

MONOGRAPHS IN COMPUTER SCIENCE

PARAMETERIZED COMPLEXITY

R.G. Downey

M.R. Fellows



Springer

Monographs in Computer Science

Editors

David Gries
Fred B. Schneider

Springer Science+Business Media, LLC

Monographs in Computer Science

Abadi and Cardelli, **A Theory of Objects**

Brzozowski and Seger, **Asynchronous Circuits**

Selig, **Geometrical Methods in Robotics**

Nielson [editor], **ML with Concurrency**

Castillo, Gutiérrez, and Hadi, **Expert Systems and Probabilistic Network Models**

Paton [editor], **Active Rules for Databases**

Downey and Fellows, **Parameterized Complexity**

R.G. Downey
M.R. Fellows

Parameterized Complexity

With 68 Figures



Springer

R.G. Downey
Victoria University, Wellington
Department of Mathematics
Wellington, New Zealand

M.R. Fellows
University of Victoria, Victoria
Department of Computer Science
Victoria, V8W 3P4
Canada

Series Editors:

David Gries
Department of Computer Science
Cornell University
Upson Hall
Ithaca, NY 14853-7501
USA

Fred B. Schneider
Department of Computer Science
Cornell University
Upson Hall
Ithaca, NY 14853-7501
USA

Library of Congress Cataloging-in-Publication Data

Downey, R. G. (Rod G.)

Parameterized complexity / R.G. Downey, M.R. Fellows.

p. cm. — (Monographs in computer science)

Includes index.

ISBN 978-1-4612-6798-0

ISBN 978-1-4612-0515-9 (eBook)

DOI 10.1007/978-1-4612-0515-9

1. Computational complexity. I. Fellows, M. R. II. Title.

III. Series.

QA267.7.D68 1997

511.3—dc21

97-22882

Printed on acid-free paper.

© 1999 Springer Science+Business Media New York

Originally published by Springer-Verlag New York, Inc. in 1999

Softcover reprint of the hardcover 1st edition 1999

All rights reserved. This work may not be translated or copied in whole or in part without the written permission of the publisher Springer Science+Business Media, LLC, except for brief excerpts in connection with reviews or scholarly analysis. Use in connection with any form of information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed is forbidden.

The use of general descriptive names, trade names, trademarks, etc., in this publication, even if the former are not especially identified, is not to be taken as a sign that such names, as understood by the Trade Marks and Merchandise Marks Act, may accordingly be used freely by anyone.

Production managed by Allan Abrams; manufacturing supervised by Jeffrey Taub.

Camera-ready copy prepared from the authors' LaTeX files.

9 8 7 6 5 4 3 2 1

ISBN 978-1-4612-6798-0

Preface

The idea for this book was conceived over the second bottle of Villa Maria's Cabernet Merlot '89, at the dinner of the Australasian Combinatorics Conference held at Palmerston North, New Zealand in December 1990, where the authors first met and discovered they had a number of interests in common. Initially, we embarked on a small project to try to formulate reductions to address the apparent parameterized intractability of **DOMINATING SET**, and to introduce a structure in which to frame our answers. Having spent several months trying to get the definitions for the reductions right (they now seem so obvious), we turned to our tattered copies of Garey and Johnson's work [239]. We were stunned to find that virtually none of the classical reductions worked in the parameterized setting. We then wondered if we'd be able to find any interesting reductions.

Several years, many more bottles, so many papers, and reductions later it [3] seemed that we had unwittingly stumbled upon what we believe is a truly central and new area of complexity theory. It seemed to us that the material would be of great interest to people working in areas where exact algorithms for a small range of parameters are natural and useful (e.g., Molecular Biology, VLSI design). The tractability theory was rich with distinctive and powerful techniques. The intractability theory seemed to have a deep structure and techniques all of its own. We also felt that the subject was mathematically beautiful and sufficiently mature that we should gather the material for the present monograph. The project turned out to be bigger than we had expected.

We have tried to make the material as self-contained as possible, while keeping the book's length relatively finite. In some cases, such as tree automata, we felt that there was no adequate text and hence a lot of background material is included. Also, we have tried to make the book accessible to several distinct audiences, which

seemed to suggest rather more background material. We particularly tried to make the text accessible to graduate students. There are a large number of exercises included. They have varying levels of difficulty, and some are research results.

Anyway the book is as it is. We hope you enjoy the result.

Acknowledgements

The authors have had a lot of support for this venture. Downey was supported by a U.S./N.Z. Cooperative Science grant INT-9020558, the ARO through the Mathematical Sciences Institute in Ithaca, New York via DAAL-03-C-0027, and Cornell University as a Visiting Scholar in 1992 and 1995. Downey especially acknowledges the generous support of the New Zealand Marsden Fund for Basic Science. Fellows acknowledges the generous, flexible, and reliable support of the National Science and Engineering Research Council of Canada.

Each of the authors have had exceptional support and encouragement from their home universities and departments. Victoria University of Wellington has provided Downey with numerous IGC grants and the special grant RGNT 779.

We would like to thank our various co-workers in the Parameterized World: Karl Abrahamson, Hans Bodlaender, Leizhen Cai, Liming Cai, Jianer Chen, Bruce Kapron, Neil Koblitz, Mike Langston, Venkatesh Raman, Ken Regan, and Igor Shparlinski. A special thanks to Mike's students Kevin Cattell, Marco Cesati, Mike Dinneen, Patricia Evans, Mike Hallett, Todd Wareham, and Xiu Yan Liu for providing stimulation in this area and especially to Todd Wareham and Mike Hallett who gave extensive help with the preparation of the Compendium. Also, special thanks to Downey's student Mike Doyle who provided very extensive and detailed corrections, and provided many of the diagrams.

Thanks to Edith Hodgen who struggled with the final preparation.

We'd also like to thank the following who provided corrections to the various drafts of the book: Ivan Murray, David Gresham, Alistair McGlinchy, Joseph Ganley, Richard Coles, Kristin Downey, and Geoffery LaForte.

Thanks of course to Martin Gilchrist, Alan Abrams, and Fred Bartlett at Springer-Verlag.

Finally, we'd like to take this chance to sincerely thank our long suffering families who survived our frequent absences and put up with two aging but still relatively able surfers. We dedicate the book to them. Downey also dedicates it to the memory of his father, Reginald Andrew Downey 1926–1994.

Contents

Preface	v
Acknowledgements	vii
List of Figures	xiii
1 Computers, Complexity, and Intractability from the Parametric Point of View	1
1.1 Introduction	1
1.2 The Role of Computational Complexity in Modern Science . .	2
1.3 The Story of Dr. $\mathcal{O}$, Continued	5
1.4 Reworking the Foundations of Computational Complexity . . .	6
1.5 A Deal with the Devil	7
1.6 How Parameters Arise in Practice	8
1.7 A Distinctive Positive Toolkit	11
1.8 $\mathcal{O}$ No?	13
1.9 The Barometer of Parametric Intractability	14
1.10 Structural Aspects of Parameterized Complexity	18
1.11 An Overview of Current Research Horizons	19
I Parameterized Tractability	21
2 The Basic Definitions	23
2.1 Fixed-Parameter Tractability	23

2.2	The Advice View	26
3	Some Ad Hoc Methods: The Methods of Bounded Search Tree and Problem Kernel	29
3.1	The Method of Bounded Search Trees	29
3.1.1	The Basic Method	29
3.1.2	Heuristic Improvements, Shrinking the Search Tree . .	35
3.2	The Method of Reduction to a Problem Kernel	39
3.2.1	The Basic Method	39
3.2.2	Hereditary Properties and Leizhen Cai's Theorem . . .	43
4	Optimization Problems, Approximation Schemes, and Their Relation with FPT	49
4.1	Optimization Problems	49
4.2	How FPT and Optimization Problems Relate	50
4.3	The Classes $MAXSNP$, $MIN F^+ \Pi_1(h)$, and FPT	54
5	The Advice View Revisited and $LOGSPACE$	61
6	Methods via Automata and Bounded Treewidth	67
6.1	Classical Automata Theory	67
6.1.1	Deterministic Finite Automata	67
6.1.2	Nondeterministic Finite Automata	69
6.1.3	Regular Languages	76
6.1.4	The Myhill–Nerode Theorem and the Method of Test Sets	80
6.1.5	Classical Tree Automata	89
6.2	Treewidth	99
6.3	Bodlaender's Theorem	105
6.4	Parse Trees for Graphs of Bounded Treewidth and an Analog of the Myhill–Nerode Theorem	113
6.5	Courcelle's Theorem	130
6.5.1	The Basic Theorem	130
6.5.2	Implementing Courcelle's Theorem	136
6.6	Seese's Theorem	138
6.7	Notes on MS_1 Theory	140
7	Well-Quasi-Orderings and the Robertson–Seymour Theorems	143
7.1	Basic WQO Theory	143
7.2	Thomas' Lemma	156
7.2.1	Thomas' Lemma Fails for Path Decompositions	164
7.3	The Graph Minor Theorem for Bounded Treewidth	169
7.4	Excluding a Forest	178
7.5	Connections with Automata Theory and Boundaried Graphs . .	181
7.6	A Sketch of the Proof of the Graph Minor Theorem	189

7.7	Immersions and the Nash-Williams Conjecture	192
7.8	Applications of the Obstruction Principle and wQO 's	194
7.9	Effectivizations of Obstruction-Based Methods	201
7.9.1	Effectivization by Self-Reduction	202
7.9.2	Effectivization by Obstruction Set Computation	205
8	Miscellaneous Techniques	211
8.1	Depth-First Search	211
8.2	Bounded-Width Subgraphs, the Plehn-Voigt Theorem, and Induced Subgraphs	215
8.3	Hashing	220
II	Parameterized Intractability	225
9	Reductions	227
10	The Basic Class $W[1]$ and an Analog of Cook's Theorem	235
11	Some Other $W[1]$-Hardness Results	255
12	The W-Hierarchy	283
13	Beyond $W[t]$-Hardness	315
14	Fixed Parameter Analogs of $PSPACE$ and k-Move Games	331
15	Provable Intractability: The Class XP	341
III	Structural and Other Results	351
16	Another Basis for the W-Hierarchy, the Tradeoff-Theorem, and Randomized Reductions	353
17	Relationships with Classical Complexity and Limited Nondeterminism	363
17.1	Classical Complexity	363
17.2	Nondeterminism in P , $LOGNP$, and the Cai-Chen Model and Other Models	369
18	The Monotone and Antimonotone Collapse Theorems: $MONOTONE$ $W[2t + 1] = W[2t]$ and $ANTIMONOTONE$ $W[2t + 2] = W[2t + 1]$	377
19	The Structure of Languages Under Parameterized Reducibilities	389
19.1	Some Tools	389
19.2	Results	415

IV Appendix 439

A	A Problem Compendium and Guide to W-Hierarchy Completeness, Hardness, and Classification; and Some Research Horizons	441
A.1	In <i>FPT</i>	441
A.2	In <i>FPT</i> (Nonuniform)	451
A.3	In Randomized <i>FPT</i>	452
A.4	$W[1]$ -Complete	453
A.5	$W[1]$ -Hard, in $W[2]$	458
A.6	$W[1]$ -Hard, in $W[P]$	460
A.7	$W[1]$ -Hard	460
A.8	$W[2]$ -Complete	463
A.9	$W[2]$ -Hard, in $W[P]$	465
A.10	$W[2]$ -Hard	466
A.11	$W[t]$ -Complete	467
A.12	$W[t]$ -Hard, for All t , in $W[P]$	468
A.13	$W[t]$ -Hard, for All t	468
A.14	$W[SAT]$ -Hard	473
A.15	$W[P]$ -Complete	473
A.16	$AW[*] = AW[1] = AW[t]$ -Complete	476
A.17	$AW[SAT]$ -Complete	478
A.18	$AW[SAT]$ -Hard	478
A.19	$AW[P]$ -Complete	479
A.20	$AW[P]$ -Hard	480
A.21	XP -Complete	480
B	Research Horizons	481
B.1	A Lineup of <i>FPT</i> Suspects	481
B.2	A Lineup of Tough Customers	483
B.3	Connections Between Classical and Parameterized Complexity	486
B.4	Classification Gaps	486
B.5	Structural Issues and Analogs of Classical Results	487
B.6	The <i>FPT</i> Toolkit and Further Business with the Devil	487
	References	489
	Index	517

List of Figures

1.1	The Problem of Cleaning up the Data	2
1.2	Classically Considered, Most Problems Yield a Seemingly Unavoidable Combinatorial Explosion	6
1.3	Parameterized Complexity Attempts to Confine the Combinatorial Explosion	7
1.4	Lichenism	13
1.5	Three Vertex Set Problems	14
1.6	Getting a Grip on Intractability	19
2.1	Examples of <i>FPT</i> Problems	25
3.1	DOMINATING SET	31
3.2	The Cases of Termination for ALMOST (x)	33
3.3	Increasing the Number of Leaves	42
6.1	Transition Diagram for Example 6.2	68
6.2	Transition Diagram for Example 6.5	70
6.3	The Deterministic Automaton for Example 6.5	72
6.4	A Nondeterministic Automaton with λ Moves	73
6.5	The Automaton Corresponding to Example 6.9	74
6.6	Case (b) of Lemma 6.17	78
6.7	Construction of an Automaton Accepting L	79
6.8	The Resulting Automaton	85
6.9	The Minimization Algorithm	86
6.10	Automaton for Exercise 6	88

6.11	An Example of a Tree	89
6.12	Gluing Labeled Trees	91
6.13	Computation of a Tree Evaluation	92
6.14	An Example of Productions and a Derivation Based on These Productions	94
6.15	Examples of 2- and 3-Trees	100
6.16	Example of Tree Decomposition of Width 2	101
6.17	Figure for Exercise 2	104
6.18	Composition Operators and Their Effects	114
6.19	Parsing Graphs of Treewidth 2	115
6.20	Normalizing the Tree Decomposition	117
6.21	Partial Parsing of the Normalized Decomposition	118
6.22	Replacement for the Myhill–Nerode Theorem	122
6.23	Graphs for BANDWIDTH	124
6.24	Test Set of Graphs for HAMILTONICITY for Treewidth 3	126
6.25	Constrained Layout for CUTWIDTH	128
6.26	Test Sets for Atomic Formulae on Three Boundaried Graphs	137
6.27	Examples of Grids	139
7.1	Kuratowski’s Obstructions for Planarity	147
7.2	A Contraction and a Topological Embedding	150
7.3	Construction of the New Tree of Bags	159
7.4	The Coles Graph	164
7.5	A Path Decomposition of G	164
7.6	Embedding Pathwidth t Tree Obstructions, Tree- t , in Binary Trees.	185
7.7	Path Decomposition of Width 5	188
7.8	Some Graphs Without a P_6 Minor	189
7.9	The Immersion Ordering	193
7.10	Obstructions for Interval Graphs	195
7.11	A 2-Fold Cover of K_5	197
8.1	An Example of a Depth-First Spanning Tree	212
8.2	A Circus Graph of Size 4	214
8.3	Graph for Exercise 2	214
10.1	Examples of Circuits	238
10.2	Gadget for RED/BLUE NONBLOCKER	245
11.1	Overview of INDEPENDENT SET $\leq_m^s$ PERFECT CODE	279
11.2	Example of INDEPENDENT SET $\leq_m^s$ PERFECT CODE	280
12.1	Gadget for CNFSAT Reducing to DOMINATING SET	286
12.2	A Topmost Layer of Large Or Gates.	297
13.1	Gadgets for DEGREE 3 SUBGRAPH ANNIHILATOR	324

13.2	Gadgets for Exercise 1	328
13.3	Gadget for WEIGHTED PLANAR CIRCUIT SATISFIABILITY . . .	329
14.1	Gadgets for SHORT GEOGRAPHY	338
15.1	Pebble Moves Where $X = F(X_{-1}, X_0, X_1)$, $X \in (Q \times \Gamma) \cup \Gamma$, $d \in \{-1, 0, 1\}$. Initially a Pebble Is on $[q_f b]$ and $t = n^k, i = 1$.	344
15.2	A Set of Rules, Where a, b , and c Are Nodes and $P(a, c)$ Is a Condition	344
15.3	Rules for the Pebble Game	346
15.4	More Rules for the Pebble Game	347
17.1	The Even Case of Theorem 17.6(a)	367
17.2	The Algorithm $\text{CSAT}(t, h)$ -SIMULATOR	372
19.1	The Arithmetical Hierarchy	390
19.2	The Assignment of Priorities and the Outcomes	399
19.3	The Priority Tree for Theorem 19.23	422

1

Computers, Complexity, and Intractability from the Parametric Point of View

1.1 Introduction

The subject of this book is perhaps best introduced through a somewhat whimsical story.

Imagine that a scientist, Dr. $\mathcal{O}$, has collected a number of data points that support a new theory of some sort and is nearly ready to publish. Dr. $\mathcal{O}$ observes that some pairs of these observations are in conflict, although most are not. The problem naturally arises of whether there is a way to determine a minimum set of observations in the data set that would “explain” all of the inconsistencies and that perhaps deserve further consideration.

Because some sophistication about computational matters is now called for in most sciences, Dr. $\mathcal{O}$ has a nodding familiarity with the subject of computational complexity and even owns a copy of the classic book by Garey and Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness* [239]. After a bit of thought, Dr. $\mathcal{O}$ realizes that the problem can be modeled by “dots and lines.” The data points are represented by dots, and a conflict between two data points is represented by a line between the two corresponding dots. The problem is then to come up with a minimum number of dots whose removal would eliminate all the conflicts along the lines of Figure 1.1. Consulting with the computer scientist who is a member of the interdisciplinary research group to which Dr. $\mathcal{O}$ belongs, Dr. $\mathcal{O}$ is informed that this is indeed a well-known problem that goes by the name of **VERTEX COVER** and is *NP*-complete. In fact, it is one of the six famous and basic *NP*-complete problems singled out for attention by Garey and Johnson.

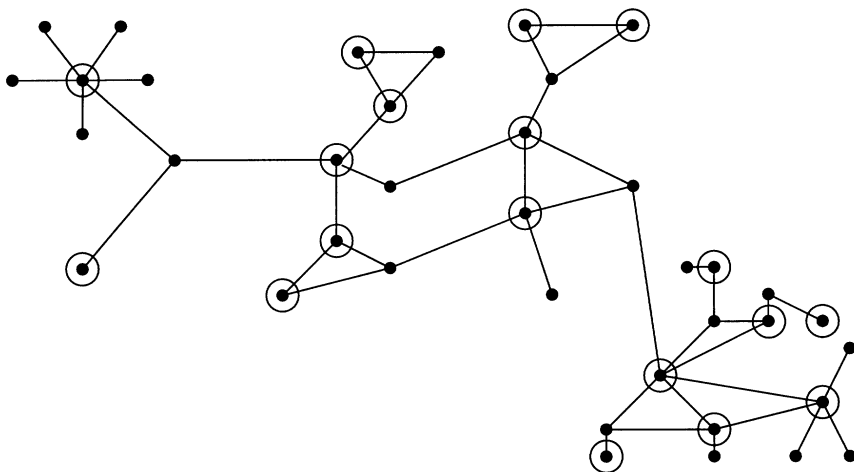


FIGURE 1.1. The Problem of Cleaning up the Data

“But my data set is pretty good!” laments Dr. $\mathcal{O}$. “There are probably only 40 or 50 troublesome data points, maybe only 20 . . .”

Apart from the fact that the problem is *NP*-complete (and hard to approximate! [41]), all that classical computational complexity has to offer is that for every *fixed* value of k , there is a polynomial time algorithm to determine whether removing k points suffices to clean up the data set—namely, the trivial brute-force algorithm of trying all k subsets. Indeed, knowing that classically the problem is *NP*-complete would seem to suggest that the *only* approach would be exhaustive search.

The running time of this exhaustive search algorithm is roughly proportional to n^k , where n is the size of the data set. If we assume that Dr. $\mathcal{O}$ has between 500 and 1000 data points, then this brute-force algorithm is impractical when k is more than about 5, since the number of steps would then exceed 10^{15} , and few realistic computations involve that much effort.

1.2 The Role of Computational Complexity in Modern Science

Computational complexity is often regarded, even by computer scientists (including the majority, who are not theoretical computer scientists and whose concerns are typically quite practical), as a mathematically forbidding, isolated, and even largely irrelevant subdiscipline. It is our view, however, that the subject of computational complexity has a profound and even central role in modern science.

The reason for this is that theories of computational complexity frame and focus the efforts of algorithm designers in certain directions (e.g., the quest for polynomial-time algorithms) and often discourage the investment of effort both on problems that are identified as “inherently intractable” and ones that are formally

regarded as asymptotically well-solved and therefore uninteresting. What efforts are put toward the “inherently intractable” for the most part could be classified as trying to establish that they are provably hard. Since efficient algorithms are vastly more important than good hardware in the grand scheme of things, the effects of this framing percolate through all of science via the tools that are eventually supplied to the other disciplines by computer science.

It is worthwhile keeping in mind that intractability must be coped with and problems must be solved in practical applications. Furthermore, it is important to keep in sight the fact that in applications, we are only dealing with a finite fragment of the “asymptotic universe,” so asymptotic classes such as P only approximate practical feasibility in the same way that classes like NP and $PSPACE$ approximate practical intractability. From an engineering point of view, there are certainly approaches to complexity other than the classical view which can yield significant insight and applications for practical problems.

The point of view of the authors is that, as expressed by the motto of the former *Natural's Gas and Garage*¹ on Shattuck Ave. in Berkeley, California, the role of computational complexity is to “Serve the Community.” The framework of computational complexity focuses efforts on particular goals and encourages the development of a corresponding toolkit of algorithmic ideas. These are used not only in the pursuit of the declared objectives but also in the design of heuristic and approximation algorithms for problems that are hard, but for which, nevertheless, something must be done.²

The central concept in the modern study of computational complexity is the notion of asymptotic polynomial-time complexity. As a model (in the sense of empirical science) of computational feasibility, this idea must be regarded as one of the most successful scientific models of all time. What this book is all about is a *practical generalization* of this basic idea, but one that requires a fundamental reworking of the foundations.

A computational problem is said to be solvable in *polynomial time* if there is an algorithm that, given any input x of size n (where size is normally measured as the length of a reasonably succinct description of the input), produces the output required by the problem specification in time $O(n^\alpha)$, that is, in time bounded by Cn^α for some constants α and C , whenever n is large enough, $n \geq n_0$.

Despite the enormous latitude for “misbehavior” that this definition allows—e.g., is an $O(n^{100})$ algorithm really practical?—such misbehaviour almost never occurs. One common type of exception to this general rule is the kind of $O(n^k)$ “try all k subsets” algorithm described above for the **DATA CLEANING** problem.

The numerical basis for the success of the notion of polynomial-time complexity as a model of computational feasibility is evident in the phenomenon of “combinatorial explosion” that haunts problems for which no good algorithm is known.

¹Now called *Motor City* and operated by Don Cartwright. Recommended.

²Some discussion of broad trends in theoretical computer science and the issue of the larger scientific role of computational complexity theory can be found in Hartmanis [262] and Hartmanis and Lin [263].

Polynomial		Exponential	
Complexity	Running time	Complexity	Running time
n^2	25 μ sec	1.5^n	6 min
n^3	1 min	2^n	130 days
n^5	3 sec	3^n	230(10^6) years

TABLE 1.1. Running Times of Several Algorithms on a Size $n = 50$ Instance with 10^8 Operations per Second

Table 1.1 illustrates the tremendous qualitative difference between polynomial and exponential running times.

As a concrete example of the problem of combinatorial explosion, suppose that our computational task is to put a deck of cards in order (the problem of **SORTING**) and that the only algorithm we know consists of repeatedly shuffling the deck, and after each shuffle, checking to see whether the cards are in order. If there are only three or four cards in the deck this algorithm will work OK. Now imagine that we run this algorithm on a vast parallel computer architecture consisting of supercomputer processors, each doing a billion shuffles per second, with one such processor for every elementary particle in the universe, and the whole ensemble of powerful hardware running for the lifetime of the universe. Now, how many cards could we expect to be able to deal with using this algorithm? The answer is: about 52.

Polynomial-time complexity has been a successful, well-behaved model of feasible computability because when one asks about the complexity of a “natural” computational problem, there are two outcomes that cover most cases:

- (1) The problem can be solved in time $O(n^3)$. The folklore of algorithm design would indicate that probably with a bit more effort this can be improved to $O(n \log n)$.
- (2) The problem appears to be hopelessly intractable. It is *NP*-complete or worse, and the only known algorithms are about as efficient as the “get lucky” procedure for sorting.

In other words, when we can do anything clever at all, we can usually achieve polynomial-time complexities with small exponent polynomials. This can be regarded as a fascinating *empirical* fact about the “natural” world of computational complexity.

Most theoretical computer scientists favor some form of the conjecture that problems of the kind (2) are indeed *inherently* difficult and simply do not admit efficient algorithms, although this is far from proved. The last 20 years have seen remarkably little progress toward a proof of the $P \neq NP$ conjecture, which is one formalization of this intuition and widely regarded as the most important unsolved problem of computer science and the foundations of mathematics.

1.3 The Story of Dr. $\mathcal{O}$, Continued

As Dr. $\mathcal{O}$ has discovered in consultation with the computer scientist in her research group, **DATA CLEANING** is one of the many problems that have been classified as *NP*-complete. But luckily, in this case, there is more to the story. In fact, there is the perspective of *this book*.

Progress since Garey and Johnson [239] on the **DATA CLEANING** (a.k.a. **VERTEX COVER**) problem has unfolded in the following way.

- In 1986, Fellows and Langston (published in [223]) observed that the powerful Robertson-Seymour graph minors theory can be invoked with no effort to conclude that for each fixed value of k , the k -**VERTEX COVER** problem can be solved in time $O(f(k)n^3)$, where n is the number of data points. (More about this very powerful and mathematically exciting complexity classification machinery can be found in Chapter 7.) The fact that this result concerning **VERTEX COVER** was news *at all* reflects, perhaps, a kind of neglect of this sort of question (“How well can we do when k is fixed?”).
- In 1987, Johnson observed that an $O(f(k)n^2)$ algorithm for k -**VERTEX COVER** could be based on a combination of tree-decomposition and finite-state dynamic programming techniques [281]. More about these general methods can be found in Chapter 6.
- In 1988, Fellows [204] gave an algorithm for the k -**VERTEX COVER** problem with a running time of $O(2^k n)$ based on a search tree technique. This simple but important idea is described in Chapter 3.
- In 1989, S. Buss [100] described an algorithm with running time $O(kn + 2^k k^{2k+2})$ based on the method of reduction to a problem kernel, described in Chapter 3. This was developed as a general method by Downey and Fellows [169].
- In 1992, by combining the the ideas in [204] and [100], R. Balasubramanian, R. Downey, M. Fellows and V. Raman in [45] (reported in R. Balasubramanian, M. Fellows and V. Raman [46]) described an improved algorithm with running time $O(kn + 2^k k^2)$.
- In 1993, Papadimitriou and Yannakakis [368] used maximum matching as a subroutine in an algorithm with running time $O(3^k n)$ and observed that this yields a polynomial-time algorithm for **VERTEX COVER** whenever k is $O(\log n)$. (In the context of computational biology, this algorithm was rediscovered by Bonet et al. in 1998 [91].)
- In 1996, Balasubramanian et al. [46] described an algorithm with running time $O(kn + (4/3)^k k^2)$, based on a combination and refinement of previous techniques.

Thanks to these developments, Dr. $\mathcal{O}$ finds that the **DATA CLEANING** problem has a good practical algorithm for data sets of *any size*, so long as the number k of “bad points” is no more than about 70.

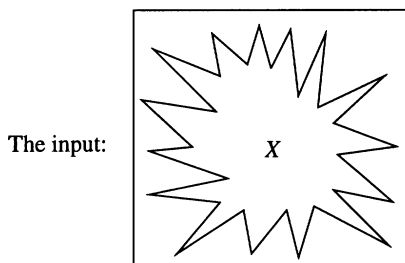


FIGURE 1.2. Classically Considered, Most Problems Yield a Seemingly Unavoidable Combinatorial Explosion

1.4 Reworking the Foundations of Computational Complexity

Perhaps the single most profound fact that has been uncovered by computer science so far is the *ubiquity of computational intractability*. Literally thousands of natural problems have now been shown to be *NP*-complete or worse. If we had to describe the situation in a picture, it might be Figure 1.2, which shows the situation for a generic computational problem Π . The input to Π is x , and the figure depicts the apparently inevitable combinatorial explosion, since, as usual, Π is *NP*-complete or worse.

The formal setup of classical computational complexity theory can be quickly sketched as follows. First of all, to keep things simple, the theory is about computational problems which receive an input x about which a yes/no question is asked. The input is described as a string over some convenient fixed alphabet. If we let Σ^* denote the set of all possible strings over this alphabet, and $L_\Pi \subseteq \Sigma^*$ denote the set of all strings x that describe an input for which the answer is “yes,” then we have the formal foundation of classical complexity in *language recognition*. Given the input x , the problem is to determine whether $x \in L_\Pi$, that is, whether x belongs to the language of yes instances of the problem.

The question is then whether there is an efficient algorithm to recognize the strings that belong to the language L_Π . The problem is that the answer is generally “no.” Among the bad news uncovered in the first decades of computer science is that for many natural problems Π , the language L_Π cannot be recognized by *any* algorithm and that for thousands of other innocuous-seeming problems, polynomial-time recognition algorithms for L_Π probably do not exist.

The bad news concerning the ubiquity of computational intractability continues to deepen. One of the most profound developments of the 1990s, based on new and sophisticated algebraic methods, has been a host of results showing that most of the *NP*-hard problems that can be formulated as optimization problems *also* probably do not have good approximation algorithms [40].

What is a practical scientist to do?

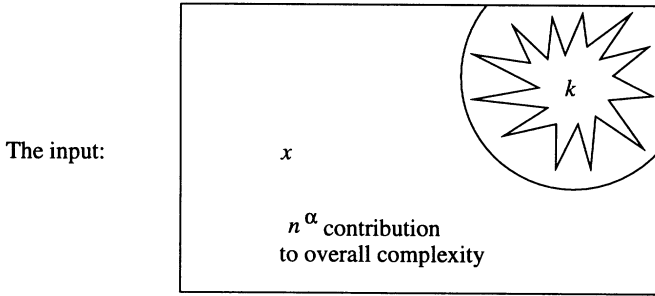


FIGURE 1.3. Parameterized Complexity Attempts to Confine the Combinatorial Explosion

1.5 A Deal with the Devil

Parameterized complexity is based on a deal with the devil of intractability such as illustrated in Figure 1.3. We regard as a worthy goal a combination of “good” and “bad” complexity behavior, where the main part of the problem contributes to the overall complexity in a good way (e.g., polynomially), but some aspects of the input (the parameter) may contribute poorly to problem complexity. We regard this outcome as acceptable, since we are hoping to confine the parameter to some small but still useful range.

As illustrated in Figure 1.3, the deal we attempt to cut with the devil is one that confines the “inevitable” combinatorial explosion (which we *expect* for interesting problems) to some limited aspect (or distribution) of the problem.

At this point, it is worth remembering the happy outcome of the story of Dr. $\mathcal{O}$. Since Dr. $\mathcal{O}$ is concerned only with parameter values k in the range $k \leq 50$, we have been able to deal entirely and completely with the *NP*-completeness of the **DATA CLEANING** problem, for input data sets of unlimited size. The moral of the story is that classical complexity frameworks such as *NP*- and *PSPACE*-completeness are not adequate to address intractability questions for problems that are parameterized, and for which important applications are covered by small ranges of the parameter.

The formal set-up of parameterized complexity theory is based on the following reworking of the foundational definitions. We consider the input to a problem as consisting of two parts, and thus as a pair of strings $(x, y) \in \Sigma^* \times \Sigma^*$. We say that y is the *parameter*. As in the classical set-up, it all boils down to recognizing the yes instances. We define a *parameterized language* to be $L \subseteq \Sigma^* \times \Sigma^*$.

This is what makes parameterized complexity fundamentally different: The input is considered to be two-part, and the target of our efforts is a viable dialectical combination of “good” and “bad.” We regard this formulation as an inevitable response to the fundamental empirical fact of ubiquitous computational intractability in the world of natural computational problems.

In the same way that the notion of *polynomial time* is central to the classical formulation of computational complexity, central to parameterized complexity is

the notion of *fixed-parameter tractability*. Here, we are considering languages that are *tractable by the slice*.

Definition 1.1 (Fixed Parameter Tractability) A parameterized problem $L \subseteq \Sigma^* \times \Sigma^*$ is *fixed-parameter tractable* if there is an algorithm that correctly decides, for input $(x, y) \in \Sigma^* \times \Sigma^*$, whether $(x, y) \in L$ in time $f(k)n^\alpha$, where n is the size of the main part of the input x , $|x| = n$, k is the parameter which we can take to be the length of y , $k = |y|$, α is a constant (independent of k), and f is an arbitrary function.

What the above definition accomplishes in generalizing polynomial time is the following:

- (1) The complexity contribution of the parameter part of the input is turned over to the devil, who contributes to the overall complexity complexity of the problem the arbitrary function $f(k)$.
- (2) The main part x of the input to the problem contributes to the overall complexity polynomially as n^α .
- (3) The connection between the two contributions is specified to be multiplicative: The overall complexity is $f(k)n^\alpha$.

One of the interesting and fundamental properties of fixed-parameter tractability is that the definition is *unchanged* if we replace (3) with

- (3') The connection between the two contributions is additive: The overall complexity is $f(k) + n^{\alpha'}$.

More about this can be found in Sections 2.1 and 2.2.

1.6 How Parameters Arise in Practice

In parameterized complexity, the focus is not on *whether* a problem is hard—the theory starts from the assumption that most interesting problems *are* intractable when considered classically—the focus is on the question: *What makes the problem computationally difficult?*

In classical complexity, a decision problem is specified by two items of information:

- (1) The input to the problem.
- (2) The question to be answered.

In parameterized complexity there are three parts of a problem specification:

- (1) The input to the problem.
- (2) The aspects of the input that constitute the parameter.
- (3) The question.

The following are some examples of ways in which parameters arise in practice. It is important to note that a parameter can be an aggregate of information. From a practitioner's point of view, parameterization fundamentally enriches the "language" by which practice engages theory in the specification of computational problems. The parameter provides a systematic way of specifying distributional restrictions, in the hope of a successful deal with the devil.

Example 1.2 (Natural Parameters—Graph Width Metrics) Many important computational problems *naturally* consist of more than one piece of information. In the case of graph problems (such as the **VERTEX COVER** problem of interest to Dr. $\mathcal{O}$), it is frequently the case that the input consists of a graph G and a positive integer k . There are a number of "width metrics" of graphs that lead to problems of this form. Let $w(G)$ denote one of these measures of graph *width*, such as the *cutwidth*, *bandwidth*, or *pathwidth* of G . (The definitions of these metrics can be found in the Appendix and in Chapter 6.) The corresponding naturally parameterized problems have the form:

Instance: A graph G and a positive integer k .

Parameter: k .

Question: Is $w(G) \leq k$?

In this form, we have the parameterized problems **CUTWIDTH**, **BANDWIDTH**, and **PATHWIDTH**. All three are *NP*-complete when considered classically. All three have interesting applications covered by small ranges of the parameter k . For example, it has been proposed that the syntactic structure of sentences of natural languages can be modeled by dependency graphs of pathwidth less than 10 (Kornai and Tuza [304]). **CUTWIDTH** and **PATHWIDTH** have applications for small parameter ranges in Very Large Scale Integrated Circuit (VLSI) design [221]. **BANDWIDTH** has important applications in numerical analysis.

We now know that **CUTWIDTH** and **PATHWIDTH** are linear fixed-parameter tractable; that is, solvable in time $f(k)n$, where n is the size of the graph. **BANDWIDTH**, in contrast, appears *not* to be fixed-parameter tractable. The best known algorithm for k -**BANDWIDTH** requires time $\Omega(n^k)$. What this means in the practical sense is that we must consider giving more to the devil for this problem, in order to strike a successful bargain. The following parameterized problem is one such possibility.

BANDWIDTH II

Instance: A graph G and a positive integer k .

Parameter: The pair $(k, \Delta(G))$, where $\Delta(G)$ is the maximum degree of G .

Question: Is the bandwidth of G at most k ?

An important point illustrated by the above discussion is that a given classical problem (such as **BANDWIDTH**) can be examined under *many different* parameterizations; some of these will lead to fixed-parameter tractability and many probably will not.

Example 1.3 (Implicit Parameters in Logic Programming, Compiler Design, and Database Applications) One of the complaints that has been raised frequently by practical computer scientists against the classical complexity framework is that problems often turn out to be much easier in practice than one would expect from theoretical analyses. For example, the problem of type inference arises in implementations of programming languages such as ML that are based on the polymorphic-type λ -calculus. In Henglein and Mairson [265], it is shown that the problem is complete for deterministic exponential time, yet it has been widely noted that in practice, the problem is efficiently solved by an algorithm that is known to be exponential in the worst case. An explanation for this discrepancy comes from noting that the logic formulas that occur in natural programs tend to have small bounded depth of *let*'s. For a parameter k bounding this depth, the problem is linear fixed-parameter tractable (Abrahamson [2]).

Another example concerns the register allocation problem for imperative programming languages. This is a problem that is *NP*-complete in general. It has been shown that for structured programs (with short-circuit evaluation), the control-flow graphs have treewidth less than 10, which allows linear *FPT* algorithms based on this hidden structural parameter to be developed in, for instance, Alstrup, Lauridsen and Thorup [18] and Thorup [447].

In relational database applications, there are many problems that take a database d and a formula ϕ as input. Normally, the database is *much* larger than the formula. Furthermore, specific formulas may be of interest for many different databases, so it is reasonable to consider parameterized problems such as the following.

QUERY NON-EMPTYNESS

Instance: A database d and a query ϕ .

Parameter: ϕ .

Question: Is the query relation defined by ϕ nonempty?

If n denotes the size of the database d , then this (and a variety of related problems) can be trivially solved in time $O(n^{|\phi|})$. Yannakakis first raised the question of whether these problems are fixed-parameter tractable in [467]. For many computational problems in diverse areas of applied logic, formula size may be an appropriate parameter based on similar considerations.

Example 1.4 (Parameters Introduced by Engineering Practices) Some proposals for implementations of public key cryptosystems have considered limiting the size or Hamming weight of keys in order to obtain faster processing times. A cautionary note is sounded by the result in Fellows and Koblitz [212] that for every fixed k , with high probability it can be determined in time $f(k)n^3$ whether an n -bit positive integer has a prime divisor less than n^k . If a similar result holds for the Discrete Logarithm problem for exponents of bounded Hamming weight, then the security of cryptographic implementations such as proposed in Agnew, Mullin, Onyszcuk and Vanstone [14] will certainly be compromised. Both problems are trivially solvable in time $O(n^{k+c})$, where c is a small constant.

For another example arising from hardware engineering, the number of processors in a physical parallel processing system is often in the range $k \leq 50$, and thus is a relevant parameter (Bodlaender and Fellows [78]).

Example 1.5 (Approximation Parameters in Computational Biology) A parameter can be composed of any relevant aspects of a problem. In Jiang, Lawler, and Wang [280], it is shown that the problem of computing a tree alignment for DNA sequences of length n for k species of cost within a factor of $1 + \epsilon$ of the cost of an optimal alignment can be accomplished in time $O((kn)^{c/\epsilon})$. It is natural to consider this problem with the parameter consisting of the pair $(k, 1/\epsilon)$ and to ask whether it is fixed-parameter tractable. We remark that computational biology is an area where there is an abundance of natural parameters and where also the size n of the problem input is frequently quite large. (We refer the reader also to, for instance, Leizhen Cai [103], Kaplan, Shamir and Tarjan [289] or Steel [433].)

The above examples are intended to illustrate some of the possibilities for problem parameterization. Computer scientists with a practical orientation have frequently complained that the classical complexity framework is not sufficiently realistic about natural input distributions (see e.g., Dean, Kirman, and Lin. [152]). As shown by several of the above examples, parameterization provides a systematic means of including distributional information in complexity analysis.

1.7 A Distinctive Positive Toolkit

The toolkit for designing *FPT* algorithms includes a variety of distinctive techniques. Some of these methods are quite intricate and technical, and not all of them are practical.

- **Well-Quasi-Ordering.** One of the first and most striking of the general methods for demonstrating fixed-parameter tractability has been supplied by the very deep results of Robertson and Seymour on graph minors and well-quasi-ordering. The proofs of the main theorems that yield these general methods are daunting, running to over 1500 densely argued journal pages. In their general form, these methods yield polynomial-time and fixed-parameter tractability results that are supremely impractical, primarily because they involve astronomical hidden constants. The parameter functions involved have steadily improved, but for now, well-quasi-ordering is primarily a complexity classification tool—a way of showing, often quite easily, that fixed-parameter tractability is possible.

For a concrete example, when the k -**VERTEX COVER** problem was first observed to be fixed-parameter tractable in time $O(f(k)n^3)$ by means of the Robertson-Seymour theorems, the initial $f(k)$ was a tower of 2's of height described by a tower of 2's of height described by a

tower of 2's of height described by ... (and so on) ... a function that is utterly impractical even for $k = 1$!

Johnson's observation that it was sufficient to work with a bounded-width tree decomposition improved the implicit $f(k)$ to something like $2^{2^{500k}}$, still completely impractical for $k = 1$. There have obviously been drastic improvements in the tractability parameter function for **DATA CLEANING** since then.

- Bounded Treewidth and Pathwidth.** Techniques for designing algorithms based on bound-width tree and path decompositions of graphs and hypergraphs have been developing for over a decade and now constitute a very extensive collection of fundamental results and perspectives that apply to an almost unlimited range of problems. One of the surprises of this area of algorithmics is the widespread nature of bounded treewidth and pathwidth combinatorics. For example, it turns out to be a structure of fundamental importance in the analysis of control-flow graphs (Thorup [447]). After several rounds of improvement, the best known general algorithm for computing a tree decomposition of width k runs in time $f(k)n$ for graphs on n vertices, where $f(k) = 2^{ck^2}$, a result due to Bodlaender [71]. This parameter function is still too impractical, and the algorithm has not been implemented. These techniques are described in Chapter 6.
- Color Coding (Hashing).** A general method of devising *FPT* algorithms using k -perfect families of hash functions derived from sophisticated derandomization techniques has been developed by Alon, Yuster, and Zwick in [17] after earlier work of Dennenberg, Gurevich, and Shelah on hashing in Dennenberg, Gurevich and Shelah [154]. A description of this approach and a number of applications are described in Chapter 8. The method works quite well for certain kinds of problems, such as finding small subgraphs in a graph (such as paths, cycles, and matchings). The parameter functions implicated by the method are typically $f(k) = 2^k$ and the resulting algorithms offer significant practical parameter ranges and no sources of large hidden constants.
- Elementary Methods.** There are a number of elementary methods of great importance, such as *bounded search trees* and *reductions to a problem kernel*. An interesting aspect of these approaches is that although they are simple algorithmic strategies, they are in some sense "new ideas" that might not immediately come to mind, *because they involve costs that are exponential in the parameter*. They are not part of the usual toolkit of polynomial-time algorithm design and, hence, are worth articulating as standard techniques in devising algorithms for problems that are fixed-parameter tractable. The current best results on **VERTEX COVER** and a number of other problems depend only on these simple ideas. Several of these basic methods are described in Chapters 3 and 8.

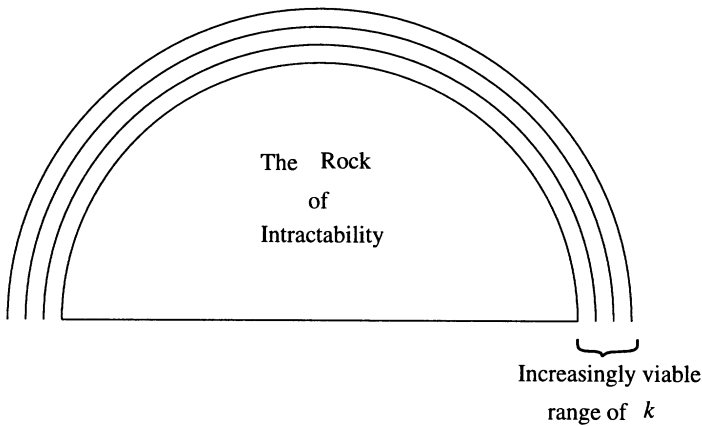


FIGURE 1.4. Lichenism

1.8 $\mathcal{O}$ No?

The perspective of parameterized complexity invites us to take a new point of view on the evaluation of algorithms by considering the performance value of an *FPT* algorithm to be the range of the parameter that it allows us to deal with successfully.

There are currently no theorems that indicate limits on what we might reasonably expect to achieve in terms of practical parameter ranges for problems that are fixed-parameter tractable. For example, **VERTEX COVER** is now well-solved for input of any size, so long as the parameter value is $k \leq 60$. Can the practical range be extended to $k \leq 1000$? Will finding tree decompositions of width up to 10 be practical one day? The algorithm design philosophy appropriate to this point of view is indicated in Figure 1.4.

There is a certain way in which the performance measure of useful parameter ranges can be formalized as an alternative to “big $\mathcal{O}$ ” analysis. One of the results described in Chapter 2 is that any parameterized problem that is fixed-parameter tractable is solvable in time $f(k) + n^c$, where c is a constant, n is the total size of the input, and k is the parameter. There are no hidden asymptotic constants involved in this expression—everything is completely explicit. The goodness of an *FPT* algorithm can thus be measured as the largest value of k for which $f(k)$ is less than some universal speed limit U , such as $U = 10^{20}$. We might call this the number of *klams* that the algorithm is worth.³

For the following well-known parameterized problems, the indicated klam values from Table 1.2 can be achieved for $c = 3$ and $U = 10^{20}$. It must be admitted that the values for some problems are currently somewhat disappointing. However, in view of the **VERTEX COVER** story, we believe that there is good hope for much larger klam values as the *FPT* toolkit continues to develop.

³To prevent cheating on this measure, we have to fix c . Since c is the exponent of the polynomial, we lose almost nothing by fixing $c = 3$.

Problem	Klam value	Reference
VERTEX COVER	60	46
MAX LEAF SPANNING TREE	4	169
PLANAR DOMINATING SET	5	169
FEEDBACK VERTEX SET	4	169

TABLE 1.2. How Many Klam's?

1.9 The Barometer of Parametric Intractability

Although many natural parameterized problems that are *NP*-complete (or worse) turn out to be fixed-parameter tractable by methods such as described in Part I of this book, it is also the case that many interesting parameterized problems appear to be parametrically *intractable*. Part II of this book explores a framework in which likely parametric intractability can be demonstrated.

Much of this framework can be illustrated as an effort to understand the complexity behavior of three very specific graph problems: **VERTEX COVER**, **INDEPENDENT SET**, and **DOMINATING SET**. All three of these problems take as input a graph $G = (V, E)$ and a positive integer k that we consider to be the parameter. The question is whether there exists a set $V' \subseteq V$ of k vertices that is, respectively, a *vertex cover*, an *independent set*, or a *dominating set*. A *vertex cover* (as we have seen in discussing the **DATA CLEANING** problem) is a set V' of vertices such that every edge in G is adjacent to at least one vertex of V' . An *independent set* is a set V' of vertices such that no two vertices in V' are adjacent in G (equivalently, the vertices of V' form a clique in the complement of G). A *dominating set* is a set V' of vertices such that every vertex in G has at least one neighbor in V' . These three properties are illustrated in Figure 1.5.

All three problems are obviously solvable by the brute-force “try all k subsets” algorithm that requires time $O(n^k)$. All three problems are *NP*-complete, and thus,

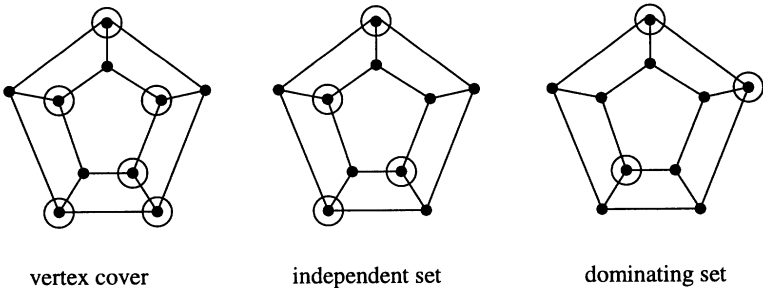


FIGURE 1.5. Three Vertex Set Problems

	$n = 50$	$n = 100$	$n = 150$
$k = 2$	625	2,500	5,625
$k = 3$	15,625	125,000	421,875
$k = 5$	390,625	6,250,000	31,640,625
$k = 10$	1.9×10^{12}	9.8×10^{14}	3.7×10^{16}
$k = 20$	1.8×10^{26}	9.5×10^{31}	2.1×10^{35}

TABLE 1.3. The Ratio $\frac{n^{k+1}}{2^k n}$ for Various Values of n and k

from the classical point of view, they are essentially indistinguishable. At the time of the origins of parameterized complexity, it had been shown that for every fixed k , the k -**VERTEX COVER** problem could be solved in time $f(k)n^3$, and it was obvious to ask whether something similar could be done for the k -**INDEPENDENT SET** and the k -**DOMINATING SET**. Given that the k -**VERTEX COVER** is now completely well-solved for $k \leq 70$, this is still a very compelling question. Table 1.3 illustrates the difference between the current best algorithm for k -**VERTEX COVER** and the current best algorithm for k -**DOMINATING SET** for various values of n and k . The best algorithm currently known for k -**DOMINATING SET** is *still* simply to try all k subsets, which requires time $O(n^{k+1})$.

The means by which **VERTEX COVER** was first shown to be fixed-parameter tractable was suggestive. Essentially, a proof of fixed-parameter tractability that is based on well-quasi-ordering methods shows that for each fixed k , there is a finite number of “forbidden substructures.” For each of these, there is an $O(n^3)$ algorithm to determine whether the substructure is present in a graph. If none of the substructures is present, then the answer is “yes.” Otherwise, the answer is “no.”

The natural question to ask is whether such methods might also apply to k -**INDEPENDENT SET** and k -**DOMINATING SET**, perhaps by means of some completely different well-quasi-order than graph minors. There is no immediately available reason for considering this to be unlikely.

It seems, however, that k -**DOMINATING SET** represents some fundamental “wall of intractability” where there is no significant alternative to trying all k subsets. The situation is highly reminiscent of the apparent wall of intractability that is located by NP -completeness. For example, in the case of 3SAT expressions on n variables, we presently know essentially no better algorithm than the one that exhaustively tries all 2^n possible truth assignments.

The goal of fixed-parameter tractability leads in an inevitable way to several flavors of parametric problem reducibility (e.g., Turing and many:1 reducibilities, and others). The essential point in the definition of reducibility is that if problem A reduces to problem B, and problem B is fixed-parameter tractable, then so too

is A. As in the theory of *NP*-completeness, this gives us a means of gathering evidence that a problem such as **DOMINATING SET** is likely *not* to be fixed-parameter tractable.

It would be preferable, given our intuitions, to prove directly that **DOMINATING SET** is *not* fixed-parameter tractable. But since this would imply $P \neq NP$, we must try to be content for now with the study of the inter-reducibility of problems.

As a concrete example of a parameterized problem reduction, we know how to take a pair (G, k) consisting of a graph G and a positive integer k , and produce a pair (G', k') such that G has an independent set of size k if and only if G' has a dominating set of size $k' = 2k$. Note that k' depends only on k and not at all on G —this is the key property of a transformation between parameterized problems (the detailed definitions can be found in Chapter 9). This transformation reduces **INDEPENDENT SET** to **DOMINATING SET**. *At present, we know no parametric transformation in the other direction, and we suspect that no such transformation exists.*

Now is the time to confess that our entire structural theory of parameterized complexity was originally motivated by these three concrete, irritating problems. The definition of the $W[t]$ classes was guided by the desire to find those classes of parameterized problems for which the recalcitrant **INDEPENDENT SET** and **DOMINATING SET** problems would be complete. These would apparently have to be different classes, very much unlike the theory of *NP*-completeness.

There are two kinds of evidence for intractability that can be gathered by studying the interreducibility of concrete problems. The first kind of evidence is, roughly speaking, sociological. This is the kind of evidence illustrated by the humorous drawings in the first few pages of the classic text by Garey and Johnson—so many problems stand or fall together that the combination of efforts expended unsuccessfully from the various vantages compels a belief in their inherent intractability. The second kind of evidence is some form of direct intuition concerning the nature of the computations that define the issue (e.g., nondeterministic as opposed to deterministic polynomial time). For parameterized complexity, both kinds of arguments can now be made.

The first step in the program to understand the apparently non-*FPT* nature of the **INDEPENDENT SET** and **DOMINATING SET** problems was to define a hierarchy of parameterized complexity classes

$$FPT \subseteq W[1] \subseteq W[2] \cdots \subseteq W[P]$$

based on the natural logical depth of the problems. The first significant results in the program of studying parameterized complexity showed that **INDEPENDENT SET** is complete for $W[1]$ and that **DOMINATING SET** is complete for $W[2]$. (The details can be found in Chapters 10 and 12.)

What we now know is that $W[1]$ and $W[2]$ are, empirically, two of the most natural and significant degrees of parameterized complexity, thus giving the first kind of evidence that **INDEPENDENT SET** and **DOMINATING SET** are unlikely to be fixed-parameter tractable. There is now a substantial list of very natural and important problems that are *precisely* $W[1]$ -complete, including **INDEPENDENT SET**, **CLIQUE**, **VAPNIK-CHERVONENKIS DIMENSION**, **MONOTONE**

QUERY NON-EMPTYNESS (i.e. **MONOTONE DATA COMPLEXITY FOR RELATIONAL DATABASES**), **k -STEP DERIVATION FOR CONTEXT SENSITIVE LANGUAGES**, **k -STEP SQUARE TILING**, **SHORT POST CORRESPONDENCE**, and **m -LENGTH COMMON SUBSEQUENCE FOR k -SEQUENCES (LCS II)**. There is a similar list of problems that are complete for $W[2]$.

The second kind of evidence that **INDEPENDENT SET** and **DOMINATING SET** are likely not to be fixed-parameter tractable is provided by the following consideration. The most fundamental of the $W[1]$ -complete problems and the parameterized analog of NP is the problem:

SHORT NONDETERMINISTIC TURING MACHINE ACCEPTANCE

Instance: A nondeterministic Turing machine M and a positive integer k .
Parameter: k .
Question: Does M have a computation path accepting the empty string in at most k steps?

If we fix k to be an arbitrary positive integer constant (e.g., $k = 1$) and consider the above problem classically, where the question is whether M halts in at most $|M|^k$ steps, then we have the defining problem for NP . A nondeterministic Turing machine is such an opaque and generic object that it simply does not seem reasonable that we should be able to decide if a Turing machine M halts in at most $|M|$ steps in time bounded by a fixed polynomial function $O(|M|^c)$, for some constant c .

We may view the importance of Cook's Theorem (Cook [131]) to be in connecting (ultimately) the thousands of natural problems that are NP -complete with this central generic problem concerning nondeterministic Turing machines, thereby providing powerful intuitive evidence that these problems are not solvable in deterministic polynomial time. It seems to us that the same intuition suggests that **SHORT NONDETERMINISTIC TURING MACHINE ACCEPTANCE** is fixed-parameter intractable.⁴ The theorem that **SHORT NONDETERMINISTIC TURING MACHINE ACCEPTANCE** is complete for $W[1]$ essentially constitutes a miniaturization of Cook's Theorem. For this reason, it is natural to regard $W[1]$ as the parameterized analog of NP , and hardness for $W[1]$ as the basic measure for likely parametric intractability.

The reader who is inclined to consider the possibility of $P = NP$ (and who is apparently not alone, Dean et. al. [152]) might wish to ponder also the $FPT = W[1]$ question. We would argue that the miniaturization presents the essential issue in a more concrete and compelling form.

The theory of parametric intractability has developed in three ways that could not have been foreseen in advance, and that could only emerge after a substantial amount of empirical investigation:

- It is remarkably applicable. It is now clear that the theory applies naturally to a very wide range of problems in diverse areas of computing.

⁴It is surely one of the gems of complexity theory that this problem is fixed-parameter tractable if and only if **CLIQUE** is fixed-parameter tractable.

It is surprisingly orthogonal to classical complexity. There are no systematic or trivial correlations between the classical complexity of a problem and the parametric complexity of its naturally parameterized forms. Many games, for example, are precisely *PSPACE*-complete classically, whereas their k -move parameterizations have widely varying parameterized complexity—some are *FPT* and others turn out to hard or complete for many different levels of the hierarchy of parameterized problem classes.

- There are surprising elements of natural order. With so many different levels of the W hierarchy and so many different intractable classes, one might have expected a certain amount of chaos. Yet it seems that the overwhelming majority of natural parameterized problems belong to no more than five or six different degrees (equivalence classes defined by parametric interreducibility). This is unexpected and unexplained, and very much akin to the remarkable universality of *NP*-completeness in the classical framework.

One useful aspect to having many different natural levels of parametric intractability is that these can be used as a kind of barometer of how close one is getting to a successful deal with the devil. For example, when the *NP*-complete **LONGEST COMMON SUBSEQUENCE** problem is parameterized by the number of strings, it is hard for $W[t]$ for all t . When it is parameterized by the number of strings and the length of the common subsequence, it is $W[1]$ -complete. When it is parameterized by the number of strings, the length of the common subsequence, and the size of the alphabet, it is *FPT* (Bodlaender, Downey, Fellows, Hallett, and Wareham [73, 75, 74]).

1.10 Structural Aspects of Parameterized Complexity

Part III of this book is concerned with various structural issues concerning the parameterized complexity framework. In general, these are much harder to resolve than the analogous questions concerning classical complexity classes and the degree structure of polynomial-time reducibility. One of the results presented in Part III provides an explanation of the difficulty of the subject. Whereas ordinary polynomial-time reducibilities are Σ_2^0 in arithmetical complexity, parameterized reducibilities yield index sets that are Σ_4^0 complete.

There are several kinds of structural questions that are explored in Part III and elsewhere:

- What is the evidence that the $W[t]$ hierarchy is proper?
- What are the relationships between classical and parameterized complexity issues?
- What are the connections between parameterized complexity and approximation complexity for optimization problems?

To what extent do analogs of classical theorems hold for parameterized complexity?

For example, could it be that $W[1] \neq FPT$, whereas for all $t \geq 1$, $W[t] = W[1]$? This is entirely compatible with $P \neq NP$. One of the results presented in Part III gives a computational basis for the $W[t]$ classes based on uniform parameterized circuit families of depth t and establishes that these circuit families underlying the $W[t]$ hierarchy form a *proper* hierarchy of nondeterministic computational power.

Another result shows that the implication $FPT = W[1] \Rightarrow P = NP$ cannot be proved by any argument that relativizes. Despite this evidence that the conjecture that the W hierarchy is proper is strictly stronger than the conjecture that $P \neq NP$, there is still some connection between the two frameworks. If $FPT = W[1]$, then it is possible to solve classical 3SAT in time $2^{o(n)}$ for expressions having n variables.

The complexity of the reductions mean that simple delayed diagonalization arguments from classical complexity theory need to be replaced, if they hold at all, by complex priority arguments. For instance, a relatively deep result presented in Part III is a partial analog of Ladner's density theorem. Many possible analogs of classical structural theorems remain open; it is unclear to what extent structural complexity theory as we know it classically can follow Cook's Theorem into parametric miniaturization.

1.11 An Overview of Current Research Horizons

As large as this book has turned out to be, the study of parameterized complexity is still in its infancy. We firmly believe that, in the long term, making systematic distinctions about the relative importance of different aspects of problem input is an essential and inevitable response to the ubiquitous nature of computational intractability. The parameterized perspective begins to give us a systematic inroad into the distribution-sensitive analysis of computational problems in a way that can actually be carried out for a wide range of concrete problems. It gives us a way to begin to come to grips with the basic engineering fact that not all kinds of numbers are created equal (i.e., $k \leq 20$ is qualitatively different from $n = 10^5$) or have the same role, as shown in Figure 1.6.

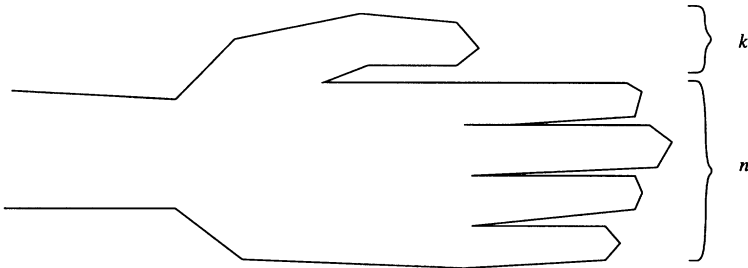


FIGURE 1.6. Getting a Grip on Intractability

The landscape of complexity revealed by the fundamental shift to “two-part input” and the “deal with the devil” that defines fixed-parameter tractability (and thereby the relevant notion of problem reducibility) is unfamiliar and orthogonal to many of our classical algorithm design and complexity analysis habits.

We now know that the program is widely applicable and can be carried out—but it is often more difficult than the P versus NP program. Analogs of classical structural theorems seem typically to have a much higher technical price. Even ordinary hardness results are generally much more intricate and demanding. The positive toolkit for designing FPT algorithms contains several key methods that are very deep and general—but for which practicality is still not yet clearly established.

Much remains to be explored.

Part I

Parameterized Tractability

2

The Basic Definitions

2.1 Fixed-Parameter Tractability

Our concerns are (decision) problems with two or more inputs. Thus, we will be considering languages $L \subseteq \Sigma^* \times \Sigma^*$. We refer to such languages as *parameterized languages*. If $\langle x, k \rangle$ is in a parameterized language L , we call k the parameter. Usually the parameter will be a positive integer, but it might be a graph or algebraic structure. However, in the interest of readability and with no loss of generality, we will usually identify the domain of the parameter as the natural numbers $\mathbb{N}$ and hence consider languages $L \subseteq \Sigma^* \times \mathbb{N}$. For a fixed k , we call $L_k = \{\langle x, k \rangle : \langle x, k \rangle \in L\}$ the *kth slice* of L .

As we have seen in the introduction, our main idea is to study languages that are tractable “by the slice.” As the reader will recall, being tractable by the slice meant that there is a constant c , independent of k , such that for all k , membership of L_k can be determined in time $O(|x|^c)$. However, such parameterized tractability can occur in many different flavors of uniformity and for the sharpest results it seems important to delineate these flavors. Before we give some precise definitions, we invite the reader to consider the following examples.

Example 2.1 VERTEX COVER

Instance: A graph $G = (V, E)$.

Parameter: A positive integer k .

Question: Does G have a vertex cover of size $\leq k$? (A vertex cover of a graph G is a collection of vertices V' of G such that for all edges $v_1 v_2$ of G either $v_1 \in V'$ or $v_2 \in V'$.)

Example 2.2 GRAPH GENUS

- Instance:* A graph $G = (V, E)$.
Parameter: A positive integer k .
Question: Does G have genus k ? (That is, can G be embedded with no edges crossing on a surface with k handles?)

Example 2.3 GRAPH LINKING NUMBER

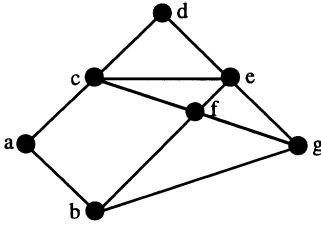
- Instance:* A graph $G = (V, E)$.
Parameter: A positive integer k .
Question: Can G be embedded into 3-space such that the maximum size of a collection of topologically linked disjoint cycles is bounded by k ?

In Figure 2.1 we give some examples to illustrate the problems above. The fact that K_6 has linking number 1 is due to Sachs [401] and Conway and Gordon [130]. If we consider the classical versions of the problems above where k is not fixed, then they are all *NP-hard*. Each of the above problems exhibits some form of parameterized tractability. In the next section, we will look at a simple technique called the method of bounded search trees that can solve **VERTEX COVER** using a single algorithm Γ running in time $2^k|G|$ for each k (Downey and Fellows [165, 169]). Fellows and Langston [216] introduced a method which allowed them to use the deep results of Robertson and Seymour [396, 397] to construct a single algorithm Φ which accepts $\langle G, k \rangle$ as input instances of **GRAPH GENUS** such that Φ determines if G has genus k in time $O(|G|^3)$. This has recently been improved by Mohar [354] to $O(|G|)$. Finally, Fellows and Langston [216] used the Robertson-Seymour methods to prove that for each k there is an algorithm Ψ_k which runs in time $O(|G|^3)$ and which determines if the graph G has linking number k . (We will look at the proofs of these results and the techniques used in Chapter 7.)

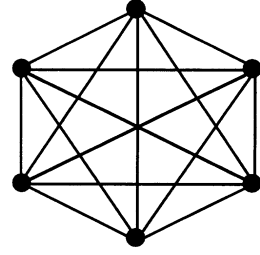
Notice the differences between the three varieties of fixed-parameter tractability. They are all $O(|G|^c)$ for some c slice-wise but:

- In Example 2.1, we have a single *known* algorithm which works for all k , and, moreover, we can compute the constant and hence the exact running time for each k . In the definitions below we refer to this behavior as *strongly uniform fixed-parameter tractability*.
- In Example 2.2, we still have a single algorithm Φ for all k , but this time we have *no* way of computing the constant in the running time. We merely know that for each k , the running time of Φ on input $\langle G, k \rangle$ is $O(|G|^3)$. This behavior is called *uniform fixed-parameter tractability*.
- In Example 2.3, all we know is the exponent of the running time for the algorithms. For each k , we have a different (and unknown) algorithm running in $O(|G|^3)$ with unknown constants. This behavior is called *nonuniform fixed-parameter tractability*.

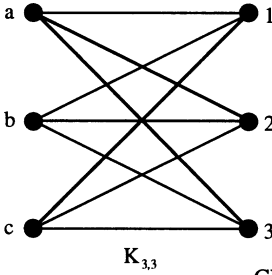
Considerations such as the examples above naturally lead us to the definitions below.



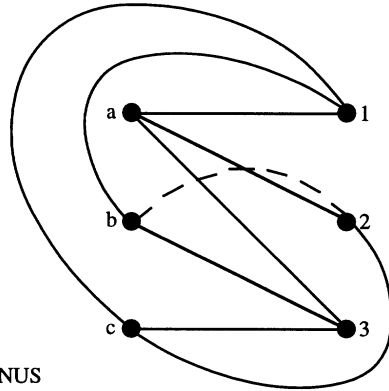
VERTEX COVER : Vertices cover edges.
Example: {c, f, b, e, h}.



GRAPH LINKING NUMBER :
 K_6 has linking number 1.



GRAPH GENUS



$K_{3,3}$ has genus 1 by
putting all lines except
< b, 2 > on a sphere
and < b, 2 > on a handle.

FIGURE 2.1. Examples of FPT Problems

Definition 2.4 (The Fundamental Definitions) Let A be a parameterized problem.

- (i) We say that A is *uniformly fixed-parameter tractable* if there is an algorithm Φ , a constant c , and an arbitrary function $f : \mathbb{N} \mapsto \mathbb{N}$ such that:
 - (a) the running time of $\Phi(\langle x, k \rangle)$ is at most $f(k)|x|^c$,
 - (b) $\langle x, k \rangle \in A$ iff $\Phi(\langle x, k \rangle) = 1$.
- (ii) We say that A is *strongly uniformly fixed-parameter tractable* if A is uniformly fixed-parameter tractable via some Φ and f such that f is recursive.
- (iii) We say that A is *nonuniformly fixed-parameter tractable* if there is a constant c , a function $f : \mathbb{N} \mapsto \mathbb{N}$, and a collection of procedures $\{\Phi_k : k \in \mathbb{N}\}$ such that for each $k \in \mathbb{N}$, and the running time of $\Phi_k(\langle x, k \rangle)$ is $f(k)|x|^c$ and $\langle x, k \rangle \in A$ iff $\Phi_k(\langle x, k \rangle) = 1$.

In Chapter 19, we will prove that even on recursive languages, the above definitions generate distinct classes of parameterized languages. The diagonalization arguments used in Chapter 19 construct *artificial* languages to provably separate the classes. At present, we are not aware of any *natural* problem that is *provably* (say) in the class of problems that are uniformly but not strongly uniformly fixed-parameter tractable. We remark, however, that there are examples of *natural* problems such as **GRAPH LINKING NUMBER** which can *at present* only be classified as nonuniformly tractable. Nevertheless, in practice, it seems that the most important types of parameterized tractability are the two varieties of uniform tractability.

Until Chapter 7, the reader can assume that unless otherwise specified, fixed-parameter tractability will mean uniform fixed-parameter tractability and most probably strongly uniform fixed-parameter tractability.

We remark that most natural *FPT* problems which have been studied so far seem to exhibit a general migration toward strongly uniform tractability; that is, once a problem has been identified as *FPT* with more precise combinatorics, we eventually show the problem to be strongly uniformly *FPT*.

One reason for this general process of improvement is that often the initial classification of the problem as *FPT* came from the application of some *general result* pertaining to a *class* of problems containing the one at hand. Later, more uniformity, together with better constants and algorithms, can be obtained after studying the particular combinatorics of the problem considered alone. It is a very interesting open question to find some natural examples of languages which are *FPT* but provably not strongly uniformly *FPT*.

