Descriptive Complexity

Neil Immerman

February 25, 2015

first printing: Jan., 1999, Springer Graduate Texts in Computer Science

This book is dedicated to Daniel and Ellie.

Preface

This book should be of interest to anyone who would like to understand computation from the point of view of logic. The book is designed for graduate students or advanced undergraduates in computer science or mathematics and is suitable as a textbook or for self study in the area of descriptive complexity. It is of particular interest to students of computational complexity, database theory, and computer aided verification. Numerous examples and exercises are included in the text, as well as a section at the end of each chapter with references and suggestions for further reading.

The book provides plenty of material for a one semester course. The core of the book is contained in Chapters 1 through 7, although even here some sections can be omitted according to the taste and interests of the instructor. The remaining chapters are more independent of each other. I would strongly recommend including at least parts of Chapters 9, 10, and 12. Chapters 8 and 13 on lower bounds include some of the nicest combinatorial arguments. Chapter 11 includes a wealth of information on uniformity; to me, the low-level nature of translations between problems that suffice to maintain completeness is amazing and provides powerful descriptive tools for understanding complexity. I assume that most readers will want to study the applications of descriptive complexity that are introduced in Chapter 14.

Map of the Book

Chapters 1 and 2 provide introductions to logic and complexity theory, respectively. These introductions are fast-moving and specialized. (Alternative sources are suggested at the end of these chapters for students who would prefer more background.) This background material is presented with an eye toward the descriptive point of view. In particular, Chapter 1 introduces the notion of queries. In Chapter 2, all complexity classes are defined as sets of boolean queries. (A boolean query is a query whose answer is a single bit: yes or no. Since traditional complexity classes are defined as sets of yes/no questions, they are exactly sets of boolean queries.)

Chapter 3 begins the study of the relationship between descriptive and computational complexity. All first-order queries are shown to be computable in the low complexity class deterministic logspace (L). Next, the notion of *first-order reduction* — a first-order expressible translation from one problem to another — is introduced. Problems complete via first-order reductions for the complexity classes L, nondeterministic logspace (NL), and P are presented.

Chapter 4 introduces the least-fixed-point operator, which formalizes the power of making inductive definitions. P is proved equal to the set of boolean queries expressible in first-order logic plus the power to define new relations by induction. It is striking that such a significant descriptive class is equal to P. We thus have a natural, machine-independent view of feasible computation. This means we can understand the P versus NP question entirely from a logical point of view: P is equal to NP iff every second-order expressible query is already expressible in first-order logic plus inductive definitions (Corollary 7.23).

Chapter 5 introduces the notion of parallel computation and ties it to descriptive complexity. In parallel computation, we can take advantage of many different processors or computers working simultaneously. The notion of quantification is inherently parallel. I show that the parallel time needed to compute a query corresponds exactly to its quantifier depth. The number of distinct variables occurring in a first-order inductive query corresponds closely with the amount of hardware — processors and memory — needed to compute this query. The most important tradeoff in complexity theory — between parallel time and hardware — is thus identical to the tradeoff between inductive depth and number of variables.

Chapter 6 introduces a combinatorial game that serves as an important tool for ascertaining what can and cannot be expressed in logical languages. Ehrenfeucht-Fraïssé games offer a semantics for first-order logic that is equivalent to, but more directly applicable than, the standard definitions. These games provide powerful tools for descriptive complexity. Using them, we can often decide whether a given query is or is not expressible in a given language.

Chapter 7 introduces second-order logic. This is much more expressive than first-order logic because we may quantify over an exponentially larger space of objects. I prove Fagin's theorem as well as Stockmeyer's characterization of the polynomial-time hierarchy as the set of second-order describable boolean queries. It follows from previous results that the polynomial-time hierarchy is the set of boolean queries computable in constant time, but using exponentially much hardware (Corollary 7.28). This insight exposes the strange character of the polynomial-time hierarchy and of the class NP.

Chapter 8 uses Ehrenfeucht-Fraïssé games to prove that certain queries are not expressible in some restrictions of second-order logic. Since second-order logic is so expressive, it is surprising that we can prove results about non-expressibility. However, the restrictions needed on the second-order languages — in particular, that they quantify only monadic relations — are crucial.

Chapter 9 studies the transitive-closure operator, a restriction of the least-fixed-point operator. I show that transitive closure characterizes the power of the class NL. When infinite structures are allowed, both the least-fixed-point and transitive-closure operators are not closed under negation. In this case there is a strict expressive hierarchy as we alternate applications of these operators with negation. However, for finite structures I show that these operators are closed under negation. A corollary is that nondeterministic space classes are closed under complementation. This was a very unexpected result when it was proved. It constitutes a significant contribution of descriptive complexity to computer science.

Chapter 10 studies the complexity class polynomial space, PSPACE, which is the set of all boolean queries that can be computed using a polynomial amount of hardware, but with no restriction on time. Thus PSPACE is beyond the realm of what is feasibly computable. It is obvious that NP is contained in PSPACE, but it is not known whether PSPACE is larger than NP. Indeed, it is not even known that PSPACE is larger than P. PSPACE is a very robust complexity class. It has several interesting descriptive characterizations, which expose more information about the tradeoff between inductive depth and number of variables.

Chapter 11 studies precomputation — the work that may go into designing the program, formula, or circuit before any input is seen. Precomputation — even less well understood than time and hardware — has an especially crisp formulation in descriptive complexity.

In order for a structure such as a graph to be input to a real or idealized machine, it must be encoded as a character string. Such an encoding imposes an ordering on the universe of the structure, e.g., on the vertices of the graph. All first-order, descriptive characterizations of complexity classes assume that a total ordering relation on the universe is available in the languages. Without such an ordering, simple lower bounds from Chapter 6 show that certain trivial properties — such as computing the PARITY of the cardinality of the universe — are not expressible. However, the ordering relation allows us to distinguish isomorphic structures which all plausible queries should treat the same. In addition, an ordering relation spoils the power of Ehrenfeucht-Fraïssé games for most languages.

The mathematically rich search for a suitable alternative to ordering is described in Chapter 12.

Chapter 13 describes some interesting combinatorial arguments that provide lower bounds on descriptive complexity. The first is the optimal lower bound due to Håstad on the quantifier depth needed to express PARITY. One of many corollaries is that the set of first-order boolean queries is a strict subset of L. The second two lower bounds are weaker: they use Ehrenfeucht-Fraïssé games without ordering and thus, while quite interesting, do not separate complexity classes.

Chapter 14 describes applications of descriptive complexity to databases and computer-aided verification. Relational databases are exactly finite logical structures, and commercial query languages such as SQL are simple extensions of first-order logic. The complexity of query evaluation, the expressive power of query languages, and the optimization of queries are all important practical issues here, and the tools that have been developed previously can be brought to bear on these issues.

Model checking is a burgeoning subfield of computer-aided verification. The idea is that the design of a circuit, protocol, or program can be automatically translated into a transition system, i.e., a graph whose vertices represent global states and whose edges represent possible atomic transitions. Model checking means deciding whether such a design satisfies a simple correctness condition such as, “Doors are not opened between stations”, or, “Division is always performed correctly”. In descriptive complexity, we can see on the face of such a query what the complexity of checking it will be.

Finally, Chapter 15 sketchs some directions for future research in and applications of descriptive complexity.

Acknowledgements:

I have been intending to write this book for more years than I would like to admit. In the mean time, many researchers have changed and extended the field so quickly that it is not possible for me to really keep up. I have tried to give pointers to some of the many topics not covered.

I am grateful to everyone who has found errors or made suggestions or encouraged me to write this book. All the errors remaining are mine alone. There will be a page on the world wide web with corrections, recent developments, etc., concerning this book and descriptive complexity in general. Just search for ”Neil Immerman” on the web and you will find it. All comments, corrections, etc., will be greatly appreciated.

Some of the people who have already provided help and helpful comments are: Natasha Alechina, Jose Balcázar, Dave Mix Barrington, Jonathan Buss, Russ Ellsworth, Miklos Erdelyi-Szabo, Ron Fagin, Erich Grädel, Jens Gramm, Martin Grohe, Brian Hanechak, Lauri Hella, Janos Makowsky, Yiannis Moschovakis, John Ridgway, Jose Antonio Medina, Gleb Naumovich, Sushant Patnaik, Nate Segerlind, Richard Shore, Wolfgang Thomas, and especially Kousha Etessami.

I am grateful to David Gries for taking his role as an editor of this series so seriously that he read this book in detail, making numerous helpful comments and corrections.

Thanks to the following institutions for financial support during the long process of writing this book: NSF Grant CCR-9505446, Cornell University Computer Science Department, and the DIMACS special year in logic and algorithms.

I want to acknowledge my debt to the many inspiring teachers that I have had over the years. I have a vivid memory of Larry Carter on crutches because he had broken a blood vessel in his leg during the previous afternoon's soccer game hopping back and forth in front of the room as he built a Turing machine to multiply two numbers. This was at an NSF sponsored summer program at the University of New Hampshire in 1969. David Kelly was the codirector of that program and has been making magic ever since, creating summer programs where mathematics as a creative and cooperative endeavor is taught and shared. These programs more than anything else taught me the value and pleasure of teaching and research. Larry later roomed with Ron Fagin as a graduate student at Berkeley, and it was because of this connection that I learned of Ron's research connecting logic and complexity.

A few other of my teachers that I would like to thank by name are Shizuo Kakutani, Angus Macintyre, John Hopcroft, and Juris Hartmanis.

I thank my wife, Susan Landau, who, in part to let me pursue my own obscure interests, has given up more than anyone should have to. I am delighted that you finished your book [DL98] first and to such acclaim. Thanks for your love, for your unswerving integrity, and for your amazing ability to keep moving forward.

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.

Chapter 1

Background in Logic

Mathematics enables us to model many things abstractly. Group theory, for example, abstracts features of such diverse activities as English change ringing and quantum mechanics. Mathematical logic carries the abstraction one level higher: it is a mathematical model of mathematics. This book shows that the computational complexity of all problems in computer science can be understood via the complexity of their logical descriptions. We begin with a high-level introduction to logic. Although much of the material is well-known, we urge readers to at least skim this background chapter as the concentration on finite and ordered structures, i.e., relational databases, is not standard in most treatments of logic.

1.1 Introduction and Preliminary Definitions

All logic books begin with definitions. We have to introduce the language before we start to speak. Thus, a *vocabulary*

$$\tau = \langle R_1^{a_1}, \ldots, R_r^{a_r}, c_1, \ldots, c_s, f_1^{r_1}, \ldots, f_t^{r_t} \rangle$$

is a tuple of relation symbols, constant symbols, and function symbols. R_i is a relation symbol of arity a_i and f_j is a function symbol of arity r_j. Two important examples are $\tau_g = \langle E^2, s, t \rangle$, the vocabulary of graphs with specified source and terminal nodes, and $\tau_s = \langle \leq^2, S^1 \rangle$, the vocabulary of binary strings.

A *structure* with vocabulary τ is a tuple,

$$\mathcal{A} \quad = \quad \langle |\mathcal{A}|, R_1^{\mathcal{A}}, \ldots, R_r^{\mathcal{A}}, c_1^{\mathcal{A}}, \ldots, c_s^{\mathcal{A}}, f_1^{\mathcal{A}}, \ldots, f_t^{\mathcal{A}} \rangle$$

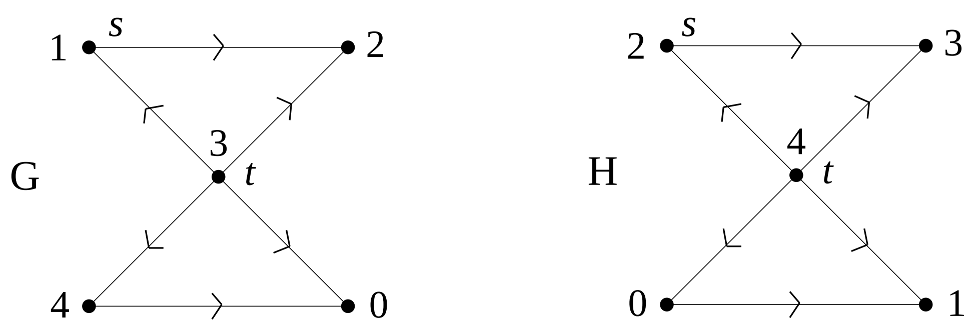

Figure 1.1: Graphs G and H

whose universe is the nonempty set $|\mathcal{A}|$. For each relation symbol R_i of arity a_i in τ, $\mathcal{A}$ has a relation $R_i^{\mathcal{A}}$ of arity a_i defined on $|\mathcal{A}|$, i.e., $R_i^{\mathcal{A}} \subseteq |\mathcal{A}|^{a_i}$. For each constant symbol $c_j \in \tau$, $\mathcal{A}$ has a specified element of its universe $c_j^{\mathcal{A}} \in |\mathcal{A}|$. For each function symbol $f_i \in \tau$, $f_i^{\mathcal{A}}$ is a total function from $|\mathcal{A}|^{r_i}$ to $|\mathcal{A}|$. A vocabulary without function symbols is called a *relational vocabulary*. In this book, unless stated otherwise, all vocabularies are relational. The notation $\|\mathcal{A}\|$ denotes the cardinality of the universe of $\mathcal{A}$.

In the history of mathematical logic most interest has concentrated on infinite structures. Indeed, many mathematicians consider the study of finite structures trivial. Yet, the objects computers have and hold are always finite. To study computation we need a theory of finite structures.

Logic restricted to finite structures is rather different from the theory of infinite structures. We mention infinite structures from time to time, most often when we comment on whether a given theorem also holds in the infinite case. However, we concentrate on finite structures. We define STRUC$[\tau]$ to be the set of finite structures of vocabulary τ.

As an example, the graph $G = \langle V^G, E^G, 1, 3\rangle$ defined by,

$$V^G \;=\; \{0,1,2,3,4\}, \qquad E^G \;=\; \{(1,2),(3,0),(3,1),(3,2),(3,4),(4,0)\}$$

is a structure of vocabulary τ_g consisting of a directed graph with two specified vertices s and t. G has five vertices and six edges. (See Figure 1.1, which shows G as well as another graph H which is isomorphic but not equal to G.)

For another example, consider the binary string $w =$ “01101”. We can code w as the structure $\mathcal{A}_w = \langle\{0,1,\ldots,4\}, \leq, \{1,2,4\}\rangle$ of vocabulary τ_s. Here $\leq$ represents the usual ordering on $0,1,\ldots,4$. Relation $S^w = \{1,2,4\}$ represents the positions where w is one. (Relation symbols of arity one, such as S^w, are sometimes called *monadic*.)

A relational database is exactly a finite relational structure. The following begins a running example of a genealogical database.

Example 1.2 Consider a genealogical database $\mathcal{B}_0 = \langle U_0, F_0, P_0, S_0\rangle$; where U_0 is a finite set of people,

$$U_0 = \{\text{Abraham, Isaac, Rebekah, Sarah, } \ldots\}$$

F_0 is a monadic relation that is true of the female elements of U_0,

$$F_0 = \{\text{Sarah, Rebekah, } \ldots\}$$

P_0 and S_0 are the binary relations for parent and spouse, respectively, e.g.,

$$\begin{array}{rcl} P_0 & = & \{\langle \text{Abraham,Isaac}\rangle, \langle\text{Sarah,Isaac}\rangle, \ldots\} \\ S_0 & = & \{\langle \text{Abraham,Sarah}\rangle, \langle\text{Isaac,Rebekah}\rangle, \ldots\} \end{array}$$

Thus, $\mathcal{B}_0$ is a structure of vocabulary $\langle F^1, P^2, S^2\rangle$. $\square$

For any vocabulary τ, define the *first-order language* $\mathcal{L}(\tau)$ to be the set of formulas built up from the relation and constant symbols of τ; the logical relation symbol =; the boolean connectives $\wedge, \neg$; variables: VAR $= \{x, y, z, \ldots\}$; and quantifier $\exists$.

We say that an occurrence of a variable v in φ is *bound* if it lies within the scope of a quantifier $(\exists v)$ or $(\forall v)$, otherwise it is *free*. Variable v is *free* in φ iff it has a free occurrence in φ. For example, the free variables in the following formula are x and y. We use the symbol "$\equiv$" to define or denote equivalence of formulas.

$$\alpha \quad \equiv \quad [(\exists y)(y+1=x)] \ \wedge \ x<y$$

In a similar way we sometimes use "$\Leftrightarrow$" to indicate that two previously defined formulas or conditions are equivalent.

Bound variables are "dummy" variables and may be renamed to avoid confusion. For example, α is equivalent to the following α' which also has free variables x and y,

$$\alpha' \quad \equiv \quad [(\exists z)(z+1=x)] \ \wedge \ x<y$$

We write $\mathcal{A} \models \varphi$ to mean that $\mathcal{A}$ satisfies φ, i.e., that φ is true in $\mathcal{A}$. Since φ may contain some free variables, we will let an *interpretation* into $\mathcal{A}$ be a map $i : V \to |\mathcal{A}|$ where V is some finite subset of VAR. For convenience, for every constant symbol $c \in \tau$ and any interpretation i for $\mathcal{A}$, we let $i(c) = c^{\mathcal{A}}$. If τ has function symbols, then the definition of i extends to all terms via the recurrence,

$$i(f_j(t_1, \ldots, t_{r_j})) \quad = \quad f_j^{\mathcal{A}}(i(t_1), \ldots, i(t_{r_j})) \; .$$

We can be completely precise about the semantics of mathematical logic. In particular, we can definitively define what it means for a sentence φ to be true in a structure $\mathcal{A}$.

Definition 1.3 (Definition of Truth) Let $\mathcal{A} \in \text{STRUC}[\tau]$ be a structure, and let i be an interpretation into $\mathcal{A}$ whose domain includes all the relevant free variables. We inductively define whether a formula $\varphi \in \mathcal{L}(\tau)$ is true in $(\mathcal{A}, i)$:

$$\begin{array}{rcl}
(\mathcal{A}, i) \models t_1 = t_2 & \Leftrightarrow & i(t_1) = i(t_2) \\
(\mathcal{A}, i) \models R_j(t_1, \ldots, t_{a_j}) & \Leftrightarrow & \langle i(t_1), \ldots, i(t_{a_j}) \rangle \in R_j^{\mathcal{A}} \\
(\mathcal{A}, i) \models \neg\varphi & \Leftrightarrow & \text{it is not the case that } (\mathcal{A}, i) \models \varphi \\
(\mathcal{A}, i) \models \varphi \wedge \psi & \Leftrightarrow & (\mathcal{A}, i) \models \varphi \text{ and } (\mathcal{A}, i) \models \psi \\
(\mathcal{A}, i) \models (\exists x)\varphi & \Leftrightarrow & (\text{there exists } a \in |\mathcal{A}|)(\mathcal{A}, i, a/x) \models \varphi \\
 & \text{where} & (i, a/x)(y) \; = \; \begin{cases} i(y) & \text{if } y \neq x \\ a & \text{if } y = x \end{cases}
\end{array}$$

Write $\mathcal{A} \models \varphi$ to mean that $(\mathcal{A}, \emptyset) \models \varphi$. □

Definition 1.3 is our first example of an inductive definition, a device that is often used by logicians. It deserves a few comments. Note that the equality symbol $(=)$ is not treated as an ordinary binary relation symbol — the definition insists that this symbol be interpreted as equality. Many students, on first seeing this definition, feel that it is circular. It is not. We are defining the meaning of the symbol "$=$" in terms of the intuitively well-understood standard equality. In the same way, we define the meaning of "$\neg$", "$\wedge$", and "$\exists$" in terms of their intuitive counterparts.

We define the "for all" quantifier as the dual of $\exists$ and the boolean "or" as the dual of $\wedge$,

$$(\forall x)\varphi \quad \equiv \quad \neg(\exists x)\neg\varphi; \qquad \alpha \vee \beta \quad \equiv \quad \neg(\neg\alpha \;\wedge\; \neg\beta)$$

It is convenient to introduce other abbreviations into our formulas. For example, "$y \neq z$" is an abbreviation for "$\neg y = z$". Similarly "$\alpha \rightarrow \beta$" is an abbreviation for "$\neg\alpha \vee \beta$", and "$\alpha \leftrightarrow \beta$" is an abbreviation for "$\alpha \rightarrow \beta \wedge \beta \rightarrow \alpha$". In some sense, the symbols we introduce formally into our language are part of our low-level "machine language", and abbreviations are analogous to what computer scientists call macros. Abbreviations are directly translatable into the real language, and they make formulas more readable. Without abbreviations and the breaking of formulas into modular descriptions, it would be impossible to communicate complicated ideas in first-order logic.

We use spacing and parentheses to make the order of operations clear. Our convention for operator precedence is that "$\neg$", "$\forall$", and "$\exists$" have highest precedence, then "$\wedge$" and "$\vee$", and finally, "$\rightarrow$" and "$\leftrightarrow$". The operations "$\wedge$" and "$\vee$" are evaluated left to right, but "$\rightarrow$" and "$\leftrightarrow$" are evaluated right to left . For example, the following two formulas are equivalent,

$$\neg R(a) \rightarrow R(b) \wedge R(c) \vee R(d) \leftrightarrow R(e)$$

$$(\neg R(a)) \rightarrow (((R(b) \wedge R(c)) \vee R(d)) \leftrightarrow R(e))$$

A *sentence* is a formula with no free variables. Every sentence $\varphi \in \mathcal{L}(\tau)$ is either true or false in any structure $\mathcal{A} \in \mathrm{STRUC}[\tau]$.

Example 1.4 We give a few examples of first-order formulas in the language of graphs:

$$\varphi_{undir} \;\equiv\; (\forall x)(\forall y)(\neg E(x,x) \;\wedge\; (E(x,y) \rightarrow E(y,x)))$$

Formula φ_{undir} says that the graph in question is undirected and has no loops.

$$\begin{aligned} \varphi_{out2} \;&\equiv\; (\forall x)(\exists yz)(y \neq z \wedge E(x,y) \wedge E(x,z) \wedge \\ &\qquad\quad (\forall w)(E(x,w) \rightarrow (w = y \vee w = z))) \\ \varphi_{deg2} \;&\equiv\; \varphi_{undir} \;\wedge\; \varphi_{out2} \end{aligned}$$

Formula φ_{out2} says that every vertex has exactly two edges leaving it. Thus, φ_{deg2} says that the graph in question is undirected, has no loops, and is regular of degree two, i.e., every vertex has exactly two neighbors.

$$\begin{aligned}
\varphi_{dist1} &\equiv x = y \ \vee \ E(x,y) \\
\varphi_{dist2} &\equiv (\exists z)(\varphi_{dist1}(x,z) \ \wedge \ \varphi_{dist1}(z,y)) \\
\varphi_{dist4} &\equiv (\exists z)(\varphi_{dist2}(x,z) \ \wedge \ \varphi_{dist2}(z,y)) \\
\varphi_{dist8} &\equiv (\exists z)(\varphi_{dist4}(x,z) \ \wedge \ \varphi_{dist4}(z,y))
\end{aligned}$$

Formulas φ_{dist1}, φ_{dist2}, and so on say that there is a path from x to y of length at most 1, 2, 4, and 8, respectively. Note that these formulas have free variables x and y.

Formulas express properties about their free variables. For example, for a pair of vertices a, b from the universe of a graph G, the meaning of

$$(G, a/x, b/y) \ \models \ \varphi_{dist8}$$

is that the distance from a to b in G is at most 8.

Sometimes we will make the free variables in a formula explicit, e.g., writing $\varphi_{dist8}(x,y)$ instead of just φ_{dist8}. This offers the advantage of making substitutions more readable: we can write $\varphi_{dist8}(a,b)$ instead of $\varphi_{dist8}(a/x, b/y)$. □

Exercise 1.5 For $n \in \mathbf{N}$, consider the logical structures

$$\mathcal{A}_n \quad = \quad \langle \{0,1,\ldots,n-1\}, \mathrm{PLUS}^{\mathcal{A}_n}, \mathrm{TIMES}^{\mathcal{A}_n}, 0, 1, n-1 \rangle$$

of vocabulary $\tau_a = \langle \mathrm{PLUS}^3, \mathrm{TIMES}^3, 0, 1, max \rangle$, where PLUS and TIMES are the arithmetic relations, i.e., for $i, j, k < n$,

$$\begin{aligned}
\mathcal{A}_n \models \mathrm{PLUS}(i,j,k) \quad &\Leftrightarrow \quad i + j = k \\
\mathcal{A}_n \models \mathrm{TIMES}(i,j,k) \quad &\Leftrightarrow \quad i \cdot j = k
\end{aligned}$$

Write formulas in $\mathcal{L}(\tau)$ that represent the following arithmetic relations,

1. DIVIDES(x, y), meaning that y is a multiple of x.
2. PRIME(x), meaning that x is a prime number.
3. $p_2(x)$, meaning that x is a power of 2.

[Hint for (3): x is a power of 2 iff 2 is the only prime divisor of x.] □

Example 1.6 Here are a few formulas in the language of strings. The first describes the set of strings that have no consecutive "1"s. It uses the abbreviation "$x < y$", meaning "$x \leq y \wedge x \neq y$".

$$\varphi_{no11} \;\equiv\; (\forall x)(\forall y)(\exists z)((S(x) \wedge S(y) \wedge x < y) \rightarrow (x < z < y \wedge \neg S(z)))$$

Formula φ_{five1} below says that the given string contains at least five "1"s. To do so, it uses the abbreviation "distinct":

$$\text{distinct}(x_1, \ldots, x_k) \;\equiv\; (x_1 \neq x_2 \;\wedge\; \cdots \;\wedge\; x_1 \neq x_k \;\wedge\; \cdots \;\wedge\; x_{k-1} \neq x_k)$$

$$\varphi_{five1} \;\equiv\; (\exists uvwxy)(\text{distinct}(u, v, w, x, y) \wedge S(u) \wedge S(v) \wedge S(w) \wedge S(x) \wedge S(y))$$

Note that φ_{five1} uses five variables to say that there are five "1"s. Using the ordering relation, we can reduce the number of variables. The following formula is equivalent to φ_{five1} but uses only two variables:

$$\begin{aligned}(\exists x)\Big(S(x) \wedge (\exists y)\Big(x < y \wedge S(y) \wedge (\exists x)\big(y < x \wedge S(x) \;\wedge \\ (\exists y)\big(x < y \wedge S(y) \wedge (\exists x)y < x \wedge S(x)\big)\big)\Big)\Big)\end{aligned}$$

Read the above sentence carefully. A good way to think of it is that we have two fingers and are trying to count the number of "1"s in a string. We put finger x down on the first "1". Then we put finger y down on the next "1" to the right. Now we don't need x anymore so we can move it to the next "1" to the right of y, and so on.

We will see later that the number of variables is an important descriptive resource. Note that the standard semantics of first-order logic (Definition 1.3) allows us to requantify variables. Each quantifier $(\exists x)$ or $(\forall x)$ bounds only the free occurrences of x within its scope. We will see in Theorem 6.31 that every first-order sentence in $\mathcal{L}(\tau_s)$ — i.e., every sentence about strings — is equivalent to a sentence with only three distinct variables. □

Exercise 1.7 Prove that if interpretations i and i' agree on all the free variables in φ then

$$(\mathcal{A}, i) \models \varphi \quad \Leftrightarrow \quad (\mathcal{A}, i') \models \varphi$$

[Hint: by induction on φ using Definition 1.3.] □

Exercise 1.8 Let $(\exists! x)\alpha(x)$ mean that there exists a unique x such that α. Show how to write $(\exists! x)\alpha(x)$ using the usual quantifiers $\forall, \exists$. □

As another example, let $\tau_{ab} = \langle \leq^2, A^1, B^1 \rangle$ consist of an ordering relation and two monadic relation symbols A and B, each serving the same role as the symbol S in τ_s. Let $\mathcal{A} \in \mathrm{STRUC}[\tau_{ab}]$, and let $n = \|\mathcal{A}\|$. Then $\mathcal{A}$ is a pair of binary strings A, B, each of length n. These binary strings represent natural numbers, where we think of the bit zero as most significant and bit $n-1$ as least significant. Here $A(i)$ is true iff bit i of A is "1".

The following sentence expresses the ordering relation on such natural numbers represented in binary.

$$\mathrm{LESS}(A, B) \quad \equiv \quad (\exists x)(B(x) \wedge \neg A(x) \wedge (\forall y . y < x)(A(y) \rightarrow B(y)))$$

The above sentence uses a very useful abbreviation, that of restricted quantifiers,

$$(\forall x.\alpha)\varphi \;\equiv\; (\forall x)(\alpha \rightarrow \varphi); \qquad (\exists x.\alpha)\varphi \;\equiv\; (\exists x)(\alpha \wedge \varphi)$$

In the next proposition we show that addition is first-order expressible. Addition of natural numbers represented in binary is one of the most basic computations. We will see in Theorem 5.2 that the first-order queries characterize the problems computable in constant parallel time. Thus the following may be thought of as an addition algorithm that runs in constant parallel time.

Proposition 1.9 *Addition of natural numbers, represented in binary, is first-order expressible.*

Proof We use the well-known "carry-look-ahead" algorithm. In order to express addition, we first express the carry bit,

$$\varphi_{carry}(x) \equiv (\exists y . x < y)[A(y) \wedge B(y) \wedge (\forall z . x < z < y) A(z) \vee B(z)]$$

The formula $\varphi_{carry}(x)$ holds if there is a position y to the right of x where $A(y)$ and $B(y)$ are both one (i.e. the carry is generated) and for all intervening positions z, at least one of $A(z)$ and $B(z)$ holds (that is, the carry is propagated). Let $\oplus$ be an abbreviation for the commutative and associative "exclusive or" operation.

