1.2 Stepwise Refinement	310
15.2 Why Is <i>Finite</i> Model Theory Appropriate?	311
15.3 Deep Mathematical Problems: P versus NP	312
15.4 Toward Proving Lower Bounds	313
15.4.1 Role of Ordering	314
15.4.2 Approximation and Approximability	314
15.5 Applications of Descriptive Complexity	315
15.5.1 Dynamic Complexity	315
15.5.2 Model Checking	316
15.5.3 Abstract State Machines	316
15.6 Software Crisis and Opportunity	317
15.6.1 How can Finite Model Theory Help?	318

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, $\mathbf{N} = \{0, 1, 2, \dots\}$. 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 \mathbf{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.