It is very important that the reader realize that parameterized complexity is orthogonal to classical complexity; that is, the parameterized complexity of a problem can bear no relationship to the classical complexity of the problem. For instance, let L be *any* recursive language. Perhaps L is not even elementary recursive and the time complexity for L might dwarf Ackermann's function. But consider the parameterized language $L' = \{(x, x) : x \in L\}$. Classically, L' has the same complexity as L , but as a *parameterized problem*, L' is in *FPT*, with a *constant* time algorithm. Later, we will see examples where the parameterized versions are hard but classical versions are easy.

2.2 The Advice View

We close this chapter with some brief remarks about an alternative view of being *FPT* provided by an analog of the classical notion of *advice* introduced by Karp and Lipton [292]. Aside from the easy result below, in this section we do not pursue the advice view further. Rather, we offer it to the reader as an alternative view that might shed some light upon the basic definitions. We will look a little more closely at the advice view in Chapter 5. The reader concerned with concrete problems may well wish to proceed directly to the next chapter.

The classical advice class $P/poly$ is concerned with languages that can be recognized in polynomial time when we are provided, for input of size $|x| = n$, a word of advice w_n , where the length of w_n is bounded by some polynomial $q(n)$. Notice that the word of advice depends *only* upon the *size* $|x| = n$ of the input, not what x actually is, and hence a single w_n works for all the Σ^n strings of length n . In Cai, Chen, Downey, and Fellows [110], it is proven that nonuniform FPT is exactly the set of problems solvable in polynomial time using a finite piece of advice $f(k)$ for each slice k . (So the whole slice L_k uses the same piece of advice.)

Theorem 2.5 (Cai, Chen, Downey, and Fellows [110])

- (i) $L \in FPT$ (nonuniform) iff there is a polynomial-time oracle Turing machine Φ and a function $f : \mathbb{N} \rightarrow \Sigma^*$ such that for all x and k

$$\langle x, k \rangle \in L \quad \text{iff} \quad \Phi^{f(k)}(\langle x, k \rangle) = 1.$$

- (ii) Furthermore, $L \in FPT$ (strongly uniform) iff there is a polynomial-time oracle Turing machine Φ and a recursive function $f : \mathbb{N} \rightarrow \Sigma^*$ such that for all x and k

$$\langle x, k \rangle \in L \quad \text{iff} \quad \Phi^{f(k)}(\langle x, k \rangle) = 1.$$

Proof. (i) The proof is easy. Let $L \in$ nonuniform FPT . Then there is a sequence of procedures M_k , a constant c , and a function g such that $\langle x, k \rangle \in L$ iff $M_k(\langle x, k \rangle)$ accepts, and M runs in time $g(k)|x|^c$. Consider the following new machine Φ accepting L . This machine runs in time $|x|^{c+1}$. Now, we know that $f(k)|x|^c$ is dominated by $|x|^{c+1}$ and so there is a string $x = 0^n$ for some n by which $|x|^{c+1} > 2g(k)|x|^c$. Once this n and an index $e(k)$ of the procedure M_k are known, we can write all the values for $L(\langle x, k \rangle)$ for $|x| \leq n$ in a table T . (This is the advice.) We define $f(k) = \langle n, e(k), T \rangle$ under suitable encoding. On input $\langle x, k \rangle$, Φ first sees if the length n of x exceeds the length of those on the lookup table $f(k) = T$. If the length exceeds n , then the machine emulates $M_k(\langle x, k \rangle)$. Otherwise, the machine Φ uses table lookup to give the desired answer.

Conversely, if Φ and f exist for L , then we can reverse the above process to get a nonuniform FPT algorithm running in time $p(\langle x, k \rangle) + h(k)$ for some function h , where p is the running time for Φ .

The proof of (ii) is similar except that now we can recursively compute everything.

We do not pursue the advice view further here but offer it to the reader as an alternative view that might give some further insight into the basic definitions. We will look a little more closely at this in Chapter 5.

Historical Notes

Ever since the discovery of the fact that many natural problems are seemingly intractable, authors have looked for feasible partial solutions. We refer the reader

to Garey and Johnson [239], particularly Chapter 4. In that chapter, Garey and Johnson look at what they call “Analysing Subproblems.” The reason for Garey and Johnson’s interest in subproblems is encapsulated in the following quote.

“If a general problem is *NP*-complete, we know that an exponential time algorithm will be needed (unless $P = NP$), but there are a variety of ways which the time complexity of an algorithm can be “exponential,” some of which might be preferable to others.” (Garey and Johnson [239, p. 106].)

In retrospect, it is obvious that many authors have devised algorithms which demonstrate the fact that the parameterized problem is *FPT*. For instance, as we will see in Chapter 4, all pseudo-polynomial-time algorithms actually demonstrate the fact that the relevant problem is *FPT*. To our knowledge, the first author to explicitly note the fact that as k varied, problems such as **DOMINATING SET** seemed to take time $\Omega(n^{f(k)})$ with $f(k) \rightarrow \infty$ was Ken Regan in some comments in [385]. The first paper to address the “asymptotic” fixed-parameter complexity of parameterized problems (i.e. the behavior as $k \rightarrow \infty$), was Abrahamson, Ellis, Fellows, and Mata [5]. Roughly speaking, those authors looked at the comparison between languages that were nonuniformly *FPT* and those that were “*P*-complete” or “dual *P*-complete” by the slice. There are a number of severe limitations and other more technical problems with the Abrahamson et al. approach. Those authors certainly could not address comparisons of the parameterized complexities of, say, **VERTEX COVER** and **DOMINATING SET**, nor parameterized versions of problems outside of *NP*. The basic definitions of this chapter were given in Downey and Fellows [165, 166, 167]. The advice view is taken from Cai, Chen, Downey, and Fellows [110].

We will give more detailed historical remarks concerning the issue of fixed-parameter *tractability* versus *intractability* in Chapter 9 of Part III, where they will be more in context. We will also give historical comments concerning the various techniques examined in this book in the relevant chapters and sections.

3

Some Ad Hoc Methods: The Methods of Bounded Search Tree and Problem Kernel

3.1 The Method of Bounded Search Trees

3.1.1 *The Basic Method*

The method of bounded search trees is probably the easiest to apply and is based on the following facts. Many combinatorial problems can be solved by algorithms that can be decomposed into two distinct parts:

- First, within the algorithm we compute, perhaps inefficiently, some search space which is often an exponential-sized search tree.
- Thereafter, we run some relatively efficient algorithm on each branch of the tree often simply based upon, say, depth first search.

The exponential worst-case complexity of such algorithms comes from problem instances where we need complete tree traversal. (Many probabilistic analyses for such problems are based upon *how often* significant tree traversal is needed.) For our purposes, the critical observation is that for many parameterized problems, the size of the tree *depends only upon the parameter*. Then, for a fixed k , the search space becomes *constant* size and the algorithm is then efficient for each fixed k . A classic case of the application of this technique is the **VERTEX COVER** problem (Problem 2.1) of Chapter 2. The reader should recall that the **VERTEX COVER** problem takes a given graph G and asks if there is a size k set of vertices that covers all the edges.

Theorem 3.1 (Downey and Fellows [165]) **VERTEX COVER** is solvable in time $O(2^k |V(G)|)$, where the hidden constant is independent of k and $|V(G)|$.

Proof. We construct a binary tree of height k as follows. Label the root of the tree with the empty set and the graph G . Choose an edge $uv \in E$. In any vertex cover V' of G , we must have either $u \in V'$ or $v \in V'$, so we create children of the root node corresponding to these two possibilities. Thus, the first child is labeled with $\{u\}$ and $G - u$, and the second child is labeled with $\{v\}$ and $G - v$. The set of vertices labeling a node represents a “possible” vertex cover, and the graph labeling the node represents what remains to be covered in G . In general, for a node labeled with the set of vertices S and the subgraph H of G , we choose an edge $uv \in E(H)$ and create the two child nodes labeled, respectively, $S \cup \{u\}$ and $H - u$, and $S \cup \{v\}$ and $H - v$. If we create a node at height at most k in the tree that is labeled with a graph having no edges, then a vertex cover of cardinality at most k has been found. There is no need to explore the tree beyond height k .

Convention. Henceforth, we will omit the phrase “where the hidden constant is independent of k and G ,” since it will be obvious and implicit. (The hidden constants will always be small.)

In passing, we remark that the proof of Theorem 3.1 above also establishes the following result.

Corollary 3.2 *The problem LOG VERTEX COVER which asks if a given graph G has a vertex cover of size $\leq \log(|G|)$ is in P and is solvable in time $O(|G|^2)$.*

Corollary 3.2 improves the work of Papadimitriou and Yannakakis [368]. We will look at the relationship of our work with that of Papadimitriou and Yannakakis in Part III.

The VERTEX COVER problem asks us to consider the size of sets where “vertices cover edges.” A well-known related problem is that of DOMINATING SET. In DOMINATING SET, “vertices cover vertices.”

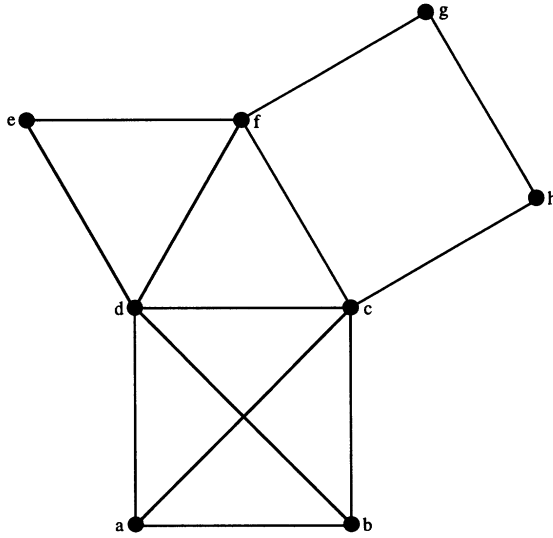
DOMINATING SET

Input: A graph G .
Parameter: A positive integer k .
Question: Does G have a dominating set of size k ? [A *dominating set* is a set $V' \subseteq V(G)$ where, for each $u \in V(G)$ there is a $v \in V'$ such that $uv \in E(G)$.]

In Part II, we will examine evidence that, despite its apparent similarity to VERTEX COVER, DOMINATING SET is *not* fixed-parameter tractable.

However, using the method of bounded search trees, it is possible to prove that several interesting variations of the DOMINATING SET problem are *FPT*. A nice example of an *FPT* variation of the basic DOMINATING SET problem is provided by PLANAR DOMINATING SET, which is the restriction of DOMINATING SET to planar graphs. We will need the following lemma.

Lemma 3.3 *If $G = (V, E)$ is a simple planar graph with a vertex partition into two sets $V = V_1 \cup V_2$ satisfying*



$\{f, c\}$ form a dominating set of size 2.

FIGURE 3.1. DOMINATING SET

- (1) *the minimum degree of vertices in V_1 is at least 3,*
- (2) *V_1 is an independent set in G , then there is a vertex $u \in V_2$ of degree at most 10 in G .*

Proof. Let G be a counterexample of minimum possible order having a maximum number of edges, and consider an embedding of G in the plane. Let H denote the subgraph of G induced by V_2 . In any face of the inherited embedding of H , there can be at most one vertex of V_1 , else an edge could be added between two vertices of V_2 on the boundary of the face and, therefore, G would not have a maximum number of edges as supposed. Let u be a vertex of degree at most 5 in H . The vertex u is on the boundary of at most 5 faces of H , and consequently in G , u has degree at most 10.

Theorem 3.4 (Downey and Fellows [169]) **PLANAR DOMINATING SET** *is solvable in time $O(11^k |G|)$.*

Proof. We construct a search tree for which each node has at most 11 children. Each node in the tree is labeled with a set of vertices S that represents a partially constructed dominating set.

The root node, labeled with the empty set, will have at most six children based on the following consideration. Since G is planar, G has a vertex v of degree at most 5 which can be found in linear time. Any dominating set for G must contain either v or one of the neighbors of v . We create a child node for each possible choice of a vertex to dominate v .

In general, for a node in the search tree, we first check whether S is a dominating set. This can be done in linear time. The levels of the tree correspond to the cardinality of the labels S , so the tree will have height at most k . To compute the children of a node, we find a vertex u in G not dominated by S that has degree at most 10 and create a child node for each possible choice y of a vertex to dominate u (there can be at most 11 possibilities, including u). The child node is labeled with $S \cup \{y\}$. We must argue that such a vertex u must be available; this being so, it can easily be found in linear time. We term such a vertex u a *splitter* for the node in the search tree.

Let U denote the set of vertices not dominated by S , and let $T = V - S - U$ be the set of vertices not in S and not in U . Let H be the subgraph of G induced by $V - S = T \cup U$, and let H' be the subgraph of H obtained by deleting from H any edges between vertices of T .

Observe that a set of vertices $W \subseteq V - S$ has the property that $S \cup W$ is a dominating set in G if and only if W is a dominating set in H' . In other words, we may restrict our attention to H' in searching for a splitter. H' satisfies condition (2) of Lemma 3.3, but there may be vertices in T that have degree 2 in H' . Necessarily, any such vertex $r \in T$ of degree 2 has two neighbors $s, t \in U$. Consider the graph H'' obtained from H' by deleting such vertices r and adding the edges st . Lemma 3.3 applies to H'' , so there is a vertex $u \in U$ in H'' of degree at most 10. The splitter vertex u also has degree at most 10 in H' , H , and G .

As our last example of the technique of bounded search trees, we consider the parameterized version of a classical problem in Garey and Johnson [239].

FEEDBACK VERTEX SET

Instance: A graph $G = (V, E)$.

Parameter: A positive integer k .

Question: Is there a set of vertices $V' \subseteq V$ of cardinality at most k such that $G - V'$ is acyclic?

Theorem 3.5 *FEEDBACK VERTEX SET is solvable in time $O((2k + 1)^k \cdot n^2)$.*

We will need the following lemma.

Lemma 3.6 (Itai and Rodeh [278]) *There is an $O(|G|^2)$ algorithm for the problem ALMOST MINIMUM below.*

ALMOST MINIMUM

Input: A graph G .

Output: Output ∞ if there is no cycle in G ; otherwise output a cycle of length $\leq m + 1$, where m is the length of the shortest cycle in G .

Proof. The algorithm runs as follows. Let $G = (V, E)$. For each $x \in V$, let $\text{ALMOST}(x)$ be the algorithm below.

Algorithm 3.7 ($\text{ALMOST}(x)$)

Step 1. Declare that x has $\text{level}(x) = 0$. Set $k(x) := \infty$.

Step $i + 1$. Using, say, a queue, examine the vertices at level i in order. (This is breadth first.) For each such vertex x' at level i , consider the set of neighbors of x' not at level $i - 1$. Suppose that we are considering v , a neighbor of a vertex x' at level i . It will be the case that v has no level and we declare that v has level $i + 1$.

Now see if there is any vertex $q \neq x'$ either at level i or already at level $i + 1$ by the time we consider v , and such that v is adjacent to q . If such a q exists, then terminate the algorithm and return $k(x) := i$.

If we consider all of level $i + 1$ and find no pair v, q , then move to Step $i + 2$, until there are no vertices without a level.

Now for each $x \in V$, $\text{ALMOST}(x)$ takes $O(|V|)$ many steps. Let $k_{\min} = \min\{k(x) : x \in V\}$. The claim is that if $k(x) = k_{\min}$, then x belongs to a minimal circuit of G . To establish the claim, suppose that w is a vertex that belongs to a minimal circuit. If m is even, then $\text{ALMOST}(w)$ stops when a vertex w' is found at level $i + 1$, completing a cycle at level i , and $k(w) = m/2 - 1$. (See Figure 3.2, even case.)

If m is odd, then we can either find w' completing a cycle at level i or completing a cycle at level $i + 1$, as we see in the two odd cases of Figure 3.2. In the odd case, $k(w) \leq (m - 1)/2$. Thus, $\text{ALMOST}(w)$ finds a cycle that is no longer than $2k_{\min} + 2$ and hence no longer than $m + 1$. It is clear that the cycle contains w .

Thus, for each $x \in V$, we compute $k(x)$. From the collection of $k(x)$, we compute $k_{\min}$ and then we are able to compute an almost minimal cycle for G . The cost is the cost of calling $\text{ALMOST}(x)$ $|V|$ many times. This is $O(|V|^2)$.

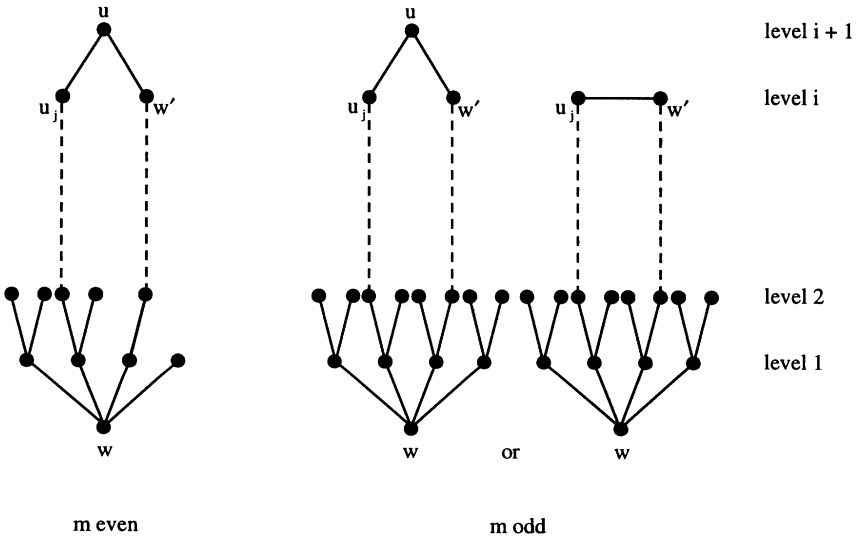


FIGURE 3.2. The Cases of Termination for $\text{ALMOST}(x)$

As Itai and Rodeh [278] observe, if we have some guarantee that the minimal cycle is of even length, then the algorithm above will return a minimal cycle. This observation applies to the case that G is bipartite.

Proof of Theorem 3.5. First, note that a graph G has a feedback vertex set of size k if and only if the *reduced* graph G' has one, where G' is obtained from G by replacing each maximal path in G having internal vertices all of degree 2 with a single edge. Note that the reduced graph G' may have loops and multiple edges, but that if G' is simple, then it has minimum degree 3. The reduced graph G' can be computed from G in linear time. Also, in linear time, a k -element feedback vertex set that has been identified in G' can be lifted to a k -element feedback vertex set in G .

As in the proofs of the theorems above, we build a search tree where each node is labeled with a set of vertices S representing a possible partial solution. The cardinality of a label corresponds to the height of the node in the tree, and we will, therefore, explore the tree to a height of no more than k . In linear time, we can check whether a set S is a solution. If the label set S of a node in the search tree is not a solution and the node has height less than k , then we can generate the children of the node, as follows.

Let H denote the graph $G - S$, and let H' be the reduction of H (as described above). If a vertex v of the H' has a self-loop, then v must belong to every feedback vertex set of H' . Corresponding to this observation, we create a single child node with label $S \cup \{v\}$.

If the reduced graph H' of the graph $H = G - S$ has multiple edges between a pair of vertices $u, v \in V(H)$, then either u or v must belong to every feedback vertex set of H' , and we correspondingly create two child nodes with labels, respectively, $S \cup \{u\}$ and $S \cup \{v\}$.

If the reduced graph H' has no loops or multiple edges, then we can make use of the following.

Claim. If a simple graph J of minimum degree 3 has a k -element feedback vertex set, then the *girth* of J (the length of a shortest cycle) is bounded above by $2k$.

We prove the Claim by induction on k . If J is simple, then by a standard result, J must contain a subdivision of K_4 ([335]), and this implies that a feedback vertex set must contain at least two elements.

For the induction step, suppose U' is a feedback vertex set consisting of $k + 1$ vertices of J . Suppose that $u, v \in U'$ with the distance from u to v , $d(u, v) \leq 2$ in J . Contracting the edges of a shortest path from u to v yields a graph J' of minimum degree 3 that has a feedback vertex set of k elements. By the induction hypothesis, there is a cycle C in J' of length at most $2k$. This implies that there is a cycle in J of length at most $2k + 2$. Otherwise, suppose no two vertices u and v of U' have $d(u, v) \leq 2$ in J . Then every vertex of $J - U'$ has degree at least 2, and so there is a cycle in J not containing any vertex of U' , a contradiction. This establishes our claim.

By the above claim, we know that for the node of the search tree that we are processing, either H' contains a cycle of length at most $2l$ where $l = k - |S|$ or S

cannot be extended to a k -element feedback vertex set. The algorithm of Itai and Rodeh [278] from Lemma 3.6 can be employed to find in H' , a cycle of length $2l$ or $2l + 1$ in time $O(n^2)$. Thus, in time $O(n^2)$, we can either decide that the node should be a leaf of the search tree (because there is no cycle in H' of length at most $2l + 1$) or we can find a short cycle and create at most $2l + 1$ children, observing that at least one vertex of the short cycle that we discover in H' must belong to any feedback vertex set.

As we will see in Chapter 8, **FEEDBACK VERTEX SET** can be shown to be solvable in time $O(|G|)$ (Downey and Fellows [165], Bodlaender [68]). The running time is $O(17k^4|G|)$. It is unknown if the directed version of **FEEDBACK VERTEX SET** is in *FPT*.

3.1.2 Heuristic Improvements, Shrinking the Search Tree

In some cases, it is possible to significantly improve the constants in the algorithms. We only look at the example of **VERTEX COVER**, which, despite its apparent simplicity, has a rich structure and admits many parameterized algorithms. Fix k and consider a graph G . Now if G does not have a vertex of degree 3 or more, then it is a fairly trivial graph consisting of a collection of cycles. For large G , such a graph cannot have a size k vertex cover. Thus, without loss of generality (at the expense of an additive constant factor), we will only concern ourselves with graphs with many vertices of degree 3 or greater. Now choose a vertex of degree 3 or greater, say v_0 . Now, either v_0 is in a vertex cover or *all of its neighbors are*. Thus, we can begin a search tree with one branch labeled v_0 and the other labeled $\{w_1, w_2, \dots, w_p\}$, where $p \geq 3$ and $\{w_1, w_2, \dots, w_p\}$ are the neighbors of v_0 . Again, consider the subgraphs of G not covered by the appropriate sets $\{v_0\}$ and $\{w_1, w_2, \dots, w_p\}$, respectively. In one, we need a size $k - 1$ vertex cover, but in the other, we only need a size $k - p$ vertex cover. Clearly, again we only consider graphs where we always have a degree 3 or greater vertex in the relevant subgraph. Thus, the complexity of the search is now determined not by a tree of size $O(2^k)$ but by a smaller one. The recurrence relation which generates the number of nodes in this new search tree is

$$a_{k+3} = a_{k+2} + a_k + 1, \quad a_0 = 0, \quad a_1 = a_2 = 1.$$

Here, we always generate one node as k increases by 1 and we get a splitting on the nodes “3 back.”

To estimate the solution for this recurrence, we will try $a_k = c^k$ and we want the best c . To take care of the 1 in the right-hand side, let us try $a_k = c^k - 1$, so that now we have to solve

$$c^{k+3} = c^{k+2} + c^k$$

or

$$c^3 = c^2 + 1.$$

When $c = 5^{1/4}$, it is true that $1 + \sqrt[4]{5} \leq 5^{3/4}$. Hence, $a_k \leq 5^{k/4} - 1$ can be verified by induction. This reasoning yields the following result.

Theorem 3.8 (Balasubramanian et al. [46]) *VERTEX COVER can be solved in time $O([5^{1/4}]^k |G|)$. (Note that $5^{1/4}$ is bounded above by 1.5, and hence this solution is feasible for $k \leq 70$.)*

It is unknown what the best estimate for c above is. It is unclear how much better one can do than $5^{1/4}$. With a little more effort, we can reduce to $c \leq 1.39$ as follows.

Theorem 3.9 (Balasubramanian et al. [46]) *There is an $O(e + (1.39)^k k^2)$ algorithm to find whether in an n vertex graph with e edges, it has a vertex cover of size k .*

Proof. We will assume that the graph is connected and is not a single cycle.

The algorithm consists of the following steps:

Step 0. First, reduce the problem to the kernel (see the next section). We assume that we have the sorted degree sequence. From now on, we spend $O(n)$, which is $O(k^2)$ time for each node in the search tree.

Step 1. If there is a degree 1 vertex, simply get rid of it and pick its neighbor in the vertex cover.

Step 2. Let x be a degree 2 vertex and let y and z be its neighbors. Out of y and z , exactly one, or both, or neither is in the vertex cover. If exactly one is in the vertex cover, then x has to be in the vertex cover to cover the edge xy or xz depending on whether z or y is in the vertex cover. Since x has no other role, we might as well pick the other vertex (y or z). So, without loss of generality, out of y and z , either both or neither will be in the vertex cover. Let C_k be the number of nodes in the search tree.

Case (a) Let y and z together have at least two neighbors (a and b , say) other than x .

Then branch as yz or xab ($C_k = 1 + C_{k-2} + C_{k-3}$).

Case (b) y and z have together only one neighbor a . Since the graph is not a cycle, a has at least another neighbor b . Now, a is in the vertex cover or not. Then branch as yzb or ax (same recurrence).

Step 3. If there is a vertex x of degree more than 4, then branch as x or $abcd$ ($C_k = 1 + C_{k-1} + C_{k-4}$).

Repeating Steps 1, 2, and 3, we will be left with a cubic graph.

Step 4. Let x be a vertex with neighbors 1, 2, and 3. In any vertex cover, if only two of the three vertices 1, 2, and 3 are present, then x must be in to cover the edge (x —the third vertex). Instead, we might as well include the third vertex. So without loss of generality, in any the vertex covers, $\{1, 2, 3\}$ or $\{1\}$ or $\{2\}$ or $\{3\}$ or none of 1, 2, and 3 will be there.

Case (a) The induced subgraph on 1, 2, and 3 is a triangle. Then the entire graph is K_4 , and pick any three of them in the vertex cover.

Case (b) The induced subgraph on 1, 2, and 3 contains a path 1-2-3. Then 1 and 3 have one other neighbor each (perhaps common, say 4). Also, 123 is in the vertex cover or only 2 is in the vertex cover. So branch as 123 or $x24$ ($C_k = 1 + 2C_{k-3}$).

Case (c) The induced subgraph on 1, 2, and 3 is (1-2 and 3). 3 has two other neighbors (say 4 and 5). Also, 1 or 2 must be in the vertex cover, and so 3 cannot be if not all are there. So branch as 123 or $x45$ ($C_k = 1 + 2C_{k-3}$).

Case (d) There is no edge among 1, 2, and 3.

Case (d.1) Two of 1, 2 and 3 (say 1 and 2) share a new neighbor, say 4. If all of {1, 2, 3} is not picked in the vertex cover, then, as we argued earlier, at most one of them will be in the vertex cover. So, x and 4 must be picked in that case. Branch as 123 or $x4$ ($C_k = 1 + C_{k-3} + C_{k-2}$).

Case (d.2) 1, 2, and 3 have disjoint neighbors. Here, all 5 cases apply and the recurrence is ($C_k = 1 + C_{k-3} + C_{k-7} + 3C_{k-6}$).

One can prove by induction that $(1.39)^k$ satisfies all the recurrence relations (Exercise 1).

Historical Notes

Except for Theorems 3.8 and 3.9, the results of this section are taken from Downey and Fellows [165] and [169].

Exercises 3.1

3.1.1. Verify that 1.39^k is an upper bound for the recurrences of Theorem 3.9.

The exercises below are all solvable by the method of bounded search trees.

3.1.2. Prove that **DOMINATING SET** restricted to graphs of maximum degree t is solvable in time $O((t+1)^k|G|)$.

3.1.3. (Downey and Fellows [165, 169]) Let q be fixed. Prove that the following problems are solvable in time $O(q^k|E|)$.

WEIGHTED MONOTONE q -CNF SATISFIABILITY

Input: Boolean formula E in conjunctive normal form, with no negated variables and maximum clause size q .

Parameter: A positive integer k .

Question: Does E have a weight k satisfying assignment? [The (Hamming) weight of an assignment is the number of variables made true in the assignment.]

WEIGHTED $\leq q$ -CNF SATISFIABILITY

Input: Boolean formula E in conjunctive normal form, with maximum clause size q .

Parameter: A positive integer k .

Question: Does E have a weight k' satisfying assignment for some $k' \leq k$?

(Hint: For the first one, simply modify the method used for **VERTEX COVER**. For the other, first see if there is a clause with only unnegated variables. If not, then we are done. If there is one, fork on the clause variables.)

3.1.4. (Downey and Fellows [169]) Prove that the following is solvable in time $O(12^k|G|)$.

PLANAR RED/BLUE DOMINATING SET

- Instance:* A planar bipartite graph $G = (V, E)$,
 $V = V_{red} \cup V_{blue}$.
- Parameter:* A positive integer k .
- Question:* Is there a set $V' \subseteq V_{red}$ of cardinality at most k such that every vertex of V_{blue} is adjacent to at least one vertex of V_{red} ?

(Hint: Modify the technique used for **PLANAR DOMINATING SET**. Here, build a search tree based on 12 children. Let $S \subseteq V_{red}$ be the label on a node in the search tree. Let $B(S) \subseteq V_{blue}$ denote the vertices in V_{blue} dominated by S . Let $T = V_{red} - S$ and $U = V_{blue} - B(S)$. It suffices to argue that there is a vertex $u \in U$ of degree at most 10 in the subgraph H induced by the vertices of $T \cup U$.

Let $T_i \subseteq T$ be the vertices of T in H of degree i , for $i = 1, 2$. Note that any two vertices x and y of T_1 adjacent to the same vertex of U in H are equivalent, in the sense that there is an extension of S that is a solution for G containing x if and only if there is a solution extension of S containing y . Thus, without loss of generality, we may assume that (*) each vertex of U in H is adjacent to at most one vertex of T_1 .

Let H' be the same graph as $H - T_1$, but considering each vertex of T_2 as a “virtual edge” between the two vertices of U to which it is adjacent. H' satisfies the conditions of Lemma 3.3 and, therefore, there is a vertex $u \in U$ of degree at most 10 in H' and in $H - T_1$ as well. Taking (*) into account, it suffices to create at most 12 children in the search tree for the node being processed.)

3.1.5. (Downey and Fellows [169]) Prove that the following problem is solvable in time $O(12^k|G|)$.

FACE COVER NUMBER FOR PLANAR GRAPHS

- Instance:* A planar graph $G = (V, E)$ together with an embedding of G in the plane.
- Parameter:* A positive integer k .
- Question:* Is there a set F of at most k faces of the embedding such that every vertex of G occurs on the boundary of at least one face $f \in F$?

(Hint. Use Exercise 4. Let G be plane. In linear time, reduce the problem of finding k faces of the embedding of G which cover all vertices of G to an instance of **RED/BLUE PLANAR DOMINATING SET**, by creating one red vertex for each face of the embedding of G and connecting it to each (blue) vertex on the boundary of the face.)

3.1.6. (Kaplan, Shamir and Tarjan [289]) A graph G is called *chordal* or *triangulated* if every cycle of length ≥ 4 contains a chord. If $G = \langle V, E \rangle$ is a graph, then a *fill-in* of G is a graph $G' = \langle V, E \cup F \rangle$ such that G' is chordal. The **MINIMUM FILL-IN** problem, which is also called **CHORDAL GRAPH COMPLETION** is the following:

MINIMUM FILL-IN

Input: A graph G .
Parameter: A positive integer k .
Question: Does G have a fill-in G' with $|F| \leq k$?

If k is allowed to vary, Yannakakis [466] has proven the problem to be *NP*-complete. Use the method of search trees to show that the parameterized version above is strongly uniformly *FPT*.¹ (Hint: Let c_n denote $\binom{2n}{n} \cdot 1/1 + n$, the n -th Catalan number. First, prove that the number of minimal triangulations of a cycle with n vertices is c_{n-2} . Build a search tree as follows. The root of the tree $\mathcal{T}$ is the graph G . To generate the children of a node G' of $\mathcal{T}$, first find a chordless cycle C of G' (i.e., of length ≥ 4) and let the children of the node labeled with G' correspond to the minimal triangulations of C . This only adds at most $c_{|C|-2}$ many children. Note that if we construct the complete tree, then each minimal triangulation of G will correspond to at least one leaf. Restrict the search to traverse paths in $\mathcal{T}$ that involve only k additional edges. One needs to prove that a chordless cycle can be found in a graph in time $O(|E|)$ (see Tarjan and Yannakakis [440]), to prove that all the minimal triangulations of a cycle C can be generated in $O(|C|)$ time (see Sleator, Tarjan, and Thurston [428]) and that the total number of nodes of $\mathcal{T}$ we need to visit is bounded by $2 \cdot 4^{2k}$. These observations give the running time of the algorithm to be $O(2^{4k}|E|)$.)

3.2 The Method of Reduction to a Problem Kernel

3.2.1 The Basic Method

The main idea of the method of reduction to a problem kernel is to reduce a problem instance I to an “equivalent” instance I' , where the size of I' is bounded by some function of the parameter k . The instance I' is then exhaustively analyzed, and a solution for I' can be lifted to a solution for I , in the case where a solution exists. Often this technique will lead to an *additive* rather than a *multiplicative* $f(k)$ exponential factor. We begin by reexamining the **VERTEX COVER** problem.

¹Kaplan, Shamir, and Tarjan [289] has used (essentially) the Problem Kernel Method of the next section to give a $O(k^5|V||E| + f(k))$ for suitably chosen recursive f . Finally, Leizhen Cai [103] also used the problem kernel method another way to give a $O(4^k((k+1)^{-3/2}[|E(G)||V(G)| + |V(G)|^2])$ time algorithm.

Theorem 3.10 (Sam Buss [100]) **VERTEX COVER** is solvable in time $O(n + k^k)$.

Proof. Observe that for a simple graph H any vertex of degree greater than k must belong to every k -element vertex cover of H .

Step 1: Locate all vertices in H of degree greater than k ; let p equal the number of such vertices. If $p > k$, there is no k -vertex cover. Otherwise, let $k' = k - p$.

Step 2: Discard all p vertices found in step 1 and the edges incident to them. If the resulting graph H' has more than $k'(k + 1)$ vertices, reject.

Step 3: If H' has no k' -vertex cover, reject. Otherwise, any k' -vertex cover of H' plus the p vertices from step 1 constitutes a k -vertex cover of H .

The bound $k'(k + 1)$ in step 2 is justified by the fact that a simple graph with a k' -vertex cover and degrees bounded by k has no more than $k'(k + 1)$ vertices. For fixed k , this makes step 3 a constant time operation, where the constant is $O(k^k)$.

Actually we can improve Buss' additive bounds by *first* using the problem kernel method and *then* using the method of search trees on the kernel itself. This technique yields the following.

Corollary 3.11 (Balasubramanian et al. [45]) **VERTEX COVER** is solvable in time $O(n + 2^k \cdot k^2)$

Proof. The constant is given via the technique of Theorem 3.1.

Notice that we can improve this result via the smaller search tree of Theorem 3.8 to get the following result.

Corollary 3.12 (Balasubramanian, et al. [45]) **VERTEX COVER** is solvable in time $O(n + (5^{1/4})^k \cdot k^2)$.

We can similarly solve the following problem.

k -LEAF SPANNING TREE

Instance: A graph $G = (V, E)$.

Parameter: A positive integer k .

Question: Is there a spanning tree of G with at least k leaves?

Theorem 3.13 (Downey, Doyle and Fellows²) **k -LEAF SPANNING TREE** is solvable in time $O(n + (2k)^{4k})$.

Proof. Note that any graph G that is a *yes* instance must be connected. A vertex v is called *useless* if (i) it has neighbors u and w of degree 2 and (ii) v has degree 2. We will argue that any sufficiently large graph without useless vertices of degree

²The original proof of this theorem in [169] contained a combinatorial error.

2 is necessarily a *yes* instance. Note also that if G has a vertex of degree at least k , then G is a *yes* instance.

Say that a useless vertex v is *resolved* by deleting v from G and adding an edge between u and w . Let G' denote the (*resolved*) graph obtained from G (in linear time) by resolving all useless vertices.

Our algorithm for k -LEAF SPANNING TREE is very simply described:

Step 1: Check whether G is a connected and whether there is a vertex of degree $\geq k$. If G has such a vertex, declare that the answer is *yes*.

Step 2: If the answer is still undetermined, then compute G' . If G' has at least $4(k+2)(k+1)$ vertices then, the answer is *yes*.

Step 3: Otherwise, exhaustively analyze G' and answer accordingly, since G' has a k -leaf spanning tree if and only if G does.

Our proof that the algorithm is correct employs the following fact.

Claim. If H is connected simple resolved graph of order at least $4(k+2)(k+1)$, then H has a spanning tree with at least k leaves.

To establish the claim, it can be shown by elementary induction that the following holds (Exercise 2).

(*) If a tree T has i internal (i.e., nonleaf) vertices of degree at least 3, then T has at least $i + 2$ leaves.

Suppose H satisfies the hypotheses of the above Claim. If H has any vertex of degree k , we are done, and hence, without loss of generality, we may presuppose that H has no vertex of degree $\geq k$. Let T be a spanning tree of H having a maximum number of leaves l , and suppose $l \leq k-1$. Now, as T has $4(k+2)(k+1)$ vertices but only l leaves, it must have at least $4(k+2)(k+1) - (k-3) - (k-1)$ vertices of degree 2 [by (*)]. Now, as the maximum degree in H of any vertex is $k-1$ and

$$4(k+1)(k+2) - (k-3) - (k-1) - (k-1)(k-1) = 3k^2 + 12k + 11,$$

it follows that there are at least $3(k+1)(k+3) + 2$ vertices of degree 2 in T that have the property that they are not connected to any leaf of T . There are now two cases.

Case 1. At least $k-2$ of the $3(k+1)(k+3) + 2$ internal vertices of degree 2 in T not connected to any leaf of T have degree 3 or more in H .

Let v be any such vertex not connected to any leaf of T having degree 3 or more in H . For such vertices, it is convenient to regard T as rooted with root v . Let v be such a degree 2 vertex of T . Regard T as a rooted tree with root v . Let $\{u_1, u_2\}$ be the children of v . Now, as v has degree 2 in T , and yet degree 3 or more in H , it follows that v is adjacent to some vertex w in H which is not a child of v (since H is simple), and furthermore, w is not a leaf of T by choice of v . We claim that there is such a choice of v such that one vertex of the path from v to w has degree

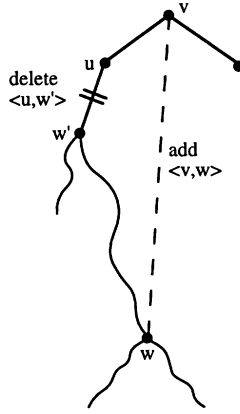


FIGURE 3.3. Increasing the Number of Leaves

2 in T . If not, then there are at least $k - 2$ internal vertices of degree 3 or more in T . But then by (*), there must be at least k leaves.

Thus, without loss of generality, suppose that there is a node u of degree 2 in T on the path from v to w . Let w' be the child of u . It is possible to change T into a new tree T' with more leaves as follows. We can make u a leaf by deleting the edge (u, w') and add an edge (v, w) where there is a path in T from w' to w . The reader should refer to Figure 3.3.

The point is that as w is not a leaf, this must increase the net number of leaves. **Case 2.** At most $k - 3$ of the $3(k + 1)(k + 3) + 2$ internal vertices of degree 2 in T not connected to any leaf of T have degree 3 or more in H .

Then each of the remaining $(3k + 8)(k + 1)$ vertices of degree 2 in both T and H are connected to at least one vertex of degree 3 or more in H (as H is resolved). Since each vertex of H has maximum degree $k - 1$ and each of the $(3k + 8)(k + 1)$ vertices are connected to at most one other such vertex (again as the graph is resolved), there are at least $\lceil \frac{(3k+8)(k+1)}{k-1} \rceil \geq 3k + 1$ vertices of degree 3 or more in H which are not leaves of T . These $3k + 1$ vertices of degree 3 or more in H must therefore be internal vertices of T . At most $k - 3$ of them can have degree 2 in T and therefore at least $2k + 4$ of them will have degree 3 or more in T . Therefore, by (*), T has at least $2k + 6$ leaves.

This concludes the proof of the claim.

It is now easy to see that the algorithm is correct since the derived graph, G' , will satisfy the hypotheses of the claim in step 2 of the algorithm.

Theorem 3.13 improves a result of Bodlaender, who showed that k -LEAF SPANNING TREE is linear-time fixed-parameter tractable with a multiplicative factor depending on k [65].

3.2.2 Hereditary Properties and Leizhen Cai's Theorem

Definition 3.14 (Hereditary Property) If Π is a property of graphs, we say a graph G is a Π graph iff G satisfies Π . We say that a property Π is a *hereditary property*, if given any Π graph G , whenever H is an induced subgraph of G , then H is a Π graph.

The literature is filled with many *graph modification* problems. As Leizhen Cai [103] remarks, in general these problems can be placed in the categories below.

- (1) The *edge deletion problem*: Find a set of edges of minimum cardinality whose removal results in a Π graph.
- (2) The *vertex deletion problem*: Find a set of vertices of minimum cardinality whose removal results in a Π graph.
- (3) The *edge/vertex deletion problem*: Categories (1) and (2) combined.
- (4) The *edge addition problem*: Find a set of new edges of minimum cardinality whose addition results in a Π graph.

It is known that categories (1), (2) and (3) are usually *NP-hard* for any nontrivial hereditary property. Leizhen Cai [103] considered the following general parametric version of the modification problem.

$\Pi_{i,j,k}$ GRAPH MODIFICATION PROBLEM

Input: A graph $G = \langle V, E \rangle$.

Parameters: Non-negative integers i , j , and k .

Question: Can we delete at most i vertices, j edges, and add at most k edges and get a Π graph?

We will say that a property Π has a *forbidden set characterization* iff there exist a set $\mathcal{F}$ of graphs such that G is a Π graph iff G does not contain a member of $\mathcal{F}$ as an induced subgraph. If we can take $\mathcal{F}$ to be finite, then we say that Π has a finite forbidden set characterization.

Clearly, if $\mathcal{F}$ is a finite set characterizing Π , then there is a P -time recognition algorithm for the $\Pi_{i,j,k}$ GRAPH MODIFICATION PROBLEM for any fixed i , j and k as follows. Let N be the maximum number of vertices of any graph in $\mathcal{F}$. Given a graph Q , we can check whether Q is a Π graph in time $O(|Q|^N)$ by checking if Q has an induced subgraph with $\leq N$ vertices in $\mathcal{F}$. Therefore, the $\Pi_{i,j,k}$ GRAPH MODIFICATION PROBLEM is solvable in time $O(|G|^{i+2j+2k+N})$. However, Cai used essentially the problem kernel method to prove the result below.

Theorem 3.15 (Leizhen Cai [103]) *Suppose that Π is any property with a finite forbidden set characterization. Then the $\Pi_{i,j,k}$ GRAPH MODIFICATION PROBLEM is FPT in time $O(N^{i+2j+2k}|G|^{N+1})$ where N denotes the maximum size of the vertex set of any graph in the forbidden set $\mathcal{F}$.*

We need the lemma below.

Lemma 3.16 (Leizhen Cai [103]) *For any hereditary property Π , if Π is recognizable in time $T(G)$ for any graph G , then for any graph G which is not a Π graph, a minimal forbidden induced subgraph for Π can be found in time $O(|V(G)|T(G))$.*

Proof. Let $\mathcal{A}$ be the recognition algorithm for Π running in time $T(G)$, so that $\mathcal{A}(G) = 1$ iff G is a Π graph. So first use $\mathcal{A}(G)$ to see if G is a Π graph. Assuming that the answer is “no,” we perform the following algorithm. Order the vertices of G as $\{v_0, \dots, v_n\}$. The algorithm proceeds in stages. At stage 0, $G_0 = G$. At stage s , using $\mathcal{A}$ applied to $G_s - \{v_s\}$, see if $G_s - \{v_s\}$ is a Π graph. If the answer is yes, then v must belong to a minimal forbidden induced subgraph. And we let $G_{s+1} = G_s$. If the answer is no, then we define $G_{s+1} = G_s - \{v_s\}$. In this case, $G_s - \{v_s\}$ must contain a minimal forbidden induced subgraph.

Clearly, if G is not a Π graph, then G_{n+1} is a minimal induced forbidden subgraph, as required.

Proof of Theorem 3.15. Fix i , j and k . Assuming that G is not a Π graph, we will repeat the following steps until we either get a Π graph, or we have used up the deletion of $\leq i$ vertices, $\leq j$ edges, and the addition of $\leq k$ edges. In this latter case, we output “no.”

Step 1. Find a minimal forbidden induced subgraph H of G .

Step 2. Modify G either by deleting an edge or a vertex, or adding an edge to H . (The crucial idea is to modify only minimal induced forbidden subgraphs.)

It is clear that the algorithm works. Now using the finite set characterization, we can recognize if G is a Π graph in time $O(|V(G)|^N)$. Therefore by Lemma 3.16, we can compute a minimal induced forbidden subgraph H of G in time $O(|V(G)|^{N+1})$. Since N is the largest number of vertices of any member of the forbidden set $\mathcal{F}$, there are at most $\binom{N}{2}$ many ways to add or delete an edge of H , and at most N many ways to delete a vertex from H . The total number of graphs we can generate in the algorithm above is thus $N^i \cdot \binom{N}{2}^{j+k} = O(N^{i+2j+2k})$. The running time of the algorithm is thus $O(N^{i+2j+2k}|V(G)|^{N+1})$ as required.

The reader should note that there are several interesting $\Pi_{i,j,k}$ graph modification problems that are *not* characterized by a finite forbidden set and yet are still *FPT*. One important example is **MINIMUM FILL-IN** (also called **CHORDAL GRAPH COMPLETION**), which asks if we can add k edges to G to make a chordal graph. (A chordal graph where any cycle of length greater than 3 contains a chord.) This problem is very important in the area of computational biology called perfect phylogeny. Chordal graph completion is a $\Pi_{0,0,k}$ graph modification problem which is classically *NP-hard* (Yannakakis [466]). There is no finite forbidden set characterization (Exercise 1). Nevertheless, chordal graph completion is *FPT*. Chordal graph completion provides an interesting example of an *FPT* problem since there are several algorithms to demonstrate this fact. Kaplan, Shamir and Tarjan [289] used the method of bounded search trees to show that the parameterized version above is strongly uniformly *FPT* in time $O(2^{4k}|E(G)|)$. (Exercise 3.1.6) Kaplan, Shamir and Tarjan [289] also used (essentially) the problem kernel method to give a $O(k^5|V(G)||E(G)| + f(k))$ for suitably chosen

recursive f . Finally Leizhen Cai [103] also used the problem kernel method to give a $O(4^k((k+1)^{-3/2}[|E(G)||V(G)| + |V(G)|^2])$ time algorithm. There are a number of interesting related problems which are not known to be *FPT*. They include **INTERVAL GRAPH COMPLETION**, **CHORDAL GRAPH DELETION**, **BIPARTITE GRAPH DELETION**, and **INTERVAL GRAPH DELETION**. In each case the parameter is a positive integer k and the problem is to add or delete (as the case may be) k edges to make the graph have the relevant property.

Exercises 3.2

- 3.2.1.** Consider the hereditary property $\Pi_{0,0,k}$ which states that we can add $\leq k$ edges to G and make a chordal graph. Prove that $\Pi_{0,0,k}$ has no finite forbidden set characterization.
- 3.2.2.** Prove (*) from Theorem 3.13; that is, prove the following.
If a tree T has i internal vertices of degree at least 3, then T has at least $i + 2$ leaves.
- 3.2.3.** Show that the following problem is *FPT* by the method of reduction to a problem kernel:

SET BASIS (Garey and Johnson, [239, Problem SP7])

Instance: A collection C of subsets of a finite set S .
Parameter: A positive integer k .
Question: Is there a collection B of subsets of S with $|B| = k$ such that, for every set $A \in C$, there is a subcollection of B whose union is exactly A ?

- 3.2.4.** Prove that the following problem is *FPT*.

GROUPING BY SWAPPING (Garey and Johnson, [239, Problem SR21])

Instance: A finite alphabet Σ , a string $x \in \Sigma^*$.
Parameter: A positive integer k .
Question: Is there a sequence of k or fewer adjacent symbol interchanges that transforms x into a string x' in which all occurrences of each symbol $a \in \Sigma$ are in a single block?

- 3.2.5.** The following problem is of some interest due to its potential connections to combinatorial cryptosystems (Fellows and Koblitz, [213, 214]). By reducing to a problem kernel, show that it is *FPT*.

UNIQUE HITTING SET

Instance: A set X and k subsets $X_1, \dots, X_k$ of X .
Parameter: A positive integer k .
Question: Is there a set $S \subseteq X$ such that for all i , $1 \leq i \leq k$, $|S \cap X_i| = 1$?

3.2.6. Show that the following two (similar) problems are *FPT* by the problem kernel method.

CONSECUTIVE BLOCK MINIMIZATION (Garey and Johnson, [239, Problem SR17])

- Instance:* An $m \times n$ matrix M of 0's and 1's and a positive integer k .
Parameter: A positive integer k .
Question: Is it possible to permute the columns of M to obtain a matrix M' that has at most k blocks of consecutive 1's? (A *block* of consecutive 1's is an interval of a row such that all entries in that interval are 1's.)

RECTILINEAR PICTURE COMPRESSION (Garey and Johnson, [239, Problem SR25])

- Instance:* An $n \times n$ matrix of 0's and 1's and a positive integer k .
Parameter: A positive integer k .
Question: Are there k rectangles that cover all the 1's?

3.2.7. (Parametric Duality) Classically, a problem that takes as input a graph G and a positive integer $k \leq n$ where n is the size of G is “the same” as the problem where the integer part of the input is presented as $n - k$. Parametrically, however, such “dual” problems are fundamentally different and may have different complexities. Show that the following parametric dual of the usual graph coloring problem is *FPT* by the method of reduction to a problem kernel. (Due to Jan Telle.)

DUAL OF COLORING

- Instance:* A graph G .
Parameter: A positive integer k .
Question: Can G be properly colored by $n - k$ colors, where n is the number of vertices of G ?

The **VERTEX COVER** problem is parametrically dual to the **INDEPENDENT SET** problem. An *irredundant set* of vertices in a graph $G = (V, E)$ is a set of vertices $V' \subseteq V$ such that for every vertex $u \in V'$, there is a vertex v in the neighborhood of u , but not in the neighborhood of any other vertex of V' . Prove that the following problem is *FPT* by the method of reduction to a problem kernel.

DUAL OF IRREDUNDANT SET

- Instance:* A graph G .
Parameter: A positive integer k .
Question: Does G have an irredundant set of size $n - k$?

- 3.2.8.** (V. Raman) If E is a 3SAT expression having n clauses, then there is always a truth assignment that satisfies at least $\lceil n/2 \rceil$ clauses. By the method of reduction to a problem kernel, show that the following problem is *FPT*.

MORE THAN HALF MAX-3SAT

Instance: A 3SAT expression E .
Parameter: A positive integer k .
Question: Is there a truth assignment that satisfies at least $\lceil n/2 \rceil + k$ clauses?

(Hint: Say that E is *homogenous* if each clause consists only of positive literals or only of negative literals. First, reduce to homogenous expressions.)

- 3.2.9.** Show that the following problems are *FPT* by a combination of search tree and problem kernel techniques

(Hint: First, reduce part of the problem to **VERTEX COVER**.)

MATRIX DOMINATION (Garey and Johnson, [239, Problem MS12])

Instance: An $n \times n$ matrix M with entries from $\{0, 1\}$.
Parameter: A positive integer k .
Question: Is there a set C of k or fewer nonzero entries in M that dominate all others, in the sense that every nonzero entry in M is in the same row or in the same column as some element of C ?

NEARLY A PARTITION

Instance: A finite set X , a family F of subsets of X .
Parameter: A positive integer k .
Question: Is there a subfamily $F' \subseteq F$ with $|F'| \leq k$ such that $F - F'$ is a partition of X ?

- 3.2.10.** (D. Bryant) Show that **HITTING SET FOR SIZE THREE SETS** is fixed-parameter tractable by the method of reduction to a problem kernel. (The problem arises in computing evolutionary phylogenies. The relationship of triples of species can be computed first, and then this information combined.)

HITTING SET FOR SIZE THREE SETS (Garey and Johnson, [239])

Instance: A collection C of 3-element subsets of a set S .
Parameter: A positive integer k .
Question: Does S contain a *hitting set* for C of size at most k , that is, a subset $S' \subseteq S$ with $|S'| \leq k$ and such that S' contains at least one element from each set in C ?

- 3.2.11.** (T. Wareham) The following “Steiner problem” in hypercubes arises in computing maximum parsimony phylogenies for k species based on binary character information. (The q -dimensional hypercube is the

graph with vertex set consisting of all length q binary vectors, with two vertices adjacent if and only if the corresponding vectors differ in a single component.) Prove that the problem is *FPT* by reducing to a problem kernel.

STEINER PROBLEM IN HYPERCUBES

Instance: k length q binary sequences $X_1, \dots, X_k$ and a positive integer M (given in binary).

Parameter: A positive integer k .

Question: Is there a subgraph S of the q -dimensional binary hypercube that includes the vertices $X_1, \dots, X_k$, such that S has at most M edges?

4

Optimization Problems, Approximation Schemes, and Their Relation with *FPT*

Optimization problems are at the heart of complexity theory. In practice, we usually wish to optimize some objective function (e.g., find a least-cost tour) rather than solve a related decision problem. Furthermore, even if we demonstrate that some problem has no fast solution, it is often acceptable to find an approximate solution to some desired performance ratio. In this section, we will see that parametric complexity and approximability are deeply related.

In particular, in this chapter, we point out some very interesting connections between the concept of parameterized tractability and the notion of having a polynomial-time approximation scheme. This connection will allow us to tie in with the literature on these well-studied concepts. We refer the reader to Garey and Johnson [239], Chapter 6, for more details on polynomial-time approximation algorithms. We will also relate parametric tractability to a syntactic class called *MAXSNP* and another called $MINF^+ \Pi_1(h)$. The relationship between these classes and *FPT* enable us to give sufficient conditions to say that a problem is *FPT* from its syntactic description alone.

4.1 Optimization Problems

The following definitions are classical and can be found in Garey and Johnson [239], for example.

Definition 4.1 An NP optimization problem Q is either a minimization problem or a maximization problem and is given as a 4-tuple (I_Q, S_Q, f_Q, opt_Q) , where

- (i) I_Q is the set of input instances. I_Q is recognizable in polynomial time.

- (ii) $S_Q(x)$ is the set of *feasible solutions* for the input $x \in I_Q$ such that there is a polynomial p and a polynomial-time computable relation Π (p and Π only depend on Q) such that for all $x \in I_Q$, $S_Q(x)$ can be expressed as $S_Q(x) = \{y : |y| \leq p(|x|) \wedge \Pi(x, y)\}$.
- (iii) $f_Q(x, y) \in \mathbb{N}$ is the *objective function* for each $x \in I_Q$ and $y \in S_Q(x)$. The function f_Q is computable in polynomial time.
- (iv) $opt_Q \in \{\max, \min\}$.

An *optimal solution* for an input instance $x \in I_Q$ is a feasible solution $y \in S_Q(x)$ such that $f_Q(x, y) = opt_Q\{f_Q(x, z) : z \in S_Q(x)\}$. We write

$$opt_Q(x) = opt_Q\{f_Q(x, z) : z \in S_Q(x)\}.$$

An algorithm A is an *approximation algorithm* for Q if, given any input instance x in I_Q , A finds a feasible solution $y_A(x)$ in $S_Q(x)$. A maximization problem (resp. minimization problem) Q is *polynomial-time approximable to a ratio $r(n)$* if there is a polynomial-time approximation algorithm A for Q such that the *relative error*

$$\mathfrak{R}_A(x) = \frac{opt_Q(x) - f_Q(x, y_A(x))}{f_Q(x, y_A(x))}$$

$$\left(\text{resp. } \mathfrak{R}_A(x) = \frac{f_Q(x, y_A(x)) - opt_Q(x)}{opt_Q(x)} \right)$$

is bounded by $r(|x|) - 1$ for all input instances $x \in I_Q$. An optimization problem Q has a *polynomial-time approximation scheme* if there is an algorithm A that takes a pair $x \in I_Q$ and $\epsilon > 0$ as input and outputs a feasible solution in $S_Q(x)$ such that the relative error is bounded by ϵ for all input instances $x \in I_Q$ and the running time of A is bounded by a polynomial of $|x|$ for each fixed ϵ . We further state that the optimization problem Q has a *fully polynomial-time approximation scheme* if the running time of the algorithm A is bounded by a polynomial of $|x|$ and $1/\epsilon$.

4.2 How FPT and Optimization Problems Relate

In this section, we demonstrate an interesting relationship between fixed-parameter tractability and approximability of *NP* optimization problems. In particular, we will show the fixed-parameter tractability for some well-known *NP* optimization classes, and investigate the approximability of *NP* optimization problems based on their fixed-parameter tractability.

We remark that this has more than passing interest. Aside from the original work in Garey and Johnson [239], there have been few advances which enable one to prove that an approximation problem has no good approximation algorithm, even assuming $P \neq NP$. The exception to this is the recent work of Arora et al. [41] who demonstrated that many *NP* optimization problems do not have good

approximation algorithms unless $P \neq NP$. We shall show that in many cases, fixed-parameter intractability of an NP optimization problem implies nonapproximability of the problem. This result applies to many problems *not* covered by the Arora et al. results. Thus, the study of fixed-parameter complexity provides a new and potentially powerful approach to proving nonapproximability of NP optimization problems. Our concern will be parameterized problems associated with *optimization problems*. To this end, for a relation R from $\{\geq, \leq, =\}$, for example, we can associate a natural decision problem.

DECISION PROBLEM ASSOCIATED WITH THE NP OPTIMIZATION PROBLEM $Q = (I_Q, S_Q, f_Q, opt_Q)$.

Input: $x \in I_Q$.
Parameter: A positive integer k .
Question: Does $R(opt_Q(x), k)$ hold?

We write Q_R for the decision problem associated with R . The complexity of the parameterized problems $Q_ =$, $Q_ \leq$, and $Q_ \geq$ have been studied in the literature. For example, Leggett and Moore [327] showed that for many NP optimization problems Q , the problems $Q_ \leq$ and $Q_ \geq$ are in $NP \cup coNP$, whereas the problem $Q_ =$ is not in $NP \cup coNP$ unless $NP = coNP$. With the following elementary argument, Liming Cai and Chen demonstrated that the problems $Q_ =$, $Q_ \leq$, and $Q_ \geq$ have the same complexity *from the viewpoint of fixed-parameter tractability*.

Lemma 4.2 (Cai and Chen [108]) *Let Q be an arbitrary NP optimization problem. Then the three parameterized problems $Q_ =$, $Q_ \leq$, and $Q_ \geq$ are either all (strongly) (uniformly) fixed-parameter tractable, or all (strongly) (uniformly) fixed-parameter intractable.*

Proof. We argue for uniformly FPT; the other arguments are similar. Suppose that the parameterized problem $Q_ \leq$ is fixed-parameter tractable. Thus, there is an algorithm $A_ \leq$ that solves the problem $Q_ \leq$ in time $O(f(k)n^c)$, where f is a function and c is a constant independent of the parameter k . To show that the parameterized problem $Q_ =$ is fixed-parameter tractable, note that the condition $k = opt_Q(x)$ is equivalent to the condition $(k \leq opt_Q(x)) \ \& \ ((k+1) > opt_Q(x))$. Therefore, given an instance $\langle x, k \rangle$ for the parameterized problem $Q_ =$, we can run the algorithm $A_ \leq$ on the input $\langle x, k \rangle$ and on the input $\langle x, k+1 \rangle$ to decide whether $k = opt_Q(x)$. The total running time is obviously bounded by $O((f(k) + f(k+1))n^c)$. Thus, the problem $Q_ =$ is fixed-parameter tractable.

Conversely, suppose that the problem $Q_ =$ is fixed-parameter tractable. Then there is an algorithm $A_ =$ that solves the problem $Q_ =$ in time $O(f'(k)n^{c'})$, where f' is a recursive function and c' is a constant independent of the parameter k . Now, given an instance $\langle x, k \rangle$ of the parameterized problem $Q_ \leq$, we run the algorithm $A_ =$ on k inputs $\langle x, 0 \rangle, \langle x, 1 \rangle, \dots, \langle x, k-1 \rangle$. It is easy to see that $\langle x, k \rangle$ is a no-instance of the parameterized problem $Q_ \leq$ if and only if the algorithm $A_ =$ concludes yes for at least one of those k inputs. Note that the running time of the above process is bounded by $O(kf'(k)n^{c'})$. Thus, the problem $Q_ \leq$ is fixed-parameter tractable.

Therefore, the parameterized problem $Q_ =$ is fixed-parameter tractable if and only if the parameterized problem $Q_ \leq$ is fixed-parameter tractable. In a similar way, we can show that the parameterized problem $Q_ \geq$ is fixed-parameter tractable if and only if the parameterized problem $Q_ =$ is fixed-parameter tractable.

The key result tying *FPT* with approximability is the following theorem which extends earlier work of Liming Cai [104] on pseudo-polynomial time and *FPT*.

Theorem 4.3 (Cai [104], Liming Cai and Chen [108]) *If an NP optimization problem has a fully polynomial-time approximation scheme, then it is fixed-parameter tractable.*

Proof. Suppose that an NP optimization problem $Q = (I_Q, S_Q, f_Q, \text{opt}_Q)$ has a fully polynomial-time approximation scheme A . Then, the algorithm A takes as input both an instance $x \in I_Q$ of Q and an accuracy requirement $\epsilon > 0$, and then outputs a feasible solution in time $O(p(1/\epsilon, |x|))$ such that the relative error $\Re_A(x)$ is bounded by ϵ , where p is a two-variable polynomial.

First, we assume that Q is a maximization problem. By Lemma 4.2, we only need to show that the problem $Q_ \leq$ is fixed-parameter tractable. Given an instance $\langle x, k \rangle$ for the parameterized problem $Q_ \leq$, we run the algorithm A on input x and $1/2k$. In time $O(p(2k, |x|))$, the algorithm A produces a feasible solution $y \in S_Q(x)$ such that

$$\Re_A(x) = \frac{\text{opt}_Q(x) - f_Q(x, y)}{f_Q(x, y)} \leq \frac{1}{2k} < \frac{1}{k}.$$

If $k \leq f_Q(x, y)$, then certainly $k \leq \text{opt}_Q(x)$ since Q is a maximization problem. On the other hand, if $k > f_Q(x, y)$, then $k - 1 \geq f_Q(x, y)$. Combining this with the inequality $(\text{opt}_Q(x) - f_Q(x, y))/f_Q(x, y) < 1/k$, we get $k > \text{opt}_Q(x)$ immediately. Therefore, $k \leq \text{opt}_Q(x)$ if and only if $k \leq f_Q(x, y)$. Since the feasible solution y can be constructed by the algorithm A in time $O(p(2k, |x|))$, we conclude that the NP optimization problem Q is fixed-parameter tractable.

The case when Q is a minimization problem can be proved similarly by showing that the problem $Q_ \geq$ is fixed-parameter tractable.

Actually, the proof above demonstrates that any optimization problem with a polynomial-time approximation algorithm $A(x, \epsilon)$ with the additional property that there exists a constant c independent of ϵ such that $A(x, \epsilon)$ runs in time $O(|x|^c)$ for a fixed ϵ is fixed-parameter tractable.

We remark that Theorem 4.3 allows us to establish that many problems are *FPT*. For instance, **BOUNDED KNAPSACK** (below) is *FPT*.

BOUNDED KNAPSACK

- Input:** $U \subseteq \mathbb{N}$ with sizes $s(a) \in \mathbb{Z}$ for $a \in U$, and $B \leq k$.
Parameter: A positive integer k .
Question: Is there some $S \subseteq U$ with $\sum_{a \in S} s(a) = B$?

This follows by the algorithm of Ibarra and Kim [275]. Similarly, from the approximation algorithm of Lipton and Tarjan [333] shows that the problem **PLANAR INDEPENDENT SET** below is *FPT*.

PLANAR INDEPENDENT SET

Input: A planar graph G .
Parameter: A positive integer k .
Question: Does G have an independent set of size k ? Here, a set S of vertices of G is called an independent set iff for all $x, y \in S$, xy is not an edge of G .

A final example, by the randomized fully polynomial-time approximation scheme of Brightwell and Winkler [95, 96] the following problem is randomized *FPT* (with the obvious meaning).

LINEAR EXTENSION COUNT

Input: A partially order set P .
Parameter: A positive integer k .
Question: Does P have at least k linear extensions? [A linear extension L of a poset $(\{a_1, \dots, a_n\}, \leq)$ is a linear ordering $(\{a_1, \dots, a_n\}, \leq^*)$ such that if $a_i \leq a_j$ in P , then $a_i \leq^* a_j$ in L .]

The problem of counting linear extension is shown (in Brightwell and Winkler [95] [96]) to be $\#P$ complete.

Theorem 4.3 implies many other natural problems are *FPT*. One powerful application comes from the contrapositive.

Corollary 4.4 (Cai and Chen [108]) *If an NP optimization problem is fixed-parameter intractable, then it has no fully polynomial-time approximation scheme. In particular, under the current working hypothesis that $W[1] \neq FPT$ of Chapter 10, the NP optimization problems that are $W[1]$ -hard under the uniform reduction have no fully polynomial-time approximation scheme.*

Following Liming Cai and Chen [108], we remark that Corollary 4.4 complements a classical result of Garey and Johnson (Theorem 6.8 in [239], page 140) that a strongly *NP*-hard optimization problem Q with $opt_Q(x) < p(|x|, num(x))$ for some two-variable polynomial p has no fully polynomial-time approximation scheme unless $P = NP$, where $num(x)$ is the largest integer appearing in the input instance x . The consequence $W[1] = FPT$ in Corollary 4.4 is weaker than $P = NP$, but the assumption in Corollary 4.4 requires neither strong *NP*-hardness nor the condition $opt_Q(x) < p(|x|, num(x))$.

There are a number of applications of Theorem 4.3. One not covered by the classical literature is that of the so-called **VC-DIMENSION** from computational learning theory has been studied extensively in the literature. As we see in Chapter 11, the **VC-DIMENSION** is $W[1]$ -hard, and so cannot have a fully polynomial-time approximation scheme unless $W[1] = FPT$. However, Papadimitriou and

Yannakakis [368] have shown strong evidence that this problem is in *NP* but neither in *P* nor *NP*-complete. Therefore, Theorem 6.8 of Garey and Johnson [239] is not applicable. The conclusion is that **VC-DIMENSION** is unlikely to have a fully polynomial-time approximation scheme even though it seems easier than *NP*-complete problems.

By making a natural distinction between different kinds of polynomial-time approximation schemes, an even stronger connection between fixed-parameter intractability and nonapproximability can be established. Consider the following results.

- For the problem **EUCLIDEAN TRAVELLING SALESMAN** there is a **PTAS** (due to Arora [38]) that runs in time $O(n^{30/\epsilon})$ where n is the size of the graph.
- For the problem **MAXIMUM INDEPENDENT SET FOR PLANAR GRAPHS** there is a **PTAS** (due to Baker [44]) that runs in time $O(f(1/\epsilon)n)$.

What we can notice about the above examples is that the running times for an approximate solution within factor of $(1 + \epsilon)$ of optimal are, for fixed ϵ , bounded by a polynomial, but in the first case the degree of the polynomial is a function of ϵ , whereas in the second case the exponent of the polynomial is independent of ϵ . This is precisely the central issue in parameterized complexity, so it is natural to make the following definition (due first to Bazgan [54]).

Definition 4.5 An approximation algorithm for an optimization problem is an *efficient PTAS* if it computes a solution with a factor on $(1 + \epsilon)$ of optimal in time $O(f(\epsilon)n^c)$ for some function f and some constant c .

Bazgan in [54] was the first to prove the following important theorem which has subsequently been independently rediscovered by several others (see, e.g., [120]).

Theorem 4.6 *If an NP optimization problem is fixed-parameter intractable, then it has no efficient PTAS unless $FPT = W[1]$.*

The theorem is proved by essentially the same argument as for Theorem 4.3. Bazgan's Theorem provides a very good connection between parametric intractability and nonapproximability.

4.3 The Classes *MAXSNP*, $MIN F^+ \Pi_1(h)$, and *FPT*

Cai and Chen [107, 108] were able to prove that large classes of important optimization problems have corresponding parameterized problems which are fixed-parameter tractable. These problems are defined from subclasses of the collection of *NP*-complete which are derived syntactically.

The class *MAXSNP* was introduced by Papadimitriou and Yannakakis [366]. Many important natural and well-studied *NP* optimization problems belong to this class. The class comes from consideration of the syntactic form of *NP* problems

via Fagin's Theorem below. Fagin's Theorem is known in finite model theory as $NP = \Sigma_1^1$. More precisely, the exact statement of the theorem is the following. (For a more detailed discussion of second-order logic, we refer the reader to Section 6.5, where the specific case of graphs is discussed in detail.)