We can express φ_{add} as follows,

$$\begin{aligned} \alpha \oplus \beta &\equiv \alpha \leftrightarrow \neg\beta \\ \varphi_{add}(x) &\equiv A(x) \oplus B(x) \oplus \varphi_{carry}(x) \end{aligned}$$

Note that the formula $\varphi_{add}(x)$ has the free variable x. Thus, φ_{add} is a description of n bits: one for each possible value of x. □

An important relation between two structures of the same type is that one may be a substructure of the other. $\mathcal{A}$ is a substructure of $\mathcal{B}$ if the universe of $\mathcal{A}$ is a subset of the universe of $\mathcal{B}$ and the relations and constants on $\mathcal{A}$ are inherited from $\mathcal{B}$.

Definition 1.10 (Substructure) Let $\mathcal{A}$ and $\mathcal{B}$ be structures of the same vocabulary $\tau = \langle R_1^{a_1}, \ldots, R_r^{a_r}, c_1, \ldots, c_s\rangle$. We say that $\mathcal{A}$ is a *substructure* of $\mathcal{B}$, written $\mathcal{A} \le \mathcal{B}$, iff the following conditions hold,

1. $|\mathcal{A}| \subseteq |\mathcal{B}|$
2. For $i = 1, 2, \ldots, r$, $R_i^{\mathcal{A}} = R_i^{\mathcal{B}} \cap |\mathcal{A}|^{a_i}$
3. For $j = 1, 2, \ldots, s$, $c_j^{\mathcal{A}} = c_j^{\mathcal{B}}$. □

See Figure 1.11 where A and B are substructures of G. Note that C is not a substructure of G for two reasons: it doesn't contain the constant t and the induced edge from vertex 1 to vertex 2 is missing.

Exercise 1.12 Let $\mathcal{A} \in \text{STRUC}[\tau]$ be a structure and let $\alpha(x)$ be a formula such that $\mathcal{A} \models (\exists x)\alpha(x)$. Assume also that for every constant symbol c in τ, $\mathcal{A} \models \alpha(c)$. Let $\mathcal{B}$ be the substructure of $\mathcal{A}$ with universe

$$|\mathcal{B}| \quad = \quad \{a \in |\mathcal{A}| \bigm| \mathcal{A} \models \alpha(a)\}$$

Let φ be a sentence in $\mathcal{L}(\tau)$. Define the *restriction* of φ to α to be the sentence φ^α, the result of changing every quantifier $(\forall y)$ or $(\exists y)$ in φ to the restricted quantifier $(\forall y.\alpha(y))$ or $(\exists y.\alpha(y))$ respectively. Prove the following,

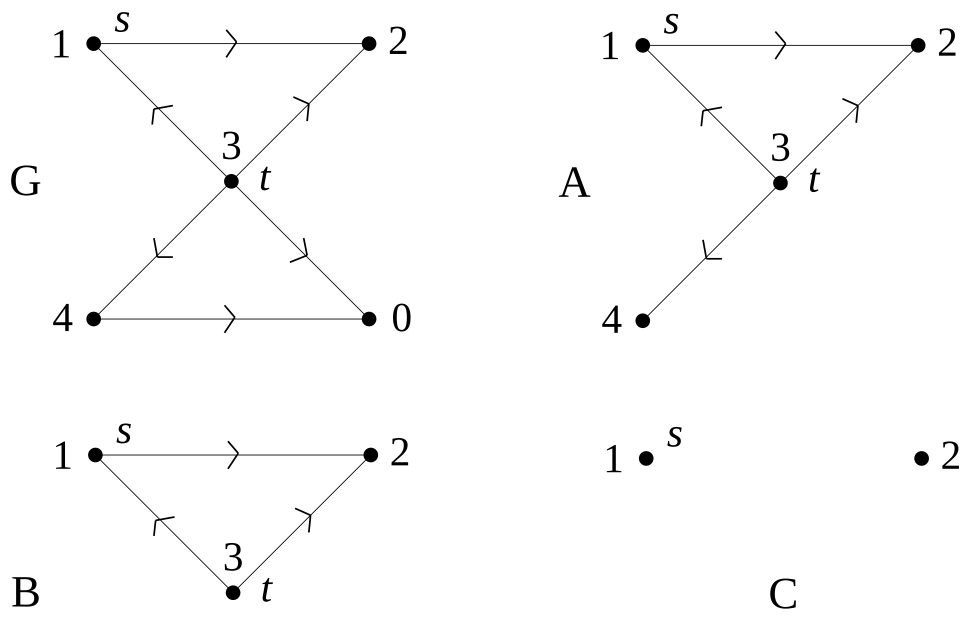

Figure 1.11: A and B but not C are substructures of G.

$$\mathcal{A} \models \varphi^{\alpha} \qquad \Leftrightarrow \qquad \mathcal{B} \models \varphi$$ □

We say that φ is *universal* iff it can be written in prenex form — i.e. with all quantifiers at the beginning — using only universal quantifiers. Similarly, we say that φ is *existential* iff it can be written in prenex form with only existential quantifiers.

The following "preservation theorems" provide a good way of proving that a formula is existential or universal.

Exercise 1.13 Prove the following preservation theorems. Let $\mathcal{A} \leq \mathcal{B}$ be structures and φ a first-order sentence.

1. Suppose φ is existential. If $\mathcal{A} \models \varphi$ then $\mathcal{B} \models \varphi$.
2. Suppose φ is universal. If $\mathcal{B} \models \varphi$ then $\mathcal{A} \models \varphi$.

[Hint: by induction on φ using Definition 1.3.] □

1.2 Ordering and Arithmetic

A logical structure such as a graph does not need to have an ordering on its vertices. However, if we use a computer to store or manipulate this graph, it must be encoded in some way that imposes an ordering on the vertices. In order to discuss computation in general, it is necessary to assume that the universes of our structures are ordered. This section introduces the issue of ordering and explains what we will assume about the ordering of structures in the remainder of this book.

When we code an input to a computer, we do so as a string of characters. There is always an ordering here: the first character, the second character, and so on. Indeed the concept of ordering is deeply embedded in the concepts of string and of computation.

For this reason, the binary relation symbol "$\leq$" plays a special role in descriptive complexity. When "$\leq$" is an element of τ, and $\mathcal{A} \in$ STRUC$[\tau]$, then $\mathcal{A}$ must interpret $\leq$ as a total ordering on its universe. In this case, we also place constant symbols $0, 1, \mathit{max}$ in τ and insist that these be interpreted as the minimum, second, and maximum elements under the $\leq$ ordering. In order for formulas in first-order logic to express general computation, they need access to a total ordering of the universe. The requirement of a total ordering in descriptive complexity is analogous to the assumption of the Axiom of Choice in set theory.

Let $\mathcal{A} \in$ STRUC$[\tau]$ be an ordered structure. Let $n = \|\mathcal{A}\|$. Let the elements of $|\mathcal{A}|$ in increasing order be $a_0, a_1, \ldots, a_{n-1}$. Then there is a 1:1 correspondence $i \mapsto a_i$, $i = 0, 1, \ldots n-1$. We usually identify the elements of the universe with the set of natural numbers less than n. In a computer these would be represented as $\lceil \log n \rceil$-bit words, and the operations plus, times, and even picking out bit j of such a word would all be wired in. The following numeric relations are useful:

1. PLUS(i, j, k) meaning $i + j = k$
2. TIMES(i, j, k) meaning $i \times j = k$
3. BIT(i, j) meaning bit j in the binary representation of i is 1

In the definition of BIT we will take bit 0 to be the low order bit, so BIT$(i, 0)$ holds iff i is odd. We will see in Chapter 11 that adding BIT (or equivalently PLUS and TIMES) to our vocabularies makes the set of first-order definable boolean queries a more robust complexity class.

When working with very weak reductions or proving normal form theorems, we will sometimes use the successor relation SUC in lieu of or in addition to $\leq$. Of course, SUC is first-order definable from $\leq$.

$$\mathrm{SUC}(x,y) \quad \equiv \quad (x<y) \wedge (\forall z)(\neg(x<z \ \wedge \ z<y))$$

The symbols $\leq$, PLUS, TIMES, BIT, SUC, $0, 1,$ *max* are called *numeric* relation and constant symbols. They depend only on the size of the universe. We call the remainder of τ the *input* relation and constant symbols. We will see in Chapter 5 that the choice of numeric relations for weak languages such as FO corresponds to the definition of uniformity for complexity classes defined by uniform sequences of circuits. The numeric relations and constants are not explicitly given in the input since they are easily computable as functions of the size of the input. Whenever any of the numeric relation or constant symbols occur, they are required to have their standard meanings.

Proviso 1.14 (Ordering Proviso) *From now on, unless stated otherwise, we assume that the numeric relations and constants:* $\leq$, PLUS, TIMES, BIT, SUC, $0, 1,$ *max are present in all vocabularies. When we define vocabularies, we do not explicitly mention or show these symbols unless they are* not *present. In Chapter 6 we prove lower bounds on what can be expressed in some first-order language. We use the notation* $\mathcal{L}(\mathrm{wo}\leq)$ *to indicate language* $\mathcal{L}$ *without any of the numeric relations. We will write* $\mathcal{L}(\mathrm{wo\,BIT})$ *to indicate language* $\mathcal{L}$*, including ordering, but not arithmetic, i.e., only the numeric relations* $\leq$ *and* SUC *and the constants* $0, 1,$ *max are included.*

The following proviso is useful. It eliminates the trivial and sometimes annoying case of the structure with only one element which would thus satisfy the equation $0 = 1$. We assume this proviso unless otherwise noted. (The only time we do not assume the existence of boolean constants is in Section 6.5.)

Proviso 1.15 (Boolean Constants) *From now on, we assume that all structures have at least two elements. In particular, we will assume that we have two unequal constants denoted by* 0 *and* 1.

Next, we define what it means to have a boolean variable in a first-order formula. Boolean variables allow a more robust measure of the number of first-order variables needed to express a query. When we measure the number of first-order variables needed, we discount the (bounded) number of boolean variables.

Definition 1.16 A *boolean variable* in a first-order formula is a variable that is restricted to being either 0 or 1. Here 0 is identified with **false** and 1 is identified with **true**. We typically use the letters b, c, d, e for boolean variables. We use the following abbreviations:

$$\begin{aligned} \text{bool}(b) &\equiv b \le 1 \\ (\exists b) &\equiv (\exists b.\text{bool}(b)) \\ (\forall b) &\equiv (\forall b.\text{bool}(b)) \end{aligned}$$

□

1.2.1 $\text{FO(BIT)} = \text{FO(PLUS, TIMES)}$

In the remainder of this section we prove that adding BIT to first-order logic is equivalent to adding PLUS and TIMES. In order to prove this, we also need to prove the Bit Sum Lemma which is interesting in its own right. The proofs in this subsection are very technical and may safely be skipped at first reading.

Theorem 1.17 *Let τ be a vocabulary that includes ordering. Then*

1. *If* $\text{BIT} \in \tau$ *then* PLUS *and* TIMES *are first-order definable.*
2. *If* $\text{PLUS}, \text{TIMES} \in \tau$ *then* BIT *is first-order definable.*

Proof To prove (1), we have essentially seen in Proposition 1.9 that PLUS is expressible using BIT. To prove that TIMES is expressible we first need the following:

Lemma 1.18 (Bit Sum Lemma) *Let* $\text{BSUM}(x, y)$ *be true iff y is equal to the number of ones in the binary representation of x.* BSUM *is first-order expressible using ordering and* BIT.

Proof The bit-sum problem is to add a column of $\log n$ 0's and 1's. The idea is to keep a running sum. Since the sum of $\log n$ 1's requires at most $\log \log n$ bits to record, we maintain running sums of $\log \log n$ bits each. With one existentially quantified variable, we can guess $\log n / \log \log n$ of these. Thus, to express $\text{BSUM}(x, y)$ we existentially quantify s — the $\log \log n \cdot (\log n / \log \log n)$ bits of running sums. In the following example, $n = 2^{16}$, so x and y each have 16 bits. To assert BSUM$(0110110110101101, 1010)$ we would guess $s = 0010010101111010$ as our partial sum bit string.

0		
1		
1		
0	0010	
1		
1		
0		
1	0101	
1		
0		
1		
0	0111	
1		
1		
0		
1	1010	BSUM$(0110110110101101, 1010)$

Next we say that for all i where $i \leq \log n / \log\log n$, running sum i, plus the number of 1's in segment $(i+1)$ is equal to the running sum $(i+1)$.

Thus, it suffices to express the bit sum of a segment of length $\log\log n$. This we can do by keeping a running sum at every position because this requires only $\log\log\log n \cdot \log\log n$, which is less than $\log n$ for sufficiently large n. $\Box$

We next show that TIMES is first-order expressible using BIT. TIMES is equivalent to the addition of $\log n$ $\log n$-bit numbers,

$$A \quad = \quad A_1 + A_2 + \cdots + A_{\log n}$$

The first trick we employ is to split each A_i into a sum of two numbers, $A_i = B_i + C_i$, so that B_i and C_i have blocks of $\log\log n$ bits separated by $\log\log n$ 0's. We compute the sum of the B_i's and of the C_i's. In this way, we insure that no carries extend more than $\log\log n$ bits. Finally, we add the two sums with a single use of PLUS. In the following, let $\ell = \lceil \log\log n \rceil$.

$$\begin{array}{rcccccccccc}
B_i & = & a_{i,1} & \cdots & a_{i,\ell} & 0 & \cdots & 0 & \cdots & a_{i,\log n+1-\ell} & \cdots & a_{i,\log n} \\
+\, C_i & = & 0 & \cdots & 0 & a_{i,\ell+1} & \cdots & a_{i,2\ell} & \cdots & 0 & \cdots & 0 \\
\hline
A_i & = & a_{i,1} & \cdots & a_{i,\ell} & a_{i,\ell+1} & \cdots & a_{i,2\ell} & \cdots & a_{i,\log n+1-\ell} & \cdots & a_{i,\log n}
\end{array}$$

Position	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Y	0	1	0	0	0	0	0	0	0	1	0	0	0	1	0	1	0
Z	0	0	0	0	1	0	0	0	0	0	0	1	0	0	1	0	0
I	0	1	1	1	1	0	0	0	0	1	1	1	0	1	1	0	0

Table 1.19: Encoding of an arithmetic fact: $2^{15} = 32,768$.

In this way, we have reduced the problem of adding $\log n$ $\log n$-bit numbers to that of adding $\log n$ $\log\log n$-bit numbers. We can simultaneously guess the sums of each of the $\log\log n$ columns in a single variable, c. Using BSUM and a universal quantifier we can verify that each section of c is correct. Finally, we can add the $\log\log n$ numbers in c maintaining all the running sums as in the last paragraph of the proof of the Bit Sum Lemma.

2. In this direction, we want to show that BIT is first-order expressible using PLUS and TIMES. We do this with a series of definitions. First, let $p_2(y)$ mean that y is a power of 2. (See Exercise 1.5).

Next, define BIT$'(x, y)$ to mean that for some i, $y = 2^i$ and BIT(x, i),

$$\text{BIT}'(x, y) \quad \equiv \quad p_2(y) \wedge (\exists uv)(x = 2uy + y + v \ \wedge \ v < y)$$

Using BIT$'$ we can copy a sequence of bits. For example, the following formula says that if $y = 2^i$ and $z = 2^j$, then bits $i + j..i$ of x are the same as bits $j..0$ of c:

$$\text{COPY}(x, y, z, c) \quad \equiv \quad (\forall u.p_2(u) \wedge u \leq z)(\text{BIT}'(x, yu) \leftrightarrow \text{BIT}'(c, u))$$

Finally, to express BIT, we would like to express the relation $2^i = y$. We express this using the following recurrence,

$$2^i = y \quad \Leftrightarrow \quad (\exists j)(\exists z.2^j = z)(i = 2j + 1 \wedge y = 2z^2 \ \vee \ i = 2j \wedge y = z^2) \tag{1.20}$$

We can guess three variables, Y, Z, I, that simultaneously include all but a bounded number of the $\log i$ computations indicated by Equation (1.20), namely all those such that $i > 2\log i$. This is done as follows: Place a "1" in positions i, j, etc., of Y. Place the binary encoding of i starting at position i of I, the binary encoding of j starting at position j of I and so on. Finally, place a "1" in Z at the end of each of the binary encodings of exponents.

Using a universal quantifier we say that the variables Y, Z, and I encode all the relevant and sufficiently large computations of Equation (1.20). Table 1.19

shows the encodings Y, Z, and I for the proposition that $2^{15} = 32,768$. Note that I records the exponent 15, which is 1111 in binary, starting at position 15; 7 which is 111 in binary, starting at position 7; and 3 which is 11 in binary, starting at position 3. We leave the details of actually writing the relevant first-order formula as an exercise. □

1.3 Isomorphism

When we impose an ordering on the universe of a structure, we have essentially labeled its elements $0, 1$, and so on. It becomes interesting and important to know when we have used this ordering in an essential way. For this we need the concept of isomorphism. Two structures are isomorphic iff they are identical except perhaps for the names of the elements of their universes:

Definition 1.21 (Isomorphism of Unordered Structures) Let $\mathcal{A}$ and $\mathcal{B}$ be structures of vocabulary $\tau = \langle R_1^{a_1}, \ldots, R_r^{a_r}, c_1, \ldots, c_s \rangle$. We say that $\mathcal{A}$ is *isomorphic* to $\mathcal{B}$, written, $\mathcal{A} \cong \mathcal{B}$, iff there is a map $f : |\mathcal{A}| \to |\mathcal{B}|$ with the following properties:

1. f is 1:1 and onto.
2. For every input relation symbol R_i and for every a_i-tuple of elements of $|\mathcal{A}|$, $e_1, \ldots, e_{a_i}$,
$$\langle e_1, \ldots, e_{a_i} \rangle \in R_i^{\mathcal{A}} \quad \Leftrightarrow \quad \langle f(e_1), \ldots, f(e_{a_i}) \rangle \in R_i^{\mathcal{B}}$$
3. For every input constant symbol c_i, $f(c_i^{\mathcal{A}}) = c_i^{\mathcal{B}}$

The map f is called an *isomorphism*. □

As an example, see graphs G and H in Figure 1.1 which are isomorphic using the map that adds one mod five to the numbers of the vertices of G.

Note that we have defined isomorphisms so that they need only preserve the input symbols, not the ordering and other numeric relations. If we included the ordering relation then we would have $A \cong \mathcal{B}$ iff $\mathcal{A} = \mathcal{B}$. To be completely precise, we should call the mapping f defined above an "isomorphism of unordered structures" and say that $\mathcal{A}$ and $\mathcal{B}$ are "isomorphic as unordered structures". (Note also that, since "unordered string" does not make sense, neither does the concept of isomorphism for strings. By a strict interpretation of Definition 1.21, two strings would be isomorphic as unordered structures iff they had the same number of each symbol.)

The following proposition is basic.

Proposition 1.22 *Suppose $\mathcal{A}$ and $\mathcal{B}$ are isomorphic. Then for all sentences $\varphi \in \mathcal{L}(\tau - \{\leq\})$, $\mathcal{A}$ and $\mathcal{B}$ agree on φ.*

Exercise 1.23 Prove Proposition 1.22. [Hint: do this by induction using Definition 1.3.] □

1.4 First-Order Queries

As mentioned in the introduction, we use the concept of query as the fundamental paradigm of computation:

Definition 1.24 A *query* is any mapping $I : \text{STRUC}[\sigma] \to \text{STRUC}[\tau]$ from structures of one vocabulary to structures of another vocabulary, that is polynomially bounded. That is, there is a polynomial p such that for all $\mathcal{A} \in \text{STRUC}[\sigma]$, $\|I(\mathcal{A})\| \leq p(\|\mathcal{A}\|)$. A *boolean query* is a map $I_b : \text{STRUC}[\sigma] \to \{0, 1\}$. A boolean query may be thought of as a subset of STRUC$[\sigma]$ — the set of structures $\mathcal{A}$ for which $I_b(\mathcal{A}) = 1$.

An important subclass of queries are the order-independent queries. (In database theory the term “generic” is often used instead of “order-independent”.) Let I be a query defined on STRUC$[\sigma]$. Then I is *order-independent* iff for all isomorphic structures $\mathcal{A}, \mathcal{B} \in \text{STRUC}[\sigma]$, $I(\mathcal{A}) \cong I(\mathcal{B})$. For boolean queries, this last condition translates to $I(\mathcal{A}) = I(\mathcal{B})$. □

From our point of view, the simplest kind of query is a first-order query. As an example, any first-order sentence $\varphi \in \mathcal{L}(\tau)$ defines a boolean query I_φ on STRUC$[\tau]$ where $I_\varphi(\mathcal{A}) = 1$ iff $\mathcal{A} \models \varphi$.

For example, let DIAM[8] be the query on graphs that is true of a graph iff its diameter is at most eight. This is a first-order query given by the formula,

$$\text{DIAM}[8] \quad \equiv \quad (\forall xy)\varphi_{dist8}$$

where φ_{dist8}, meaning that there is a path from x to y of length at most eight, was written in Example 1.4.

As another example, consider the query I_{add}, which, given a pair of natural numbers represented in binary, returns their sum. This query is defined by the first-order formula φ_{add} from Proposition 1.9. More explicitly, let $\mathcal{A} = \langle |\mathcal{A}|, \leq, A, B\rangle$ be any structure in STRUC$[\tau_{ab}]$. $\mathcal{A}$ is a pair of natural numbers each of $n = \|\mathcal{A}\|$ bits. Their sum is given by $I_{add}(\mathcal{A}) = \langle |\mathcal{A}|, S\rangle$ where,

$$S = \big\{a \in |\mathcal{A}| \;\big|\; (\mathcal{A}, a/x) \models \varphi_{add}\big\} \tag{1.25}$$

The first-order query I_{add} : STRUC$[\tau_{ab}] \to$ STRUC$[\tau_s]$ maps structure $\mathcal{A}$ to another structure with the same universe, i.e., $|\mathcal{A}| = |I_{add}(\mathcal{A})|$. The following is a general definition of a k-ary first-order query. Such a query maps any structure $\mathcal{A}$ to a structure whose universe is a first-order definable subset of all k-tuples from $|\mathcal{A}|$. Each relation R_i over $I(\mathcal{A})$ is a first-order definable subset of $|I(\mathcal{A})|^{a_i}$. The constants of $I(\mathcal{A})$ are first-order definable elements of $|\mathcal{A}|^k$.

Definition 1.26 (First-Order Queries) Let σ and τ be any two vocabularies where $\tau = \langle R_1^{a_1}, \ldots, R_r^{a_r}, c_1, \ldots, c_s\rangle$, and let k be a fixed natural number. We want to define the notion of a first-order query,

$$I : \text{STRUC}[\sigma] \to \text{STRUC}[\tau] \ .$$

I is given by an $r+s+1$-tuple of formulas, $\varphi_0, \varphi_1, \ldots, \varphi_r, \psi_1, \ldots, \psi_s$, from $\mathcal{L}(\sigma)$. For each structure $\mathcal{A} \in$ STRUC$[\sigma]$, these formulas describe a structure $I(\mathcal{A}) \in$ STRUC$[\tau]$,

$$I(\mathcal{A}) \quad = \quad \langle |I(\mathcal{A})|, R_1^{I(\mathcal{A})}, \ldots, R_r^{I(\mathcal{A})}, c_1^{I(\mathcal{A})}, \ldots, c_s^{I(\mathcal{A})}\rangle \ .$$

The universe of $I(\mathcal{A})$ is a first-order definable subset[1] of $|\mathcal{A}|^k$,

$$|I(\mathcal{A})| \quad = \quad \big\{\langle b^1, \ldots, b^k\rangle \;\big|\; \mathcal{A} \models \varphi_0(b^1, \ldots, b^k)\big\}$$

Each relation $R_i^{I(\mathcal{A})}$ is a first-order definable subset of $|I(\mathcal{A})|^{a_i}$,

$$R_i^{I(\mathcal{A})} \quad = \quad \big\{(\langle b_1^1, \ldots, b_1^k\rangle, \ldots, \langle b_{a_i}^1, \ldots, b_{a_i}^k\rangle) \in |I(\mathcal{A})|^{a_i} \;\big|\; \mathcal{A} \models \varphi_i(b_1^1, \ldots, b_{a_i}^k)\big\} \ .$$

Each constant symbol $c_j^{I(\mathcal{A})}$ is a first-order definable element of $|I(\mathcal{A})|$,

$$c_j^{I(\mathcal{A})} \quad = \quad \text{the unique } \langle b^1, \ldots, b^k\rangle \in |I(\mathcal{A})| \text{ such that } \mathcal{A} \models \psi_j(b^1, \ldots, b^k) \ .$$

[1]Usually we will take $\varphi_0 \equiv$ **true**, thus letting $|I(\mathcal{A})| = |\mathcal{A}|^k$, cf. Remark 1.32.

When we need to be formal, we let $a = \max\{a_i \,|\, 1 \leq i \leq r\}$ and let the free variables of φ_i be $x_1^1, \ldots x_1^k, \ldots, x_{a_i}^1, \ldots, x_{a_i}^k$. The free variables of φ_0 and the ψ_j's are $x_1^1, \ldots, x_1^k$.

If the formulas ψ_j have the property that for all $\mathcal{A} \in \text{STRUC}[\sigma]$,

$$\left|\left\{\langle b^1, \ldots, b^k\rangle \in |\mathcal{A}|^k \;\middle|\; (\mathcal{A}, b^1/x_1^1, \cdots, b^k/x_1^k) \models \varphi_0 \wedge \psi_j\right\}\right| = 1$$

then we write $I = \lambda_{x_1^1 \ldots x_a^k}\langle \varphi_0, \ldots, \psi_s\rangle$ and say that I is a *k-ary first-order query* from STRUC$[\sigma]$ to STRUC$[\tau]$.

It is often possible to name constant $c_j^{I(\mathcal{A})}$ explicitly as a k-tuple of constants, $\langle t^1, \ldots, t^k\rangle$. In this case, we may simply write this tuple in place of its corresponding defining formula,

$$\psi_j \quad \equiv \quad x_1^1 = t^1 \wedge \cdots \wedge x_1^k = t^k \ .$$

As another example, in a 3-ary query I, the numerical constants $0, 1$, and *max* will be mapped to the following:

$$0^{I(\mathcal{A})} = \langle 0,0,0\rangle; \quad 1^{I(\mathcal{A})} = \langle 0,0,1\rangle; \quad \mathit{max}^{I(\mathcal{A})} = \langle \mathit{max}, \mathit{max}, \mathit{max}\rangle$$

A *first-order query* is either boolean, and thus defined by a first-order sentence, or is a k-ary first-order query, for some k.

Let FO be the set of first-order boolean queries. Let $Q(\text{FO})$ be the set of all first-order queries. □

Example 1.27 Consider the genealogical database from Example 1.2. The following pair of formulas define a unary query, $I_{sa} = \lambda_{xy}\langle \textbf{true}, \varphi_{sibling}, \varphi_{aunt}\rangle$, from genealogical databases to structures of vocabulary $\langle \text{SIBLING}^2, \text{AUNT}^2\rangle$:

$$\begin{aligned}
\varphi_{sibling}(x,y) &\equiv (\exists fm)(x \neq y \wedge f \neq m \;\wedge\; P(f,x) \wedge P(f,y) \wedge P(m,x) \wedge P(m,y)) \\
\varphi_{aunt}(x,y) &\equiv (\exists ps(P(p,y) \wedge \varphi_{sibling}(p,s) \\
&\qquad \wedge (s = x \vee S(x,s))) \;\wedge\; F(x)
\end{aligned}$$

Codd defined a database query language as "complete" if it could express all first-order queries. As we will see, many queries of interest are not first-order. One such example is the ancestor query on genealogical databases (Exercise 6.46). □

As another example, the first-order query I_{add} (Equation (1.25)) is a unary query, i.e., $k = 1$, given by $I_{add} = \lambda_x\langle \textbf{true}, \varphi_{add}\rangle$. In this case, $\varphi_0 = \textbf{true}$ means that the universe of $I_{add}(\mathcal{A})$ is equal to the universe of $\mathcal{A}$.

We will see later that $Q(\text{FO})$ is a very robust class of queries. For now, the reader should check the following proposition, which says that first-order queries are closed under composition.