Theorem 4.7 (Fagin's Theorem, [201]) *NP is the class of all problems that can be put in the form*

$$\{I : \exists S \in 2^{[n]^k} \forall x \in [n]^p \exists y \in [n]^q \phi(I, S, x, y)\}.$$

Here $I \subseteq [n]^m$ is the input relation, x and y are tuples ranging over $[n] = \{1, 2, \dots, n\}$, and ϕ is quantifier free.

For instance, **SATISFIABILITY** can be written as $\exists T \forall c \exists x [(P(c, x) \wedge x \in T) \vee (N(c, x) \wedge x \notin T)]$.

Proof Sketch. We sketch the proof, leaving the details to the reader (Exercise 2). First, to get $\Sigma_1^1 \subseteq NP$, let $\mathcal{C} = \{\mathcal{A} : \mathcal{A} \models \exists R \psi\}$, with ψ first order, as above. Then, given $\mathcal{A}$, we can guess relations R and verify that $(\mathcal{A}, R) \models \psi$, which is polynomial-time. Conversely, $NP \subseteq \Sigma_1^1$, as follows. Given an NP language L , and hence a nondeterministic polynomial-time machine M , we have that $\mathcal{A} \in L$ iff M accepts the encoding of $(\mathcal{A}, <)$. One then needs only construct a first-order formula $\psi(<, X)$ which means “ X encodes an accepting computation of M on the encoding of $(\mathcal{A}, <)$.” Then $L = \{\mathcal{A} : \mathcal{A} \models \exists < \exists X (“<” \text{ is a total order} \wedge \psi)\}$.

The key to the following is that some problems have a form that is *simpler* than the basic Fagin Theorem. We can write **3SAT** with one less alternation of quantifiers by assuming that the input consists of four relations C_0, C_1, C_2 , and C_3 , where C_i consists of all clauses with exactly i negative literals, with $(x_1, x_2, x_3) \in C_j$ means that there is a clause with $x_1, \dots, x_j$ occurring negatively and $x_{j+1}, \dots, x_3$ positively. **3SAT** is then

$$\exists T \forall (x_1, x_2, x_3) [\wedge_{j=0}^3 ((x_1, x_2, x_3) \in C_j \rightarrow (\wedge_{i \leq j} x_i \notin T \wedge \wedge_{i > j} x_i \in T))].$$

The problems such as **3SAT** which can be expressed as $\exists S \forall x \psi(x, G, S)$ constitute a class called *simple NP* or *SNP*. For each $\Pi \in NP$, one can then define the *maximization version* $MAX \Pi$ as

$$\max_S |\{x : \exists y \psi(x, y, G, S)\}|.$$

The immediate relevance of all of the above to our studies comes from the following consequential definition.

Definition 4.8 (Papadimitriou and Yannakakis [366]) A maximization problem $Q = (I_Q, S_Q, f_Q, opt_Q)$ is in the class $MAXSNP$ if its optimum $opt_Q(X)$, $X \in I_Q$, can be expressed as

$$opt_Q(X) = \max_S |\{v : \psi(v, X, S)\}|,$$

where the input instance $X = (U, P^1, \dots, P^b)$ is described by a finite structure over the finite universe U and P^i is a predicate of arity r_i for some integer $r_i \geq 1$, S is also a finite structure over the universe U , v is a vector of fixed arity of elements in U , and ψ is a quantifier-free formula.

Despite the fact that Papadimitriou and Yannakakis [366] proved that any problem in *MAXSNP* can be approximated to within some constant ratio, Arora et al. [41] proved the beautiful result that *MAXSNP-complete*¹ optimization problems have no fully polynomial-time approximation scheme unless $P = NP$. Because of this, we cannot use Theorem 4.3 to demonstrate that *MAXSNP* problems are *FPT*. Nevertheless, Liming Cai and Chen were able to establish the fixed-parameter tractability of all problems in *MAXSNP*.

Theorem 4.9 (Cai and Chen [108]) *All maximization problems in the class MAXSNP are fixed-parameter tractable.*

Proof. Let c be a fixed constant. Consider the following *MAX c-SAT* problem.

MAX c-SAT

- Input:* A set of Boolean formulas $\phi_1, \phi_2, \dots, \phi_m$, each with at most c variables.
- Question:* Find a truth assignment that satisfies the maximum number of the formulas.

We first show that the corresponding parameterized problem $(\text{MAX } c\text{-SAT})_=$ is fixed-parameter tractable for every constant c . By Papadimitriou and Yannakakis [366], for any input instance $x = \{\phi_1, \dots, \phi_m\}$ of the problem *MAX c-SAT*, the optimal value of x is at least $m/2^c$. Moreover, since each formula in x has at most c variables, the total number of variables appearing in x is bounded by cm . Therefore, the following algorithm demonstrates the fixed-parameter tractability of the parameterized problem $(\text{MAX } c\text{-SAT})_=$: Given a pair $\langle x, k \rangle$, where $x = \{\phi_1, \dots, \phi_m\}$ is an instance of the problem *MAX c-SAT* and k is an integer, we first compare the values k and $m/2^c$. If $k < m/2^c$, then $\langle x, k \rangle$ is obviously a no-instance for the parameterized problem $(\text{MAX } c\text{-SAT})_=$. If $k \geq m/2^c$, then we check all assignments to the variables in x to verify that whether the maximum number of formulas in x that can be satisfied by a single assignment is k . Since the total number of variables in x is bounded by cm , there are at most 2^{cm} different assignments to the variables in x . Therefore, the above algorithm runs in time $O(2^{cm})$, which is bounded by $O(2^{O(k)}m)$ since $m = O(k)$. This shows that the parameterized problem $(\text{MAX } c\text{-SAT})_=$ is fixed-parameter tractable.

According to Papadimitriou and Yannakakis [366], every *MAXSNP* problem Q can be reduced to the problem *MAX c-SAT* for some constant c in the following sense: There is a polynomial-time algorithm that given an input instance $x \in I_Q$ of Q , it constructs an input instance x' of *MAX c-SAT* such that the optimal value

¹The appropriate reduction here is called an *L*-reduction. See Exercise 4.

for the instance x of the problem Q is equal to the optimal value for the instance x' of the problem $MAX\ c\text{-}SAT$. Therefore, the membership of the input instance $\langle x, k \rangle$ for the parameterized problem $Q_{=}$ is identical to the membership of the input instance $\langle x', k \rangle$ for the parameterized problem $(MAX\ c\text{-}SAT)_{=}$. Since x' can be constructed from x in polynomial time and the parameterized problem $(MAX\ c\text{-}SAT)_{=}$ is fixed-parameter tractable, we conclude that the optimization problem Q in $MAXSNP$ is also fixed-parameter tractable. Since Q is an arbitrary problem in the class $MAXSNP$, the theorem follows.

Liming Cai and Chen also showed that an important class of minimization problems introduced by Kolaitis and Thakur [303] are fixed-parameter tractable.

Definition 4.10 ($MIN F^+ \Pi_1(h)$) Define $MIN F^+ \Pi_1(h)$, $h \geq 2$, to be the class of all minimization problems Q whose optimum can be expressed as

$$opt_Q(X) = \min_S \{ |S| : \forall v \psi(v, S, X) \},$$

where $S = (U, P_0)$ and P_0 is a predicate of arity 1 over the finite universe U , $|S|$ denotes the weight of the predicate P_0 , i.e., the number of elements of the universe U on which the predicate P_0 has truth value, and $\psi(v, S, X)$ is a quantifier-free CNF formula in which all occurrences of P_0 are positive, and P_0 occurs at most h times in each clause. Let $MIN F^+ \Pi_1 = \bigcup_{h \geq 2} MIN F^+ \Pi_1(h)$.

The class $MIN F^+ \Pi_1$ contains a number of well-studied minimization problems, including **VERTEX COVER** and a large number of *vertex deletion* and *edge deletion* graph problems [303]. It is known that all minimization problems in $MIN F^+ \Pi_1$ are polynomial-time approximable to a constant ratio [303]. Theorem 4.3 is not applicable to the class $MIN F^+ \Pi_1$ since the problem **VERTEX COVER** is $MAXSNP$ -hard [366], so it does not have a fully polynomial-time approximation scheme unless $P = NP$.

Theorem 4.11 (Liming Cai and Chen [108]) *All minimization problems in the class $MIN F^+ \Pi_1$ are fixed-parameter tractable.*

Proof. Let Q be a minimization problem in the class $MIN F^+ \Pi_1(h)$, where $h \geq 2$ is a fixed integer. Then the optimum $opt_Q(X)$ for any input instance X of Q can be expressed as

$$opt_Q(X) = \min_S \{ |S| : \forall v \psi(v, S, X) \},$$

where $S = (U, P_0)$, and P_0 is a predicate of arity 1 over the finite universe U , $v = (v_1, \dots, v_d)$ is a vector over U of arity d for some fixed d , $\psi(v, S, X)$ is a quantifier-free CNF formula in which all occurrences of P_0 are positive, and P_0 occurs at most h times in each clause.

We show below that the parameterized problem $Q_{\leq}$ is fixed-parameter tractable. Fix an input instance X of Q . Consider the formula

$$F(X, S) = \bigwedge_{v \in U^d} \psi(v, S, X).$$

Note that the formula $F(X, S)$ is a quantifier-free CNF formula in which the only unknowns are $P_0(u)$, where $u \in U$. Moreover, according to the definition of ψ , each clause of $F(X, S)$ contains at most h unknowns $P_0(u)$, $u \in U$ and all of them occur positively. Since the number of elements of the universe U is bounded by the length $|X|$ and d is a fixed integer, the formula $F(X, S)$ can be constructed from the input instance X in polynomial time.

Now, given an instance $\langle X, k \rangle$ of the parameterized problem $Q_{\leq}$, we first construct the formula $F(X, S) = \{C_1, C_2, \dots, C_s\}$, then execute the algorithm below.

Algorithm 4.12 (WEIGHTEDCNF(F, k))

INPUT: A CNF formula $F = \{C_1, \dots, C_s\}$ and an integer k , where each C_i is a clause in which there are at most h unknowns, all of the form $P_0(u)$, $u \in U$, and appearing positively.

QUESTION: Is there a finite structure $S = (U, P_0)$, $k > |S|$, and S satisfies F ?

- IF $k \leq 0$, then REJECT.
- IF $F = \phi$, then ACCEPT.
- Suppose that $C_1 = \{P_0(u_1), P_0(u_2), \dots, P_0(u_c)\}$, where $u_j \in U$ and $c \leq h$.
- FOR $j = 1$ to c Do
 - Make $P_0(u_j) = 1$;
 - Let F' be the CNF formula obtained from the formula F by deleting all clauses that contain the occurrence $P_0(u_j)$;
 - Recursively call WEIGHTEDCNF($F', k - 1$).

It is easy to verify the correctness of the algorithm WEIGHTEDCNF(F, k). If the algorithm WEIGHTEDCNF(F, k) accepts, then there is a finite structure $S_0 = (U, P_0)$, $|S_0| < k$, and $F(X, S_0) = \forall v \psi(v, S_0, X) = 1$. Therefore,

$$opt_Q(X) = \min_S \{ |S| : \forall v \psi(v, S, X) \} \leq |S_0| < k.$$

Thus, $\langle X, k \rangle$ is a no instance of the parameterized problem $Q_{\leq}$. Otherwise, we should have

$$opt_Q(X) = \min_S \{ |S| : \forall v \psi(v, S, X) \} = \min_S \{ |S| : F(X, S) \} \geq k.$$

Thus, $\langle X, k \rangle$ is a yes-instance of the parameterized problem $Q_{\leq}$. Therefore, the problem $Q_{\leq}$ can be solved through the algorithm WEIGHTEDCNF.

We claim that the running time of the algorithm WEIGHTEDCNF(F, k) is bounded by $O(h^k |F|)$. This is certainly true when $k \leq 1$. If $k \geq 2$, then the loop body of Step 4 is executed at most h times, each time involves a recursive call WEIGHTEDCNF($F', k - 1$), where $|F'| \leq |F|$. By the inductive hypothesis, the running time of each call of WEIGHTEDCNF($F', k - 1$) is bounded by $O(h^{k-1} |F|)$.

Note that the length of the formula F is bounded by a polynomial of the length of the input X which is independent of the parameter k , and h is a constant. Consequently, the parameterized problem $Q_{\leq}$ is fixed-parameter tractable.

Historical Remarks

The results of this section are taken from Liming Cai [104], and Liming Cai and Chen [107, 108]. Fagin's Theorem was proven in Fagin [201]. (See Fagin [202] for more of the history of the area.) There is a great deal of material now available on finite model theory and its relation to complexity classes. We refer the reader to Ebbinghaus and Flum [190] and Immermann [276]. Downey et al. [175] have material relating descriptive complexity to the parametric complexity classes of the sections to follow in Part II. The material on L -reductions is drawn from Papadimitriou and Yannakakis [366] where the reductions in the Exercises can be found.

Exercises 4.3

4.3.1. Consider the problem **BIN PACKING** below.

BIN PACKING

Input: $U = \{u_1, \dots, u_n\}$ and a size function $s(u_i) : U \mapsto \mathbb{Q}$ with $0 \leq s(u_i) \leq 1$.

Question: Find a partition of U into disjoint subsets $U_1, \dots, U_k$ such that for each j , $\sum_{u_i \in U_j} s(u_i) \leq 1$, and k is least.

Prove that the “first fit” algorithm, which begins with all the U_i being empty and then for $i \leq n$ in order, puts u_i into the first acceptable U_j , is a polynomial-time approximation algorithm with performance ratio $\leq \frac{1}{2}$. Actually, Johnson et al. [283] proved that the performance of this algorithm is always at least as good as $\frac{17}{10} \text{opt}(I) + 2$.

4.3.2. Prove Fagin's Theorem in detail.

4.3.3. (Stockmeyer) Prove that “SO=PH”; that is, prove that $\cup_n \Sigma_n^1$ coincides with the polynomial-time hierarchy.

4.3.4. (Papadimitriou and Yannakakis [366]) We say that for two optimization problems Π and Π' that Π L -reduces to Π' iff there exist two polynomial algorithms f and g and positive constants α and β such that for each instance I of Π ,

- (i) $f(I)$ is an instance of Π' with $\text{opt}(f(I)) \leq \alpha \text{opt}(I)$
- (ii) Given any solution of $I' = f(I)$ with cost c' , g produces a solution to I with cost c such that $|c - \text{opt}(I)| \leq \beta |c' - \text{opt}(I')|$.
- (a) Prove that L -reductions compose.
- (b) Prove that if Π L -reduces to Π' and are both maximization problems, then if there is a polynomial-time approximation algorithm for Π' with worst-case error ϵ' , then there is a polynomial-time approximation algorithm for Π with worst-case error $\alpha\beta\epsilon'$.

4.3.5. (Papadimitriou and Yannakakis [366]) Prove that the following are complete for *MAXSNP* under *L*-reductions. (See Exercise 4.)

MAX 3SAT

Input: A boolean formula X in 3CNF form.
Problem: Find an assignment satisfying the most number of clauses.

MAX 3SAT-B

Input: A boolean formula X in 3CNF form where the number of occurrences of variables are bounded by a constant B .
Problem: Find an assignment satisfying the most number of clauses.

INDEPENDENT SET-B

Input: A graph with the maximum degree $\leq B$.
Question: Find the largest independent set.

DOMINATING SET-B

Input: A graph with the maximum degree $\leq B$.
Question: Find the smallest dominating set.

MAX CUT

Input: A graph G .
Question: Find a partition of the nodes of G into sets S and $\bar{S}$ such that the number of edges from S to $\bar{S}$ is greatest.

MAX k -COLORABLE SUBGRAPH

Input: A graph G .
Question: Find the largest k -colorable subgraph. (Here, k is a fixed parameter.)

4.3.6. (Papadimitriou and Yannakakis [366]) Prove that every problem in *MAXSNP* can be approximated to within some constant ratio.

(Hint: It is enough to consider the case of **MAX 3SAT** by the above. Approximate to within $\frac{1}{2}$ by repeatedly choosing the truth value that satisfies the most as yet unsatisfied clauses.)

5

The Advice View Revisited and *LOGSPACE*

In this brief chapter, we will further examine the *advice view* of parameterized tractability introduced in Chapter 2, Section 2.2. The material of the present section is a little technical and might well be omitted on the first reading of the book. Such omission will have no effect on the readability or coherence of the rest of the book.

The reader should recall that, in Section 2.2, we offered the advice view of being *FPT* as an analog of the Karp-Lipton class *P/poly*. As we recall from Section 2.2, the classical advice class *P/poly* is concerned with languages that can be recognized in polynomial time when we are provided, for input of size n , a word of advice w_n , where the length of w_n is bounded by some polynomial $q(n)$. As we noted earlier, the point is that the word of advice depends *only* upon the *size* $|x|$ of the input, not what x actually is, and hence a single w_n works for all the 2^n strings of length n . Liming Cai, Chen, Downey, and Fellows [110] introduced the view that nonuniform *FPT* is exactly the set of problems solvable in polynomial time using a finite piece of advice $f(k)$ for each slice k . The theorem was Theorem 2.5 which we restate below.

Theorem 5.1 (Cai, Chen, Downey, and Fellows [110])

- (i) $L \in \text{nonuniform FPT}$ iff there is an polynomial-time oracle Turing machine Φ and a function $f : \mathbb{N} \rightarrow \Sigma^*$ such that for all x, k

$$\langle x, k \rangle \in L \text{ iff } \Phi^{f(k)}(\langle x, k \rangle) = 1.$$

- (ii) $L \in \text{strongly uniform FPT}$ iff there is a polynomial-time oracle Turing machine Φ and a recursive function $f : \mathbb{N} \rightarrow \Sigma^*$ such that for all x, k

$$\langle x, k \rangle \in L \text{ iff } \Phi^{f(k)}(\langle x, k \rangle) = 1.$$

There is nothing special about polynomial time here and we can easily extend the notion of advice to general complexity classes. In discussing space-bounded complexity classes, we employ the standard model of a Turing machine with a read-only two-way input tape, a write-only output tape, and a work tape (where the space complexity of a computation is measured).

Definition 5.2 (Cai, Chen, Downey, and Fellows [110])

- (i) Let $\mathcal{C}$ be a class of functions representing time (space) resource bounds. We say that a (parameterized) language L is $\mathcal{C} + \text{advice}$ if there is a function $g \in \mathcal{C}$, an oracle Turing machine Γ , and a function $w : \omega \rightarrow \Sigma^*$ (the *advice* function) such that

$$\langle x, k \rangle \in L \text{ iff } \Gamma^{w(k)}(\langle x, k \rangle) \text{ accepts,}$$

and furthermore for all k and x , the running time (space) of $\Gamma^{w(k)}(\langle x, k \rangle)$ is $\leq g(|x|)$.

- (ii) If w is recursive, then we say that L is in *uniform* $\mathcal{C} + \text{advice}$.

As with the proof of Theorem 5.1, the finite advice $w(k)$ is often a table of hard instances, perhaps with advice to the name of some procedure. We can now economically rewrite the statement of Theorem 5.1 as

$$\begin{aligned} \text{nonuniform } FPT &= P + \text{advice,} \\ \text{strongly uniform } FPT &= \text{uniform } P + \text{advice.} \end{aligned}$$

Theorem 5.1 essentially says that, assuming that $P \neq NP$, for NP -complete problems whose parameterized versions are in FPT , hard instances can only be found when the size of the parameter is close to that of the object under consideration, and therefore being FPT yields real qualitative insight into the distribution of hard instances.

In the same spirit as Definition 5.2, we define the class of **SLICEWISE** $\mathcal{C}$ as those languages L accepted in time (space) $f(k)g(|x|)$. As with the other definitions, we append the adjective “uniform” if, additionally, f is recursive. Under this definition, FPT is **SLICEWISE** P . Thus,

$$P + \text{advice} = \text{SLICEWISE } P.$$

Theorem 5.1 depends little on the fact that we are using P , only certain closure properties. One can easily establish the following results.

Theorem 5.3 (Cai, Chen, Downey, and Fellows [110])

- (i) (uniform) **SLICEWISE** **POLYLOGSPACE**¹
 $=$ (uniform) **POLYLOGSPACE** + *advice*.

¹The reader is reminded that **POLYLOGSPACE** is the complexity class $\bigcup_{k=0}^{\infty} \mathbf{DSPACE}(\log^k(|x|))$.

(ii) (uniform) *SLICEWISE PSPACE* = (uniform) *PSPACE* + advice.

Proof. See Exercise 4

In the present section, we will look at natural subclasses of *FPT* where the collapses represented by Theorems 5.1 and 5.3 do not occur. A class of particular interest is $\mathcal{C} = \text{LOGSPACE}$. The reader should recall that *LOGSPACE* is the class of problems which are solvable in space logarithmic in the length of the input. Of course, by direct simulation, $\text{LOGSPACE} \subseteq P$ and it is believed that $\text{LOGSPACE} \subset P$. In the previous sections, we have looked at two algorithm design paradigms: the method of search trees and the method of reduction to a problem kernel. With care, each of these methods can often be used to demonstrate more than mere membership in *FPT*.

For instance, the method of reduction to a problem kernel applied to a language L works by taking an instance $I = \langle x, k \rangle$ to an equivalent “small” instance $I' = \langle x', k' \rangle$. This method immediately yields that $L \in P + \text{advice}$, with the advice w_k consisting of an exhaustive analysis of all instances of size at most $f(k)$. If the equivalent small instance $\langle x', k' \rangle$ can be constructed in small space, then we may be able to further show that L belongs to $\text{LOGSPACE} + \text{advice}$ or *SLICEWISE LOGSPACE*. [The latter may be the best we can do; for instance, for I and I' graphs, in the reduction of I to I' we often need I' to be described by something like $f(k)$ many vertices of I and thus may require $f(k) \log(|I|)$ space to describe I' .] A good example of this method comes from analyzing the proofs of the previous two sections that *VERTEX COVER* is in *FPT* (Theorems 3.1 and 3.10).

Theorem 5.4 (Cai, Chen, Downey, and Fellows [110]) *VERTEX COVER* is in uniform *LOGSPACE* + advice.

Proof. (1) First, we recall the proof of Buss from Section 3.2. This goes as follows: We first observe that any vertex of degree $> k$ must belong to any vertex cover of a given G . So, count the number of vertices in G of degree $> k$, until we stop with a value $p \leq k$ or we exceed k , in which case we declare that there is no k -element vertex cover. In the former case, let $m = k - p$. Now try to compute the number q of vertices that are not covered by a vertex of degree $> k$. We do this until either we find that there are $> m(k + 1)$ such vertices, in which case we reject, or we discover that there are $\leq m(k + 1)$ vertices corresponding to edges with no high degree vertex covering them. [The bound here is generated by the fact that a simple graph with an m -element vertex cover and all degrees bounded by k has no more than $m(k + 1)$ vertices.]

Now notice that the above only needs a few counters and can easily be done in uniform *LOGSPACE*.

Finally, if we are in the case that there are at most $m(k + 1)$ vertices not covered by high-degree vertices, we need to do a complete check. We need to do this in very small space. What we do is build an isomorphic copy of the induced subgraph H of G generated by the vertices of edges not covered by those of high degree. The idea is to use the natural “intrinsically definable” lexicographic order on the uncovered

vertices to induce the isomorphism. We will generate an incidence array of our copy H' of H . We do this by first locating the lexicographically least uncovered vertex. Then, we generate the first row of the $q \times q$ incidence array, look at the vertices of G in lexicographic order, and put an entry if the one presently being considered is uncovered. The entry is, of course, dependent upon whether the first uncovered vertex is connected to the current one. We then increment our counter to the next entry of the first row, until we fill in all q entries. We continue this process inductively, until we determine the whole $q \times q$ matrix. Now, in constant space we can check if H' has an m -element vertex cover. If not, then G does not have a k element one. If H' has such a vertex cover, then G has a k -element one generated by the p elements of degree $> k$ and the one induced by lifting the m element one in H' .

It is also possible to demonstrate the validity of Theorem 5.4 using a modification of the Downey-Fellows search tree proof of Section 3.1.

Proof. (2) Again, we use the intrinsic definability of the construction via lexicographic ordering. Recall that the proof of Theorem 3.1 went as follows. Build a tree of 2^k possible vertex covers by taking an edge $\langle v_{i_1}, w_{j_1} \rangle$ and beginning a tree of possibilities by placing v_{i_1} on one node and w_{j_1} on the other. Inductively, for a node σ of the bounded search tree, take an edge not covered by the vertices defined by $\sigma = v_{q_1} \wedge \dots \wedge v_{q_m}$ and use this to make the children of σ . The tree has depth k and then we check order if everyone is covered.

We claim that this process can actually be accomplished in small space. Consider the standard tree $T = 2^k$. We can regard a node η of T as “intrinsically” coding a potential vertex cover as follows. Let $\eta = i_1 \wedge \dots \wedge i_k$. Then, the vertex of G coded by i_1 is the lexicographically least vertex of the lexicographically least edge of G if $i_1 = 0$, and the other vertex of the lexicographically least edge of G if $i_1 = 1$. This description can be extended inductively, the children of the γ being associated with the lexicographically least edge not covered by the vertices intrinsically defined by γ . We can easily show by induction that there is a *LOGSPACE* procedure $\Phi(\gamma, n)$ which computes the vertex of G associated with the γ at level n , by cycling through candidates in lexicographic order. Now to see if all edges are covered by η , we can again cycle through all the edges of G and for each edge invoke $\Phi(\eta, j)$ sequentially for each $i \leq k$ until either the edge is found to be covered by a vertex associated with η or we reject the potential vertex cover coded by η .

There are many other applications of the technique above to problems of the earlier sections. We refer the reader to the exercises.

Historical Notes

All of the results in this section are taken from Liming Cai, Chen, Downey, and Fellows [110].

Exercises 5.0

- 5.0.1.** (Cai, Chen, Downey, and Fellows [110]) Prove that the following problem is in *(uniform) constant space + advice*.

FIXED SUBSET SUM (for positive integers)

Instance: A set $B = \{a_1, \dots, a_n\}$ with “sizes” $s(a_i) \in \mathbb{N}$.

Parameter: A positive integer k .

Question: Is there a subset B' of B with $\sum_{a_i \in B'} s(a_i) = k$?

(Hint: Reading the input, make a table of the sizes $s \leq k$ occurring in B together with their frequency $f(s)$ until $f(s) \times s$ exceeds k . Check all possibilities see if k is attainable.)

- 5.0.2.** (Cai, Chen, Downey, and Fellows [110]) Prove that the the following problem is in *LOGSPACE + advice*.

BOUNDED SUBSET SUM

Instance: A set $B = \{a_1, \dots, a_n\}$ with “sizes” $s(a_i) \in \mathbb{Z}$.

Parameter: A positive integer k .

Question: Is there a subset B' of B with $\sum_{a_i \in B'} s(a_i) = k$?

(Hint: Reduce from SUBSET SUM.)

- 5.0.3.** (Cai, Chen, Downey, and Fellows [110]) Prove the following. Let $\mathcal{C}$ be a complexity class and $g \in \mathcal{C}$.

- (i) Suppose that $g \geq \log |x|$ and h are functions, and Φ is a Turing machine with L a language so that for all x and k ,

$$\langle x, k \rangle \in L \text{ iff } \Phi \text{ accepts } \langle x, k \rangle,$$

and $\Phi(\langle x, k \rangle)$ runs in time (space) $g(|x|) + h(k)$. Then, L is in $\mathcal{C} + \text{advice}$. Moreover, if g and h are recursive, then L is in *uniform* $\mathcal{C} + \text{advice}$.

- (ii) Furthermore, the converse holds for space bounds, provided the witness function is recursive; that is, if $\mathcal{C}$ is a class of space bounds and L is in *uniform* $\mathcal{C} + \text{advice}$ with $w(k)$ computable from k , then there exist Φ , g , and h as in (i).

- 5.0.4.** (Cai, Chen, Downey, and Fellows [110]) Verify Theorem 5.3; that is, prove that

- (i) *(uniform) SLICEWISE POLYLOGSPACE = (uniform) POLYLOGSPACE + advice.*
- (ii) *(uniform) SLICEWISE PSPACE = (uniform) PSPACE + advice.*

- 5.0.5.** (Cai, Chen, Downey, and Fellows [110]) Prove that the following problems (from Section 3.1) are in *(uniform) LOGSPACE + advice*:

- (i) **RESTRICTED ALTERNATING HITTING SET.**

(ii) **WEIGHT $\leq k$ q -CNF SATISFIABILITY** for any fixed q .

(iii) **DOMINATING SET FOR PLANAR GRAPHS.**

(Hint: Use the method of Theorem 5.4.)

5.0.6. (Cai, Chen, Downey, and Fellows [110]) Consider also the following variation on the k -LEAF SPANNING TREE problem of Section 3.2:

k -LEAF SPANNING FOREST

Input: A graph G .

Parameter: A positive integer k .

Question: Does G have a k -leaf spanning forest? That is, does there exist a collection of trees $\{T_i : i \in I\}$ such that collectively the trees have at least k leaves and each T_i is a spanning tree for its component of G . Thus, if G is connected, then a spanning forest is simply a spanning tree.

Modify the proof of Theorem 3.13 to obtain the following:

(i) k -LEAF SPANNING TREE is in (uniform) $NLOGSPACE + advice$.

(ii) k -LEAF SPANNING FOREST is in (uniform) $LOGSPACE + advice$.

5.0.7. (Cai, Chen, Downey, and Fellows [110]) This exercise is more technical than the others and is solved by techniques from Chapter 19, where it is repeated as an exercise (Exercise 3). Readers unfamiliar with the techniques of Downey and Fellows [168] might find that it is best done after reading Chapter 19. Prove the following:

(uniform) $LOGSPACE + advice \neq$ (uniform) $SLICEWISE LOGSPACE$.

6

Methods via Automata and Bounded Treewidth

6.1 Classical Automata Theory

6.1.1 Deterministic Finite Automata

Many of the methods we use on graphs are analogs of methods used for languages accepted by finite automata. Because of this, in the next few sections we will give a self-contained account of the aspects of classical formal language theory which we need. We begin with finite automata.

Definition 6.1 (Finite Automaton) A *deterministic finite automaton* is a quintuple $M = \langle K, \Sigma, \delta, S, F \rangle$, where

- K is a finite set called the set of states,
- $S \subseteq K$ is called the set of *initial* states,
- $F \subseteq K$ is called the set of *accepting* states,
- Σ is the finite underlying set, and
- $\delta : K \times \Sigma \mapsto K$ is a (partial) function called the *transition* function.

We will usually denote states by $\{q_i : i \in G\}$ for some set G and have $S = \{q_0\}$. As the reader will recall, we think of automata as physical machines that act to accept or reject strings from Σ^* . We consider the machine M as starting on the leftmost symbol of a string $\sigma \in \Sigma^*$, in state q_0 . The transition function δ induces a rule “If M is in state q_i reading $a \in \Sigma$, move one symbol right and change the internal state of M to $\delta(q_i, a)$.” For each $\gamma \in \Sigma^*$, we may write this action as

$$\langle q_i, a\gamma \rangle \vdash \langle \delta(q_i, a), \gamma \rangle.$$

Here, the symbol $\vdash$ is read as “yields in one step.”

We say that M *accepts* a string σ if, after starting M on input σ in state q_0 and then applying $\vdash |\sigma|$ times, M is in one of the states in F . (More formally, one can define M to accept σ if, for some $f \in F$,

$$\langle q_0, \sigma \rangle \vdash^* \langle f, \lambda \rangle,$$

where λ denotes the empty string, and $\vdash^*$ denotes the transitive closure of $\vdash$.) We let

$$L(M) = \{\sigma \in \Sigma^* : M \text{ accepts } \sigma\}.$$

Example 6.2 Let $K = \{q_0, q_1\}$, $\Sigma = \{0, 1\}$, $S = \{q_0\}$, and $F = \{q_0\}$.

state(q)	symbol(a)	$\delta(q, a)$
q_0	0	q_0
q_0	1	q_1
q_1	1	q_0
q_1	0	q_1

$\langle q_0, 00110 \rangle \vdash \langle q_0, 0110 \rangle$
 $\vdash \langle q_0, 110 \rangle$
 $\vdash \langle q_1, 10 \rangle$
 $\vdash \langle q_0, 0 \rangle$
 $\vdash \langle q_0, \lambda \rangle.$

Now as $q_0 \in F$, M accepts 00110 and, hence, $00110 \in L(M)$. On the other hand, one can easily see that $\langle q_0, 0010 \rangle \vdash^* \langle q_1, \lambda \rangle$ and, since $q_1 \notin F$, $0010 \notin L(M)$.

The traditional way of representing automata is via *transition (or state) diagrams*. Here, there is one vertex for each of the states of M and arrows from one state to another labeled by the possible transitions given by δ . Accept states are usually represented by a “double circle” at the node. For instance, the machine M would be represented by the diagram in Figure 6.1. In this simple machine, q_0 has a double circle around it since it is an accept state, whereas q_1 has only a single circle around it since it is not an accept state.

The crucial fact we will need in our investigations is that, for a fixed M , one can quickly decide if $\sigma \in M$; that is, we note the following trivial observation.

DETERMINISTIC FINITE AUTOMATA ACCEPTANCE

Input: A string $\sigma \in \Sigma^*$.
Parameter: M , a deterministic finite automata over Σ^* .
Question: Is $\sigma \in L(M)$?

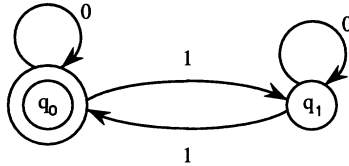


FIGURE 6.1. Transition Diagram for Example 6.2

Observation 6.3 DETERMINISTIC FINITE AUTOMATA ACCEPTANCE is $O(|\sigma|)$, and hence is strongly uniformly FPT.

The point of the story is that finite automata provide a canonical way of demonstrating that a given problem is *FPT*. Although not many graph theoretical problems are accepted by standard automata like those above, we *can* use the classical theory as a good basis for our approach to modified automata that are more attuned to graph theoretical properties. Before we do so, we will need to develop a bit more of the classical theory. The following exercises should provide the reader unfamiliar with automata with a little warm-up to the next sections.

Historical Notes

The study of what we now call finite automata essentially began with McCulloch and Pitts [346]. Kleene [298] introduced the model we now use. Many others had similar ideas and introduced various models. For instance, the reader is referred to Huffman [273], Moore [355], and Mealy [349]. Hopcroft and Ullman [270] discuss the development of finite automata in more detail.

Exercises 6.1

6.1.1. Prove that for the machine M of Example 6.2,

$$L(M) = \{\sigma : \sigma \text{ contains an even number of ones}\}.$$

6.1.2. Construct automata to accept the following languages $L_i \subseteq \{0, 1\}^*$:

1. $L_1 = \{\sigma : \sigma \text{ contains at most 5 ones}\}.$
2. $L_2 = \{\sigma : \sigma \text{ contains no 1 followed by a 0}\}.$
3. $L_3 = \{\sigma : \text{the number of ones in } \sigma \text{ is a multiple of 3}\}.$

6.1.3. Prove that if L_1 and L_2 are languages accepted by deterministic finite automata, so is $L_1 \cup L_2$.

6.1.2 Nondeterministic Finite Automata

The basic idea behind nondeterminism is that from any position, there are a number of possible computation paths. For automata nondeterminism is manifested as the generalization the transition relation δ from a function to a multifunction. Thus, δ now becomes a relation so that from a given (state, symbol) there may be several possibilities for $\delta(\text{state}, \text{symbol})$. Formally, we have the following definition.

Definition 6.4 (Nondeterministic Finite Automaton) A *nondeterministic finite automaton* is a quintuple $M = \langle K, \Sigma, \Delta, S, F \rangle$, where K , S , F , and Σ are the same as for a deterministic finite automaton, but $\Delta \subseteq K \times \Sigma \times K$ is a relation called the *transition* relation.

Now, we can interpret the action of a machine M being in state q_i reading a symbol a , as “being able to move to any one of the states q_k with $\langle q_i, a, q_k \rangle \in \Delta$.” Abusing notation slightly, we write

$$\langle q_i, a\gamma \rangle \vdash \langle q_k, \gamma \rangle,$$

but now we will interpret $\vdash$ to mean “can *possibly* yield.” Again, we will let $\vdash^*$ denote the reflexive transitive closure of $\vdash$ and declare that

$$\sigma \in L(M) \text{ iff } \langle q_0, \sigma \rangle \vdash^* \langle f, \lambda \rangle,$$

for some $f \in F$. The interpretation of the definition of $\vdash^*$ is that we will accept σ provided that there is some way to finish in an accepting state; that is, *we will accept σ provided that there is some computation path of M accepting σ* . Of course, there may be many computation paths on input σ . Perhaps only one leads to acceptance. Nevertheless, we would accept σ .

Example 6.5 We specify M by the following sets:

$K = \{q_0, q_1, q_2, q_3, q_4, q_5\}$, $\Sigma = \{a, b\}$, $S = \{q_0\}$, $F = \{q_4\}$, and

$\Delta = \{\langle q_0, a, q_0 \rangle, \langle q_0, b, q_0 \rangle, \langle q_0, b, q_5 \rangle, \langle q_5, b, q_0 \rangle, \langle q_5, a, q_1 \rangle, \langle q_1, b, q_2 \rangle, \langle q_1, a, q_3 \rangle, \langle q_1, b, q_4 \rangle, \langle q_2, a, q_4 \rangle, \langle q_2, b, q_4 \rangle, \langle q_3, b, q_4 \rangle, \langle q_3, a, q_3 \rangle, \langle q_4, a, q_4 \rangle, \langle q_4, b, q_4 \rangle\}$.

Note that upon reading, say, bab , M could finish in states q_0, q_2, q_4 , or q_5 . The reader should verify that

$$L(M) = \{\sigma \in \{a, b\}^* : bab \text{ or } ba(a \dots a)ab \text{ is a substring of } \sigma\}.$$

At first glance, it would appear that nondeterministic automata are much more powerful than deterministic ones. This is not the case, as we see by the classic theorem of Rabin and Scott below. We say that two automata M_1 and M_2 are equivalent if $L(M_1) = L(M_2)$.

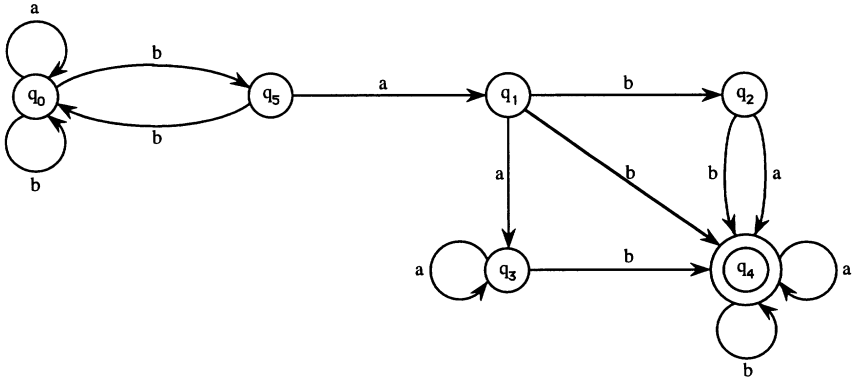


FIGURE 6.2. Transition Diagram for Example 6.5

Theorem 6.6 (Rabin and Scott's Theorem [380]) *Every nondeterministic finite automata is equivalent to a deterministic one. Furthermore, if M is a (nondeterministic) machine with n states, then we may compute in $O(2^n)$ steps a deterministic machine M' with at most 2^n states. Here, the O only depends on $|\Sigma|$.*

Proof. The construction to follow is often referred to as the *subset construction*. Let $M = \langle K, \Sigma, S, F, \Delta \rangle$ be the nondeterministic automaton. The idea is quite simple. We will consider the states of the new machine we create from M as corresponding to *sets* of states from M . We will assume that $S = \{q_0\}$. Now, on a given symbol $a \in \Sigma^*$, M can go to any of the states q_k with $\langle q_0, a, q_k \rangle \in \Delta$. Let $\{q_{k_1}, \dots, q_{k_p}\}$ list the states with $\langle q_0, a, q_{k_j} \rangle \in \Delta$. The idea is that we will create a state $Q_1 = \{q_{k_1}, \dots, q_{k_p}\}$, consisting of the *set of states of M reachable from q_0 on input a* . Consider Q_1 . For each symbol $b \in \Sigma^*$, and for each $q_{k_j} \in Q_1$, compute the set D_{k_j} of states reachable from q_{k_j} via b , and we let Q_2 denote $\cup_{q_{k_j} \in Q_1} D_{k_j}$. Formally,

$$Q_0 = S = \{q_0\},$$

$$Q_1 = \{q \in K : \langle q_0, a \rangle \vdash \langle q, \lambda \rangle\},$$

and

$$Q_2 = \cup_{q \in Q_1} \{r \in K : \langle q, b \rangle \vdash \langle r, \lambda \rangle\}.$$

The idea continues inductively as we run through all the members of Σ and longer and longer input strings. We continue until no new states are generated. Notice that path lengths are bounded by $n = |K|$, and that the process will halt after $O(2^n)$ steps. The accept states of the machine so generated consists of those subsets of K that we construct and contain at least one accept state of M . It is routine to prove that this construction works. (See Exercise 3.)

The algorithm implicit in the proof above is often called “Thompson’s construction” [446]. Notice that we may not necessarily construct all the 2^n subsets of K as states. The point here is that many classical proofs of Rabin and Scott’s theorem simply say to generate all the 2^n subsets of K . Then, let the initial state of M' be the set of start states of M , the accept states of M' are the subsets of K containing at least one accept state of M , and, if $F \subseteq K$ and $a \in \Sigma$, define

$$\delta(F, a) = \cup_{q \in F} \{r \in K : \langle q, a \rangle \vdash \langle r, \lambda \rangle\}.$$

The difference is that the latter proof often constructs many *useless* states since many of the $F \subseteq K$ simply *cannot* be accessed upon *any* input string. But no matter what, if we use the latter proof, we will indeed take $O(2^n)$ steps.

Example 6.7 We apply Thompson’s [446] construction to the automaton from Example 6.5.

The initial state of Example 6.5 is $Q_0 = \{q_0\}$. We obtain the transition diagram of Figure 6.3.

Notice that the deterministic automaton generated by Thompson’s construction has only 8 states, whereas the worst possible case would be $2^5 = 32$ states. We re-

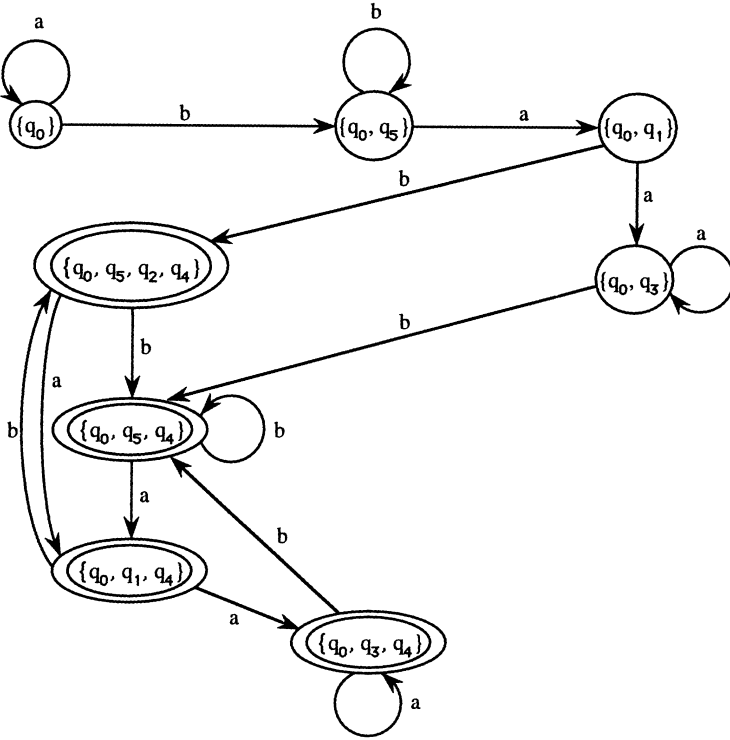


FIGURE 6.3. The Deterministic Automaton for Example 6.5

mark that it is possible for Thompson's construction and, indeed, *any* construction to generate *necessarily* $\Omega(2^n)$ states. This is because there exist examples of n state nondeterministic automata where any deterministic automata accepting the same strings necessarily need $\Omega(2^n)$ states. We give one example of this phenomenon in the next section (Example 6.19).

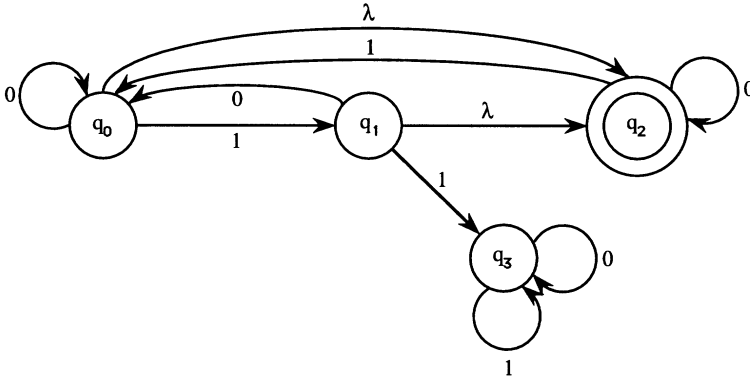
As with deterministic automata, we can extract the following algorithmic problem.

NONDETERMINISTIC FINITE AUTOMATA ACCEPTANCE

Input: A string $\sigma \in \Sigma^*$.
Parameter: A nondeterministic finite automaton M .
Question: Does M accept σ ?

Theorem 6.6 has the following important algorithmic consequence.

Corollary 6.8 NONDETERMINISTIC FINITE AUTOMATA ACCEPTANCE is $O(n)$ and strongly uniformly FPT.

FIGURE 6.4. A Nondeterministic Automaton with λ Moves

It is very convenient for us to extend the model of a nondeterministic automaton to *nondeterministic finite automata with λ moves*. Here, one allows “spontaneous” movement from one state to another with no head movement.

Example 6.9 Consider the automaton M given by Figure 6.4.

For this automaton, on input 110 it is possible for the machine to follow a computation path to *any* of the four states except q_1 . For instance, to get to state q_0 , M could read 1 in state q_0 , move to state q_1 , spontaneously move into state q_2 via the empty transition from q_1 to q_2 , travel the 1 transition from q_2 back to q_0 , and then finally travel the 0 transition from q_0 to q_0 . We invite the reader to check that all the other states are accessible on input 110.

Again, it is easy to show that there is no real increase in computational power for the new class of automata.

Theorem 6.10 (McNaughton and Yamada [348], Rabin and Scott [380]) *Every nondeterministic finite automaton with λ moves is equivalent to one without λ moves. Moreover, there is a construction, which when applied to M , a nondeterministic finite automaton with λ moves and n states, yields an equivalent nondeterministic automaton M' with $O(n)$ states in $O(n)$ steps.*

Proof. Let M be a given nondeterministic finite automaton with λ moves. We construct an equivalent M' without λ moves. For a state q of M , define the equivalence class of q via

$$E(q) = \{p \in K(M) : \forall w \in \Sigma^* (\langle q, w \rangle \vdash^* \langle p, w \rangle)\};$$

that is, $E(q)$ denotes the collection of states accessible from q with no head movement. In Example 6.9, $E(q_0) = \{q_0, q_2\}$. As with the proof of Theorem 6.6, for M' , we replace states by sets of states of M . This time the set of states are the equivalence classes $E(q)$ for $q \in M$. Of course, the accept states are just those $E(q)$ containing accept states of M , and the new transition on input a takes $E(q)$

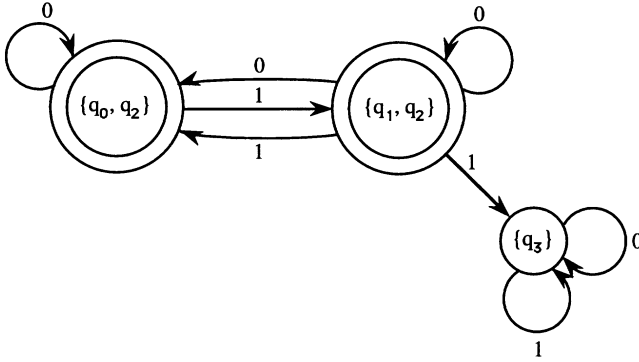


FIGURE 6.5. The Automaton Corresponding to Example 6.9

to $E(r)$, provided that there is some q_i in $E(q)$ and some q_j in $E(r)$ with Δ taking q_i to q_j on input a . It is easy to see that this construction works (Exercise 4).

When applied to the automaton of Example 6.9, the method used in the proof of Theorem 6.10 yields the automaton in Figure 6.5.

Definition 6.11 If L is accepted by a finite automaton (over Σ^*), we shall say that L is *finite state* (over Σ^*).

We can use these results to establish a number of closure properties for finite-state languages.

Theorem 6.12 Let $\mathcal{L}$ denote the collection of languages accepted by finite automata. Then $\mathcal{L}$ is closed under the following:

- (i) union
- (ii) intersection
- (iii) star (i.e., if $L \in \mathcal{L}$, then so is L^* , the collection of strings obtainable from L)
- (iv) complementation
- (v) concatenation (i.e., if $L_1, L_2 \in \mathcal{L}$, then so is $L_1 L_2 = \{xy : x \in L_1 \wedge y \in L_2\}$).

Proof. Let $M_i = \langle K_i, \Sigma_i, S_i, \Delta_i, F_i \rangle$ for $i = 1, 2$, be automata over Σ . By renaming, we can suppose that $K_1 \cap K_2 = \emptyset$.

For (i), we simply add a new start state S' with a λ move to each of the start states of M_1 and M_2 . The accept states for $M_1 \cup M_2$ would be $F_1 \cup F_2$.

For (iv), we need to prove that $\overline{L(M_1)}$ is finite state. This is easy. Let $M' = \langle K_1, \Sigma, S_1, \delta_1, K_1 - F_1 \rangle$. Clearly, $\overline{L(M_1)} = L(M')$.

For (ii), we can use the fact that $L(M_1) \cap L(M_2) = \overline{\overline{L(M_1)} \cup \overline{L(M_2)}}$.

We leave (iii) and (v) for the reader (Exercise 6).

Historical Notes

Nondeterministic Finite Automata were introduced by Rabin and Scott [380]. Rabin and Scott established the equivalence of nondeterministic finite automata with deterministic finite automata. McNaughton and Yamada consider automata with λ moves. Thompson's construction is taken from Thompson [446].

Exercises 6.1

- 6.1.1. Compute a deterministic finite automaton equivalent to the automaton of Example 6.9.
- 6.1.2. Describe the language L accepted by the automaton of Example 6.2.
- 6.1.3. Carefully complete the details of the proof of Theorem 6.6 by proving that if $\langle q_0, \sigma \rangle \vdash_M^* \langle r, \lambda \rangle$, then $\langle Q_0, \sigma \rangle \vdash_{M'}^* \langle R, \lambda \rangle$ for some state R of M' containing r .
- 6.1.4. Carefully formalize the proof of Theorem 6.10.
- 6.1.5. Prove that for any deterministic finite automaton $M = \langle K, \Sigma, S, F, \delta \rangle$ for all $\sigma, \tau \in \Sigma^*$ and $q \in K$,

$$\delta(q, \tau\sigma) = \delta(\delta(q, \tau), \sigma).$$

- 6.1.6. Prove (iii) and (v) of Theorem 6.12; that is, prove that the collection of finite-state languages over Σ is closed under star and concatenation.
- 6.1.7. It is possible to generalize the notion of a nondeterministic automaton as follows. Now M is a quintuple $\langle K, \Sigma, \delta, S, F \rangle$ with all as before save that δ is a multifunction from a subset of $K \times \Sigma^* \mapsto K$ giving a rule:

$$\langle q_i, \gamma\sigma \rangle \vdash \langle \delta(q_i, \gamma), \sigma \rangle.$$

The difference is that in the standard definition of an automaton, we can only have $a \in \Sigma$, whereas now we can have $\gamma \in \Sigma^*$. (λ moves can be thought of as a special case of this phenomenon with $\gamma = \lambda$.) We will call such automata nondeterministic automata with *string* moves. Prove that any nondeterministic finite automaton with string moves, M is equivalent to a standard nondeterministic finite automaton, M' . What is the relationship between the number of states of M and M' ?

- 6.1.8. (i) Suppose that L , L_1 , and L_2 are finite state. Prove that the following are decidable if we are given finite automata M , M_1 , and M_2 with $L = L(M)$, $L_1 = L(M_1)$, and $L_2 = L(M_2)$.
 - (a) Is $L = \emptyset$?
 - (b) Is $L = \Sigma^*$?
 - (c) Is $L_1 \subseteq L_2$?
 - (d) Is $L_1 = L_2$?
- (ii) Prove that (a) and (b) are solvable in $O(2^n)$ for a fixed Σ .

6.1.3 Regular Languages

Finite automata provide a *semantical* approach to formal language theory, with implicit algorithmic consequences. There is another well-known *syntactical* approach to finite-state languages. It has been extensively studied due to its importance in practical applications such as the **GREP** family of commands in **UNIX** (and surely because of the beauty of the resulting theory). We refer, of course, to the class of regular languages. The reader should recall that a regular language is one that has a representation via a finite string of symbols taken from the members of Σ , together with $\cup$ (union),¹ concatenation (which we represent by juxtaposition for regular expressions and strings), $*$ (Kleene star), and $\emptyset$. Formally, we have the following definitions.

Definition 6.13 (Regular Expression) A *regular expression* over Σ is an expression in the alphabet $\Sigma, \cup, (,),$ and $\emptyset$ defined by induction on logical complexity as follows:

- (i) $\emptyset$ and each member of Σ is a regular expression.
- (ii) If α and β are regular expressions, so are $(\alpha\beta)$ and $(\alpha \cup \beta)$.
- (iii) If α is a regular expression, so is $(\alpha)^*$.
- (iv) Nothing else.

Naturally, we drop the brackets where the meaning is clear.

Definition 6.14 (Regular Language) We associate a language $L(\alpha)$ with a regular expression α as follows.

- (i) If $\alpha = \emptyset$, then $L(\alpha) = \emptyset$.
- (ii) If $\alpha = a$ with $a \in \Sigma$, then $L(\alpha) = \{a\}$.
- (iii) If $\alpha = \gamma\beta$, then $L(\alpha) = L(\gamma)L(\beta)$ [=def $\{xy : x \in L(\gamma) \text{ and } y \in L(\beta)\}$].
- (iv) If $\alpha = \gamma \cup \beta$, then $L(\alpha) = L(\gamma) \cup L(\beta)$.
- (v) If $\alpha = \beta^*$, then $L(\alpha) = (L(\beta))^*$ [=def the set of all strings obtainable from $L(\beta)$.]

We call a language L *regular* if $L = L(\alpha)$ for some regular expression α .

Example 6.15 Let $L = \{0, 1\}$.

$$\begin{aligned}
 L(0(0 \cup 1)^*) &= L(0)L((0 \cup 1)^*) \text{ by (iii)} \\
 &= \{0\}(L(0 \cup 1))^* \text{ by (ii) and (v)} \\
 &= \{0\}(L(0) \cup L(1))^* \text{ by (iv)} \\
 &= \{0\}(\{0\} \cup \{1\})^* \text{ by (ii)} \\
 &= \{0\}\{0, 1\}^* \\
 &= \{x \in \{0, 1\}^* : x \text{ starts with a } 0\}.
 \end{aligned}$$

¹Often union is represented by $|$.

The relationship with the work of the previous sections is described by the following classic theorem.

Theorem 6.16 (Kleene [298]) *For a language L , L is finite state iff L is regular. Furthermore, from a regular expression of length n , one can construct a deterministic finite automaton with $O(2^n)$ states accepting L .*

Proof. We show how to construct automata to accept regular languages. Clearly, $\emptyset$ and $a \in \Sigma$ are fine. Otherwise, a regular language is built from one of lower logical complexity via one of parts (iii)-(v) of Definition 6.14. But now the result follows by the closure properties of Theorem 6.12. The complexity $O(2^n)$ comes from looking at the complexity of the inductive definitions of the machines for union, star, and concatenation.

Conversely, suppose that L is finite state. Let $L = L(M)$ with $M = \langle K, \Sigma, \delta, S, F \rangle$ a deterministic finite automaton. We need a regular language R with $R = L(M)$. Let $K = \{q_1, \dots, q_n\}$, and $S = \{q_1\}$. For $i, j \in \{1, \dots, n\}$, and $1 \leq k \leq n + 1$, let $R(i, j, k)$ denote the set of all strings in Σ^* derivable in M from state q_i to state q_j without using any state q_m for $m \geq k$.

That is, we let

$$R(i, j, k) = \{\sigma : \langle q_i, \sigma \rangle \vdash^* \langle q_j, \lambda \rangle \wedge \forall \gamma [\langle q_i, \sigma \rangle \vdash^* \langle q_m, \gamma \rangle \rightarrow (m < k \vee (\gamma = \lambda \wedge m = j) \vee (\gamma = \sigma \wedge m = i))]\}.$$

Notice that $R(i, j, n + 1) = \{\sigma : \langle q_i, \sigma \rangle \vdash^* \langle q_j, \lambda \rangle\}$. Thus,

$$L(M) = \cup\{R(i, j, n + 1) : q_i \in S \wedge q_j \in F\}.$$

Since the union of a finite number of regular languages is again regular, Theorem 6.16 will follow once we verify the lemma below.

Lemma 6.17 *For all i, j, k , $R(i, j, k)$ is regular.*

Proof of Lemma 6.17. We prove Lemma 6.17 by induction upon k , the hypothesis being that $R(i, j, p)$ is regular for all $p \leq k$.

For $k = 1$,

$$R(i, j, 1) = \{\sigma : \delta(q_i, \sigma) = q_j\}.$$

Then $R(i, j, 1)$ is finite and hence regular. Now, we claim that

$$R(i, j, k + 1) = R(i, j, k) \cup R(i, k, k)R(k, k, k)^*R(k, j, k).$$

To see that the claim holds, note that $R(i, j, k + 1)$ is the collection of strings σ with $\langle q_i, \sigma \rangle$ moving to $\langle q_j, \lambda \rangle$ without using states with indices $\geq k + 1$. Now to go from state q_i to state q_j without involving q_m for $m \geq k + 1$, we must choose one of the following options.

- (a) Go from q_i to q_j and only involve states q_p for $p < k$. This is the set $R(i, j, k)$.
- (b) Otherwise go from q_i to q_j and necessarily involve q_k . Therefore, we must choose one of the three suboptions below.

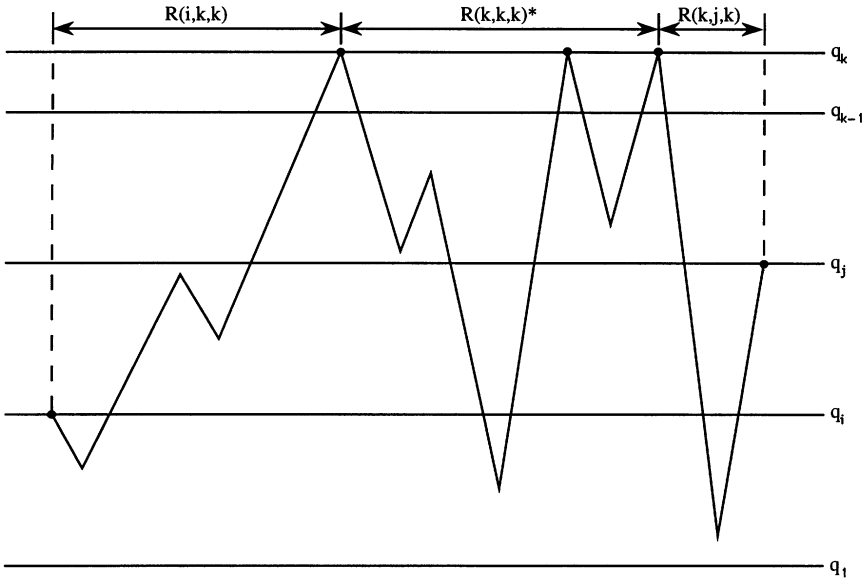


FIGURE 6.6. Case (b) of Lemma 6.17

- (i) First go from q_i to q_k using only q_p for $p < k$. [This is $R(i, k, k)$.]
- (ii) Go from q_k to q_k some number of times, using only states q_p for $p < k$. [This is $R(k, k, k)$ each time and hence gives $R(k, k, k)^*$.]
- (iii) Finally, go from q_k to q_j only using states q_p for $p < k$. [This is $R(k, j, k)$.]

Figure 6.6 may be helpful in visualizing option (b) above. Thus, the contribution of (b) is $R(i, k, k)R(k, k, k)^*R(k, j, k)$. Now we can apply the induction hypothesis to each of the expressions in the decompositions of $R(i, j, k + 1)$ of the claim. This observation establishes the claim, and hence the lemma and finishes the proof of Theorem 6.16.

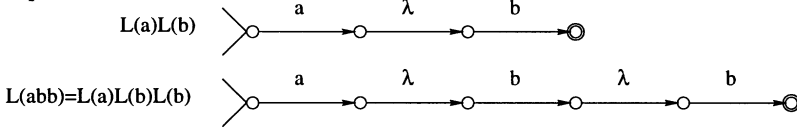
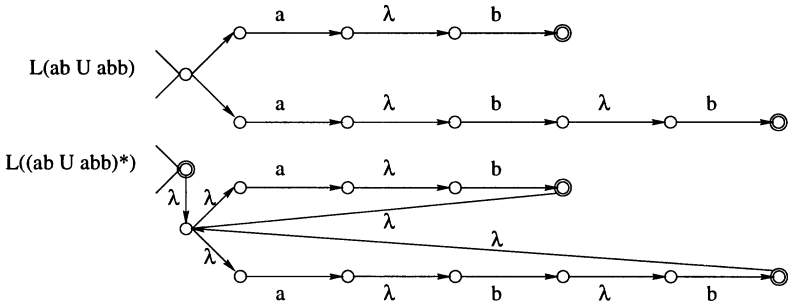
Example 6.18 Figure 6.7 demonstrates the construction of a nondeterministic automaton accepting a given regular expression. Let $L = L((ab \cup abb)^*)$.

Example 6.19 Armed with Theorem 6.16, it is easy to write a language that takes exponential time for Thompson's construction. A deterministic automaton accepting the language

$$L = (0 \cup 1)^* 0 \underbrace{(0 \cup 1) \cdots (0 \cup 1)}_{n \text{ times}}$$

can be shown to require $O(2^n)$ states but is accepted by a nondeterministic automaton with $O(n)$ states (Exercise 1).

Step 1

Step 2

Step 3

 FIGURE 6.7. Construction of an Automaton Accepting L

We remark that it is possible to construct a nondeterministic finite automaton with $k + 1$ states for which *any* deterministic automaton requires $2^k + 2$ states (Leiss [328]) (See Exercise 2).

Historical Notes

Kleene's theorem is taken from Kleene [298]. The now-standard proof that finite-state languages are regular was formulated by McNaughton and Yamada [348].

Exercises 6.1

- 6.1.1.** (i) Following the proof of Theorem 6.16, construct a nondeterministic finite automaton with $O(n)$ states accepting the language L of Example 6.19.
 (ii) Prove that any deterministic finite automaton accepting such L requires $\Omega(2^n)$ states.

6.1.2. (Leiss [328]) Let L be the language denoted by the regular expression

$$2(0 \cup 1)^* 12^*((0 \cup 1)2^*)^n.$$

Prove that L can be represented by a nondeterministic finite automaton with $n + 2$ states, but any deterministic finite automaton accepting L needs at least $2^n + 1$ states.

6.1.4 The Myhill–Nerode Theorem and the Method of Test Sets

The Myhill–Nerode Theorem is due to Nerode [361], although Myhill has a similar result in [358]. (See the Historical Notes at the end of the section.) The Myhill–Nerode Theorem is a beautiful, purely algebraic characterization of the computational notion of having a finite automaton accepting a language. The characterization has *no* reference to computation or dynamic notions such as regular expressions. The theorem is of great importance in terms of practical algorithms for minimizing the number of states in a finite automaton. As we will see when we look at the method of test sets, the Myhill–Nerode Theorem is also of importance in constructing automata for regular languages with only *implicit* computational descriptions, rather than *explicit* ones such as being specified by a regular expression. This latter property will be used extensively when we look at graphs where often all we have is such an implicit description. However, first we will need the theorem!

Definition 6.20 (Right Congruence)

- (a) We call a relation R on a set Σ^* a *right congruence* (with respect to concatenation) iff
 - (i) R is an equivalence relation on Σ^*
 - (ii) for all $x, y \in \Sigma^*$, xRy iff for all $z \in \Sigma^*$, $xzRyz$.
- (b) For a language L , the *canonical right congruence* (induced by L) is the relation $\sim_L$ defined by

$$x \sim_L y \text{ iff for all } z \in \Sigma^*, xz \in L \text{ iff } yz \in L.$$

Intuitively, right congruences must agree on all extensions. Similarly, we can define a *left congruence* by replacing “ $xzRyx$ ” in Definition 6.20 by “ $zxRzy$.” Finally, we say that R is a congruence if it is both a left and a right congruence. An equivalence relation on Σ^* is said to have *finite index* if it has only a finite number of equivalence classes over Σ^* .

Theorem 6.21 (The Myhill–Nerode Theorem)

- (a) *The following are equivalent over Σ^* .*
 - (i) L is finite state.
 - (ii) L is the union of a collection of equivalence classes of a right congruence of finite index over Σ^* .
 - (iii) $\sim_L$ has finite index.
- (b) *Furthermore, any right congruence satisfying (ii) is a refinement² of $\sim_L$.*

²Recall that an equivalence relation E is said to be a refinement of an equivalence relation F iff the equivalence classes of F are unions of equivalence classes of E .

Proof. (i) $\rightarrow$ (ii). Suppose that L is regular and is accepted by the deterministic finite automaton $M = \langle K, \Sigma, S, F, \delta \rangle$. Without loss of generality, let $S = \{q_0\}$. We define $R = R_M$ via

$$xRy \text{ iff there is a } q \in K \text{ such that } \langle q_0, x \rangle \vdash^* \langle q, \lambda \rangle \text{ and } \langle q_0, y \rangle \vdash^* \langle q, \lambda \rangle$$

(that is, on input x or y , we finish up in the same state q). It is evident that R_M is a right congruence since, for all $z \in \Sigma^*$,

$$\begin{aligned} \langle q_0, xz \rangle &\vdash^* \langle q, z \rangle \\ \langle q_0, yz \rangle &\vdash^* \langle q, z \rangle. \end{aligned}$$

Also, the index of R_M is finite since there are only finitely many states in K . The theorem follows since $L = \cup\{\sigma \in \Sigma^* : \langle q_0, \sigma \rangle \vdash^* \langle q, \lambda \rangle\}$ with $q \in F$.

(ii) $\rightarrow$ (iii). We prove (b) from which (ii) $\rightarrow$ (iii) follows. Let R be a right congruence of finite index so that L is a union of some of R 's equivalence classes. By the fact that R is a right congruence, for all z ,

$$xz \in L \text{ iff } yz \in L.$$

Part (b) follows immediately from this.

(iii) $\rightarrow$ (i). Assume that (iii) holds. Then, we have the following implicit description of deterministic finite automaton M accepting L .

1. The states of M are the equivalence classes of $\sim_L$.
2. The initial state is $[\lambda]_{\sim_L}$, the equivalence class of λ .
3. The transition function δ of M is defined by $\delta([a], x) = [ax]$ for $a, x \in \Sigma$.
4. The accepting states of M are those x with $[x] \subseteq L$.

Notice that item 3 is well defined as $\sim_L$ is a right congruence. (The point is that $\sim_L$ being a right congruence makes the definition of δ independent of the particular representative of the equivalence class: If $a \sim_L b$, then

$$\delta([a], x) = [ax] = [bx] = \delta([b], x).)$$

We see that $L = L(M)$ and this concludes our proof.

We remark that it is possible to use the Myhill–Nerode Theorem directly to establish that certain languages are finite state. Example 6.23 is an illustration of such use of the Myhill–Nerode Theorem. First, we need a definition.

Definition 6.22 (Myhill's Congruence) Define $x \approx_L y$ to mean that for all $u, v \in \Sigma^*$,

$$uxv \in L \text{ iff } uyv \in L.$$

It is easy to prove that $\approx_L$ has finite index iff $\sim_L$ has finite index (Exercise 1).

Example 6.23 *If L is finite state, so is the reverse L^{rev} of L .*

To see this, define xRy to hold iff $x^{rev} \approx_L y^{rev}$. As $\approx_L$ is an equivalence relation, so is R . Furthermore, by the Myhill–Nerode Theorem, $\sim_L$ has finite index and hence $\approx_L$ has finite index. This implies that R has finite index. Finally, to see that R is a right congruence, suppose that xRy and $z \in \Sigma^*$. Now $x^{rev} \approx_L y^{rev}$ and since $\approx_L$ is a (left) congruence, we have $z^{rev}x^{rev} \approx_L z^{rev}y^{rev}$ and hence $(xz)^{rev} \approx_L (yz)^{rev}$. This implies that $xzRyz$ and hence R is a right congruence.

Our result will then follow from the Myhill–Nerode Theorem once we can establish that L^{rev} is a union of equivalence classes of R . We need to show that if xRy , then $x \in L^{rev}$ implies $y \in L^{rev}$. If $x \in L^{rev}$, then $x^{rev} \in L$. If xRy , then $x^{rev} \approx_L y^{rev}$ and hence $y^{rev} \in L$, which implies that $y \in L^{rev}$.

In Exercises 2-4, we invite the reader to apply the technique of Example 6.23 to other languages.

Another place we can use the Myhill–Nerode characterization is to efficiently demonstrate *non* regularity of languages by exhibiting infinitely many equivalence classes for $\sim_L$. This technique is often significantly more elegant than classical approaches using the “Pumping Lemma” (Corollary 6.25 and Exercise 8). Here is an example.

Example 6.24 We will demonstrate that the language $L = \{a^n b^n : n > 1\}$ is not regular using the Myhill–Nerode Theorem. For all $n < m$, $a^n \not\sim_L a^m$ since $a^n b^n \in L$ yet $a^m b^n \notin L$. Hence, $\sim_L$ has infinite index, and therefore L is not regular.

In fact, the classical Pumping Lemma is an easy consequence of the Myhill–Nerode Theorem as we now see.

Corollary 6.25 (Pumping Lemma, Bar-Hillel, Perles and Shamir [51]) *Suppose that L is regular and infinite. Then there exist strings x , y , and z such that for all $n \geq 1$,*

$$xy^n z \in L.$$

Proof. Define a collection C of strings inductively. Let $y_0 = \lambda$ and y_{n+1} denote the lexicographically least string extending y_n such that (i) $|y_{n+1}| = n + 1$ and (ii) y_{n+1} has infinitely many extensions in L . By the Myhill–Nerode Theorem, there exist $n < m$ such that $y_n \sim_L y_m$. Let $x = y_n$ and $y_m = xy$ and suppose that $y_m z \in L$. Then, by the Myhill–Nerode Theorem, for all k , $xy^k z \in L$.

We remark that the classical proof of Theorem 6.25 is given in Exercise 8.

Observation 6.26 (The Method of Test Sets) *The proof of (iii) in Theorem 6.21 gives rise to what we call The Method of Test Sets.*

Suppose that we have a language L which we know is regular but we are not given a regular expression for L , only a representation via a decision procedure for membership (via, say, a Turing machine). Is it possible to deduce an automaton accepting L ? Using a simple coding argument, we see that the answer is “no!”

Observation 6.27 *In general, there is no algorithm to deduce an automaton for a regular language from a machine description of the language.*

Proof. We define a family $\{\varphi_{s(x)} : x \in \mathbb{N}\}$ of recursive functions with $\varphi_{s(x)}(y) = 1$ if $\varphi_x(x) \downarrow$; and $\varphi_{s(x)}(y) \uparrow$, otherwise. Here, φ_x denotes the x -th partial recursive function. Let $V_x = \text{dom}(\varphi_{s(x)})$. Then, V_x is either $\emptyset$ or $\mathbb{N}$ and, furthermore, $V_x = \mathbb{N}$ iff $\varphi_x(x) \downarrow$. Now since we can easily see if a language is empty or not from an automaton for L if we could construct automata for the V_x , then we could decide if $V_x = \emptyset$ or not, and hence decide if $\varphi_x(x) \downarrow$. This contradicts the algorithmic undecidability of the Halting Problem.

Despite Observation 6.27, we *can* deduce the automaton if we additionally know, for instance, some bound on the number of states of such an automaton. The method of test sets allows to solve the following “promise problem.”

Input: A decision procedure for a language L .
Promise: L is regular and there is some M accepting L with at most k states.
Question: Can we compute an automaton M' accepting L ?

Theorem 6.28 The promise problem above is solvable and, furthermore, we can compute M' with at most k states accepting L .

Proof. The idea is simple. We use the proof (iii) $\rightarrow$ (i) of Theorem 6.21. The machine is described by items 1 to 4 of that proof. The only fact that we need to check is that we can decide if $x \sim_L y$ for $x, y \in L$. By definition, $x \sim_L y$ iff for all $z \in L$, $xz \in L$ iff $yz \in L$. Suppose that M is any automaton accepting L with n states, and $x \not\sim_L y$. We claim that there some $z \in \Sigma^*$ with $|z| \leq n$ and z a witness to $x \not\sim_L y$. The point is that there is some z' such that, say, $xz' \in L$ yet $yz' \notin L$. This means that if we input xz' into L , we will get to an accept state, and if we input yz' into M , we must get to a nonaccept state. But since there are at most n states in M , the longest path from one state to another can have length n . Thus, we can refine z' down to a string of length $\leq n$ by deleting all loops.

Thus, there is some z with $|z| \leq n$ witnessing that $x \not\sim_L y$. Hence, to decide if $[x] = [y]$, we simply need go through all $z \in \Sigma^*$ with $|z| \leq n$ and see if for those z , $xz \in L$ iff $yz \in L$. The same reasoning demonstrates that for all $x \in \Sigma^*$, there is some $x' \in [x]$ with $|x'|, |y'| \leq n$. Hence, to implement the method of (iii) $\rightarrow$ (i), we need only look at a table of strings of length $\leq n$ and test them with strings of length $\leq n$. The fact that $\sim_L$ defines the coarsest congruence means that the number of states of M' we obtain from this technique must be minimal.

The reader should note that essentially the same proof shows that the following modified problem is solvable.

Input: A decision procedure for a language L , and a decision algorithm for $\sim_L$.
Question: Can we compute an automaton M' accepting L ?

Test Strings z															
$\times$	λ	0	1	00	01	10	11	000	001	010	011	100	101	110	111
λ	No	No	No	No	Yes	No	No	No	No	No	Yes	No	Yes	No	No
0	No	No	Yes	No	No	No	Yes	No	Yes	No	No	No	Yes	No	Yes
1	No	No	No	No	Yes	No	No	No	No	No	Yes	No	Yes	No	No
00	No	No	No	No	Yes	No	No	No	No	No	Yes	No	Yes	No	No
01	Yes	No	Yes	No	Yes	No	Yes	No	No	No	Yes	No	Yes	No	Yes
10	No	No	Yes	No	No	No	Yes	No	Yes	No	No	No	Yes	No	Yes
11	No	No	No	No	Yes	No	No	No	No	No	Yes	No	Yes	No	No
000	No	No	Yes	No	No	No	Yes	No	Yes	No	No	No	Yes	No	Yes
001	No	No	No	No	Yes	No	No	No	No	No	Yes	No	Yes	No	No
010	No	No	Yes	No	No	No	Yes	No	Yes	No	No	No	Yes	No	Yes
011	Yes	No	Yes	No	Yes	No	Yes	No	No	No	Yes	No	Yes	No	Yes
100	No	No	No	No	Yes	No	No	No	No	No	Yes	No	Yes	No	No
101	Yes	No	Yes	No	Yes	No	Yes	No	No	No	Yes	No	Yes	No	Yes
110	No	No	Yes	No	No	No	Yes	No	Yes	No	No	No	Yes	No	Yes
111	No	No	No	No	Yes	No	No	No	No	No	Yes	No	Yes	No	No

TABLE 6.1. Table of Test Set Answers

Example 6.29 Here is an example of the method of test sets. We will pretend that we have been implicitly given a regular language over $\{0, 1\}^*$ and know that there is some automaton M with ≤ 4 states accepting L . Suppose that we obtained the table of answers of Table 6.1 from the decision procedure for L .

We note that

$$\begin{aligned}
 [\lambda] &= [1] = [11] = [111] = [100] = [00] = [001], \\
 [0] &= [10] = [110] = [000] = [010], \\
 [01] &= [101] = [011].
 \end{aligned}$$

Hence the three states of M' are $[\lambda]$, $[0]$, and $[01]$. Note that $[01]$ will be the accept state since $01 \in L$. The transition diagram is given in Figure 6.8.

An important consequence of the proof of the Myhill–Nerode Theorem is the fact that the automaton in (iii) $\rightarrow$ (i) has the least number of states of any deterministic automaton accepting L . Implicitly, the proof provides us with a *state minimization*

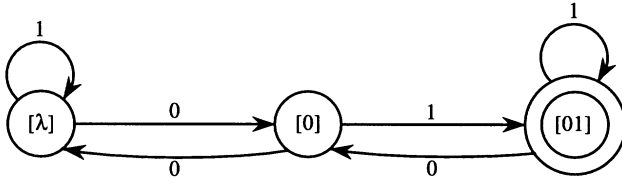


FIGURE 6.8. The Resulting Automaton

procedure; that is, we can take an automaton M accepting L , and identify states equivalent under $\sim_L$. Again, since the number of states of L determines the length of the longest path in L , we only need to look at strings of length $\leq |K|$. One minimization algorithm derived from these observations is given below.

Algorithm 6.30 (State Minimization Algorithm)

Step 1. Make a table of entries $(p, q) \in K \times K$ for $p \neq q$.

Step 2. (Round 1) For each (p, q) with $p \in F \wedge q \notin F$, mark (p, q) .

Step 3. Repeat the following in rounds until there is no change from one round to the next.

(Round $i + 1$ for $1 \leq i$) For all (p, q) not yet marked by the end of round i , if for some $a \in \Sigma$, $(\delta(p, a), \delta(q, a))$ is marked, then mark (p, q) .

Step 4. Any unmarked squares representing (p, q) represent states that are equivalent and hence can be identified.

In Figure 6.9, we give an application of the minimization algorithm.

We remark that in Round 2, the only unmarked (x, y) are (a, e) , (b, c) , (b, d) , and (d, c) . We mark (c, d) since $\delta(c, 0) = d$, $\delta(d, 0) = e$ and (d, e) was marked in Round 1. We mark (b, d) since $\delta(b, 0) = d$ and $\delta(d, 0) = e$, and, finally, we mark (e, a) since $\delta(e, 1) = e$ and $\delta(a, 1) = b$. In Round 3, there is no change from Round 2 and the algorithm terminates. Notice that the cost of the algorithm is at most $O(n^2)$ if M has n states (for a fixed Σ).

Historical Notes

The Myhill–Nerode Theorem is due to Nerode [361]. Myhill [358] proved Theorem 6.21 but with the congruence $\approx_L$ of Definition 6.22 in place of $\sim_L$ in (iii), and proved that $\approx_L$ is the coarsest congruence (rather than *right* congruence) such that L is a union of equivalence classes of $\approx_L$. The algorithm to minimize the number of states of a finite automaton can be found in Huffman [273] and Moore [355]. The Pumping Lemma is due to Bar-Hillel et al. [51]. The Method of Test Sets is implicit in the Myhill–Nerode Theorem but, as best we can tell, was only made explicit in Abrahamson and Fellows [7] where it was applied in a general setting of languages consisting of parse trees of graphs of bounded treewidth. (We will discuss this point more fully in Section 6.4.)

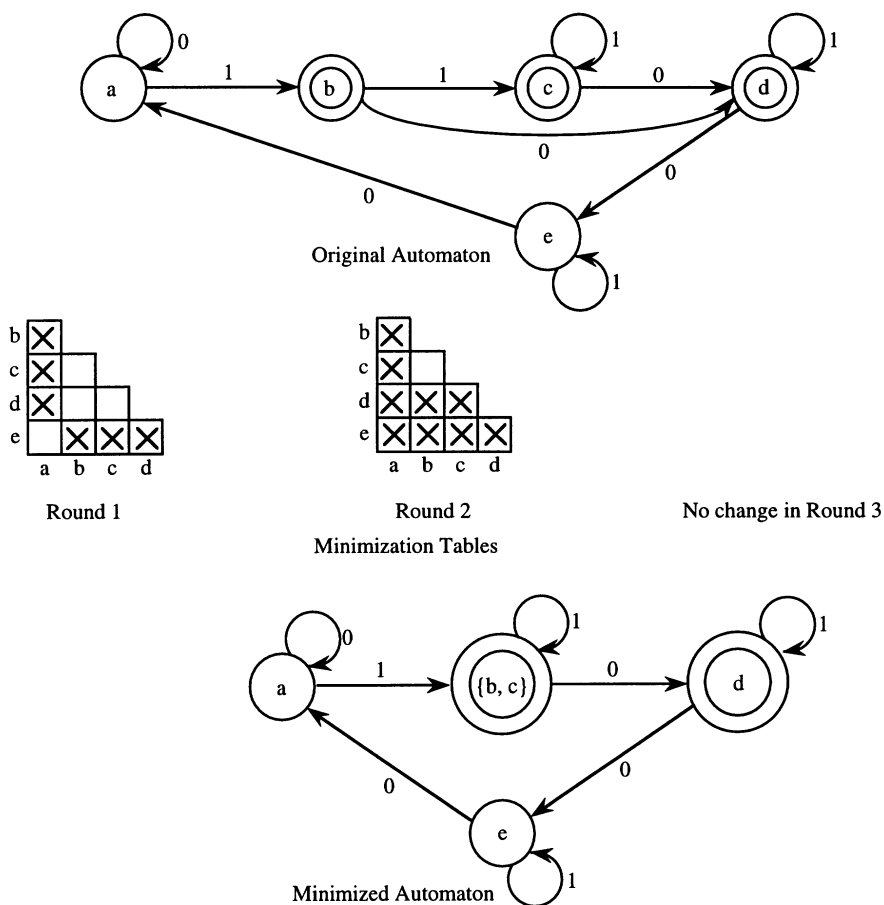


FIGURE 6.9. The Minimization Algorithm

Exercises 6.1

6.1.1. Prove that the equivalence relation $\approx_L$ of Definition 6.22 has finite index iff $\sim_L$ does.

6.1.2. Let L_1 and L_2 be finite state. Define $shuffle(L_1, L_2) = \{x : x \in shuffle(u, v) \text{ where } u \in L_1 \text{ and } v \in L_2\}$. Here, $shuffle(u, v) = \{y : y \text{ is obtained by some interleaving of the sequences of letters of } u \text{ and } v\}$.

Prove that $shuffle(L_1, L_2)$ is finite state using the technique of Example 6.23.

(Hint: Define R via: xRy iff $(x \sim_{L_1} y \wedge x \sim_{L_2} y \wedge S_x = S_y)$, where $S_z = \{[u]_{\sim_{L_1}}, [v]_{\sim_{L_2}} : z \in shuffle(u, v)\}$.)

6.1.3. Let $x, y \in \Sigma^*$. Define the *Hamming distance* between x and y to be

$$d(x, y) = \begin{cases} \text{the number of letters that } x \text{ and } y \text{ differ,} & \text{if } |x| = |y|, \\ \infty & \text{otherwise,} \end{cases}$$

Use the method of Example 6.23 to prove that for any $r \in \mathbb{Z}$, if L is finite state, then so is

$$H_r(L) = \{x \in \Sigma^* : \exists y \in L[d(x, y) \leq r]\}.$$

(Hint: For $x \in \Sigma^*$, let $S_{x,i} = \{[u]_{\sim_L} : d(x, u) = i\}$. For $i \leq r$, let $xR_i y$ iff $S_{x,i} = S_{y,i}$. Argue via $R = \bigcap_{0 \leq i \leq r} R_i$.)

6.1.4. Use the method of Example 6.23 to prove that if L is finite state, then so is $(1/3)L = \{x \in \Sigma^* : \exists y \in \Sigma^*(xy \in L \wedge |xy| = 3|x|)\}$.

(Hint: Define xRy to hold iff $x \approx_L y \wedge S_x = S_y$, where $S_z = \{[u]_{\sim_L} : |zu| = 3|z|\}$.)

6.1.5. Apply the method of test sets (of Observation 6.26) to the data in the table of Table 6.2.

6.1.6. Apply the state minimization algorithm (Algorithm 6.30) to the automaton of Figure 6.10.

6.1.7. Use the technique of Example 6.24 to demonstrate the nonregularity of the following languages:

- (i) $\{a^p : p \text{ is prime}\}$.
- (ii) $\{a^n b^m : n \neq m\}$.

6.1.8. (The Pumping Lemma) In this exercise, we will use the classical approach to proving Pumping Lemma: *If L is an infinite regular language, then there exist $x, y, z \in \Sigma^*$ with $y \neq \lambda$ such that*

$$\forall n > 0 (xy^n z \in L).$$

Prove the Pumping Lemma (Corollary 6.25) by using induction on the length of the regular expression α with $L = L(\alpha)$.

6.1.9. (Kozen [306]) For a language A and a function f , define

$$A_f = \{y : \exists x(|y| = f(|x|)) \wedge xy \in A\}.$$

We call a function *regularity preserving* if A is regular if A_f is regular. A function $f : \mathbb{N} \mapsto \mathbb{N}$ is called *eventually periodic* if there exists a $p > 0$ such that for almost all x , $f(x) = f(x + p)$. Finally, a language $U \subseteq \mathbb{N}$ is called eventually periodic if there exists a q such that for almost all n ,

$$n \in U \text{ iff } n + p \in U.$$

Prove that the following are equivalent for $f : \mathbb{N} \mapsto \mathbb{N}$.

- (i) f is regularity preserving.
- (ii) A is regular iff $A'_f =_{\text{def}} \{x : \exists y(y \in A \wedge f(|y|) = |x|)\}$.

Test Strings z															
$\times$	λ	0	1	00	01	10	11	000	001	010	011	100	101	110	111
λ	No	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes
0	No	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes
1	Yes	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes
00	No	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes
01	Yes	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes
10	No	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes
11	Yes	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes
000	No	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes
001	Yes	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes
010	No	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes
011	Yes	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes
100	No	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes
101	Yes	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes
110	No	No	Yes	No	Yes	No	Yes	No	No	No	Yes	No	Yes	No	Yes
111	Yes	No	Yes	No	No	No	Yes	No	Yes	No	No	No	Yes	No	Yes

TABLE 6.2. Table for Exercise 5

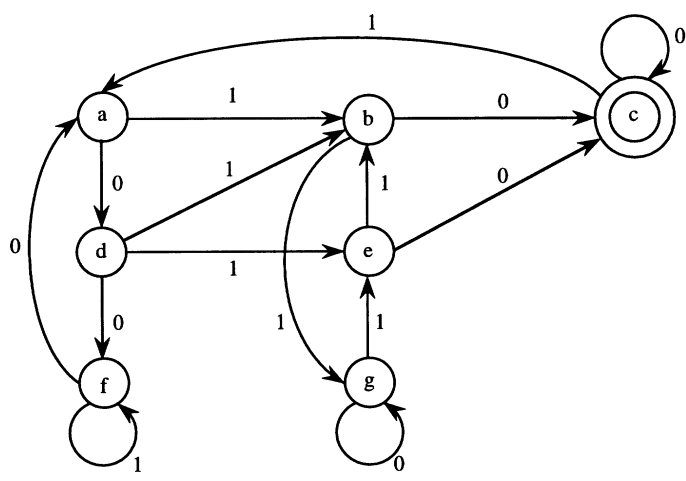


FIGURE 6.10. Automaton for Exercise 6

(iii) f is eventually periodic modulo m for all $m \geq 1$ and $f^{-1}(\{n\})$ is eventually periodic for all $n \in \mathbb{N}$.

(iv) f preserves eventual periodicity.

(Hint: Use Myhill–Nerode.)

6.1.5 Classical Tree Automata

The purpose of this section is to generalize the class of objects accepted by automata to labeled rooted trees. To allow such a generalization, we will need to modify our notion of automaton. Once we have done this modification, we find that the resulting theory follows the standard theory fairly closely. Our treatment will be a little nonstandard and will be basically tailored to the graph-theoretic applications in the sections to follow. In particular, as best we can, we will avoid the abstraction to universal algebra that many authors use. It suffices to say that it is possible to put all these results in a more general context such as by Gecseg and Steinby [242], Steinby [434], Mezei and Wright [351], or Kozen [305].

By a *tree*, we will mean a (directed) rooted tree³ with labels chosen from Σ at each of the nodes, and each node has fanin bounded by some constant f . We will denote the collection of such trees over Σ as Σ^{**} . If $T \in \Sigma^{**}$, we will denote the root of T by r_T .

In Figure 6.11, we give an example of a tree over $\{0, 1\}$. Although the example of Figure 6.11 is a binary tree, any fixed fanin f will do. The reader would do well to keep the binary example in mind, for most applications seem to need only binary trees. Notice that we can consider a string as a *unary* tree with the single leaf of a string such as $a_1 \dots a_n$ being the leftmost symbol a_1 and a_n being the root. As we have seen, automata act on such unary trees by reading the symbols from leaf to root, changing states as they move down the tree. The extension to more general trees is natural. In a *leaf-to-root* tree automaton M , we will move

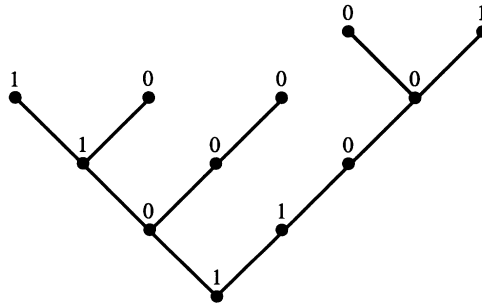


FIGURE 6.11. An Example of a Tree

³We will not indicate the directions of the edges on the trees since it will be totally clear from context which vertex is the root.

from the leaves to the root, changing states as we are directed by the labels on the nodes. We know what such movement means for *unary* nodes, but we have not specified what such movement means for k -ary nodes for $k \leq f$. The idea is clear. Suppose that v is a node of a tree T with k inputs $i_1, \dots, i_k$, and suppose that v is labeled $a \in \Sigma$. Inductively, we may suppose that corresponding to the k input lines $i_1, \dots, i_k$, the machine would be in states $q_{j_1}, \dots, q_{j_k}$ (i.e., “simultaneously”). Then, at node v the machine would move to the state specified by the $k + 1$ -tuple $\langle q_{j_1}, \dots, q_{j_k}, a \rangle$. (Here, of course, we are regarding the nodes v of the tree as being lexicographically ordered in some standard fashion by $\leq_L$. Thus, $i_1 <_L \dots <_L i_k$, for example.)

Definition 6.31 (Deterministic Finite Tree Automaton) A *Tree Automaton* over Σ is thus a quintuple $\langle K, \Sigma, \delta, S, F \rangle$ and a bound f (the *fanin* bound) where K, S, F are as for a standard automaton (as in Definition 6.1) and δ is a function from (a subset of) $\cup_{1 \leq i \leq f} \underbrace{(K \times \dots \times K)}_{i \text{ times}} \times \Sigma$ to K .

We remark that it is also possible to study *root-to-leaf* automata (Exercise 1).

Notice that if the trees were all binary and all nonleaf nodes had fanin 2 and $S = \{q_0\}$, then we could equally well specify δ simply by specifying $\delta(q_0, a)$ for all $a \in \Sigma$ and $\delta(q_i, q_j, a)$ for $q_i, q_j \in K$ and $a \in \Sigma$.

We will have an analogy of the “yield” relation $\vdash^*$ for standard deterministic finite automata. We call this an evaluation rule *eval*. Let $T_1, \dots, T_n$ be labeled rooted trees. We denote by $T_1 \odot_a \dots \odot_a T_n$ the tree obtained by adding a further node r and attaching the roots $r_{T_1}, \dots, r_{T_n}$ to r and labeling r by a . The example in Figure 6.12 makes this basic action clear.

It is obvious that all trees in Σ^{**} are obtainable from Σ and $\odot_a$ for $a \in \Sigma$. We regard $\lambda \odot_a = a$ for $a \in \Sigma$, so, in fact, all of Σ^{**} can be generated by λ and $\odot_a$, for $a \in \Sigma$.

Definition 6.32 (Tree Automaton Acceptance) If $M = \langle K, \Sigma, \delta, S, F \rangle$ is a finite deterministic tree automaton, the function $eval_M$ is defined via

- (i) $eval_M(q_j, a) = \delta(q_j, a)$ for $a \in \Sigma$ and $q_j \in S$ [and, without loss of generality, we will assume that $S = \{q_0\}$ and write $eval(a)$ for $eval(q_0, a)$].
- (ii) $eval_M(T_1 \odot_a \dots \odot_a T_n) = \delta(eval_M(T_1), \dots, eval_M(T_n), a)$ for $a \in \Sigma$.

We say that $T \in \Sigma^{**}$ is *accepted* iff $eval_M(T) \in F$.

Example 6.33 Specify M as follows. Let $\Sigma = \{0, 1\}$ so that Σ^{**} is the collection of all rooted, labeled binary trees. Let $S = \{q_0\}$ and $F = \{q_1\}$. We specify δ by the following table.

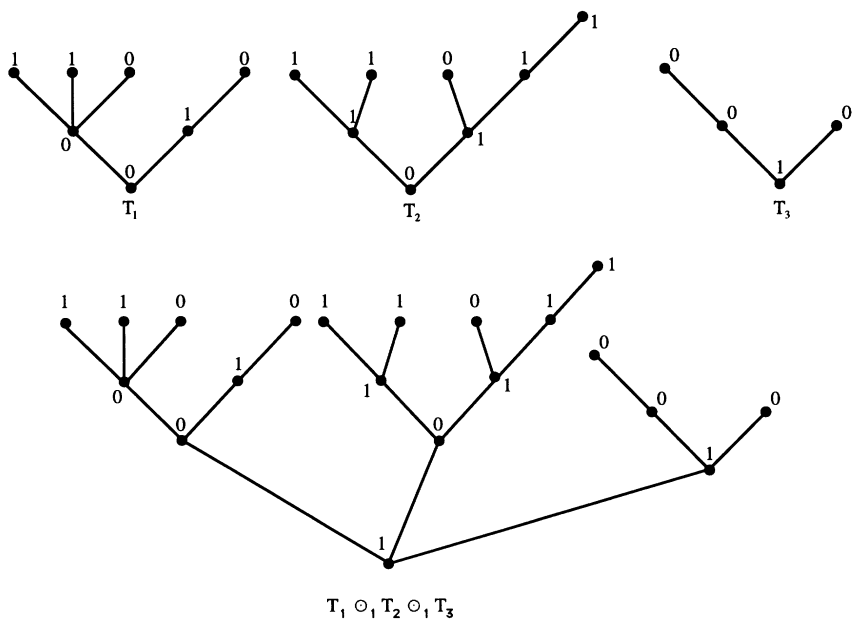


FIGURE 6.12. Gluing Labeled Trees

States	Symbols	$\delta(\text{states},\text{symbol})$
q_0	0	q_0
q_0	1	q_1
q_1	0	q_1
q_1	1	q_1
q_0, q_0	0	q_0
q_0, q_0	1	q_1
q_0, q_1	0	q_0
q_0, q_1	1	q_1
q_1, q_0	0	q_0
q_1, q_0	1	q_0
q_1, q_1	0	q_0
q_1, q_1	1	q_1

In Figure 6.13, we have marked the stages of computing $eval(T)$ for the given tree. Note that $eval(T) = q_0$ and hence $T \notin L(M)$ for the given tree automaton M .

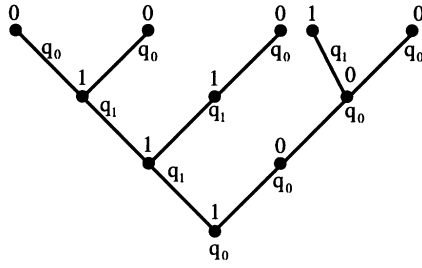


FIGURE 6.13. Computation of a Tree Evaluation

If we were to represent the evaluation of Figure 6.13 via $\odot$, we would have the following:

$$\begin{aligned}
T &= [(0 \odot_1 0) \odot_1 (0 \odot_1)] \odot_1 [(1 \odot_0 0) \odot_0] \\
eval(T) &= \delta(eval((0 \odot_1 0) \odot_1 (0 \odot_1)), eval((1 \odot_0 0) \odot_0), 1) \\
&= \delta(\delta(eval(0 \odot_1 0), eval(0 \odot 1), 1), \delta(eval(1 \odot_0 0), 0), 1) \\
&= \delta(\delta(\delta(eval(0), eval(0), 1), \delta(eval(0), 1), 1), \\
&\quad \delta(\delta(eval(1), eval(0), 0), 0), 1) \\
&= \delta(\delta(\delta(\delta(q_0, 0), \delta(q_0, 0), 1), \delta(\delta(q_0, 0), 1), 1), \\
&\quad \delta(\delta(\delta(q_0, 1), \delta(q_0, 0), 0), 0), 1) \\
&= \delta(\delta(\delta(q_0, q_0, 1), \delta(q_0, 1), 1), \delta(\delta(q_1, q_0, 0), 0), 1) \\
&= \delta(\delta(q_1, q_1, 1), \delta(q_0, 0), 1) \\
&= \delta(q_1, q_0, 1) \\
&= q_0.
\end{aligned}$$

We remark that there is no problem with defining *nondeterministic finite tree automata* by generalizing δ to a multifunction as we did in Section 6.1.2. We invite the reader to check that the details of Rabin and Scott's theorem (Theorem 6.6) go through exactly as before since the subset/Thompson construction will work in the same way (Exercises 3 and 4).

We do encounter a little more difficulty when we hit the result that says “finite-state languages are regular” since we do not yet have an analog of the concept of regularity for trees. We need a syntactic definition of languages accepted by tree automata. As we will see, there are two such notions.

For our first syntactic definition, we will view finite-state tree languages as a subset of the set of *context-free languages*. (See, e.g., Hopcroft and Ullman [270].) For such purposes, it is easiest to think in terms of the algebraic approach given in Example 6.33. It is easiest to think in terms of *productions* which involve “replacements,” as we see below. For this purpose, we augment the set of trees with some new *leaf* symbols. Let R be any set of new symbols disjoint from Σ . We will say a tree T is a tree *over* Σ_R if T is the result of taking a tree over Σ and (possibly) replacing some of the *leaves* by members of R . We will let Σ_R^{**} denote the collection of trees over Σ_R .

Definition 6.34 (Tree Grammar) A tree grammar G over Σ is a quadruple $\langle \Sigma, S, R, P \rangle$ where

- (i) R is a set of symbols with $R \cap \Sigma = \emptyset$ called *nonterminal* symbols,
- (ii) $S \in R$ is a distinguished symbol called the *initial* symbol, and
- (iii) P is a set of *productions* of the form

$$N \rightarrow Q \text{ with } N \in R \text{ and } Q \in \Sigma_R^{**}.$$

We will sometimes call Σ the set of *terminal* symbols.

For $T_1, T_2 \in \Sigma_R^{**}$, we will write $T_1 \rightarrow T_2$ if there exist words X_i and Y_i for $i \in \{1, 2\}$, a *leaf* $N \in R$ of T_1 , and a production $N \rightarrow Q$ such that $T_1 = X_1 N Y_1$ and $T_2 = X_2 Q Y_2$. Clearly, we can extend the notion of $\rightarrow$ inductively as we did with $\vdash$ and, hence, for $T \in \Sigma_R^{**}$, we say that $T \rightarrow^* T'$ iff there are $T_1, \dots, T_n$ such that

$$T \rightarrow T_1 \rightarrow \dots \rightarrow T_n \rightarrow T'.$$

We refer to such a string as a *derivation* of T' from T . Finally, we say that $T \in L(G)$ if

- (i) $T \in \Sigma^{**}$ (that is, T has no remaining nonterminal symbols), and
- (ii) there is a derivation of T obtainable from S using the productions of G .

Example 6.35 Here is a simple example. Let $\Sigma = \{0, 1, 2\}$. The productions are $S \rightarrow N_1$ together with the following.

1. $N_1 \rightarrow 0, N_2 \rightarrow 1,$
2. $N_1 \rightarrow (N_1 \odot_0),$
3. $N_1 \rightarrow (N_1 \odot_0 N_2 \odot_0 N_3),$
4. $N_2 \rightarrow ((N_2 \odot_0 2 \odot_0 1) \odot_1 ((0 \odot_0) \odot_1) \odot_1 1),$ and
5. $N_3 \rightarrow (1 \odot_2)$

In Figure 6.14, we represent the productions above graphically, and represent graphically the derivation of the tree T with algebraic derivation below.

$$\begin{aligned}
 S &\rightarrow N_1 \\
 &\rightarrow (N_1 \odot_0) && \text{(by 2),} \\
 &\rightarrow ((N_1 \odot_0 N_2 \odot_0 N_3) \odot_0) && \text{(by 3)} \\
 &\rightarrow ((0 \odot_0 N_2 \odot_0 N_3) \odot_0) && \text{(by 1)} \\
 &\rightarrow ((0 \odot_0 ((N_2 \odot_0 2 \odot_0 1) \odot_1 ((0 \odot_0) \odot_1) \odot_1 1) \odot_0 N_3) \odot_0) && \text{(by 4)} \\
 &\rightarrow ((0 \odot_0 ((1 \odot_0 2 \odot_0 1) \odot_1 ((0 \odot_0) \odot_1) \odot_1 1) \odot_0 N_3) \odot_0) && \text{(by 1)} \\
 &\rightarrow ((0 \odot_0 ((1 \odot_0 2 \odot_0 1) \odot_1 ((0 \odot_0) \odot_1) \odot_1 1) \odot_0 (1 \odot_2)) \odot_0) && \text{(by 5)}
 \end{aligned}$$

Lemma 6.36 (Grammar Normalization Lemma) *Suppose that there is a tree grammar G accepting L . Then, we can compute an equivalent grammar G' from G with the property that if $N \rightarrow T$ is a production of G' , then the height of T is ≤ 1 . Furthermore, we can ensure that G has no productions of the form $N_1 \rightarrow N_2$ with both N_1 and N_2 in $R(G')$. We will call such G' *normalized*.*

Proof. First, note that all productions of the form $N_1 \rightarrow N_2$ with both N_1 and N_2 in R can be deleted, provided that we add to P all productions of the form

the set of labels of roots of trees Q with $N \rightarrow Q$, a production of G . If $a \in \Sigma$, let $\text{root}(a) = \{a\}$. Finally, if $C \in \Sigma \cup R$, we let $\text{entry}(C)$, the entry state of C , be $\{q_0\}$ if $C \in \Sigma$, and $\text{entry}(C) = \{q_C\}$ otherwise.

For each $N \in R$, we will create a state q_N . Now consider a production $N \rightarrow Q$. If $Q = b \in \Sigma$, then we will create a transition $\delta(q_0, b) = q_N$. If Q has height 1, then Q is of the form $b_1 \odot_a \cdots \odot_a b_m$ for some $a \in \Sigma$ and $b_1, \dots, b_m \in \Sigma \cup R$. We will associate new states $q'_i = q'_{b_i, N, Q}$ for $1 \leq i \leq m$, and transitions of the form

$$\delta(\text{entry}(b_i), \text{root}(b_i)) = q'_i,$$

and,

$$\delta(q'_1, \dots, q'_m, a) = q_N.$$

The point here is that the leaves of Q are either labeled with some symbol $M \in R$ or they are in Σ . In the latter case, the only possible entry state that the automaton can have is q_0 . On the other hand, if the leaf is labeled $M \in R$, then it can only be replaced by another tree in Σ_R^{**} via some production of the form $M \rightarrow Q'$. Now we know that the entry state to Q generated by such a production will be the exit state of Q' , which will be q_M . The symbol that corresponds to the production $M \rightarrow Q'$ will be the root symbol of Q' .

Conversely, suppose that M is a tree automaton accepting L . We need to construct a tree grammar G for L . For each state q_i of M , we will have a nonterminal symbol Q_i . Corresponding to a transition of the form $\delta(q_{i_1}, \dots, q_{i_m}, a) = q_k$, we will have a production of the form

$$Q_k \rightarrow Q_{i_1} \odot_a \cdots \odot_a Q_{i_m}.$$

We also have productions of the form $Q_0 \rightarrow \lambda$ and $S \rightarrow Q_d$ for any accept state q_d . The reader should establish by induction that $L = L(G)$ (Exercise 7).

It is also possible to get another characterization of finite-state tree languages via regular tree expressions. We need a definition.

Definition 6.38 (Product of Tree Languages)

- (i) Let $L, W \in \Sigma^{**}$. For $a \in \Sigma$, the *a-product* of L and W is the language $L \cdot_a W$ obtained by taking trees $T \in W$ and replacing every occurrence of a as a leaf of T by a tree T' of L . (Different leaves can have different T' . If no occurrence of a occurs as a leaf of T , then this action is taken to be vacuous.)
- (ii) We denote by L^{*a} the *a-star* of L ,

$$L^{*a} = \bigcup_{n \geq 1} \underbrace{L \cdot_a \cdots \cdot_a L}_n.$$

Definition 6.39 (Regular Tree Expression) A *regular (tree) expression* is defined by induction via the following:

- (i) $\emptyset$ is a regular expression.
- (ii) If $a \in \Sigma$, then a is a regular expression.
- (iii) If X and Y are regular expressions and $a \in \Sigma$, then $X \cdot_a Y$ is a regular expression.
- (iv) If X is a regular expression and $a \in \Sigma$, then X^{*a} is a regular expression.
- (v) If $m \geq 1$, $a \in \Sigma$, and $X_1, \dots, X_m$ are regular expressions, then so is $X_1 \odot_a \dots \odot_a X_m$.
- (vi) Nothing else.

Of course, we associate a language $L(\alpha)$ with a regular expression α . This is defined in the obvious way: $L(\emptyset) = \emptyset$, $L(a) = \{a\}$, $L(X \cdot_a Y) = L(X) \cdot_a L(Y)$, $L(X^{*a}) = [L(X)]^{*a}$, and, finally,

$$L(X_1 \odot_a \dots \odot_a X_m) = \{T_1 \odot_a \dots \odot_a T_m : T_i \in L(X_i)\}.$$

Naturally, we say that L is regular iff $L = L(\alpha)$ for some regular expression α .

The true analog of Kleene's theorem (Theorem 6.16) is provided by the following.

Theorem 6.40 *For Σ^{**} , a language is finite state iff it is regular*

Proof. First, suppose that L is finite state and is accepted by the tree automaton $M = \langle K, \Sigma, \delta, S, F \rangle$. We model our proof on Theorem 6.16. But we use a trick. Our induction will involve trees in Σ_K^{**} ; that is, the induction will involve trees that are normal except that they may have leaves labeled by states from M . The interpretation of the transition is that δ can only act on a state q if it is state q : $\delta(q, q) = q$.

Let $V \subseteq K$. Let $M(V, j, k)$ denote the language that contains all trees in Σ_V^{**} which take a set of states as entry states and transform them via δ to state j without using any state with index $\geq k$.

We prove by induction on k that $M(V, j, k)$ is regular for all V . Notice that this is enough since $V = \emptyset$ means that $\Sigma_K^{**} = \Sigma^{**}$. This is obvious for $k = 1$. For the induction step, a tree T is in $M(V, j, k + 1)$ iff either $T \in M(V, j, k)$, or

- (i) subtrees of T are taken to state q_k , then
- (ii) perhaps to q_k many times, and finally
- (iii) to q_j from q_k without involving q_i for $i \geq k$.

As with Theorem 6.16, such considerations give the regular expression α corresponding to

$$M(V, j, k) \cup M(V, k, k + 1) \cdot_{q_k} M(V \cup \{q_k\}, k, k + 1)^{*} \cdot_{q_k} M(V, j, k).$$

It is easy to argue that $L = L(\alpha)$ (Exercise 8).

The converse is straightforward and works by building a nondeterministic finite tree automaton corresponding to a regular expression α by induction on complexity of α (Exercise 9).

Finally, we turn to an analog of the Myhill–Nerode Theorem.

Definition 6.41 ($\sim_L$ for Trees) Let $x \notin \Sigma$ be a new symbol. For labeled trees T_1, T_2 , we will say that $T_1 \sim_L T_2$ iff for all trees $T \in \Sigma_{\{x\}}^{**}$ which have *exactly* one leaf labeled x (and all other nodes in Σ^{**}), we have

$$T_1 \cdot_x T \in L \text{ iff } T_2 \cdot_x T \in L.$$

[That is, for all trees T , if we concatenate the root of T_1 to the leaf of T labeled x , the result is in L iff the same holds for T_2 . The analogy with the classical $\sim_L$ is obvious.]

In a similar fashion, we may define an equivalence relation $\equiv$ to be a right congruence over Σ^{**} iff for all $T_1, T_2 \in \Sigma^{**}$, $T_1 \equiv T_2$ iff for all $T \in \Sigma_{\{x\}}^{**}$ with exactly one leaf labeled x ,

$$T_1 \cdot_x T \equiv T_2 \cdot_x T.$$

Theorem 6.42 (The Myhill–Nerode Theorem for Tree Automata) *The following are equivalent:*

- (i) L is finite state over Σ^{**} .
- (ii) L is a union of equivalence classes of some right congruence R of finite index over Σ^{**} .
- (iii) $\sim_L$ has finite index over Σ^{**} .

Proof. (i) $\rightarrow$ (ii) and (ii) $\rightarrow$ (iii) are virtually identical to the arguments for standard languages. For instance, consider (i) $\rightarrow$ (ii). Let M be finite state accepting L . As with Theorem 6.21, we declare that two trees T_1 and T_2 are R -equivalent iff M takes them to the same state (i.e., from q_0 on all the leaves). Then R is a right congruence and has finite index as M has a finite number of states. Hence, (i) $\rightarrow$ (ii).

We need a little work for (iii) $\rightarrow$ (i) since δ is now more complex. Suppose that L is the union of equivalence classes of $\sim_L$ which has finite index. Again we can let the states of the machine M we construct be the congruence classes of $\sim_L$. First, we observe that for any trees $T_1 \sim_L T_2$, and $a \in \Sigma$, for $d < f$, the fanin bound of Σ^{**} , and trees $Q_1, Q_2, \dots, Q_d \in \Sigma^{**}$, we have

$$T_1 \odot_a Q_1 \odot_a \dots \odot_a Q_d \sim_L T_2 \odot_a Q_1 \odot_a \dots \odot_a Q_d.$$

This follows from the fact that $T_1 \odot_a B = T_1 \cdot_x x \odot_a B$ for any $B \in \Sigma^{**}$. Replacing the Q_i one at a time, the observation generalizes to establish the following fact. For any trees $T_1, \dots, T_m$ and $T'_1, \dots, T'_m$ with $m \leq f$, if for all i , $T_i \sim_L T'_i$, then for all $a \in \Sigma$

$$T_1 \odot_a \dots \odot_a T_m \sim_L T'_1 \odot_a \dots \odot_a T'_m.$$

Now define

$$\delta([T_1], \dots, [T_m], a) = [T_1 \odot_a \dots \odot_a T_m].$$

From the fact above, it follows that if we specify δ in this way, then δ is well

defined. The accept states are again $\{[T] : T \in L\}$, and again, as in Theorem 6.21, the initial state is $[\lambda]$.

Notice that again we will get an implicit minimization algorithm from the Myhill–Nerode Theorem for Trees, and a Method of Test Sets (Exercises 10 and 11).

Historical Notes

Tree automata were introduced by many authors. The works of Doner [160], Thatcher and Wright [441], Mezei and Wright [351] are some of the relevant papers, but we refer the reader to Gecseg and Steinby [242, pages 121–124], Kozen [305], and Steinby [434] for more detailed discussions of the evolution of this subject.

Exercises 6.1

- 6.1.1. (i) Define the notion of a root-to-leaf tree automaton.
 (ii) Prove that there are languages accepted by root-to-leaf nondeterministic automata that are not accepted by any root-to-leaf deterministic automaton.
- 6.1.2. Prove that any finite state $L \subseteq \Sigma^{**}$ can be accepted in linear time.
- 6.1.3. Formulate the notion of a (leaf to root) *nondeterministic finite tree automaton*. Carefully verify that the details of the analog of Rabin and Scott's theorem (Theorem 6.6) and the Thompson construction go through.
- 6.1.4. Formulate the notion of a *nondeterministic finite tree automaton with λ moves*, and check that the analog of Theorem 6.10 holds (i.e., prove that λ moves give no additional computational power).
- 6.1.5. Construct a normalized tree grammar corresponding to the grammar of Example 6.35.
- 6.1.6. Define a notion of *regular grammar* for strings, and prove that a language $L \subseteq \Sigma^*$ is finite state iff L is accepted by a regular grammar.
- 6.1.7. Complete the details of the proof of Theorem 6.37. (That is, complete the proof that $A(G)$ accepts $L(G)$ for a tree grammar G ; and prove that the grammar constructed from G accepts $L(M)$.)
- 6.1.8. Carefully argue that α corresponding to the expression $M(V, j, k) \cup M(V, k, k+1) \cdot_{q_k} M(V \cup \{q_k\}, k, k+1)^* \cdot_{q_k} M(V, j, k)$ of Theorem 6.40 accepts $M(V, j, k+1)$.
- 6.1.9. Prove that if α is a regular expression over Σ^{**} , then there is an automaton M accepting $L(\alpha)$.
 (Hint: Model the proof on Theorem 6.16, where the corresponding result is proven for Σ^* by the inductive nondeterministic definition of M .)

- 6.1.10.** Formulate an algorithm based on the proof of the Myhill–Nerode Theorem for Trees, which minimizes the number of states of an automaton accepting a finite-state language L over Σ^{**} . (Refer to Algorithm 6.30.)
- 6.1.11.** Formulate a Method of Test Sets for Σ^{**} . (Refer to Observation 6.26.)

6.2 Treewidth

One of the major outgrowths from the beautiful work of Robertson and Seymour [397], [396] on graph minors and immersions was the exploration of a new parameter which measures in a precisely defined sense how “tree-like” a graph is. The fundamental idea is that we should be able to lift many results from trees to graphs that are “tree-like.” The extent to which Robertson and Seymour’s intuition has been vindicated is quite astonishing.⁴

In the present section we will look at a new parameter (treewidth) and associated parameters such as pathwidth. Later we will look at the uses of bounded treewidth for algorithm design. We remark in passing that the notion of treewidth seems completely intertwined with computational issues as we see with Courcelle’s and Seese’s theorems later and as can be seen by, e.g. Thorup [447].

The best known definition of treewidth comes from *partial k -trees*.

Definition 6.43 (k -Tree)

- (a) The class of k -trees is the smallest class obeying (i) and (ii) below.
 - (i) K_{k+1} , the complete graph on $k + 1$ vertices, is a k -tree.
 - (ii) If G is a k -tree and H is a subgraph of G isomorphic to K_k , then the graph G' constructed from G by first adding a new vertex v to G and then adding edges to make $H \cup \{v\}$ a copy of K_{k+1} , is a k -tree.
- (b) If G_1 is a subgraph of a k -tree, then G_1 is called a *partial k -tree*.

Example 6.44 In Figure 6.15, we give examples of 2- and 3- trees.

Definition 6.43 leads us to a very important concept.

Definition 6.45 (Treewidth) We say that a graph G has *treewidth* k if k is the minimum positive integer such that G is a partial k tree.

For algorithmic purposes, Definition 6.45 is not the most useful. An equivalent characterization of treewidth is provided by the following.

⁴In fact, equivalent notions to tree decomposition have been subsequently, and apparently independently discovered in other contexts such as the artificial intelligence literature. For instance, see Exercise 1.

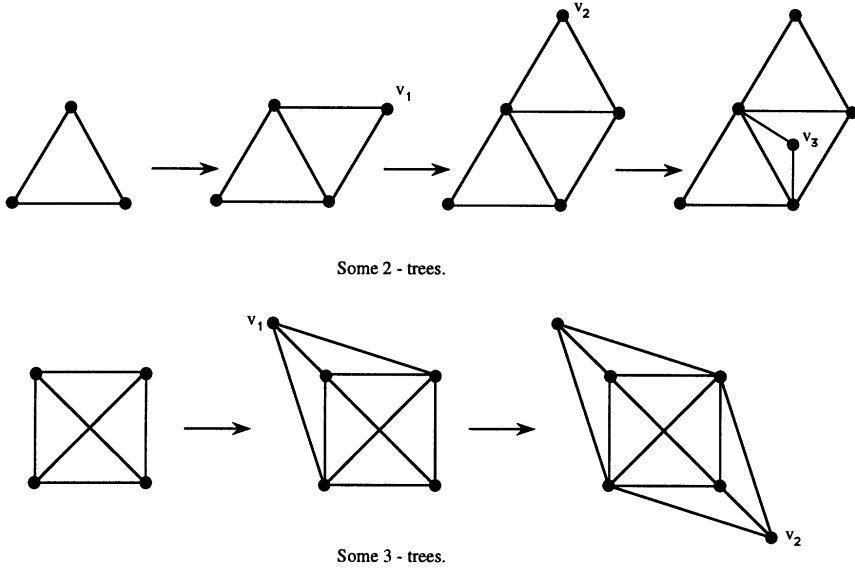


FIGURE 6.15. Examples of 2- and 3-Trees

Definition 6.46 (Tree Decomposition)

- (a) A *tree-decomposition* of a graph $G = (V, E)$ is a tree $\mathcal{T}$ together with a collection of subsets T_x (called *bags*) of V labeled by the vertices x of $\mathcal{T}$ such that $\cup_{x \in \mathcal{T}} T_x = V$ and (1) and (2) below hold:
- (1) For every edge uv of G , there is some x such that $\{u, v\} \subseteq T_x$.
 - (2) (Interpolation Property) If y is a vertex on the unique path in $\mathcal{T}$ from x to z , then $T_x \cap T_z \subseteq T_y$.
- (b) The *width* of a tree decomposition is the maximum value of $|T_x| - 1$ taken over all the vertices x of the tree $\mathcal{T}$ of the decomposition.

Before continuing our analysis of treewidth, we remark that there are a number of important variations of the notion. (See, e.g., Bodlaender and Engelfriet [77]). One of these is to tree decompositions what regular languages are to regular tree languages.

Definition 6.47 (Path Decomposition, Pathwidth) If the tree $\mathcal{T}$ of a tree decomposition is a path, then we say that the tree decomposition is a *path decomposition*, and use *pathwidth* in place of treewidth.

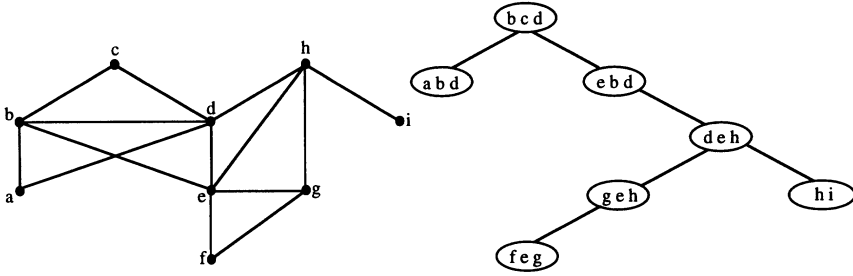


FIGURE 6.16. Example of Tree Decomposition of Width 2

We look further at pathwidth in the Exercises.

Example 6.48 Figure 6.16 gives an example of a tree decomposition of width 2.

Some feel for the notion may be obtained from the following easy observation. We will use this lemma implicitly henceforth.

Lemma 6.49 Let $\{T_x : x \in \mathcal{T}\}$ be a tree decomposition of G .

- (i) Let $x \in V(G)$. Then the collection of $y \in \mathcal{T}$ with $x \in T_y$ forms a subtree of $\mathcal{T}$.
- (ii) Suppose that C is a clique of G . Then there is some $x \in \mathcal{T}$ with $C \subseteq T_x$.

Proof. (i) This follows by property (2) of Definition 6.46. To see (ii), for each pair of vertices x, y in C , there is some T_z with $x, y \in T_z$. By (i), the collection of z with T_z containing x form a subtree $\mathcal{T}_x$ of $\mathcal{T}$. Now let $\{x_1, \dots, x_n\}$ list C . Each x_i thus generates a subtree $\mathcal{T}_i$ of $\mathcal{T}$. The intersection $\cap \mathcal{T}_i$ is nonempty by (i) and property (2), and hence there is some z with $C \subseteq T_z$.

The following basic result demonstrates that the two notions of width are identical.

Theorem 6.50 (Arnborg) A graph has treewidth k iff G has a tree decomposition of width k .

Proof. Let G be a graph with a tree decomposition $\{T_x : x \in \mathcal{T}\}$ of width k . Thus, $\max_{x \in \mathcal{T}} |T_x| = k + 1$. We prove that G is a partial k -tree, and, in fact, G is a subgraph of a k -tree $K(G)$ in which every $\leq k$ element subset of a bag T_x is part of a k -clique. We use induction on trees $\mathcal{T}$. Let x be a leaf of $\mathcal{T}$. Let $\mathcal{T}'$ be the result of removing x from $\mathcal{T}$, and let G' be a graph corresponding to $\mathcal{T}'$. By hypothesis, G' is a subgraph of a k -tree $K(G')$ with the desired properties. Let y be attached to x in $\mathcal{T}$. Let $T_x \cap T_y = T'_x$ and $T_x - T'_x = T''_x$. If $T''_x = \emptyset$, then $T_x \subseteq T_y$ and hence $G = G'$. So suppose that $T''_x \neq \emptyset$. By property (2) of Definition 6.46, for all $z \neq x$, $T''_x \cap T_z = \emptyset$. It follows that T'_x is a proper subset of T_x , and hence as the width is k , T'_x is a subset of at most k elements of T_y . By hypothesis, this means that T'_x is a subset of part of a k -clique C of $K(G')$. The idea is now to simply add T''_x one node at a time. We use the following algorithm.

Algorithm 6.51

- (1) Initially, we let $K = K(G')$. For $T_x'' \neq \emptyset$ we repeat the following steps 2–5:
- (2) Pick a vertex v of T_x'' .
- (3) Add v to K and also add all edges between vertices of C and v .
- (4) Let $T_x'' \leftarrow T_x'' - \{v\}$.
- (5) If there is any vertex $c \in C - T_x$, let $C \leftarrow [C - \{c\}] \cup \{v\}$.

By induction on the algorithm, one can see that after each iteration, C is a k -clique of K . Furthermore, each step emulates the definition of a k -tree, and hence as $K(G')$ is a k -tree, so too is K_{fin} , the result of Algorithm 6.51 applied to $K(G')$. By induction upon the algorithm, we also see that all of T_x'' is added together with all possible edges that can exist between vertices in $T_x = T_x' \cup T_x''$. Thus, K_{fin} contains G and has the desired properties.

Conversely, suppose that G is a partial k -tree. Since it suffices to consider k -trees, we will, in fact, suppose that G is a k -tree. We prove by induction on $|V(G)|$ that G will have a tree decomposition of width k . We use the stronger hypothesis that if G' is a k -tree with less than $|V(G)|$ vertices, then G' has a tree decomposition $\{T_x : x \in \mathcal{T}\}$ where $|T_x| \leq k + 1$ for all $x \in \mathcal{T}$, and each k -clique of G' occurs in some T_x . The result is clear if $G = K_{k+1}$. So suppose that G has more than $k + 1$ nodes and let v be the last node added in the formation of G (that is, v is a *simplicial* node). Let $N(v)$ be the neighbors of v in G . Let G' be the result of the removal of v and the k edges vu with $u \in N(v)$. Then, we can apply the induction hypothesis to G' and obtain a tree decomposition $\{T'_x : x \in \mathcal{T}'\}$ of G' with the desired properties. Now, $N(v)$ is a k -clique and hence, by hypothesis, there is some $y \in \mathcal{T}'$ with $N(v) \subseteq T'_y$. Create a new tree $\mathcal{T}$ from $\mathcal{T}'$ by attaching a new leaf $\ell = \ell(v)$ at y . Let $T_\ell = N(v) \cup \{v\}$. By hypothesis, $\mathcal{T}$ will be the relevant tree decomposition of G .

Henceforth, we will not distinguish between the two notions of treewidth. Treewidth is important from an algorithmic point of view because many problems which are intractable for the class of *all* graphs become tractable for classes of graphs with bounded treewidth.

Definition 6.52 (Bounded Treewidth) We say that a class $\mathcal{C}$ of graphs has *bounded treewidth* if there is some k such that for all $G \in \mathcal{C}$, G has treewidth $\leq k$.

Historically, authors often tried to get around classical intractability by looking at the problems at hand on special classes of graphs. Authors often discovered that intractable problems became tractable if the problems were restricted to, say, “outerplanar” graphs. Such restriction is not purely an academic exercise since, in many practical situations, the graphs that arise do not, in fact, demonstrate the full pathology of the class of all graphs. (Consider, for instance, **TRAVELLING SALESMAN** concerned with a given city. The graph will almost certainly be of small maximum clique size.) The table (from Van Leeuwen [457]) below lists some families of graphs that have been studied and have bounded treewidth.

Families of Graphs	Bound on Treewidth
Trees	1
Almost Trees (k)	$k + 1$
Partial k -trees	k
Bandwidth k	k
Cutwidth k	k
Planar of Radius k	$3k$
Series Parallel	2
Outerplanar	2
Halin	3
k -Outerplanar	$3k - 1$
Chordal with Maximum Clique Size k	$k - 1$
Undirected Path with Maximum Clique Size k	$k - 1$
Directed Path with Maximum Clique Size k	$k - 1$
Interval with Maximum Clique Size k	$k - 1$
Proper Interval with Maximum Clique Size k	$k - 1$
Circular Arc with Maximum Clique Size k	$2k - 1$
Proper Circular Arc with Maximum Clique Size k	$2k - 2$

We remark that we present the above table only for background information and hence it is not really important if the reader is unfamiliar with some of the graph families above. The reader is invited to verify some of these bounds in the Exercises. We consider the cases of **CUTWIDTH** and **BANDWIDTH** in some detail in Section 6.4.

What is important, however, is the fact that treewidth subsumes many of these properties in the sense that a demonstration of tractability for graphs of bounded treewidth will demonstrate tractability for all these families. *Furthermore, and perhaps even more importantly, as we see in the next section, the notion of tree decomposition allows one to give general automata-theoretic methods for demonstrating tractability of a wide class of properties for graphs of bounded treewidth.*

These automata-theoretic methods will be used to replace the ad hoc combinatorial methods previously employed by many authors and which usually entailed heroic case analysis. The quintessential example of this phenomenon will be Courcelle's theorem (Theorem 6.5), which demonstrates for graphs of bounded treewidth, a linear time algorithm for recognition of any property of graphs that can be described in a powerful fragment of second-order arithmetic (e.g., **HAMILTONICITY**). All of these arguments are based on the fact that the following problem is fixed parameter tractable.

TREewidth

Input: A graph G .
Parameter: A positive integer k .
Question: Does G have treewidth k ?

Theorem 6.53 (Bodlaender's Theorem, Bodlaender [71]) **TREewidth** is strongly FPT. In fact, for each k , there is a linear time algorithm for recognition of graphs of treewidth k . Furthermore, given a graph G of treewidth k , the algorithm will produce a tree decomposition of G of width k in linear time.

Bodlaender's theorem is proven in Bodlaender [71], via Bodlaender and Kloks [84, 86]. Modulo this latter result, we give a complete proof of Bodlaender's theorem in the next section.

Exercises 6.2

6.2.1. (W. Merkle) An ordering $I = \langle I_1, \dots, I_n \rangle$ of a set of subsets of V is called an *acyclic hypergraph* iff for all $j \leq n$, there is a $q < j$ such that $I_j \cap (\cup_{i < j} I_i) \subseteq I_q$. (This is also called the *running intersection property*, and we refer to Lauritzen and Spiegelhalter [325].) Given a finite graph, we say that an acyclic hypergraph I is an acyclic hypergraph decomposition of G iff $V = V(G)$ and for each edge e of G , there is some i such the vertices of e are in I_i . Prove that G has a tree decomposition of width w iff G has an acyclic hypergraph decomposition I such that for all $i \leq n$, $|I_i| \leq w$.

(Hint: Prove that one can construct the tree decomposition from I to have the I_i as the actual bags, and conversely.)

6.2.2. Apply the algorithm implicit in Theorem 6.50 to embed the graph described by the tree decomposition of Figure 6.17 into a 3-tree.

6.2.3. Recall the definition of pathwidth, Definition 6.47. Give a path decomposition of width 2 of the graph of Figure 6.16.

6.2.4. Prove that if a graph G has a size k vertex cover, then it must also have treewidth $\leq k$.

6.2.5. A *Halin graph* is constructed as follows: Take a tree without vertices of degree 2. Then, embed it in the plane, and add a cycle through its leaves, in order (i.e., the result remains planar). Prove that a Halin graph must have treewidth 3.

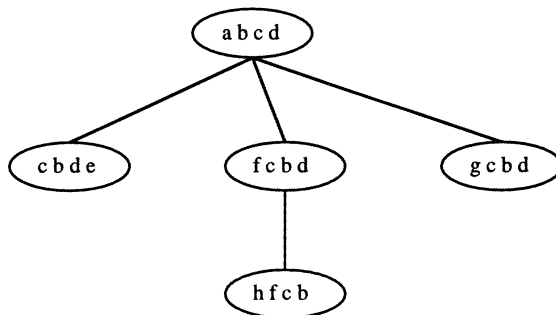


FIGURE 6.17. Figure for Exercise 2

- 6.2.6.** A graph is called *outerplanar* iff there exists a plane embedding G of the graph such that G has a cycle C with the property that every edge of G is either a member of C or a chord of C . Prove that if G is outerplanar, then it has treewidth 2.
- 6.2.7.** A graph G is called *series parallel* if it can be generated by the following operations. Take a loop or an edge and interpolate a vertex v . Take an edge e and create a parallel edge (i.e., with the same vertices) e' . Prove that if G is series parallel, then G has treewidth 2.
- 6.2.8.** A graph G is said to be *almost tree* (k) iff it has at most k cycles. Prove that if G is almost tree (k), then G has treewidth $\leq k + 1$.

6.3 Bodlaender's Theorem

In this section, we prove Bodlaender's theorem. Our proof that there is a linear time *FPT* algorithm which

- (i) determines if a graph G has treewidth at most k and
 - (ii) produces a tree decomposition of width at most k if the answer to (i) is "yes,"
- relies upon the following theorem.

Theorem 6.54

- (i) (Lagergren and Arnborg [324], Abrahamson and Fellows [6], Bodlaender and Kloks, [84, 86]) *There is an $O(|G|)$ FPT algorithm which has parameter k and p with $k \leq p$ which, given a width p tree decomposition of G determines if G has treewidth k .*
- (ii) (Bodlaender and Kloks, [84, 86]) *Furthermore, if the treewidth of G is k , there is a linear time algorithm which will output a width p k tree decomposition for G .*

There are several different approaches to proving Theorem 6.54(i). We can deduce it from the material of this book as follows. By Robertson and Seymour [391], as we see in Chapter 7, there is a finite set of graphs $H_1, \dots, H_q$ such that a graph G has treewidth at most k iff H_i is not a minor of G for $i \in \{1, \dots, q\}$.⁵ Actually, since we know a bound on the treewidth of the H_i , it is possible to calculate the set $H_1, \dots, H_q$, as we see in Chapter 7.8. In any case, as we see in Section 6.5, the minor relation is definable in monadic second-order logic (see Section 6.5), and hence by Courcelle's theorem (Theorem 6.85), we can construct an automaton which runs in linear time and accepts only parse trees of width k from

⁵We say a graph H is a minor of a graph G if H can be obtained from G by deletion and contractions of edges. We refer to Definition 7.9.

the input of (parse trees in linear time derived from) width p tree decompositions.⁶ The algorithm of Bodlaender and Kloks, which produces the tree decomposition if there is one, is concrete and elementary, but too involved to easily fit in the book. (We will give some further notes on the development of Bodlaender's theorem in the Historical Notes.)

We turn to the proof of Bodlaender's theorem. The main idea of the proof of Bodlaender's theorem is that given G , we first partition the vertices into vertices of *low* and of *high* (to be defined later) degree. There are then two cases.

Case 1. There are many vertices of low degree adjacent to each other. In this case, it is shown that a maximal matching contains *many* edges. One then *contracts*⁷ all edges in a maximal matching and generates a new derived graph G' . Now, recursively get a treewidth k decomposition for G' , or get a certificate that the treewidth of G' and therefore G exceeds k . In the former case it is possible to generate a treewidth $2k + 1$ decomposition for G from that of G' using the Bodlaender-Kloks Algorithm.

Case 2. There are only few vertices of low degree adjacent to another vertex of low degree. In this case, we will *enrich* the graph by adding certain new edges to form a new graph G' of the same treewidth as G (if it is t). This new graph G' will have many vertices which are *simplicial* in the sense that their neighbors form cliques and can be removed to form G'' . Given a tree decomposition of G'' , one can easily get one for G .

We remark that it will be easy to see that the algorithm runs in linear time. We now turn to some details.

Lemma 6.55 (Folklore) *Suppose that G has treewidth k . Let $\{X_i : i \in T\}$ be a tree decomposition for G . Then the following hold:*

- (i) $|E| < k|V| - \frac{1}{2}k(k+1)$.
- (ii) If $W \subseteq V$ is a clique, for some $i \in T$, $W \subseteq X_i$.
- (iii) If W_1 and W_2 induce a complete bipartite subgraph of G , then for some $i \in T$, $W_1 \subseteq X_i$.
- (iv) Suppose that $v, w \in X_i$ and $v \neq w$. Then $\{X_i : i \in T\}$ is a tree decomposition for $G + vw$.
- (v) Suppose that x and y are vertices of G with at least $k + 1$ common neighbors. Then the treewidth of $G + xy$ is k and $\{X_i : i \in T\}$ is also a tree decomposition for $G + xy$.

⁶The reader might notice an apparent circularity in all of this. Is not Bodlaender's theorem actually *used* in the proof of Courcelle's theorem? The proof of Courcelle's theorem actually demonstrates that, for a fixed p , monadic second order statements have linear time recognition from a given treewidth p decomposition. This is independent of Bodlaender's theorem, which can now be used to add that "hence monadic second order statements have linear time recognition algorithms."

⁷Recall that a contraction of an edge uv in a graph G is the operation of identifying u with v . We refer to Figure 7.2.

Proof. See Exercise 1.

A tree decomposition $\{X_i : i \in \mathcal{T}\}$ of G is called *smooth* iff for all $i \in \mathcal{T}$, X_i has $k + 1$ elements and for all nodes p, q of $\mathcal{T}$ with p the child of q , $|X_p \cap X_q| = k$. Part (i) of the following lemma is well known and has a proof similar to the proof of Theorem 6.72. (See Exercise 2.)

Lemma 6.56

- (i) Any tree decomposition can be converted into a smooth one for the same graph by applying the following operations to nodes p, q in $\mathcal{T}$ with p the child of q or q the child of p .
- If $X_p \subseteq X_q$, contract pq let the bag be X_q .
 - If $X_p \not\subseteq X_q$, but $|X_q| < k + 1$, take any vertex from $X_p - X_q$ and add it to X_q .
 - If $|X_p| = |X_q| = k + 1$ and $|X_p - X_q| > 1$, interpolate new bags between X_p and X_q .
- (ii) (Bodlaender) If $\{X_i : i \in \mathcal{T}\}$ is a smooth tree decomposition of G of width k , then $\mathcal{T}$ has exactly $|V(G)| - k$ nodes.

Proof. See Exercise 2.

Algorithm Parameters. Choose rationals $0 < c_1, c_2 < 1$ such that

$$c_2 = \frac{1}{4k^2 + 12k + 16} - \frac{c_1 k^2 (k + 1)}{2} > 0.$$

Also, define

$$d = \max \left\{ k^2 + 4k + 4, \left\lceil \left(\frac{2k}{c_1} \right) \right\rceil \right\}.$$

Definition 6.57 (High, Low and Friendly, Bodlaender [71]) If the degree of a vertex v exceeds d , then we say that v has *high degree*; if it is less than or equal to d , it has *low degree*; and finally if v has both low degree and is also adjacent to one of low degree, we say that it is *friendly*.

Recall that a *maximal matching* in a graph G is a collection of disjoint edges to which no further disjoint edges can be added. The greedy algorithm gives a simple maximal matching algorithm that runs in linear time.

Lemma 6.58

- (i) If G has treewidth k , then it contains fewer than $c_1 |V|$ high-degree vertices.
- (ii) If there are f friendly vertices in G , then any maximal matching contains at least $\frac{f}{2d}$ edges.

Proof. (i) is clear. (ii) Let M be a maximal matching. Note that any friendly vertex must either be an endpoint of an edge of M or adjacent to an endpoint

of an edge of M . Now, associated with each edge e of M there can be at most $2d$ friendly vertices which are either endpoints of e or adjacent to a low-degree friendly endpoint of e . Hence, $M > \frac{f}{2d}$.

Definition 6.59 (M -Derived Graph)

- (i) Let M be a maximal matching of G . The M -derived graph is the one obtained by contracting all the edges of M .
- (ii) The M -derivation function is defined via $g_M(v) = v$ if v is not a vertex of an edge of M , and $g_M(v) = g_M(w)$ is the vertex v' resulting from the contraction of vw of M .

We need one further lemma. Its proof is left to the reader (Exercise 3).

Lemma 6.60

- (i) If $\{X_i : i \in T\}$ is a width k tree decomposition of G' , where G' denotes the M -derived graph of G , then $\{Y_i : i \in T\}$ defined via $Y_i = \{v \in V : g_M(v) \in X_i\}$ is a width $\leq 2k + 1$ tree decomposition of G .
- (ii) The treewidth of G' is less than or equal to that of G .