Proposition 1.28 *Let* $I_1 : \text{STRUC}[\sigma] \to \text{STRUC}[\tau]$ *be a* k*-ary first-order query and let* $I_2 : \text{STRUC}[\tau] \to \text{STRUC}[\upsilon]$ *be an* m*-ary first-order query. Then* $I_2 \circ I_1 : \text{STRUC}[\sigma] \to \text{STRUC}[\upsilon]$ *is an* mk*-ary first-order query.*

Exercise 1.29 Consider the following binary first-order query from graphs to graphs: $I = \lambda_{x,y,x',y'}\langle \mathbf{true}, \alpha, \langle 0,0\rangle, \langle \mathit{max}, \mathit{max}\rangle\rangle$, where

$$\alpha(x,y,x',y') \quad \equiv \quad (x = x' \wedge E(y,y')) \;\vee\; (\text{SUC}(x,y) \wedge x' = y' = y)$$

Recall that part of the meaning of this query is that given a structure $\mathcal{A} \in \text{STRUC}[\tau_g]$, with $n = \|\mathcal{A}\|$,

$$|I(\mathcal{A})| \;=\; \big\{\langle i,j\rangle \;\big|\; i,j \in |\mathcal{A}|\big\};\; s^{I(\mathcal{A})} = \langle 0,0\rangle;\; t^{I(\mathcal{A})} = \langle n-1, n-1\rangle\ .$$

1. Show that I has the following interesting property: For all undirected graphs G,

   $$(G \text{ is connected}) \quad \Leftrightarrow \quad (t \text{ is reachable from } s \text{ in } I(G))$$

2. Recall that a graph is *strongly connected* iff for every pair of vertices g and h, there is a path in G from g to h. Modify I to be a 3-ary query I' such that for all directed graphs G,

   $$(G \text{ is strongly connected}) \quad \Leftrightarrow \quad (t \text{ is reachable from } s \text{ in } I'(G))\ .$$

   [Hint: I almost works, but we need to also make sure that there is a path in G from *max* to 0.] □

Exercise 1.30 Show that the set of first-order queries is closed under composition, i.e., prove Proposition 1.28. □

Exercise 1.31 The first-order query I_{add} defined in Equation (1.25) has the defect that it ignores the possibility that the sum of two n-bit numbers might be $n+1$ bits. Show how to define a more robust first-order query that returns the always correct $n+1$-bit sum. Going further, show how to define the first-order query that always returns the correct sum and has no superfluous leading 0's. □

Remark 1.32 If I is a first-order query on ordered structures, then it must include first-order definitions of the numeric relations and constants. Unless we state otherwise, the ordering on $I(\mathcal{A})$ will be the lexicographic ordering of k-tuples $\le^k$ inherited from $\mathcal{A}$: $\le^1 = \le$ and inductively,

$$\begin{aligned}\langle x_1, \ldots, x_k\rangle \le^k \langle y_1, \ldots, y_k\rangle \quad \equiv \quad & x_1 < y_1 \ \lor \ (x_1 = y_1 \ \land \\ & \langle x_2, \ldots, x_k\rangle \le^{k-1} \langle y_2, \ldots, y_k\rangle)\end{aligned}$$

In the following exercise, you are asked to write the definition of the remaining numeric relations and constants, assuming that $\varphi_0 \equiv$ **true**. For the first-order queries in this book, we usually limit ourselves to the case that $\varphi_0 \equiv$ **true**. If not, we must express the new numeric relations explicitly.

Exercise 1.33 Let I be a first-order query on ordered structures. The successor and bit relations must be defined.

1. Give the formulas defining 0, 1, and *max* the minimum, second, and maximum elements of the new universe under the lexicographical ordering. Note that if $\varphi_0 \equiv$ **true**, then the resulting constants are just k-tuples of constants:

   $$0^{I(\mathcal{A})} = \langle 0, \ldots, 0\rangle; \ 1^{I(\mathcal{A})} = \langle 0, \ldots, 0, 1\rangle; \ \mathit{max}^{I(\mathcal{A})} = \langle \mathit{max}, \ldots, \mathit{max}\rangle$$

   However, in the more general case you must use quantifiers to say that the given element is the minimum, second, maximum in the lexicographical ordering.

2. Assuming that $\varphi_0 \equiv$ **true**, write a quantifier-free formula defining the new SUC relation.

3. Assuming that $\varphi_0 \equiv$ **true**, write the formula defining the new BIT relation. [Hint: by Theorem 1.17 you may define addition and multiplication on k-tuples.]

□

Without the assumption that $\varphi_0 \equiv$ **true**, BIT need not be first-order definable in the image structures. For example, if $\sigma = \tau_s$ and $\varphi_0(x) \equiv S(x)$, then the parity of the universe of $I(\mathcal{A})$ is not first-order expressible in $\mathcal{A}$ (Theorem 13.1). If BIT were definable in $I(\mathcal{A})$ then so would the parity of its universe. For this reason, when we define first-order reductions, we restrict our attention to very simple formulas, φ_0, that define the universe of the image structure.

Historical Notes and Suggestions for Further Reading

There are many excellent introductions to logic. We especially recommend [End72] and [EFT94]. The recent books on finite model theory [EF95], [LR96] and [Ott97] complement this book. For history of logic, it is wonderful to go back to some of the original sources, carefully translated and annotated in [vH67].

The definition of semantics for first-order logic (Definition 1.3) is usually attributed to Tarski [Tar36].

Lemma 1.18 was originally proved by Barrington, Immerman, and Straubing in [BIS88]. We got part 2 of Theorem 1.17 from Lindell [L], who says that the result comes from page 299 of Hajek and Pudlak [HP93]. However, on page 406 of [HP93] the result is attributed to Bennett [Ben62]. See also [DDLW98], where Dawar, Lindell, and Weinstein prove that ordering is definable when the only given numeric predicate is BIT. It follows that Theorem 1.17 remains valid when ordering is not given because ordering is easily definable from PLUS.

Exercise 1.12 is from [vD94].

See [CH80a] for perhaps the first study of the set of order-independent, computable queries.

One very important topic in a standard course on first-order logic that is omitted here is the study of proofs. In particular, the following two theorems are basic and appear in every standard logic book. They were originally proved by Gödel in his Ph.D. thesis [Göd30].

Theorem 1.34 (Completeness Theorem for First-Order Logic) *There is a complete recursive axiomatization for the set of formulas valid in all — finite and infinite — structures.*

Theorem 1.35 (Compactness Theorem for First-Order Logic) *Let* Γ *be a set of first-order formulas with the property that every finite subset of* Γ *has a (perhaps infinite) model. Then* Γ *has a (perhaps infinite) model.*

These theorems fail when we restrict our attention to finite structures. From the Completeness Theorem it follows that the set of valid formulas for first-order logic is recursively enumerable (r.e.), and VALID is in fact r.e.-complete. Thus, the set of satisfiable formulas is not r.e. For finite structures, Trahtenbrot's Theorem says that the reverse is true: The set of formulas satisfiable in a finite structure is r.e.-complete, so the set of formulas valid in all finite structures is not r.e., and thus not axiomatizable [Tra50].

For a finite alphabet, $\Sigma = \{\sigma_1, \ldots, \sigma_r\}$, consider the vocabulary τ_Σ = $\langle S^1_{\sigma_1}, \ldots, S^1_{\sigma_r} \rangle$. The set STRUC[$\tau_\Sigma$] consists of the set of non-empty words of vocabulary Σ. Languages without BIT are well-studied for these vocabularies. The following two theorems are fundamental:

Theorem 1.36 *The set of boolean queries expressible in second-order, monadic logic, without* BIT*, over the vocabularies* τ_Σ *consist exactly of the regular languages. In symbols,*

$$\mathrm{SO(monadic)(wo\ BIT)} \qquad = \qquad \text{Regular} \ .$$

Theorem 1.37 *The set of boolean queries expressible in first-order logic, without* BIT*, over the vocabularies* τ_Σ *consist exactly of the star-free regular languages. In symbols,*

$$\mathrm{FO(wo\ BIT)} \qquad = \qquad \text{star-free Regular} \ .$$

Theorem 1.36 is due to Büchi [Büc60] and Theorem 1.37 is due to McNaughton and Papert, [MP71]. From the point of view of this book, the languages without BIT are slightly too weak to provide a robust view of computation. A good reference for these results and other relations between logic and automata theory is [Str94] by Straubing.

Chapter 2

Background in Complexity

Computational Complexity measures the amount of computational resources, such as time and space, that are needed, as a function of the size of the input, to compute a query. This chapter introduces the reader to complexity theory. We define the complexity measures and complexity classes that we study in the rest of the book. We also explain some of their basic properties, complete problems, and inter-relationships.

2.1 Introduction

In the 1930's many models of computation were invented, including Church's lambda calculus, Gödel's recursive functions, Markov algorithms and Turing machines. It is very striking that these interesting and apparently different models — all independent efforts to precisely define the intuitive notion of "mechanical procedure" — were proved equivalent. This leads to the universally accepted "Church's thesis", which states that the intuitive concept of what can be "automatically computed" is appropriately captured by the Turing machine (and all its variants).

If one appropriately measures the complexity of computation in a Markov algorithm, a lambda expression, a recursive function, or a Turing machine, one obtains equivalent values. A consequence of this is that efficiency is not model-dependent, but is in fact a fundamental concept.

We get the same theory of complexity whether we approach it via Turing machines or any of these other models. As we will see in this book, descriptive complexity gives definitions of complexity that are equivalent to those of all the above models. Different formalisms lend themselves to different ways of thinking and working. We find that the insights gained from the descriptive approach to complexity offer a different point of view from more traditional machine-based complexity. In particular, there are well understood methods in logic for ascertaining what can and cannot be expressed in a given language. We will introduce some of these methods in Chapter 6.

In Descriptive complexity, we measure the difficulty of describing queries. We will see that natural measures of descriptive complexity such as depth of nesting of quantifiers and number of variables correspond closely to natural notions of complexity in Turing machines.

2.2 Preliminary Definitions

We assume that the reader is familiar with the Turing machine. We start from there and present a survey of computational complexity theory.

We write $M(w)\downarrow$ to mean that Turing machine M accepts input w, and we write $L(M)$ to denote the language accepted by M,

$$L(M) \;=\; \big\{w \in \{0,1\}^* \;\big|\; M(w)\downarrow\big\}\;.$$

Instead of just accepting or rejecting, Turing machines may compute functions from binary strings to binary strings. We use $T(w)$ to denote the binary string that Turing machine T leaves on its write-only output tape when it is started with the binary string w on its input tape. If T does not halt on input w, then $T(w)$ is undefined.

Everything that a Turing machine does may be thought of as a query from binary strings to binary strings. In order to make Descriptive complexity rich and flexible it is useful to consider queries that use other vocabularies. To relate such queries to Turing machine complexity, we fix a scheme that encodes the structures of vocabulary τ as boolean strings. To do this, for each τ, we define an encoding query,

$$\text{bin}_\tau : \text{STRUC}[\tau] \to \text{STRUC}[\tau_s]$$

Recall that $\tau_s = \langle S^1\rangle$ is the vocabulary of boolean strings. The details of the encoding are not important, but it is useful to know that for each τ, bin_τ and its inverse are first-order queries (Exercise 2.3).

Definition 2.1 (The binary encoding of structures: bin($\mathcal{A}$)) Let $\tau = \langle R_1^{a_1}, \ldots, R_r^{a_r}, c_1, \ldots, c_s\rangle$ be a vocabulary, and let $\mathcal{A} = \langle\{0, 1, \ldots, n -$

$1\}, R_1^{\mathcal{A}} ... R_r^{\mathcal{A}}, c_1^{\mathcal{A}} ... c_s^{\mathcal{A}}\rangle$ be an ordered structure of vocabulary τ. The relation $R_i^{\mathcal{A}}$ is a subset of $|\mathcal{A}|^{a_i}$. We encode this relation as a binary string $\text{bin}^{\mathcal{A}}(R_i)$ of length n^{a_i} where "1" in a given position indicates that the corresponding tuple is in $R_i^{\mathcal{A}}$. Similarly, for each constant $c_j^{\mathcal{A}}$, its number is encoded as a binary string $\text{bin}^{\mathcal{A}}(c_j)$ of length $\lceil \log n \rceil$.

The binary encoding of the structure $\mathcal{A}$ is then just the concatenation of the bit strings coding its relations and constants,

$$\text{bin}_\tau(\mathcal{A}) \quad = \quad \text{bin}^{\mathcal{A}}(R_1)\text{bin}^{\mathcal{A}}(R_2)\cdots\text{bin}^{\mathcal{A}}(R_r)\text{bin}^{\mathcal{A}}(c_1)\cdots\text{bin}^{\mathcal{A}}(c_s)$$

We do not need any separators between the various relations and constants because the vocabulary τ and the length of $\text{bin}_\tau(\mathcal{A})$ determines where each section belongs. Observe that the length of $\text{bin}_\tau(\mathcal{A})$ is given by

$$\hat{n}_\tau \quad = \quad \|\text{bin}_\tau(\mathcal{A})\| \quad = \quad n^{a_1} + \cdots + n^{a_r} + s\lceil \log n \rceil \qquad (2.2)$$

Note: We do not bother to include any numeric predicates or constants in $\text{bin}_\tau(\mathcal{A})$ since they can be easily recomputed. However, the coding $\text{bin}_\tau(\mathcal{A})$ does presuppose an ordering on the universe. There is no way to code a structure as a string without an ordering. Since a structure determines its vocabulary, in the sequel we usually write $\text{bin}(\mathcal{A})$ instead of $\text{bin}_\tau(\mathcal{A})$ for the binary encoding of $\mathcal{A} \in \text{STRUC}[\tau]$. Here bin is the union of bin_τ over all vocabularies τ. In the special case where τ includes no input relations symbols, we pretend that there is a unary relation symbol that is always false. For example, if $\tau = \emptyset$, then $\text{bin}(\mathcal{A}) = 0^{\|\mathcal{A}\|}$. We do this to insure that the size of $\text{bin}(\mathcal{A})$ is at least as large as $\|\mathcal{A}\|$. $\square$

When $\tau = \tau_s$, the map bin_{τ_s} maps strings to strings. The reader should check from the above definition that in this case, bin_{τ_s} is the identity map and thus $\hat{n}_{\tau_s} = n$.

In complexity theory, n is usually reserved for the length of the input. However, in this book, n is used to denote the size of the input structure, $n = \|\mathcal{A}\|$. When the inputs are structures of vocabulary τ, the length of the input is $\hat{n}_\tau$. For the case of binary strings, these two sizes coincide because $\hat{n}_{\tau_s} = n$. When τ is understood, we write $\hat{n}$ for $\hat{n}_\tau$. Observe that n and $\hat{n}$ are always polynomially related.

There are two requirements of a coding function such as "bin". First, it must be computationally very easy to encode and decode. Secondly, the coding must be fairly space efficient, e.g., coding in unary would not be acceptable. In the next exercise, the reader is asked to show that both the encoding and decoding of bin are first-order expressible.

Exercise 2.3 Show that for any vocabulary τ, the queries $\text{bin}_\tau : \text{STRUC}[\tau] \to \text{STRUC}[\tau_s]$ and its inverse $\text{bin}_\tau^{-1} : \text{STRUC}[\tau_s] \to \text{STRUC}[\tau]$ are first-order queries. More explicitly, let $\tau = \langle R_1^{a_1}, \ldots, R_r^{a_r}, c_1, \ldots, c_s \rangle$.

1. Construct a first-order query β_τ that is equal to the mapping bin_τ.
2. Construct a first-order query $\delta_\sigma : \text{STRUC}[\tau_s] \to \text{STRUC}[\tau]$, such that for all $\mathcal{A} \in \text{STRUC}[\tau]$, $\quad \delta_\sigma(\text{bin}_\tau(\mathcal{A})) = \mathcal{A}$. The query δ should be unary, that is, $k = 1$ in the definition of k-ary first-order query (Definition 1.26). $\square$

Using the encoding bin, we define what it means for a Turing machine to compute a query:

Definition 2.4 Let $I : \text{STRUC}[\sigma] \to \text{STRUC}[\tau]$ be a query. Let T be a Turing machine. Suppose that for all $\mathcal{A} \in \text{STRUC}[\sigma]$, $T(\text{bin}(\mathcal{A})) = \text{bin}(I(\mathcal{A}))$. Then we say that T *computes* I. $\square$

We use the notation DTIME$[t(n)]$ to denote the set of boolean queries that are computable by a deterministic, multi-tape Turing machine in $O(t(n))$ steps for inputs of universe size n.[1] Similarly we use NTIME$[t(n)]$, DSPACE$[s(n)]$, and NSPACE$[s(n)]$ to denote nondeterministic time, deterministic space and nondeterministic space, respectively. We assume that the reader is familiar with the following classical complexity classes: L = DSPACE$[\log n]$, NL = NSPACE$[\log n]$, P = polynomial time = $\bigcup_{k=1}^{\infty}$ DTIME$[n^k]$, NP = nondeterministic polynomial time = $\bigcup_{k=1}^{\infty}$ NTIME$[n^k]$, PSPACE = polynomial space = $\bigcup_{k=1}^{\infty}$ DSPACE$[n^k]$= $\bigcup_{k=1}^{\infty}$ NSPACE$[n^k]$, and EXPTIME = exponential time = $\bigcup_{k=1}^{\infty}$ DTIME$[2^{n^k}]$. To talk about space $s(n)$, for $s(n) < \hat{n}$, the Turing machine is assumed to have a read-only input tape of length $\hat{n}$ and some number of work tapes of total length $O(s(n))$.

In the definition of complexity classes, we consider only boolean queries. This is in order to be consistent with the standard definitions of complexity classes as sets of decision problems, i.e., boolean queries. For any complexity class $\mathcal{C}$, we use the notation $Q(\mathcal{C})$ to denote the set of all queries that are computable in the complexity class $\mathcal{C}$. Since $\mathcal{C}$ just consists of boolean queries, what does it mean for

[1]The usual definition in complexity theory writes $t(n)$ as the function $t'(\hat{n})$, a polynomially-related function of the length of the encoding of the input. We use n to be the size of the universe of the input structure and measure all sizes in this uniform way.

a general query to be "computable in $\mathcal{C}$"? It means that each bit of bin$(I(\mathcal{A}))$ is uniformly computable in $\mathcal{C}$ from bin$(\mathcal{A})$. In other words,

Definition 2.5 ($Q(\mathcal{C})$, the Queries Computable in $\mathcal{C}$) Let I : STRUC$[\sigma] \to$ STRUC$[\tau]$ be a query. We say that *I is computable in $\mathcal{C}$* iff the boolean query I_b is an element of $\mathcal{C}$, where

$$I_b \ = \ \left\{(\mathcal{A}, i, a) \ \middle| \ \text{The } i^{\text{th}} \text{ bit of bin}(I(\mathcal{A})) \text{ is "}a\text{"}\right\} . \tag{2.6}$$

Let $Q(\mathcal{C})$ be the set of all queries computable in $\mathcal{C}$:

$$Q(\mathcal{C}) \quad = \quad \mathcal{C} \cup \left\{I \ \middle| \ I_b \in \mathcal{C}\right\} \qquad \square$$

For each of the above resources (deterministic and nondeterministic time and space) there is a hierarchy theorem saying that more of the given resource enables us to compute more boolean queries (see Exercise 2.8). These theorems are proved by diagonalization arguments.

We say that a function $s : \mathbf{N} \to \mathbf{N}$ is *space constructible* (resp. *time constructible*) iff there is a deterministic Turing machine running in space $O(s(n))$, (resp. time $O(s(n))$) that on input 0^n, i.e., n in unary, computes $s(n)$ in binary. This is the same thing as saying that $s' \in Q($DSPACE$[s(n)])$, resp. $s' \in Q($DTIME$[s(n)])$ where s' is the function that on input 0^n computes $s(n)$ in binary.

Every reasonable function is constructible, as is every function one finds in this book. Many theorems in this book need to assume that the time and space bounds in question are constructible.

Exercise 2.7

1. Show that the following functions are time constructible: n, n^2, $\lceil n \log n \rceil$, 2^n.

2. Show that the following are space constructible: n, n^2, $\lceil n \log n \rceil$, 2^n, $\lceil \log n \rceil$.

$\square$

Exercise 2.8 Prove the **Space Hierarchy Theorem:** For all space constructible $s(n) \geq \log n$, if $\lim_{n\to\infty}(t(n)/s(n)) = 0$, then DSPACE$[t(n)]$ is strictly contained in DSPACE$[s(n)]$.

[Hint: this is a diagonalization argument, but you have to be careful. On input M, the diagonalization program marks off $s(|M|)$ tape cells and then simulates machine M on input M. If $M(M)$ exceeds the given space or loops, then it should accept. Otherwise, do the opposite of what M would do.] $\square$

When comparing different resources, we are able to prove much less. For example, by Savitch's Theorem (Theorem 2.32), for $s(n) \geq \log n$,

$$\text{DSPACE}[s(n)] \;\subseteq\; \text{NSPACE}[s(n)] \;\subseteq\; \text{DSPACE}[(s(n))^2] \;;$$

However, we know only the trivial relationships between nondeterministic and deterministic time:

$$\text{DTIME}[t(n)] \;\subseteq\; \text{NTIME}[t(n)] \;\subseteq\; \text{DTIME}[2^{O(t(n))}] \;.$$

Consider the following series of containments. It follows from Savitch's Theorem and the Space Hierarchy Theorem that NL is not equal to PSPACE; but even now, more than twenty-five years after the introduction of the classes P and NP, no other inequalities, including that L is not equal to NP, are known.

$$\text{L} \;\subseteq\; \text{NL} \;\subseteq\; \text{P} \;\subseteq\; \text{NP} \;\subseteq\; \text{PSPACE}$$

2.3 Reductions and Complete Problems

There are several ways to compare the complexity of boolean queries. Perhaps the most natural is the Turing reduction. Complexity in this model is defined via oracles. Let A and B be boolean queries that may be difficult to compute. An *oracle* for B is a mythical device that when given a structure $\mathcal{B}$ will answer in unit time whether or not $\mathcal{B}$ satisfies query B.

Turing gave his original definition of oracles for the case of unsolvable problems. After proving that the halting problem, K, is undecidable, he considered the question of what can be decided by a Turing machine that had an oracle for K. In complexity theory, we use oracles for computable but difficult sets. Such an oracle can speed up some computations.

We say that A is *Turing reducible* to B if it is easy to compute query A given an oracle for B. The following makes this definition more precise.

Definition 2.9 An *oracle Turing machine* is a Turing machine equipped with an extra tape called the query tape. Let M be an oracle Turing machine and B be any boolean query. We write M^B to denote the oracle Turing machine M equipped with an oracle for set B. M^B may write on its query tape like any other tape. At any time, M^B may enter the "query state". Assume that the string $w = \text{bin}(\mathcal{A})$ is written on the query tape when M^B enters the query state. At the next step, "1" will appear on the query tape if $\mathcal{A} \in B$ and "0" otherwise. Thus, M^B may answer any membership question "Does $\mathcal{A}$ satisfy B?" in linear time: the time to copy the string $\text{bin}(\mathcal{A})$ to its query tape.

Given two boolean queries A, B and a complexity class $\mathcal{C}$, we say that A is *$\mathcal{C}$-Turing reducible* to B iff there exists an oracle Turing machine M such that M^B runs in complexity class $\mathcal{C}$ and $\mathcal{L}(M^B) = A$. We denote this by $A \leq_{\mathcal{C}}^{T} B$. The superscript "$T$" stands for Turing reduction. An important example is the polynomial-time Turing reduction, $\leq_{\text{p}}^{T}$. □

Example 2.10 As an example of a Turing reduction, define the boolean query CLIQUE to be the set of pairs $\langle G, k\rangle$ such that G is a graph having a complete subgraph of size k. The vocabulary for CLIQUE is $\tau_{gk} = \langle E^2, k\rangle$. As usual, we can identify the universe of a structure $\mathcal{A} \in \text{STRUC}[\tau_{gk}]$ with the set $\{0, 1, \ldots, n-1\}$, where $n = \|\mathcal{A}\|$ is the number of vertices of $\mathcal{A}$, and the constant k thus represents not only a vertex, but a number between 0 and $n-1$. (We will see later that CLIQUE is an NP-complete problem.)

Define the query $\text{MAX-CLIQUE}(G)$ to be the size of a largest clique in graph G. We show that the boolean version of MAX-CLIQUE is polynomial-time Turing reducible to CLIQUE. In symbols, this would be written,

$$\text{MAX-CLIQUE}_b \leq_{\text{p}}^{T} \text{CLIQUE} .$$

Where MAX-CLIQUE_b is defined as in Equation (2.6),

$$\text{MAX-CLIQUE}_b \;=\; \big\{(G, i, a) \;\big|\; \text{bit } i \text{ of } \text{MAX-CLIQUE}(G) \text{ is "a"}\big\} .$$

The reduction is as follows. Given (G, i, a), perform binary search using an oracle for CLIQUE to determine the size s of the maximum clique for G. This is done by asking if $(G, n/2) \in$ CLIQUE. If yes, ask about $(G, 3n/4)$, if no, ask about $(G, n/4)$. After $\log n$ queries to the oracle, s has been computed. Now accept iff bit i of s is "a". □

A simpler and more popular kind of reduction in complexity theory is the many-one reduction. (In descriptive complexity, we use first-order reductions. These are first-order queries that are at the same time many-one reductions.)

Definition 2.11 (Many-One Reduction) Let $\mathcal{C}$ be a complexity class, and let $A \subseteq \text{STRUC}[\sigma]$ and $B \subseteq \text{STRUC}[\tau]$ be boolean queries. Suppose that the query $I : \text{STRUC}[\sigma] \to \text{STRUC}[\tau]$ is an element of $Q(\mathcal{C})$ with the property that for all $\mathcal{A} \in \text{STRUC}[\sigma]$,

$$\mathcal{A} \in A \qquad \Leftrightarrow \qquad I(\mathcal{A}) \in B$$

Then I is a *$\mathcal{C}$-many-one reduction* from A to B. We say that A is *$\mathcal{C}$-many-one reducible* to B, in symbols, $A \leq_{\mathcal{C}} B$. For example, when I is a first-order query (Definition 1.26), it is a first-order reduction ($\leq_{\text{fo}}$), when $I \in Q(\text{L})$, it is a logspace reduction ($\leq_{\text{log}}$); and when $I \in Q(\text{P})$, it is a polynomial-time reduction ($\leq_{\text{p}}$). $\square$

Observe that a many-one reduction is a particularly simple kind of Turing reduction: To decide whether $\mathcal{A}$ is an element of A, compute $I(\mathcal{A})$ and ask the oracle whether $I(\mathcal{A})$ is an element of B. Many-one reductions are simpler than Turing reductions, and they seem to suffice in most situations.

Example 2.12 We give a first-order reduction from PARITY to MULT$_b$. PARITY is the boolean query on binary strings that is true iff the string has an odd number of ones. We will see later that PARITY is not first-order (Theorem 13.1). MULT, the multiplication query, maps a pair of boolean strings of length n to their product: a boolean string of length $2n$. Let $\tau_{ab} = \langle A^1, B^1 \rangle$ be the vocabulary of structures that are a pair A, B of boolean strings. Then MULT: $\text{STRUC}[\tau_{ab}] \to \text{STRUC}[\tau_s]$. Since reductions map boolean queries to boolean queries, we actually deal with the boolean version of MULT. MULT$_b$ is a boolean query on structures of vocabulary $\tau_{abcd} = \langle A^1, B^1, c, d \rangle$ that is true iff bit c of the product of A and B is "d".

Recall that $\tau_s = \langle S^1 \rangle$ is the vocabulary of boolean strings. The first-order reduction $I_{PM} : \text{STRUC}[\tau_s] \to \text{STRUC}[\tau_{abcd}]$ is given by the following formulas:

$$\begin{aligned}
\varphi_A(x,y) &\equiv y = \mathit{max} \;\wedge\; S(x) \\
\varphi_B(x,y) &\equiv y = \mathit{max} \\
I_{PM} &\equiv \lambda_{xy}\langle \textbf{true}, \varphi_A, \varphi_B, \langle 0, \mathit{max}\rangle, \langle 0, 1\rangle\rangle
\end{aligned}$$

Observe that the effect of this reduction is to line up all the bits of string $\mathcal{A}$ into column $n-1$ of the generated product. (See Figure 2.13.) It follows that

$$\mathcal{A} \in \text{ PARITY} \qquad \Leftrightarrow \qquad I_{PM}(\mathcal{A}) \in \text{ MULT}_b$$

as desired. Thus, PARITY $\leq_{\text{fo}}$ MULT$_b$. It follows that if MULT were first-order, then PARITY would be as well. We will see later that PARITY is not first-order (Theorem 13.1), so we can conclude that MULT is not first order. $\square$

$$\begin{array}{llccccccccccccc}
 & & \multicolumn{4}{c}{\overbrace{\hspace{6em}}^{n}} & \multicolumn{4}{c}{\overbrace{\hspace{6em}}^{n}} & & \multicolumn{4}{c}{\overbrace{\hspace{6em}}^{n}} \\
A & & 0 & \dots & 0 & s_0 & 0 & \dots & 0 & s_1 & \dots & 0 & \dots & 0 & s_{n-1} \\
B & \times & 0 & \dots & 0 & 1 & 0 & \dots & 0 & 1 & \dots & 0 & \dots & 0 & 1 \\
\hline
 & & 0 & \dots & 0 & s_0 & 0 & \dots & 0 & & \dots & & & & \\
 & & 0 & \dots & 0 & s_1 & 0 & \dots & 0 & & \dots & & & & \\
 & & 0 & \dots & 0 & \vdots & 0 & \dots & 0 & & \dots & & & & \\
 & & 0 & \dots & 0 & s_{n-1} & 0 & \dots & 0 & & \dots & & & & \\
\hline
 & & & \dots & & \boxed{P} & & & & & \dots & & & &
\end{array}$$

Figure 2.13: First-order reduction of PARITY to MULT

When $\mathcal{C}$ is a weak complexity class such as FO or L, the intuitive meaning of $A \le_{\mathcal{C}} B$ is that the complexity of problem A is less than or equal to the complexity of problem B. The intuitive meaning of A being complete for $\mathcal{C}$ is that A is a hardest query in $\mathcal{C}$ and in fact every query in $\mathcal{C}$ can be rephrased as an instance of A; more precisely,

Definition 2.14 (Completeness for a Complexity Class) Let A be a boolean query, let $\mathcal{C}$ be a complexity class, and let $\le_r$ be a reducibility relation. We say that A is *complete for* $\mathcal{C}$ *via* $\le_r$ iff

1. $A \in \mathcal{C}$, and,
2. for all $B \in \mathcal{C}$, $B \le_r A$.