For a graph G , we define the *improved graph* $\widehat{G}$ of G to be that obtained by adding an edge uv for all pairs u, v of V with at least $k + 1$ common neighbors in G . Now we get to the key theorem. We need a preliminary definition.

Definition 6.61 ($\mathcal{T}$ -Simplicial Vertex)

- (i) We say that a vertex v of G is *simplicial in G* if its set of neighbors form a clique in G .
- (ii) A vertex w is called $\mathcal{T}$ -simplicial vertex if it is simplicial in the improved graph $\widehat{G}$ of G , it is of low degree and it is not friendly in G .

Lemma 6.62 Let $\{X'_i : i \in T'\}$ be a tree decomposition of the graph G' obtained by removing all $\mathcal{T}$ -simplicial vertices (and adjacent edges) from the improved graph $\widehat{G}$ of G . Then, for all $\mathcal{T}$ -simplicial vertices v , there is some node i_v such that the set of neighbors of v in G all lie in X'_{i_v} .

Proof. See Exercise 4.

The decisive lemma upon which Bodlaender's theorem is based is the following.

Theorem 6.63 (Bodlaender [71]) For every graph G , one of the following holds:

- (i) G contains at least $\frac{|V|}{4k^2 + 12k + 16}$ friendly vertices.
- (ii) The improved graph $\widehat{G}$ of G contains at least $c_2|V|$ $\mathcal{T}$ -simplicial vertices.
- (iii) The treewidth of G exceeds k .

Before we verify Theorem 6.63, we can now state Bodlaender's algorithm.

Algorithm 6.64 (Bodlaender's Algorithm) For graphs larger than some fixed constant (which is computable), the following algorithm determines if a graph G has treewidth $\leq k$ and, if so, produces a treewidth k tree decomposition for G .

Step 1. Check if $|E| \leq k|V| - \frac{1}{2}k(k+1)$. If the answer is *no*, output that the treewidth of G exceeds k [Lemma 6.55 (i)]. If the answer is *yes*, go to step 2.

Step 2. There are two cases depending on the number f of friendly vertices.

Case 1. $f \geq \frac{|V|}{4k^2+12K+16}$.

Action.

- (i) Find a maximal matching M in G .
- (ii) Compute the derived graph (G', E') by contracting edges of M .
- (iii) Recursively apply the algorithm to G' . If the treewidth of G' exceeds k , stop and output that the treewidth of G exceeds k [Lemma 6.60 (ii)]. Otherwise, the recursive call yields a treewidth k tree decomposition $\{X'_i : i \in T'\}$ for G' . Now apply Lemma 6.60(i) to get a treewidth $2k+1$ tree decomposition $\{Y_i : i \in T'\}$ of G .
- (iv) Now apply, for instance, the Bodlaender-Kloks Algorithm to the treewidth $2k+1$ tree decomposition of G to either return that G has treewidth exceeding k or returns a treewidth k tree decomposition of G .

Case 2. $f < \frac{|V|}{4k^2+12K+16}$.

Action.

- (i) Compute the improved graph $\widehat{G}$ of G . If there is a $\mathcal{T}$ -simplicial vertex of degree $\geq k+1$, then stop and output that the treewidth of G exceeds k since $\widehat{G}$ contains a $k+2$ -clique.
- (ii) Otherwise, put all $\mathcal{T}$ -simplicial vertices into some set S . Now, compute the graph G' obtained by deleting all $\mathcal{T}$ -simplicial vertices and adjacent edges from G .
- (iii) If $|S| < c_2|V|$, then stop and output that the treewidth of G exceeds k [Theorem 6.63 (ii)].
- (iv) Otherwise, recursively apply the algorithm to G' . If the treewidth of G' is greater than G , then the treewidth of G is greater than k , as G' is a subgraph of G . Otherwise, the algorithm will return a treewidth k tree decomposition $\{X'_i : i \in T'\}$ for G' .
- (v) In that case, for all $v \in S$, find an $i_v \in \mathcal{T}$ such that the neighbors $N_G(v)$ of v in G all lie in bag X'_{i_v} [Lemma 6.62]. We then add a new node j_v to $\mathcal{T}$ adjacent to i_v in $\mathcal{T}$ with $X_{j_v} = \{v\} \cup N_G(v)$.

The fact that the algorithm is correct is immediate by the results preceding it. Note that we either recursively apply the algorithm on a graph with $(1 - \frac{1}{2d(4k^2+12k+16)})|V|$ vertices or on a graph of $(1 - c_2)|V|$ vertices. Since both $1 - c_2$ and $1 - \frac{1}{2d(4k^2+12k+16)}$ are greater than or equal to 0 and less than 1, it follows that the algorithm must run in linear time, given that the called nonrecursive parts have linear time implementations, which they do.

It remains then to prove Theorem 6.63. We do this in the following four lemmas. We say that a vertex v is *localized* with respect to a tree decomposition $\{X_i : i \in T\}$ if it has low degree, it is not friendly, and for some bag X_i , all of the neighbors of v are in X_i .

Lemma 6.65 *Suppose that $\{X_i : i \in T\}$ is a smooth tree decomposition of G of width k . Then, (i) and (ii) below hold.*

- (i) *To each leaf i of T one can associate a vertex v_i which is either friendly or localized, and furthermore for all $j \neq i$, $v_i \notin X_j$.*
- (ii) *To each path P of the form $i_0, \dots, i_{k^2+3k+2}$ in T with i_j of degree 2 for $j \in \{0, \dots, k^2 + 3k + 2\}$, there is a vertex $v_P \in \bigcup_{j=0}^{k^2+3k+2} X_{i_j}$ such that v_P is either friendly or localized, and for all q not on the path P , $v_P \notin X_q$.*

Proof. 1. Let i be a leaf of T , and j a neighbor. Let v be the unique vertex in $X_i - X_j$. Note that the only neighbors of v are in X_i so it has low degree. If all the neighbors of v have high degree, then it is localized. Otherwise it is friendly.

2. $|\bigcup_{j=0}^{k^2+3k+2} X_{i_j}| = k^2 + 4k + 4$ (by smoothness) and hence $\leq d$. Hence, all the vertices in $(\bigcup_{j=0}^{k^2+3k+2} X_{i_j}) - (X_{i_0} \cup X_{i_{k^2+3k+2}})$ are of low degree. Suppose that all of them are unfriendly, and hence only adjacent to vertices of high-degree in $X_{i_0} \cup X_{i_{k^2+3k+2}}$. Let $w_1, \dots, w_r$ list the high degree vertices in X_{i_0} . For $s \in \{1, \dots, r\}$, we know by smoothness that there is a number n_s such that $w_s \in X_{i_0}, \dots, X_{i_{n_s}}$. We may suppose that $n_1 \leq n_2 \leq \dots \leq n_r$. If some low-degree vertex v belongs to exactly one X_{i_j} on the path P it must be of low-degree and hence localized. If some low-degree vertex belongs to only bags from $X_{i_{n_j+1}}, \dots, X_{i_{n_{j+1}}}$, then it is localized since all of its neighbors must belong to $X_{i_{n_j+1}}$. All vertices in $\bigcup_{j=0}^{k^2+3k+2} X_{i_j}$ not of these two types must belong to at least one of the distinguished bags

$$X_{i_0}, X_{i_{n_1}}, \dots, X_{i_{n_r}}, X_{i_{k^2+3k+2}}.$$

This is only a total of $k^2 + 4k + 3$ vertices, and the result follows.

Now we demonstrate that we can nontrivially apply the lemma. We define a *leaf-path collection* to be a set of leaves of T plus a set of paths of T of length at least $k^2 + 3k + 4$ all the nodes of which except the endpoints have degree 2 and do not belong to any other path in the collection.

Lemma 6.66

- (i) *If T is any tree with n nodes then it must have a leaf path collection of size $\geq \frac{n}{2k^2+6k+8}$ (i.e., contains that many leaves and paths).*

- (ii) Consequently, if $\{X_i : i \in \mathcal{T}\}$ is a smooth tree decomposition of width k , then G has at least $\frac{|V|}{2k^2+6k+8} - 1$ vertices that are localized or friendly.

Proof. (i) This is a calculation. Let n_3 be the number of nodes in $\mathcal{T}$ of degree ≥ 3 , n_1 the number of leaves, and n_2 the number of nodes of degree 2. We note that all nodes of degree 2 belong to $< n_1 + n_3$ connected components of the forest obtained from $\mathcal{T}$ by removing all nodes of degree ≥ 3 . Each such component has at most $k^2 + 3k + 3$ nodes if it does not contribute to some leaf-path collection. Therefore, there are fewer than $(n_1 + n_3)(k^2 + 3k + 3)$ nodes of degree 2 not on a path in the leaf-path collection $\mathcal{C}$ of maximum size. Therefore, $\mathcal{C}$ has at least

$$\frac{n_2 - (n_1 + n_3)(k^2 + 3k + 3)}{k^2 + 3k + 4}$$

paths. Therefore, $\mathcal{C}$ has size at least

$$\frac{n_2 - (n_1 + n_3)(k^2 + 3k + 3)}{k^2 + 3k + 4} + n_1 \geq \frac{\frac{1}{2}n}{k^2 + 3k + 4},$$

giving part (i). For (ii), by Lemma 6.56(ii), $\mathcal{T}$ has $|V| - k$ many nodes and, hence, an application of (i) and Lemma 6.65 yields the result.

We say that a set of high-degree vertices $Y \subseteq V$ is *important* with respect to the decomposition $\{X_i : i \in \mathcal{T}\}$, if $Y \subseteq X_i$ for some $i \in \mathcal{T}$ and for any other set Y' of high-degree vertices with $Y' \subseteq X_j$ for some j , Y is not a proper subset of Y' .

Lemma 6.67 *Let $\{X_i : i \in \mathcal{T}\}$ be a decomposition of G of width k . The the number of important sets is at most the number of high-degree vertices of G .*

Proof. Let H be the set of high-degree vertices of G . Then, $\{X_i \cap H : i \in \mathcal{T}\}$ is a tree decomposition of the subgraph induced by H . Each important set Y is a set of the form $X_i \cap H$ not contained in any other $X_j \cap H$. Repeatedly contract the edges ii' with $X_i \cap H \subseteq X_{i'} \cap H$, forming a new node containing all the vertices of $X_{i'}$. Clearly, the resulting tree decomposition will have size at most H and will contain all the important sets of G .

Fix a tree decomposition $\{X_i : i \in \mathcal{T}\}$ of G . Since localized vertices are not friendly, there is a function f , called a *localized to important function*, which associates each localized vertex v with an important set Y such that $N_G(v) \subseteq Y$.

Lemma 6.68 *Let f be such a localized to important function for a smooth decomposition $\{X_i : i \in \mathcal{T}\}$ of width k for G . Then, for any important set Y , there are at most $\frac{1}{2}k^2(k+1)$ localized vertices in $f^{-1}(Y)$ which are not $\mathcal{T}$ -simplicial.*

Proof. To each non- $\mathcal{T}$ -simplicial localized vertex v , associate a pair of its neighbors that are nonadjacent in the improved graph. Notice that we cannot so associate more than k vertices with the same pair, lest the graph have $\geq k+1$ common low-degree vertices which would necessarily have an edge between them in the improved graph. By simply counting, we see that the number of $\mathcal{T}$ -simplicial vertices with $f(v) = Y$ is at most $\frac{1}{2}|Y|(|Y| - 1)$, which is $\leq \frac{1}{2}k^2(k+1)$.

Proof of Theorem 6.63, completed. To complete the proof of Theorem 6.63, suppose that G contains fewer than $\frac{|V|}{4k^2+12k+16}$ friendly vertices and that the treewidth of G is at most k . By Lemma 6.58(i), there are at most $c_1|V|$ high-degree vertices in G . Therefore, by Lemma 6.67, the number of important sets for any smooth tree decomposition $\{X_i : i \in \mathcal{T}\}$ of G of width k is $\leq c_1|V|$. Therefore, by Lemma 6.68, at most $\frac{1}{2}k^2(k+1)(c_1|V| - 1)$ localized vertices are not $\mathcal{T}$ -simplicial. Using Lemma 6.66(ii), we see that

$$\frac{|V|}{2k^2+6k+8} - 1 - \frac{|V|}{4k^2+12k+16} - \frac{1}{2}k^2(k+1)(c_1|V| - 1) \geq c_2|V|$$

vertices are $\mathcal{T}$ -simplicial, which establishes Theorem 6.63.

Historical Notes

The material of this section is drawn from Bodlaender [71]. The problem of deciding if a graph has treewidth k is NP -complete if k is allowed to vary, as discussed in Arnborg, Corneil and Proskurowski [32]. In Ref. [32], the authors also showed that there was an $O(n^{k+2})$ recognition algorithm for treewidth k graphs. Robertson and Seymour [396] gave the first FPT (it was $O(n^2)$) algorithm for the treewidth and recognition problem. Their algorithm was based upon the minor well-quasi-ordering theorem and thus highly nonconstructive, nonelementary and had huge constants. Algorithms for recognition of bounded treewidth graphs have been subsequently improved, approximated, and parallelized in the work of, for instance, Lagergren [322], Lagergren and Arnborg [324], Reed [382], Fellows and Langston [216], and Matousek and Thomas [345].

Linear time algorithms had been demonstrated for recognition of treewidth k graphs $k \leq 3$ (e.g., Matousek and Thomas [345]). For a general k , linear time recognition of treewidth k graphs from among treewidth $k \leq m$ graphs given by treewidth m decompositions is due independently to Lagergren and Arnborg [324], Abrahamson and Fellows [6] and Bodlaender and Kloks [84, 86] although it follows also by Courcelle's theorem as we noted earlier. Bodlaender and Kloks gave the explicit algorithm which allows us to actually derive the relevant tree decomposition needed for Bodlaender's theorem. Their algorithm and Bodlaender's Theorem are both impractically exponential in k , namely 2^{ck^3} where $c \approx 32$. It would be very interesting if this could be reduced to an exponential with exponent linear in k . Bodlaender's Theorem is found in Bodlaender [63, 71].

Exercises 6.3

6.3.1. Prove Lemma 6.55.

(Hint: For (i) consider a k -tree, for (ii) and (iii) use the interpolation property in the definition of tree decomposition, (iv) is easy, and for (v), argue that x and y must occur in some bag together since otherwise we could apply (iii) and then (ii) to get a $k+2$ -clique in a treewidth k graph.)

6.3.2. Prove Lemma 6.56. (Hint: For (ii), use induction on $|V(\mathcal{T})|$.)

6.3.3. Prove Lemma 6.60.

6.3.4. Prove lemma 6.62. (Hint: Use Lemma 6.55.)

6.4 Parse Trees for Graphs of Bounded Treewidth and an Analog of the Myhill–Nerode Theorem

The principal tools we use to connect the theory of tree automata with graphs of bounded treewidth are *parse trees*. The idea is that graphs of bounded treewidth can be generated by a set of simple operations, in the same way that one might parse a grammatical expression. The parse trees that we obtain will be used as the input for the relevant tree automata. In turn, the Myhill–Nerode Theorem for trees will be used to generate an analog for “ t -boundaried” graphs.

Definition 6.69 (t -Boundaried Graphs) A t -boundaried graph is a graph G with t distinguished vertices being given the labels $\{1, \dots, t\}$. The vertices possessing the labels $\{1, \dots, t\}$ are called the *boundary*, and we let $\partial(G)$ denote the set of boundary vertices of G .

Sometimes, such as in the proof of Courcelle’s theorem in Section 6.5, we will write a t -boundaried graph as $G = (V, E, B, f)$, where V and E are, as usual, the vertices and edges of G , and where $\partial(G) = B$ with f the labeling function: f is a bijection from $B \mapsto \{1, \dots, t\}$.

The basic operation we will perform on t -boundaried graphs will be gluing them together.

Definition 6.70 (Gluing by $\oplus$) Let G_1 , and G_2 be t -boundaried graphs. We denote by $G_1 \oplus G_2$ the t -boundaried graph obtained by taking the disjoint union of G_1 and G_2 and identifying each vertex of $\partial(G_1)$ with the vertex of $\partial(G_2)$ with the same label; that is, we glue them together on the boundaries.

Operators on the class of t -boundaried graphs allow us to generate graphs of bounded treewidth. Again, notice the persistent analogy with $\odot$ of the previous sections.

Definition 6.71 (t -Boundaried Composition Operator $\otimes$, Abrahamson and Fellows [7]) A n -ary t -boundaried composition operator is a t -boundaried graph $G_\otimes = \langle V, E \rangle$, together with n injective mappings $f_i : \{1, \dots, t\} \mapsto V$.

Suppose that $G_1, \dots, G_n$ are n t -boundaried graphs and that $\otimes$ is a composition operator according to Definition 6.71. Then, we will denote by

$$G_1 \otimes \dots \otimes G_n$$

the t -boundaried graph obtained by gluing each of the G_i to $G_\otimes$ as indicated by f_i , deleting G_i ’s labels afterward; that is, for each v in $\partial(G_i)$, if v has label ℓ ,

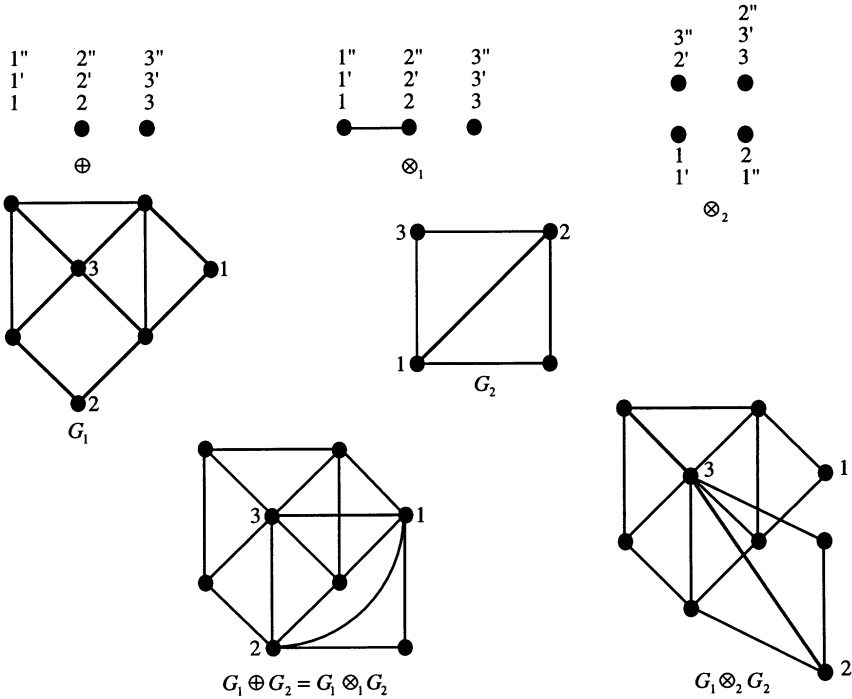


FIGURE 6.18. Composition Operators and Their Effects

then we will identify v with the vertex $f_i(\ell)$ of $G_\otimes$ and regard G_i as unlabeled once we have completed all such identifications. In Figure 6.18, we illustrate this process. The notation $i^{(n)}$ in this figure denotes the attachment point for the vertex labeled i in the n -th argument of $\otimes$. Thus, for instance, labeling a vertex v by 1, 2', 3'' indicates that we should attach to v the vertex labeled 2 of G_1 and the vertex labeled 3 of G_2 ; and furthermore v would finish up in the boundary of the composite graph with label 1. Notice that, as Figure 6.18 illustrates, $\oplus$ is a special example of a composition operator.

We will use special families of composition operators for parsing graphs of fixed treewidth. Before giving a general recipe for parsing treewidth k graphs, we will demonstrate the idea by examining treewidth 2. We will use the (composition) parsing operators from Figure 6.19. We need $\oplus$ and four other operators, $\emptyset$ a nullary operator that simply produces two vertices, together with two binary operators $\otimes_a$ and $\otimes_b$. (We do not worry about the empty graph, nor the one with one vertex.)

It is not difficult to establish that a graph G has treewidth 2 iff G can be parsed using the given parse operators (Exercise 1). We say that these operators form a *complete set* of parsing operators for treewidth 2 graphs. As we will now see, all treewidth t graphs can be parsed by a simple set of operators. For simplicity, we will prove that all treewidth t graphs can be parsed using a set of *binary*, *unary*, and *nullary* operators on the class of $t + 1$ boundaried graphs. In fact, it is possible

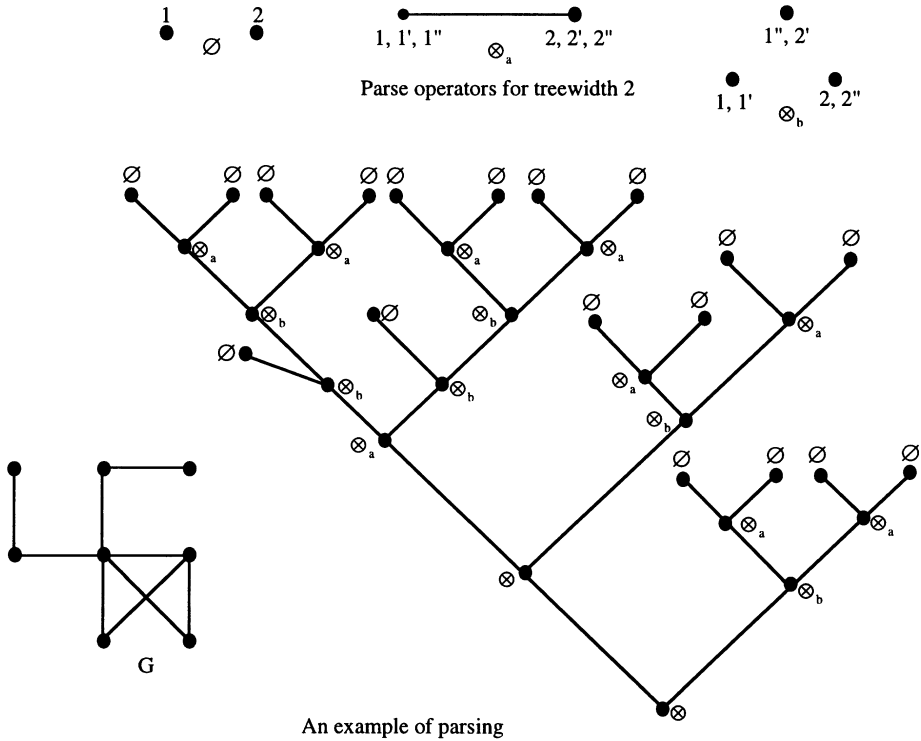


FIGURE 6.19. Parsing Graphs of Treewidth 2

to prove that all treewidth t graphs can be parsed by a finite set of operators on the class of t -boundaried graphs but these are more complicated (Exercise 3).

The collection of size $t + 1$ parsing operators consists of the following set of six operators:

$$\{\emptyset, \oplus, \gamma, i, u, e\}.$$

Here,

- $\emptyset$ is again the nullary operator which creates boundary vertices labeled $1, \dots, t + 1$.
- γ is unary and denotes a cyclic shift of the boundary [that is, γ moves label j to the vertex with label $j + 1 \pmod{t + 1}$].
- i is a unary operator and denotes an *interchange*: i simply puts label 1 on the vertex currently with label 2, and label 2 on the vertex with label 1.
- e is the unary operator that adds an edge between the vertices labeled 1 and 2.
- u is the unary operator that adds a new vertex and labels it 1. (We call this pushing.)

Notice that the only operators that add vertices are u and $\oplus$. The only operator that adds an edge is e . Notice that the combination of γ and i enables us to permute the boundary at will, since the symmetric group can be generated by a cycle and a transposition. We remark that the following result is implicit in Abrahamson and Fellows [6], but similar results have been noted by several authors such as Kloks [301].

Theorem 6.72 (Parsing Theorem)

- (a) *There is a linear time algorithm which converts a tree decomposition of width t into a parse tree using the size $t + 1$ parsing operators.⁸*
- (b) *Conversely, if G is any graph generated from a parse tree T using the size $t + 1$ parsing operators, then G has treewidth $\leq t$.*

Proof. Let G be a graph with a tree decomposition $\mathcal{T} = \{X_i : i \in T\}$ of width t . We need to construct a parse tree from $\mathcal{T}$. This is done in several steps.

Step 1. We first perform a preliminary normalization of $\mathcal{T}$.

- (i) First, we make the tree $\mathcal{T}$ binary. This is accomplished by duplicating the nodes at which the branch occurs and interpolating as children of the branch node. Thus, if a branch node η had fanout $m > 2$ we would duplicate the node η , $m - 2$ times, eliminating the branches one at a time. (See Figure 6.20.)
- (ii) Next, we make all the sets in the tree decomposition uniform of size $t + 1$ by adding new vertices to the node if necessary. This can be accomplished by using vertices from the node to a child to build up the number to $t + 1$. Notice that this action cannot affect the interpolation property. (See Figure 6.20.) (This is easily verified by induction.)
- (iii) Now, we ensure that every branch node has identical children. This property can be ensured by duplicating branch nodes below the branch node.
- (iv) Finally, we ensure that all sets differ by at most one. Thus we will ensure that if $\mathcal{T}' = \{X'_i : i \in T'\}$ is the normalized version of $\mathcal{T}$, then for all edges $uv \in T'$,

$$|X_u \cap X_v| \geq t.$$

This property can be ensured by adding interpolating nodes between X_i and X_j which differ by more than 1. (See Figure 6.20.)

Thus, henceforth, we will presuppose that $\mathcal{T}$ is normalized according to (i)-(iv) of step 1.

Step 2. Now, for each edge $e = uv$ of G , choose a vertex $v(e)$ of $\mathcal{T}$ where e is covered.

⁸It is important that the reader understand that this theorem only gives a parse tree for the graph *up to isomorphism*. This point is important in some of the work to follow.

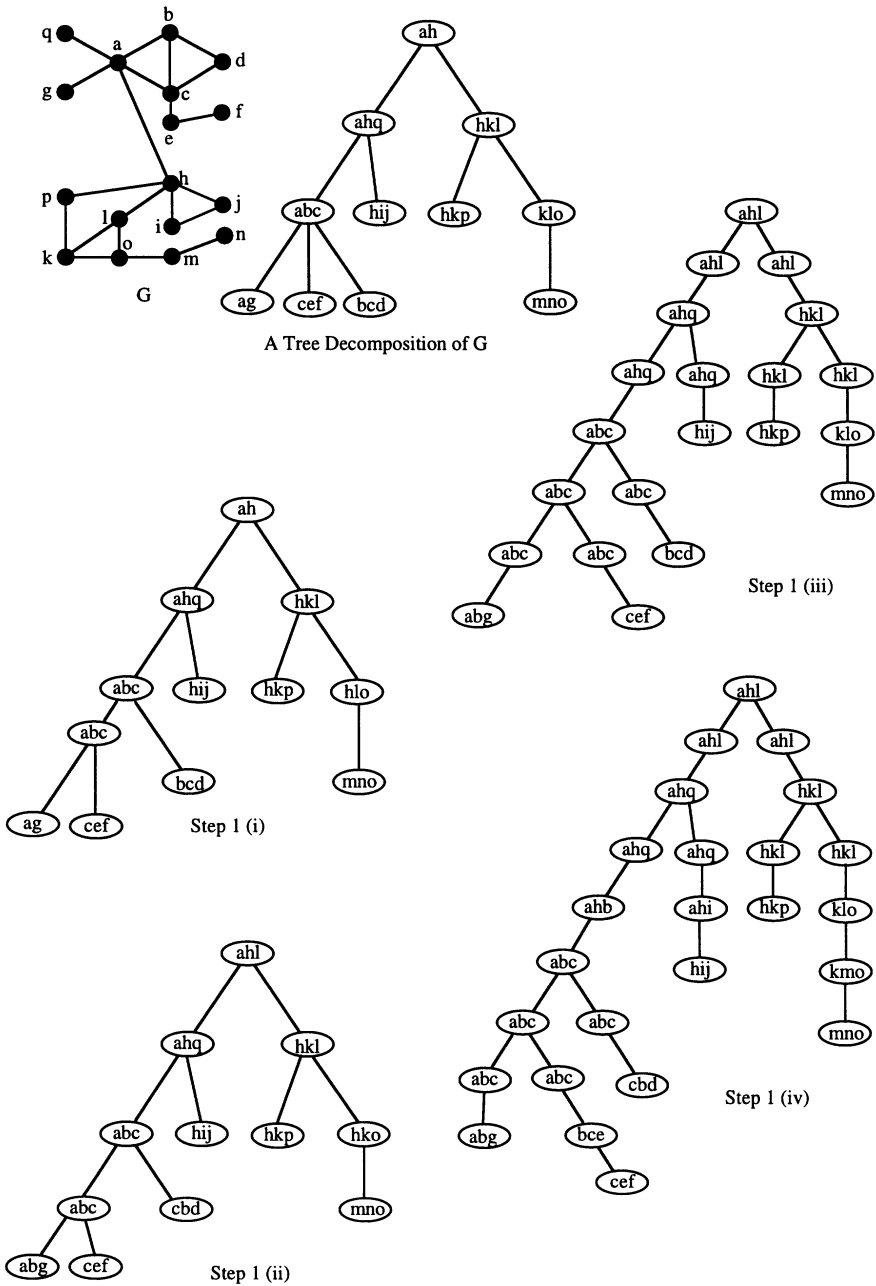


FIGURE 6.20. Normalizing the Tree Decomposition

Step 3. For each leaf of $\mathcal{T}$, label the vertices with $\{1, \dots, t+1\}$ in any order. Now, starting with the leaves of $\mathcal{T}$ and proceeding to parents, generate a parse tree for G via $\mathcal{T}$, as follows:

- (i) First, locate the edge to be covered. The inductive hypothesis will be that this corresponds to labeled vertices.
- (ii) Using γ and i , rearrange the labels so that 1 and 2 fall on the vertices u and v corresponding to the edge to be added.
- (iii) Now, use e to add the edge between u and v .

There are now two subcases, depending on whether the parent is a branch node or a nonbranch node.

- (iv) (a) If the parent is a branch node, then use γ and i to align the boundaries so that corresponding vertices have the same labels. (Here is where we use the fact that children of branches are identical.) Branches will be represented by $\oplus$.
- (b) If the parent is not a branch node, then without loss of generality, the parent differs from the child by exactly one new vertex (v). First, rearrange the labels on the vertices so that 1 is the label of the vertex to be deleted. Now use u to add a new vertex (labeled 1). This will be the vertex to represent v . Part of this process is represented in Figure 6.21.

It is clear that in this way we may parse G . Moreover, the size of the parse tree and the time needed to construct it is $O(|G|)$.

Part (b) of the theorem is left to the reader (Exercise 4).

The way is now clear. To design algorithms for the recognition of whether a bounded treewidth graph obeys a given property, we translate the problem into the arena of tree automata via parse trees. In the arena of finite labeled trees, we have many standard tools at our disposal, and we would hope to be able to use automata theory on parse trees as a *standard design tool* for problems of bounded treewidth. The culmination of this idea is Courcelle's theorem, which we treat in Section 6.5.

There are two universes of discourse for the discussion to follow.

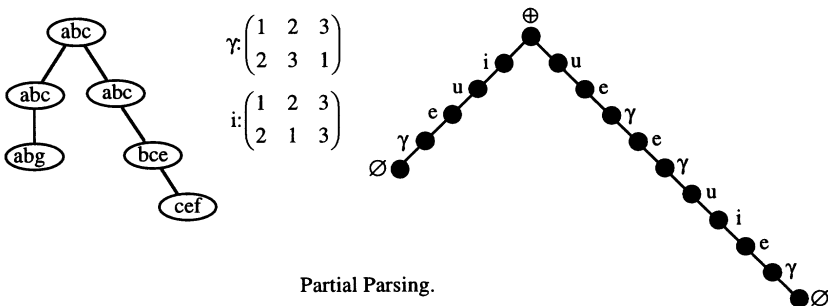


FIGURE 6.21. Partial Parsing of the Normalized Decomposition

Definition 6.73 (Small and Large Universes)

- (i) We let $\mathcal{U}_t^{large}$ denote the *large* universe, which consists of *all* t -boundaried graphs.
- (ii) We let $\mathcal{U}_t^{small}$ denote the *small* universe of t -boundaried graphs obtained from parse trees for treewidth t graphs; that is, for any parse operator $\otimes$ with n arguments, if $H_1, \dots, H_n \in \mathcal{U}_t^{small}$, then $H_1 \otimes \dots \otimes H_n \in \mathcal{U}_t^{small}$.

Completeness Convention. Definition 6.73(ii) depends upon the particular choice of parse operators, but we will assume that the set we have chosen forms a complete set for treewidth t graphs.

Naturally, we would expect that the automata-theoretic properties of the parse trees will have natural reflections in the underlying graphs. Before we give an important example, we will need to note one crucial property of the parsing operators we will use. This property is the following.

The Parsing Replacement Property For every operator $\otimes$ with arguments $H_1, \dots, H_n \in \mathcal{U}_t^{small}$ and any i , there is a graph $G \in \mathcal{U}_t^{small}$ such that

$$H_1 \otimes \dots \otimes H_n \cong H_i \oplus G \text{ as unlabeled graphs.}$$

It is important here, and in the following, that the isomorphism $\cong$ has $H_1 \oplus \dots \oplus H_n$ isomorphic to $H_i \oplus G$, but as graphs where the t -boundaries are forgotten.

Observation 6.74 *The standard size $t + 1$ parse operators of Theorem 6.72 have the parsing replacement property.*

Proof. Since we drop the boundaries for $\cong$, there is nothing to prove for γ , i , and $\emptyset$ since in all cases we can use $\emptyset$ for G . For e , note that $e(H)$ equals $H \oplus e(\emptyset)$. There is also nothing to prove for $\oplus$. Finally, for u , we need to note that $u(H)$ is isomorphic to, as an unlabeled graph, $H \oplus u(\emptyset)$.

Convention. We will henceforth assume that we have a complete set of parse operators for treewidth t graphs that additionally have the parsing replacement property.

Following ideas of Abrahamson and Fellows [7], we can now prove a graph-theoretic analog of the Myhill–Nerode Theorem.

Definition 6.75 (Canonical and Right Congruences for Graphs) Let $\mathcal{U}$ be a universe of t -boundaried graphs. (For our purposes, $\mathcal{U}$ will either be $\mathcal{U}_t^{large}$ or $\mathcal{U}_t^{small}$.)

- (i) Let $G_1, G_2 \in \mathcal{U}$. Let F be a family of graphs in $\mathcal{U}$. Then, we say that $G_1 \sim_F G_2$ (relative to $\mathcal{U}$) iff for all $H \in \mathcal{U}$,

$$G_1 \oplus H \in F \text{ iff } G_2 \oplus H \in F.$$

- (ii) More generally, we will say that $\equiv$ is a *right congruence* over $\mathcal{U}$ iff for all $G_1, G_2 \in \mathcal{U}$,

$$G_1 \equiv G_2 \text{ iff for all } H \in \mathcal{U}, G_1 \oplus H \equiv G_2 \oplus H.$$

Notice that $\sim_F$ is a right congruence since $\oplus$ is associative.

Definition 6.76 (*t*-Finite State, *t*-Cutset Regular) Let F be a family of graphs. Let F_t denote the restriction of F to $\mathcal{U}_t^{small}$ ($= F \cap \mathcal{U}_t^{small}$).

- (i) We will say that F is *t*-finite state⁹ iff the collection of parse trees corresponding to F_t is finite state.
- (ii) We will say that F is *finite state* iff F is *t*-finite state for all t .

Theorem 6.77 (Analog of Myhill–Nerode for Treewidth t Graphs) *Let F be a family of graphs. Then the following are equivalent:*

- (i) F is *t*-finite state.
- (ii) F_t is a union of equivalence classes of a right congruence of finite index over $\mathcal{U}_t^{small}$.
- (iii) $\sim_{F_t}$ has finite index over $\mathcal{U}_t^{small}$.

Remark. The reader should carefully note that in Theorem 6.77, (i) expresses a property *in the universe of trees*, whereas (ii) and (iii) express properties *in the universe of graphs*. A large part of the proof is devoted to moving from one arena to another.

Proof. We use the following notation: If T is a parse tree of a t -boundaried graph, then $G(T)$ denotes the corresponding graph. Similarly, if G is a treewidth t graph, let $T(G)$ denote any standard parse tree for G , obtained by, say, the proof of the Parsing Theorem, Theorem 6.72.

• (i) $\rightarrow$ (ii) [and (iii)] Suppose that F is *t*-finite state. Then F_t generates a finite-state collection $L = L(F_t)$ of parse trees relative to Σ^{**} . By the Myhill–Nerode Theorem for Tree Automata, Theorem 6.42, $\sim_L$ is a right congruence of finite index over Σ^{**} . Recall that $\sim_L$ being a right congruence over Σ^{**} means that for all trees $T_1, T_2 \in \Sigma^{**}$, and $T \in \Sigma_x^{**}$ with exactly one (leaf) node labeled x , $T_1 \sim_L T_2$ means that

$$T_1 \cdot_x T \in L \text{ iff } T_2 \cdot_x T \in L.$$

We claim that (iii), and hence (ii), holds. Suppose that $G_1, G_2, \dots$ are an infinite family of graphs in $\mathcal{U}_t^{small}$ which are inequivalent with respect to $\sim_{F_t}$. Then, for all $i \neq j$, there exist witnesses $H_{i,j} \in \mathcal{U}_t^{small}$ such that

$$G_i \oplus H_{i,j} \in F_t \text{ iff } G_j \oplus H_{i,j} \notin F_t.$$

Thus, for all $i \neq j$,

$$T(G_i) \cdot_x [x \oplus T(H_{i,j})] \in L \text{ iff } T(G_j) \cdot_x [x \oplus T(H_{i,j})] \notin L,$$

since $G(T(G_k) \cdot_x [x \oplus T(H_{i,j})]) = G_k \oplus H_{i,j}$. But then the family $\{T(G_i) : i \in \mathbb{N}\}$ is an infinite set of trees inequivalent with respect to $\sim_L$, contradicting the Myhill–Nerode Theorem for Trees. Hence, $\sim_{F_t}$ has finite index after all.

⁹In the literature, “*t*-finite state” is often referred to as *t*-cutset regular.

(ii) $\rightarrow$ (iii) This uses the same argument as (ii) $\rightarrow$ (iii) of Theorems 6.21 and 6.42, arguing that $\sim_{F_t}$ is the coarsest such right congruence.

(iii) $\rightarrow$ (i) Suppose that $\sim_{F_t}$ has finite index. We need to argue that F is t -finite state. To do this, by the Myhill–Nerode Theorem for Trees, we need to prove that $\sim_L$ has finite index. Here is where we need the Parsing Replacement Property. Suppose that $T_1, T_2, \dots$ are trees in $\Sigma_{\{x\}}^{**}$ with $T_i \not\sim_L T_j$ for $i \neq j$. Note that without loss of generality, we must have $G(T_i) \in \mathcal{U}_t^{small}$ for all i . [It is only possible for $G(T_i) \notin \mathcal{U}_t^{small}$ for one i , since all trees that do not represent t -boundaried graphs are equivalent.] Then, for all $i \neq j$, there is a tree $T_{i,j}$ with x on exactly one leaf such that

$$T_i \cdot_x T_{i,j} \in L \text{ iff } T_j \cdot_x T_{i,j} \notin L.$$

Let $T'_{i,j}$ denote the tree obtained by replacing x with $\emptyset$. Note that $G(T'_{i,j}) = G(\emptyset \cdot_x T_{i,j}) \in \mathcal{U}_t^{small}$. We claim that there is a tree $Q_{i,j}$, with $G(Q_{i,j}) \in \mathcal{U}_t^{small}$ such that

$$G(T_k \cdot_x T_{i,j}) \cong G(T_k \oplus Q_{i,j}), \text{ for } k \in \{i, j\}.$$

To see that this holds, we argue inductively on the length ℓ of the path in $T_{i,j}$ from the root to x . If $\ell = 1$, then $T_k \cdot_x T_{i,j}$ is the same as $H_1 \otimes \dots \otimes T_k \otimes \dots \otimes H_m$ for some operator $\otimes$, and trees H_w with $G(H_w) \in \mathcal{U}_t^{small}$. By the Parsing Replacement Property, we have that

$$G(H_1) \otimes \dots \otimes G(T_k) \otimes \dots \otimes G(H_m) \cong G(T_k) \oplus R_{i,j},$$

for some $R_{i,j} \in \mathcal{U}_t^{small}$. But then we can take $Q_{i,j} = T(R_{i,j})$.

So suppose that $\ell > 1$. The idea is that we will work our way up the path π from the root to x in $T_{i,j}$. Let W_k be the subtree of $T_k \cdot_x T_{i,j}$ containing T_k , generated by the child of the root of $T_k \cdot_x T_{i,j}$. (See Figure 6.22.)

Note that $G(W_k) \in \mathcal{U}_t^{small}$. We may argue, as in the case $\ell = 1$, that there are $H_1, \dots, H_m$ with

$$T_k \cdot_x T_{i,j} = H_1 \otimes \dots \otimes W_k \otimes \dots \otimes H_m$$

and with $G(H_b) \in \mathcal{U}_t^{small}$ for all b . Then, by the Parsing Replacement Property, there is a graph $R_{i,j}^1 \in \mathcal{U}_t^{small}$ with

$$\begin{aligned} G(T_k \cdot_x T_{i,j}) &= G(H_1 \otimes \dots \otimes W_k \otimes \dots \otimes H_m) \\ &= G(H_1) \otimes \dots \otimes G(W_k) \otimes \dots \otimes G(H_m) \cong G(W_k) \oplus R_{i,j}^1. \end{aligned}$$

Now, we can argue in exactly the same fashion with W_k in place of $T_k \cdot_x T_{i,j}$. That is, consider the subtree W'_k of W_k above the child of the root of W_k containing T_k . Again, there must be $H'_1, \dots, H'_{m'}$ with $G(H'_b) \in \mathcal{U}_t^{small}$ for all b and an operator $\otimes'$, such that

$$W_k = H'_1 \otimes' \dots \otimes' W'_k \otimes' \dots \otimes' H'_{m'}.$$

The reasoning above will yield a graph $R_{i,j}^2$ such that

$$G(W_k) \cong G(W'_k) \oplus R_{i,j}^2.$$

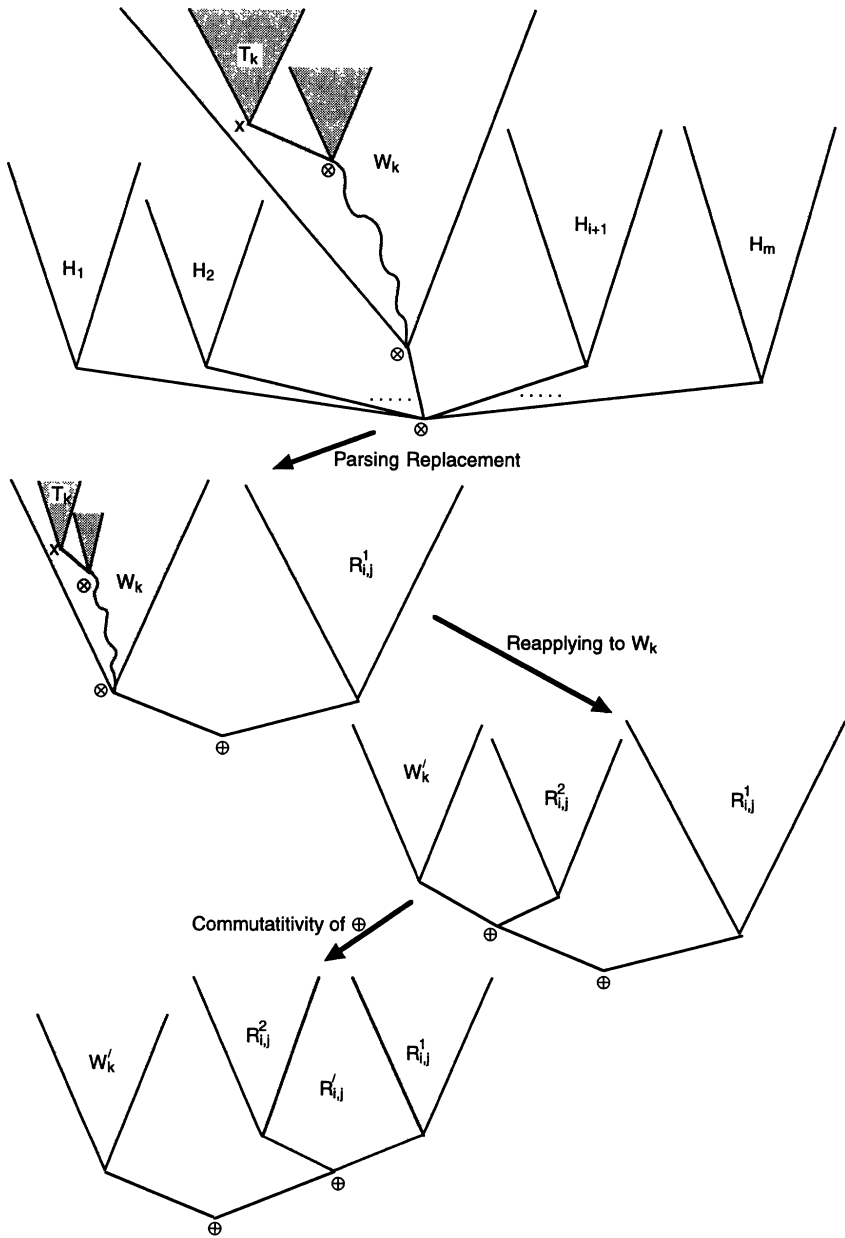


FIGURE 6.22. Replacement for the Myhill–Nerode Theorem

But then,

$$G(T_k \cdot_x T_{i,j}) \cong G((W'_k \oplus R_{i,j}^2) \oplus R_{i,j}^1) \cong G(W'_k \oplus R'_{i,j}) \text{ where } R'_{i,j} = R_i^1 \oplus R_{i,j}^2.$$

Continuing in this fashion, we work our way up the path from the root to x in $T_{i,j}$ to eventually get a graph $R_{i,j}^* \in \mathcal{U}_t^{small}$ such that

$$G(T_k \cdot_x T_{i,j}) \cong G(T_k) \oplus R_{i,j}^*.$$

Hence, $G(T_k) : k = 1, 2, \dots$ are inequivalent (as witnessed by $R_{i,j}^*$) with respect to $\sim_{F_t}$. This contradicts the fact that $\sim_{F_t}$ has finite index.

Again, because of the fact that $\sim_{F_t}$ is the coarsest right congruence on $\mathcal{U}_t^{small}$, there is an implicit state minimization process in the above. Notice that the proof also shows that if $T_1 \cdot_x T_2$ is the parse tree of a graph G in $\mathcal{U}_t^{small}$, then there is a parse tree of G of the form $T_1 \oplus T_2'$.

Abrahamson and Fellows (also) call the property of $\sim_{F_t}$ a right congruence of finite index on $\mathcal{U}_t^{small}$ *t-cutset regularity*. We will call this property *t-finite index*.

Just as we can use the Myhill–Nerode Theorems for trees or formal languages to prove that certain languages are not finite state, we may use Theorem 6.77 to establish that various graph-theoretic properties are not *t*-finite state.

As an illustration, we will look at the example of **BANDWIDTH**.

Definition 6.78 (Bandwidth, Layout)

- (i) A *layout* of a graph $G = (V, E)$ is a one-to-one function $f : V \rightarrow \{1, \dots, |V|\}$.
- (ii) The *bandwidth* of layout f of G is the $\max_{uv \in E} |f(u) - f(v)|$. The *bandwidth* of G is the minimum bandwidth of a layout of G .

The classical problem stemming from VLSI design is

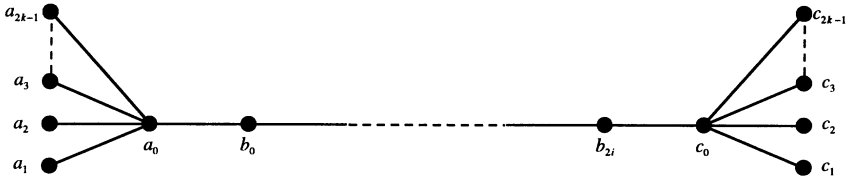
BANDWIDTH

- Input:* A graph G .
Parameter: A positive integer k .
Question: Does G have bandwidth $\leq k$?

As we will later see, not only is the nonparameterized version of **BANDWIDTH** NP-complete, but, in fact, we believe the parameterized version above to be fixed-parameter intractable. Certainly, as we will now see, it is provably not finite state.

Theorem 6.79 (Abrahamson and Fellows, [7]) *The family F_k of graphs having bandwidth at most k is not *t*-finite state if $t \geq 1$ and $k \geq 2$.*

Proof. It suffices to prove the theorem for $t = 1$, since additional boundary vertices can easily be added. Our proof consists simply of exhibiting an infinite set of representatives of distinct equivalence classes of the equivalence relation $\sim_{F_k}$. The representatives are $(X_i, i \geq 1)$, where X_i is the graph illustrated in Figure 6.23.

FIGURE 6.23. Graphs for **BANDWIDTH**

The single boundary vertex of X_i is b_i . The stars at either end of X_i strongly constrain the layouts of bandwidth at most k . They must look like Figure 6.23, up to permutation of the layouts of the stars. Notice that each X_i is a tree and hence has treewidth 1.

Let Y_i be a path of $2i + 1$ vertices, whose single boundary vertex is the centroid of the path. Suppose $n \geq m$. The reader can easily check that $X_m \not\sim_{F_k} X_n$ by observing the following:

- (1) $X_n \oplus Y_{kn-n} \in F_k$;
- (2) $X_m \oplus Y_{kn-n} \notin F_k$.

The layout needed to verify assertion (1) is constructed by placing $k - 1$ vertices of Y_{kn-n} between each pair (b_i, b_{i+1}) , for $i = 0, \dots, 2n - 1$. There is not enough room in X_m to do the same thing, so some edge will have to be stretched to length more than k to accommodate Y_{kn-n} within the layout for X_m .

Thus, by Theorem 6.77, **BANDWIDTH** for $k \geq 2$ is not t -finite state.

Theorem 6.79 leaves only the case $k = 1$ undetermined. A graph G has a layout of bandwidth 1 if and only if G is a disjoint union of paths. So each equivalence class is determined by where the boundary vertices lie on paths: whether they are isolated, at the end of a path, or somewhere in the interior of a path. Hence, for t -boundaried graphs, there are 3^t equivalence classes.

Method of Test Sets for Treewidth t Graphs. The proof above encapsulates the Method of Test Sets for Graphs. We need to follow the steps below.

Step 1: Identify a finite set S of tests T to be performed on a t -boundaried graph X . (For example, a test might consist of a t -boundaried graph Y for which we determine whether $X \oplus Y$ is in F .)

Step 2: Argue that the equivalence relation $X \sim Y$ defined as: $X \sim Y$ if and only if $S_X = S_Y$ where $S_X = \{T \in S : X \text{ passes the test } T\}$ has the properties (1) $X \sim Y$ and $X \in F$ implies $Y \in F$, and (2) $\sim$ is a right congruence.

We conclude this section with a couple of examples: **HAMILTONICITY** and **CUTWIDTH**. We begin with **HAMILTONICITY**. Recall that a graph is called Hamiltonian iff it contains a simple cycle through all of the vertices. Determining if a graph is Hamiltonian is a famous *NP*-complete problem. The first positive illustration of the method of test sets for graphs is to prove that **HAMILTONICITY** is solvable in time $O(n)$ for graphs of bounded treewidth.

Theorem 6.80

- (i) **HAMILTONICITY** has t -finite index.
- (ii) Consequently, **HAMILTONICITY** is FPT (for parameter t) via an $O(n)$ algorithm for graphs of treewidth $\leq t$.

Proof. (ii) follows from (i) via Bodlaender's theorem, Theorem 6.53. So, we need only prove (i). This is a classic example of the application of the method of test sets.

Let $\mathcal{H}$ denote the collection of Hamiltonian graphs of treewidth $\leq t$. We need to establish that for all t , $\sim_{\mathcal{H}}$ has finite index on $\mathcal{U}_t^{\text{small}}$. Actually, we establish that $\sim_{\mathcal{H}}$ has finite index on $\mathcal{U}_t^{\text{large}}$. Now, the idea is to examine how *information can flow across a t -boundary*. Remember, we are considering $\sim_{\mathcal{H}}$ and hence looking at graphs of the form $G_i \oplus K$ for $i \in \{1, 2\}$. There are only so many ways that a graph K can influence whether or not $G_i \oplus K$ is Hamiltonian. Take, for instance, $t = 4$. Then, G_i has four boundary vertices with labels $\{1, 2, 3, 4\}$, say, $V = \{v_1, v_2, v_3, v_4\}$. Now, any Hamilton path in $G_i \oplus K$ must cover the vertices of V . You might, for instance, start in G , pass through v_2 into K back through v_4 into G , then through v_1 but remain in G , and then through v_3 but remain in G . Running through possibilities like this gives a canonical set of alternatives, which forms the collection of test sets $\mathcal{T}$. In Figure 6.24, we list the possibilities for treewidth 3, with the first one being the example just described.

One easily sees that if

$$G_1 \oplus T \in \mathcal{H} \text{ iff } G_2 \oplus T \in \mathcal{H},$$

for all T in the collection of test sets of Figure 6.24, then $G_1 \sim_{\mathcal{H}} G_2$ (Exercise 6). The same reasoning will extend for any t since the number of test sets is finite for each t . Since $\sim_{\mathcal{H}}$ has finite index, for each t , **HAMILTONICITY** is finite state.

A more intricate example is provided by **CUTWIDTH**.

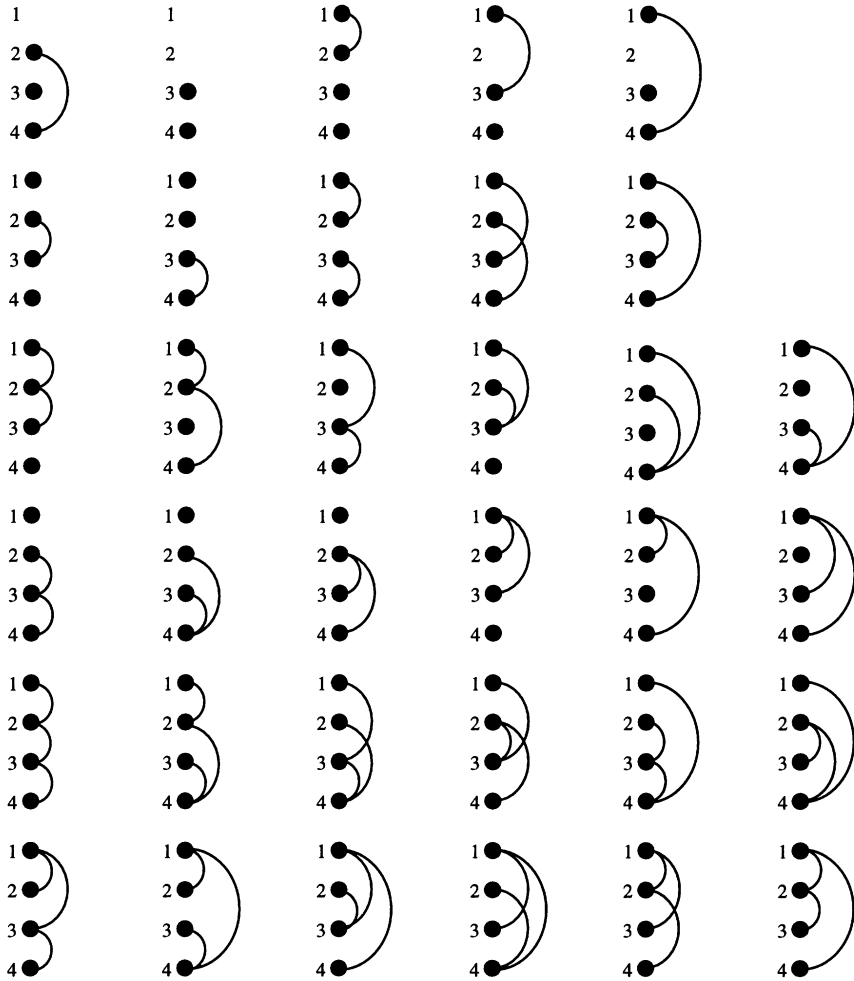
Definition 6.81 (Cutwidth)

- (i) Let f be a layout of a graph $G = \langle V, E \rangle$. We regard $[1, |V|]$ as an interval on the real line $\mathbb{R}$. If $\alpha \in \mathbb{R}$, the *value of the cut at α* is the number of edges $uv \in E$ with $f(u) < \alpha$ and $f(v) > \alpha$.
- (ii) The *cutwidth* of a layout f of G is the maximum of the value of the cut over all α . The cutwidth of G is the minimum of the cutwidths of all possible layouts of G .

CUTWIDTH

- Input:* A graph G .
Parameter: A positive integer k .
Question: Does G have cutwidth $\leq k$?

Theorem 6.82 (Abrahamson and Fellows [7]) *For every k and for every t , the problem of deciding whether a graph has a layout of cutwidth at most k is t -finite state. Hence, for graphs of treewidth t , **CUTWIDTH** is $O(n)$ and hence FPT.*

FIGURE 6.24. Test Set of Graphs for **HAMILTONICITY** for Treewidth 3

Proof. Fix k and t , and let F denote the family of graphs of cutwidth at most k . Let $w : \mathbb{R} \rightarrow \mathbb{N}$ be a non-negative integer-valued function which we will refer to as a *weighting* of the real line $\mathbb{R}$. We will consider the cutwidth of a graph relative to a weighting of $\mathbb{R}$ in the following way.

If l is a layout of a graph $G = (V, E)$, the *value of a cut at $\alpha \in \mathbb{R}$ with respect to w* is the sum of $w(\alpha)$ and the number of edges uv of G with $l(u) < \alpha$ and $l(v) > \alpha$. The *cutwidth of a layout l of G with respect to w* is the maximum value of a cut at any point $\alpha \in \mathbb{R}$. The *cutwidth of G with respect to w* is the minimum cutwidth of a layout of G with respect to w .

A test $T = (\pi, S)$ of size n consists of the following:

- (1) A map $\pi : \{1, \dots, t\} \rightarrow \{1, \dots, n\}$.
- (2) A sequence S of non-negative integers $S = (S(0), \dots, S(n))$.

To a test T of size n we associate a weighting w_T which consecutively assigns to the intervals of $\mathbb{R}$: $(0, 1), (1, 2), \dots, (n, n + 1)$, the values $S(0), \dots, S(n)$. Precisely, $w_T(\alpha) = S(i)$ if $i < \alpha < i + 1$ and $w_T(\alpha) = 0$ otherwise.

We say that a t -boundaried graph $G = (V, E, B, f)$ passes a test T of size n if there is a layout l of G of cutwidth at most k relative to w_T that further satisfies the following conditions:

- (1) For all $v \in V$, $0 < l(v) < n + 1$.
- (2) If u is a boundary vertex of G with label j , $f(u) = j$, then $l(u) = \pi(j)$. Thus, the boundary vertices are laid out in the order and position described by π .
- (3) If u is not a boundary vertex, $u \in V - B$, then $l(u) \notin N$. Thus, non-boundary vertices of G are assigned layout positions in the interiors of the weighted intervals of $\mathbb{R}$.

Let G and H be t -boundaried graphs and define $G \sim H$ if and only if G and H pass exactly the same set of tests. We will establish the following.

Claim 1. There is a finite set of *reduced tests* such that G and H agree on all tests if and only if they agree on the set of reduced tests.

Claim 2. If $G \sim H$, then $G \sim_F H$.¹⁰

Claim 3. If $G \in F$ and $H \sim G$, then $H \in F$.

From these three claims, it follows by Theorem 6.77 that F is finite state.

Note that Claim 3 is trivial, since given a layout l for G , we can describe a test T_l where the weightings of the intervals of $\mathbb{R}$ are zero, which G passes. Since H also passes T_l , there is a cutwidth k layout of H , which furthermore places the boundary vertices in the same order.

We first prove Claim 2. Suppose Z is a t -boundaried graph such that $G \oplus Z \in F$ and $H \oplus Z \notin F$. Let l be a layout of $G \oplus Z$. From l can be described a test T_l that is passed by G but failed by H . The test T_l records the interval weights and boundary positions given by l restricted to Z .

Finally, we turn to Claim 1. We define an equivalence relation on tests, with $T \sim T'$ if and only if every t -boundaried graph that passes T passes T' and *vice versa*. We will argue that there are a finite number of equivalence classes of tests.

Call the sequence of integer weights *between two consecutive images* of $\{1, \dots, t\}$ under the map π of a test T a *load pattern* of T . Each test $T = (\pi, S)$ thus has $t + 1$ load patterns between the images of the boundary vertices prescribed by π .

¹⁰Here, we wish to use the consequence of Theorem 6.77 that if the canonical equivalence relation $\sim_F$ defined on the large universe of t -boundaried graphs has finite index, then F is t -finite state.

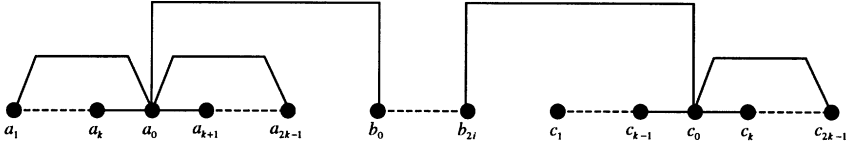


FIGURE 6.25. Constrained Layout for CUTWIDTH

We will use the following notation. The symbols s , s' , and s_i denote sequences of integers taken from the set $J = \{0, \dots, k\}$. Letters such as a, b , and c will be used to denote single particular values in J . Two load patterns s and s' are termed *equivalent* if $T \sim T'$ for any test T for which s is a load pattern, where T' is the test obtained by replacing s with s' . Write $s \geq s'$ if every t -boundaried graph which passes a test T for which s is a load pattern also passes the test T' where the load pattern s has been replaced by s' . Thus, for example, we have $s = (5, 4, 1, 3, 2, 7) \geq (5, 1, 1, 1, 1, 7) \geq (5, 1, 7) = s'$ and $(5, 1, 7) \geq (5, 5, 1, 7, 7, 7) \geq (5, 4, 1, 3, 2, 7)$, and therefore $s \sim s'$, since decreasing the weight of an interval only makes finding a cutwidth k layout easier, and consecutively repeating weights (or deleting such repetitions) makes no difference.

We have the following reduction rules for load patterns:

- (R1) If $s = s_1 a b c s_2$ with $a \leq b \leq c$ (or $a \geq b \geq c$) then $s \sim s_1 a c s_2$.
- (R2) If $s = s_1 a s_2 b s_3$ where each element of s_2 is greater than or equal to the maximum of a and b , and $c = \max(s_2)$, then $s \sim s_1 a c b s_3$.
- (R3) If $s = s_1 a s_2 b s_3$ where each element of s_2 is at most the minimum of a and b , and $c = \min(s_2)$, then $s \sim s_1 a c b s_3$.

R1 is shown by the sequence of implications: $s_1 a b c s_2 \geq s_1 a a c s_2 \geq s_1 a c s_2 \geq s_1 a c c s_2 \geq s_1 a b c s_2$. To see R2 let $m = \max(a, b)$, and suppose c is in the $i + 1$ position of s_2 , and that $|s_2| = i + j + 1 = q$. We have the sequence of implications: $s_1 a s_2 b s_3 \geq s_1 a m^i c m^j b s_3 \geq s_1 a^{i+1} c b^{j+1} s_3 \geq s_1 a c b s_3 \geq s_1 a c^q b s_3 \geq s_1 a s_2 b s_3$. The argument for R3 is similar.

Note that since we are concerned with cutwidth k where k is fixed, every load pattern is trivially equivalent to one where the largest integer weight occurring is at most k . Define a load pattern s to be *reduced* if it contains only integer values in the range $\{0, \dots, k\}$ and none of the reduction rules R1-R3 can be applied to s . Define a test T to be *reduced* if each of the $t + 1$ load patterns of T is reduced.

We next argue that a reduced load pattern s over $\{0, \dots, k\}$ has length at most $2k + 2$. Otherwise, s must contain some value a at least three times. Consider the factorization $s = s_1 a s_2 a s_3 a s_4$ where a does not occur in s_2 or s_3 . Neither of the subsequences s_2 or s_3 can be empty, or s is not reduced. Let b be the rightmost integer in s_2 and let c be the leftmost integer in s_3 . If $b < a < c$ or $b > a > c$, then R1 can be applied. So both of b and c must be greater than a , or both must be less than a . Suppose the latter. (The other case is handled similarly.)

Let d denote the rightmost occurrence in the subsequence $a s_2$ of a value greater than or equal to a . Let e denote the leftmost occurrence in the subsequence $s_3 a$ of

a value greater than or equal to a . There are at least three integers in s properly between d and e , and each intervening value is at most a . By R3 s is reducible.

It follows that there are finitely many reduced tests, completing the proof.

Since a graph has cutwidth t only if it has treewidth $\leq t$, by Bodlaender's Theorem, we have the following corollary.

Corollary 6.83 *CUTWIDTH is strongly FPT with a $O(n)$ recognition algorithm.*

Historical Notes

The methods of the present section were introduced by Abrahamson and Fellows [6] from which grew [7]. Both of these articles have incorrect proofs of Theorem 6.77, due to the fact that the Myhill–Nerode theorem for trees is incorrectly stated. The present form of Theorem 6.77 and its proof are new here. Also, the proof for **CUTWIDTH** has consequential changes. The proof of **HAMILTONICITY** being *FPT* for bounded treewidth t graphs appears here for the first time, although, as we will see, it is an immediate corollary of Courcelle's theorem of the next section. Test sets for **HAMILTONICITY** for graphs of bounded pathwidth appear in Xiu Yan Liu's thesis [465]. Doyle [186] derived the formula in Exercise 5 for the number of test sets for **HAMILTONICITY**.

Exercises 6.4

- 6.4.1.** Prove that a graph has treewidth 2 iff it can be parsed by the operators of Figure 6.19.
- 6.4.2.** Construct the parse tree corresponding to the graph of Figure 6.20.
- 6.4.3.** Prove that the class of treewidth t graphs with at least $t + 1$ vertices can be parsed by $\oplus$, $\emptyset$, γ , and i together with the operators generated by taking the collection of all subgraphs of K_{t+1} with $t + 1$ vertices with the labels $\{1', \dots, t'\}$ and $\{1, \dots, t\}$ are put on the vertices in all possible ways.
(Hint: Use the partial t -tree characterization, and induction: argue that any partial t -tree P embedded in a t -tree, E can be so parsed so that any given copy of K_t in E will have at the end the labels $\{1, \dots, t\}$. In the induction, take the target K_t and consider the cutsets.)
- 6.4.4.** Prove Theorem 6.72(b) to establish that the six size $t + 1$ operators form a complete set for graphs of treewidth t .
- 6.4.5.** Construct a formula which computes the number of test sets for **HAMILTONICITY** based on the number of partitions of a set of size t .
- 6.4.6.** Prove that $G_1 \sim_{\mathcal{H}} G_2$ iff for all T of Figure 6.24,

$$G_1 \oplus T \in \mathcal{H} \text{ iff } G_2 \oplus T \in \mathcal{H}.$$

6.5 Courcelle's Theorem

6.5.1 The Basic Theorem

Courcelle's theorem is a high-level machine for establishing that various graph-theoretic properties are finite state for bounded treewidth. Similar results were obtained independently by Borie, Parker and Tovey [94]. The high-level language is the fragment of the second-order theory of graphs called the *Monadic Second-Order Theory of Graphs*.

The syntax of the second-order monadic logic of graphs includes the logical connectives $\wedge, \vee, \neg$, variables for vertices, edges, sets of vertices and sets of edges, the quantifiers $\forall, \exists$ that can be applied to these variables, and the five binary relations:

- (1) $u \in U$, where u is a vertex variable and U is a vertex set variable.
- (2) $d \in D$, where d is an edge variable and D is an edge set variable.
- (3) $\text{inc}(d, u)$, where d is an edge variable, u is a vertex variable, and the interpretation is that the edge d is incident on the vertex u .
- (4) $\text{adj}(u, v)$, where u and v are vertex variables and the interpretation is that u and v are adjacent vertices.
- (5) Equality for vertices, edges, sets of vertices and sets of edges.

We will use lowercase letters for vertices or edge variables and uppercase letters for variables representing sets of edges or sets of vertices.

Example 6.84 (HAMILTONICITY) A graph G is Hamiltonian iff the edges of G can be partitioned into two sets *red* and *blue* such that

- (i) each vertex has exactly two incident red edges, and
- (ii) the subgraph induced by the red edges spans G . In the example below we will represent the set of red edges by R and the set of blue edges by B .

Hamiltonian $\equiv \exists R \exists B \forall u \forall v [\text{part}(R, B) \wedge \text{deg}(u, R) = 2 \wedge \text{span}(u, v, R)]$, where *span*, *deg* and *part* are described below.

$\text{part}(R, B) \equiv \forall e [(e \in R \vee e \in B) \wedge \neg(e \in R \wedge e \in B)]$.

$\text{deg}(u, R) = 2 \equiv [\exists e_1, e_2 (e_1 \neq e_2 \wedge \text{inc}(e_1, u) \wedge \text{inc}(e_2, u) \wedge e_1 \in R \wedge e_2 \in R)] \wedge \neg[\exists e_1, e_2, e_3 (e_1 \neq e_2 \neq e_3 \wedge (\text{inc}(e_i, u) \wedge e_i \in R \text{ for } i \in \{1, 2, 3\}))]$.

$\text{span}(u, v, R) \equiv \forall V, W [(\text{part}(V, W) \wedge u \in V \wedge v \in W)$

$\rightarrow (\exists e, x, y (\text{inc}(e, x) \wedge \text{inc}(e, y) \wedge x \in V \wedge y \in W \wedge e \in R))]$.

Actually, this language is referred to in the literature as the *extended* monadic second-order language, and the theory the *extended* monadic second-order theory or MS_2 theory (e.g. Seese [415], Arnborg, Lagergren, and Seese [35]). This is because we are looking at a two-sorted structure with predicates for edges and vertices plus an incidence relation. Another natural language has only vertex symbols and one must use binary relations for edges. Following Courcelle, we call this one-sorted theory the MS_1 theory. Naturally, the *monadic* second-order theories are different.

For instance, Hamiltonicity is *not* MS_1 expressible (Exercise 1). For the purposes of the present section, we will concentrate on the theory with the richest expressive power MS_2 , and briefly discuss the MS_1 theory in Section 6.7. Hence, unless otherwise specified, “monadic second-order theory” means “ MS_2 theory.”

Theorem 6.85 (Courcelle's MS_2 Theorem [135, 134]) *If F is a family of graphs described by a sentence in second-order monadic logic, then F is finite state.*

Proof. Let ϕ be the sentence that describes the graph family F , and write $\sim_\phi$ for $\sim_F$. We prove the theorem by induction on formulas, for t an arbitrary positive integer describing the boundary size. For convenience, we may write $\partial(X)$ to denote the boundary of a t -boundaried graph X . By the *interior* of X , we refer to the vertices in $V(X) - \partial(X)$.

To each formula ϕ [possibly with a nonempty set $Free(\phi)$ of free variables] we associate an equivalence relation $\sim_\phi$ on the set of all t -boundaried graphs that are partially equipped with distinguished vertices, edges, sets of vertices, and sets of edges corresponding to the free variables in the set $Free(\phi)$. (The details follow shortly.)

Two partially equipped t -boundaried graphs X and Y are defined to be $\sim_\phi$ related if and only if they have the “same” partial equipment, and for every compatibly equipped t -boundaried graph Z , the formula ϕ is true for $X \oplus Z$ if and only if it is true for $Y \oplus Z$.

The compatibility condition on Z ensures that the partial equipment of Z agrees with that of X and Y on the boundary vertices and edges and further ensures that all of the free variables have interpretations as the distinguished elements and sets in the product graphs $X \oplus Z$ and $Y \oplus Z$. The formalization of what we mean by “partial equipment” is tedious but unproblematic; it can be accomplished as follows.

Let $fr(0)$, $fr(1)$, $Fr(0)$, and $Fr(1)$ denote, respectively, sets of variables for vertices, edges, sets of vertices, and sets of edges. Let $Free$ denote the disjoint union of these four sets. Allow S^2 to denote the set of all 2-element subsets of a set S . Define a *partial equipment signature* σ for $Free$ to be given by the following data.

- (1) Disjoint subsets $int_0(\sigma)$ and $\partial_0(\sigma)$ of $fr(0)$, and a map $f_\sigma^0 : \partial_0(\sigma) \rightarrow \{1, \dots, t\}$.
- (2) Disjoint subsets $int_1(\sigma)$ and $\partial_1(\sigma)$ of $fr(1)$, and a map $f_\sigma^1 : \partial_1(\sigma) \rightarrow \{1, \dots, t\}^2$.
- (3) For each vertex set variable U in $Fr(0)$, a subset $U_\sigma \subseteq \{1, \dots, t\}$.
- (4) For each edge set variable D in $Fr(1)$, a subset $D_\sigma \subseteq \{1, \dots, t\}^2$.

Notice there will be a difference between *partial* and *full* equipment. Let $Free(\sigma)$ denote the union of the sets $int_0(\sigma)$, $\partial_0(\sigma)$, $int_1(\sigma)$, $\partial_1(\sigma)$, $Fr(0)$, and $Fr(1)$. We recall the notation $X = (V, E, B, f)$ for a t -boundaried graph: V and E are, as usual, the vertices and edges of G , and $\partial(H) = B$ with f the labeling function: f is a bijection from $B \mapsto \{1, \dots, t\}$. Using this notation, we will say that a t -

boundaried graph $X = (V, E, B, f)$ is σ -*partially equipped* if it has distinguished vertices, edges, sets of vertices, and sets of edges corresponding to the variables in $Free(\sigma)$, where the correspondence is compatible with the data that describe ϕ . More precisely, the following conditions must be satisfied by these distinguished elements and sets¹¹:

- (1) If u is a vertex variable, $u \in int_0(\sigma)$, then the distinguished vertex in X corresponding to u must be in the interior of X .
- (2) If u is a vertex variable, $u \in \partial_0(\sigma)$, then the distinguished vertex in X corresponding to u must be the unique vertex $x \in V$ for which $f(x) = f_\sigma^0(u)$.
- (3) If d is an edge variable, $d \in int_1(\sigma)$, then the distinguished edge in X corresponding to d must have at least one endpoint in the interior of X .
- (4) If d is an edge variable, $d \in \partial_1(\sigma)$, then the distinguished edge in X corresponding to d must have as endpoints the pair of vertices x, y of $\partial(X)$ for which $f_\sigma^1(d) = \{f(x), f(y)\}$.
- (5) If U is a vertex set variable, $U \in Fr(0)$, then the distinguished set of vertices V_U in X corresponding to U must satisfy $f(V_U \cap \partial(X)) = U_\sigma$.
- (6) If D is an edge set variable, $D \in Fr(1)$, then the distinguished set of edges E_D in X corresponding to D must satisfy $\{\{f(x), f(y)\} : \{x, y\} \in E_D\} \cap \partial(X)^2 = D_\sigma$.

All of this is just as expected. We say that partial equipment signatures σ and $\bar{\sigma}$ for $Free$ are *complements* with respect to $Free$ if the data describing them is in agreement with respect to the boundary and if together they provide for a complete interpretation of the variables of $Free$. More precisely, $\bar{\sigma}$ must satisfy the next listed (symmetric) conditions with respect to σ . (Note that $\bar{\sigma}$ is completely determined by σ .)

- (1) $\partial_0(\bar{\sigma}) = \partial_0(\sigma)$ and $f_{\bar{\sigma}}^0 = f_\sigma^0$.
- (2) $int_0(\bar{\sigma}) = fr(0) - \partial_0(\sigma) - int_0(\sigma)$.
- (3) $\partial_1(\bar{\sigma}) = \partial_1(\sigma)$ and $f_{\bar{\sigma}}^1 = f_\sigma^1$.
- (4) $int_1(\bar{\sigma}) = fr(1) - \partial_1(\sigma) - int_1(\sigma)$.

¹¹The reader needs to be aware that our universe has now changed from the universe of (isomorphism types of) graphs to the universe of (isomorphism types of) σ -*partially equipped* graphs. These objects will thus be *tuples* consisting of graphs together with distinguished parts. Isomorphism here consists of the obvious maps that must send graphs to graphs as well as preserving the relevant distinguished sets of edges, vertices, etc. One can think of these distinguished sets of objects as colored portions of the graphs. It is important that the reader understand that many of the statements to follow talk about differing universes of not just t -boundaried graphs but ψ -*partially equipped* t boundaried graphs for various ψ .

(5) For each vertex set variable U in $Fr(0)$, $U_{\bar{\sigma}} = U_{\sigma}$.

(6) For each edge set variable D in $Fr(1)$, $D_{\bar{\sigma}} = D_{\sigma}$.

The important thing is that if partial equipment signatures σ and $\bar{\sigma}$ are complements with respect to $Free$ and if X and Z are t -boundaried graphs with X σ -partially equipped and Z $\bar{\sigma}$ -partially equipped then every variable in $Free$ has a consistent interpretation in $X \oplus Z$.

Definition. If σ is a partial equipment signature for $Free(\phi)$ and X and Y are σ -partially equipped t -boundaried graphs, then we define $X \sim_{\phi} Y$ if and only if for every $\bar{\sigma}$ -partially equipped t -boundaried graph Z : $X \oplus Z \models \phi$ if and only if $Y \oplus Z \models \phi$.

The induction claim may now be stated precisely. In what follows, we will avoid mentioning signatures when they can be deduced from the context.

Claim. *For every second-order monadic formula ϕ , with equipment signature σ , $\sim_{\phi}$ has finite index on the universe of t -boundaried σ -partially equipped graphs.*

We must first show that the claim holds for atomic formulas. Since vertex adjacency is easy to express as the existence of an edge incident on both vertices, it suffices to assume that the only atomic formulas are $d \in D$, $u \in U$, and $inc(e, u)$. Our consideration breaks up into cases according to the possible partial equipment signatures. All of these are easy and are left entirely to the reader. For example, let $\phi = inc(e, u)$ and suppose σ is described by $e \in int_1(\sigma)$ and $u \in \partial_0(\sigma)$. The relation $\sim_{\phi}$ has index 2. For another example, let ϕ be the formula $d \in D$, where d is an edge variable and D is an edge set variable, and suppose $d \in \partial_1(\sigma)$ and $D_{\sigma} = \emptyset$. The equivalence relation $\sim_{\phi}$ then has index 1.

For the induction step, we may suppose (without loss of generality) that ϕ is a formula obtained from simpler formulas in one of the following ways:

- (1) $\phi = \neg\phi'$
- (2) $\phi = \phi_1 \wedge \phi_2$
- (3) $\phi = \exists u\phi'$, where u is a vertex variable free in ϕ'
- (4) $\phi = \exists d\phi'$, where d is an edge variable free in ϕ'
- (5) $\phi = \exists U\phi'$, where U is a vertex set variable free in ϕ'
- (6) $\phi = \exists D\phi'$, where D is an edge set variable free in ϕ'

We treat each of these six cases separately, giving a complete argument for (1), (2), (3), and (6). An appropriate equivalence relation is defined for (4) and (5) and the reader will have no difficulty adapting the arguments given for the others to these two cases.

Case 1. $\phi = \neg\phi'$

It is enough to argue that $X \sim_{\phi'} Y$ implies $X \sim_{\phi} Y$. By contraposition, if $\exists Z, X \oplus Z \models \phi$ and $Y \oplus Z \models \neg\phi$ then immediately $X \oplus Z \models \neg\phi'$ and $Y \oplus Z \models \neg\phi \equiv \phi'$. The converse is just as easy, and in fact $\sim_{\phi} = \sim_{\phi'}$.

Case 2. $\phi = \phi_1 \wedge \phi_2$

Here we have $\text{Free}(\phi) = \text{Free}(\phi_1) \cup \text{Free}(\phi_2)$. Let σ be a partial equipment signature for $\text{Free}(\phi)$. Our strategy is to define a convenient equivalence relation $\sim$ on the σ -equipped t -boundaried graphs, and then show two things: that $\sim$ has finite index, and that it refines $\sim_{\phi}$.

The sets of variables in $\text{Free}(\phi_i)$, $i = 1, 2$, are subsets of the sets of variables in $\text{Free}(\phi)$, so the partial equipment signature σ for $\text{Free}(\phi)$ induces partial equipment signatures σ_1 for $\text{Free}(\phi_1)$ and σ_2 for $\text{Free}(\phi_2)$ just by forgetting the unneeded equipment. It is with respect to these induced signatures that we define $\sim$ to be $\sim_{\phi_1} \cap \sim_{\phi_2}$. Since it is, by the induction hypothesis, the intersection of equivalence relations of finite index, $\sim$ has finite index.

It remains to argue that $X \sim Y$ implies $X \sim_{\phi} Y$. By contraposition, if $\exists Z, X \oplus Z \models \phi$ and $Y \oplus Z \models \neg\phi$, then either (i) $X \oplus Z \models \phi_1$ and $Y \oplus Z \models \neg\phi_1$, or (ii) $X \oplus Z \models \phi_2$ and $Y \oplus Z \models \neg\phi_2$. So either (i) X and Y are not $\sim_{\phi_1}$ equivalent, or (ii) X and Y are not $\sim_{\phi_2}$ equivalent. In either case, it follows that X and Y are not $\sim$ equivalent.

Case 3. $\phi = \exists u \phi'$, where u is a vertex variable free in ϕ'

For $i \in \{1, \dots, t\}$, let X_i denote the partially equipped t -boundaried graph X further equipped with the boundary vertex with label i in correspondence with the variable u .

Let $\alpha(X, Z)$ denote the statement: There is a vertex x in the interior of X such that $X_u \oplus Z \models \phi'$, where X_u is X additionally equipped with the vertex x in correspondence with the variable u .

Define $X \sim_{\exists u} Y$ if and only if for all Z , $\alpha(X, Z) \leftrightarrow \alpha(Y, Z)$.

This is an equivalence relation of finite index, since, by the commutativity of $\oplus$ and the definition of α , the conditions $\alpha(X, Z)$ and $Z \sim_{\phi'} Z'$ together imply $\alpha(X, Z')$. Hence, $\alpha(X, Z)$ depends in the second argument only on the equivalence class of Z .

Define $X \sim Y$ if and only if

- (a) $X \sim_{\phi'} Y$
- (b) $X_i \sim_{\phi'} Y_i$ for all $i \in \{1, \dots, t\}$, and
- (c) $X \sim_{\exists u} Y$

The relation $\sim$ is an equivalence relation of finite index since it is the intersection of finitely many such relations on the $\text{Free}(\phi)$ -partially equipped t -boundaried graphs. [In the case of (c), this is the method of test sets in action.]

It remains only to argue that $X \sim Y$ implies $X \sim_{\phi} Y$. If not, then $\exists Z$ with $X \oplus Z \models \phi$ and $Y \oplus Z \models \neg\phi$. Choose an instantiation of u to a vertex v in $X \oplus Z$ making ϕ true.

Case (a). The vertex v is in the interior of Z . Let Z_u denote Z further equipped with the vertex v in correspondence with the variable u . Then, $X \oplus Z_u \models \phi'$ and by (a), $Y \oplus Z_u \models \phi'$ which implies $Y \oplus Z \models \phi$, a contradiction.

Case (b). The vertex v is a boundary vertex of X . Then for some $i \in \{1, \dots, t\}$, we have $X_i \oplus Z_i \models \phi'$ which by (b) implies $Y_i \oplus Z_i \models \phi'$ and therefore $Y \oplus Z \models \phi$, a contradiction.

Case (c). The vertex v belongs to the interior of X . Then, (c) implies that there is a vertex v' in the interior of Y such that $Y_u \oplus Z \models \phi'$ where Y_u is Y further equipped with the vertex v' in correspondence with the variable u . This implies $Y \oplus Z \models \phi$, a contradiction.

Case 4. $\phi = \exists d \phi'$, where d is an edge variable free in ϕ'

For each $\{i, j\} \in \{1, \dots, t\}^2$, let X_{ij} denote X further equipped with the edge ij in correspondence with the variable d .

Let $\beta(X, Z)$ be the statement: There is an edge d' with at least one interior endpoint in X such that $X_d \oplus Z \models \phi'$, where X_d denotes X equipped with d' in correspondence with d .

Define $X \sim_{\exists d} Y$ if and only if for all Z , $\beta(X, Z) \leftrightarrow \beta(Y, Z)$. This is an equivalence relation of finite index, by the commutativity of $\oplus$ and the definition of β .

Define $X \sim Y$ if and only if

- (a) $X_{ij} \sim_{\phi'} Y_{ij}$ for all $\{i, j\} \in \{1, \dots, t\}^2$
- (b) $X \sim_{\phi'} Y$
- (c) $X \sim_{\exists d} Y$

The relation $\sim$ is an equivalence relation of finite index since it is the intersection of finitely many such relations. The verification that $X \sim Y$ implies $X \sim_{\phi} Y$ is straightforward.

Case 5. $\phi = \exists U \phi'$, where U is a vertex set variable free in ϕ'

For each subset $S \subseteq \{1, \dots, t\}$ let $\gamma_S(X, Z)$, defined for Z equipped with $V_U = S$, be the statement: There is a subset U_0 of the interior of X such that $X_U \oplus Z \models \phi'$ where X_U is X equipped with the vertex set $U_0 \cup S$ in correspondence with the variable U .

Define $X \sim_S Y$ if and only if for all Z , $\gamma_S(X, Z) \leftrightarrow \gamma_S(Y, Z)$.

It follows easily from the commutativity of $\oplus$ and the definition of γ_S that $\gamma_S(X, Z)$ and $Z \sim_{\phi'} Z'$ imply $\gamma_S(Y, Z)$, and from this and the induction hypothesis that $\sim_S$ is an equivalence relation of finite index.

Define $X \sim Y$ if and only if for all $S \subseteq \{1, \dots, t\}$, $X \sim_S Y$.

The relation $\sim$ is an equivalence relation of finite index, since it is the intersection of finitely many such relations. The verification that $X \sim Y$ implies $X \sim_{\phi} Y$ is straightforward.

Case 6. $\phi = \exists D \phi'$, where D is an edge set variable.

For each subset $S \subseteq \{1, \dots, t\}^2$, let $\delta_S(X, Z)$ be the statement: There is a set of edges D_0 , each having at least one interior endpoint in X , such that $X_D \oplus Z \models \phi'$, where X_D is X equipped with the edge set $D_0 \cup S$ in correspondence with the variable D .

Define $X \sim_S Y$ if and only if $\forall Z \delta_S(X, Z) \leftrightarrow \delta_S(Y, Z)$. As in the other cases, this is an equivalence relation of finite index, by the induction hypothesis and the fact that $\delta_S(X, Z)$ depends in the second argument only on the equivalence class of Z with respect to $\sim_{\phi'}$.

Define $X \sim Y$ if and only if $\forall S \subseteq \{1, \dots, t\}^2 X \sim_S Y$.

The relation $\sim$ is an equivalence relation of finite index since it is the intersection of finitely many such relations. It remains only to show that $X \sim Y$ implies $X \sim_\phi Y$. If not, then $\exists Z$ such that (without loss of generality) $X \oplus Z \models \phi$ and $Y \oplus Z \models \neg\phi$. Fix an instantiation of the variable D to a set of edges E_D in $X \oplus Z$, making ϕ true. Let $S = E_D \cap \partial(X)$. Let Z_D denote Z additionally equipped with the set of edges of E_D in Z in correspondence with the variable D . Thus, we have $\delta_S(X, Z_D)$. Since $X \sim Y$, we have also $\delta_S(Y, Z_D)$, which implies $Y \oplus Z \models \phi$, a contradiction.

6.5.2 Implementing Courcelle's Theorem

The proof above actually contains the kernel of a possible implementation of Courcelle's theorem. Here, we follow Doyle [186].

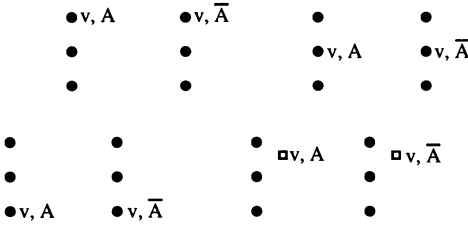
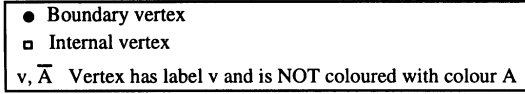
The main idea of the implementation is to use colors and labels on the edges and vertices of a graph to represent possible partial equipment of a graph. The implementation follows the proof quite closely, starting with the creation of test sets for the atomic formulas and at each stage building a new test set for the extended formula.

An edge or vertex can have a color and/or a label or neither. Colors are used to represent the instantiation of edge set and vertex set variables. Labels represent the instantiation of edge and vertex variables.

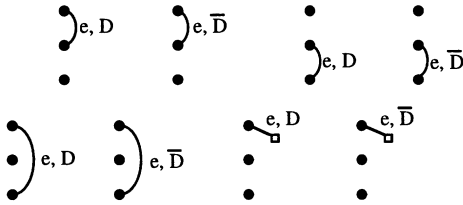
Each vertex set and edge set variable is represented by a different color and each vertex and edge variable by a different label, thus, the collection of colors and labels used is bounded in size by the number of variables in the formula. In fact, at any stage, the number of colors and labels used is bounded by the number of free variable in the formula.

Thus, the universes of discourse will change from t -boundaried graphs to colored t -boundaried graphs with extra parameters. In particular, we will modify our parse operators to generate such objects. Taking the six operators on $t + 1$ boundaried graphs, we add parameters to $\emptyset$, u and e to allow them to add labels and colors to the vertices and edges created. $\emptyset$ becomes $\emptyset(\langle l_1, l_2, \dots, l_{t+1} \rangle, \langle c_1, c_2, \dots, c_{t+1} \rangle)$, u becomes $u(l, c)$ and e becomes $e(l, c)$ where the l 's are labels corresponding to free vertex/edge variables, and the c 's are colors corresponding to free vertex set/edge set variables. $\emptyset(\langle l_1, l_2, \dots, l_{t+1} \rangle, \langle c_1, c_2, \dots, c_{t+1} \rangle)$ puts label l_1 on vertex 1 of the new boundary and colours it c_1 , and so on. Any of the l_i 's or c_j 's may be blank, in which case no label/color is assigned to the corresponding vertex. Using these new parse operators, we can generate the test sets for the atomic formulas (Figure 6.26).

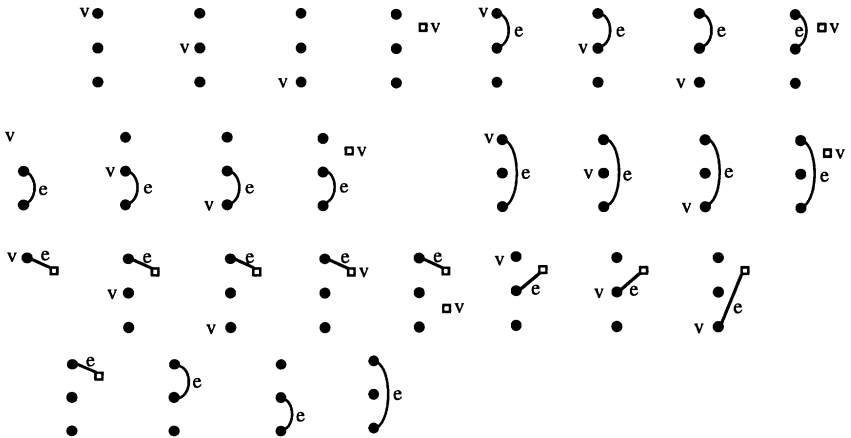
In the proof, use of $\oplus$ was restricted to X and Z such that if X was σ -partially equipped, then Z was $\bar{\sigma}$ -partially equipped. The definition of $\bar{\sigma}$ meant that X and Z had to have exactly the same boundary. In the implementation, we loosen this constraint to allow $\oplus$ to act on any two graphs X and Z as long as $X \oplus Z$ has a consistent interpretation for all variables in *free*.



Test set for $v \in A$, where v is a vertex variable and A is a vertex set variable



Test set for $e \in D$, where e is an edge variable and D is an edge set variable



Test set for $\text{inc}(e, v)$, where e is an edge variable and v is a vertex variable

FIGURE 6.26. Test Sets for Atomic Formulae on Three Boundaried Graphs

Test sets for more complex formulas are built up from the test sets for atomic formulas as follows:

- Given a test set for a formula ϕ , the test set for $\neg\phi$ is exactly the same, but we need to remember when constructing the automaton that accept states become nonaccept states and vice versa.
- Given test sets T_1 and T_2 for ϕ_1 and ϕ_2 , the test set for $\phi_1 \wedge \phi_2$ is $\{t_1 \oplus t_2 : t_1 \in T_1 \wedge t_2 \in T_2 \wedge t_1 \oplus t_2 \text{ is consistent}\}$.
- Given a test set for a formula ϕ , the test set for $\exists A$, where A is a variable free in ϕ , is obtained by applying the six parse operators to the elements of the test set until the test set becomes closed for the label or color corresponding to A . Now, remove the color or label corresponding to A from all members of the test set and delete any repeated elements. The test set can be minimized at this stage by employing the state minimization algorithm.

Using colors and labels and adding parameters too, the parse operators do not affect the final automaton because all variables in ϕ are quantified so there will be no colors or labels left in the test set for ϕ . Therefore, the elements in the test set for ϕ can be parsed using the standard parse operators.

The full implementation can be found as C+ code in Mike Doyle's thesis [186].

6.6 Seese's Theorem

In a pretty paper [415], Detleff Seese proved a converse to Courcelle's theorem.

Theorem 6.86 (Seese [415]) *Suppose that $\mathcal{F}$ is any family of graphs with a decidable monadic second-order (MS_2) theory. Then, there is a number n such that for all $G \in \mathcal{F}$, the treewidth of G is less than n .*

Notice the very interesting dichotomy: from linear time decidability the monadic second-order theory of bounded treewidth graphs to the undecidability of the monadic second-order theory for families of graphs failing to have a bound on the treewidth.

In this section, we sketch a proof of Seese's theorem. It relies on material from Robertson and Seymour and on some classical undecidability results.

We begin with *generalized grids* Q_n and Q_n^m . For $n \geq 2$ and $m \geq 0$, we define these as follows:

$$\begin{aligned}
 V(Q_n) &= \{(i, j) : 0 \leq i, j \leq n\}, \\
 E(Q_n) &= \{((i, j), (i', j')) : |i - i'| + |j - j'| = 1 \text{ and } 0 \leq i, i', j, j' < n\}, \\
 V(Q_n^m) &= \{(i, j) : 0 \leq i, j \leq (m+1)(n-1)\}, \\
 E(Q_n^m) &= \{((i, j), (i', j')) : 0 \leq i, i', j, j' \leq (m+1)(n-1) \\
 &\quad \wedge |i - i'| + |j - j'| = 1 \wedge [(i \equiv 0 \pmod{m+1}) \wedge i = i'] \\
 &\quad \vee (j \equiv 0 \pmod{m+1}) \wedge j = j']\}
 \end{aligned}$$

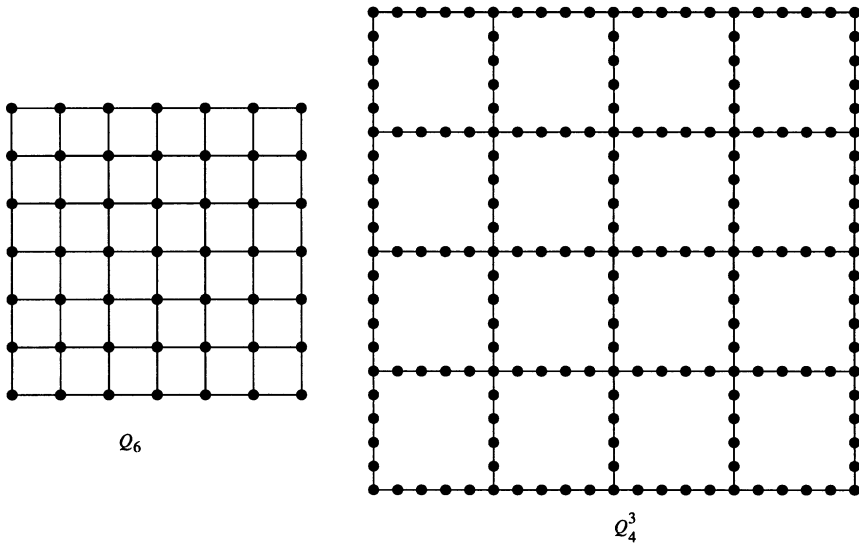


FIGURE 6.27. Examples of Grids

Clearly, Q_n is the $n \times n$ grid and Q_n^m is a grid with extra points. For instance, see Figure 6.27, where we exhibit Q_6 and Q_4^3 .

The following is an easy exercise left to the reader (Exercise 3).

Observation 6.87 *If G is any planar graph, then there is a number n such that $G \leq_{\text{minor}} Q_n$.¹²*

As well as the well quasi-ordering of graphs, Robertson and Seymour proved the following remarkable result.

Theorem 6.88 (Robertson and Seymour [393]) *Suppose that H is a finite planar graph, and $\mathcal{C}$ is a class of graphs no member of which has H as a minor. (In standard terminology, we say that $\mathcal{C}$ excludes H .) Then there is a number n such that for all $G \in \mathcal{C}$, the tree width of G is less than n .*

It follows that if a class has unbounded treewidth, then it contains graphs containing arbitrarily large grids Q_n as minors.¹³ Now in MS_2 for a fixed n , one can express the fact that a graph has Q_n as a minor (Exercise 4). The point is that this result means that the monadic second-order theory of grids can be interpreted in $\mathcal{C}$ if $\mathcal{C}$ is a class with unbounded treewidth. But one can code Turing machines or Undecidable Tiling Problems into this theory, the details being spelled out in Seese

¹²A graph G is said to be a *minor* of a graph H iff G can be obtained from H by a sequence of contractions and deletions. We refer the reader to Definition 7.9. We will discuss this concept in some detail in the next chapter.

¹³The proof of this result is beyond the scope of this book, but a related “excluded minor” theorem is treated in Section 7.4.

[414]. It follows that the MS_2 theory of $\mathcal{C}$ is undecidable if $\mathcal{C}$ contains graphs of arbitrarily large treewidth.

6.7 Notes on MS_1 Theory

The material on Courcelle’s theorem and Seese’s theorem both looked at the MS_2 theory based on the two-sorted language with predicate symbols for edges, vertices, and incidence. As we mentioned earlier, there is a long history concerning the “basic” monadic second-order theory where we have no predicate for edges but need binary relations for these objects. Now, the expressive power changes. For instance, “ G is Hamiltonian” is no longer expressible in our theory (Exercise 1).

Naturally, the methods we have used for Courcelle’s MS_2 theorem still work. What is unclear at the time of writing of this book is whether the analog of Seese’s theorem holds for MS_1 theories.

Seese was able to prove the following theorem.

Theorem 6.89 (Seese [415]) *If a class of planar graphs has a decidable MS_1 theory, then that class has uniformly bounded treewidth.*

To prove Theorem 6.89, Seese needed some intricate arguments to interpret the theory of the Q_n ’s into a given class of planar graphs of unbounded treewidth. The work build heavily on Seese [414], and Kuratowski’s theorem. (Theorem 7.7 of the next chapter.) The decisive lemma is the following.

Lemma 6.90 (Seese [415]) *Let $\mathcal{K}$ be any class of graphs such that for every planar graph H there is a planar $G \in \mathcal{K}$ with $H \leq_{\text{minor}} G$. Then, the monadic second-order theory (MS_1) theory of $\mathcal{K}$ is undecidable.*

The principle difficulty in the proof is then to prove that MS_1 interpretability occurs. This is quite intricate and heavily relies on planarity to be able to interpret the Q_n ’s into the class as minors. We refer the reader to [415] for details.

Subsequently, Courcelle and others have extended Theorem 6.89 to much wider classes of graphs. But it is still unknown whether Seese’s MS_2 theorem can be extended to MS_1 . The main idea in these other results is to replace Q_n by more intricate structures such as the Q_n^m to make up for the lack of expressive power. We refer the reader to Courcelle [141] for the current state of this question for both graphs and hypergraphs.

Historical Notes

The study of decidability results for monadic second-order theories of classes of combinatorial objects and their connections with automata has a long and interesting history, as can be found in Büchi [99]. Probably the first result is the one of Büchi [99] who proved that a language is regular iff it is definable by some formula in monadic second-order logic (of strings). The famous hallmark result here

is the decidability result of Rabin [379], with its simplified proof by Harrington and Gurevich [252].

Detleff Seese was the first to conjecture that classes of graphs with decidable theories were “similar to trees” (Seese [414]). Seese was able to prove that if a class of planar graphs with maximal degree ≤ 3 avoids certain graphs as induced subgraphs, then it has a very nice parse tree. Since Rabin [379] had proven the decidability of the monadic second-order theory of trees, Seese was able to deduce that these theories were decidable. Of course, Rabin’s method is not very feasible. Seese continued to pursue his conjecture that “treelike” graphs were the boundary of decidability culminating in the results of this section.

Courcelle began from a more syntactical viewpoint. Courcelle and Franchi-Zannettacci began to study the monadic second-order theories of graph classes in [145] where they studied “non-circularity properties of attribute grammars.” (The dependence graph of an attribute grammar is a good example of a context-free set of graph.) Such studies were extended in a universal algebra setting to general context-free graph grammars in Courcelle [133] and Bauderon-Courcelle [52]. In these articles, Courcelle and others were following the finite model theory path of enriching first-order logic with additional predicates (such as “path predicates”) and proving various recognizability properties.

The breakthrough was the idea to take the classical connection between Monadic Second-Order Logic and classes of languages and trees and seek a model in “tree-like” graphs. What was needed was a set of parse operations. In the mid-1980’s Courcelle was able to prove the principal lemma that the k -theory of a graph $f(G, H)$ is a function of the k -theories of the graphs G and H , where the k -theory of a structure is the (finite) set of monadic second-order formulas of quantifier depth at most k which are valid in the structure and f is an appropriate gluing operation. Actually, this is essentially the same as the Fefferman-Vaught theorem for first-order formulas and analogous results of Shelah [419] for monadic second-order formulas.

Courcelle’s results and their linear time decidability (for a fragment of Monadic Second-Order Formulas see [52, 133]) for graphs given by parse trees were presented in a conference [133]. The full paper is Courcelle [136]. Arnborg, Lagergren and Seese extended the basic Courcelle results to a wider class of formulas and to some minimization and maximization problems. Arnborg et al.’s results were presented to ICALP in 1988 [34] and appeared in final form in [34]. The “Basic Lemma” follows from a general one given in [140] for relational structures.

Courcelle’s results were obtained by the so-called method of interpretation. One obtains a correspondence between the two relevant logics and proposes a translation from one to the other. Early on, his methods and results were couched in the language of universal algebra.

Abrahamson and Fellows in [6, 7] were able to give straightforward automata-theoretic proofs of Courcelle’s theorems. This is the approach we use here. In fact, both of these papers contain flawed proofs of the Myhill–Nerode Theorem for graphs. These flaws are remedied here, as well as a consequential change in the

proof that **CUTWIDTH** is finite state. Results connecting finite-state graph properties to well-quasi-orderings appear in Abrahamson-Fellows [6, 7].

Seese's theorem appears in Seese [415].

It is still unknown if there is a full analog of Büchi's theorem [99] (that regularity equates to monadic second-order definability for formal languages) that holds for graphs. The conjecture of Courcelle is that definability in *Counting Monadic Second-Order Logic* equates to acceptance by Tree Automata for Graphs of bounded treewidth. Here, to get counting monadic second-order logic, one adds predicates of the form $\text{mod}_{p,q}$ for $p < q$ where $\text{mod}_{p,q}(V)$ has the interpretation true iff $|S| = p \bmod q$, where S is the set denoted by the set variable V . In a recent article, Kabinets [285] has verified the conjecture for graphs of bounded pathwidth, and the conjecture has been verified for treewidth ≤ 3 (Courcelle [137] for ≤ 2 and Kaller [286] for width 3).

Exercises 6.7

- 6.7.1.** Prove that hamiltonicity is not expressible in MS_1 .
- 6.7.2.** Construct a test set for the class of graphs that contain a cycle of length 4.
- 6.7.3.** Prove that if G is planar then there is some Q_n with $G \leq_{\text{minor}} Q_n$.
(Hint : Use induction.)
- 6.7.4.** Prove that $G \leq_{\text{minor}} H$ is MS_2 expressible for a fixed G .

7

Well-Quasi-Orderings and the Robertson-Seymour Theorems

7.1 Basic WQO Theory

As we will see, well-quasi-orderings (WQO's) provide a powerful engine for demonstrating that classes of problems are *FPT*. In this section, we will look at the rudiments of the theory of WQO's, and in subsequent sections, we will examine applications to combinatorial problems.

A *quasi-ordering* on a set S is a reflexive and transitive relation on S . We will usually represent a quasi-ordering as $\leq_S$ or simply $\leq$ when the underlying set S is clear. Let $\langle S, \leq \rangle$ be a quasi-ordered set. We will write $x < y$ if $x \leq y$ and $y \not\leq x$, $x \equiv y$ if $x \leq y$ and $y \leq x$, and, finally, $x \mid y$ if $x \not\leq y$ and $y \not\leq x$. Note that if $\langle S, \leq \rangle$ is a quasi-ordered set, then $S / \equiv$ is partially ordered by the quasi-order induced by $\leq$.¹

Definition 7.1 (Ideal and Filter) Let $\langle S, \leq \rangle$ be a quasi-ordered set. Let S' be a subset of S .

- (i) We say that S' is a *filter* if it is closed under $\leq$ upward; that is, if $x \in S'$ and $y \geq x$, then $y \in S'$. The *filter generated by S'* is the set $F(S') = \{y \in S : \exists x \in S' (x \leq y)\}$.²
- (ii) We say that S' is a (*lower*) *ideal* if S' is $\leq$ closed downward; that is, if $x \in S'$ and $y \leq x$, then $y \in S'$. The *ideal generated by S'* is the set $I(S') = \{y \in S : \exists x \in S' (x \geq y)\}$.

¹Recall that a *partial ordering* is a quasi-ordering that is also antisymmetric.

²Sometimes, filters are called *upper ideals*.

- (iii) Finally, if S' is a filter (an ideal) that can be generated by a finite subset of S' , then we say that S' is *finitely generated*.

We will need some distinguished types of sequences of elements.

Definition 7.2 Let $\langle S, \leq \rangle$ be a quasi-ordered set. Let $A = \{a_0, a_1, \dots\}$ be a sequence of elements of S . Then, we say the following.

- (i) A is *good* if there is some $i < j$ with $a_i \leq a_j$.
- (ii) A is *bad* if it is not good.
- (iii) A is an ascending *chain* if for all $i < j$, $a_i \leq a_j$.
- (iv) A is an *antichain* if for all $i \neq j$, $a_i \not\leq a_j$.
- (v) $\langle S, \leq \rangle$ is *Noetherian* if S contains no infinite (strictly) descending sequences (i.e., there is no sequence $b_0 > b_1 > b_2 \dots$).
- (vi) $\langle S, \leq \rangle$ has the *finite basis property* if for all subsets $S' \subseteq S$, $F(S')$ is finitely generated.

Theorem 7.3 (Folklore, after Higman [268]) *Let $\langle S, \leq \rangle$ be a quasi-ordered set. The following are equivalent:*

- (i) $\langle S, \leq \rangle$ has no bad sequences.
- (ii) Every infinite sequence in S contains an infinite chain.
- (iii) $\langle S, \leq \rangle$ is Noetherian and S contains no infinite antichain.
- (iv) $\langle S, \leq \rangle$ has the finite basis property.

Proof. (i) $\rightarrow$ (ii) and (ii) $\rightarrow$ (iii) are easy. (See Exercise 1.)
 (iii) $\rightarrow$ (iv). Suppose that $\langle S, \leq \rangle$ is Noetherian and S contains no infinite antichain. Let F be a filter of S with no finite basis. We generate a sequence from F in stages as follows. Let f_0 be any element from F . As F is not finitely generated, there is some $f_1 \in F$ with $f_1 \notin F(\{f_0\})$. For step $i + 1$, having chosen $\{f_0, \dots, f_i\}$, we find some $f_{i+1} \in F$ with $f_{i+1} \notin F(\{f_0, \dots, f_i\})$. Since F is not finitely generated, this process will not terminate. Notice that for all $i < j$, it cannot be that $f_i \leq f_j$ since, in particular, $f_j \notin F(\{f_i\})$. For $i < j$, color the pairs f_i, f_j *red* if $f_i > f_j$ and color the pair *blue* if $f_i \not\leq f_j$. (Every pair is colored either red or blue by the construction of the sequence.) By Ramsey's theorem,³ there is an infinite homogeneous set; that is there is a sequence $f_{i_0} f_{i_1} \dots$ and a color $\chi \in \{\text{red}, \text{blue}\}$, such that for all $i_k < i_l$, the pair f_{i_k}, f_{i_l} is colored χ . If χ is red, then we have an infinite de-

³Ramsey's Theorem (Ramsey [381]) states that if B is an infinite set and k is a positive integer, then if we colour the subsets of B of size k with colours chosen from $\{1, \dots, m\}$, then there is an infinite subset $B' \subseteq B$ such that all the size k subsets of B' have the same colour. B' is referred to a *homogeneous* subset.

scending sequence. If χ is blue, we have an infinite antichain. In either case, we contradict (iii).

(iv) $\rightarrow$ (i) Suppose that $\langle S, \leq \rangle$ has the finite basis property. Let $B = \{b_0, b_1, \dots\}$ be a sequence of elements of S . Let $B' \subseteq B$ be a finite basis for $F(B)$. Let $m = \max\{i : b_i \in B'\}$. Choose $b_j \in B - B'$ with $j > m$. Then, for some $b_i \in B'$, we must have $b_i \leq b_j$ since $b_j \in F(B')$, and $i < j$ by construction. Hence, B is good.

The several equivalent characterizations above lead to an important class of quasi-orderings.

Definition 7.4 (Well-Quasi-Ordering) Let $\langle S, \leq \rangle$ be a quasi-ordering. If $\langle S, \leq \rangle$ satisfies any of the characterizations of Theorem 7.3, then we say that $\langle S, \leq \rangle$ is a *well-quasi-ordering* (WQO).

The reader might well wonder what any of this abstract pure mathematics has to do with algorithmic considerations. The key is provided by the finite basis characterizations of a WQO. The answer is provided by the WQO principle below.

Well-Quasi-Ordering Principle Suppose that $\langle S, \leq \rangle$ is a WQO. Suppose further that for any x , the parameterized problem **ABOVE**(x) below is polynomial time computable. Then, for any filter F of $\langle S, \leq \rangle$, the decision problem “Is $y \in F$?” is polynomial time.

ABOVE(x)

Input: $y \in S$.
Parameter: $x \in S$.
Question: Is $x \leq y$?

The proof of the Well-Quasi-Ordering Principle is clear: If F is a filter, then F has a finite basis $\{b_1, \dots, b_n\}$. Then, to decide if $y \in F$, we need only ask “ $\exists i \leq n (b_i \leq_S y)$?” Since for each fixed i , **ABOVE**(b_i) is in P , the question “ $\exists i \leq n (b_i \leq_S y)$?” is in P too.

Actually, the Well-Quasi-Ordering Principle is often stated in a dual form for ideals. If F is a filter of $\langle S, \leq \rangle$, then $S - F$ is an ideal, and vice versa.

Definition 7.5 (Obstruction Set and Obstruction Principle)

- (i) Let $\langle S, \leq \rangle$ be a quasi-ordering. Let I be an ideal of $\langle S, \leq \rangle$. We say that a set $O \subseteq S$ forms an *obstruction set* for I if

$$x \in I \text{ iff } \forall y \in O (y \not\leq x).$$

That is, O is an obstruction set for I if I is the complement of $F(O)$.

- (ii) Suppose that $\text{ABOVE}(x)$ is in P for all $x \in S$, and that I has a finite obstruction set. Then, membership of I is in P too. Hence, we have the following Obstruction Principle: *If $\langle S, \leq \rangle$ is a wQO, then every ideal of I has a finite obstruction set, and furthermore if $\text{ABOVE}(x)$ is in P for all $x \in S$, every ideal has a P -time membership test.*

So here is our new engine for demonstrating that problems are in P . Prove that the problem is characterized by a finitely generated filter or ideal with a finite obstruction set in a quasi-ordering with $\text{ABOVE}(q)$ in P . Even better is to demonstrate that the ordering is a WQO.

The question arises: Are there interesting examples of such quasi-orderings? The answer is “absolutely.” Moreover, the demonstrations that some quasi-orderings are well-quasi-orderings will take us to some of the deepest theorems of mathematics.

The best known example of an obstruction set is provided by topological ordering.

Definition 7.6 (Topological Ordering) A *homeomorphic or topological embedding* of a graph $G_1 = (V_1, E_1)$ in a graph $G_2 = (V_2, E_2)$ is an injection from vertices V_1 to V_2 with the property that the edges E_1 are mapped to disjoint paths of G_2 . (These disjoint paths in G_2 represent possible *subdivisions* of the edges of G_1 .) The set of homeomorphic embeddings between graphs gives a partial order, called the *topological order*. We write $G_1 \leq_{\text{top}} G_2$.

The reader should refer to Figure 7.2. Although topological ordering is not a WQO, there are a number of important finite basis results. The most famous, and the archetype of all graph-theoretic finite basis results is Kuratowski’s theorem⁴ below.

Theorem 7.7 (Kuratowski’s Theorem, Kuratowski [316]) *The graphs $K_{3,3}$ and K_5 of Figure 7.1 form an obstruction set for the ideal of planar graphs in the topological ordering.*

Proof. We give this proof for completeness. We follow the argument of Dirac and Schuster [159] in the style of Harary [260]. We invite the reader to check that if

⁴Apparently, this result was independently discovered by Pontryagin. We refer the reader to Kennedy, Quintas, and Syslo [294].

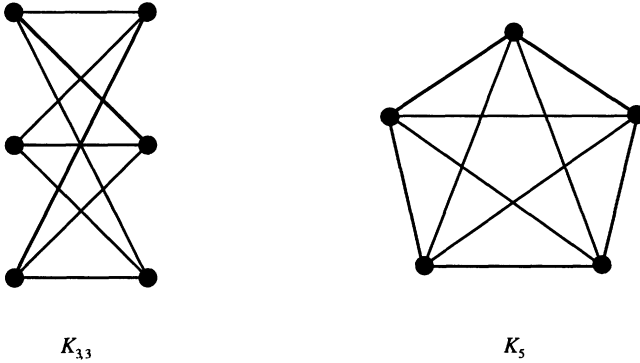


FIGURE 7.1. Kuratowski's Obstructions for Planarity

the graph has embedded copies of either K_5 or $K_{3,3}$, then it is nonplanar (Exercise 3). Now, suppose that G is a nonplanar graph with the minimal number of edges with no embedded copies of either K_5 or $K_{3,3}$. Note that G must have no cutpoints and must have minimal degree ≥ 3 . Let $e = uv$ be an edge of G and note that $G' = G - \{e\}$ is planar.

Claim. G' has a cycle containing e .

To establish the claim, assume that it fails. Then, there must be a cutpoint w of G' lying on each u, v path. Form G'' by adding to G' edges wu and wv if they are not already in G' . Define a *block* to be a maximal component with no cutpoints. Note that in the graph G'' , u and v still lie in differing blocks B_u and B_v . Let B be either B_u or B_v . Note that the minimality of G implies that either B is planar or it contains an embedded copy of either K_5 or $K_{3,3}$. However, in the latter case, if we can embed K_5 or $K_{3,3}$ into G'' , then we can also do so in G , by replacing the edge wu by the path from w to u in G . Therefore, both B_u and B_v are planar. Take a plane embedding of B_u and B_v , where the edges wu and wv occur on the boundary. Now, we can remove wu and wv , and put e back in the exterior to make a plane embedding of G , a contradiction. This establishes the claim.

Now, embed G' so that the cycle C containing u and v has the maximal number of regions interior to C possible. Also, we order the edges of the cycle in a cyclic fashion. Let $C[u, v]$ denote the path from u to v in C and similarly for $C[v, u]$. Let $C(x, y)$ denote the subpath resulting from $C[x, y]$ by removing x and y . We define the exterior of C to be the subgraph of G' induced by vertices lying outside of C . We define an *outer piece* of C to be the subgraph of G' induced by all edges incident with at least one vertex of a component of G' exterior to C or by an edge exterior to C meeting two vertices of C . We similarly define interior vertices, components, and pieces.

Definition 7.8 (u, v -separating) We call an outer piece or inner piece u, v -separating if it meets both $C(u, v)$ and $C(v, u)$.

Notice that since there are no cutvertices, each outer piece must meet C in at least two vertices. Moreover, no outer piece can meet $C(u, v)$ or $C(v, u)$ in more than one vertex. To see this, suppose that the outer piece R meets $C(u, v)$, say, in more than one vertex. Suppose that it meets $C(u, v)$ at x and y . Then, we could replace C by the cycle that is the same as C until it reaches x , then travels via R to y , and then copies C . The resulting C' would also be a cycle but would have more internal regions than C . Therefore, no outer piece can meet $C(u, v)$ or $C(v, u)$ in more than one vertex. Similarly, no outer piece can meet u or v . It follows that *every outer piece is u, v -separating and meets C in exactly two vertices $\neq u, v$* . Finally also note that since we cannot add the edge $e = uv$ and have a planar graph, there must be at least one u, v -separating outer piece.

Claim. *There is a u, v -separating outer piece meeting $C(u, v)$ at u' and $C(v, u)$ at v' so that there is an inner piece which is both u, v -separating and u', v' -separating.*

To see that the claim holds suppose that it fails. Consider the u, v -separating inner piece I_1 nearest to u in the sense of encountering inner pieces moving along C from u . Continuing from u , we can thus order inner pieces $I_1, I_2, \dots$

Let f and l be the first and last points of I_1 meeting $C(u, v)$ and f' and l' the first and last points of I_1 meeting $C(v, u)$. Notice that every outer piece has both of its common vertices with C on either $C[l', f]$ or $C[l, f']$. [Otherwise there would be an outer piece meeting $C(u, v)$ at u' and $C(v, u)$ at v' which is both u, v -separating and u', v' -separating.] Therefore, we can draw an edge E joining l' to f in the exterior of C . It follows that we can transfer I_1 out of the interior of C in a plane manner. Thus, *all* the remaining u, v -separating inner pieces can be transferred outside of C in a plane manner. But then we can add the edge $e = uv$ to G' and the resulting graph will be plane, a contradiction. This contradiction establishes the claim.

To conclude the proof, let H be an inner piece given by the claim. (So, H is both u, v -separating and u', v' -separating.) Let w_0, w_1 and w'_0, w'_1 be the vertices at which H meets, respectively, $C(u, v)$, $C(v, u)$ and $C(u', v')$, $C(v', u')$.

There are now four cases to consider, depending upon the relative positions of the vertices.

Case 1. *One of the vertices w'_0 or w'_1 is on $C(u, v)$.*

In this case, we can contract w_0 to w'_0 and w_1 to w'_1 (and hence regard $w_0 = w'_0$ and $w_1 = w'_1$) so that G embeds $K_{3,3}$ via the vertex sets $\{u, v, v'\}$ and $\{w_0, w_1, u'\}$.

The other three cases are as follows:

Case 2. *Both of the vertices w'_0 and w'_1 are on either $C(u, v)$ or on $C(v, u)$.*

Case 3. $w'_0 = v$ and $w'_1 \neq u$.

Case 4. $w'_0 = v$ and $w'_1 = u$.

We leave these cases to the reader (Exercise 4). (K_5 can only arise in the final case.)

Naturally, we would like to know if the following problem is fixed-parameter tractable.

TOPOLOGICAL CONTAINMENT

Instance: A graph $G = (V, E)$.

Parameter: A graph H

Question: Is H topologically contained in G ?

We remark that **TOPOLOGICAL CONTAINMENT** can be solved in $O(n^{O(|E|)})$ time by brute force together with the k -DISJOINT PATHS algorithm of Robertson and Seymour [397]. Nevertheless, for the example of $H \in \{K_5, K_{3,3}\}$, $H \leq_{top} G$ is, in fact, linear time decidable. Hence, **PLANARITY** is $O(|G|)$. We conjecture that **TOPOLOGICAL CONTAINMENT** is, in fact, in *FPT*.

It turns out that $\leq_{top}$ is not the most amenable ordering for graphs, both algebraically and algorithmically. For instance, it is not a **WQO**. (See Exercise 5.) An equivalent formulation of $\leq_{top}$ is the following. $G \leq_{top} H$ iff G can be obtained from H by a sequence of the following two operations.

- (i) (deletions) Deleting vertices or edges.
- (ii) (degree 2 contractions) The *contraction* of an edge xy in a graph W is obtained by indentifying x with y . A contraction has degree 2 iff one of x or y has degree 2.

The *minor ordering* is a generalization of $\leq_{top}$ is obtained by relaxing the degree 2 requirement in (ii).

Definition 7.9 (Minor Ordering) We say that G is a *minor* of H if G can be obtained from H by a sequence of deletions and contractions. We write $G \leq_{minor} H$.

An example of the minor ordering is given in Figure 7.2. The proof of Kuratowski's theorem also gives the following.

Theorem 7.10 (Kuratowski's Theorem (II)) $K_{3,3}$ and K_5 are an obstruction set for the ideal of planar graphs under $\leq_{minor}$.

Kuratowski's theorem was so beautiful that it inspired many generalizations and conjectures. Wagner [461] conjectured the following.

Wagner's Conjecture: Finite graphs are well-quasi-ordered by the minor ordering.

Notice that graphs of genus $\leq g$ for a fixed g form an ideal in the minor ordering. Hence, a consequence of Wagner's conjecture is a Kuratowski theorem for surfaces. *Graphs of genus $\leq g$ have a finite obstruction set.* One of the triumphs of 20th century mathematics is the following great theorem of Neal Robertson and Paul Seymour.

Theorem 7.11 (Graph Minor Theorem, [397]) *Wagner's conjecture holds: Finite graphs are well-quasi-ordered by the minor ordering.*

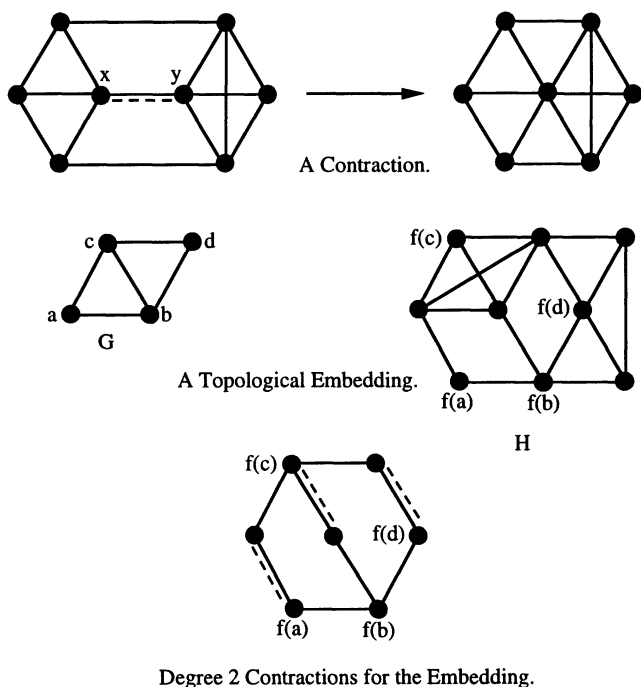


FIGURE 7.2. A Contraction and a Topological Embedding

The proof of Theorem 7.11 is spread over 22 articles and at the present time has not yet fully appeared. To prove Theorem 7.11, Robertson and Seymour needed to invent a whole new graph structure theory. A small part of this theory was the invention of the notion of treewidth. In the previous chapters, we have already seen the importance of the notion of treewidth.

Robertson and Seymour also demonstrated the startling algorithmic consequences of the methods used to prove Theorem 7.11 by proving the following.

Theorem 7.12 (Robertson and Seymour [396]) **MINOR ORDER TEST** *below is $O(n^3)$ for any fixed graph H and hence FPT.*

MINOR ORDER TEST

Instance: Graphs $G = (V, E)$ and $H = (V', E')$.

Parameter: A graph H .

Question: Is $H \leq_{\text{minor}} G$?

Thus, by the Obstruction Principle, *any* ideal in the minor ordering will not only be a language in P but will have an $O(n^3)$ membership test. The following example was quoted in Chapter 2.

GRAPH GENUS

Instance: A graph $G = (V, E)$.
Parameter: A positive integer k .
Question: Does G have genus k ?

(That is, can G be embedded with no edges crossing on a surface with k handles?)

We should emphasize that the Graph Minor Theorem is *intrinsically nonconstructive*. By this, we mean that there is no algorithm which, when given an index for a minor closed ideal $\mathcal{I}$, will compute an obstruction set for $\mathcal{I}$.

Theorem 7.13 (Fellows and Langston [216, 220]) *There is no algorithm A which, when given an index e for a minor closed ideal $\mathcal{I}$, outputs $A(e)$, an obstruction set for $\mathcal{I}$. (It is not important what A does on those e which are not indices for minor closed ideals.)⁵*

Proof. Let φ_e denote the e -th partial recursive function. Let $\varphi_{e,s}(j)$ denote the result of computing s steps in the computation of φ_e on input j . Now, define $\Phi_e(j)$ to be the least s if any, such that $\varphi_{e,s}(j) \downarrow$ but $\varphi_{e,s-1}(j) \uparrow$. Let $G_0, G_1, \dots$ be a computable enumeration of finite graphs such that if $G_i \leq_{\text{minor}} G_j$, then $i < j$. Let

$$d(e, s) = \begin{cases} 1, & \text{if } \varphi_{e,s}(e) \uparrow \text{ or } (\Phi_e(e) < s \wedge G_{\Phi_e(e)} \not\leq_{\text{minor}} G_s) \\ 0, & \text{otherwise.} \end{cases}$$

Then, the sets $\mathcal{I}_e = \{G_s : d(e, s) = 1\}$ evidently form an ideal in the minor ordering, and we can compute an index for $\mathcal{I}_e$ for each e . However, the obstruction set for $\mathcal{I}_e$ is $\emptyset$ iff $\varphi_e(e) \uparrow$. [If $\varphi_e(e) \downarrow$, then $\{G_{\Phi_e(e)}\}$ serves as an obstruction set for $\mathcal{I}_e$.] Hence, any algorithm to compute obstruction sets when applied to the collection $\mathcal{I}_e : e \in \mathbb{N}$ will result in an algorithm for the halting problem.

The Graph Minor Theorem thus provides an important philosophical challenge to the notion that polynomial time is a “good” measure of *practical tractability*. Here, we can show that a practical problem is in P , but we have no way of even knowing the polynomial-time algorithm solving the problem. Furthermore, the news is even worse. Even when we can compute the obstruction sets, the *constants* in Theorem 7.12 are gigantic (around 2^{500}), and evidence suggests that the sizes of the obstruction sets is very large. At the time of the writing of this book, an exact obstruction set for embedding graphs on the torus is unknown. These facts caused a prominent computer scientist to remark, echoing the famous words of Gordon, “This is not computer science, this is a mathematical curiosity!”

We do not agree. We believe that the Graph Minor Theorem is of immense value to computer science. At the very least, an application of the Graph Minor Theorem to yield a $O(n^3)$ algorithm for some natural problem surely demonstrates *potential* practical tractability. (Sometimes, this happens to problems which were previously not even known to be effectively solvable.) The claim that P is a good

⁵It is also possible to establish this result via Rice’s theorem.

measure of tractability is supported by the fact that it has good closure properties and, more importantly, “Practical Algorithm Builders” believe that, “If a natural problem is in P , then it has a low degree, small constant algorithm demonstrating its membership in P .”

It seems to us that if this heuristic principle is correct, then an application of the Graph Minor Theorem to prove a natural problem is in P is an important first step in the gradual discovery of a truly practical algorithm for the problem. For instance, inspired by the Robertson and Seymour result, Mohar [354] has recently demonstrated a constructive and feasible linear time algorithm for **GRAPH GENUS**. On the other hand, if this heuristic principle is *not* correct, then the Graph Minor Theorem must surely provide a good vehicle for the search for counterexamples.

In any case, we believe that there are two *reasons* for the noncomputability and impracticality of the Graph Minor Theorem. The first reason stems from its great generality. What is needed is a general program to develop techniques to extract truly practical algorithms for classes of natural problems. The second is the insistence that we need to work from a machine description for membership of the class at hand. One important point is that for practical problems, one usually has a lot more combinatorial information than merely a decision procedure. A similar situation can be found in regular languages where if we are only provided with a machine description for a class of languages, then, in general, there is no process which effectively yields automata. But in natural languages, we usually have *much more* than merely machine descriptions of the languages. Indeed, in the case of formal languages, we usually have a regular expression describing the language. We believe that similar considerations will apply to natural applications of the Graph Minor Theorem. For instance, we will often have some nice syntactic description of the class.⁶

We will look at some applications and effectivizations of the Graph Minor Theorem in Section 7.8. Later, we will look at other WQO theorems such as Robertson and Seymour’s result that the *immersion order* is also a WQO (Section 7.6).

A proof of the Graph Minor Theorem is currently beyond the scope of this book. In the next three sections, we will give a proof of the Graph Minor Theorem for graphs of bounded treewidth, and we give a brief sketch of the full Graph Minor Theorem. We will finish the present section by proving the archetype of all WQO results for graphs, Kruskal’s Theorem for trees. We remark that Kruskal’s Theorem and its consequences are very important for what follows.

Theorem 7.14 (Kruskal’s Theorem [309]) *The class of rooted finite trees are well-quasi-ordered by $\leq_{top}$ and by $\leq_{minor}$. [Here by $\leq_{top}$ we mean **rooted topological containment** where we require that for rooted trees T_1, T_2 , if $T_1 \leq_{top} T_2$ via the topological embedding f and v and w are vertices of branches of T_1 meeting at z , then $f(v)$ and $f(w)$ are vertices of branches of T_2 meeting at $f(z)$ in T_2 .]*

⁶However, Courcelle, Downey, and Fellows [162] have recently shown that one cannot compute an obstruction from a monadic second order description of the ideal.

Proof. We can consider trees as 1-trees and hence as 1-boundaried graphs. Trees can be parsed by the operators $\oplus$ and 1, where 1 is the operator which adds a vertex, makes it a boundary vertex, and joins an edge from the old boundary vertex to the new one (Exercise 2).

Suppose that $T_0, T_1, \dots$ is a minimal bad sequence of trees; that is we suppose that we have chosen T_0 to be the smallest tree (by cardinality) to begin a bad sequence, and from the bad sequences of trees beginning with T_0 , we have chosen T_1 to be smallest, etc. Let P_i be a corresponding parse tree for T_i . Either there are infinitely many P_i with root $\oplus$ or almost all P_i have root 1. Both cases are easy. Suppose that $P_{i_0}, P_{i_1}, \dots$ all have root $\oplus$. Now, for a parse tree P , let $T(P)$ denote the tree obtained from P . Let Q_i and R_i denote the parse trees obtained by deleting the roots from P_i , and consider the collection

$$\mathcal{C} = \cup_j (T(Q_{i_j}) \cup T(R_{i_j})).$$

We claim that $\langle \mathcal{C}, \leq_{top} \rangle$ is a WQO. If this is not the case, then there is a bad sequence $B_0, B_1, \dots$ from $\mathcal{C}$. For some least k , $B_0 \in \{Q_k, R_k\}$. Then, the sequence $T_0, T_1, \dots, T_{k-1}, B_0, B_1, \dots$ is bad, contradicting the minimality of $T_0, T_1, \dots$. Hence, $\langle \mathcal{C}, \leq_{top} \rangle$ is a WQO. It is easy to prove that $\mathcal{C} \times \mathcal{C}$ is well-quasi-ordered by the product ordering [i.e., $(W_1, V_1) \leq_{top} (W_2, V_2)$ iff $W_1 \leq_{top} W_2 \wedge V_1 \leq_{top} V_2$]. But then there are $m < n$ with

$$T(Q_{i_m}) \leq_{top} T(Q_{i_n}) \wedge T(R_{i_m}) \leq_{top} T(R_{i_n}).$$

Clearly, this implies that $T_{i_m} \leq_{top} T_{i_n}$. The 1 case is similar but easier.

Let $\langle Q, \leq_Q \rangle$ be a quasi-ordered set. Extend the definition of topological ordering to labeled topological ordering by saying that a Q -labeled tree T_1 topologically embeds into a labeled tree T_2 via f by asking that f takes edges to disjoint paths, and furthermore, if v has label q and $f(v)$ has label q' , then $q \leq_Q q'$. The proof above easily extends to the following.

Corollary 7.15 (Kruskal's Theorem for Labeled Trees, [309]) *If $\langle Q, \leq \rangle$ is a WQO, then the collection of Q -labeled trees is a WQO under labeled topological embedding.*

The main modification in the proof is that one argues from an infinite sequence of roots $\{q_{i_0}, q_{i_1}, \dots\}$ with the property $q_{i_0} \leq_Q q_{i_1} \leq_Q \dots$ which are guaranteed to exist by the fact that Q is a WQO. We remark that the standard proofs of Kruskal's Theorem do not use parse trees. Rather, they use the following lemma which is of importance in its own right. (Also see Exercise 7.)

Let $\langle S, \leq \rangle$ be any quasi-ordered set. One can define the induced ordering on $\mathcal{S}$, the set of finite sequences of elements of S via $\langle s_1, \dots, s_n \rangle \leq \langle s'_1, \dots, s'_m \rangle$ iff there is a strictly increasing map $j : \{1, \dots, n\} \rightarrow \{1, \dots, m\}$ such that for all $i \leq n$,

$$s_i \leq s'_{j(i)}.$$

For instance, $\langle 1, 2, 1, 4 \rangle \leq \langle 0, 2, 2, 0, 1, 6, 1 \rangle$, but $\langle 1, 2, 1 \rangle \not\leq \langle 0, 1, 1, 2, 0, 0 \rangle$.

Lemma 7.16 (Higman's Lemma, [268]) *Suppose that $\langle S, \leq \rangle$ is a WQO. Then, the induced ordering is a well-quasi-ordering of S .*

Proof. We can give an easy proof of this lemma by using the labeled version of Kruskal's theorem. We can consider a sequence $\langle a_0, a_1, \dots, a_n \rangle$ as a tree with root labeled a_0 , immediate successor $a_1, \dots$. To a bad S sequence would correspond a sequence of trees, bad with respect to labeled topological ordering.

Before we leave our section on basic WQO theory, we mention two further WQO results whose proofs are beyond the scope of the present book. The *induced* minor ordering is the same as the normal minor ordering except only vertex deletions and contractions are allowed.

Theorem 7.17 (Thomas [443]) *Finite graphs of treewidth ≤ 2 are well-quasi-ordered by the induced minor ordering.*

We remark that the bound 2 is sharp in Theorem 7.17 (Exercise 10).

Finally, we turn to a WQO result in a different arena. Let P and Q be partially ordered sets (posets). We say that P is a *chain minor* of Q , $P \ll Q$ iff there exists a partial mapping $f : Q \mapsto P$ such that for each chain C_P of P , there is a chain C_Q of Q such that $f \upharpoonright C_Q$ is an isomorphism of chains from C_Q to C_P .

Theorem 7.18 (Gustedt [255]) *Finite posets are well-quasi-ordered by $\ll$.*

Gustedt noted the following.

Theorem 7.19 (Gustedt [255])

- (i) *Let P and Q be finite posets with $c = |P|$. (Assume that $c > 3$.) Then there is a constant l depending only upon P and an algorithm A which decides if $P \ll Q$ running in time $O(c^2|Q|^2 + l)$.*
- (ii) *Consequently, if $\mathcal{F}$ is any class of finite posets closed under the chain minor ordering, then $\mathcal{F}$ has a polynomial-time recognition algorithm.*

Proof. Exercise 9.

Theorem 7.19 leaves open the following question. Is the following problem FPT?

CHAIN MINOR ORDERING

Input: A finite poset Q .
Parameter: A finite poset P .
Question: Is P a chain minor of Q ?

Exercises 7.1

7.1.1. Prove that (i) implies (ii), and (ii) implies (iii) in Theorem 7.3.

7.1.2. Prove that we can parse trees with the operators $\oplus$ and 1 ; that is, prove that we can consider trees as 1-boundaried graphs which can be parsed

by the operators $\oplus$ and 1, where 1 is the operator which adds a vertex, makes it a boundary vertex, and joins an edge from the old boundary vertex to the new one. (Use induction.)

7.1.3. Prove that if the graph has embedded copies of either K_5 or $K_{3,3}$, then it is nonplanar.

7.1.4. Verify that one can embed K_5 or $K_{3,3}$ in the remaining cases of Theorem 7.7.

7.1.5. Prove that $\leq_{top}$ is not a WQO.

(Hint. Consider n circles each consisting of four points $\{x_i, y_i, z_i, q_i\}$ with edges $\{x_i y_i, y_i z_i, z_i, q_i, q_i x_i\}$, joined so that $q_i = x_{i+1}$.)

7.1.6. Give a direct proof of Higman's Lemma.

(Hint: Suppose that $\langle S, \leq \rangle$ is a WQO. Then, we need to prove that the induced ordering is a well-quasi-ordering of $\mathcal{S}$. Suppose that $\mathcal{S}$ is not well-quasi-ordered by the induced ordering $\leq$. Let $a_0, a_1, \dots$ be a minimal bad sequence for $\mathcal{S}$; that is, we suppose that $a_0, a_1, \dots$ is a bad sequence of finite sequences of members of S , chosen so that:

- (i) a_0 is the sequence of minimal length which begins any bad sequence.
- (ii) Among bad sequences beginning with a_0 , we choose a_1 to have minimal length, etc.