In this book, when we say that a problem is complete for a complexity class and we do not say under what reduction, then we mean via first-order reductions $\le_{\text{fo}}$. It will follow from Theorem 3.1 that if a problem is complete via first-order reductions, then it is complete via logspace and polynomial-time reductions. □

For reasons that are not well understood, naturally occurring problems tend to be complete for important complexity classes such as P, NP, and NL. Completeness was originally defined via reductions such as polynomial-time many-one reductions ($\le_{\text{p}}$) and later, logspace reductions ($\le_{\text{log}}$). However, problems complete via $\le_{\text{p}}$ and $\le_{\text{log}}$ tend to remain complete via $\le_{\text{fo}}$.

Most natural complexity classes include a large number of interesting complete problems. Here are a few boolean queries that are complete for their respective complexity classes. We state these problems very informally here, just to give the reader an idea. More details on these problems and completeness proofs or references are provided later in the text.

Complete for L**:**

- CYCLE: Given an undirected graph, does it contain a cycle?
- REACH$_d$: Given a directed graph, is there a deterministic path from vertex s to vertex t? (A deterministic path is such that for every edge (u, v) on the path, there is only one edge in the graph from u.)

Complete for NL**:**

- REACH: Given a directed graph, is there a path from vertex s to vertex t?
- 2-SAT: Given a boolean formula in conjunctive normal form with only two literals per clause, is it satisfiable?

Complete for P**:**

- CIRCUIT-VALUE-PROBLEM (CVP): Given an acyclic boolean circuit, with inputs specified, does its output gate have value one?
- NETWORK-FLOW: Given a directed graph, with capacities on its edges, and a value V, is it possible to achieve a steady-state flow of value V through the graph?

Complete for NP**:**

- SAT: Given a boolean formula, is it satisfiable?
- 3-SAT: Given a boolean formula in conjunctive normal form with only three literals per clause, is it satisfiable?
- CLIQUE: Given an undirected graph and a value k, does the graph have a complete subgraph with k vertices?

Complete for PSPACE**:**

- QSAT: Given a quantified boolean formula, is it satisfiable?

- HEX, GEOGRAPHY, GO: Given a position in the generalized versions of the games hex, geography, or go, is there a forced win for the player whose move it is?

Exercise 2.15

1. Show that the relations $\le_{\mathrm{fo}}, \le_{\mathrm{log}}$, and $\le_{\mathrm{p}}$ are transitive.
2. Let $\le_{\mathrm{r}}$ be a transitive, many-one reduction, and let A be complete for complexity class $\mathcal{C}$ via $\le_{\mathrm{r}}$. Let T be any boolean query. Show that T is complete for $\mathcal{C}$ via $\le_{\mathrm{r}}$ iff the following two conditions hold:

$$T \in \mathcal{C} \qquad \text{and} \qquad A \le_{\mathrm{r}} T$$

□

Exercise 2.16

1. Show the the value $\lceil \log n \rceil$ is first-order expressible.
2. Define the majority query as follows:

$$\mathrm{MAJ} \quad = \quad \left\{ \mathcal{A} \in \mathrm{STRUC}[\tau_s] \;\middle|\; \text{string } \mathcal{A} \text{ contains more than } \|\mathcal{A}\|/2 \text{ “1”s} \right\}$$

Modify the reduction in Example 2.12 to show that MAJ $\le_{\mathrm{fo}}$ MULT$_b$.

□

To prove a natural problem complete for a complexity class, one usually reduces another natural complete problem to it as in Exercise 2.15, part (2). Proving that the first natural complete problem is complete is more subtle, and we defer these proofs to later chapters. The following exercise, however, introduces an unnatural but universal kind of complete problem.

Exercise 2.17 Consider the following boolean query over strings from $\{0, 1, \#\}^\star$.

$$U_{\mathrm{time}} \quad = \quad \left\{ M\#w\#^r \;\middle|\; M(w)\!\downarrow \text{ in } r \text{ steps} \right\}$$

Show that U_{time} is complete for P via first-order, many-one reductions ($\le_{\mathrm{fo}}$). Hint: you must show that $U_{\mathrm{time}} \in$ P and that for any problem $B \in$ P, $B \le_{\mathrm{fo}}$

U_{time}. The latter is the easier part in this case: let M_B be a polynomial-time Turing machine that accepts B; then the first-order reduction maps w to $M_B\#w\#^r$, where r is sufficiently large. □

We next describe the problem SAT, which we will see later is NP-complete via first-order reductions (Theorem 7.16).

Example 2.18 SAT is the set of boolean formulas in conjunctive normal form (CNF) that admit a satisfying assignment, i.e., a way to set each boolean variable to **true** or **false** so that the whole formula evaluates to **true**. For example, consider the following boolean formulas:

$$\begin{array}{rcl}\varphi_0 & = & (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee x_4) \wedge (x_2 \vee \overline{x_3} \vee x_5) \\ \varphi_1 & = & (x_1 \vee \overline{x_2} \vee x_3) \wedge (x_1 \vee x_2 \vee \overline{x_3}) \wedge (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee \overline{x_3}) \\ & & \wedge (\overline{x_1} \vee \overline{x_4} \vee x_5) \wedge (\overline{x_1} \vee x_4 \vee \overline{x_5}) \wedge (\overline{x_1} \vee x_4 \vee x_5) \wedge (\overline{x_1} \vee \overline{x_4} \vee \overline{x_5})\end{array}$$

Here the notation $\overline{v}$ means $\neg v$. The reader should verify that $\varphi_0 \in$ SAT and $\varphi_1 \notin$ SAT.

It is easy to see that SAT is in NP. In linear time, a nondeterministic algorithm can write down a "0" or a "1" for each boolean variable. Then it can deterministically check that each clause has been assigned at least one "1", and if so, accept.

A boolean formula φ that is in CNF may be thought of as a set of clauses, each of which is a disjunction of literals. Recall that a literal is an atomic formula — in this case a boolean variable — or its negation. Thus, a natural way to encode φ is via the structure $\mathcal{A}_\varphi = \langle A, P, N\rangle$.

The universe A is a set of clauses and variables. The relation $P(c, v)$ means that variable v occurs positively in clause c and $N(c, v)$ means that v occurs negatively in c. We can think of every element of the universe as a variable and a clause. Thus, $n = \|\mathcal{A}_\varphi\|$ is equal to the maximum of the number of variables and the number of clauses occurring in φ. If v is really a variable but not a clause, we can harmlessly make it the clause $(v \vee \overline{v})$ by adding the pair (v, v) to the relations P and N. For example, a structure coding φ_0 in this way is:

$$\begin{array}{rcl}\mathcal{A}_{\varphi_0} & = & \langle\{1,2,3,4,5\}, P, N\rangle \\ P & = & \{(1,1),(1,3),(2,4),(3,2),(3,5),(4,4),(5,5)\} \\ N & = & \{(1,2),(2,1),(2,2),(3,3),(4,4),(5,5)\}\end{array}$$

□

We next show that SAT $\leq_{\text{fo}}$ CLIQUE. It then follows from Theorem 7.16 that CLIQUE is also NP-complete.

Example 2.19 We show that SAT is first-order reducible to CLIQUE. Let $\mathcal{A}$ be a boolean formula in CNF with clauses $C = \{c_1, \ldots, c_n\}$ and variables $V = \{v_1, \ldots, v_n\}$.

Let $L = \{v_1, \ldots, v_n, \overline{v}_1, \ldots, \overline{v}_n\}$. Define the instance of the clique problem $g(\mathcal{A}) = (V^{g(\mathcal{A})}, E^{g(\mathcal{A})}, k)$ as follows:

$$\begin{array}{rcl} V^{g(\mathcal{A})} & = & (C \times L) \cup \{w_0\} \\ E^{g(\mathcal{A})} & = & \big\{(\langle c_1, \ell_1\rangle, \langle c_2, \ell_2\rangle) \bigm| c_1 \neq c_2 \text{ and } \overline{\ell}_1 \neq \ell_2\big\} \cup \\ & & \big\{(w_0, \langle c, \ell\rangle), (\langle c, \ell\rangle, w_0) \bigm| \ell \text{ occurs in } c\big\} \\ k^{g(\mathcal{A})} & = & n+1 \quad = \quad \|\mathcal{A}\| + 1 \end{array} \tag{2.20}$$

The graph $g(\mathcal{A})$ is an $n \times n$ array of vertices containing a row for every clause in $\mathcal{A}$ and a column for every literal in L, plus a top vertex w_o. (See Figure 2.21). There are edges between vertices $\langle c_1, \ell_1\rangle$ and $\langle c_2, \ell_2\rangle$ iff $c_1 \neq c_2$, i.e., the points come from different clauses; and, $\overline{\ell}_1 \neq \ell_2$, i.e., literals ℓ_1 and ℓ_2 are not the negations of each other. The other edges in the graph are between w_0 and those $\langle c, \ell\rangle$ such that literal ℓ occurs in clause c.

Observe that a clique of size $n+1$ must involve w_0 and one vertex from each clause. This corresponds to a satisfying assignment, because no literal and its negation can be in a clique. Conversely, any satisfying assignment to $\mathcal{A}$ determines an $n+1$-clique consisting of w_0 together with one literal per clause that is assigned "true". It follows that mapping g is indeed a many-one reduction,

$$(\mathcal{A} \in \text{SAT}) \quad \Leftrightarrow \quad (g(\mathcal{A}) \in \text{CLIQUE}) \tag{2.22}$$

We now give the rather technical details of writing g as a first-order query, $g = \lambda_{x^1x^2x^3y^1y^2y^3}\langle \varphi_0, \varphi_1, \psi_1\rangle$. We encode the vertices as triples $\langle x^1, x^2, x^3\rangle$, where x^1 corresponds to the clause, x^2 to the variable, and $x^3 = 1$ means the variable is positive and $x^3 = 2$ means the variable is negative. Vertex w_0 is $\langle 1, 1, 3\rangle$, the only triple with $x^3 > 2$. See Figure 2.21, which shows part of $g(\mathcal{A})$ for a formula $\mathcal{A}$ whose first clause is $(x_1 \vee \overline{x_2} \vee \overline{x_n})$. The numeric formula φ_0, which describes the universe of $g(\mathcal{A})$, is the following,

$$\varphi_0 \quad \equiv \quad (x^3 \leq 2) \ \vee \ (x^1x^2x^3 = 113)$$

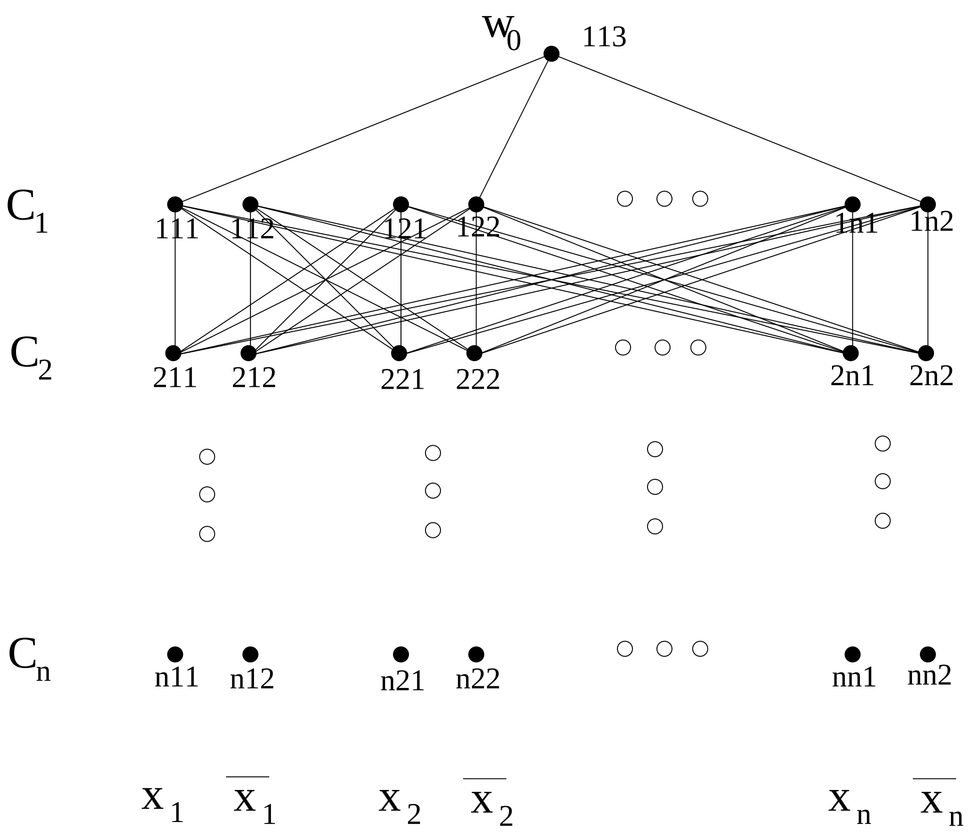

Figure 2.21: SAT $\le_{\text{fo}}$ CLIQUE, $\quad C_1 = (x_1 \lor \overline{x_2} \lor \overline{x_n})$

To define the edge relation, let

$$\begin{aligned}
\varphi_1'(\bar{x},\bar{y}) &\equiv \alpha_1 \ \vee\ (\alpha_2 \wedge P(y^1,y^2)) \ \vee\ (\alpha_3 \wedge N(y^1,y^2))\\
\alpha_1 &\equiv x^1 \neq y^1 \ \wedge\ x^3 < 3 \wedge y^3 < 3 \ \wedge\ (x^2 = y^2 \to x^3 = y^3)\\
\alpha_2 &\equiv x^3 = 3 \ \wedge\ y^3 = 1\\
\alpha_3 &\equiv x^3 = 3 \ \wedge\ y^3 = 2
\end{aligned}$$

Next, let φ_1 be the symmetric closure of φ_1',

$$\varphi_1(x^1,x^2,x^3,y^1,y^2,y^3) \quad\equiv\quad \varphi_1'(x^1,x^2,x^3,y^1,y^2,y^3) \vee \varphi_1'(y^1,y^2,y^3,x^1,x^2,x^3)$$

Notice that φ_1 is a direct translation of Equation (2.20). We are thinking of the elements of the ordered universe as $1, 2, \ldots, n$ instead of the usual $0, 1, \ldots n-1$. For this reason, the number $n+1$ which would usually by 011 in lexicographic order, is instead 122. Formula ψ_1 identifies k as $n+1$:

$$\psi_1(x^1,x^2,x^3) \quad\equiv\quad x^1x^2x^3 = 122$$

It follows that we have correctly encoded the desired first-order reduction g, and Equation (2.22) holds as desired. □

Exercise 2.23 Construct sets S and T such that S is polynomial-time, Turing reducible to T, but S is not polynomial-time, many-one reducible to T, i.e., $S \leq_{\mathrm{p}}^{T} T$, $S \not\leq_{\mathrm{p}} T$.

[Hint: construct S and T systematically so that for all w, $w \in S$ iff $(0w \in T) \oplus (1w \in T)$ and systematically satisfy the conditions,

$$C_i \ \equiv \left(\begin{array}{c}\text{Turing machine } M_i \text{ running in time } i \cdot n^i \text{ does}\\ \text{not compute a many-one reduction from } S \text{ to } T.\end{array}\right)$$

If you are careful, you can carry out this construction so that S and T are in ETIME = DTIME$[2^{O(n)}]$.] □

2.4 Alternation

The concept of a nondeterministic acceptor of a boolean query has a long and rich history, going back to various kinds of nondeterministic automata. On the other hand, it is important to remember that these are fictitious machines which cannot be built. We elaborate on the unreasonableness of nondeterministic machines.

For $A \subseteq \text{STRUC}[\tau]$ a boolean query, define its complement $\overline{A} = \text{STRUC}[\tau] - A$. Given a complexity class $\mathcal{C}$, one can define the complementary class as follows,

$$\text{co-}\mathcal{C} \quad = \quad \{\overline{A} \mid A \in \mathcal{C}\}$$

For example, since SAT is in NP, its complementary problem $\overline{\text{SAT}}$ = UN-SAT is in co-NP. The question whether NP is closed under complementation, i.e., is NP equal to co-NP? is open. Most people believe that these classes are different. Notice that if one could really build an NP machine, then one could also build a co-NP machine: All that is needed is a single gate to invert the former machine's answer. Thus from a very practical point of view, the complexity of a problem A and its complement, $\overline{A}$ are identical.

One way to imagine a realization of an NP machine is via a parallel or biological machine with many processors. At each step, each processor p_i creates two copies of itself and sets them to work on two slightly different problems. If either of these offspring ever accepts, i.e., says "yes" to p_i, then p_i in turn says "yes" to its parent. These "yes"es travel up a binary tree to the root and the whole nondeterministic process accepts. In such a view of nondeterminism, in time $t(n)$ we can build about $2^{t(n)}$ processors. However, we do not make very good use of them. Their pattern of communication is very weak. Each processor can compute only the "or" of its children. Thus, the whole computation is one big "or" of its leaves.

There is a natural way to generalize the concept of nondeterminism so that it is closed under complementation and makes better use of its processors. Namely, we can let the processors compute either the "or" or the "and" of their children. This leads to the notion of *alternation*. An alternating Turing machine has both "or" states like a nondeterministic Turing machine and "and" states.

We now formally define and study alternating machines. We will see that in many ways the concept of alternation is more robust than the concept of nondeterminism.

Definition 2.24 An *alternating Turing machine* is a Turing machine whose states are divided into two groups: the existential states and the universal states. Recall that a *configuration* of any Turing machine — also called an *instantaneous description* (ID) — consists of the machine's state, work-tape contents, and head positions. The notion of when such a machine accepts an input is defined by induction: The alternating Turing machine in a given configuration *c accepts* iff

1. c is in a final accepting state, or
2. c is in an existential state and there exists a next configuration c' that accepts, or
3. c is in a universal state, there is at least one next configuration, and all next configurations accept.

Note that this is a generalization of the notion of acceptance for a nondeterministic Turing machine, which is an alternating Turing machine all of whose states are existential.

Turing machines have an awkward way of accessing their tapes, which makes it difficult for them to do anything in sublinear time. Since alternating Turing machines can sensibly use sublinear time, it is more reasonable to use machines that have a more random access nature. As a compromise, from now on we assume that our Turing machines have a *random access* read-only input. This works as follows: there is an *index tape*, which can be written and read like other tapes. Whenever the value v, written in binary, appears on the index tape, the read head automatically scans bit v of the input. $\square$

Define the complexity classes ASPACE$[s(n)]$ and ATIME$[t(n)]$ to be the set of boolean queries accepted by alternating Turing machines using a maximum of $O(s(n))$ tape cells, respectively a maximum of $O(t(n))$ time steps in any computation path on an input of length n. The main relationships between alternating and deterministic complexity are given by the following theorem.

Theorem 2.25 *For $s(n) \geq \log n$, and for $t(n) \geq n$,*

$$\begin{aligned}\bigcup_{k=1}^{\infty} \text{ATIME}[(t(n))^k] &= \bigcup_{k=1}^{\infty} \text{DSPACE}[(t(n))^k] \\ \text{ASPACE}[s(n)] &= \bigcup_{k=1}^{\infty} \text{DTIME}[k^{s(n)}]\end{aligned}$$

In particular, ASPACE$[\log n]$ = *P, and alternating polynomial time is equal to* PSPACE.

Figure 2.26 shows the computation graph of an alternating machine. We assume for convenience that such machines have a unique accepting and a unique rejecting configuration and that each configuration has at most two possible next

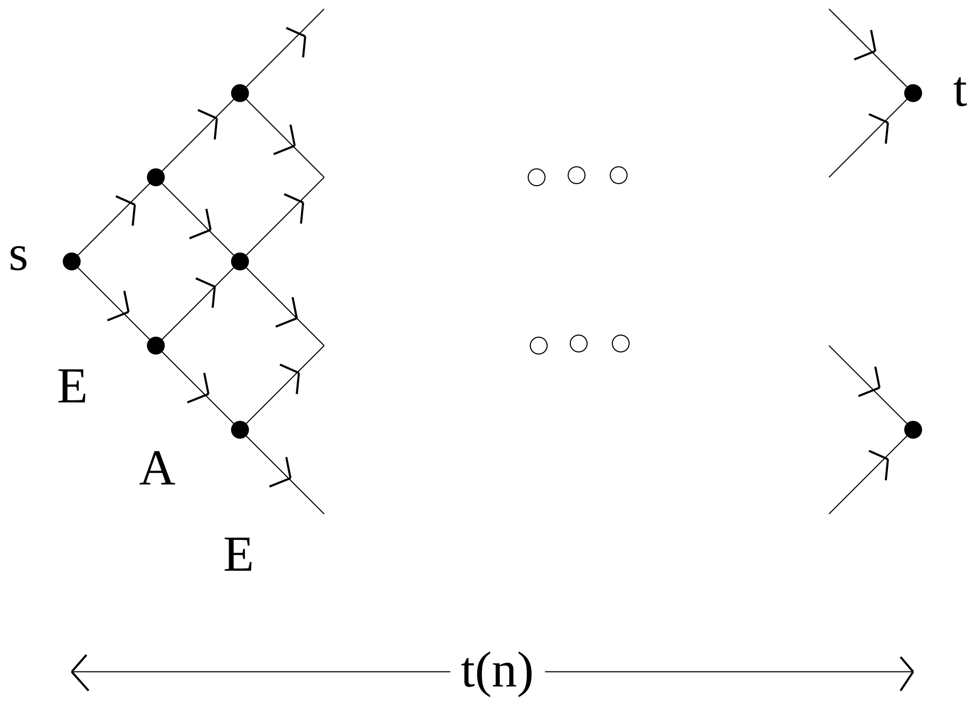

Figure 2.26: Computation graph of an alternating Turing machine

moves. The start configuration is labeled "s" and the accept configuration is labeled "t". We also assume that these machines have clocks that uniformly cause the machines to shut off at a fixed time that is a function of the length of the input. Shutting off means entering the reject configuration unless the machine is already in the accept configuration.

Observe Figure 2.26. The letters "E" and "A" below the vertices indicate whether the corresponding configurations are existential or universal. If they were all existential, then this would be a nondeterministic computation. The time $t(n)$ measures the depth of the computation graph. It is convenient to think of alternating Turing machines as a parallel model in which at each branching move an extra processor is created, and these two processors take the two branches. Eventually these two processors complete their tasks and report their answers to their parent. If the parent was existential, then it reports "accept" iff either of its children accept. If the parent is universal, then it reports "accept" iff both of its children accept. Notice that in time $t(n)$ potentially $2^{O(t(n))}$ processors are created. For purely nondeterministic machines these processors have a very poor system of communication: each parent can perform only the "or" of its children. The ability to perform "and"s as well as "or"s lets alternating machines make more extensive use of their extra processors. We will see in Chapter 4 that alternating time $t(n)$ corresponds to a reasonable notion of parallel time $t(n)$ when $2^{O(t(n))}$ processors are available. As we will see, we can allow at most polynomially many processors by restricting the alternating machines to use no more than logarithmic space. The space used by an alternating machine is the maximum amount of space used in any path through its computation graph.

Before we prove some simulation results concerning alternating Turing machines, we give some examples of their use. The first example involves the circuit value problem (CVP). We will see in Exercise 3.28 that CVP is complete for P.

Definition 2.27 A *boolean circuit* is a directed, acyclic graph (DAG),

$$C = (V, E, G_\wedge, G_\vee, G_\neg, I, r) \in \text{STRUC}[\tau_c]; \quad \tau_c = \langle E^2, G^1_\wedge, G^1_\vee, G^1_\neg, I^1, r \rangle$$

Internal node w is an and-gate if $G_\wedge(w)$ holds, an or-gate if $G_\vee(w)$ holds, and a not-gate if $G_\neg(w)$ holds. The nodes v with no edges entering them are called leaves, and the input relation $I(v)$ represents the fact that the leaf v is on. Often we will be given a circuit C, and separately we will be given its input relation I.

Define the Circuit Value Problem (CVP) to consist of those circuits whose root gate r evaluates to one. Define the monotone, circuit value problem (MCVP) to be the subset of CVP in which no negation gates occur. □

Exercise 2.28 Show that a boolean circuit can be evaluated in linear time.

[Hint: do this bottom up: from the leaves to the root. Each edge should be processed only once. By "linear time" we mean time $O(n + m)$ on a random-access machine (RAM). We assume that the edges of the input circuit are given as an adjacency list; m is the number of edges. Such machines correspond better to real computers than do multi-tape Turing machines. There is a polynomial-size memory and an $O(\log n)$-bit word size. Here we assume that the edges of the input circuit are given as and adjacency list, m is the number of edges.] □

Proposition 2.29 MCVP *is recognizable in* ASPACE$[\log n]$.

Proof Let G be a monotone boolean circuit as in Definition 2.27. Define the procedure "EVAL(a)", where a is a vertex of G, as follows:

1. **if** $I(a)$ **then** accept

```
2.         else if $a$ has no outgoing edges  then reject
3.  if $G_\wedge(a)$  then in a universal state choose a child $b$ of $a$
4.         else in an existential state choose a child $b$ of $a$
5.  Return (EVAL($b$))
```

The machine M simply calls EVAL(r). Observe that EVAL(a) returns "accept" iff gate a evaluates to one. Furthermore, the space used by EVAL is just the space to name two vertices a, b. Thus, M is an ASPACE[$\log n$] machine accepting MCVP, as desired. Observe that the alternating time required for this computation is the depth of circuit G — the length of the longest path in G starting at r. Recall that we have said that all alternating machines have timers. In this case, an appropriate time limit would be $n = \|G\|$, which is an upper bound on the length of the longest path. □

Another boolean query that is well suited for alternating computation is the quantified satisfiability problem:

Definition 2.30 The *quantified satisfiability problem* (QSAT) is the set of true formulas of the following form:

$$\Psi \quad = \quad (Q_1 x_1)(Q_2 x_2) \cdots (Q_r x_r)\varphi$$

where φ is a boolean formula and each Q_i's is each either $\forall$ or $\exists$, and $x_1, \ldots x_r$ are the boolean variables occurring in φ. □

Observe that for any boolean formula φ on variables $\overline{x}$,

$$\varphi \in \mathrm{SAT} \quad \Leftrightarrow (\exists \overline{x})\varphi \in \mathrm{QSAT} \quad \text{and} \quad \varphi \notin \mathrm{SAT} \quad \Leftrightarrow (\forall \overline{x})\neg\varphi \in \mathrm{QSAT}$$

Thus QSAT logically contains SAT and $\overline{\mathrm{SAT}}$.

Proposition 2.31 QSAT *is recognizable in* ATIME*[n].*

Proof Construct an alternating machine A that works as follows. To evaluate the sentence

$$\Phi \equiv (\exists x_1)(\forall x_2) \cdots (Q_r x_r)\alpha(\overline{x})$$

in an existential state, A writes down a boolean value for x_1, in a universal state it writes a bit for x_2, and so on. Next A must evaluate the quantifier-free boolean formula α on these chosen values. This is especially easy for an alternating machine:

for each "$\wedge$" in α, A universally chooses which side to evaluate and for each "$\vee$", A existentially chooses. Thus A only has to read one of the chosen bits x_i and accept iff it is true and occurs positively, or false and occurs negatively. Observe that A runs in linear time and accepts the sentence Φ iff Φ is true. $\square$

The next theorem explains the relationship between alternating time and deterministic and nondeterministic space.

Theorem 2.32 *Let $s(n) \geq \log n$ be space constructible. Then,*

$$\text{NSPACE}[s(n)] \ \subseteq \ \text{ATIME}[s(n)^2] \ \subseteq \ \text{DSPACE}[s(n)^2]$$

Proof For the first inclusion, let N be an NSPACE$[s(n)]$ Turing machine. Let w be an input to N. Let G_w denote the computation graph of N on input w. Note that N accepts w iff there is a path from s to t in G_w. We construct an ATIME$[s(n)^2]$ machine A that accepts the same language as N. A does this by calling the subroutine, $P(d, x, y)$, which accepts iff there is a path in G_w of length at most 2^d from x to y. For $d > 0$, P is defined as follows:

$$P(d, x, y) \ \equiv \ (\exists z)(P(d-1, x, z) \ \wedge \ P(d-1, z, y))$$

P works by existentially choosing a middle configuration z, universally choosing the first half or the second half, and then checking that the appropriate path of length 2^{d-1} exists. Thus, the time $T(d)$ taken to compute $P(d, x, y)$ is the time to write down a new, middle configuration plus the time to compute $P(d-1, x', y')$. The number of bits in a configuration of G_w is $O(s(n))$ where $n = |w|$. Thus,

$$T(d) \ = \ O(s(n)) + T(d-1) \ = \ O(d \cdot s(n))$$

The length of the maximum useful path in G_w is bounded by the number of configurations of N on input w, i.e. $2^{cs(n)}$ for an appropriate value of c. Thus, on input w, A calls $P(cs(n), s, t)$ and receives its answer in time $O(cs(n)s(n)) = O(s(n)^2)$, as desired.