Note that if $a_0, a_1, \dots, a_{i-1}, b_i, b_{i+1}, \dots$ is a bad sequence, then $|a_i| \leq |b_i|$. Let q_i denote the final term of a_i . Since $\langle S, \leq \rangle$ is a WQO, there is an infinite collection,

$$q_{i_0} \leq q_{i_1} \leq \dots$$

Now, let $a'_i : i \in \mathbb{N}$ denote the collection obtained from $a_i : i \in \mathbb{N}$ by deleting q_i . Then, by the minimality condition, the sequence

$$a_0, a_1, \dots, a_{i_0-1}, a'_{i_0}, a'_{i_1}, \dots$$

is good. Therefore, there exist $m < n$ with $a'_{i_m} \leq a'_{i_n}$. Since $q_{i_m} \leq q_{i_n}$, we have that $a_{i_m} \leq a_{i_n}$ by definition, contradicting the badness of the sequence $a_0, a_1, \dots$

7.1.7. Without using parse trees, give a direct proof of Kruskal's theorem via Higman's Lemma.

(Hint: (Nash-Williams [360]) Let $\mathcal{T} = T_1, T_2, \dots$ be a bad sequence, minimal by size. Consider the sequence of forests

$$(T_1^1, \dots, T_1^{n_1}), (T_2^1, \dots, T_2^{n_2}), \dots$$

obtained by removing the root from each T_i . Use minimality of $\mathcal{T}$ to prove that $\cup_{i,j} T_j^i$ is well-quasi-ordered by topological ordering. Now, prove that $\mathcal{T}$ is not bad by using Higman's Lemma.)

7.1.8. Let $\langle S, \leq \rangle$ be a quasi-order. Let S^ω denote the collection $\{(x_0, x_1, \dots) : x_i \in S\}$ of countably infinite sequences of members of S . Then, we can extend $\leq$ to S^ω in the obvious way: $(x_0, x_1, \dots) \leq (y_0, y_1, \dots)$ iff

there is an increasing (1-1) function $f : \mathbb{N} \mapsto \mathbb{N}$ such that for all $i \in \mathbb{N}$, $x_i \leq y_{f(i)}$.

Give an example of a **WQO** $\langle S, \leq \rangle$ with S^ω not well-quasi-ordered by the induced ordering.

(Hint: [Rado]⁷ Consider $\mathcal{R} = \{(i, j) : i < j < \omega\}$, partially ordered by

$$(i, j) \leq (i', j') \text{ iff } j \leq i' \text{ or } i = i' < j \leq j'.$$

7.1.9. (Gustedt [255]) Prove Theorem 7.19(i).

(Hint: Prove that if $P \ll Q$, then there is a poset P' with $P \ll P'$, P' an induced subordering of Q , and $|P'| \leq |Q|$.)

7.1.10. Prove that there is an infinite bad sequence of finite graphs under the induced minor ordering.

7.1.11. (folklore) Prove that if F is a class of finite graphs with a finite obstruction set under $\leq_{\text{minor}}$, then it has a finite obstruction set under $\leq_{\text{top}}$.

7.2 Thomas' Lemma

The first proof of the well-quasi-ordering of graphs of bounded treewidth was in the original (unpublished) version of Robertson and Seymour [391]. It was a long and complex argument. Subsequently, the argument was considerably simplified and extended by Robin Thomas [442, 443] and Kriz and Thomas [310, 311]. The Thomas approach was based on an important theorem (Thomas' Lemma) which said that graphs of bounded treewidth had some very well-behaved tree decompositions. Thomas' Lemma was actually used in the final published version of Robertson and Seymour's Graph Minors IV [391] since it simplified Robertson and Seymour's proof so much. This section is devoted to giving Thomas' proof.

The reader should recall Definition 6.50 of a tree decomposition as a tree labeled with subsets with each edge included in some bag and the interpolation property. For subsets X and Y of the vertices of G , let $c(X, Y)$ denote the *separation number* of X and Y . That is the size of the smallest set (cutset) A of vertices of G such that each path from X to Y uses a member of A .

Definition 7.20 A tree decomposition $\{V_i : i \in \mathcal{T}\}$ of a graph G is called a *Thomas decomposition* if additionally the *Thomas property* below holds for the decomposition.

- For any subsets $V'_i \subseteq V_i$ and $V'_j \subseteq V_j$ of bags V_i and V_j , if $c(V'_i, V'_j) < \min\{|V'_i|, |V'_j|\}$, then there is a bag V_q between V_i and V_j in $\mathcal{T}$, with $|V_q| \leq c(V'_i, V'_j)$.

⁷Rado actually proved that this example is canonical in the sense that if Q is a **WQO** such that Q^ω is *not* a **WQO**, then the quasi-ordering $\langle S, \leq \rangle$ of this hint will embed into Q .

Actually, the conclusion of the above statement can be replaced by " $|V_q| = c(V'_i, V'_j)$ " via the next lemma, whose proof is left to the reader (Exercise 5).

Lemma 7.21 (Folklore, e.g. Thomas [442]) *Suppose that $\{V_i : i \in T\}$ is a tree decomposition and P is a path in G which meets bags V_i and V_j in G . Then, P meets any bag V_q between V_i and V_j in T .*

Theorem 7.22 (Thomas' Lemma [442, 443]) *A treewidth k graph has a treewidth k Thomas decomposition.*

Thomas' lemma can be viewed as an extension for graphs of bounded treewidth of the following classical result, which we shall need for our proof of Thomas' lemma. For vertices v and w of a graph G , we say that v is k -connected to w in G if it requires the removal of at least k vertices from G to disconnect v from w . We can similarly define k -edge-connected by replacing "vertices" by "edges" in the above definition.

Lemma 7.23 (Menger's Theorem [350]) *Let G be a (directed) graph and v and w be vertices of G . Then, there exist k (vertex) (respectively edge) disjoint paths in G connecting v to w iff v is k -connected (respectively k -edge-connected) to w in G . Furthermore, if G is k -vertex-connected, then for any k -vertex cutset $S = \{v_1, \dots, v_k\}$ separating v and w in G and any disjoint paths $P_1, \dots, P_k$, there exists a permutation π of $\{1, \dots, k\}$ such that with $v_i \in P_{\pi(i)}$ for $i = 1, \dots, k$.*

Proof. We begin with the edge case. We will only consider the undirected case since it suffices for our purposes. As with many important algorithmic theorems, this is a consequence of the Max Cut/Min Flow Theorem (Exercise 1). Apply Max Cut/Min Flow to the graph obtained by replacing each edge in G containing v as directed out of v , each edge containing w as one into w , and replacing all other edges by two directed edges, one each way. Give all edges in the graph so derived weight 1. Now, apply the Max Cut/Min Flow Theorem, noting that a minimum cut corresponds to the edge connectivity. For the vertex case, take G and replace each vertex $u \notin \{v, w\}$ by a new edge e_u and then apply the edge case to the graph G' so construed.

For the proof of Thomas' Lemma, the following definition is quite useful.

Definition 7.24 Let T be a tree with i and j nodes of T . Then, the *subtree of T extending node j determined by i* is the collection of nodes

$$\{k : j \in [i, k]_T\}.$$

For instance, in the tree (path) T with vertices $\{1, 2, 3, 4, 5\}$ of Figure 7.3, the subtree of T extending 3 determined by 1 is $\{3, 4, 5\}$.

Proof of Theorem 7.22. We follow Thomas [442]. An (n, d) -cell is a (connected) subtree of T of at least d bags, all of which have at least n vertices. Let $c_{n,d}$ denote

the number of (n, d) -cells in $\mathcal{T}$. The size of a tree decomposition is the sequence of numbers

$$c_{n,d} : n, d \in \mathbb{N}.$$

We order the sizes lexicographically. Naturally, this is a well-founded ordering among treewidth k decompositions. Let $\{V_i : i \in \mathcal{T}\}$ be a tree decomposition minimal with respect to size.

Main Claim $\{V_i : i \in \mathcal{T}\}$ is a Thomas decomposition.

The remainder of the section is devoted to a proof of the Main Claim above. Suppose that the Thomas property fails for $\{V_i : i \in \mathcal{T}\}$. Choose bags $V'_i \subseteq V_i$, $V'_j \subseteq V_j$ witnessing the failure of the Thomas property so that the $c(V'_i, V'_j) = m$, say, is chosen to be as small as possible, and then the path $[i, j]$ from i to j in $\mathcal{T}$ has been chosen as short as possible for m . Now, take a set A of vertices of G separating V'_i from V'_j of size $m = c(V'_i, V'_j)$, chosen so that $\sum_{x \in A} d_x$ is minimal, where d_x is the minimum of the distances in $\mathcal{T}$ of any bag containing x and bags in $[i, j]$. (So, for instance, if x belongs to a bag in $[i, j]$, then $d_x = 0$.)

Take sets F_i, F_j with F_i, F_j , and A pairwise disjoint so that (i)-(iii) below hold.

- (i) $F_i \cup F_j \cup A = V(G)$
- (ii) A separates F_i and F_j
- (iii) $V'_i \subseteq F_i \cup A$ and $V'_j \subseteq F_j \cup A$.

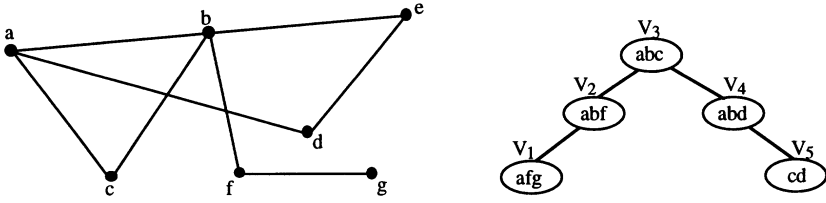
By Menger's theorem (Theorem 7.23), we know there are exactly m disjoint paths $P_1, \dots, P_m$ between V'_i and V'_j , each using exactly one vertex a_t of A .

Now, we form a new tree $\{W_s : s \in \widehat{\mathcal{T}}\}$ of bags as follows. Take two copies of $\mathcal{T}$, say $\mathcal{T}_i$ and $\mathcal{T}_j$. Let $\pi_q : \mathcal{T}_q \mapsto \mathcal{T}$ denote the relevant isomorphism. We assume that the vertices of the trees are disjoint. To form $\widehat{\mathcal{T}}$, combine $\mathcal{T}_i$ and $\mathcal{T}_j$ and add one further edge $\widehat{e}$ from vertex $\pi^{-1}(i)$ (corresponding to i) in $\mathcal{T}_j$ to vertex $\pi^{-1}(j)$ in $\mathcal{T}_i$. To finish the specification of $\widehat{\mathcal{T}}$, we only need to specify the bags W_s . Let $s \in V(\mathcal{T}_q)$. For clarity, let $q = i$ (the roles of i and j being reversed in the case that $q = j$). Define

$$W_s = (F_i \cap V_{\pi_i(s)}) \cup A_s,$$

where A_s denotes the set of all $a_t \in A$ with $a_t \in \cup\{V_n : n \text{ is on the subtree of } \mathcal{T} \text{ extending } \pi_i(s) \text{ determined by } j\}$ such that the corresponding path P_t includes at least one element of $(F_j \cup A) \cap V_{\pi_i(s)}$. (Refer to Definition 7.24.) Figure 7.3 gives an illustration of the construction of $\{W_s : s \in \widehat{\mathcal{T}}\}$.

The reader should study this construction carefully since it is the key to the remainder of the proof. The main idea is that A , being a Menger's theorem generated cutset, splits G into the two A -separated pieces F_i and F_j . We take two copies of $\mathcal{T}$ and basically generate the new bags by intersecting the copy $\mathcal{T}_q$ with F_q . Additionally, we put in the bag $W_{\pi_q^{-1}(r)}$ on $\mathcal{T}_q$ corresponding to node q in $\mathcal{T}$, possibly adding some additional vertices of A (this is the set $A_{\pi_q^{-1}(r)}$) as determined by the path P_r in $\mathcal{T}$. The fact that paths must run through A will easily allow us to prove that the new tree $\widehat{\mathcal{T}}$ of bags contains all vertices and each edge is represented in



Parameters: $i=2, j=3, V_2=\{f,b\}, V_3=\{a,b\}, c(V_2', V_3)=1, A=\{b\}, P_b=b, F_2=\{f,g\}, F_3=\{a,c,e,d\}$

Notation: $\prod_q(p)$ is the node representing V_p on T_q ; $A_{q,p}$ similarly.

Calculations of $A_{q,p}$

$A_{2,p} = \{b\} \cap \{V_t : t \text{ is on the subtree extending node } p \text{ determined by } V_3\}$ with

$P_b = \{b\}$ meeting $[(F_3 \cup A) \cap V_p = \{b,a,c,e,d\} \cap V_p] \ (\equiv V_p \cap \{b\} \neq \emptyset)$

$A_{2,1} = \emptyset$

$A_{2,2} = \{b\} \cap V_2 = \{b\}$

$A_{2,3} = \{b\} \cap (V_2 \cup V_3 \cup V_4) = \{b\}$

$A_{2,4} = \{b\} \cap V_4 = \{b\}$

$A_{2,5} = \emptyset$

$A_{3,p} = \{b\} \cap \{V_t : t \text{ is on the subtree extending the node } p \text{ determined by } V_2\}$ with

$P_b = \{b\}$ meeting $[(F_2 \cup A) \cap V_p = \{b,f,q\} \cap V_p] \ (\equiv V_p \cap \{b,f\} \neq \emptyset)$

$A_{3,1} = \{b\} \cap V_1 = \emptyset$

$A_{3,2} = \{b\} \cap (V_1 \cup V_2 \cup V_3 \cup V_4) = \{b\}$

$A_{3,3} = \{b\} \cap (V_3 \cup V_4) = \{b\}$

$A_{3,4} = \{b\} \cap V_4 = \{b\}$

$A_{3,5} = \emptyset$

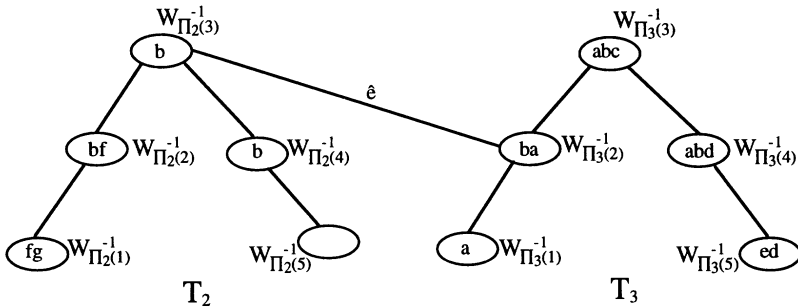


FIGURE 7.3. Construction of the New Tree of Bags

some bag. (Here, it is important that the A is a subset.) The most difficult parts are to then verify the interpolation property, the fact that the width does not increase, and the fact that we get a lexicographic decrease in the size.

We need the following technical observations. We state these for $q = i$ but again they hold with roles reversed for $q = j$.

Lemma 7.25

(a) $V_i' \subseteq (F_i \cup A) \cap V_i$.

(b) $A \subseteq W_{\pi_j^{-1}(i)}$.

(c) $V_{\pi_i(s)} \cap (F_i \cup A) \subseteq W_s$.

(d) $(F_i \cup A) \cap P_t$ is a subpath of P_t , for $t = 1, \dots, m$.

Proof of Lemma 7.25. (a) follows from the choice of sets F_i and F_j so that (iii) $V'_i \subseteq F_i \cup A$ and $V'_j \subseteq F_j \cup A$ and since $V'_i \subseteq V_i$. To obtain (b), first observe that by (a),

$$\text{for } t = 1, \dots, m, \quad \emptyset \neq P_t \cap V'_i \subseteq P_t \cap (F_i \cup A) \cap V_i.$$

Now note that $\{n \in \mathcal{T} : n \text{ is on the subtree of } \mathcal{T} \text{ extending } \pi_j(\pi_j^{-1}(i)) \text{ determined by } i\}$ is $\{n \in \mathcal{T} : i \in [i, n]\} = V(\mathcal{T})$. Hence, $A_{\pi_j^{-1}(i)}$ is the set of all $x_t \in A \cap V(\mathcal{T})$ (i.e., $x_t \in A$), with the corresponding path containing at least one element of $(F_i \cup A) \cap V'_i$, which it must. Hence (b) follows.

Now for (c). Note that $V_{\pi_i(s)} \cap (F_i \cup A) = (V_{\pi_i(s)} \cap F_i) \cup (V_{\pi_i(s)} \cap A)$. Thus, since $W_s = (F_i \cap V_{\pi_i(s)}) \cup A_s$, to establish (c) we need only observe that $(V_{\pi_i(s)} \cap A) \subseteq A_s$. Recall that A_s is the set of all $a_t \in A$ with $a_t \in \cup\{V_n : \pi_i(s) \in [n, j]_{\mathcal{T}}\}$ such that the corresponding path P_t includes at least one element of $(F_j \cup A) \cap V_{\pi_i(s)}$. Naturally, if there is some element of $V_{\pi_i(s)} \cap A$, this will correspond to a path with an element in A in $(F_j \cup A) \cap V_{\pi_i(s)}$. Hence (c) follows from the definition of W_s .

Finally, we turn to (d). Note that $P_t \cap A = \{a_t\}$ by the application of Menger's theorem used in the definition of A . Also by Menger's theorem, the path must pass through A and cannot return. Therefore, (d) follows.

We now turn to the verification that $\{W_s : s \in \widehat{\mathcal{T}}\}$ is a Thomas decomposition. We leave it to the reader to prove that $\cup_{s \in \widehat{\mathcal{T}}} W_s = V(G)$. (Exercise 3) and that for each edge $e = vw$ of G , there is some bag W_s with $v, w \in W_s$ (Exercise 4). To verify the interpolation property, let $s_1, s_2, s_3 \in \widehat{\mathcal{T}}$, and suppose that s_2 is on the path from s_1 to s_3 . Let $v \in W_{s_1} \cap W_{s_3}$. We need to prove that $v \in W_{s_2}$. Since i and j are symmetric, there are two cases.

Case 1. $s_1, s_2, s_3 \in V(\mathcal{T}_i)$.

If $v \notin A$, $v \in F_i \cap V_{\pi_i(s_1)} \cap V_{\pi_i(s_3)}$, by definition of $\mathcal{T}_i$. But then, $v \in F_i \cap V_{\pi_i(s_2)}$ by the interpolation property of $\mathcal{T}$, and hence $v \in W_{s_2}$ by Lemma 7.25(c). On the other hand, if $v \in A$, then $v \in A_{s_1} \cap A_{s_3}$ by applying the fact that F_i , F_j , and A are disjoint to the definition of $W_s = (F_i \cap V_{\pi_i(s)}) \cup A_s$. Thus, $v = x_p$ for some p with the additional property that the corresponding path meets $(F_j \cup A) \cap V_{\pi_i(s_z)}$ for $z \in \{1, 3\}$. Consequently, by Lemma 7.25(d) and Lemma 7.21, P_p meets $(F_j \cup A) \cap V_{s_2}$. If $\pi_i(s_3) \in [\pi_i(s_1), j]$ or $\pi_i(s_1) \in [\pi_i(s_3), j]$, then

$$\begin{aligned} v = x_p &\in A_{s_1} \cap A_{s_3} \subseteq A \\ &\cap \left(\bigcup \{V_n : n \text{ in the subtree of } \mathcal{T} \text{ extending } \pi_i(s_1) \text{ determined by } j\} \right) \\ &\cap \left(\bigcup \{V_n : n \text{ in the subtree of } \mathcal{T} \text{ extending } \pi_i(s_3) \text{ determined by } j\} \right). \end{aligned}$$

But since $s_2 \in [s_1, s_3]$ and hence

$$\begin{aligned} & \{n \in \mathcal{T} : \pi_i(s_1) \in [n, j]\} \cap \{n \in \mathcal{T} : \pi_i(s_3) \in [n, j]\} \\ & \subseteq \{n \in \mathcal{T} : \pi_i(s_2) \in [n, j]\}, \end{aligned}$$

it follows that $x_p \in A \cap (\bigcup \{V_n : \pi_i(s_2) \in [n, j]\}_{\mathcal{T}})$. Consequently, $v \in A_{s_2}$, as required.

Finally, suppose that $\pi_i(s_3) \notin [\pi_i(s_1), j]$ and $\pi_i(s_1) \notin [\pi_i(s_3), j]$. Again,

$$\begin{aligned} v &= x_p \in A_{s_1} \cap A_{s_3} \\ &\subseteq A \cap \{V_n : \text{in the subtree of } \mathcal{T} \text{ extending } \pi_i(s_1) \text{ determined by } j\} \\ &\quad \cap \{V_n : \text{in the subtree of } \mathcal{T} \text{ extending } \pi_i(s_3) \text{ determined by } j\}. \end{aligned}$$

But now we can apply Lemma 7.25(c) and use the interpolation property of $\{V_z : z \in \mathcal{T}\}$ to see that $x_p \in A \cap V_{\pi_i(s_2)} \subseteq W_{s_2}$, as required.

Case 2. $s_1, s_2 \in V(\mathcal{T}_i)$ and $s_3 \in V(\mathcal{T}_j)$.

This means that $s_2 \in [\pi_i^{-1}(j), s_1]_{\widehat{\mathcal{T}}}$. Consequently, by the fact that F_1, F_2 , and A are pairwise disjoint, Lemma 7.25, and Case 1,

$$v \in W_{s_1} \cap W_{s_3} = A_{s_1} \cap A_{s_3} \subseteq W_{s_1} \cap A \subseteq W_{s_1} \cap W_{\pi_i^{-1}(j)} \subseteq W_{s_2}.$$

Thus, we have established the fact that $\{W_s : s \in \widehat{\mathcal{T}}\}$ is a tree decomposition. To complete the proof, we need to demonstrate that the decomposition has width $\leq k$ and that its size is lexicographically less than $\{V_n : n \in \mathcal{T}\}$.

The first is immediate since, for $s \in \mathcal{T}_i$, the mapping $\varphi : W_s \mapsto V_{\pi_i(s)}$ defined by $\varphi(v) = v$ if $v \in F_i \cap V_{\pi_i(s)}$, and $\varphi(v)$ is the appropriate element of $P_p \cap (F_j \cup A) \cap V_{\pi_i(s)}$ in the case that $v = x_p$, must be injective by choice of A . (Remember, the paths are disjoint, filter through A , and A separates F_i and F_j .)

To demonstrate that we get a lexicographically lower tree decomposition, we begin with a lemma. Recall that $c(V'_i, V'_j) = m$. Let

$$M = \max\{m, |V_t| : t \in V(\mathcal{T}) \text{ with } |V_t| > \max\{|W_{\pi_i^{-1}(t)}|, |W_{\pi_j^{-1}(t)}|\}\}.$$

Lemma 7.26

- (a) If $t \in V(\mathcal{T})$, $s_i \in V(\mathcal{T}_i)$, and $s_j \in V(\mathcal{T}_j)$ with $t = \pi_i(s_i) = \pi_j(s_j)$ and with $|W_{s_i}|, |W_{s_j}| > M$, then $|V_t| > \max\{|W_{s_i}|, |W_{s_j}|\}$.
- (b) $\max\{|W_{\pi_i^{-1}(j)}|, |W_{\pi_j^{-1}(i)}|\} < M$.
- (c) If K is an $(n, 1)$ -cell in $\{W_s : s \in \widehat{\mathcal{T}}\}$, and $n \geq M$, then $V(K) \subseteq V(\mathcal{T}_i)$ or $V(K) \subseteq V(\mathcal{T}_j)$.
- (d) There exists a $d \in \mathbb{N}$ and a (M, d) -cell K in $\{V_q : q \in \mathcal{T}\}$, such that for any (M, d) -cell L in $\{W_s : s \in \widehat{\mathcal{T}}\}$,

$$\{\pi_r(v) : v \in V(L)\} \cap V(K) = \emptyset,$$

where $r \in \{i, j\}$ and $V(L) \subseteq V(\mathcal{T}_r)$.

Proof. To prove Lemma 7.26(a), let $t = \pi_i(s_i) = \pi_j(s_j)$ as above and $|V_t| = |W_{s_i}|$, say. It suffices to show that $|W_{s_j}| \leq M$. First, we claim that $V_t \cap F_j = \emptyset$. For a contradiction, suppose that $V_t \cap F_j = \{v_1, \dots, v_p\}$. Since $|V_t| = |W_{s_i}|$, the mapping φ above is surjective for $s = s_i$. Hence, each v_q belongs to some path P_z for $z \in \{1, \dots, m\}$, and each P_z can contain at most one element of $V_t \cap (F_j \cup A)$. Without loss of generality, suppose that v_z is a member of P_z for $z = 1, \dots, p$. Since φ is surjective, $a_z \in W_{s_i} - V_t$ for $z \in \{1, \dots, p\}$. The point is that $a_z \in \cup\{V_n : n \text{ is on the subtree of } \mathcal{T} \text{ extending } t \text{ determined by } j\}$. Let V be either V_t or V'_i . We claim that $B = \{v_1, \dots, v_p, a_{p+1}, \dots, a_m\}$ separates V and V'_j . Let P be a path connecting $v \in V$ to $v_j \in V'_j$, avoiding B . Naturally, this path P must use a_z for $z \leq p$ if $V = V'_i$. Suppose that $V = V_t$. Then by the fact that $F_i \cup F_j \cup A = V(G)$, since $v \in V - B \subseteq F_i \cup A$ and $v_j \in V'_j \subseteq F_j \cup A$, it is also the case that the path P must use a_z for $z \leq p$. Now, take a_z and a subpath P' of P such that P' joins a_z to V'_j and lies in $F_j \cup \{a_z\}$. Let n be in the subtree of $\mathcal{T}$ extending t determined by j with $a_z \in V_n$. Then, $P' \cap V_n \neq \emptyset \neq P' \cap V_j \subseteq P' \cap V_j$. Hence, by Lemma 7.25, we get the following contradiction:

$$\emptyset \neq P' \cap V_t \subseteq P \cap (F_j \cup \{a_z\}) \cap V_t = P \cap F_j \cap V_t = P \cap \{v_1, \dots, v_p\} = \emptyset.$$

Therefore, B separates V'_i and V'_j and also separates V_t and V'_j . If $t \in [i, j] - \{i\}$, then we contradict the fact that we chose the path $[i, j]$ to be as short as possible. If $t \notin [i, j] - \{i\}$, then since $a_z \in \cup\{V_n : n \text{ is in the subtree of } \mathcal{T} \text{ extending } t \text{ determined by } j\} - V_t$, one can see that $d_{v_z} < d_{a_z}$, which, in turn, demonstrates that

$$\sum_{q \in B} d_q < \sum_{q \in A} d_q,$$

contradicting the choice of A . Therefore, $V_t \cap V'_j = \emptyset$, $W_{s_j} \subseteq A$, and thus $|W_{s_j}| \leq m$, a contradiction which establishes (a).

To see that (b) holds, suppose that $m < M \leq |W_{\pi_j^{-1}(i)}|$. By Lemma 7.25(a) and (c), $V'_i \subseteq (F_i \cup A) \cap V_i \subseteq W_{\pi_i^{-1}(i)}$. Hence, $m < |V'_i| \leq |W_{\pi_i^{-1}(i)}|$. But then, the definition of M and Lemma 7.25(a),

$$M \leq |W_{\pi_j^{-1}(i)}| \leq \max\{|W_{\pi_i^{-1}(i)}|, |W_{\pi_i^{-1}(i)}|\} < |V_i| \leq M,$$

a contradiction which establishes (b).

Note that (c) follows immediately from (b) since any connected subtree of bags including the edge $\widehat{e}$ must include the bag $W_{\pi_j^{-1}(i)}$ which has size less than M by (b).

To complete the proof of Lemma 7.26, we need (d). Note that $|V_t| > m$ for each $t \in [i, j]_{\mathcal{T}}$, since we assume that V_i, V'_i, V_j , and V'_j were counterexamples to the Thomas property. Hence, some $(m+1, 1)$ -cell $\mathcal{C}$ of $\mathcal{T}$ contains $[i, j]_{\mathcal{T}}$. If $M = m+1$, then choose K to be $\mathcal{C}$. Otherwise, $M > m+1$. Choose $t \in \mathcal{T}$ with $M = |V_t| > \max\{|W_{\pi_i^{-1}(t)}|, |W_{\pi_j^{-1}(t)}|\}$. This time let K be the $(M, 1)$ -cell in $\mathcal{T}$ containing V_t . Let $d = |V(K)|$, in both cases.

Now let L be any (M, d) -cell in $\widehat{T}$. By (c) we have that, without loss of generality, $V(L) \subseteq V(T_i)$. Since $|V_{\pi_i(s)}| \geq |W_s|$ for any s , $L' = \{\pi_i(s) : s \in V(L)\}$ is contained in some (M, d) -cell of T . By choice of d , either $L' \cap V(K) = \emptyset$ or $L' = V(K)$. Because of Lemma 7.26(b) and (c), the latter case is impossible in the case that $M = m + 1$. But the latter case is also impossible if $M > m + 1$ by choice of d .

We conclude the proof with the following lemma.

Lemma 7.27 *The size of $\{W_s : s \in \widehat{T}\}$ is lexicographically less than the size of $\{V_n : n \in T\}$.*

Proof. Let A be the size of $\{V_n : n \in T\}$ and B be the size of $\{W_s : s \in \widehat{T}\}$. For $n \geq M$, we shall define a mapping Γ from (n, d) -cells in $\{W_s : s \in \widehat{T}\}$ to (n, d) -cells in $\{V_n : n \in T\}$. [Here, we remark that d is given by the previous lemma, and remind the reader that if $d' \geq d$, then if K is an (n, d) -cell, it is also a (n, d') -cell.]

Fix $n \geq M$. Let K be a (n, d) -cell in $\{W_s : s \in \widehat{T}\}$. By Lemma 7.26(c), $V(K) \subseteq V(T_i)$ (by symmetry), and by the surjectivity of φ , the set $\{\pi_i(v) : v \in V(K)\}$ is a subset of some (n, d) -cell K' in $\{V_n : n \in T\}$. Define $\Gamma(K) = K'$. We can apply Lemma 7.26(d) to find an (n, d) which is lexicographically greatest such that Γ is not a bijection from the collection of (n, d) -cells in $\{W_s : s \in \widehat{T}\}$ to the collection of (n, d) -cells in $\{V_n : n \in T\}$. [Hence, Γ is a bijection from the collection of (n', d') -cells in $\{W_s : s \in \widehat{T}\}$ to the collection of (n', d') -cells in $\{V_n : n \in T\}$ whenever $(n, d) \leq_{\text{lex}} (n', d')$.] We claim that Γ is injective on the set of (n, d) -cells in $\{W_s : s \in \widehat{T}\}$.

Suppose that this claim fails, and that there are distinct (n, d) -cells K_1 and K_2 in $\{W_s : s \in \widehat{T}\}$ with $\Gamma(K_1) = \Gamma(K_2)$. By definition of cell, we have that $V(K_1)$ is disjoint from $V(K_2)$. We may suppose that K_1 is chosen to be the (n, d) -cell C of maximal size in $\widehat{T}$ with $\Gamma(C) = \Gamma(K_2)$.

Case 1. $|V(K_1)| < |V(\Gamma(K_1))|$. Let $d' = |V(\Gamma(K_1))|$. Let K be any (n, d) -cell in $\{W_s : s \in \widehat{T}\}$ with $\Gamma(K) = \Gamma(K_1)$. We have that

$$|V(K)| \leq |V(K_1)| < |V(\Gamma(K_1))|.$$

Hence, K cannot be a (n, d') -cell. But this means that Γ is not onto for (n, d') -cells, which contradicts the choice of (n, d) .

Case 2. $|V(K_1)| = |V(\Gamma(K_1))|$. By Lemma 7.26(c), we can suppose that $V(K_1) \subseteq V(T_i)$. Hence, $V(K_2) \subseteq V(T_j)$. Let $s_2 \in V(K_2)$. Then, $\pi_j(s_2) \in V(\Gamma(K_1)) = \{\pi_i(v) : v \in V(K_1)\}$. Thus, there exists $s_1 \in V(K_1)$ such that $\pi_i(s_1) = \pi_j(s_2)$. By Lemma 7.26(a) and the definition of M , we have that $M \leq n \leq |W_{s_1}| = |W_{s_2}| = |V_{\pi_i(s_1)}| = |V_{\pi_j(s_2)}| \leq M$, a contradiction.

Therefore, we see that Γ is injective on the set of (n, d) -cells in $\{W_s : s \in \widehat{T}\}$. By the choice of (n, d) , this implies that $A_{n,d} > B_{n,d}$, and if (n', d') is lexicographically greater than (n, d) , then $A_{n',d'} = B_{n',d'}$; hence, the size of $\{W_s : s \in \widehat{T}\}$ is lexicographically less than that of $\{V_n : n \in T\}$. This proves Lemma 7.27, and hence concludes the proof of Thomas' Lemma.

7.2.1 Thomas' Lemma Fails for Path Decompositions

The reader might wonder if there is an analog of Thomas' Lemma for paths.

Definition 7.28 A path decomposition P of a graph G is a *Thomas path decomposition* if whenever there are two bags B_1 and B_2 and subsets of vertices $V_1 \subseteq B_1$ and $V_2 \subseteq B_2$ such that $s(V_1, V_2) < \min\{|V_1|, |V_2|\}$, then there is a bag B_i between B_1 and B_2 in P such that $|B_i| = s(V_1, V_2)$.

As we now see this is a vacuous definition.

Theorem 7.29 (Coles [128]) *For any $k > 1$, there exists a graph G that has pathwidth k but does not have a Thomas path decomposition of width k .*

Proof. First, we give a graph which has pathwidth 2 and prove it has no Thomas path decomposition of pathwidth 2. It will then be clear how to take our counterexample and generalise it for pathwidth $k > 0$.

Consider the graph $G = (V, E)$ with $V = \{x, y, z, a, u_1, u_2, u_3, w_1, w_2, w_3\}$ and $E = \{(x, y), (y, z), (x, z), (a, z), (a, u_1), (u_1, u_2), (u_2, u_3), (a, w_1), (w_1, w_2), (w_2, w_3)\}$.

This is given in Figure 7.4.

A path decomposition of G of pathwidth 2 is given in Figure 7.5.

Clearly, because G has a clique of size 3 it cannot have pathwidth less than 2, and hence has pathwidth exactly 2.

Let P be a given path decomposition of G with pathwidth 2. We show that P cannot have the Thomas property. Since the vertices x, y , and z form a clique in G , and P has pathwidth 2, there must be a bag in P containing exactly the vertices x, y , and z . We label this bag B_1 .

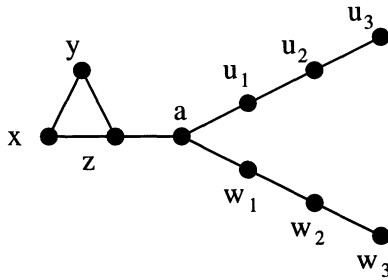


FIGURE 7.4. The Coles Graph

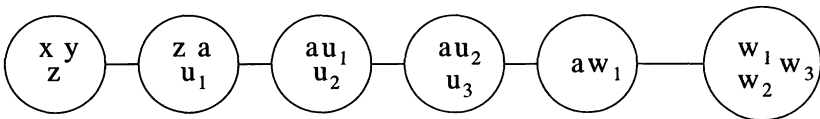


FIGURE 7.5. A Path Decomposition of G

Notice that B_1 cannot have bags both sides of it in P because otherwise the interpolation property would force some other vertex to be in bag B_1 , contradicting the pathwidth of P . So, without loss of generality we refer to B_1 as the leftmost bag in P , and this, in turn, allows us to refer to pairs of bags in P being to the left or to the right of one another.

Lemma 7.30 *Let $U = \{u_1, u_2, u_3\}$ and $W = \{w_1, w_2, w_3\}$.*

- (i) $P_U = \{B_i \in P \mid B_i \cap U \neq \emptyset\}$ is a subpath of P ; similarly for P_W .
- (ii) Either $|B_i| \geq 2$ for all $B_i \in P_U$ or else $|B_i| \geq 2$ for all $B_i \in P_W$.

Proof. (i) Consider P_U . Suppose P_U is not a subpath of P . Then, there is some bag B_* in P that does not contain any element of U and cutting B_* from P produces two subpaths of P , each containing at least one bag of P_U . By the interpolation property, the bags in P_U to the left of B_* in P have no vertices in common with any of the bags in P_U to the right of B_* in P . However, (u_1, u_2) and (u_2, u_3) are in E and, therefore, u_1 and u_2 must occur in the same bag somewhere in P_U , and u_2 and u_3 must occur in the same bag somewhere in P_U .

We write $left(P_U)$ to denote the bags of P_U to the left of B_* in P , and write $right(P_U)$ to denote the bags of P_U to the right of B_* in P . Because of symmetry, there are three cases to consider:

- (a) $u_1 \in left(P_U)$ and $u_2 \in right(P_U)$,
- (b) $u_1, u_2 \in left(P_U)$ and $u_3 \in right(P_U)$,
- (c) $u_1, u_2, u_3 \in left(P_U)$.

By the interpolation property, subcase (a) implies that at least one of u_1 and u_2 are in B_* , subcase (b) implies that at least one of u_2 and u_3 are in B_* , and subcase (c) implies that at least one of u_1 , u_2 , and u_3 are in B_* . In each case, this contradicts the assumptions made about B_* . It follows that P_U must be a subpath of P .

An identical argument proves that P_W is a subpath of P .

(ii) Since there are edges (a, u_1) and (a, w_1) in G , then P has a bag containing both a and u_1 , and a bag containing both a and w_1 . Notice that the bag in P containing both a and z can only contain at most one of u_1 and w_1 , for otherwise, P would have pathwidth 3.

By the symmetry of G , we only need to consider two cases:

Case 1: There is a bag in P containing a , z , and u_1 . Label this bag B_{u_1} . Without loss of generality, we may assume there is no bag between B_1 and B_{u_1} that contains w_1 , for otherwise, there would be a bag to the left of B_{u_1} containing a , z , and w_1 . If there are bags to the left of B_{u_1} which are in P_U , then they must also contain the vertex a by the interpolation property and hence contain at least two vertices.

Subcase (a): Suppose w_1 does not belong to any bag of P_U . Then, the leftmost bag containing w_1 must be to the right of the end bag of P_U . Since P_U is a subpath of P , by the interpolation property, every bag of P_U contains a . Hence, every bag of P_U contains at least two vertices.

Subcase (b): Suppose w_1 is in some bag of P_U . Then, all bags between B_{u_1} and B_{w_1} (the label of the leftmost bag of P containing w_1) inclusive must contain a . If B_{w_1} is the rightmost bag of P_U , then every bag of P_U contains at least two vertices.

Suppose then that there is at least one bag to the right of B_{w_1} in P_U . We claim that either each bag of P_U to the right of B_{w_1} contains at least one member of W , or each bag of P_W contains at least two vertices.

If the former is true, then, clearly, every bag of P_U contains at least two vertices. Otherwise, there is a bag to the right of B_{w_1} in P_U which does not contain any vertex of W . Since P_U and P_W are subpaths of P , and P_W is contained within P_U it must be that every bag of P_W contains at least two vertices.

Case 2: There is a bag in P containing a and u_1 but not containing z . Label this bag B_{u_1} . Again, we may assume without loss of generality that there is no bag between B_1 and B_{u_1} that contains B_{w_1} .

Following the same reasoning as in Case 1, we can show that either every bag of P_U contains at least two vertices, or every bag of P_W contains at least two vertices.

This completes the proof of the lemma.

Now, take the set of vertices U if P_U has all bags of size 2 or more, or take W . Without loss of generality, assume it to be U . Observe that P_U must have at least two bags since a, u_1, u_2 , and u_3 cannot all be in the same bag if P has pathwidth 2. Furthermore, there must be a bag V_1 such that $V'_1 \subseteq V_1$ with $V'_1 = \{a, u_1\}$ and a bag V_2 such that $V'_2 \subseteq V_2$ with $V'_2 = \{u_2, u_3\}$. Clearly, $s(V'_1, V'_2) = 1$, but there is no bag of size 1 between V_1 and V_2 in P_U , and hence in P . Therefore, P does not have the Thomas property.

This completes the proof for the case $k = 2$.

By replacing the copy of the complete graph K_3 in G by K_{k+1} , we obtain the theorem for any given $k > 0$.

Historical Notes

The material on treewidth of this section is taken from Thomas [442]. In an early draft of Graph Minors IV, in place of the Thomas property, Robertson and Seymour used the weaker property that if the treewidth of G is $\leq k$, then G has a tree decomposition of width $\leq 3.2^k$ such that

For any bags V_i and V_j , and any k , if there are only k disjoint paths from vertices in V_i to vertices in V_j in G , then there is a bag V_m on the path in T such that $|V_m| \leq k$.

Upon hearing the well-quasi-ordering result of Robertson and Seymour, Thomas gave an independent proof of the well-quasi-ordering of graphs of bounded treewidth under the minor ordering (Thomas [443]) based upon the stronger lemma of this section. The final version of Graph Minors IV also used Thomas' lemma.

Together with Kriz, Thomas [310] extended the lemma to infinite graphs and, using this, was able to prove the well-quasi-ordering of infinite graphs of bounded treewidth. Infinite graphs of unbounded treewidth are not well-quasi-ordered

(Thomas [444]). The question of whether countable graphs are well-quasi-ordered under the minor relation is currently open and seems hard.

The maximum flow problem was formulated by Fulkerson and Danzig in 1955 [236]. The algorithm given in Exercise 1 was the solution found by Ford and Fulkerson [226] who also proved the Max Flow/Min Cut Theorem. Edmonds and Karp [192] proved that the Ford-Fulkerson algorithm runs in time $O(|V||E|^2)$, provided that the shortest augmenting paths are used in the implementation. There have been numerous improvements continuing to this day. A noteworthy addition was the algorithm of Goldberg and Tarjan [246] which is based on the concept of “preflow/push” rather than augmenting paths. (In a preflow, the flow in can exceed the flow out. The basic idea is to find a preflow that pushes as much from the source to the sink as possible. Having done this, one trims the excess moving back from the sink to the source. The algorithm is controlled by tokens.) We refer the reader to, for instance, Ahuja-Orlin [15].

Menger's theorem was proven using different techniques by Menger [350]. Recently, Galil and Yu [238] have proven the following extension of Menger's theorem. *Suppose that G is a simple undirected graph and there are k edge disjoint paths between u and v . Then, there are k edge disjoint paths between u and v of average length $O(\frac{n}{\sqrt{k}})$. Furthermore, if all the vertices have degree at least k , then there are k edge disjoint paths between u and v each of which has length at least $O(\frac{n}{k})$.*

Exercises 7.2

7.2.1. (Max Cut/Min Flow Theorem, Ford and Fulkerson [226]) A *transportation network* is a weighted digraph G with a single vertex s of indegree 0 (called the *source*) and a single vertex of outdegree 0 called the *sink*. In this context, the (non-negative) weights on the edges xy are called the *capacities* $c(x, y)$. For a vertex x , let $A(x) = \{y : xy \in E(G)\}$, and $B(x) = \{y : yx \in E(G)\}$. For sets of vertices X, Y and functions $g : E(G) \mapsto \mathbb{R}$, we let $(X, Y) = \{xy : x \in X, y \in Y \text{ and } xy \in E(G)\}$, and we let

$$g(X, Y) = \sum_{xy \in E(G)} g(xy).$$

A *flow* is a non-negative function $f : E(G) \mapsto \mathbb{R}$ such that (i) $f(xy) \leq c(x, y)$ (the flow is less than the capacity), (ii) $f(x, A(x)) - f(B(x), x) = 0$, if $x \notin \{s, t\}$ (that is, flow in equals flow out), and (iii) $f(s, A(s)) = f(B(t), t)$. [$f(s, A(s))$ is called the *value* of the flow and denote it by $val(f)$.]

The intuition here is that if we think of the network as a set of oil pipes, say, then the flow represents the amount of oil currently traveling through the system. For $X \subset V(G)$, if $s \in X$ and $t \notin \overline{X}$, we call the set of edges $(X, \overline{X})$ a *cut*.

- (i) Prove that for all cuts $(X, \bar{X})$, $\text{val}(f) = f(X, \bar{X}) - f(\bar{X}, X)$ and hence $f(X, \bar{X}) \leq c(X, \bar{X})$.

The *maximal* flow is the maximum possible value of $\text{val}(F)$, and the minimal cut capacity is the minimal value of $c(X, \bar{X})$ over all cuts $(X, \bar{X})$. The fundamental problem in transposition networks is to find a flow which is maximal. The classical method of Ford and Fulkerson uses what are called *augmenting paths*. We say that a sequence of nonrepeated vertices in G , $s = v_0 v_1 \dots v_n = t$ from s to t together with a set of edges of the form $v_i v_{i+1} \in E(G)$ or $v_{i+1} v_i \in E(G)$ for all i is an augmenting path if the following two conditions are met.

- (a) if $v_i v_{i+1} \in E(G)$, then $f(v_i v_{i+1}) < c(v_i, v_{i+1})$,
- (b) if $v_{i+1} v_i \in E(G)$, then $f(v_{i+1} v_i) > 0$.

The value of the augmenting path is the minimum over all i of $c(v_i, v_{i+1}) - f(v_i v_{i+1})$ if (a) pertains, and $f(v_{i+1} v_i)$ if (b) pertains.

- (ii) Prove that if a transportation network contains an augmenting path, then it is possible to improve the flow. The improvement can be made as the value q of the augmenting path. The improved flow is obtained by raising the flow from v_i to v_{i+1} by q if (a) pertains, and dropping the flow from v_{i+1} to v_i if (b) pertains.
- (iii) Design an algorithm based upon (ii) that maximizes the flow in the network, based on finding augmenting paths until there are no more. We shall refer to this as the Ford-Fulkerson algorithm. Demonstrate that the algorithm runs in polynomial time.
- (iv) Prove that a transportation network with current flow f , either contains an augmenting path or a cut whose capacity is $\text{val}(f)$. Hence deduce the Max Flow/Min Cut Theorem: In a transportation network, the maximum flow capacity is the same as the minimum cut capacity. Furthermore, for any minimum cut $(X, \bar{X})$, a maximum flow f will have $f(X, \bar{X}) = \text{val}(f)$ and $f(\bar{X}, X) = 0$.

7.2.2. A *matching* in a bipartite graph $G = \langle V_1 \cup V_2, E \rangle$ is a collection M of edges of G such that for all $xy \neq pq$ with $xy, pq \in E$, $x \neq p$ and $y \neq q$. Design an algorithm based on the Ford-Fulkerson algorithm above which computes a matching of largest possible size.

7.2.3. Verify the following claim about the construction of $\widehat{T}$ in the proof of Thomas' Lemma. For each $v \in V(G)$, there is some W_s such that $v \in W_s$.

(Hint; Use Lemma 7.25 (c).)

7.2.4. Verify the following claim about the construction of $\widehat{T}$ in the proof of Thomas' Lemma. *For any edge $vw = e$ of G , both v and w must appear together in some bag W_s of $\widehat{T}$.*

(Hint: v and w must occur in V_t for some $t \in \mathcal{T}$. We know that A separates F_i and F_j . Hence, without loss of generality, we can assume that both v and w are in $F_i \cup A$. But then both v and w must be in $V_t \cap (F_i \cup A)$ which is a subset of $W_{\pi_i^{-1}(t)}$ by Lemma 7.25(c).)

7.2.5. Prove Lemma 7.21.

7.2.6. Prove Lemma 7.25.

7.2.7. Use the proof of Thomas' Lemma to obtain an algorithm which converts a standard tree decomposition into a Thomas decomposition. What is the complexity of the algorithm.

7.2.8. Construct a Thomas decomposition for the width-3 graph of Figure 6.16 of Example 6.48.

7.3 The Graph Minor Theorem for Bounded Treewidth

In this section, we shall give a proof of the Graph Minor Theorem for graphs of bounded treewidth. We follow Robertson and Seymour [391]. For this section, we shall mean *directed, rooted* tree unless otherwise stated. For a tree T , $r(T)$ denotes the root of T . For a graph G , we say a subset X is *nonadjacent* iff no pair of elements in X are adjacent in G . We begin with an extension of Kruskal's theorem (Theorem 7.14).

Lemma 7.31 *Let $T_1, T_2, \dots$ be any countable collection of disjoint trees and M any graph such that $V(M) = V(T_1 \cup T_2 \cup \dots)$, with the property below.*

For any $j > i$, if $u \in V(T_i)$ is adjacent in M to $w \in V(T_j)$ and $v \in V(T_j) - \{r(T_j)\}$ lies on the path of T_j from the root to w , then u is adjacent in M to v .

Suppose that the collection of roots $\{r(T_k) : k = 1, 2, \dots\}$ is nonadjacent. Then, there is an infinite nonadjacent set $X \subseteq V(M)$ such that for all k , $|X \cap V(T_k)| \leq 1$, and such that the set of heads of edges of $\cup_k T_k$ with tails in X has no infinite M -nonadjacent subset.

Proof. See Exercise 1.

The idea of the use of Lemma 7.31 can be gained from the the following proof of Kruskal's theorem. Let $T_1, T_2, \dots$ be an infinite sequence of disjoint trees. Let M be an infinite graph defined so that $V(M) = \cup_i V(T_i)$, such that if $i < j$, then we make $u \in V(T_i)$ M -adjacent to $v \in V(T_j)$ iff the subtree of T_i with root u is topologically embeddable into the subtree of T_j below v . Then, the conclusion of Lemma 7.31 fails, so that the sequence of roots $\{r(T_i); i = 1, 2, \dots\}$ must have adjacent vertices, giving Kruskal's theorem.

Actually, Robertson and Seymour use a generalization (Lemma 7.33) of Lemma 7.31. We need a definition.

Definition 7.32 (Precedence) Let $\phi : E(T) \mapsto \{0, \dots, n\}$ be a mapping from edges of a tree to integers less than or equal to n . We say a vertex v *precedes* a vertex w of T with respect to ϕ if (i)-(iv) below hold.

- (i) $v \neq r(T)$.
- (ii) There is a (directed) path P of T from v to w .
- (iii) $\phi(e) = \phi(f)$ for the edge e of T with head v and the edge f of T with head w .
- (iv) $\phi(g) \geq \phi(e)$ for all edges g of P .

Lemma 7.33 Let $T_1, T_2, \dots$ be any countable collection of disjoint trees and $\phi_i : E(T_i) \mapsto \{0, \dots, n\}$ arbitrary functions with n fixed. Suppose that M is any graph such that $V(M) = V(T_1 \cup T_2 \cup \dots)$, and with the property below.

- For any $j > i$, if $u \in V(T_i)$ is adjacent in M to $w \in V(T_j)$ and $v \in V(T_j)$ precedes w with respect to ϕ_j , then u is adjacent in M to v .

Suppose that the collection of roots $\{r(T_k) : k = 1, 2, \dots\}$ is nonadjacent. Then, there is an infinite nonadjacent set $X \subseteq V(M)$ such that for all k , $|X \cap V(T_k)| \leq 1$, and such that the set of heads of edges of $\cup_k T_k$ with tails in X has no infinite M -nonadjacent subset.

Proof. Use induction on n , the base case $n = 0$ being Lemma 7.31. Suppose the lemma for $n - 1$ and that $n > 0$. Define $F_i = \{e \in E(T_i) : \phi(e) = 0\}$. Let S_i be the contraction of T_i onto F_i [meaning that $E(S_i) = F_i$ and $V(S_i)$ is the set of roots of the components of $T_i - F_i$, and in which $v \in V(S_i)$ is the head of $f \in F_i$ iff the head of f belongs to the component of $T_i - F_i$ with root v (similarly for tails)]. Let N be the restriction of M to $V(S_1) \cup V(S_2) \cup \dots$.

Claim 1. For $j > i$, if $u \in V(S_i)$ is N -adjacent to $w \in N(S_j)$ and $v \neq r(S_i)$ lies on the path of S_j from w to u , then u is N -adjacent to v .

To see Claim 1, suppose that e and f are edges of T_j with heads u and w respectively. Since $u, w \in V(S_j)$, it follows that $e, f \in F_j$. Hence, $\phi(e) = \phi(f) = 0$. Since $\phi(g) \geq 0$ for each $g \in E(T_j)$, we note that u precedes w with respect to ϕ_j . Therefore, u is M - and hence N -adjacent to v , proving the claim.

Now apply Lemma 7.31 to $S_1, S_2, \dots$, and N . There must be an *increasing*⁸ sequence $z_1, z_2, \dots$ of vertices of M which is M -nonadjacent such that each z_j is the root of some component R_j of some $T_i - F_i$, and the set A of heads (in $T = T_1 \cup T_2 \cup \dots$) of all edges in $F_1 \cup F_2 \cup \dots$ with tails (in T) in $R_1 \cup R_2 \cup \dots$ contains no infinite M -nonadjacent subset.

⁸That is, for any $j_1 < j_2$ whenever $z_{j_1} \in V(T_{i_1})$ and $z_{j_2} \in V(T_{i_2})$, then $i_1 < i_2$.

For each $j \geq 1$, let $\phi'_j : E(R_j) \mapsto \{0, \dots, n-1\}$ be defined via $\phi'_j(e) = \phi_i(e) - 1$, where $z_j \in V(T_i)$. For the inductive hypothesis applied to $R_1, R_2, \dots$, and to the restriction of M to $V(\cup_i R_i)$, we see the following.

There is an infinite M -nonadjacent set $X \subseteq V(\cup_i R_i)$ such that $|X \cap V(T_i)| \leq 1$ and the set B of heads of all edges of $\cup_i R_i$ with tails in X has no infinite M -nonadjacent subset.

To see that X satisfies the lemma, suppose that C is the set of heads of all edges of T with tails in X and suppose that $z \in C$. Then, z is the head of some edge f in some T_i with tail in X . If $\phi(f) = 0$, then $f \in F_i$, and hence $z \in A$. If $\phi(f) > 0$, then f is an edge of some R_j since its tail is in some $V(R_j)$ and hence $z \in B$. Consequently, $C \subseteq A \cup B$. But we have seen that both A and B have no infinite M -nonadjacent subsets, giving the lemma.

To get the full power of the Robertson-Seymour result, we will introduce a generalization of the notion of tree decomposition for hypergraphs. A *hypergraph* consists of a finite set $V(G)$ of vertices, a collection E (disjoint from V) of (hyper-) edges together with an incidence relation. Be aware that $e \in E$ can have many vertices incident with it. We denote such edges by $V(e)$. Similarly, we can define $H \leq G$ to mean that H is a subhypergraph of G , where $V(H) \subseteq V(G)$, and for $e \in E(H)$, $E_H(e) = E_G(e)$. A *separation* is a pair (G_1, G_2) of subhypergraphs with $G_1 \cup G_2 = G$ and $E(G_1 \cap G_2) = \emptyset$. The order of the separation is $|V(G_1 \cap G_2)|$. A *march* in a vertex set V is a finite sequence of distinct elements of V . A *rooted hypergraph* is a pair (G, π) consisting of a hypergraph G and a march π . A *separation* in a rooted hypergraph G of order n is a pair of rooted hypergraphs (G_1, G_2) which form a separation of the underlying hypergraph of order n , and such that $\pi(G_2) = \pi(G)$ and the vertices of $\pi(G_1) = V(G_1 \cap G_2)$.

Definition 7.34 (Tree Decomposition for Rooted Hypergraphs) A *tree decomposition* of a rooted hypergraph G is a pair (T, b) , where T is a (rooted) tree and b is the *bag* function which assigns to each $t \in T$ a rooted subhypergraph $b(t)$ of G such that

- (i) $\cup_{t \in T} b(t) = G$ as unrooted hypergraphs.
- (ii) $t_1 \neq t_2$ implies $E(b(t_1) \cap b(t_2)) = \emptyset$.
- (iii) (interpolation) $t_1, T-2, t_3 \in T$ and t_2 lying on the path from t_1 to t_3 imply that $b(t_1) \cap b(t_3) \subseteq b(t_2)$. [Equivalently, by (ii), $V(b(t_1)) \cap V(b(t_3)) \subseteq V(b(t_2))$.]
- (iv) The march of the rooted hypergraph which is at the root of T is the same as that of G .
- (v) For each edge $t_1 t_2$ of T , the vertices of the march $\pi(b(t_1))$ equal the vertices of $b(t_1) \cap b(t_2)$.

Given a tree T and an edge e of T , we can naturally split $T - \{e\}$ into two components, T^e and T_e , where T_e has as root the tail of e and T^e has root $r(T)$.

If $\widehat{T}$ is a subtree of T and (T, b) is a tree decomposition of a rooted hypergraph (G, π) , the *rooted subhypergraph induced by $\widehat{T}$* is the pair $(\widehat{G}, \pi)$ with

$$\widehat{G} = \cup_{t \in V(\widehat{T})} b(t) \text{ and } \pi(\widehat{G}) = \pi(b(r(\widehat{T}))).$$

Lemma 7.35 (Robertson and Seymour [390, 392])

- (i) Suppose that (T, π) is a tree decomposition of G and that $e \in T$. Then, the pair of induced subhypergraphs $((T^e, \pi), (T_e, \pi))$ is a separation of G^9 .
- (ii) Suppose that $F \subseteq E(T)$. Let S denote the contraction of T onto F , and for each $s \in V(S)$, let $\widehat{\pi}(s)$ be the subhypergraph of T induced by the component of $T - F$ with root s . The (S, σ) is a tree decomposition of G .

Proof. Exercise 2.

Following Robertson and Seymour [391], we define a relation *simulation* and a class $\mathcal{R}$ of rooted hypergraphs which obey certain axioms. These axioms are enough to get the well-quasi-ordering result. Henceforth, we will assume them to hold and introduce the first two below. The simulation relation is a generalization of the minor relation to our rooted hypergraphs.

Axiom 1. If $G \in \mathcal{R}$ and H is a rooted hypergraph with $H \leq G$ (as hypergraphs), then $H \in \mathcal{R}$.

Axiom 2. “ G is simulated in H ” is a quasi-order on $\mathcal{R}$.

We will need certain distinguished tree decompositions called *star decompositions*. A star is a tree S such that $r(S)$ is connected to every vertex of S . Naturally, a star decomposition of G is a tree decomposition with the underlying tree a star. Let (T, b) be a tree decomposition of G and suppose that s is a vertex of T . Let S be the maximal star with root s which is a subtree of T . In this case, define $\sigma(s) = b(s)$, and for each t with st an edge of S , let $\sigma(t)$ be the rooted subhypergraph of (T, b) induced by T^t . Then, we note that (S, σ) is a star decomposition of the rooted subhypergraph from (T, b) induced by T^s . It is called the *branching of (T, b) at s* . The *index* of a star decomposition (S, σ) of (G, π) is $\max\{|\{v : v \in \pi(\sigma(s))\}| : s \in V(S)\}$, and the index of a class of a class of star decompositions is the maximum, if any, of the indices of the members of the class.

Definition 7.36 (Good Star Decomposition Class) We say a class $\mathcal{S}$ of star decompositions is *good* if

- (i) The rooted subhypergraph of (S, σ) induced by S [i.e., the rooted hypergraph *corresponding to (S, σ)*] is in the class $\mathcal{R}$ for all $(S, \sigma) \in \mathcal{S}$.
- (ii) $\mathcal{S}$ has finite index.

⁹The order of this separation is called the *order* of e in (T, b) .

- (iii) For each countable sequence (S_i, σ_i) ($i = 1, 2, \dots$) such that the collection $\{v : \exists i[v \in V(S_i) - \{r(S_i)\}]\}$ is well-quasi-ordered by simulation, there exists $i' > i \geq 1$ such that the rooted hypergraph corresponding to (S_i, σ_i) is simulated in $(S_{i'}, \sigma_{i'})$.

Let (T, b) be a tree decomposition with corresponding rooted hypergraph in $\mathcal{R}$. We will say that a subset F of edges of T is *linked* in (T, b) if for each directed path P of T , with first edge f_2 and last edge f_1 so that f_2 and f_1 have the same order and no element of $E(P) \cap F$ has smaller order, the subhypergraph of (T, b) induced by T^{f_1} is simulated in the subhypergraph of (T, b) induced by T^{f_2} . Finally, we say that (T, b) is linked if $E(T)$ is linked in (T, b) .

Lemma 7.37 *Let S be good. Suppose that $\{(T_i, b_i) : i \in \mathbb{N}\}$ is a sequence of linked tree decompositions, each branching of which is in S . Then, there exists an $i' > i \geq 1$ such that the rooted hypergraph corresponding to (T_i, b_i) is simulated in the rooted hypergraph corresponding to $(T_{i'}, b_{i'})$.*

Proof. Let n be the index of S . We can suppose that the T_i are pairwise disjoint. Define an infinite graph M with $V(M) = \cup_i V(T_i)$, and for $i' > i \geq 1$, let $u \in V(T_i)$ be M -adjacent to $v \in V(T_{i'})$ iff the rooted subhypergraph from (T_i, b_i) induced by T_i^u is simulated in the rooted subhypergraph from $(T_{i'}, b_{i'})$ induced by $T_{i'}^v$. For each $i \geq 1$ and each $e \in E(T_i)$, let $\phi_i(e)$ equal the order of e in (T_i, b_i) . The $0 \leq \phi_i(e) \leq n$ by hypothesis (since each branching belongs to S). Define “precedes” as in Lemma 7.33. Then,

Claim 1 *If $i' \geq i \geq 1$ and $u \in V(T_i)$ is M -adjacent to $w \in V(T_{i'})$, and $v \in V(T_{i'})$ precedes w , then u is M -adjacent to v .*

To see that Claim 1 holds, we first note that since u and w are M -adjacent, the subhypergraph from (T_i, b_i) induced by T_i^u is simulated in the subhypergraph from $(T_{i'}, b_{i'})$ induced by $T_{i'}^w$. Since v precedes w , $v \neq r(T_{i'})$. Let e and f be edges of $T_{i'}$ with heads v and w , respectively. Since v precedes w , e and f must have the same order and no edge of the path in $T_{i'}$ can have smaller order. But since $(T_{i'}, b_{i'})$ is linked, we see that the subhypergraph from $(T_{i'}, b_{i'})$ induced by $T_{i'}^w$ is simulated in the one induced by $T_{i'}^v$. By Axiom 2 and the definition of M -adjacency, we get Claim 1.

Claim 2 *There is no infinite subset $X \subseteq V(M)$ such that the following three properties hold:*

- (i) $|X \cap V(T_i)| \leq 1$ for each i .
- (ii) No two elements of X are M -adjacent.
- (iii) The collection of heads of edges of $\cup_i T_i$ contains no infinite subset of pairwise M -nonadjacent elements.

To see that Claim 2 holds, suppose that such a set $X = \{x_1, x_2, \dots\}$ exists, where we choose notation such that $j' > j$ and $x_j \in V(T_i)$ and $x_{j'} \in V(T_{i'})$ implies that $i' > i$. Now, for $x_j \in V(T_i)$, let (S_j, σ_j) be the branching of (T_i, b_i) at x_j , so that the rooted subhypergraph induced by S_j is the same as the one induced by $T_i^{x_j}$. Since X

has no M -adjacent pairs, one sees that for $j' > j$, the rooted subhypergraph from T_i induced by S_j is not simulated in the rooted subhypergraph from $T_{j'}$ induced by $S_{j'}$. Let $\mathcal{N}$ denote the set of all nonroots of the (S_j, σ_j) 's. Since $\mathcal{N}$ is good, it is not well-quasi-ordered by simulation, and hence there is a bad sequence $L_1, L_2, \dots$ of members of $\mathcal{N}$ such that if $i < j$, then L_i is not simulated in L_j . Now, for each j , there is an i and $v_j \in T_i$ such that L_j is the rooted subhypergraph from T_i induced by $T_i^{v_j}$. Furthermore, v_j is the head of an edge in T_i with tail in X . The v_j are all distinct, since otherwise if $v_j = v_{j'}$, then L_j is simulated in $L_{j'}$, contradiction. Since the v_i are all distinct, we see that the set $\{v_i : i \in \mathbb{N}\}$ is an infinite set which is pairwise nonadjacent. This contradicts the assumption that the set of heads of all the edges of $\cup_i T_i$ with tails in X contains no infinite nonadjacent subset.

Now we can apply Claims 1 and 2 together with Lemma 7.33 to conclude that the collection of roots $\{r(T_i) : i \in \mathbb{N}\}$ must contain an M -adjacent pair, giving the lemma.

Still following Robertson and Seymour [391], we specify the third axiom.

Axiom 3. Let $G \in \mathcal{R}$, and suppose that (G_1, G_2) is a separation of G of order $|\{v : v \in \pi(G)\}|$, such that there is no separation (H_1, H_2) of G with $G_1 \leq H_1$ (as hypergraphs) and of order less than that of (G_1, G_2) . Then, there is a march π_1 having the same vertices as $\pi(G)$ such that (G_1, π_1) is simulated in G .

Definition 7.38 (Rotund) Let (T, b) be a tree decomposition.

- (i) We say that $F \subseteq E(T)$ is *rotund* in (T, b) iff for each directed path P with first edge f_1 and last edge f_2 such that f_1 and f_2 have the same order, say, k , and such that no element of $F \cap E(P)$ has order less than k , there is no separation (H_1, H_2) of the hypergraph G corresponding to (T, b) with the subhypergraph $T^{f_i} \subseteq H_i$ and having smaller order than k for $i = 1, 2$.

- (ii) We say that (T, b) is *rotund* iff $E(T)$ is rotund in (T, b) .

Lemma 7.39 Let (T, b) be a tree decomposition with associated rooted hypergraph G in $\mathcal{R}$. Suppose that F is rotund in (T, b) . Then, there is a tree decomposition (T, b') of G such that $V(b(t)) = V(b'(t))$ for all $t \in T$, and such that F is linked in (T, b') .

Proof. For each edge $f \in F$, we shall choose a march π_f with the same vertices as the march of the subhypergraph induced by T^f so that the following property holds.

Property 1. For every directed path P with first edge $e \in F$ and last edge $f \in F$, such that e and f have the same order and no edge of P has lower order, (G^f, π_f) is simulated in (G^e, π_e) , where G^k denoted the (nonrooted) subhypergraph induced by T^k .

The marches are defined inductively. We begin at the root. We suppose that we have defined a subtree S of T with $r(S) = r(T)$ such that π_f is defined so that Property 1 holds for all $f \in E(S) \cap F$, but not yet defined for $f \in F - E(S)$. If

$f \subseteq E(S)$, then we are done. Otherwise, we choose $f \in F - E(S)$ so that every $e \in F$ which lies on the path in T from $r(T)$ to f lies in $E(S)$. Suppose that f has order k . Let Q be the maximal path in T with last edge f such that every edge of Q has order $\geq k$. If no edge of Q in F has order k except f , then we can simply set π_f to be the march corresponding to the rooted subhypergraph from T induced by T^f . In view of this, we assume that $p \geq 2$ and $f_1, \dots, f_p = f$ list, in order of appearance in Q , the edges of $E(Q) \cap F$ of order k .

Claim. *There is a march π_f with the same vertices as the march of the rooted subhypergraph from T induced by T^f , such that (G^f, π_f) is simulated in $(G_{f_{p-1}}, \pi_{f_{p-1}})$.*

To see that the claim holds, suppose that (H_1, H_2) is a separation of order $< k$, the rooted subhypergraph from T induced by $T^{f_{p-1}}$ such that $G^f \subseteq H_1$. Then, $(K_1, K_2) =_{\text{def}} (H_1, H_2 \cup G_{f_{p-1}})$ (as unrooted hypergraphs) is a separation of $G^{r(T)}$, contradicting the rotundity of F in (T, b) . (For both f_{p-1} and f have order k , no edge of the path between them has order below k ; nevertheless, we have that $G^f \subseteq K_1$ and $G^{f_{p-1}} \subseteq K_2$.) Therefore, there is no such separation (H_1, H_2) . This allows us to establish the Claim via an application of Axiom 3 to the separation $((G^f, \pi), (G_{f_{p-1}}^{f_{p-1}}, \pi_{f_{p-1}}))$ of $(G_{f_{p-1}}, \pi_{f_{p-1}})$.

Now, we let $S' = S \cup Q$. We demonstrate that Property 1 holds for every directed path P of S' . By the Claim, (G^f, π_f) is simulated in $(G_{f_{p-1}}^{f_{p-1}}, \pi_{f_{p-1}})$. And by hypothesis, we have that $(G_{f_{p-1}}^{f_{p-1}}, \pi_{f_{p-1}})$ is simulated in (G^i, π_i) for all $i < p-1$. From Axiom 2, we get that Property 1 holds.

Finally, define $b'(t) = b(t)$ if t is not the head of an edge in F , and define $b'(t) = (V(b(t)), \pi_f)$ otherwise. Then, we see that (T, b') satisfies the lemma.

A class $\mathcal{S}$ of star decompositions is called *complete* iff for each (S, σ) in $\mathcal{S}$, and each star decomposition (S, σ') such that for each $s \in S$, $V(\sigma(s)) = V(\sigma'(s))$ and $V(\pi(\sigma(s))) = V(\pi(\sigma'(s)))$, we have that $(S, \sigma') \in \mathcal{S}$.

Lemma 7.40 *Suppose that S is good and complete. Let $(T_i, b_i), i \in \mathbb{N}$, be a countable sequence of tree decompositions with corresponding rooted hypergraphs G_i , each branching of which is in S . Then, there exist $i' > i$ such that $G_{i'}$ is simulated in G_i .*

Proof. Exercise 3.

Now we will apply Thomas' Lemma of the last section. The crucial property we need is the following. (We state this in the language of rooted hypergraphs.)