For the second inclusion, let A be an ATIME$[t(n)]$ machine. On input w, A's computation graph — pictured in Figure 2.26 — has depth $O(t(n))$ and size $2^{O(t(n))}$. A deterministic Turing machine can systematically search this entire and-or graph using space $O(t(n))$. This is done by keeping a string of length $O(t(n))$: $c_1 c_2 \ldots c_r \star \ldots \star$ denoting that we are currently simulating step r of A's computation, having made choices $c_1 \ldots c_r$ on all of the existential and universal branches up until this point. The rest of the simulation will report an **answer** as to whether choices $c_1 \ldots c_r$ will lead to acceptance. This is done as follows:

If one of the following conditions holds:

1. $c_r = 1$, or
2. **answer** = "yes" and step r was existential, or
3. **answer** = "no" and step the r was universal,

then let $c_r = \star$ and report **answer** back to step $r - 1$. Otherwise, set $c_r = 1$ and continue. Note, that we do not have to store intermediate configurations of the simulation because the sequence $c_1 c_2 \ldots c_r \star \ldots \star$ uniquely determines which configuration of A to go to next. □

An immediate corollary is that NSPACE$[s(n)]$ is contained in DSPACE$[s(n)^2]$. This is Savitch's theorem (Theorem 2.32), and it is the best known simulation of nondeterministic space by deterministic space. It is unknown whether equality holds in either or both of the inclusions of Theorem 2.32. Another corollary of Theorem 2.32 is the first part of Theorem 2.25.

We complete the proof of Theorem 2.25 by showing that ASPACE$[s(n)]$ is DTIME$[O(1)^{s(n)}]$. One direction of this is obvious: an ASPACE$[s(n)]$ machine has $O(1)^{s(n)}$ possible configurations. Thus, its entire computation graph is of size $O(1)^{s(n)}$ and thus may be traversed in DTIME$[O(1)^{s(n)}]$. The same traversal algorithm as in the second half of the proof of Theorem 2.32 works in this case.

In the other direction, we are given a DTIME$[k^{s(n)}]$ machine M. Let w be an input to M and let $n = |w|$. We can view M's computation as a $k^{s(n)} \times k^{s(n)}$ table — see Figure 2.34. Cell (t, p) of this table contains the symbol that is in position p of M's tape at time t of the computation. Furthermore, if M's head was at position p at time t, then this cell should also include M's state at time t.

Define an alternating procedure $C(t, p, a)$ that accepts iff the contents of cell p at time t in M's computation on input w is symbol a. $C(0, p, a)$ holds iff a is the correct symbol in position p of M's initial configuration on input w. This means that position 1 contains $\langle q_0, w_1 \rangle$ where q_0 is M's start state, and w_1 is the first symbol of w. Similarly, for $2 \le p \le n$, $C(0, p, a)$ holds iff $a = w_p$.

Inductively, $C(t+1, p, a)$ holds iff the three symbols a_{-1}, a_0, a_1 in tape positions $p-1, p, p+1$ lead to an "a" in position p in one step of M's computation. We denote this symbolically as $(a_{-1}, a_0, a_1) \overset{M}{\to} a$. This condition can be read directly from M's transition table.

$$\begin{aligned} C(t+1, p, a) \quad \equiv \quad & (\exists a_{-1}, a_0, a_1)\Big((a_{-1}, a_0, a_1) \overset{M}{\to} a \quad \wedge \\ & \qquad (\forall i \in \{-1, 0, 1\})(C(t, p+i, a_i))\Big) \end{aligned} \qquad (2.33)$$

	Space								
	1	2		p		n			$T(n)$
Time 0	$\langle q_0, w_1 \rangle$	w_2		$\cdots$		w_n	b	$\cdots$	b
1	w_1	$\langle q_1, w_2 \rangle$		$\cdots$		w_n	b	$\cdots$	b
	$\vdots$	$\vdots$		$\vdots$				$\vdots$	
t			a_{-1}	a_0	a_1				
$t+1$				a					
	$\vdots$	$\vdots$		$\vdots$				$\vdots$	
$T(n)$	$\langle q_f, 0 \rangle$	$\cdots$		$\cdots$				$\cdots$	

Figure 2.34: A DTIME[T(n)] computation on input $w_1 w_2 \cdots w_n$

See Figure 2.34 which shows values of $C(t, p, \star)$ for a DTIME[$T(n)$] Turing machine. In the present case, $T(n) = k^{s(n)}$.

Formula 2.33 can be evaluated by an alternating machine using the space to hold the values of t and p. This is $O(\log k^{s(n)})$ = $O(s(n))$. Note that M accepts w iff $C(k^{s(n)}, 1, a_f)$ holds, where a_f is the contents of the first cell of M's accept configuration. For example, we can use $a_f = \langle q_f, 0 \rangle$, where q_f is M's accept state.

This completes the proof of Theorem 2.25.

2.5 Simultaneous Resource Classes

Let the classes ASPACE-TIME[$s(n), t(n)$] (resp. ATIME-ALT[$t(n), a(n)$]) be the sets of boolean queries accepted by alternating machines simultaneously using space $s(n)$ and time $t(n)$ (resp. time $t(n)$ and making at most $a(n)$ alternations between existential and universal states, and starting with existential. Thus ATIME-ALT[$n^{O(1)}, 1$] $=$ NP). Two more important complexity classes may now be defined using these simultaneous alternating classes. Define the polynomial-time hierarchy (PH) to be the set of boolean queries accepted in polynomial time by alternating Turing machines making a bounded number of alternations between existential and universal states:

$$\text{PH} \quad = \quad \bigcup_{k=1}^{\infty} \text{ATIME-ALT}[n^k, k] \; . \tag{2.35}$$

Also define NC (Nick's Class) to be the set of boolean queries accepted by alternating Turing machines in $\log n$ space and poly log time:

$$\mathrm{NC} \quad = \quad \bigcup_{k=1}^{\infty} \text{ASPACE-TIME}[\log n, \log^k n] \qquad (2.36)$$

See Theorem 5.33 for the more usual definition of NC as the class of boolean queries accepted by a parallel random access machine using polynomially much hardware in poly log parallel time.

2.6 Summary

We conclude this section with a list of the complexity classes defined so far. These will be a main focus for much of what follows:

$$\mathrm{L} \subseteq \mathrm{NL} \subseteq \mathrm{NC} \subseteq \mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PH} \subseteq \mathrm{PSPACE} \qquad (2.37)$$

The above containments are easy to prove. (It is a good exercise for the reader to now show how to simulate each of these complexity classes by the next larger one.)

On the other hand, despite intense effort, there is very little known about the strictness of the above inclusions. It has not yet been proved that L is not equal to PH, or that P is not equal to PSPACE. It would probably be safe to assume the strictness of each of the inequalities in Equation (2.37) as an axiom and go on with the rest of one's life.

The fact that we cannot prove these inequalities reveals just the tip of the iceberg of what we do not know concerning the computational complexity of important computational problems. As an example, for the thousands of known NP complete problems, the best known algorithms to get an exact solution are all exponential time in the worst case. However, we have no proof that these are not computable in linear time.

See Figure 2.38 for a view of the computability and complexity world. The classes in the diagram that have not yet been defined will be described later in the text. The intuitive idea behind this diagram is that there is a set of boolean queries called "truly feasible". These are the queries that can be computed exactly with an "affordable" amount of time and hardware, on all "reasonably sized" instances.

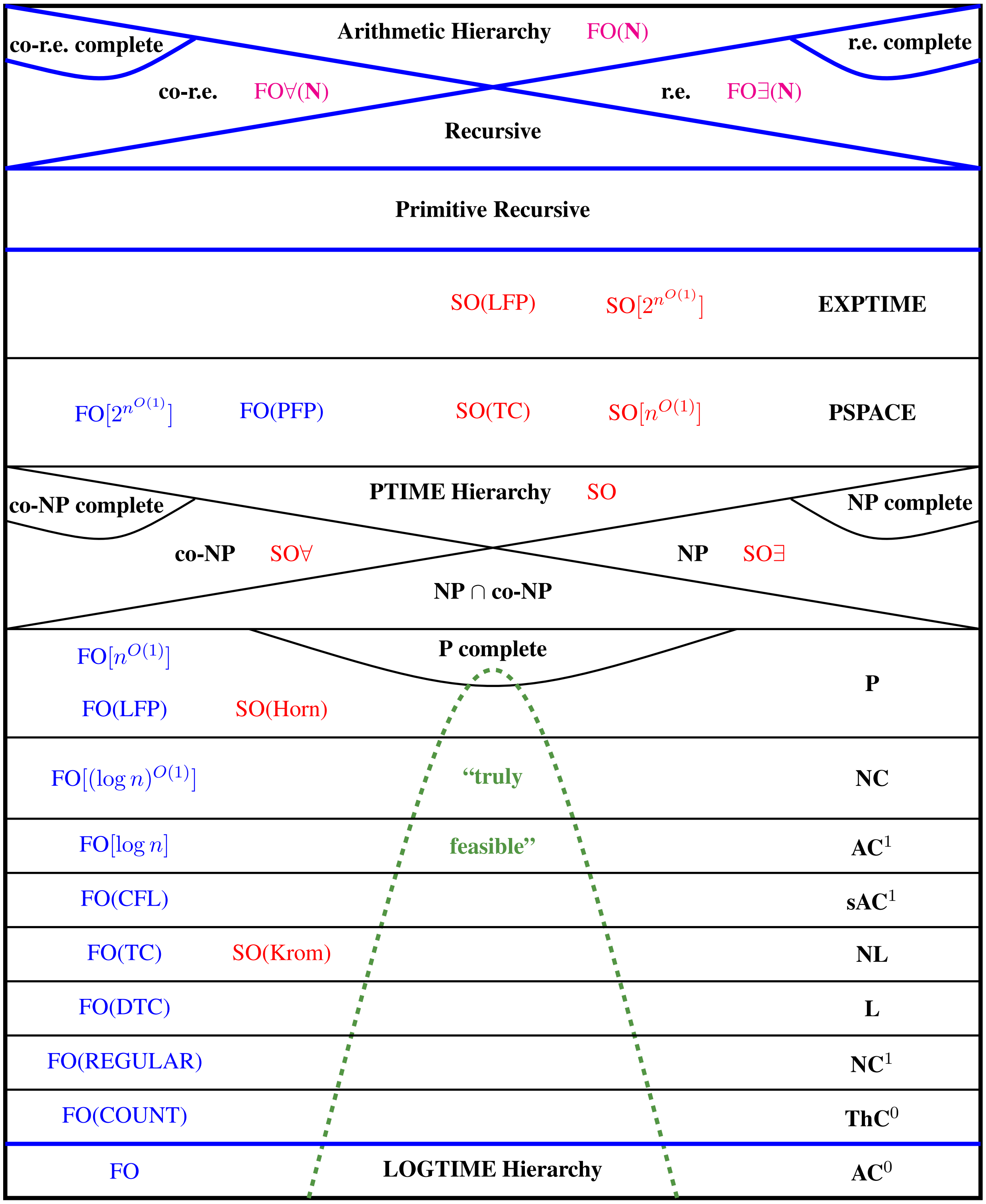

Figure 2.38: The World of Descriptive and Computational Complexity.

The truly feasible queries are a proper subset of P.[2] Many important problems that we need to solve are not truly feasible. For example, they may be NP-complete, EXPTIME-complete, or even r.e.-complete. The theory of algorithms and complexity helps us determine whether the problem we need to solve is feasible, and if not, how to choose a limited set of instances of the problem — or easier versions of them — that are feasible.

As the reader will learn, the theory of complexity via Turing machines is isomorphic to descriptive complexity, i.e., the theory of complexity via logic formulas. We will give descriptive characterizations of almost all of the classes in Figure 2.38. For example, the logarithmic-time hierarchy is equal to the set of first-order boolean queries (LH $=$ FO, Theorem 5.30). The polynomial-time hierarchy is the set of second-order boolean queries (PH $=$ SO, Theorem 7.21). The arithmetic hierarchy is an analogous but much larger class defined to be the set of boolean queries that are describable in the first-order theory of the natural numbers.

The descriptive characterizations of complexity classes in this book are all constructive. Efficient algorithms can be automatically translated into efficient descriptions and vice versa. The descriptive approach has added significant new insights and brought new methods to bear on the basic problems in complexity. Descriptive complexity affords an elegant and simple view of complexity. We hope that the reader will use this and perhaps other approaches to add a few more pieces to the puzzle.

Historical Notes and Suggestions for Further Reading

The reader is referred to [Pap94] for much more information on computational complexity. See also [BDG88] for more detail on "structural complexity" and [Str94] for a combination of algebraic, logical, and automata-theoretic approaches to complexity theory.

Chandra and Harel [CH80a] introduced the general notion of queries as in Definition 1.24. We got the idea for using queries as the general paradigm of computation from a lecture by Phokion Kolaitis.

Alternating machines were defined and Theorem 2.25 was proved independently by Kozen and by Chandra and Stockmeyer [CKS81]. Kozen titled his description of these machines, "Parallelism in Turing Machines", which is very apt

[2]Those readers who believe that the class BPP properly contains P should change this sentence to, "... a proper subset of BPP."

(cf. Theorem 5.33); but Chandra and Stockmeyer coined the name "alternating Turing machine".

The QSAT boolean query was so named in [Pap94] because it is a natural generalization of SAT. Previously, it had been called the quantified boolean formula problem (QBF).

Many-one reductions have their name for an historical reason: to contrast them with 1:1 reductions in which the corresponding mapping is a 1:1 function.

Exercise 2.16 was suggested by Jose Balcázar, cf. [CSV84].

Nick's class (NC) was originally defined by Nick Pippenger; see [Coo85].

Chapter 3

First-Order Reductions

First-order reductions are simple translations that have very little computational power of their own. Thus it is surprising that the vast majority of natural complete problems remain complete via first-order reductions. We provide examples of such complete problems for many complexity classes. All the complexity classes and descriptive languages studied in this book are closed under first-order reductions. Once we express a complete problem for some complexity class $\mathcal{C}$ in a language $\mathcal{L}$, it will follow that $\mathcal{L}$ contains all queries computable in $\mathcal{C}$.

3.1 FO $\subseteq$ L

Recall that FO is the set of first-order definable boolean queries (Definition 1.26). It may be expected that the computational complexity of easy-to-describe queries is low. It was very surprising to us at first that descriptive classes like FO are identical to natural complexity classes.

We will see in Chapter 5 that FO is equal to the set of boolean queries computable in constant parallel time. The following theorem will get us started comparing descriptive versus machine characterizations of complexity. We show that first-order definable queries are all computable in logspace. We will see in Chapter 13 that this containment is strict.

Theorem 3.1 *The set of first-order boolean queries is contained in the set of boolean queries computable in deterministic logspace:* FO $\subseteq$ L.

Proof Let $\sigma = \langle R_1^{a_1}, \ldots, R_r^{a_r}, c_1, \ldots, c_s \rangle$. Let $\varphi \in \mathcal{L}(\sigma)$ determine a first-order boolean query, I_φ : STRUC$[\sigma] \to \{0, 1\}$,

$$\varphi \;\equiv\; (\exists x_1)(\forall x_2) \ldots (Q_k x_k)\alpha(\bar{x})$$

where α is quantifier-free. We must construct a logspace Turing machine M such that for all $\mathcal{A} \in$ STRUC$[\sigma]$, $\mathcal{A}$ satisfies φ iff M accepts the binary encoding of $\mathcal{A}$. In symbols,

$$\mathcal{A} \models \varphi \qquad \Leftrightarrow \qquad M(\text{bin}(\mathcal{A}))\downarrow \tag{3.2}$$

We construct the logspace Turing machine M inductively on k, the number of quantifiers occurring in φ. If $k = 0$, then $\varphi = \alpha$ is a quantifier-free sentence. Thus, α is a fixed, finite boolean combination of atomic formulas. The atomic formulas are either occurrences of input relations $R_i(p_1, \ldots, p_{a_i})$ or numeric relations $p_1 = p_2$, $p_1 \le p_2$, or BIT(p_1, p_2), and the p_i's are members of $\{c_1, \ldots, c_s, 0, 1, \mathit{max}\}$. Once we know that M can determine, on input $\mathcal{A}$, whether $\mathcal{A}$ satisfies each of these atomic formulas, M can then determine whether $\mathcal{A} \models \alpha$, by performing the fixed, finite boolean combination using its finite control.

The reader should convince herself that a logspace machine that knows its input is of the form bin$(\mathcal{A})$, for some $\mathcal{A} \in$ STRUC$[\sigma]$, can calculate the values n and $\lceil \log n \rceil$. Then, by counting, the machine can go to the appropriate constants and copy the p_i's that it needs onto its worktape. To calculate one of the input predicates, M can just look up the appropriate bit of its input.

For example, to calculate $R_3(c_2, \mathit{max}, c_1)$, M first copies the values $c_2, n-1, c_1$ to its worktape. Next it moves its read head to location $n^{a_1} + n^{a_2} + 1$, which is the beginning of the array encoding R_3. Finally, it moves its read head $n^2 \cdot c_2 + n \cdot (n-1) + c_1$ spaces to the right. The bit now being read is "1" iff $\mathcal{A} \models R_3(c_2, \mathit{max}, c_1)$.

It is easy to see that a logspace Turing machine may test the numeric predicates. This completes the construction of M in the base case.

Inductively, assume that all first-order queries with $k-1$ quantifiers are logspace computable. Let

$$\psi(x_1) \quad = \quad (\forall x_2) \ldots (Q_k x_k)\alpha(\bar{x}) \; .$$

Let M_0 be the logspace Turing machine that computes the query $\psi(c)$. Note that c is a new constant symbol substituted for the free variable x_1. To compute the query $\varphi \equiv (\exists x_1)(\psi(x_1))$ we build the logspace machine that cycles through all possible values of x_1, substitutes each of these for c, and runs M_0. If any of these lead M_0 to accept, then we accept, else we reject. Note that the extra space needed is just $\log n$ bits to store the possible values of x_1. Simulating a universal quantifier is similar. $\square$

3.2 Dual of a First-Order Query

A first-order query I from STRUC$[\sigma]$ to STRUC$[\tau]$ maps any $\mathcal{A} \in$ STRUC$[\sigma]$ to $I(\mathcal{A}) \in$ STRUC$[\tau]$. It does this by defining the relations of $I(\mathcal{A})$ via first-order formulas. In a similar way, I has a natural dual $\widehat{I}$, which translates any formula in $\mathcal{L}(\tau)$ to a formula in $\mathcal{L}(\sigma)$.

In this section we define and characterize this dual mapping. The dual is useful in showing that relevant languages and complexity classes are closed under first-order reductions.

Let I be a k-ary first-order query. For each formula $\varphi \in \mathcal{L}(\tau)$, the formula $\widehat{I}(\varphi) \in \mathcal{L}(\sigma)$ is constructed as follows: Replace each variable by a k-tuple of variables. Replace each symbol of τ by its definition in I. It follows that the length of $\widehat{I}(\varphi)$ is linear in the length of φ. In the following, we give the details.

Definition 3.3 (Dual of I) Let $I = \lambda_{x_1 \dots x_d} \langle \varphi_0, \dots, \psi_s \rangle$ be a k-ary first-order query from STRUC$[\sigma]$ to STRUC$[\tau]$. Then I also defines a dual map, which we call $\widehat{I} : \mathcal{L}(\tau) \to \mathcal{L}(\sigma)$, as follows:

Let $\tau = \langle R_1^{a_1}, \dots, R_r^{a_r}, c_1, \dots, c_s \rangle$. For $\varphi \in \mathcal{L}(\tau)$, $\widehat{I}(\varphi)$ is the result of replacing all relation and constant symbols in φ by the corresponding formulas in I, using a map f_I defined as follows:

- Each variable is mapped to a k-tuple of variables: $f_I(v) = v^1, \dots, v^k$
- Input relations are replaced by their corresponding formulas:

$$f_I(R_i(v_1, \dots, v_{a_i})) = \varphi_i(f_I(v_1), \dots, f_I(v_{a_i}))$$

- Constant c_i is replaced by a special k-tuple of variables[1]:

$$f_I(c_i) = z_i^1, \dots, z_i^k$$

[1]For many applications, each constant from τ may be replaced by a corresponding k-tuple of constants from σ.

- Quantifiers are replaced by restricted quantifiers:

$$f_I(\exists v) = (\exists f_I(v) \,.\, \varphi_0(f_I(v)))$$

- The equality relation and the other numeric relations are replaced by their appropriate formulas as in Exercise 1.33.
- Second-order quantifiers — which we will need in Chapter 7 — have the arities of the relations being quantified multiplied by k:

$$f_I(\exists R^a) = (\exists R^{ka})$$

- On boolean connectives, f_I is the identity.

The only thing to add is that the variables $z_i^1, \ldots, z_i^k$ corresponding to the constant symbol c_i must be quantified before they are used. It does not matter where these quantifiers go because the values are uniquely defined. Typically, we can place these quantifiers at the beginning of a first-order formula. (For a second-order formula, they would be placed just after the second-order quantifiers.)

Thus, the mapping $\widehat{I}$ is defined as follows, for $\theta \in \mathcal{L}(\tau)$,

$$\widehat{I}(\theta) \;=\; (\exists z_1^1 \ldots z_1^k \,.\, \psi_1(z_1^1 \ldots z_1^k)) \cdots (\exists z_s^1 \ldots z_s^k \,.\, \psi_s(z_s^1 \ldots z_s^k))(f_I(\theta)) \qquad \square$$

Exercise 3.4 To get the idea of what the dual mapping does, consider the query I_{PM} from Example 2.12. Here is one sample value of the map $\widehat{I}_{PM}$:

$$\begin{aligned} \widehat{I}_{PM}(A(c)) &\equiv (\exists z_1 z_2 \,.\, z_1 = 0 \wedge z_2 = \mathit{max})(z_2 = \mathit{max} \wedge S(z_1)) \\ &\equiv S(0) \end{aligned}$$

Compute the value of $\widehat{I}_{PM}$ on the following,

1. $(\forall v)(A(v) \leftrightarrow B(v))$
2. $A(\mathit{max})$
3. $A(0)$ $\square$

It follows immediately from Definition 3.3 that:

Proposition 3.5 *Let σ, τ, and I be as in Definition 3.3. Then for all sentences $\theta \in \mathcal{L}(\tau)$ and all structures $\mathcal{A} \in \text{STRUC}[\sigma]$,*

$$\mathcal{A} \models \widehat{I}(\theta) \qquad \Leftrightarrow \qquad I(\mathcal{A}) \models \theta$$

Remark 3.6 *Proposition 3.5 goes through for formulas with free variables as well. In this case, I must behave appropriately on interpretations of variables. That is, $I(\mathcal{A}, i) = (I(\mathcal{A}), i')$, where $i'(x)$ is defined iff all of $i(x^1), \ldots, i(x^k)$ are defined. In this case, $i'(x) = \langle i(x^1), \ldots, i(x^k)\rangle$.*

In the following exercise you are asked to show that structures of any vocabulary σ may be transformed to graphs via a first-order, invertible query. It then follows from Proposition 3.5 that every formula in $\mathcal{L}(\sigma)$ may be translated into the language of graphs. This is true even without the ordering relation. Exercise 2.3 shows the same thing about binary strings, i.e., that everything can be coded in a first-order way as a binary string; but in the case of strings, ordering and arithmetic are required.

Exercise 3.7 (Everything is a Graph) Let σ be any vocabulary, and let $\tau_e = \langle E^2 \rangle$ be the vocabulary with one binary relation symbol, i.e., the vocabulary of graphs with no specified points. In this exercise, you will show that every structure may be thought of as a graph.

Show that there exist first-order queries $I_\sigma : \text{STRUC}[\sigma] \to \text{STRUC}[\tau_e]$ and $I_\sigma^{-1} : \text{STRUC}[\tau_e] \to \text{STRUC}[\sigma]$ with the following property,

$$\text{for all } \mathcal{A} \in \text{STRUC}[\sigma], \qquad I_\sigma^{-1}(I_\sigma(\mathcal{A})) \cong \mathcal{A} \tag{3.8}$$

[Hint: to build the graph $I_\sigma(\mathcal{A})$, you can construct "gadgets", i.e., small recognizable graphs to label different sorts of vertices, e.g., those corresponding to elements of $|\mathcal{A}|$, those corresponding to tuples from each relation $R_i^{\mathcal{A}}$, etc.] Note that this exercise does not require ordering, which is why Equation (3.8) says only that the two objects are isomorphic. If we include ordering, we can require that equality holds. □

If A and B are boolean queries and A is first-order reducible to B ($A \le_{\text{fo}} B$), then intuitively the complexity of A is not greater than the complexity of B. The following definition makes this intuitive idea explicit.

Definition 3.9 (Closure Under First-Order Reductions) A set of boolean queries $\mathcal{S}$ is *closed under first-order reductions* iff whenever there are boolean queries A

and B such that $B \in \mathcal{S}$ and $A \leq_{\text{fo}} B$, we have that $A \in \mathcal{S}$. We say that a language $\mathcal{L}$ is closed under first-order reductions iff the set of boolean queries definable in $\mathcal{L}$ is closed. $\square$

The following proposition follows immediately from Theorem 3.1.

Proposition 3.10 *Let $\mathcal{S}$ be any set of boolean queries that is closed under logspace reductions. Then $\mathcal{S}$ is also closed under first-order reductions.*

The next proposition cannot be proved immediately. It is a global exercise. It is striking that with the exception of the dynamic-complexity classes, all complexity classes we discuss in the book are closed under first-order reductions. Every time a new language or complexity class is introduced, the reader should check that it is closed under first-order reductions. For languages, we mean that the set of boolean queries definable in that languages is closed under first-order reductions. This is immediate if the language is closed under quantification and boolean operations. Most languages we consider are so closed. Some languages such as SO$\exists$ — see Theorem 7.8 — do not seem to be closed under negation. However, they still allow full use of first-order logic at their bottom levels and are thus closed under first-order reductions.

Meta-Proposition 3.11 *With the exception of the dynamic complexity classes defined in Chapter 14, all the complexity classes $\mathcal{C}$ that we discuss in this book are closed under first-order reductions. All the languages $\mathcal{L}$ that we discuss in this book are closed under first-order reductions.*

Framework 3.12 Here is a method for proving this proposition whenever a new complexity class or logical language is encountered. For complexity classes we can usually use Proposition 3.10 as most complexity classes are closed under logspace reductions.

For languages, let $A \leq_{\text{fo}} B$ be two boolean queries, where B is expressible as the formula φ_B in language $\mathcal{L}$. Let I_{AB} be the first-order reduction from A to B. We know that for all structures $\mathcal{S}$,

$$\mathcal{S} \in A \qquad \Leftrightarrow \qquad I_{AB}(\mathcal{S}) \in B$$

It follows from Proposition 3.5 that

$$\mathcal{S} \in A \qquad \Leftrightarrow \qquad \mathcal{S} \models \widehat{I}_{AB}(\varphi_B)$$

So if $\widehat{I}_{AB}(\varphi_B)$ is in $\mathcal{L}$ then the proof is complete. Since the definition of $\widehat{I}(\varphi)$ (Definition 3.3) is a simple substitution that doesn't change the structure of φ very much, we will find that for the languages we consider $\widehat{I}_{AB}(\varphi_B)$ will be in $\mathcal{L}$ as desired. $\square$

First-order reductions are simple and natural reductions. It is very surprising that they seem to suffice in almost all complexity theoretic settings. We will see that "natural" problems that are complete via polynomial-time reductions for some complexity class tend to remain complete via first-order reductions.

Suppose that we know that a boolean query A is complete via first-order reductions for a complexity class $\mathcal{C}$. Suppose further that A is expressible in a language $\mathcal{L}$ which is closed under first-order reductions. It follows immediately that $\mathcal{L}$ expresses everything in $\mathcal{C}$.

Suppose that $\mathcal{L}$ is a set of boolean queries describable in some language and that $\mathcal{C}$ is a complexity class, that is, a set of boolean queries computable in some complexity bound. In the sequel our paradigm for proving that $\mathcal{L} = \mathcal{C}$ will be the following four steps:

1. Show that $\mathcal{L} \subseteq \mathcal{C}$ by producing for each formula φ from the language an algorithm in $\mathcal{C}$ that computes the boolean query,
$$\text{MOD}[\varphi] \quad = \quad \{\mathcal{A} \mid \mathcal{A} \models \varphi\}$$
2. Produce a boolean query T that is complete for $\mathcal{C}$ via first-order reductions.
3. Show that $\mathcal{L}$ is closed under first-order reductions.
4. Express T in the language, thus showing that $T \in \mathcal{L}$.

A typical example is in a proof of Theorem 7.8, which says that NP $=$ SO$\exists$. We can show: (1) Each SO$\exists$ formula can be checked by an NP machine; (2) The problem SAT is complete for NP via $\leq_{\text{fo}}$; (3) SO$\exists$ is closed under first-order reductions; and finally, (4) SAT is expressible in SO$\exists$. (Actually, in Chapter 7 we present a different proof for Theorem 7.8 that provides more information.)

In the remainder of this chapter, we give several examples of first-order reductions as we produce natural complete problems for the complexity classes L, NL, and P. The proofs encode Turing machine computations using first-order formulas. The proofs are quite intricate. For this reason, it would be fine to skim the remainder of this chapter on first reading. On the other hand, since many of the results on capturing complexity classes by logics depend on this material, at some point this material should be read.

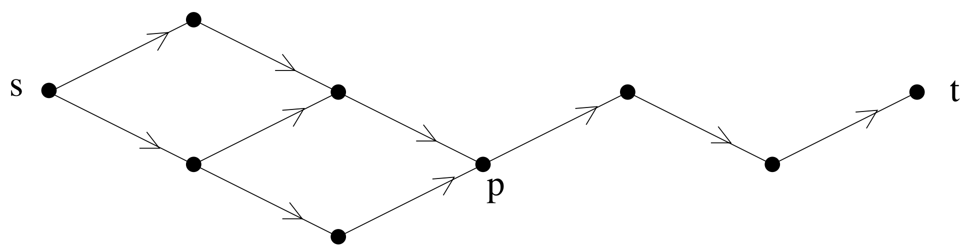

Figure 3.14: A graph that is in REACH but not REACH$_d$

3.3 Complete problems for L and NL

Natural complete problems for L and NL are the REACH$_d$ and REACH problems.

Definition 3.13 Define REACH to be the set of directed graphs G such that there is a path in G from s to t. Define REACH$_d$ to be the subset of REACH such that the path from s to t is deterministic. This means that for each edge (u, x) on the path, this is the unique edge in G leaving u. See Figure 3.14 for a directed graph that is in REACH but not REACH$_d$. Note that there is a directed path in this figure from p to t. Also, define REACH$_u$ — the undirected graph reachability problem — to be the restriction of REACH to undirected graphs,

$$\text{REACH}_u \quad = \quad \left\{ G \in \text{REACH} \;\middle|\; G \models (\forall xy)(E(x,y) \to E(y,x)) \right\} \qquad \Box$$

The following NSPACE[$\log n$] algorithm recognizes REACH. Note that the space used is just the $O(\log n)$ bits needed to name the two vertices a and b.

Algorithm 3.15 Recognizing REACH in NL

```
1. $b := s$
2. while $(b \neq t)$ do {
3.         $a := b$
4.         nondeterministically choose new $b$
5.         if $(\neg E(a, b))$ then reject }
6. accept
```

Theorem 3.16 REACH *is complete for* NL *via first-order reductions.*

Proof Let $S \subseteq \text{STRUC}[\sigma]$ be a boolean query in NL. Let N be the nondeterministic logspace Turing machine that accepts S. We construct a first-order reduction $I : \text{STRUC}[\sigma] \to \text{STRUC}[\tau_g]$ such that for all $\mathcal{A} \in \text{STRUC}[\sigma]$,

$$N(\text{bin}(\mathcal{A}))\downarrow \qquad \Leftrightarrow \qquad I(\mathcal{A}) \in \text{REACH} \tag{3.17}$$

Let c be such that N uses at most $c \log n$ bits of worktape for inputs bin($\mathcal{A}$), with $n = \|\mathcal{A}\|$. Let $\sigma = \langle R_1^{a_1}, \ldots, R_r^{a_r}, c_1, \ldots, c_s \rangle$ and let $a = \max\{a_i \,|\, 1 \le i \le r\}$. Let $k = 1 + a + c$. Consider a run of N on input bin($\mathcal{A}$). We code an instantaneous description (ID) of N's computation as a k-tuple of variables:

$$\text{ID} \quad = \quad (p, r_1, \ldots r_a, w_1, \ldots, w_c)$$

The idea is that variables $r_1, \ldots, r_a$ encode where in one of the input relations the read head of N is looking. If for example it is looking at relation R_i, then,

$$N\text{'s read head is looking at a “1”} \qquad \Leftrightarrow \qquad \mathcal{A} \models R_i(r_1, \ldots, r_{a_i}) \tag{3.18}$$

Variables $w_1, \ldots, w_c$ encode the contents of N's work tape. Remember that each variable represents an element of $\mathcal{A}$'s n-element universe, so it corresponds to a $\log n$-bit number. We are assuming the presence of the numerical relations $\le$ and BIT. Of these, $\le$ is necessary (see Proposition 6.14), but BIT is merely convenient (see Proposition 9.16). Finally, we need $O(\log \log n)$ bits of further information to encode: (1) the state of N, (2) which input relation or constant symbol the read head is currently scanning, and (3) the position of the work head. We assume that n is sufficiently large that all of this information can be encoded into a single variable, p.

Now we start to build the desired k-ary first-order query I and show that it satisfies Equation (3.17). I will be constructed as follows:

$$I \quad = \quad \lambda_{\text{ID,ID}'} \langle \textbf{true}, \varphi_N, \alpha, \omega \rangle$$

where

1. The universe relation being “**true**” indicates that for any $\mathcal{A} \in \text{STRUC}[\sigma]$, the universe of $I(\mathcal{A})$ consists of *all* k-tuples from the universe of $\mathcal{A}$, $|I(\mathcal{A})| = |\mathcal{A}|^k$.
2. $\mathcal{A} \models \varphi_N(\text{ID}, \text{ID}')$ iff (ID, ID') is a valid move of N on input bin($\mathcal{A}$),
3. $\mathcal{A} \models \alpha(\text{ID}_i)$ iff ID_i is the unique initial ID of N, for inputs of size $\|\mathcal{A}\|$, and,

4. $\mathcal{A} \models \omega(\text{ID}_f)$ iff ID_f is the unique accept ID of N for inputs of size $\|\mathcal{A}\|$.

Formulas α and ω are the following,

$$\begin{array}{rcl} \alpha(x_1, \ldots, x_k) & \equiv & x_1 = x_2 = \ldots = x_k = 0 \\ \omega(x_1, \ldots, x_k) & \equiv & x_1 = x_2 = \ldots = x_k = \mathit{max} \end{array} \tag{3.19}$$

Formula φ_N is not hard, but it is more tedious. It is essentially a disjunction over N's finite transition table.

A typical entry in the transition table is $(\langle q, b, w\rangle, \langle q', i_d, w', w_d\rangle)$. This says that in state q, looking at bit b with the input head and bit w with the work head, N may go to state q', move its input head one step in direction i_d, write bit w' on its work tape and move its work head one step in direction w_d. The corresponding disjunct in φ_N must decode the old state from variable p and must decode from p which input relation is being read. Say it is R_i. Then the bit b is “1” iff $R_i(r_1, \ldots, r_{a_i})$ holds. Similarly, we must extract from p the segment j of the work tape that is currently being scanned together with the position s on that worktape. Thus, bit w is “1” iff $\text{BIT}(w_j, s)$ holds.

With these details completed, it now follows that for any $\mathcal{A} \in \text{STRUC}[\sigma]$, $I(\mathcal{A})$ is the computation graph of N on input $\text{bin}(\mathcal{A})$. It follows that N accepts $\text{bin}(\mathcal{A})$ iff there is a path in $I(\mathcal{A})$ from s to t, i.e., Equation (3.17) holds. $\square$

Exercise 3.20 There are several gaps left in the proof of Theorem 3.16 that the reader should now fill in.

1. Using numeric relation BIT, write first-order formulas to uniquely identify elements $l_1 = \lceil \log n \rceil$ and $l_2 = \lceil \log \log n \rceil$ of the universe.

2. Show that since the coding is somewhat arbitrary, we may use Equation (3.19) as our definitions of α and ω.

3. Assume that the first l_2 bits of p encode the work head's position, s. Write a formula to uniquely identify element s.

4. Do the same problem as (3) but this time assume that the bits of s are encoded in the last l_2 bits of p. In order to do this you will need addition, which is available (Theorem 1.17). $\square$

We next show that REACH_d is complete for L via first-order reductions. We first must show that REACH_d is in L. A modification of Algorithm 3.15 recognizes REACH_d in logspace. Since a deterministic path has at most one edge leaving each vertex, nondeterminism is no longer needed. We add a counter to detect cycles:

Algorithm 3.21 Recognizing REACH$_d$ in L

```
1. $b := s;\ i := 0;\ n := \|G\|$
2. **while** $b \neq t \wedge i < n \wedge (\exists! a)(E(b, a)))$ **do** {
3.         $b :=$ the unique $a$ for which $E(b, a)$
4.         $i := i + 1$ }
5. **if** $b = t$ **then** accept **else** reject
```

The definition of REACH$_d$ was made just so that the following theorem would be true:

Theorem 3.22 REACH$_d$ *is complete for* L *via first-order reductions.*

Proof This proof is similar to the proof of Theorem 3.16. In fact, we copy the whole construction with $S \subseteq$ STRUC$[\sigma]$ an arbitrary boolean query from L. The only difference is that now N is a deterministic logspace Turing machine that computes S. Since N is deterministic, for any $\mathcal{A} \in$ STRUC$[\sigma]$, the graph $I(\mathcal{A})$ has at most one edge leaving any vertex. It follows that $I(\mathcal{A})$ is in REACH iff it is in REACH$_d$. Thus,

$$N(\text{bin}(\mathcal{A}))\downarrow \quad \Leftrightarrow \quad I(\mathcal{A}) \in \text{REACH}_d \qquad \square$$

3.4 Complete Problems for P

Alternation provides a nice way to characterize P, namely, P $=$ ASPACE$[\log n]$ (Theorem 2.25). This leads to a natural analogue of the reachability problem that is complete problem for P.

Definition 3.23 Let an *alternating graph* $G = (V, E, A, s, t)$ be a directed graph whose vertices are labeled universal or existential. $A \subseteq V$ is the set of universal vertices. Let $\tau_{ag} = \langle E^2, A^1, s, t\rangle$ be the vocabulary of alternating graphs.

Alternating graphs have a different notion of accessibility. Let $P_a^G(x, y)$ be the smallest relation on vertices of G such that:

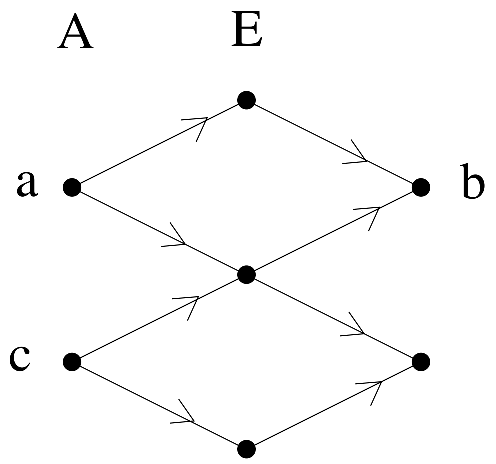

Figure 3.24: An alternating graph with two universal nodes: a, c.

1. $P_a^G(x, x)$
2. If x is existential and $P_a^G(z, y)$ holds for some edge (x, z) then $P_a^G(x, y)$.
3. If x is universal, there is at least one edge leaving x, and $P_a^G(z, y)$ holds for all edges (x, z) then $P_a^G(x, y)$.

See Figure 3.24 where $P_a^G(a, b)$ holds but $P_a^G(c, b)$ does not. Let

$$\text{REACH}_a \quad = \quad \left\{ G \;\middle|\; P_a^G(s, t) \right\} \qquad \square$$

Observe that the following marking algorithm computes REACH_a in linear time.

Algorithm 3.25 Recognizing REACH_a in linear time on a RAM.

```
1. make QUEUE empty; mark($t$); insert $t$ into QUEUE
2. while QUEUE not empty  do {
3.         remove first element, $x$, from QUEUE
4.         for each unmarked vertex $y$ such that $E(y, x)$  do {
5.             delete edge $(y, x)$
6.            if $y$ is existential or $y$ has no outgoing edges
7.            then mark($y$); insert $y$ into QUEUE } }
8. if $s$ is marked then accept else reject
```

Not surprisingly we have,

Theorem 3.26 REACH$_a$ *is complete for* P *via first-order reductions.*

Proof This proof is similar to the proof of Theorem 3.16. Let $S \subseteq \text{STRUC}[\sigma]$ be an arbitrary boolean query. Assume that $S \in \text{P}$ and let T be the alternating, logspace Turing machine that computes S. We construct a first-order reduction $I_a : \text{STRUC}[\sigma] \to \text{STRUC}[\tau_{ag}]$ such that for all $\mathcal{A} \in \text{STRUC}[\sigma]$,

$$T(\text{bin}(\mathcal{A}))\downarrow \qquad \Leftrightarrow \qquad I_a(\mathcal{A}) \in \text{REACH}_a \tag{3.27}$$

Indeed, the only difference between I, the query from the proof of Theorem 3.16, and I_a is that I_a must also describe the relation A that identifies the universal states of T. Assume for simplicity that the universal states are exactly the odd-numbered states. Assume further that the variable p in an ID encodes its state in its low-order bits. Thus the state of an ID is universal iff the corresponding p is odd. This occurs iff BIT$(p, 0)$ holds. Thus, let

$$\psi_A \;=\; \text{BIT}(p, 0), \qquad I \;=\; \lambda_{\text{ID},\text{ID}'}\langle \textbf{true}, \varphi_T, \psi_A, \alpha, \omega\rangle$$

where φ_T, α, and ω are defined exactly as in the proof of Theorem 3.16.

It follows that,

$$T(\text{bin}(\mathcal{A}))\downarrow \qquad \Leftrightarrow \qquad I_a(\mathcal{A}) \in \text{REACH}_a \qquad \square$$

Exercise 3.28 Recall the circuit value problem (CVP) and the monotone circuit value problem (MCVP) from Definition 2.27.

1. Produce a first-order reduction from REACH$_a$ to MCVP.
2. Conclude that MCVP is complete for P via first-order reductions.
3. Conclude that CVP is also complete via first-order reductions. Be slightly careful because it is certainly not true that for any two boolean queries S, T, if S is complete for $\mathcal{C}$ and $T \in \mathcal{C}$ and $S \subseteq T$ then T is complete for $\mathcal{C}$. $\square$

Historical Notes and Suggestions for Further Reading

Savitch proved that REACH is complete for NL via logspace reductions, [Sav73]. Hartmanis, Immerman, and Mahaney showed that REACH_d is complete for L via one-way logspace reductions, [HIM78]. The completeness of these problems via first-order reductions was proved in [I83]. In these references, REACH was called "GAP" and REACH_d was called "1GAP".

The complexity of REACH_u is also a rich subject. Aleliunas, Karp, Lipton, Lovász, and Rackoff, proved that taking a random walk in an undirected graph will — with very high probability — quickly reach all reachable vertices [AKL79]. It follows that boolean query REACH_u is computable in "random logspace". Lewis and Papadimitriou define a restriction of nondeterministic space called "symmetric space" and prove that REACH_u is complete via logspace reductions for symmetric logspace [LP82]. Reingold proved in [Rei05] the breakthrough result that REACH¡sub¿u¡/sub¿ is in L, and thus that symmetric logspace is equal to L."

REACH_a was shown to be complete for P via logspace reductions in [I80] and via first-order reductions (and in fact quantifier-free projections) in [I83].

Exercise 3.28 shows that the monotone circuit value problem is complete for P via first-order reductions. The completeness of CVP for P via logspace reductions was first shown by Ladner in [L75]. The completeness of MCVP via logspace reductions was originally shown by Goldschlager in [Go77].

Chapter 7

Second-Order Logic and Fagin's Theorem

Second-order logic consists of first-order logic plus the power to quantify over relations on the universe. We prove Fagin's theorem which says that the queries computable in NP are exactly the second-order existential queries. A corollary due to Stockmeyer says that the second-order queries are exactly those computable in the polynomial-time hierarchy.

7.1 Second-Order Logic

Second-order logic consists of first-order logic plus new relation variables over which we may quantify. For example, the formula $(\forall A^r)\varphi$ means that for all choices of r-ary relation A, φ holds. Let SO be the set of second-order expressible boolean queries.

Any second-order formula may be transformed into an equivalent formula with all second-order quantifiers in front. If all these second-order quantifiers are existential, then we have a second-order existential formula. Let (SO∃) be the set of second-order existential boolean queries. Consider the following example, in which R, Y, and B are unary relation variables. To indicate their arity, we place exponents on relation variables where they are quantified.

$$\begin{aligned}\Phi_{\text{3-color}} \equiv\ & (\exists R^1)(\exists Y^1)(\exists B^1)(\forall x)\Big[\big(R(x) \vee Y(x) \vee B(x)\big) \ \wedge\ (\forall y)\Big(E(x,y) \rightarrow \\ & \neg\big(R(x) \wedge R(y)\big) \ \wedge\ \neg\big(Y(x) \wedge Y(y)\big) \ \wedge\ \neg\big(B(x) \wedge B(y)\big)\Big)\Big]\end{aligned}$$

Observe that a graph G satisfies $\Phi_{\text{3-color}}$ iff G is 3-colorable. Three colorability of graphs is an NP complete problem (3-COLOR). In Section 7.2, we see that three colorability remains complete via first-order reductions. It will then follow that every query computable in NP is describable in SO$\exists$.

Second-order logic is extremely expressive. For this reason, it is very easy to write second-order specifications of queries. For the same reason, such specifications are not feasible to execute without further refinement (cf. Section 9.6). Recall that the first-order queries are those that can be computed on a CRAM in constant time, using polynomially many processors (Theorem 5.2). We will see that the second-order queries are those that can be computed in constant parallel time, but using exponentially many processors (Corollary 7.27).

Here are a few other examples of SO$\exists$ queries.

Example 7.1 SAT is the set of boolean formulas in conjunctive normal form (CNF) that admit a satisfying assignment (Example 2.18).

The boolean query SAT is expressible in SO$\exists$ as follows:

$$\Phi_{\text{SAT}} \equiv (\exists S)(\forall x)(\exists y)((P(x,y) \wedge S(y)) \vee (N(x,y) \wedge \neg S(y))) .$$

Φ_{SAT} asserts that there exists a set S of variables — the variables that should be assigned **true** — that is a satisfying assignment of the input formula. □

Example 7.2 Boolean query CLIQUE is the set of pairs $\langle G, k\rangle$ such that graph G has a complete subgraph of size k (Example 2.10). The vocabulary for CLIQUE is $\tau_{gk} = \langle E^2, k\rangle$.

The SO$\exists$ sentence Φ_{CLIQUE} says that there is a numbering of the vertices such that those vertices numbered less than k form a clique. In order to describe this numbering it is convenient to existentially quantify a function f. This can be replaced by a binary relation in the usual way (Exercise 7.3). Let Inj(f) mean that f is an injective function,

$$\begin{aligned} \text{Inj}(f) &\equiv (\forall xy)(f(x) = f(y) \rightarrow x = y) \\ \Phi_{\text{CLIQUE}} &\equiv (\exists f^1.\text{Inj}(f))(\forall xy)((x \neq y \wedge f(x) < k \wedge f(y) < k) \rightarrow E(x,y)) \end{aligned}$$

□

Exercise 7.3 Show how formula Φ_{CLIQUE} may be rewritten using an existentially quantified relation F of arity two, rather than function f. $\square$

Exercise 7.4 Hamiltonian-Circuit (HC) is the boolean query that is true of an undirected graph iff it has a Hamiltonian circuit, i.e., a path that starts and ends at the same vertex and visits every other vertex exactly once. Write an SO$\exists$ sentence that expresses HC. [Hint: say that there exists a total ordering of the vertices that determines a Hamiltonian circuit.] $\square$

Exercise 7.5 Write an SO$\exists$ sentence that expresses TSP — the traveling salesperson problem. Boolean query TSP has as input an undirected graph G with weights on its edges and an integer L. The TSP query is true iff G admits a Hamiltonian circuit whose total weight is at most L. In order to code TSP instances as logical structures, we must decide on an appropriate range for the integer weights. To be quite general, you should code these integers as binary strings. Let the vocabulary for TSP be $\tau_{\text{tsp}} = \langle W^3, L^1\rangle$, consisting of a binary string $W(x, y, \cdot)$ for each potential edge (x, y) and a binary string $L(\cdot)$ representing limit L. For pairs (x, y) that are not edges, we can let the edge weight be the maximum value, i.e., the string of all 1's.

[Hint: you can assert the existence of the correct Hamiltonian-Circuit as in Exercise 7.4. To say that the total is at most L, you should assert the existence of a ternary relation R that maintains the running sum.] $\square$

We finish this section by proving the easy direction of Fagin's Theorem.

Proposition 7.6 *The second-order existentially definable boolean queries are all computable in NP. In symbols,* SO$\exists$ $\subseteq$ NP.

Proof Given is a second-order existential sentence $\Phi \equiv (\exists R_1^{r_1}) \ldots (\exists R_k^{r_k})\psi$. Let τ be the vocabulary of Φ. Our task is to build an NP machine N such that for all $\mathcal{A} \in \text{STRUC}[\tau]$,

$$(\mathcal{A} \models \Phi) \quad \Leftrightarrow \quad (N(\text{bin}(\mathcal{A}))\downarrow) \tag{7.7}$$

Let $\mathcal{A}$ be an input structure to N and let $n = \|\mathcal{A}\|$. What N does is to nondeterministically write down a binary string of length n^{r_1} representing R_1, and similarly for R_2 through R_k. By nondeterministically write down a binary string, we mean that at each step, N nondeterministically chooses whether to write a 0 or a 1. After this polynomial number of steps, we have an expanded structure $\mathcal{A}' = (\mathcal{A}, R_1, R_2, \ldots, R_k)$. N should accept iff $\mathcal{A}' \models \psi$. By Theorem 3.1, we can test if $\mathcal{A}' \models \psi$ in logspace, so certainly in NP. Notice that N accepts $\mathcal{A}$ iff there is some choice of relations R_1 through R_k such that $(\mathcal{A}, R_1, R_2, \ldots, R_k) \models \psi$. Thus, Equivalence 7.7 holds. □

7.2 Proof of Fagin's Theorem

The following theorem characterizes complexity class NP in an elegant and machine independent way. This was originally proved in Ron Fagin's 1973 doctoral thesis. It was the theorem that began the subject of descriptive complexity.

Theorem 7.8 (Fagin's Theorem) NP *is equal to the set of existential, second-order boolean queries,* NP = SO∃. *Furthermore, this equality remains true when the first-order part of the second-order formulas is restricted to be universal.*

Proof Let N be a nondeterministic Turing machine that uses time $n^k - 1$ for inputs bin($\mathcal{A}$) with $n = \|\mathcal{A}\|$. We write a second-order sentence,

$$\Phi \quad = \quad (\exists C_1^{2k} \ldots C_g^{2k} \Delta^k)\varphi$$

that says, "There exists an accepting computation $\overline{C}, \Delta$ of N." More precisely, first-order sentence φ will have the property that $(\mathcal{A}, \overline{C}, \Delta) \models \varphi$ iff $\overline{C}, \Delta$ is an accepting computation of N on input $\mathcal{A}$. That is, Equation 7.7 holds.

We describe how to code N's computation. $\overline{C}$ consists of a matrix $\overline{C}(\bar{s}, \bar{t})$ of n^{2k} tape cells with space $\bar{s}$ and time $\bar{t}$ varying between 0 and $n^k - 1$. We use k-tuples of variables $\bar{t} = t_1, \ldots, t_k$ and $\bar{s} = s_1, \ldots s_k$ each ranging over the universe of $\mathcal{A}$, i.e. from 0 to $n - 1$, to code these values. For each $\bar{s}, \bar{t}$ pair, $\overline{C}(\bar{s}, \bar{t})$ codes the tape symbol σ that appears in cell $\bar{s}$ at time $\bar{t}$ if N's head is not on this cell. If the head is present, then $\overline{C}(\bar{s}, \bar{t})$ codes the pair $\langle q, \sigma \rangle$ consisting of N's state q at time $\bar{t}$ and tape symbol σ. Let $\Gamma = \{\gamma_0, \ldots, \gamma_g\} = (Q \times \Sigma) \cup \Sigma$ be a listing of the possible contents of a computation cell. We will let C_i be a $2k$-ary relation variable for $0 \leq i \leq g$. The intuitive meaning of $C_i(\bar{s}, \bar{t})$ is that computation cell $\bar{s}$ at time $\bar{t}$ contains symbol γ_i.

At each step, the nondeterministic Turing machine will make one of at most two possible choices.[1] We encode these choices in k-ary relation Δ. Intuitively, $\Delta(\bar{t})$ is true, if step $\bar{t}+1$ of the computation makes choice "1"; otherwise it makes choice "0". Note that these choices can be determined from $\bar{C}$, but the formula is simplified when we explicitly quantify Δ. See Figure 7.9 for a view of N's computation.

It is now fairly straightforward to write the first-order sentence $\varphi(\overline{C}, \Delta)$ saying that $\overline{C}, \Delta$ codes a valid accepting computation of N. The sentence φ consists of four parts,

$$\varphi \quad \equiv \quad \alpha \wedge \beta \wedge \eta \wedge \zeta,$$

where α asserts that row 0 of the computation correctly codes input bin$(\mathcal{A})$, β says that it is never the case that $C_i(\bar{s},\bar{t})$ and $C_j(\bar{s},\bar{t})$ both hold, for $i \neq j$, η says that for all $\bar{t}$, row $\bar{t}+1$ of $\overline{C}$ follows from row $\bar{t}$ via move $\Delta(\bar{t})$ of N, and ζ says that the last row of the computation includes the accept state. We can write sentence ζ explicitly. We may assume that when N accepts it clears its tape and moves all the way to the left and enters a unique accept state q_f. Let γ_{17} be the member of Γ corresponding to the pair $\langle q_f, 1\rangle$ of state q_f, looking at the symbol 1. Then ζ = $C_{17}(\overline{0}, \overline{max})$.

Sentence α must assert that the input is of length $I_\tau(n)$ for some n and that $\mathcal{A}$ has been correctly coded as bin$(\mathcal{A})$ (cf. Exercise 2.3). For example, suppose that τ includes relation symbol R_1 of arity one. Assume that cell symbols γ_0, γ_1 are '0','1', respectively. Then α includes the following clauses, meaning that cell $0\ldots 0s_k$ contains 1 if $R_1(s_k)$ holds and 0 if it doesn't.

$$\begin{array}{ll} \cdots \quad \wedge & \Big(\bar{t} = 0 = s_1 = \ldots = s_{k-1} \wedge s_k \neq 0 \wedge R_1(s_k) \ \rightarrow \ C_1(\bar{s},\bar{t})\Big) \\ \wedge & \Big(\bar{t} = 0 = s_1 = \ldots = s_{k-1} \wedge s_k \neq 0 \wedge \neg R_1(s_k) \ \rightarrow \ C_0(\bar{s},\bar{t})\Big) \ \wedge \cdots \end{array}$$

The following sentence η asserts that the contents of tape cell $(\bar{s}, \bar{t}{+}1)$ follows from the contents of cells $(\bar{s}-1,\bar{t})$, $(\bar{s},\bar{t})$, and $(\bar{s}+1,\bar{t})$ via the move $\Delta(\bar{t})$ of N. Let $\langle a_{-1}, a_0, a_1, \delta\rangle \overset{N}{\rightarrow} b$ mean that the triple of cell contents a_{-1}, a_0, a_1 lead to cell b via move δ of N.

[1]A nondeterministic Turing machine can make one of at most a bounded number of choices at any step. By reducing this to a binary choice per step, we slow the machine down by a small constant factor and make the analysis simpler.

	Space									
	0	1		p		$n-1$	n		n^k-1	Δ
Time 0	$\langle q_0, w_0\rangle$	w_1		$\cdots$		w_{n-1}	$\sqcup$	$\cdots$	$\sqcup$	δ_0
1	w_0	$\langle q_1, w_1\rangle$		$\cdots$		w_{n-1}	$\sqcup$	$\cdots$	$\sqcup$	δ_1
	$\vdots$	$\vdots$		$\vdots$				$\vdots$		$\vdots$
t			$\boxed{a_{-1}}$	$\boxed{a_0}$	$\boxed{a_1}$					δ_t
$t+1$				$\boxed{b}$						δ_{t+1}
	$\vdots$	$\vdots$		$\vdots$				$\vdots$		$\vdots$
n^k-1	$\langle q_f, 1\rangle$	$\cdots$		$\cdots$				$\cdots$		

Figure 7.9: An NP computation on input $w_0w_1\cdots w_{n-1}$; $\sqcup$ denotes blank

$$\eta_1 \equiv (\forall \bar{t}.\bar{t} \neq \overline{max})(\forall \bar{s}.\bar{0} < \bar{s} < \overline{max}) \bigwedge_{\langle a_{-1},a_0,a_1,\delta\rangle \overset{N}{\to} b} \Big(\neg^{\delta}\Delta(\bar{t}) \vee \neg C_{a_{-1}}(\bar{s}-1,\bar{t}) \vee \neg C_{a_0}(\bar{s},\bar{t}) \vee \neg C_{a_1}(\bar{s}+1,\bar{t}) \vee C_b(\bar{s},\bar{t}+1)\Big)$$

Here, $\neg^{\delta}$ is $\neg$ if $\delta = 1$ and is the empty symbol if $\delta = 0$.

Finally, let $\eta \equiv \eta_0 \wedge \eta_1 \wedge \eta_2$ where η_0 and η_2 encode the same information when $\bar{s} = \bar{0}$ and $\overline{max}$ respectively. □

Observe that the first-order part of formula Φ in the proof of Theorem 7.8 is universal and is in conjunctive normal form. Furthermore, if N is a deterministic polynomial-time machine, then we do not need choice relation Δ, so the first-order part of Φ is a Horn formula.[2] We obtain the following corollary, which is part of Grädel's Theorem (Theorem 9.32).

Corollary 7.10 *Every polynomial-time query is expressible as a second-order, existential Horn formula:* P $\subseteq$ SO$\exists$-*Horn.*

The proof of Theorem 7.8 shows that nondeterministic time n^k is contained in (SO$\exists$, arity $2k$). Lynch improved this to arity k. His proof uses the numeric predicate PLUS. Fagin's theorem holds even without numeric predicates, since we

[2]A Horn formula is a formula in conjunctive normal form with at most one positive literal per clause (Definition 9.26).

can existentially quantify binary relations and assert that they are $\le$ and BIT. However, without the numeric predicates, we need an existential first-order quantifier to specify time $\bar{t}+1$, given time $\bar{t}$.

Theorem 7.11 (Lynch's Theorem)

$$\textit{For } k \ge 1, \quad \mathrm{NTIME}[n^k] \quad \subseteq \quad \mathrm{SO}\exists(\text{arity } k)\ .$$

Proof This is analogous to Lemma 5.31. We modify the proof of Fagin's theorem so that instead of guessing the entire tape at every step only a bounded number of bits per step is guessed. The following relations need to be guessed.

1. $Q_i(\bar{t})$ meaning that the state at move $\bar{t}$ is q_i,
2. $S_i(\bar{t})$ meaning that the symbol written at move $\bar{t}$ is σ_i,
3. $D(\bar{t})$ meaning that the head moves one space to the right after move $\bar{t}$; otherwise it moves one space to the left.

We must write a first-order formula asserting that $\overline{Q}, \overline{S}, D$ encode a correct accepting computation of N. The only difficulty in doing this is that for each move $\bar{t}$, we must ascertain the symbol $\rho_{\bar{t}}$ that is read by N. $\rho_{\bar{t}}$ is equal to σ_i where $S_i(\bar{t}')$ holds, and $\bar{t}'$ is the last time before $\bar{t}$ that the head was in its present location (or it is the corresponding input symbol if this is the first time the head is at this cell).

To express $\rho_{\bar{t}}$, we need to express the function $\bar{s} = p(\bar{t})$ meaning that at time $\bar{t}$, the head is at position $\bar{s}$. Since we are restricted to relations of arity k, we cannot guess the $k \log n$ bits per time needed to specify the function p. The solution to this problem is to do the next best thing and existentially quantify the current head position once every $\log n$ steps. We do this by quantifying k bits per step in relations $P_i(\bar{t})$, $i = 1, 2, \ldots, k$. When we string $\log n$ of these together, from time $r \log n$ through time $(r+1)\log n - 1$, we have a total of $k \log n$ bits which encode the head position at time $r \log n$.

The idea is similar to the proof of Bit Sum Lemma 1.18. Recall that numeric predicate BIT allows us to use each first-order variable to store $\log n$ bits. Furthermore, predicate BSUM(x, y), meaning that the number of one's in the binary expansion of x is y, is first-order (Lemma 1.18). This enables us to assert that relations $\overline{P}$ are consistent with the head movements given by D and thus correctly code the head position at $\log n$ step intervals. Finally, using BSUM again, we can ascertain the head position at any time $\bar{t}$. $\square$

The converse of Lynch's Theorem is an open question:

Open Problem 7.12 Prove or disprove: $\mathrm{SO}\exists(\text{arity } k) \;=\; \mathrm{NTIME}[n^k]$

The subtlety in Open Problem 7.12 is that the first-order part of an $\mathrm{SO}\exists(\text{arity } k)$ formula may have more than k universal quantifiers. Thus, a first step in answering Open Problem 7.12 may be to answer:

Open Problem 7.13 Is there a fixed k such that FO $\subseteq$ DTIME$[n^k]$? Is there a fixed k such that FO $\subseteq$ NTIME$[n^k]$?

Grandjean gave a close relationship between nondeterministic time n^k and the class (SO$\exists$, fun, $k\forall$) of properties expressible by second-order existential sentences including function variables and containing only k universal first-order quantifiers.

Fact 7.14 *For* $k \geq 2$, $\mathrm{NTIME}[n^k] \;\subseteq\; (\mathrm{SO}\exists, \mathit{fun}, k\forall) \;=\; (\mathrm{SO}\exists, \mathit{fun}, k\forall, \mathit{arity}\ k) \;\subseteq\; \mathrm{NTIME}[n^k(\log n)^2]$.

By considering the nondeterministic random access machine (NRAM) instead of the Turing machine, Grandjean later gave an exact bound,

Fact 7.15 *For* $k \geq 1$,

$$\mathit{NRAM\text{-}TIME}[n^k] \;=\; (\mathrm{SO}\exists, \mathit{fun}, k\forall, \mathit{arity}\ k)\ .$$

7.3 NP-Complete Problems

In 1971, Cook proved that SAT (Example 2.18) is NP-complete via polynomial-time Turing reductions [Coo71]. In fact, the problem is NP-complete via significantly weaker reductions:

Theorem 7.16 SAT *is complete for* NP *via first-order reductions.*

Proof This follows from Fagin's theorem. Given any boolean query $B \in$ NP, we know that $B = \mathrm{MOD}[\Phi]$ where $\Phi = (\exists S_1^{a_1} \cdots S_g^{a_g} \Delta^k)(\forall x_1 \cdots x_t)\psi(\bar{x})$, with ψ quantifier-free. We may assume that $\psi(\bar{x}) = \bigwedge_{j=1}^{r} C_j(\bar{x})$ is in conjunctive normal form.

For any input structure $\mathcal{A}$ with $n = \|\mathcal{A}\|$, define the boolean formula $\gamma(\mathcal{A})$ as follows: $\gamma(\mathcal{A})$ has boolean variables: $S_i(e_1, \ldots, e_{a_i})$ and $D(e_1, \ldots, e_k)$, $i =$

$1, \ldots, g, e_1, \ldots, e_{a_i} \in |\mathcal{A}|$. The clauses of $\gamma(\mathcal{A})$ are $C_j(\bar{e})$, $j = 1, \ldots, r$ as $\bar{e}$ ranges over all t-tuples from $|\mathcal{A}|$. In each $C_j(\bar{e})$, there may be some occurrences of numeric or input predicates: $P(\bar{e})$. These should be replaced by **true** or **false** according to whether they are true or false in $\mathcal{A}$.

It is clear from the construction that

$$\mathcal{A} \in B \qquad \Leftrightarrow \qquad \mathcal{A} \models \Phi \qquad \Leftrightarrow \qquad \gamma(\mathcal{A}) \in \text{SAT} .$$

Furthermore, the mapping from $\mathcal{A}$ to $\gamma(\mathcal{A})$ is a $t+1$-ary first-order query. $\square$

Now that we know that SAT is NP-complete via first-order reductions, we can reduce SAT to other SO$\exists$ boolean queries. This is possible iff these other problems are also NP-complete via first-order reductions (Exercise 2.15).

Proposition 7.17 *Let* 3-SAT *be the subset of* SAT *in which each clause has at most three literals. Then* 3-SAT *is* NP-*complete via first-order reductions.*

Proof We show that SAT $\leq_{\text{fo}}$ 3-SAT. Here is an example of the idea behind the reduction. Let $C = (\ell_1 \vee \ell_2 \vee \cdots \vee \ell_7)$ be a clause with more than three literals. Observe that $C \in$ SAT iff $C' \in$ 3-SAT, where C' is the following clause in which new variables $d_1, \ldots, d_4$ are introduced.

$$\begin{aligned} C' \quad \equiv \quad & (\ell_1 \vee \ell_2 \vee d_1) \wedge (\overline{d_1} \vee \ell_3 \vee d_2) \wedge (\overline{d_2} \vee \ell_4 \vee d_3) \wedge \\ & (\overline{d_3} \vee \ell_5 \vee d_4) \wedge (\overline{d_4} \vee \ell_6 \vee \ell_7) \end{aligned}$$

The first-order reduction from SAT to 3-SAT proceeds as follows. Let $\mathcal{A} \in$ STRUC$[\langle P^2, N^2 \rangle]$ be an instance of SAT with $n = \|\mathcal{A}\|$. Each clause c of $\mathcal{A}$ is replaced by $2n$ clauses as follows:

$$\begin{aligned} c' \quad \equiv \quad & ([x_1]^c \vee d_1) \wedge (\overline{d_1} \vee [x_2]^c \vee d_2) \wedge (\overline{d_2} \vee [x_3]^c \vee d_3) \wedge \cdots \wedge \\ & (\overline{d_n} \vee [\overline{x_1}]^c \vee d_{n+1})(\overline{d_{n+1}} \vee [\overline{x_2}]^c \vee d_{n+2}) \wedge \cdots \wedge (\overline{d_{2n-1}} \vee [\overline{x_n}]^c) \end{aligned}$$

Here $[\ell]^c$ means the literal ℓ if this occurs in c and **false** otherwise. It is not hard to see that c' is satisfiable iff c is satisfiable and that c' is definable in a first-order way from c. $\square$

Proposition 7.18 3-COLOR *is* NP-*complete via first-order reductions.*

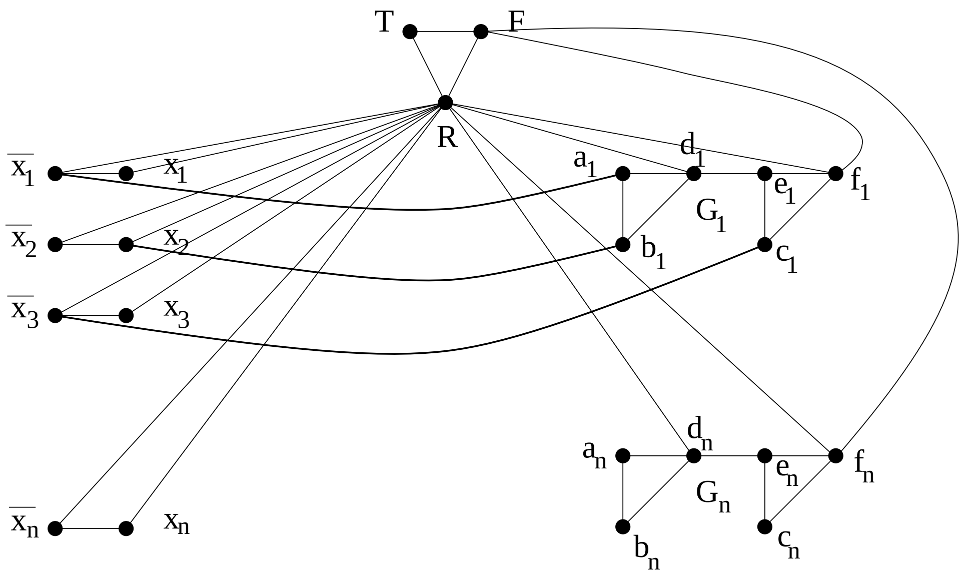

Figure 7.19: 3-SAT $\leq_{\text{fo}}$ 3-COLOR; G_1 encodes clause $C_1 = (\overline{x_1} \vee x_2 \vee \overline{x_3})$

Proof We will show that 3-SAT $\leq_{\text{fo}}$ 3-COLOR. We are given an instance $\mathcal{A}$ of 3-SAT and we must produce a graph $f(\mathcal{A})$ that is three colorable iff $\mathcal{A} \in$ 3-SAT. Let $n = \|\mathcal{A}\|$, so $\mathcal{A}$ is a boolean formula with at most n variables and n clauses.

The construction of $f(\mathcal{A})$ is shown in Figure 7.19. Notice the triangle, with vertices labeled T, F, R. Any three-coloring of the graph must color these three vertices distinct colors. We may assume without loss of generality that the colors used to color T, F, R are true, false, and red, respectively.

Graph $f(\mathcal{A})$ also contains a ladder each rung of which is a variable x_i and its negation $\overline{x_i}$. Each of these is connected to R, meaning that any valid three-coloring colors one of $x_i, \overline{x_i}$ true and the other false.

Finally, for each clause $C_i = \ell_1 \vee \ell_2 \vee \ell_3$, $f(\mathcal{A})$ contains the gadget G_i consisting of six vertices. G_i has three inputs a_i, b_i, c_i, connected to literals ℓ_1, ℓ_2, ℓ_3, respectively, and it has one output, f_i. See Figure 7.19 where gadget G_1 corresponds to clause $C_1 = \overline{x_1} \vee x_2 \vee \overline{x_3}$.

Observe that the triangle a_1, b_1, d_1 serves as an "or"-gate in that d_1 may be colored true iff at least one of its inputs $\overline{x_1}, x_2$ is colored true. Similarly, output f_1 may be colored true iff at least one of d_1 and the third input, $\overline{x_3}$ is colored true. Since f_i is connected to both F and R, f_i can only be colored true. It follows that a three coloring of the literals can be extended to color G_i iff the corresponding truth assignment makes C_i true. Thus, $f(\mathcal{A}) \in$ 3-COLOR iff $\mathcal{A} \in$ 3-SAT.

The details of first-order reduction f are easy to fill in. $f(\mathcal{A})$ consists of one triangle, a ladder with n rungs, and n copies of the gadget. The only dependency on the input $\mathcal{A}$ — as opposed to its size — is that there is an edge from literal ℓ to input j of gadget G_i iff ℓ is the j^{th} literal occurring in C_i. □

7.4 The Polynomial-Time Hierarchy

We defined the polynomial-time hierarchy (PH) to be the set of boolean queries accepted in polynomial time by alternating Turing machines making a bounded number of alternations between existential and universal states (Equation (2.35)). The original definition of the polynomial-time hierarchy was via nondeterministic polynomial-time Turing reductions (Definition 2.9).

Definition 7.20 (Polynomial-Time Hierarchy via Oracles) Let $\Sigma_0^p = \text{P}$ be level 0 of the polynomial-time hierarchy. Inductively, let

$$\Sigma_{i+1}^p \quad = \quad \{L(M^A) \mid M \text{ is an NP oracle TM}, A \in \Sigma_i^p\}$$

Equivalently, Σ_{i+1}^p is the set of boolean queries that are nondeterministic polynomial-time Turing reducible to a set from Σ_i^p,

$$\Sigma_{i+1}^p \quad = \quad \{B \mid B \leq_{np}^t A, \ \text{for some } A \in \Sigma_i^p\}$$

Define Π_i^p to be co-Σ_i^p, $\Pi_i^p = \{\overline{A} \mid A \in \Sigma_i^p\}$. Finally, $\text{PH} = \bigcup_{k=1}^{\infty} \Sigma_k^p$. □

The relationship between second-order boolean queries and the levels of the polynomial hierarchy are given by the following:

Theorem 7.21 *Let $S \subseteq \text{STRUC}[\tau]$ be a boolean query, and let $k \geq 1$. The following are equivalent,*

1. *$S = \text{MOD}[\Phi]$, for some $\Phi \in \Sigma_k^{\text{SO}}$. (Here Σ_k^{SO} is the set of all second-order sentences with second-order quantifier prefix $(\exists\overline{R_1})(\forall\overline{R_2})\ldots(Q_k\overline{R_k})$.)*
2. *$S = \{x \mid (\exists y_1.|y_1| \leq |x|^c)(\forall y_2.|y_2| \leq |x|^c)\cdots(Q_k y_k.|y_k| \leq |x|^c)R(x,\overline{y})\}$ where R is a deterministic polynomial-time predicate on $k+1$ tuples of binary strings and c is a constant.*

3. $S \in \text{ATIME-ALT}[n^{O[1]}, k]$.
4. $S \in \Sigma_k^p$.

Corollary 7.22 *A boolean query is in the polynomial-time hierarchy iff it is second-order expressible,*

$$\text{PH} \quad = \quad \text{SO} \, .$$

From Theorem 4.10 and Corollary 7.22, we obtain the following descriptive characterization of the P? = NP question: P is equal to NP iff every second-order query — over finite, ordered structures — is expressible as a first-order inductive definition.

Corollary 7.23 *The following conditions are equivalent:*

1. P = NP.
2. *Over finite, ordered structures,* FO(LFP) = SO.

Proof If FO(LFP) = SO, then P $\subseteq$ NP $\subseteq$ PH = P. Conversely, if P = NP, then PH = NP, so FO(LFP) = SO. □

Exercise 7.24 Prove Theorem 7.21. [Hint: By induction on k. The subtle part is relating Σ_k^p to the other conditions. For this, note that an NP machine with an oracle $A \in \Sigma_{k-1}^p$ can guess all the answers to its oracle queries. Then, at the end of its computation, it can check that these answers were all correct. This is a polynomial number of Σ_{k-1}^p and Π_{k-1}^p questions.] □

As seen in the following, the polynomial-time hierarchy is robust enough to finesse the difficulty that occurs in Open Problem 7.12,

Exercise 7.25 Prove that for any k,

$$\text{SO(arity } k) \;=\; \text{PH-TIME}[n^k] \;=\; \text{ATIME-ALT}[n^k, O(1)]$$ □

Exercise 7.26 Fagin's Theorem (Theorem 7.8) is a generalization of the Spectrum Theorem. Define the *spectrum* of a first-order sentence φ to be the set of cardinalities of the finite models of φ,

$$\text{spec}(\varphi) \;=\; \big\{ n \;\big|\; n = |\mathcal{A}| \text{ for some } \mathcal{A} \in \text{MOD}[\varphi] \big\} \, .$$

As an example let φ_{field} be the conjunction of the field axioms, so spec(φ_{field}) is the set of prime powers. An interesting question is whether the set of spectra of first-order sentences is closed under complementation, i.e., if S is a spectrum then is $\overline{S} = \mathbf{Z}^+ - S$ one also? As we now see, this is equivalent to an important open question in complexity theory. The Spectrum Theorem says that a set $S \subseteq \mathbf{Z}^+$ is the spectrum of a first-order sentence iff $S \in$ NTIME$[2^{O[n]}]$. Fagin originally called the finite models of SO$\exists$ sentences "generalized spectra".

1. Write a first-order sentence whose spectrum is the set of even positive integers
2. Modify part 1 to get a first-order sentence whose spectrum is the set of odd positive integers.
3. Prove the Spectrum Theorem.

   [Hint: Show how it follows from Theorem 7.8. Note that a problem $S \subseteq \mathbf{Z}^+$ is assumed to be a set of binary strings coding natural numbers. Thus $S \in$ NTIME$[2^{O[n]}]$ iff S coded in unary is in NP.]
4. Show using the Spectrum Theorem that $\overline{\text{spec}(\varphi_{\text{field}})}$ is a spectrum. □

As a corollary to the proof of Theorem 5.2, we obtain the following characterization of PH as a parallel complexity class. Up to this point, we had been assuming for notational simplicity that a CRAM has at most polynomially many processors. However, the class CRAM-PROC$[t(n), p(n)]$ still makes sense for numbers of processors $p(n)$ that are not polynomially bounded.

Corollary 7.27 PH *is equal to the set of boolean queries recognizable by a* CRAM *using exponentially many processors and constant time,*

$$\text{PH} \quad = \quad \bigcup_{k=1}^{\infty} \text{CRAM-PROC}[1, 2^{n^k}]$$

Proof The inclusion SO $\subseteq$ CRAM-PROC$[1, 2^{n^{O[1]}}]$ follows just as in the proof of Lemma 5.4. A processor number is now large enough to give values to all the relational variables as well as to all the first-order variables. Thus, as in Lemma 5.4, the CRAM can evaluate each first or second-order quantifier in three steps.

The inclusion CRAM-PROC$[1, 2^{n^{O[1]}}] \subseteq$ SO follows just as in the proof of Lemma 5.3. The only difference is that we use second-order variables to specify the processor number. □

In fact, Corollary 7.27 can be extended to,

Corollary 7.28 *For all constructible $t(n)$,*

$$\text{SO}[t(n)] \quad = \quad \text{CRAM-PROC}[t(n), 2^{n^{O[1]}}] \, .$$

Observe that Corollary 7.27 suggests that PH is a rather strange complexity class. No one would ever buy exponentially many processors and then use them only for constant time. See Corollary 10.30 for an interesting characterization of the much more robust complexity class PSPACE as exponentially many processors running in polynomial time.

Historical Notes and Suggestions for Further Reading

Theorem 7.8 (Fagin's theorem) was proved in Fagin's thesis, [Fag73, Fag74]. The idea of using choice relation Δ is due to Papadimitriou [Pap94]. The Spectrum Theorem discussed in Exercise 7.26 is due to Jones and Selman [JS74]. See [B82] for a history of the spectrum problem.

Theorem 7.16 was first proved by Lovász and Gács [LG77]. Dahlhaus proved that SAT is NP-complete via quantifier-free, first-order reductions [Da84].

The polynomial-hierarchy (PH) was defined by Stockmeyer [Sto77]. Corollary 7.22 appears there as well. Item 2 of Theorem 7.21 is due to Wrathall [Wra76].

Some of the simple ways to write NP-complete problems as SO$\exists$ formulas, like CLIQUE (Example 7.2) are due to Jose Antonio Medina [MI94].

Theorem 7.11 is due to Lynch, [Lyn82]. Facts 7.14 and 7.15 are due to Grandjean; see [Gra84, Gra85, Gra89] for their proofs. An interesting place to start investigating Open Problem 7.13 is to consider the deterministic time complexity of problem CLIQUE(k) — the set of graphs containing a k-clique — for a fixed k. The best known algorithm is due to Boppana and Halldórsson [BH92].

Exercise 7.25 is from [I83]. Corollary 7.27 is from [I89a].

Bibliography

[AV91] S. Abiteboul and V. Vianu, "Generic Computation And Its Complexity," *32nd IEEE Symposium on FOCS* (1991), 209-219.

[AHV95] S. Abiteboul, R. Hull, and V. Vianu, *Foundations of Databases*, 1995, Addison-Wesley.

[AVV97] S. Abiteboul, M.Y. Vardi, and V. Vianu, "Fixpoint Logics, Relational Machines, and Computational Complexity," *J. Assoc. Comput. Mach.* 44(1) (1997), 30 – 56.

[Agr01] M. Agrawal, "The First-Order Isomorphism Theorem," *Foundations of Software Technology and Theoretical computer Science,* (2001), 58-69.

[AAI97] M. Agrawal, E. Allender, R. Impagliazzo, T. Pitassi and S. Rudich, "Reducing the Complexity of Reductions," *ACM Symp. Theory Of Comput.* (1997), 730–738.

[AHU74] A.V. Aho, J.E. Hopcroft and J.D. Ullman, *The Design and Analysis of Computer Algorithms,* Addison- Wesley (1974).

[Ajt83] M. Ajtai, "Σ^1_1 Formulae on Finite Structures," *Annals of Pure and Applied Logic* 24 (1983), 1-48.

[Ajt87] M. Ajtai, "Recursive Construction for 3-Regular Expanders," *28th IEEE Symp. on Foundations of Computer Science* (1987), 295-304.

[Ajt89] M. Ajtai, "First-Order Definability on Finite Structures," *Annals of Pure and Applied Logic* (1989), 211-225.

[AF90] M. Ajtai and R. Fagin, "Reachability is Harder for Directed than for Undirected Graphs," *J. Symb. Logic,* 55 (1990), 113-150.

[AFS97] M. Ajtai, R. Fagin, and L. Stockmeyer, "The Closure of Monadic NP," IBM Research Report RJ 10092 (1997). The closure of monadic NP, with Miklos Ajtai and Larry Stockmeyer. J. Computer and System Sciences 60, June 2000, pp. 660-716. (Special issue for selected papers from the 1998 ACM Symp. on Theory of Computing).

[AG87] M. Ajtai and Y. Gurevich, "Monotone versus Positive," *J. Assoc. Comput. Mach.* 34 (1987), 1004–1015.

[AKL79] R. Aleliunas, R. Karp, R. Lipton, L. Lovász, and C. Rackoff, "Random Walks, Universal Traversal Sequences, and the Complexity of Maze Problems," *IEEE Found. of Comp. Sci. Symp.* (1979), 218-233.

[ABI97] E. Allender, N. Immerman, J. Balcázar,"A First-Order Isomorphism Theorem," *SIAM J. Comput.* 26(2) (1997), 557-567.

[ACF94] B. Alpern, L. Carter, E. Feig, and T. Selker, "The Uniform Memory Hierarchy Model of Computation," *Algorithmica, 12(2-3)* (1994), 72–109.

[AG94] G. Almasi and A. Gottlieb, *Highly Parallel Computing (Second Edition)* 1994, Benjamin-Cummings.

[ASE92] N. Alon, J. Spencer, and P. Erdős, *The Probabilistic Method,* 1992, John Wiley and Sons, Inc.

[ADN95] H. Andréka, I. Düntsch, and I. Németi, "Expressibility of Properties of Relations," *J. Symbolic Logic* 60(3) (1995), 970 - 991.

[AF97] S. Arora and R. Fagin, "On Winning Strategies in Ehrenfeucht-Fraïssé Games," *Theoret. Comp. Sci.* 174(1-2) (1997), 97-121.

[ALMSS] S. Arora, C. Lund, R. Motwani, M.Sudan, and M. Szegedy, "Proof Verification and the Hardness of Approximation Problems," *J. Assoc. Comput. Mach.* 45(3) (1998), 501-555.

[AS92] S. Arora and S. Safra, "Probabilistic Checking of Proofs: a New Characterization of NP," *J. Assoc. Comput. Mach.* 45(1) (1998), 70-122. A preliminary version appeared in *IEEE Found. of Comp. Sci. Symp.* (1992), 2-13.

[Ba81] L. Babai, "Moderately Exponential Bound for Graph Isomorphism,"in *Proc. Int. Conf. Fundamentals of Computation Theory* (1981), Springer LNCS, 34 – 50.

[BK80] L. Babai and L. Kučera, Canonical Labelling of Graphs in Linear Average Time," *20th IEEE Symp. on Foundations of Computer Science* (1980), 39-46.

[BL83] L. Babai and E. Luks, "Canonical Labelling of Graphs," *15th ACM STOC Symp.*, (1983), 171-183.

[BDG88] J. Balcázar, J. Días, and J. Gabarró, *Structural Complexity,* Vols. I and II, EATCS Monographs on Theoretical Computer Science, 1988, Springer-Verlag.

[BI97] D.M. Barrington and N. Immerman, "Time, Hardware, and Uniformity," in *Complexity Theory Retrospective II*, L. Hemaspaandra and A. Selman, editors, 1997, Springer-Verlag, 1-22.

[BIS88] D.M. Barrington, N. Immerman, and H. Straubing, "On Uniformity Within NC^1," *Third Annual Structure in Complexity Theory Symp.* (1988), 47-59.

[BBI97] D.M. Barrington, J. Buss, and N. Immerman, "Capturing Deterministic Space via Number of Variables," in preparation.

[Bar77] J. Barwise, "On Moschovakis Closure Ordinals," J. Symb. Logic 42 (1977), 292-296.

[Bea86] P. Beame, "Limits on the Power of Concurrent-Write Parallel Machines," *18th ACM STOC* (1986), 169-176.

[Bea96] P. Beame, "A Switching Lemma Primer," manuscript, http://www.cs.washington.edu/homes/beame/papers.html.

[Ben62] J. Bennett, "On Spectra" (1962), Ph.D. thesis, Princeton University.

[BGK85] A. Blass, Y. Gurevich and D. Kozen, "A Zero–One Law for Logic With a Fixed Point Operator," *Information and Computation* 67 (1985), 70-90.

[Bol82] Béla Bollobás, *Random Graphs,* Academic Press (1982).

[BH92] R. Boppana and M. Halldórsson, "Approximating Maximum Independent Sets by Excluding Subgraphs," *BIT* 32(2) (1992), 180-196.

[BS90] R. Boppana and M. Sipser, "The Complexity of Finite Functions," in *Handbook of Theoretical Computer Science, Vol. A* 1990, Jan van Leeuwen, ed., Elsevier, Amsterdam and M.I.T. Press, Cambridge, MA.

[B82] E. Börger, “Decision Problems in Predicate Logic,” in *Logic Colloquium ’82,* G. Lolli, G. Longo and A. Marcia (editors) North-Holland, 1984, 263 – 301.

[BCD88] A. Borodin, S.A. Cook, P.W. Dymond, W.L. Ruzzo, and M. Tompa, “Two Applications of Complementation via Inductive Counting,” *Third Annual Structure in Complexity Theory Symp.* (1988), 116-125.

[BCH86] P. Beame, S. Cook, H.J. Hoover, “Log Depth Circuits for Division and Related Problems,” *SIAM J. Comput. 15:4* (1986), 994-1003.

[BCP83] A. Borodin, S. Cook, and N. Pippenger, “Parallel Computation for Well-Endowed Rings and Space-Bounded Probabilistic Machines,” *Information and Control,* 58 (1983), 113-136.

[Bra96] J. Bradfield, “On the Expressivity of the Modal Mu-Calculus,” *Symp. Theoretical Aspects Comp. Sci.* (1996).

[Büc60] R. Büchi, “Weak Second-Order Arithmetic and Finite Automata,” *Zeit. Math. Logik. Grund. Math. 6* (1960), 66-92.

[CFI92] J.-Y. Cai, M. Fürer, N. Immerman, “An Optimal Lower Bound on the Number of Variables for Graph Identification,” *Combinatorica* 12 (4) (1992) 389-410.

[CH80a] Ashok Chandra and David Harel, “Computable Queries for Relational Databases,” *JCSS* 21(2) (1980), 156-178.

[CH80b] Ashok Chandra and David Harel, “Structure and Complexity of Relational Queries,” *IEEE Found. of Comp. Sci. Symp.* (1980), 333-347. Also appeared in a revised as [CH82]

[CH82] A. Chandra and D. Harel, “Structure and Complexity of Relational Queries,” *JCSS* 25 (1982), 99-128.

[CKS81] A. Chandra, D. Kozen, and L. Stockmeyer, “Alternation,” *JACM,* 28, No. 1, (1981), 114-133.

[CSV84] A. Chandra, L. Stockmeyer and U. Vishkin, “Constant Depth Reducibility,” *SIAM J. of Comp.* 13, No. 2, 1984, (423-439).

[CE81] E. Clarke and E.A. Emerson, “Design and Synthesis of Synchronization Skeletons Using Branching Time Temporal Logic,” in *Proc. Workshop on Logic of Programs*, LNCS 131, 1981, Springer-Verlag, 52-71.

[Coh66] P. Cohen, *Set Theory and the Continuum Hypothesis,* 1966, Benjamin.

[Co88] K. Compton, "0-1 laws in logic and combinatorics," in *NATO Adv. Study Inst. on Algorithms and Order*, I. Rival, editor, 1988, D. Reidel, 353–383.

[Coo71] S. Cook, "The Complexity of Theorem Proving Procedures," *Proc. Third Annual ACM STOC Symp.* (1971), 151-158.

[Coo85] S. Cook, "A Taxonomy of Problems with Fast Parallel Algorithms," *Information and Control* 64 (1985), 2-22.

[Cop94] D. Coppersmith, "A Left Coset Composed of n-cycles," Research Report RC19511 IBM (1994).

[Cou90] B. Courcelle, "The Monadic Second-Order Logic of GraphsI: Recognizable Sets of Finite Graphs," *Information and Computation* 85 (1990), 12 - 75.

[Cou97] B. Courcelle, "On the Expression of Graph Properties in Some Fragments of Monadic Second-Order Logic," in *Descriptive Complexity and Finite Models,* N. Immerman and Ph. Kolaitis, eds., 1997, American Mathematical Society, 33 - 62.

[Da84] E. Dahlhaus, "Reduction to NP-Complete Problems by Interpretations," in *Logic and Machines: Decision Problems and Complexity,* Börger, Rödding, and Hasenjaeger eds., Lecture Notes In Computer Science 171, Springer-Verlag (1984), 357-365.

[Daw93] A. Dawar, "Feasible Computation Through Model Theory," PhD Dissertation, University of Pennsylvania (1993).

[DGH98] A. Dawar, G. Gottlob, L. Hella, "Capturing Relativized Complexity Classes without Order," to appear in *Mathematical Logic Quarterly*.

[DH95] Anuj Dawar and Lauri Hella, "The Expressive Power of Finitely Many Generalized Quantifiers," *Information and Computation* 123(2) (1995), 172-184.

[DDLW98] A. Dawar, K. Doets, S. Lindell, and S. Weinstein, "Elementary Properties of the Finite Ranks," /it Mathematical Logic Quarterly 44 (1998), 349-353.

[DLW95] A. Dawar, S. Lindell, and S. Weinstein, "Infinitary logic and inductive definability over finite structures," *Information and Computation* , 119 (1995), 160-175.

[DGS86] L. Denenberg, Y. Gurevich and S. Shelah, "Definability by Constant-Depth Polynomial-Size Circuits", *Information and Control* 70 (1986), 216-240.

[deR84] M. de Rougemont, "Uniform Definability on Finite Structures with Successor," *16th ACM STOC Symp.*, (1984), 409-417.

[DL98] W. Diffie and S. Landau, *Privacy on the Line: the Politics of Wiretapping and Encryption,* MIT Press, 1998.

[DS95] G. Dong, J. Su, "Space-Bounded FOIES," *ACM Symp. Principles Database Systems* (1995), 139 -150.

[DS93] G. Dong, J. Su, "Incremental and Decremental Evaluation of Transitive Closure by First-Order Queries," *Information and Computation* , 120(1) (1995), 101-106. Preliminary results presented at the 1993 Australian Computer Science Conference.

[EF95] H.-D. Ebbinghaus, J. Flum, *Finite Model Theory* 1995, Springer 1995.

[EFT94] H.-D. Ebbinghaus, J. Flum, and W. Thomas, *Mathematical Logic*, 2nd edition 1994, Springer-Verlag.

[Ehr61] A. Ehrenfeucht, "An Application of Games to the Completeness Problem for Formalized Theories," Fund. Math. 49 (1961), 129-141.

[End72] H. Enderton, *A Mathematical Introduction to Logic,* Academic Press, 1972.

[ES74] P. Erdős and J. Spencer, *Probabilistic Methods in Combinatorics,* 1974, Academic Press.

[Ete95] K. Etessami, "Counting Quantifiers, Successor Relations, and Logarithmic Space," *IEEE Structure in Complexity Theory Symp.* (1995), 2-11.

[Ete95a] K. Etessami, "Ordering and Descriptive Complexity" Ph.D. thesis, 1995, UMass, Amherst.

[EI95] K. Etessami and N. Immerman, "Reachability and the Power of Local Ordering," *Theoret. Comp. Sci.* 148(2) (1995), 261-279.

[EI95a] K. Etessami and N. Immerman, "Tree Canonization and Transitive Closure," to appear in *Information and Computation* . A preliminary version appeared in *IEEE Symp. Logic In Comput. Sci.* (1995), 331-341.

[Fag73] R. Fagin, "Contributions to the Model Theory of Finite Structures", Ph.D. Thesis (1973), U. C. Berkeley.

[Fag74] R. Fagin, "Generalized First-Order Spectra and Polynomial-Time Recognizable Sets," in *Complexity of Computation,* (ed. R. Karp), *SIAM-AMS Proc. 7* (1974), 43-73.

[Fag75] R. Fagin, "Monadic generalized spectra," *Zeitschr. f. math. Logik und Grundlagen d. Math.* 21 (1975), 89-96.

[Fag76] R. Fagin, "Probabilities on Finite Models," *J. Symbol. Logic* 41, No. 1 (1976),50-58.

[Fag93] R. Fagin, "Finite-Model Theory – a Personal Perspective," *Theoret. Comp. Sci.* 116 (1993), 3-31.

[Fag97] R. Fagin, "Easier Ways to Win Logical Games," in *Descriptive Complexity and Finite Models,* N. Immerman and Ph. Kolaitis, eds., 1997, American Mathematical Society, 1 - 32.

[FSV95] R. Fagin, L. Stockmeyer, and M.Y. Vardi, "On monadic NP vs. monadic co-NP," *Information and Computation* 120(1) (1995), 78-92.

[FV98] T. Feder and M.Y. Vardi, "The Computational Structure of Monotone Monadic SNP and Constraint Satisfaction: A Study through Datalog and Group Theory," (1998).

[Fel50] W. Feller, *An Introduction to Probability Theory and Its Applications,* Vol. 1, 1950, John Wiley, New York.

[FRW84] F. Fich, Prabhakar Ragde, and, Avi Wigderson (1984), "Relations Between Concurrent-Write Models of Parallel Computation," *Third ACM Symp. on Principles of Distributed Computing,* 179-189.

[FW78] S. Fortune and J. Wyllie, "Parallelism in Random Access Machines," *ACM Symp. Theory Of Comput.* (1978), 114-118.

[Fra54] R. Fraïssé, "Sur les Classifications des Systems de Relations," Publ. Sci. Univ. Alger I (1954).

[Fur87] M. Fürer, "A Counterexample In Graph Isomorphism Testing – Extended Abstract," manuscript (October, 1987).

[FSS84] M. Furst, J.B. Saxe, and M. Sipser, "Parity, Circuits, and the Polynomial-Time Hierarchy," *Math. Systems Theory*, 17 (1984), 13-27.

[Gab81] D. Gabbay, "Expressive Functional Completeness in Tense Logic," in: *Aspects of Philosophical Logic,* 1981, ed. Monnich, D. Reidel, Dordrecht, 91-117.

[Gai81] H. Gaifman, "On Local and Non-Local Properties," *Proc. Herbrand Logic Colloq.* (1981), 105-135.

[GH97] F. Gire and H. Hoang, "An Extension of Fixpoint Logic with a Symmetry-Based Choice Construct," *Information and Computation* 144(1) (1998), 40-65.

[Göd30] Gödel, K., "Die Vollständigkeit der Axiome des Logischen Funktionenkalküls," *Monatshefte für Mathematik und Physik 37* (1930), 349 - 360, (English translation in [vH67]).

[Go82] L. Goldschlager, "A Universal Interconnection Pattern for Parallel Computers," JACM, October 1982.

[Go77] L. Goldschlager, "The Monotone and Planar Circuit Value Problems are Log Space Complete for P," *SIGACT News* 9(2) (1977).

[Grä92] E. Grädel, "Capturing Complexity Classes by Fragments of Second Order Logic," *Theoret. Comp. Sci.* 101 (1992), 35-57.

[Grä92a] E. Grädel, "On Transitive Closure Logic," *Computer Science Logic* (1992), LNCS, Springer, 149–163.

[GM95] E. Grädel and G. McColm, "On the Power of Deterministic Transitive Closures," *Information and Computation* 119 (1995), 129-135.

[GM96] E. Grädel and G. McColm, "Hierarchies in Transitive Closure Logic, Stratified Datalog and Infinitary Logic," *Annals of Pure and Applied Logic 77* (1996), 166–199.

[GO93] E. Grädel and M. Otto, "Inductive Definability with Counting on Finite Structures," *Computer Science Logic* 1993, LNCS 702, Springer, 231–247.

[Gra84] E. Grandjean, “The Spectra of First-Order Sentences and Computational Complexity,” SIAM J. of Comp. 13, No. 2 (1984), 356-373.

[Gra85] E. Grandjean, “Universal quantifiers and time complexity of Random Access Machines,” *Math. Syst. Th.* (1985), 171-187.

[Gra89] E. Grandjean, “First-order spectra with one variable,” to appear in *J. Comput. Syst. Sci.*

[Gro95] M. Grohe, “Complete Problems for Fixed-Point Logics,” *J. Symbolic Logic* 60 (1995), 517-527.

[Gro96] M. Grohe, “Equivalence in Finite-Variable Logics is Complete for Polynomial Time,” *IEEE Found. of Comp. Sci. Symp.* (1996).

[Gro96a] M. Grohe, “Arity Hierarchies,” *Annals of Pure and Applied Logic* 82 (1996), 103-163.

[Gro97] M. Grohe, “Large Finite Structures With Few L^k-Types,” *IEEE Symp. Logic In Comput. Sci.* (1997).

[GS98] M. Grohe and T. Schwentick, “Locality of order-invariant first-order forumlas. MFCS’98, 437-445. Full version to appear in ACM TOCL.

[Gro97a] M. Grohe, “Canonization for L^k-Invariants is Hard,” *Annual Conference of the European Association for Computer Science Logic* (1997), M. Nielsen and W. Thomas, eds., 185-200.

[Gro97b] M. Grohe, “Fixed-Point Logics on Planar Graphs,” manuscript (1997).

[Gur83] Y. Gurevich, ”Algebras of feasible functions,” *IEEE Found. of Comp. Sci. Symp.* (1983), 210-214.

[Gur84] Y. Gurevich, “Toward Logic Tailored for Computational Complexity,” *Computation and Proof Theory* (M.M. Richter et. al., eds.). Springer-Verlag Lecture Notes in Math. 1104 (1984), 175-216.

[Gur88] Y. Gurevich, “Logic and the Challenge of Computer Science,” in *Current Trends in Theoretical Computer Science,* ed. E. Börger, Computer Science Press (1988), 1-57.

[Gur91] Y. Gurevich, “ Evolving Algebras: A Tutorial Introduction,” *Bulletin of EATCS, 43* (1991), 264-284.

[Gur93] Y. Gurevich, ”Evolving Algebras 1993: Lipari Guide”, *Specification and Validation Methods,* ed. E. Börger, Oxford University Press, 1995, 9–36.

[GS85] Y. Gurevich and S. Shelah, “Fixed-Point Extensions of First-Order Logic,” *Annals of Pure and Applied Logic 32* (1986), 265–280.

[GS96] Y. Gurevich and S. Shelah, “On Finite Rigid Structures,” *J. Symbolic Logic* 61(2) (1996), 549 - 562.

[HP93] P. Hajek and P. Pudlak, *Metamathematics of First-Order Arithmetic,* 1993, Springer, Berlin.

[Ha65] W. Hanf, “Model-Theoretic Methods in the Study of Elementary Logic,” in J. Addison, L. Henkin, and A. Tarski, eds., *The Theory of Models,* 1965, North Holland, 105-135.

[HIM78] J. Hartmanis, N. Immerman, and S. Mahaney, “One-Way Log Tape Reductions,” *IEEE Found. of Comp. Sci. Symp.* (1978), 65-72.

[Has86] J. Hastad, “Almost Optimal Lower Bounds for Small Depth Circuits,” *18th ACM STOC Symp.,* (1986), 6-20.

[He96] L. Hella, “Logical Hierarchies in PTIME,” *Information and Computation* 129(1) (1996), 1-19.

[HKL97] L. Hella, Ph. Kolaitis, and K. Luosto, “How to Define a Linear Order on Finite Models,”*Annals of Pure and Applied Logic* 87 (1997), 241-267.

[HKL96] L. Hella, Ph. Kolaitis, and K. Luosto, “Almost Everywhere Equivalence of Logics in Finite Model Theory,” *Bulletin of Symbolic Logic* 2(4) (1996), 422 - 443.

[HLN97] L. Hella, L. Libkin, and J. Nurmonen, “Notions of locality and their logical characterizations over finite models,” manuscript.

[Hil85] D. Hillis, *The Connection Machine* 1985, MIT Press.

[Hon82] J.-W. Hong, “On Some Deterministic Space Complexity Problems,” *SIAM J. Comput.* 11 (1982), 591-601.

[Ho86] J.-W. Hong, *Computation: Computability, Similarity, and Duality,* 1986, John Wiley & Sons.

[HT72] J. Hopcroft and R. Tarjan, "Isomorphism of Planar Graphs," in *Complexity of Computer Computations,* R. Miller and J.W. Thatcher, eds., (1972), Plenum Press, 131-152.

[HU79] J. Hopcroft and J. Ullman, *Introdution to Automata Theory, Languages, and Computation,* Addison-Wesley (1979).

[I79] N. Immerman, "Length of Predicate Calculus Formulas as a New Complexity Measure," *20th IEEE FOCS Symp.* (1979), 337-347. Revised version: "Number of Quantifiers is Better than Number of Tape Cells," *JCSS* 22(3), June 1981, 65-72.

[I80] N. Immerman, "Upper and Lower Bounds for First Order Expressibility,"*21st IEEE FOCS Symp.* (1980), 74-82. Revised version: *JCSS* 25(1) (1982), 76-98.

[I82] N. Immerman, "Relational Queries Computable in Polynomial Time," *14th ACM STOC Symp.* (1982), 147-152. Revised version: *Information and Control,* 68 (1986), 86-104.

[I83] N. Immerman, "Languages Which Capture Complexity Classes," *15th ACM STOC Symp.* (1983), 347-354. Revised version: "Languages That Capture Complexity Classes," *SIAM J. Comput.* 16(4) (1987), 760-778.

[I87] N. Immerman, "Expressibility as a Complexity Measure: Results and Directions," *Second Structure in Complexity Theory Conf.* (1987), 194-202.

[I88] N. Immerman, "Nondeterministic Space is Closed Under Complementation," *SIAM J. Comput.* 17(5) (1988), 935-938. Also appeared in *Third Structure in Complexity Theory Conf.* (1988), 112-115.

[I89] N. Immerman, "Descriptive and Computational Complexity,"in *Computational Complexity Theory,* ed. J. Hartmanis, Lecture Notes for AMS Short Course on Computational Complexity Theory, *Proc. Symp. in Applied Math.* 38, American Mathematical Society (1989), 75-91.

[I89a] N. Immerman, "Expressibility and Parallel Complexity," *SIAM J. of Comput.* 18 (1989), 625-638.

[I91] N. Immerman, "DSPACE$[n^k]$ $=$ VAR$[k+1]$," *Sixth IEEE Structure in Complexity Theory Symp.* (July, 1991), 334-340.

[IKL95] N. Immerman, Ph. Kolaitis, and J. Lynch, "A Tutorial on Finite Model Theory," DIMACS, August, 1995.

[IK87] N. Immerman and D. Kozen, "Definablitity with Bounded Number of Bound Variables," *Second LICS Symp.* (1987), 236-244.

[IL95] N. Immerman, S. Landau, "The Complexity of Iterated Multiplication," *Information and Computation* 116(1) (1995), 103-116.

[IL90] N. Immerman and E. Lander, "Describing Graphs: A First-Order Approach to Graph Canonization," in *Complexity Theory Retrospective,* Alan Selman, ed., Springer-Verlag (1990), 59-81.

[IPS96] N. Immerman, S. Patnaik and D. Stemple, "The Expressiveness of a Family of Finite Set Languages," *Theoretical Computer Science* 155(1) (1996), 111-140. A preliminary version appeared in *Tenth ACM Symposium on Principles of Database Systems* (1991), 37-52.

[IV97] N. Immerman and M.Y. Vardi, "Model Checking and Transitive Closure Logic," *Proc. 9th Int'l Conf. on Computer-Aided Verification* (1997), Lecture Notes in Computer Science, Springer-Verlag, 291 - 302.

[JL77] N. Jones and W. Laaser, "Complete Problems for Deterministic Polynomial Time," *Theoret. Comp. Sci.* 3 (1977), 105-117.

[JLL76] N. Jones, E. Lien and W. Laaser, "New Problems Complete for Nondeterministic Logspace," *Math. Systems Theory* 10 (1976), 1-17.

[JS74] N. Jones and A. Selman, "Turing Machines and the Spectra of First-Order Formulas," *J. Symbolic Logic* 39 (1974), 139-150.

[Ka79] R. Karp, "Probabilistic Analysis of a Canonical Numbering Algorithm for Graphs," *Relations between combinatorics and other parts of mathematics,* Proceedings of Symposia in Pure Mathematics 34, 1979, American Mathematical Society, 365 - 378.

[KL82] R. Karp and R. Lipton, "Turing Machines That Take Advice," *Ensiegn. Math.* 28 (1982), 192-209.

[KV95] Ph. Kolaitis and J. Väänänen, "Generalized Quantifiers and Pebble Games on Finite Structures," *Annals of Pure and Applied Logic*, 74(1) (1995), 23–75.

[KV98] Ph. Kolaitis and M.Y. Vardi, "Conjunctive-Query Containment and Constraint Satisfaction," *ACM Symp. Principles Database Systems* (1998).

[KV92a] Ph. Kolaitis and M.Y. Vardi, "0-1 Laws for Fragments of Second-Order Logic: an Overview," in Y. Moschovakis, editor, *Logic From Computer Science* 1992, Springer-Verlag, 265–286.

[Ku87] L. Kučera, "Canonical Labeling of Regular Graphs in Linear Average Time," *28th IEEE FOCS Symp.* (1987), 271-279.

[Kur64] S. Kuroda, "Classes of Languages and Linear-Bounded Automata," *Information and Control* 7 (1964), 207-233.

[Kur94] R. Kurshan, *Computer-Aided Verification of Coordinating Processes,* 1994, Princeton University Press, Princeton, NJ.

[L75] R. Ladner, "The Circuit Value Problem is log space complete for P," *SIGACT News,* 7(1) (1975), 18 – 20.

[LR96] R. Lassaigne and M. de Rougemont, *Logique et Complexité,* 1996, Hermes.

[LJK87] K.J. Lange, B. Jenner, and B. Kirsig, "The Logarithmic Hierarchy Collapses: $A\Sigma_2^L = A\Pi_2^L$," *14th ICALP* (1987).

[Lei87] D. Leivant, "Characterization of Complexity Classes in Higher-Order Logic," *Second Structure in Complexity Theory Conf.* (1987), 203–217.

[Lei89] D. Leivant, "Descriptive Characterizations of Computational Complexity," *J. Comput. Sys. Sci.* 39 (1989), 51-83.

[LP81] H. Lewis and C. Papadimitriou, *Elements of the Theory of Computation* 1982, Prentice-Hall.

[LP82] H. Lewis and C. Papadimitriou, "Symmetric Space Bounded Computation," *Theoret. Comput. Sci.* 19 (1982),161-187.

[LV93] M. Li and P. Vitányi, *An Introduction to Kolmogorov Complexity and its Applications,* 1993, Springer-Verlag, New York.

[L92] S. Lindell, "A purely logical characterization of circuit uniformity," *7th Structure in Complexity Theory Conf.* (1992), 185-192.

[L] S. Lindell, ”How to define exponentiation from addition and multiplation in first-order logic on finite structures,” manuscript.

[Lin66] P. Lindström, ”First Order Predicate Logic with Generalized Quantifiers,” *Theoria, 32* (1966), 186 195.

[LG77] L. Lovász and P. Gács, “Some Remarks on Generalized Spectra,” *Zeitchr. f. math, Logik und Grundlagen d. Math,* 23 (1977), 547-554.

[Luk82] E. Luks, ”Isomorphism of Graphs of Bounded Valence Can be Tested in Polynomial Time,” *J. Comput. Sys. Sci.* 25 (1982), pp. 42-65.

[Lyn82] J. Lynch, “Complexity Classes and Theories of Finite Models,” *Math. Sys. Theory* 15 (1982), 127-144.

[MP94] J. Makowsky, Y. Pnueli, “Arity versus alternation in Second-Order Logic,” in *Logical Foundations of Computer Science,* A. Nerode, Y. Matiyasevich eds., Springer LNCS 813 1994, 240 - 252.

[McM93] K. McMillan, *Symbolic Model Checking,* 1993, Kluwer.

[MP71] R. McNaughton and S. Papert, *Counter-Free Automata,* 1971, MIT Press, Cambridge, MA.

[Man89] U. Manber, *Introduction to Algorithms: A Creative Approach,* Addison-Wesley, (1989).

[MT97] O. Matz and W. Thomas, “The Monadic Quantifier Alternation Hierarchy Over Graphs is Infinite,” *IEEE Symp. Logic In Comput. Sci.* (1997), 236-244.

[MI94] J.A. Medina and N. Immerman, “A Syntactic Characterization of NP-Completeness,” *IEEE Symp. Logic In Comput. Sci.* (1994), 241-250.

[MI96] J.A. Medina and N. Immerman, “A Generalization of Fagin’s Theorem,” *IEEE Symp. Logic In Comput. Sci.* (1996), 2 – 12.

[MSV94] P. Miltersen, S. Subramanian, J. Vitter, and R. Tamassia, “Complexity Models for Incremental Computation,” *Theoret. Comp. Sci.* (130:1) (1994), 203-236.

[Mos74] Y. Moschovakis, *Elementary Induction on Abstract Structures,* North Holland (1974).

[Mos80] Y. Moschovakis, *Descriptive set theory,* 1980, North-Holland Pub. Co., Amsterdam, 637 p.

[NT95] N. Nisan and A. Ta-Shma, “Symmetric Logspace is Closed Under Complement,” *Chicago J. Theoret. Comp. Sci.* (1995).

[O01] M. Otto, “Two Variable first-Order Logic Over Ordered Domains,” *J. Symbolic Logic* 66(2) (2001), 685-702.

[Ott96] M. Otto, “The Expressive Power of Fixed-Point Logic with Counting,” *J. Symbolic Logic* 61(1) (1996), 147 - 176

[Ott97] M. Otto, *Bounded Variable Logics and Counting: A Study in Finite Models,* 1997, Lecture Notes in Logic, vol. 9, Springer-Verlag.

[Pap94] C. Papadimitriou, *Computational Complexity* 1994, Addison-Wesley.

[Pap85] C. Papadimitriou, “A Note on the Expressive Power of Prolog,” *EATCS Bulletin 26* (1985), 21-23.

[PY91] C. Papadimitriou and M. Yannakakis, “Optimization, Approximation, and Complexity Classes,” *J. Comput. Sys. Sci.* , 43 (1991), 425-440.

[PI94] S. Patnaik and N. Immerman, “Dyn-FO: A Parallel, Dynamic Complexity Class,” *J. Comput. Sys. Sci.* 55(2) (1997), 199-209. A preliminary version appeared in *ACM Symp. Principles Database Systems* (1994), 210-221.

[Poi82] B. Poizat, ”Deux ou trois choses que je sais de Ln” *JSL* 47 (1982), 641-658.

[R96] G. Ramalingam *Bounded Incremental Computation,* 1996, Springer LNCS 1089.

[Raz87] A. Razborov, “Lower Bounds on the Size of Bounded Depth Networks Over a Complete Basis With Logical Addition,” *Matematischi Zametki* 41 (1987), 598-607 (in Russian). English translation in *Mathematical Notes of the Academy of Sciences of the USSR* 41, 333-338.

[Rei87] J. Reif, “On Threshold Circuits and Polynomial Computation,” *Second Annual Structure in Complexity Theory Symp.* (1987), 118-123.

[Rei05] O. Reingold, “Undirected ST-Connectivity in Log-Space”, *ACM Symp. Theory Of Comput.* 2005, 376 - 385.

[RS72] D. Rödding and H. Schwichtenberg, "Bemerkungen zum Spektralproblem," *Zeitschrift fr̈ math. Logik und Grundlagen der Mathematik* 18 (1972), 1-12.

[Ros82] J. Rosenstein, *Linear Orderings,* 1982, Academic Press.

[Ruz81] L. Ruzzo, "On Uniform Circuit Complexity," *J. Comp. Sys. Sci.,* 21, No. 2 (1981), 365-383.

[Sav70] W. Savitch, "Relationships Between Nondeterministic and Deterministic Tape Complexities," *J. Comput. System Sci.* 4 (1970), 177-192.

[Sav73] W. Savitch, "Maze Recognizing Automata and Nondeterministic Tape Complexity," *J. Comput. Sys. Sci.* 7 (1973), 389-403.

[Sch97] N. Schweikardt, "The Monadic Quantifier Alternation Hierarchy over Grids and Pictures," *Annual Conference of the European Association for Computer Science Logic* (1997), M. Nielsen and W. Thomas, eds., 383-397.

[Sch94] T. Schwentick, "Graph Connectivity and Monadic NP," *IEEE Found. of Comp. Sci. Symp.* (1994), 614-622.

[Sch97a] T. Schwentick, "Padding and the Expressive Power of Existential Second-Order Logics," *Annual Conference of the European Association for Computer Science Logic* (1997), M. Nielsen and W. Thomas, eds., 399-412.

[SB98] T. Schwentick and K. Barthelmann, "Local Normal Forms for First-Order Logic with Applications to Games and Automata," to appear in *Symp. Theoretical Aspects Comp. Sci.* (1998).

[See95] D. Seese, "FO-Problems and Linear Time Computability," Tech Report, Institut für Informatik und Formale Beschreibungsverfahren, Universität Karlsruhe, Germany (1995).

[Sip83] M. Sipser, "Borel Sets and Circuit Complexity," *15th Symp. on Theory of Computation* (1983), 61-69.

[Smo87] R. Smolensky, "Algebraic Methods in the Theory of Lower Bounds for Boolean Circuit Complexity," *19th ACM STOC* (1987), 77-82.

[Spe93] J. Spencer, "Zero-One Laws With Variable Probability," *J. Symbolic Logic* 58 (1993), 1–14.

[Ste94] I. Stewart, “On completeness for NP via projection translations,” Math. Syst. Theory 27 (1994), 125–157.

[Ste91] I. Stewart, “Comparing the Expressibility of Languages Formed Using NP-Complete Operators”, *J. Logic and Computation* 1(3) (1991), 305-330.

[Sto77] L. Stockmeyer, “The Polynomial-Time Hierarchy,” *Theoretical Comp. Sci.* 3 (1977), 1-22.

[SV84] L. Stockmeyer and U. Vishkin, “Simulation of Parallel Random Access Machines by Circuits,” *SIAM J. of Comp.* 13, No. 2 (1984), 409-422.

[Str94] H. Straubing, *Finite Automata, Formal Logic, and Circuit Complexity,* 1994, Birkhäuser.

[Sze88] R. Szelepcsényi, “The Method of Forced Enumeration for Nondeterministic Automata,” *Acta Informatica* 26 (1988), 279-284.

[Tar36] A. Tarksi, “Der Wahrheitsbegriff in den Formalisierten Sprachen,” *Studia Philosophica 1* (1936).

[Tar55] A. Tarksi, “A Lattice-Theoretical Fixpoint Theorem and its Applications,” *Pacific. J. Math.*, 55 (1955), 285-309.

[Tho86] S. Thomas, “Theories With Finitely Many Models,” *J. Symbolic Logic,* 51, No. 2 (1986), 374-376.

[Tra50] B. Trahtenbrot, “The Impossibility of an Algorithm for the Decision Problem for Finite Domains,” *Doklady Academii Nauk SSSR*, n.s., vol 70 (1950), 569-572 (in Russian).

[Tur84] G. Turán, “On the Definability of Properites of Finite Graphs,” *Discrete Math. 49* (1984), 291-302.

[TPP97] A. Turk, S. Probst, and G. Powers, “Verification of a Chemical Process Leak Test Procedure,” in *Computer Aided Verification, 9th International Conf.*, O. Grumberg, ed. 1997, Springer, 84-94.

[Tys97] J. Tyszkiewicz, “The Kolmogorov Expression Complexity of Logics,” *Information and Computation* 135(2) (1997), 113-136.

[Vaa99] . Väänänen, ed., *Generalized Quantifiers and Computation,* Ninth European Summer School in Logic, Language, and Information, 1997, Springer LNCS 1754.

[Val82] L. Valiant, “Reducibility By Algebraic Projections,” *L’Enseignement mathématique,* 28, 3-4 (1982), 253-68.

[vD94] D. van Dalen, *Logic and Structure, Third Edition,* 1994, Springer-Verlag.

[vH67] J. van Heijenoort, *From Frege to Gödel: A Source Book in Mathematical Logic, 1879 - 1931* 1967, Harvard University Press.

[Var82] M.Y. Vardi, “Complexity of Relational Query Languages,” *14th Symposium on Theory of Computation* (1982), 137-146.

[Wei76] B. Weisfeiler, ed., *On Construction and Identification of Graphs,* Lecture Notes in Mathematics 558, Springer, 1976.

[Wra76] C. Wrathall, “Complete Sets and the Polynomial Hierarchy,” *Theoret. Comp. Sci.* 3 (1976).

[Yao85] A. Yao ,“Separating the Polynomial-Time Hierarchy by Oracles,” *26th IEEE Symp. on Foundations of Comp. Sci.* (1985), 1-10.