Weak Thomas Property. *Let G be a graph of classical treewidth w . Then $(G, \emptyset)$ has a tree decomposition (T, b) of width w such that for all $t_1, t_2 \in V(T)$, and all $k \geq 0$, either there are k vertex disjoint paths in G from $V(b(t_1))$ to $V(b(t_2))$, or some edge of the path between t_1 and t_2 has order $< k$.*

Lemma 7.41 (Key Lemma) *If G is a rooted hypergraph of width w , then G has a rotund tree decomposition of width w .*

Proof. Let K be a simple graph with $V(G) = V(K)$ in which v_1 and v_2 are adjacent iff either they are incident in G or v_1 and v_2 are both in $\pi(G)$.

Claim 1 $(K, \emptyset)$ has treewidth $\leq k$.

To establish Claim 1, let (T, b) be a tree decomposition of G of width $\leq w$. For each edge e of K , there is a $t \in V(T)$ such that both ends of e are in $V(b(t))$. Consequently, we can choose edge disjoint rooted subgraphs $b'(t)$ of K such that $V(b(t)) = V(b'(t))$ for each $t \in V(T)$ and such that (T, b') is a tree decomposition of $(K, \emptyset)$. Claim 1 is established.

Claim 2

- (i) *There is a tree decomposition (T, b) of $(K, \emptyset)$ of width $\leq w$ such that for distinct $t_1, t_2 \in V(T)$, and all $k \geq 0$, there are k mutually vertex disjoint paths in K between $V(b(t_1))$ and $V(b(t_2))$ or some edge of the path between t_1 and t_2 in T has order $< k$.*
- (ii) *For every complete subgraph X of K , there exists $t \in V(T)$ such that $V(X) \subseteq V(b(t))$.*

To prove Claim 2, first note that (i) is an immediate consequence of Thomas' lemma and Claim 1.

Part (ii) follows by Lemma 7.35.

From Claim 2 and the fact that (ii) does not depend on which vertex is the root, we can renumber and choose (T, b) [redefining the $\pi(b(t))$'s as necessary] so that $V(\pi(G)) \subseteq V(b(r(T)))$.

From this and Claim 2(ii) again, we conclude that *there is a tree decomposition (T, b') of G such that for each $t \in V(G)$, $V(b(t)) = V(b'(t))$.*

Claim 3 (T, b') is *rotund*.

To establish Claim 3, and hence prove the key Lemma, let P be a directed path of T with first edge f_2 and last f_1 , such that f_1 and f_2 both have order k and no edge between them has order $< k$ in (T, b') . Let the first and last vertices of P be t_2 and t_1 , respectively. By Claim 2(i), there are k mutually vertex disjoint paths $Q_1, \dots, Q_k$ in K between $V(b(t_2))$ and $V(b(t_1))$. Let (H_1, H_2) be a separation of the G as an unrooted hypergraph such that the unrooted subhypergraph from (T, b') induced by $T^{f_1} \subseteq H_1$ and the unrooted subhypergraph from (T, b') induced by $T_{f_2} \subseteq H_2$. Each Q_i has one end in $V(H_1)$ and the other in $V(H_2)$. Recall that for each edge e of K and hence Q_i , either both ends of e are ends in G of some e' in $E(G)$, or both ends of e are in $V(\pi(G))$. In either case, both ends of e lie in some $V(b'(t))$ and, as a consequence, either both lie in H_1 or H_2 . Hence, each $V(Q_i)$ meets $V(H_1 \cap H_2)$ and we conclude that $|V(H_1 \cap H_2)| \geq k$; that is, (H_1, H_2) has order at least k , giving Claim 3.

Let $\mathcal{S}_n$ denote the class of all star decompositions (S, σ) with corresponding rooted hypergraph in $\mathcal{R}$ and such that $|V(\sigma(r(S)))| \leq n$.

Axiom 4. *For each $n \geq 0$, $\mathcal{S}_n$ is good.*

Lemma 7.42 *Let $\{G_i : i \in \mathbb{N}\}$ be a sequence of elements of $\mathcal{R}$ each of treewidth $\leq n$. Then, there exist $i' > i$ such that G_i is simulated in $G_{i'}$.*

Proof. For each G_i , apply Lemma 7.41 to get a rotund tree decomposition. All of its branchings are in $\mathcal{S}_{n+1}$. Since $\mathcal{S}_{n+1}$ is both good and complete, the result follows by Lemma 7.40.

Theorem 7.43 (Robertson and Seymour [391]) *For every k , the class of treewidth $\leq k$ graphs are well-quasi-ordered by the minor relation.*

Proof. Let $\{G_i : i \in \mathbb{N}\}$ be a countable sequence of graphs of treewidth uniformly bounded by k . Let $\mathcal{R}$ be the class of all rooted graphs G which when considered as unrooted graphs, are subgraphs of some G_i . For $H_1, H_2 \in \mathcal{R}$, say that H_1 is simulated in H_2 if $|V(\pi(H_1))| = |V(\pi(H_2))|$ and there exists $H \in \mathcal{R}$ such that as unrooted graphs, $H \subseteq H_2$ and $\pi(H) = \pi(H_2)$, and such that a rooted graph isomorphic to H_1 can be obtained from H by edge contraction. In Exercise 4 we invite the reader to verify Axioms 1-4, and then the result follows from Lemma 7.42.

Exercises 7.3

7.3.1. Prove Lemma 7.31.

(Hint: Use a minimal bad sequence argument. Call a sequence of vertices of M , $z_1, z_2, \dots$, *increasing* if for any $j_1 < j_2$, if $v_{j_1} \in V(T_{j_1})$ and $v_{j_2} \in V(T_{j_2})$, then $i < k$. Let $V_0 = V(M) - \{r(T_i) : i = 1, 2, \dots\}$. We define a *section* to be an increasing infinite sequence of elements of V_0 which is M -nonadjacent. Clearly, we must assume there is a section since, otherwise, V_0 has no infinite M -nonadjacent subset, and hence the sequence of roots satisfies the lemma. Now for each $v \in V(V_0)$, with $v \in V(T_i)$, say, let T^v denote the maximal subtree of T_i with root v . Define inductively a (minimal bad) sequence of elements $x_1, x_2, \dots$ of V_0 such that

- (i) for each $i \geq 1$, there is a section with first i terms $x_1, \dots, x_i$,
- (ii) for each $i \geq 1$ there is no section with first i terms $x_1, \dots, x_{i-1}, x$, where $x \in V(T^{x_i}) - \{x_i\}$.

Let $X = \{x_1, x_2, \dots\}$, be the section so defined. Prove that X satisfies the theorem. First, show that for $j_2 > j_1$ if z is the head of an edge of $T_1 \cup T_2 \cup \dots$ with tail x_{j_1} , then x_{j_2} is not M -adjacent to z . Then, prove that if we let Y denote the set of all heads of edges of $T_1 \cup T_2 \cup \dots$ with tails in X , then no infinite sequence of elements of Y is a section, from which the lemma follows.)

7.3.2. Prove Lemma 7.35.

7.3.3. Prove Lemma 7.40.

(Hint. Since $\mathcal{S}$ is good and contains the branching of (T_i, b_i) at its root, it follows that each G_i is in $\mathcal{R}$. Now apply Lemma 7.39 to get a linked decomposition (T_i, b'_i) of G_i with $V(b_i(t)) = V(b'_i(t))$ for all $t \in V(T)$. By (iv) and (v) of the definition of tree decomposition, we see that $V(\pi(b'_i(t))) = V(\pi(b_i(t)))$

for all $t \in V(T_i)$. By symmetry, each branching of (T, b'_i) is in S . Now, apply the goodness of S to get the result.)

7.3.4. For the simulation relation on the class $\mathcal{R}$ of Theorem 7.43, verify Axioms 1-4.

7.4 Excluding a Forest

The Graph Minor Theorem demonstrates that *any* minor closed class has a finite obstruction set. However, at present, we do not know many of these obstruction sets; nor do we know the types of classes one gets from excluding a given set as minors. There are a number of important such “excluded minor” theorems. One that we used heavily in the proof of Seese’s theorem, (see Section 6.6) was that if a class $\mathcal{C}$ of graphs *exclude* a given planar graph H as a minor (i.e., no graph in the collection $\mathcal{C}$ has H as a minor), then the class has uniformly bounded treewidth (Robertson and Seymour [393]). Unfortunately, we do not have enough room in this book to include a proof of this result, but we can give a proof of a related result of Bienstock, Robertson, Seymour and Thomas via an elegant proof of Diestel [156] to give the reader some idea of this style of result. More comments are given in the Historical Notes.

Theorem 7.44 (Bienstock, Robertson, Seymour & Thomas [61]) *Let F be a forest, and $\mathcal{C}$ a class of graphs excluding F . Then, the pathwidth of the members of $\mathcal{C}$ is bounded $< |F| - 1$.*

Proof. (Diestel [156]) The proof is inductive and relies upon the following definitions. For $X \subseteq V(G)$, let $G[X]$ denote the subgraph induced by X and $\text{attach}(X)$ the set of vertices that attach to $G - X$, namely those vertices in X which have neighbors in $G - X$ (Bienstock et al.[61]). For each $n > 0$ and graph G , let $\mathcal{B}_n$ denote the unique minimal subset of the power set of $V(G)$ such that $\emptyset \in \mathcal{B}_n$ and such that for all $X \in \mathcal{B}_n$,

$$X \subseteq Y \subseteq V(G) \text{ and } |\text{attach}(X)| + |Y - X| \leq n \text{ implies } Y \in \mathcal{B}_n.$$

Notice that a set X is in $\mathcal{B}_n$ iff there is a sequence of subsets of $V(G)$ $\emptyset = X_0 \subseteq X_1 \subseteq \dots \subseteq X_s = X$ such that for all $r < s$, $|\text{attach}(X_r)| + |X_{r+1} - X_r| \leq n$.¹⁰

Lemma 7.45 *If $V(G)$ is $\mathcal{B}_n$, then G has pathwidth $< n$.*

Proof. If $V(G)$ is in $\mathcal{B}_n$ then there is a sequence $\emptyset = X_0 \subseteq X_1 \subseteq \dots \subseteq X_s = V(G)$. Now, consider $W_r = \text{attach}(X_{r-1}) \cup (X_r - X_{r-1})$.

Diestel’s simplification of the earlier proof uses the following elegant observation.

¹⁰The reader should note that if $W_1, \dots, W_d$ are the bags in order of a path decomposition of G , then the sets $W_1 \cup \dots \cup W_r$ are in $\mathcal{B}_n$. The reader should compare this with the *running intersection property* of Exercise 6.2.1.

Lemma 7.46 (Diestel [156]) *Suppose that Y is in $\mathcal{B}_n$ and $Z \subseteq Y$. Further suppose that there is a set $\{P_z : z \in \text{attach}(Z)\}$ of disjoint paths such that each P_z starts in Z , has no other vertex in Z , and ends in $\text{attach}(Y)$. The Z is in $\mathcal{B}_n$.*

Proof. Given the hypotheses, there exist $\emptyset = Y_0 \subseteq Y_1 \subseteq \dots \subseteq Y_s = Y$ with $|\text{attach}(Y_r)| + |Y_{r+1} - Y_r| \leq n$ for all $r < s$. Let $Z_r = Y_r \cap Z$. The claim is that $|\text{attach}(Z_r)| + |Z_{r+1} - Z_r| \leq n$, giving the lemma. Notice that $Z_{r+1} - Z_r \subseteq Y_{r+1} - Y_r$. Therefore, we need only prove that $|\text{attach}(Z_r)| \leq |\text{attach}(Y_r)|$.

So suppose that $z \in \text{attach}(Z_r) - \text{attach}(Y_r)$. It follows that z has a neighbor in $Y_r - Z_r = Y_r - Z$, and hence $z \in \text{attach}(Z)$. By the hypotheses of the theorem, we have a path P_z from Z_r , and hence Y_r to $\text{attach}(Y)$, and $\text{attach}(Y_r)$ separates these two sets in G . It follows that P_z has a vertex $y \neq z$ in $\text{attach}(Y_r)$. Since z is the only vertex of P_z in Z , it must be that $y \in \text{attach}(Y_r) - \text{attach}(Z_r)$. Since the paths are disjoint, for each such z , there is a distinct y , it must be that $|\text{attach}(Z_r) - \text{attach}(Y_r)| \leq |\text{attach}(Y_r) - \text{attach}(Z_r)|$, and hence $|\text{attach}(Z_r)| \leq |\text{attach}(Y_r)|$.

Proof of Theorem 7.44. We assume that F is given and is a tree. Suppose that G has pathwidth $\geq n = |F| - 1$. Using topological ordering, say, assume that $V(F) = \{v_1, \dots, v_n\}$ such that for $i \leq n$, $F[v_1, \dots, v_i]$ is a subtree of F (i.e., connected) and hence exactly one v_i for $i < n$ is adjacent to v_{i+1} .

For $0 \leq j \leq n$, we shall define a set $\mathcal{S}^i = \{S_0^i, \dots, S_i^i\}$ of disjoint subgraphs of G so that $S_j^k \subseteq S_j^i$ for all $j \leq k \leq i$, so that for $d > 0$, S_d^i is connected, and such that (i)-(iv) below hold. We define $X^i = V(\cup\{S_j^i : j \leq i\})$.

- (i) Whenever $1 \leq j < k \leq i$ and $v_j v_k \in E(F)$, G contains an edge between S_j^i and S_k^i , so that $F[v_1, \dots, v_i]$ is a minor of $G[S_1^i \cup \dots \cup S_i^i]$.
- (ii) $|\text{attach}(X^i)| = i$ and $|V(S_j^i) \cap \text{attach}(X^i)| = 1$ for $1 \leq j \leq i$.
- (iii) X^i is in $\mathcal{B}_n$.
- (iv) $|\text{attach}(X)| > i$ for all $X \in \mathcal{B}_n$ with X^i a proper subset of X .

First, let S_0^0 be the inclusion maximal subgraph of G with $V(S_0^0)$ in $\mathcal{B}_n$, and $|\text{attach}(S_0^0)| = 0$. Clearly, (i)-(iv) hold for S_0^0 . For an induction, assume that $\mathcal{S}^i$ has been defined for some $i \leq n$.

Case 1. $i = 0$. Then, $G - S_0^0$ is nonempty since we can apply Lemma 7.45 to the fact that $V(S_0^0) \in \mathcal{B}_n$, yet $V(G) \notin \mathcal{B}_n$. Thus, in this case, we let x be any vertex of $G - S_0^0$. Let $X = V(S_0^0) \cup \{x\}$.

Case 2. $i > 0$. In this case, find the unique $j < i$ such that $v_j v_i$ is an edge of F , and let $x \in G - X^i$ be a neighbor of the unique vertex in $V(S_j^i) \cap \text{attach}(X^i)$. We let $X = X^i \cup \{x\}$.

If $i = n$, then (i) and the choice of x together imply that F is a minor of $G[X]$, giving (i)-(iv). Thus, assume that $i < n$. Notice that by (ii) and (iii), we have $X \in \mathcal{B}_n$. Also, it follows that $|\text{attach}(X)| > i$ by (iv). In turn, this means that $\text{attach}(X) = \text{attach}(X^i) \cup \{x\}$ and $|\text{attach}(X)| = i + 1$. Let Y be the inclusion

maximal member of $\mathcal{B}_n$ with both $X \subseteq Y$ and $|\text{attach}(Y)| = i + 1$. (This set is the later X^{i+1} .)

Now, we apply Menger's theorem (Lemma 7.23). Menger's theorem shows that $G[Y]$ contains a set P of disjoint paths from X to $\text{attach}(Y)$ and a separator D between these sets, each path having exactly one member of D . Now, let Z denote the union of D and all the vertex sets of components of $G - D$ not meeting X . Then, X^i is a proper subset of X which is a subset of Z . Furthermore, by choice of D and definition of $\text{attach}(Y)$, we see that $Z \subseteq Y$. Observe that $\text{attach}(Z) = D$. Consequently, by Lemma 7.46, Z is in $\mathcal{B}_n$. By (iv), $|P| = |D| = |\text{attach}(Z)| > i$. By definition of $\text{attach}(X)$, each of the paths in P meet $\text{attach}(X)$. We conclude that $i < |P| \leq |\text{attach}(X)| = i + 1$, and hence the paths in P give a perfect matching between $\text{attach}(X)$ and $\text{attach}(Y)$.

We are now ready to define S^{i+1} . Let $S_0^{i+1} = S_0^i \cup G[Y - (X \cup V(\cup P))]$. For $1 \leq j \leq i$, let x_j be the unique vertex of S_j^i in $\text{attach}(X^i)$ and the path P_j in P containing it. Define S_j^{i+1} as the union of S_j^i and the final segment of P_j starting at x_j . Finally, we let S_{i+1}^{i+1} denote the final segment from x of the path in P containing x . Then, $X^{i+1} = V(\cup S^{i+1}) = Y$. We note that S^{i+1} satisfies (i) by choice of x , (ii), and (iii) since $X^{i+1} = Y$, and, finally, (iv) by choice of Y and the fact that $X^i \subseteq Y = X^{i+1}$, together with (iv) for i .

The theorem now follows by Lemma 7.45 applied to the definition of X in the case $i = n$.

Historical Notes

The material of this section is drawn from Diestel [156]. The first “excluding forest” result was due to Robertson and Seymour [389], who demonstrated that for each forest F , there exists an integer m such that each class of graphs that exclude F has pathwidth below m . The argument was complex and long. Subsequently, the argument was sharpened to the bound $m = |F| - 1$ of this section and considerably simplified by Bienstock et al. [61]. Nevertheless, the argument still relied on a minimax theorem and still remained relatively involved. The short proof of the present section is due to Diestel [156], who eliminated the need for the minimax theorem. We remark that the bound $|F| - 1$ is sharp because K_{n-1} has pathwidth $n - 2$ but has no forest minor on n vertices. Furthermore, if F is not a forest, then Theorem 7.44 fails in general. For instance, trees can have arbitrarily large pathwidth, but cannot contain cycles as minors. We also refer the reader to Corollary 7.62, where we give an easy proof of the basic Robertson-Seymour result using parsing operators. This is based upon the pebbling algorithm of Cattell, Dinneen, and Fellows [117].

As we remarked earlier, there are many beautiful “excluded minor” theorems, and the idea is a cornerstone of the proof of the Graph Minor Theorem. Noteworthy is that of Archdeakon [30] and Glover, Huneke, and Wang [244] who exhibited a 35-element obstruction set for the property of being embeddable upon the projective plane. Robertson and Seymour [393] proved that if a class of graphs excludes a

planar graph, then it must have bounded treewidth, and the treewidth bound can be theoretically calculated by the methods of Fellows and Langston [217, 218]. However, there are very few obstruction sets that have been explicitly calculated due to the complexity of the computations currently involved. For instance, the obstruction set for embeddability upon the torus is, at present, open. Finally, we remark that there are many other “forbidden subobject” results for other orderings. We met one in Leizhen Cai’s theorem of Section 3.2.2.

7.5 Connections with Automata Theory and Boundaried Graphs

Well-quasi-orderings have deep connections with automata theory and finite-state recognition of graph families of bounded treewidth and pathwidth. In this section, we will examine some of these connections which have been explored by a number of researchers, including Abrahamson and Fellows [6, 7], Ehrenfeucht et al. [193], and Luca and Varricchio [338].

Definition 7.47 (Canonical Partial Order) Let $L \subseteq \Sigma^*$. We define the *canonical quasi-ordering induced by L* , $\leq_L$, via

$$x \leq_L y \text{ iff } \forall u, v \in \Sigma^*, u y v \in L \text{ implies } u x v \in L.$$

Lemma 7.48 (Abrahamson and Fellows [6, 7]) Suppose that $\leq_L$ is a well-quasi-ordering of Σ^* . Then, $\leq_{L^c}$ is a well-quasi-ordering of Σ^* , where L^c denotes $\Sigma^* - L$, the complement of L .

Proof. Suppose not. Then, there is a bad L^c sequence $\mathcal{B} = x_0, x_1, \dots$. Now, by Ramsey’s theorem, $\mathcal{B}$ either has an infinite $\leq_L$ ascending sequence (which it cannot since it is bad) and infinite $\leq_L$ descending sequence, or an infinite $\leq_L$ antichain. (For Ramsey’s theorem, see Theorem 7.3.) Suppose that $\mathcal{B}$ has an infinite $\leq_L$ antichain. Call the antichain $\mathcal{A} = a_0, a_1, \dots$. Then, for each $i \neq j$, there exists p, q, r, s such that

$$r a_i p \in L^c \wedge r a_j p \notin L^c \text{ and } s a_i q \notin L^c \wedge s a_j q \in L^c.$$

But this means that

$$r a_i p \notin L \wedge r a_j p \in L \text{ and } s a_i q \in L \wedge s a_j q \notin L.$$

Hence, the sequence $\mathcal{A}$ is an L -antichain too, contradicting the fact that $\leq_L$ is a **WQO**.

Therefore, $\mathcal{B}$ has an infinite descending sequence. Renaming, we have $x_0 \geq_{L^c} x_1 \geq_{L^c} x_2 \dots$. This corresponds to an L -ascending sequence, $x_0 \leq_L x_1 \leq_L \dots$. Since $\mathcal{B}$ is bad, there exists a sequence of pairs of witness strings

$$\mathcal{W} = (w_{1,2}, v_{1,2}), (w_{2,3}, v_{2,3}), \dots$$

such that for all i ,

$$w_{i,i+1}x_{i+1}v_{i,i+1} \in L^c \text{ but } w_{i,i+1}x_i v_{i,i+1} \notin L^c.$$

Now, since $\leq_L$ is a **WQO**, $\leq_L \times \leq_L$ is a **WQO** of $\Sigma^* \times \Sigma^*$. Hence, $\mathcal{W}$ contains an infinite $\leq_L \times \leq_L$ ascending sequence. Hence, there exist indices $i < j - 1$ and $w, w', v, v' \in \Sigma^*$ with (i)-(iii) below holding:

- (i) $x_i \leq_L x_{i+1} \leq_L x_j \leq_L x_{j+1}$.
- (ii) $wx_i v \notin L \wedge wx_{i+1} v \in L$ and $w'x_j v' \notin L \wedge w'x_{j+1} v' \in L$.
- (iii) $w \leq_L w'$ and $v \leq_L v'$.

But this is a contradiction:

$$\begin{aligned} w'x_{j+1}v' \in L &\rightarrow wx_{j+1}v' \in L \text{ (since } w \leq_L w') \\ &\rightarrow wx_{j+1}v \in L \text{ (since } v \leq_L v') \\ &\rightarrow wx_i v \in L \text{ (since } x_i \leq_L x_{j+1}). \end{aligned}$$

Lemma 7.48 has the following consequence which neatly ties together the concepts of well-quasi-ordering and being finite state. It is a well-quasi-ordering analog of the Myhill–Nerode theorem.

Theorem 7.49 (Abrahamson and Fellows [6, 7] and Luca and Varricchio [338])
A formal language L is finite state iff $\leq_L$ is a well-quasi-ordering.

Proof. Suppose that $\leq_L$ is a **WQO**. Recall from Definition 6.22 that $x \approx_L y$ means that for all $u, v \in L$,

$$uxv \in L \text{ iff } uyv \in L.$$

Hence, $x \approx_L y$ iff $x \leq_L y \wedge x \leq_{L^c} y$. By Lemma 7.48, if $\leq_L$ is a **WQO**, then so is L^c . Hence, $\leq_L \times \leq_{L^c}$ is a **WQO**, and hence $\approx_L$ has finite index. By Exercise 1, this implies that $\sim_L$ has finite index. By the Myhill–Nerode Theorem, L is finite state.

Conversely, if L is finite state, then $\sim_L$ has finite index by the Myhill–Nerode Theorem. Hence, $\approx_L$ has finite index. If $\leq_L$ is not a **WQO**, then there is an L -bad sequence $x_0, x_1, \dots$. Clearly, this L -bad sequence generates infinitely many $\approx_L$ equivalence classes.

There is a graph-theoretic analog of Theorem 7.49.

Definition 7.50 (Canonical Partial Order for Graphs) Let L be a family of t -boundaried graphs in $\mathcal{U}_{small}^t$. We write $G_1 \leq_L G_2$ iff for all $Z \in \mathcal{U}_{small}^t$,

$$G_2 \oplus Z \in L \text{ implies } G_1 \oplus Z \in L.$$

Theorem 7.51 (Abrahamson and Fellows [7]) *A family F of graphs is t -finite state iff $\leq_F$ is a well-quasi-ordering of $\mathcal{U}_{small}^t$.*

Proof. The proof is a simple analog of Theorem 7.49. We begin with an analog of Lemma 7.48

Lemma 7.52 (Abrahamson and Fellows [7]) *If F is a family of graphs for which $\leq_F$ is a well-quasi-order on $U_{large}^t(U_{small}^t)$, then $\leq_{F^c}$ is a well-quasi-order on $U_{large}^t(U_{small}^t)$, where F^c denotes the complement of F .*

Proof. As in Lemma 7.48, if the statement is false, then we may take $(X_1, X_2, \dots)$ to be a bad $\leq_{F^c}$ sequence which is $\leq_F$ ascending. For $i < j$ let $Z_{i,j}$ be a choice of evidence for the badness of the sequence: $X_i \oplus Z_{i,j} \in F$ and $X_j \oplus Z_{i,j} \in F^c$. Consider the sequence of t -boundaried graphs $(Z_{1,2}, Z_{3,4}, Z_{5,6}, \dots)$. Since $\leq_F$ is a well-quasi-order, we must obtain the situation (reindexing for convenience): $X_1 \leq_F X_2 \leq_F X_3 \leq_F X_4$, $X_1 \oplus Z_{1,2} \in F$, $X_2 \oplus Z_{1,2} \notin F$, $X_3 \oplus Z_{3,4} \in F$, $X_4 \oplus Z_{3,4} \notin F$, and $Z_{1,2} \leq_F Z_{3,4}$. But this is a contradiction, since $X_3 \oplus Z_{3,4} \in F$ implies $X_3 \oplus Z_{1,2} \in F$, and this implies $X_2 \oplus Z_{1,2} \in F$.

Now, note that for t -boundaried graphs X and Y , $X \sim_F Y$ if and only if $X \leq_F Y$ and $X \leq_{F^c} Y$, by definition. If $\leq_F$ is a well-quasi-order, then by Lemma 7.52, $\leq_{F^c}$ is a well-quasi-order. The Cartesian product of $\leq_F$ and $\leq_{F^c}$ is then a well-quasi-order, and, therefore, $\sim_F$ has finite index. The result then follows by the Myhill–Nerode theorem for t -boundaried graphs.

As we will see later in this section, $\leq_L$ and finite obstructions can be used to provide smooth proofs for algorithms on t -boundaried graphs and formal languages. Before we look at one such application, we would like to point out that $\leq_L$ is also strongly related to *arbitrary* well-quasi-orderings of $\Sigma^*(U_{small}^t)$.

Definition 7.53 (Well-Behaved Quasi-Order)

- (i) A quasi-order $\leq$ on Σ^* is *well-behaved* if it is (1) Noetherian, and (2) a congruence with respect to concatenation; that is, $x \leq x'$ and $y \leq y'$ imply $xx' \leq yy'$.
- (ii) Similarly, a quasi-order $\leq$ on U_{small}^t is *well-behaved* if (1) $\leq$ is Noetherian and (2) $\leq$ is a congruence with respect to $\oplus$; that is, $X \leq Y$ and $X' \leq Y'$ imply $X \oplus X' \leq Y \oplus Y'$.

Theorem 7.54 (Abrahamson and Fellows [7]) *Let $\mathcal{R} \in \{\Sigma^*, U_{small}^t\}$. A well-behaved quasi-order $\leq$ on $\mathcal{R}$ is a WQO if and only if $\leq_L$ is a WQO for every lower ideal L of $\langle \mathcal{R}, \leq \rangle$.*

Lemma 7.55 (Abrahamson and Fellows [7]) *Let $\leq$ be a Noetherian quasi-order on U_{small}^t (on Σ^* .) If every lower ideal of $\leq$ is finite state, then $\leq$ is a well-quasi-order.*

Proof. If $\leq$ is not a well-quasi-order, then there is an infinite antichain A . Each (possibly infinite) subset S of A determines a distinct lower ideal $I_S = \{x : x \leq a \in S\}$. If S and T are subsets of A with $S \neq T$, then $I_S \neq I_T$. Thus, A yields a collection of uncountably many distinct lower ideals. Since there are only countably many finite-state automata, there must be a lower ideal I_S that is not finite-state.

Proof of Theorem 7.54. Suppose that $\mathcal{R} = \Sigma^*$. By Lemma 7.55 and Theorem 7.49, it is enough to argue that if L is a lower ideal of a well-behaved quasi-order

$\leq$ on Σ^* , then $\leq_L$ is a well-quasi-order. If not, then there is a bad $\leq_L$ sequence $(x_1, x_2, \dots)$. But then, since $\leq$ is a WQO, there are indices $i < j$ such that $x_i \leq x_j$. Since it is not the case that $x_i \leq_L x_j$, there are words $u, v \in \Sigma^*$ such that $ux_i v \in L$ and $ux_j v \notin L$. This contradicts that L is a lower ideal, since $x_i \leq x_j$ implies $x_i v \leq x_j v$, which implies $ux_i v \leq ux_j v$.

The case that $\mathcal{R} = \mathcal{U}_{small}^t$ is entirely similar and is left to the reader (Exercise 1).

We conclude this section with the Cattell-Dinneen-Fellows *FPT* approximation algorithm for pathwidth. As we have seen in Section 6.3, after a long series of improvements, the best algorithm for finding tree decompositions (and path decompositions) is Bodlaender's algorithm. For each fixed k , this algorithm in time $O(2^{ck^3} n)$ either determines that the treewidth is greater than k or produces a tree decomposition of width at most k . By first running this algorithm and then applying the algorithm of [84] and [301], a similar result holds for pathwidth. Both Bodlaender's algorithm and the Bodlaender-Kloks algorithm are quite complicated.

Cattell, Dinneen, and Fellows [117] produce a very simple algorithm based on "pebbling" the graph using a pool of $O(2^k)$ pebbles, that in linear time (for fixed k), either determines that the pathwidth of a graph is more than k or finds a path decomposition of width at most the number of pebbles actually used. The main advantages of this algorithm over previous results are (1) the simplicity of the algorithm and (2) the improvement of the hidden constant for a determination that the pathwidth is greater than k . The main disadvantage is in the width of the resulting "approximate" decomposition when the width is less than or equal to k .

It is easy to see that the family of graphs of pathwidth at most t is a lower ideal in the topological (and minor) order. We need the following facts.

Lemma 7.56 (Folklore) *If G has pathwidth t and $|V(G)| = n$, then $|E(G)| \leq nt - \frac{(t^2+t)}{2}$.*

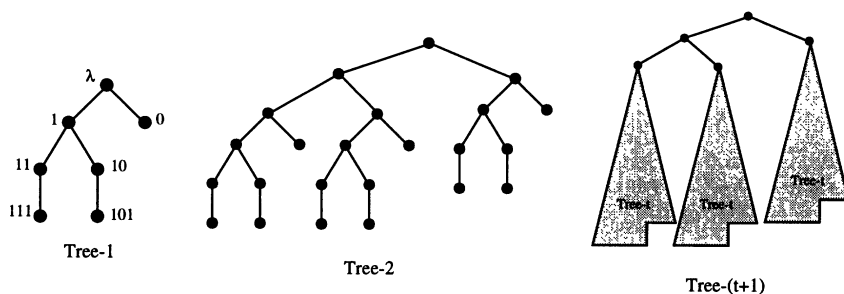
Proof. Exercise 2.

Let B_h denote the complete binary tree of height h and order $2^h - 1$. Let $h(t)$ be the least value of h such that $B_{h(t)}$ has pathwidth greater than t , and let $f(t)$ be the number of vertices of $B_{h(t)}$. To get a bound for $f(t)$, $B_{h(t)}$ needs to contain at least one obstruction of pathwidth t .

Lemma 7.57 (Ellis, Sudborough, and Turner [194]) *All topological tree obstructions of pathwidth t can be recursively generated by the following rules:*

- (i) *The single edge tree K_2 is the only obstruction of pathwidth 0.*
- (ii) *If T_1 , T_2 , and T_3 are any three tree obstructions for pathwidth t , then the tree T consisting of a new degree 3 vertex attached to any vertex of T_1 , T_2 , and T_3 is a tree obstruction for pathwidth $t + 1$.*

Proof. Exercise 3.

FIGURE 7.6. Embedding Pathwidth t Tree Obstructions, Tree- t , in Binary Trees.

From Lemma 7.57, we see that the orders of the tree obstructions of pathwidth t are precisely $(5 \cdot 3^t - 1)/2$, (e.g., orders 2, 7, 22 and 57 for pathwidth $t = 0, 1, 2$ and 3). We can easily embed at least one of the tree obstructions for pathwidth t , as shown in Figure 7.6, in the complete binary tree of height $2t + 2$. Thus, the complete binary tree of order $f(t) = 2^{2t+2} - 1$ has pathwidth greater than t .

Using the $f(t)$ bound given above, the main result of the Cattell, Dinneen, and Fellows article [117] now follows:

Theorem 7.58 (Cattell, Dinneen, and Fellows [117]) *Let H be an arbitrary undirected graph, and let t be a positive integer. One of the following two statements must hold:*

- (a) *The pathwidth of H is at most $f(t) - 1$.*
- (b) *H can be factored: $H = A \oplus B$, where A and B are boundaried graphs with boundary size $f(t)$, the pathwidth of A is greater than t and less than $f(t)$.*

Proof. We describe an algorithm that terminates either with a path decomposition of H of width at most $f(t) - 1$, or with a path decomposition of a suitable factor A with the last vertex set of the decomposition consisting of the boundary vertices.

If we find an homeomorphic embedding of the *guest tree* $B_{h(t)}$ in the *host graph* H , then we know that the pathwidth of H is greater than t . During the search for such an embedding, we work with a partial embedding. We refer to the vertices of $B_{h(t)}$ as *tokens*, and call tokens *placed* or *unplaced* according to whether or not they are mapped to vertices of H in the current partial embedding. A vertex v of H is *tokened* if a token maps to v . At most one token can be placed on a vertex of H at any given time. We recursively label the tokens by the following standard rules:

- (1) The root token is labeled by the empty string λ .
- (2) The left child token and right child token of a height h parent token $P = b_1 b_2 \dots b_h$ are labeled $P \cdot 1$ and $P \cdot 0$, respectively.

Let $P[i]$ denote the set of vertices of H that have a token at time step i . The sequence $P[0], P[1], \dots, P[s]$ will describe a path decomposition either of the

entirety of H or of a factor A fulfilling the conditions of Theorem 7.58. In the case of outcome (b), the boundary of the factor A is indicated by $P[s]$.

The placement algorithm is described as follows. Initially, consider that every vertex of H is colored *blue*. In the course of the algorithm, a vertex of H has its color changed to *red* when a token is placed on it, and stays red if the token is removed. Only blue vertices can be tokened, and so a vertex can only be tokened once.

```

function GrowTokenTree
1  if root token  $\lambda$  is not placed on  $H$  then
    arbitrarily place  $\lambda$  on a blue vertex of  $H$ 
  endif
2  while there is a vertex  $u \in H$  with token  $T$  and blue neighbor  $v$ ,
    and token  $T$  has an unplaced child  $T \cdot b$  do
    2.1 place token  $T \cdot b$  on  $v$ 
  endwhile
3  return { tokened vertices of  $H$  }

program PathDecompositionOrSmallFatFactor
1   $i \leftarrow 0$ 
2   $P[i] \leftarrow$  call GrowTokenTree
3  until  $|P[i]| = f(t)$  or  $H$  has no blue vertices repeat
    3.1 pick a token  $T$  with untokened children
    3.2 remove  $T$  from  $H$ 
    3.3 if  $T$  had one tokened child then
        replace all tokens  $T \cdot b \cdot S$  with  $T \cdot S$ 
    endif
    3.4  $i \leftarrow i + 1$ 
    3.5  $P[i] \leftarrow$  call GrowTokenTree
  enduntil
done

```

Before we prove the correctness of the algorithm, we note some properties: (1) the root token will need to be placed (step 1 of the `GrowTokenTree`) at most once for each component of H ; (2) the `GrowTokenTree` function only returns when $B_{h(t)}$ has been embedded in H or all parent tokens of degree less than 2 have no blue neighbors; (3) the algorithm will terminate since during each iteration of step 3.2, a tokened red vertex becomes untokened, and this can happen at most n times, where n denotes the order of the host H .

Since tokens are placed only on blue vertices and are removed only from red vertices, it follows that the interpolation property of a path-decomposition is satisfied. Suppose the algorithm terminates at time s with all of the vertices colored red. To see that the sequence of vertex sets $P[0], \dots, P[s]$ represents a path decomposition of H , it remains only to verify that for each edge (u, v) of H , there is a time i with both vertices u and v in $P[i]$. Suppose vertex u is tokened first and

untokened before v is tokened. But vertex u can be untokened only if all neighbors, including vertex v , are colored red [see step 3.1 and comment (2) above].

Suppose the algorithm terminates with all tokens placed. The argument above establishes that the subgraph A of H induced by the red vertices with boundary set $P[s]$ has pathwidth at most $f(t)$. To complete the proof, we argue that in this case, the sequence of token placements establishes that A contains a subdivision of $B_{h(t)}$, and hence must have pathwidth greater than t . Since the `GrowTokenTree` function only attaches pendant tokens to parent tokens, we need to only observe that the operation in step 3.3 subdivides the edge between T and its parent.

Corollary 7.59 (Cattell, Dinneen, and Fellows [117]) *Given a graph H of order n and an integer t , there exists a $O(n)$ time algorithm that gives evidence that the pathwidth of H is greater than t or finds a path decomposition of width at most $O(2^t)$.*

Proof. We show that program `PathDecompositionOrSmallFatFactor` runs in linear time. First, if H has more than $t \cdot n$ edges, then the pathwidth of H is greater than t . By the proof of Theorem 7.58, the program terminates with either the embedded binary tree as evidence, or a path decomposition of width at most $f(t)$.

Note that the guest tree $B_{h(t)}$ has constant order $f(t)$, and so token operations that do not involve scanning H are constant time. In the function `GrowTokenTree`, the only nonconstant time operation is the check for blue neighbors in step 2. While scanning the adjacent edges of vertex u , any edge to a red vertex can be removed, in constant time. Edge (u, v) is also removed when step 2.1 is executed. Therefore, across all calls to `GrowTokenTree`, each edge of H needs to be considered at most once, for a total of $O(n)$ steps. In the program `PathDecompositionOrSmallFatFactor`, all steps except for `GrowTokenTree` are constant time. The total number of iterations through the loop is bounded by n , by the termination argument following the program.

The last result for this section shows that we can improve the pathwidth algorithm by restricting the guest tree. This allows us to use the subdivided tree obstructions given in Figure 7.6.

Corollary 7.60 (Cattell, Dinneen and Fellows [117]) *Any subtree of the binary tree $B_{h(t)}$ that has pathwidth greater than t may be used in the algorithm for Theorem 7.58.*

Proof. The following simple modifications allow the algorithm to operate with a subtree. The subtree is specified by a set of *flagged* tokens in $B_{h(t)}$. At worst, the algorithm can potentially embed all of $B_{h(t)}$.

In step 2 of `GrowTokenTree`, the algorithm only looks for a flagged untokened child $T \cdot b$ to place, since unflagged tokens need not be placed. The stopping condition in step 3 of `PathDecompositionOrSmallFatFactor` is changed to “all flagged tokens of $B_{h(t)}$ are placed or . . .,” so that termination occurs as soon as the subtree has been embedded. The relabeling in step 3.3 can place unflagged tokens on vertices of H , since all the rooted subtrees of a fixed height are not isomorphic.

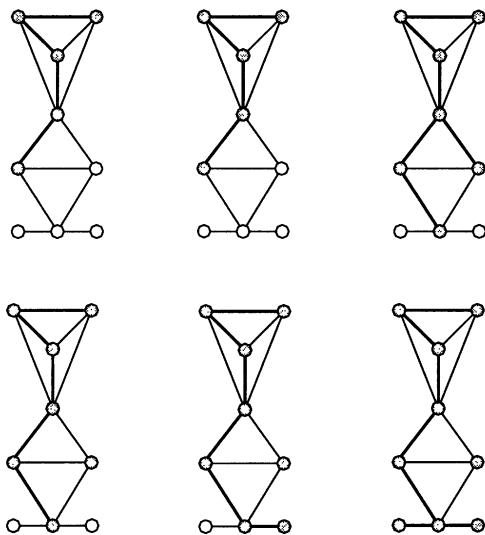


FIGURE 7.7. Path Decomposition of Width 5

If that happens, we expand our guest tree with those new tokens. (It is easy to see that the new guest is still a tree.) These tokens can then get relabeled by future edge subdivisions that occur above the token in the host tree. Thus, duplication of token labels will not happen, and the $f(t)$ width bound is preserved.

Example 7.61 Using $K_{1,3}$ as the guest tree of pathwidth 2, the program trace represented in Figure 7.7 terminates with all vertices colored red (gray) yielding a path decomposition of width 5.

Observe that the Cattell-Dinneen-Fellows pathwidth algorithm provides an easy proof of basically the main result we saw in Section 7.4 from Bienstock, Robertson, Seymour and Thomas [61] (or its earlier variant in Robertson and Seymour [389]) that for any forest F , there is a constant c , such that any graph not containing F as a minor has pathwidth at most c .

Corollary 7.62 *Every graph with no minor isomorphic to forest F , where F is a minor of a complete binary tree B , has pathwidth at most $c = |B| - 2$.*

Proof. Without loss of generality, we can run our pathwidth algorithm using (as the guest) any subtree T of B that contains F as a minor. Since at most $|B| - 1$ pebbles are used when we do *not* find an embedding of T (in any host graph), the resulting path decomposition has width at most $|B| - 2$.

Note that the Cattell-Dinneen-Fellows constant c is identical to the one given in [61] when $F = B$. This constant is the best possible in that case. We refer to Section 7.4 for more details on excluding forests.

Exercises 7.5

- 7.5.1.** Verify the proof of Theorem 7.54 in the case $\mathcal{R} = \mathcal{U}_{small}^t$.
- 7.5.2.** Prove Lemma 7.56; that is, prove that if G has pathwidth t , and n vertices, then $|E(G)| \leq nt - \frac{(t^2+t)}{2}$.
- 7.5.3.** Prove Lemma 7.57.
(Hint: Use a case analysis and induction.)

7.6 A Sketch of the Proof of the Graph Minor Theorem

We have proved the Graph Minor Theorem (GMT) for bounded treewidth, which covers a large proportion of the applications to fixed-parameter tractability. A proof of the full Graph Minor Theorem is currently far beyond what we could hope to present in a book such as this. Despite the daunting length and intricacy of detail in the proof of the GMT, there is a beautiful sequence of structural stages and overall architecture for which we offer the following sketch.

Central to the argument are several stages of excluded substructure theorems that can be organized in an elegant table. By an *excluded substructure theorem*, we mean a statement about “what happens” when a graph of a certain generic (parameterized) kind is forbidden to be a minor. For example, consider the consequences for a graph G if it is forbidden to have a path of length 5 as a minor. Obviously, this means that G cannot contain any path of length 5 or more. Consequently, G must be approximately “pointlike” and could be a graph such as shown in Figure 7.8. In fact, it is possible to formulate a notion of graph *pointwidth* and to prove that every graph that excludes a path of length k (as a minor) has pointwidth at most $f(k)$, for some function f .

At the next stage, the excluded substructure is a tree (generically, the height k complete binary tree). The result of excluding a fixed tree T as a minor of a graph G is that G is forced to have a “pathlike” structure. The parameterized structure theorem at this level says that if G is a graph that does not have the complete binary tree B_k as a minor, then the pathwidth of G is bounded by $f(k)$. (See Theorem 7.44.) Note that every tree is a minor of some B_k , and every tree of sufficiently large pathwidth has B_k as a minor (Theorem 7.58).

Summarising matters so far: if a path is excluded, then G is pointlike; if a tree is excluded, then G is pathlike. The next stage considers the exclusion of

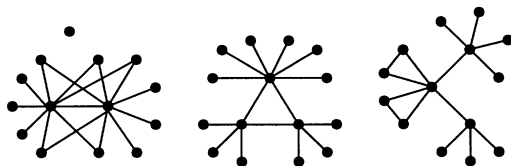


FIGURE 7.8. Some Graphs Without a P_6 Minor

planar graphs, for which the generalized grids are generic (Observation 6.87). The structure theorem at this stage establishes that excluding a planar graph as a minor forces G to have a treelike structure, or, to be precise, to have bounded treewidth. The proof of even this theorem is beyond the scope of this book, although we have stated the result as Theorem 6.88.

The final structural stage goes roughly as follows. The last structure that we consider excluding is an arbitrary graph, for which the complete graphs are generic. The resulting structure that is imposed on G is being “surfacelike.” This is codified in a fairly complicated way that we will refer to, suggestively but mythically, as *surfacewidth*, for which there is ultimately a mythical theorem analogous to Theorem 6.88.

The structure theory can be summarised in the following table.

Excluded Substructure	Resulting Structure	Structural Parameter
path	pointlike	pointwidth
tree	pathlike	pathwidth
planar	treelike	treewidth
arbitrary	surfacelike	surfacewidth

Note the beautiful structural diagonals! This theory is itself extremely deep, but it is only part of the story of the proof of the GMT. The other main part of the story is how the stages of **WQO** results exploit this structure theory.

If we consider the family of all graphs that are paths and unions of paths, then obviously this is well-quasi-ordered by minors. We can think of graphs of bounded pathwidth as represented by colored or labeled paths, with the number of colors being a function of the width bound. (We have done this, essentially, in Theorem 7.58.) With some work, we can lift the obvious **WQO** of paths to an appropriate **WQO** of labeled paths, and eventually obtain that graphs of bounded pathwidth, for each fixed width bound, are a **WQO**. We might consider doing this by inducting on the number of colors.

From this, we could argue that trees are a **WQO** in the minor ordering as follows. Suppose $(T_1, T_2, \dots)$ is an infinite bad sequence of trees. Since all of the trees in the sequence after T_1 exclude T_1 as a minor, our second structure theorem from the table tells us that there is a bound B on the pathwidth of the trees in the infinite bad subsequence $(T_2, T_3, \dots)$. But now, we have reached a contradiction, since graphs (including trees) of bounded pathwidth are a **WQO**. (The fact that we had a different way of proving trees to be a **WQO** does not harm this illustration.)

At the next stage, similarly, we “generalize” from trees to labeled trees, using the labels to represent the local structure of a graph of bounded treewidth. (We have done this in the section on bounded treewidth.) With some work, possibly by induction on the number of labels, we get that graphs of bounded treewidth are a **WQO**.

From this, we can conclude that planar graphs are a **WQO** in the minor order as follows. Suppose $(G_1, G_2, \dots)$ is an infinite bad sequence of planar graphs. Since

all of the graphs in the sequence after G_1 do not have G_1 as a minor, we know (by Theorem 6.88) that the graphs in the infinite bad subsequence $(G_2, G_3, \dots)$ all have treewidth bounded by some constant B , which contradicts that graphs of bounded treewidth are a **WQO**.

Now that we have planar graphs as a **WQO**, we extend this to graphs of bounded surfacewidth (mythical) by further labeling schemes that represent local structure. The details get more and more complicated, and the later stages have substages, but the general pattern of argument goes roughly like that.

To give an example of the type of structural result obtained by Robertson and Seymour, we need some definitions. A graph J is called a *clique-join* of graphs G and H if $G \cap H$ is a clique and J is a subgraph of $G \cup H$. (Think here of $\oplus$.) Now, let P_n be a path with vertices $t_1, \dots, t_n$ and C_n the circuit on the same vertices. Given an integer r , an *r -ring with perimeter $t_1, \dots, t_n$* is a graph R on the vertices $t_1, \dots, t_n$ such that there is a collection of bags $X = \{X_{t_i} : i = 1, \dots, n\}$ for which (P_n, X) is a path decomposition of R or width $r - 1$, and for each i , $t_i \in X_{t_i}$. Assume that we are given a hypergraph where each edge is incident with at least two edges. If a hyperedge is incident with exactly two vertices, we call it an *edge*, and otherwise we will say it is a *hyperedge*. We regard a hypergraph as embedded on a surface Σ if the following all hold. Each vertex is a point, each edge is a simple closed curve, each hyperedge is a closed disk with its incident vertices on its boundary and no other intersections occur.

Given a surface Σ and integer r , a (Σ, r) -*outgrowth* is a pair (G, H) , where G is a graph, H is a hypergraph embedded on Σ , such that for each hyperedge h of H , if $(h_1, \dots, h_n)$ is a cyclic ordering of the vertices of h induced by the embedding in Σ , then there is an r -ring R_h with perimeter $h_1, \dots, h_n$ such that G is the graph induced by the edges of

$$H \cup \bigcup_{h \text{ a hyperedge of } H} R_h.$$

Note that $V(G) = V(H)$. If Θ is a set of surfaces, then a graph G is a (Θ, r) -*outgrowth* if there is a $\Sigma \in \Theta$ and an embedded hypergraph such that (G, H) is a (Σ, r) -outgrowth. [We ask that $(\emptyset, \emptyset)$, the pair of empty graphs, are a (Σ, r) -outgrowth for any (Σ, r) .] Finally, given an integer k , we say that M is a $\leq k$ -*vertex extension* of G , if there is a set S of at most k vertices in M such that $G = M - S$.

The following is an example of one of the main technical results used to prove the Graph Minor Theorem as we moved to surfaces.

Theorem 7.63 (Robertson and Seymour [397]) *Let G be a graph and $\Theta(G)$ the set of surfaces into which G cannot be embedded. Then, there are numbers $r(G)$ and $w(G)$ such that every graph with no G -minor can be constructed by clique joins, starting from $\leq w(G)$ vertex extensions of $(\Theta(G), r(G))$ -outgrowths.*

Note that in the case of G a planar graph, then $\Theta(G) = \emptyset$, and the theorem says that graphs excluding G can be obtained from graphs with at most $w(G)$ -vertices by clique joins. That is, Theorem 7.63 can be applied to give Theorem 6.88, namely that graphs excluding a planar graph must have bounded treewidth. We remark that

Theorem 7.63 has already found other applications such as in Ding, Oporowski, Sanders, and Vertigan [157] where it was used to verify a conjecture of Thomas by proving the following theorem.

Theorem 7.64 (Ding et. al. [157]) *For every graph G , there is an integer k such that every graph excluding G has a two coloring of either its vertices or its edges where each color induces a graph of treewidth at most k .*

Historical Notes

The Robertson-Seymour Theorems are proved in an ever-increasing sequence of papers called collectively *Graph Minors*. Some can be found in the bibliography. The proofs are very long and difficult, and in some sense this is provably necessary since Friedman, Robertson, and Seymour [235] have shown that the Graph Minor Theorem is not provable in a logical system that encompasses most of classical mathematics (Π_1^1 comprehension, see Simpson [425]).

Exercises 7.6

7.6.1. (M. Dinneen [158]) Let $\gamma(G) \leq p(|G|)$ map graphs to integers, where p is some bounding polynomial. Assume $\mathcal{F}_k = \{G \mid \gamma(G) \leq k\}$ is an minor-order lower ideal. Also let $p_\gamma(G, k)$ denote the corresponding graph problem that determines if $\gamma(G) \leq k$ where G and k are the input.

Show that if $p_\gamma(G, k)$ is *NP*-complete then $f_k = |\mathcal{O}(\mathcal{F}_k)|$ must be super-polynomial else the polynomial-time hierarchy $\mathcal{PH}$ collapses to Σ_3^P .

(Hints. Use the following two results of Yap [469]:

- The following are equivalent for $i > 0$:
 - (i) $\Pi_i^P \subseteq \Sigma_i^P/\text{poly}$ or $\Sigma_i^P \subseteq \Pi_i^P/\text{poly}$
 - (ii) $\Sigma_i^P/\text{poly} = \Pi_i^P/\text{poly}$
 - (iii) $\mathcal{PH}/\text{poly} = \Sigma_i^P/\text{poly}$ or $\mathcal{PH}/\text{poly} = \Pi_i^P/\text{poly}$
- For $i > 0$, $\Sigma_i^P/\text{poly} = \Pi_i^P/\text{poly} \Rightarrow \Sigma_{i+2}^P = \Pi_{i+2}^P$.)

7.7 Immersions and the Nash-Williams Conjecture

In the same series of articles where they proved the Graph Minor Theorem, Robertson and Seymour verified another famous **WQO** conjecture made by Nash-Williams [360].

Definition 7.65 (Immersion Ordering, Lifting) Let G and H be graphs. We say that $G \leq_{\text{immersion}} H$ if we can obtain G from H by applications of the operations *deletion* and *edge lifting*.

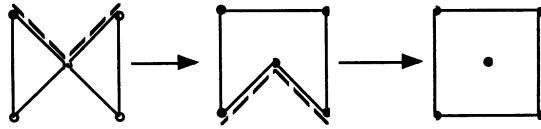


FIGURE 7.9. The Immersion Ordering

Here, if G is a graph with a vertex v having edges wv and qv , $w \neq q$, then *lifting* through $\{w, v, q\}$ is the result of removing the edges wv and qv and adding an edge wq . The reader is referred to Figure 7.9.

Theorem 7.66 (Graph Immersion Theorem, Robertson and Seymour [397]) *Finite graphs are well-quasi-ordered by $\leq_{\text{immersion}}$.*

The algorithmic applicability of the Graph Immersion Theorem relies on the Obstruction Principle, together with the following result.

Theorem 7.67 (Fellows and Langston [217]) *For every fixed graph H , the parameterized problem which takes as input a graph G and asks if $H \leq_{\text{immersion}} G$ is solvable in polynomial time.*

Proof. Letting k denote the number of edges in H , we replace $G = (V, E)$ with $G' = (V', E')$, where $|V'| = k|V| + |E|$ and $|E'| = 2k|E|$. Each vertex in V is replaced in G' with k vertices. Each edge e in E is replaced in G' with a vertex and $2k$ edges connecting this vertex to all of the vertices that replace e 's endpoints. We can now apply the disjoint-connecting paths algorithm of [396], since it follows that $H \leq_i G$ if and only if there exists an injection from the vertices of H to the vertices of G' such that each vertex of H is mapped to some vertex in G' that replaces a distinct vertex from G and such that G' contains a set of k vertex-disjoint paths, each one connecting the images of the endpoints of a distinct edge in H .

We do not know if the problem below is *FPT*, but again we conjecture that it is.

IMMERSION ORDER TEST

Instance: A graph $G = (V, E)$ and a graph $H = (V', E')$.
Parameter: A graph H .
Question: Is $H \leq_{\text{immersion}} G$?

Exercises 7.7

- 7.7.1.** Give an upper bound on the smallest $m \times m$ grid in which the complete graph K_n is immersed.
- 7.7.2.** Prove that acyclic graphs are well-quasi-ordered by immersions.
- 7.7.3.** Prove that **IMMERSION ORDER TEST** is linear fixed-parameter tractable for planar graphs.

7.7.4. A *plane graph* is a planar graph G together with an embedding f of G in the plane. Define a natural notion of the immersion of one plane graph in another and investigate whether plane graphs are well-quasi-ordered by plane immersions.

7.7.5. Use the fact that the immersion order is a WQO to prove that each slice of the following problem can be solved in polynomial time. (Due to Govindan, Langston and Plaut [249].)

MINIMUM DEGREE GRAPH PARTITION

- Instance:* A graph $G = (V, E)$ and positive integers k and d .
Parameter: k, d .
Question: Can V be partitioned into disjoint subsets $V_1, \dots, V_m$ so that, for $1 \leq i \leq m$, $|V_i| \leq k$ and at most d edges have exactly one endpoint in V_i ?

7.8 Applications of the Obstruction Principle and WQO's

It is important to point out that the minor and immersion WQO's do not exhaust the interesting possibilities for general methods based on the Obstruction Principle. The minor and immersion orders are defined in terms of sequences of local operations on graphs by which one graph can be derived from another. Given the set of allowed operations, the order is defined by $G \geq H$ if H can be derived from G by a sequence of these operations. The nature of this definition immediately yields transitivity. There are obvious opportunities for generalizing by considering various sets of local operations on various kinds of combinatorial objects. Many of these avenues have not been explored, and some present difficult mathematical challenges.

The following are some examples of simple local operations on graphs:

- (E)Delete an edge.
- (V)Delete a vertex.
- (C)Contract an edge.
- (T)Contract an edge incident on a loopless vertex of degree 2. (Also called removing a *subdivision*.)
- (L)Lift an edge.
- (F)Fracture a vertex. Replace v with two new vertices v_1 and v_2 and partition the edges incident on v into two classes, making these incident, respectively, on v_1 and v_2 .
- (I)Identify two vertices.
- (S)Simplify. Delete an edge of which there is more than one copy, or remove a loop.

The minor order is thus generated by the set of operations $\{E, V, C\}$ and the immersion order is generated by the set of operations $\{E, V, L\}$. The topological order is generated by $\{E, V, T\}$ and the *induced minor* order is generated by the

set of operations $\{V, C\}$. As a matter of historical interest, the minor order was quite obscure *until* the work by Robertson and Seymour.

Most graph properties are preserved by some set of local operations that can be used to define a quasi-order for which the property defines an ideal, thus potentially allowing an algorithmic approach based on the Obstruction Principle. We illustrate this with two brief examples.

Example 7.68 (Cutwidth) We recall from Section 6.4 the cutwidth problem.

CUTWIDTH

Instance: A graph $G = (V, E)$.

Parameter: k .

Question: Can the vertices of G be linearly ordered by a 1:1 function $f : V \rightarrow \{1, \dots, |V|\}$ so that $\forall i \in \{1, \dots, |V| - 1\}$:
 $|\{uv : f(u) \leq i \text{ and } f(v) > i\}| \leq k$?

For each fixed k , the family of graphs that are yes instances is closed under the set of operations $\{E, V, T, L, F\}$. Since this includes the set of operations that generate the immersion order, we immediately know that the problem can be solved in polynomial time for every fixed k . It is not hard to show that graphs of cutwidth at most k have pathwidth bounded by a function of k . By Bodlaender's theorem and the fact that immersions are expressible in monadic second-order logic (and using Courcelle's theorem), we can conclude that **CUTWIDTH** is linear time *FPT*. (See Section 6.4.)

In the immersion order there are 15 obstructions to cutwidth $k = 2$, whereas in the “improved” order native to the problem there are 5 (Makedon and Sudborough [344]).

Example 7.69 (Interval Graphs) Interval graphs are defined in Section 3.2.2. Like many families of graphs that are defined by the existence of representations based on intersections (in the case of interval graphs, these are intersections of line segments; in the case of chordal graphs, these are intersections of subtrees of a tree), the family of interval graphs is closed under the set of operations $\{V, C\}$ and is therefore a lower ideal in the induced minor order. Even though induced minors are not a **WQO**, interval graphs have a finite obstruction set in this order consisting of the five graphs shown in Figure 7.10.

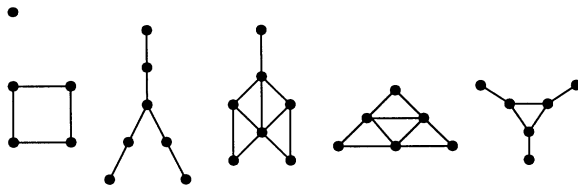


FIGURE 7.10. Obstructions for Interval Graphs

We can assume that many of the quasi-orders that are generated by natural sets of closure operations for interesting graph properties are not WQO's in general, but there are two ways in which they may still be useful from the point of view of the Obstruction Principle:

- (1) Particular lower ideals might be characterized by finite obstruction sets, as in the example of interval graphs in the induced minor order.
- (2) The order of interest may be a WQO on important restricted classes of graphs. For example, the topological order is a WQO for trees and for graphs of maximum degree 3, and the induced minor order is a WQO for series-parallel graphs. The important question to ask about an interesting order is: *Where* is this a WQO?

Many of the FPT applications of the Graph Minor Theorem and other WQO results can be discussed in terms of general parameterized operators that take as an argument one or more lower ideals and from these describe a new lower ideal. Rather than give a formal definition of an operator on ideals, it is best to illustrate with some examples that cover a number of familiar computational problems.

Example 7.70 (Within k Vertices of $\mathcal{F}$) Let $\mathcal{F}$ be an ideal in the minor order and k a positive integer. Say that a graph $G = (V, E)$ is *within k vertices of $\mathcal{F}$* if there is a set $V' \subseteq V$ of at most k vertices such that $G - V' \in \mathcal{F}$. It can be easily verified that for every k , the family $\mathcal{W}_k(\mathcal{F})$ of graphs that are within k vertices of $\mathcal{F}$ is a minor order ideal.

By taking the argument ideal $\mathcal{F}$ to be the family of graphs with no edges, we can conclude that **VERTEX COVER** is FPT. By taking $\mathcal{F}$ to be the family of acyclic graphs (i.e., forests), we can conclude that **FEEDBACK VERTEX SET** is FPT. Graphs that are within k vertices of planar are also of interest; for $k = 1$, these are termed *apex graphs*. The obstruction set for apex graphs is currently unknown.

Example 7.71 (k -Star Augmentation) Let $\mathcal{F}$ be an ideal in the minor order and k a positive integer. Say that a graph $G = (V, E)$ is *k -star augmentable to $\mathcal{F}$* if there is a partition of the vertices of G into k classes $V_1, \dots, V_k$ so that the graph G' obtained from G by adding k new vertices $v_1, \dots, v_k$ with (for $i = 1, \dots, k$) v_i connected to each vertex $u \in V_i$ belongs to $\mathcal{F}$. It is not hard to show that for each positive integer k , the family $\mathcal{S}_k(\mathcal{F})$ of graphs that are k -star augmentable to $\mathcal{F}$ is a minor ideal.

By taking $\mathcal{F}$ to be the family of planar graphs, we can conclude that **PLANAR FACE COVER** below is FPT.

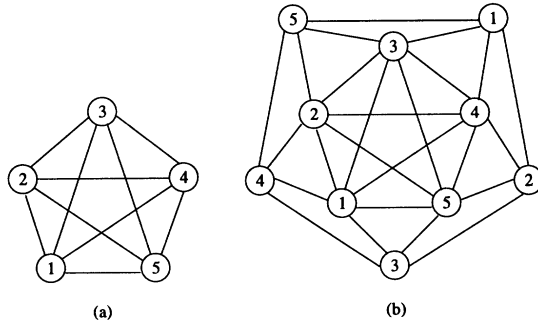
PLANAR FACE COVER

Instance: A graph $G = (V, E)$.

Parameter: A positive integer k .

Question: Can G be embedded in the plane so that there are k faces which cover all vertices?

This problem has been studied by several authors and has some applications in VLSI layout. For $k = 1$, we get the family of outerplanar graphs.


 FIGURE 7.11. A 2-Fold Cover of K_5

The notion of a *covering space* of a topological space is fundamental in some areas of algebra, topology, and complex analysis. Since graphs can be considered as topological 1-complexes, the notion is available in graph theory, and covering spaces of graphs have been considered in the context of parallel and distributed computing by, for instance, Angluin [26], Abello, Fellows and Stillwell, [1], and Fellows [203].

To illustrate the notion of a covering space, consider the problem of the Flatlanders attempting to build a 5-processor parallel machine with the complete graph connection topology. By Theorem 7.7, this is impossible. However, they can build the machine shown in the Figure 7.11.

With this, the Flatlanders can emulate a K_5 machine by running two identical copies of each of the five processes. One can see by examining the figure that each copy of each of the five processes has a copy of each of the other processes in its immediate neighborhood. Another way to say this is that the nonplanar graph K_5 has a 2-fold planar cover. A beautiful unsolved problem is the conjecture of Negami that a graph has a finite planar cover if and only if it embeds in the projective plane. Generalizing, we have the following parameterized operator on minor ideals.

Example 7.72 (k -Fold $\mathcal{F}$ -Coverability) Let $\mathcal{F}$ be a minor ideal and k a positive integer. It is not hard to show that for every k , the family $\mathcal{C}_k(\mathcal{F})$ of graphs that have a k -fold cover in $\mathcal{F}$ is a minor ideal.

WQO's and the Obstruction Principle can be used directly, as in the above examples, to prove *FPT* and polynomial-time complexity bounds, and they can also be employed indirectly. We next describe two general methods for indirect applications.

Definition 7.73 (RS Poset) Let $(S, \leq)$ denote a partially ordered set. We will say that $(S, \leq)$ is an *RS poset* if:

- (1) $(S, \leq)$ is a WQO.
- (2) The order testing problem is *FPT*.

Theorem 7.74 (Method A, Fellows and Langston [216]) If $(S, \leq)$ is an RS poset and T is a subset of S such that

(a) $\mathcal{F} \subseteq T$, and

(b) $x \in \mathcal{F}$, $y \leq x$, and $y \in T$ together imply $y \in \mathcal{F}$,

then there is a polynomial-time decision algorithm to determine for input $x \in T$ whether $x \in \mathcal{F}$.

Proof. Let M denote the (finite) set of minimal elements in $T - \mathcal{F}$. Hypothesis (b) implies that if x is in T , then $x \in \mathcal{F}$ if and only if it is not the case that $z \leq x$ for some $z \in M$. Since $(S, \leq)$ is RS and M is finite, this can be determined in polynomial-time.

Theorem 7.75 (Method B, Fellows and Langston [216]) *If $(S, \leq)$ is an RS poset and if there is a polynomial time computable map $t : D \rightarrow S$ such that for $\mathcal{F} \subseteq D$,*

(a) $t(\mathcal{F})$ is closed under $\leq$, and

(b) either $t^{-1}(x) \cap \mathcal{F} = \emptyset$ or $t^{-1}(x) \subseteq \mathcal{F}$ for each $x \in t(D)$,

then there is a polynomial-time decision algorithm to determine for input $z \in D$ whether $z \in \mathcal{F}$.

Proof. First, compute $t(z)$. There is a finite set M of minimal elements in the complement of the image of $\mathcal{F}$ because $(S, \leq)$ is a WQO. By hypothesis (a), $t(z)$ is in the image of $\mathcal{F}$ if and only if it is not the case that $y \leq t(z)$ for some $y \in M$. This can be determined in polynomial time by performing an order test for each element of M . By hypothesis (b), $z \in \mathcal{F}$ if and only if $t(z)$ belong to the image of $\mathcal{F}$.

We give two examples of how these indirect methods can be applied.

Example 7.76 (Compact Routing Schemes) A fundamental problem in parallel processing networks is that messages between processors must somehow be routed through the network. The classical solution is to provide each processor v with a routing table T_v that specifies, for each possible destination x , the neighbor $u \in N[v]$ to which a message having the destination x should be passed along. In very large networks, this becomes impractical and inefficient. *Interval routing* is one method that has been proposed to deal with this situation (see, for instance, Santoro and Khatib [407], Frederickson and Janardan [232] and Bodlaender, van Leeuwen, Tan and Thilikos [87]) and that has been used in hardware such as the INMOS T9000 Transputer. In a k -interval routing scheme, each neighbor of a vertex is assigned k intervals in a circular ordering of the processors. (The details are not important for this illustration of Method A, and can be found in [87].)

The question naturally arises of what graphs admit a k -interval routing scheme that is *optimal*, i.e., as good as any classical scheme. It should be obvious (regardless of the details of interval routing) that this family of graphs cannot be closed in the minor order *in general*, since routing schemes are not defined for disconnected graphs!

It is shown in [87] that Method A can be applied to this problem when $\mathcal{F}$ is taken to be the family of connected graphs; that is, if G is a connected graph that admits an optimal k -interval routing scheme and if H is a *connected* minor of G , then H

admits an optimal interval routing scheme. By Method A, this means that that the problem of recognizing graphs that admit optimal k -interval routing schemes is FPT for parameter k .

Example 7.77 (Gate Matrix Layout) This is a problem that has practical applications in VLSI layout, for a variety of layout styles, and that has appeared in the literature in a number of combinatorially equivalent forms, being known also as “PLAs with multiple folding” and “Weinberger arrays.” The GATE MATRIX LAYOUT problem was first defined in the form we consider by Lopez and Law [334]. The problem is defined as follows.

GATE MATRIX LAYOUT¹¹

Instance: A boolean matrix M and a positive integer k .

Parameter: k .

Question: Is there a permutation of the columns of M so that if in each row we change to * every 0 lying between the row's leftmost and rightmost 1, then no column contains more than k 1's and *'s?

To apply technique B, we map a matrix M to the graph $f(M)$ for which the vertex set consists of the rows of the matrix, with two vertices joined by an edge if there is some column in which both corresponding rows have entry 1. The map is many-to-1, so that condition (b) is clearly necessary. The proof that this mapping works was first given in [223], and can safely be left as an exercise (Exercise 7).

In addition to providing an important paradigm for the exact solution of problems, the Obstruction Principle also offers the possibility of new kinds of heuristic and approximation algorithms. For example, although Kuratowski's theorem gives us two obstructions that characterize the property of being planar, almost all of the information is provided by just one of these. In fact, it can be shown that any nonplanar graph of minimum degree 3 is either *isomorphic* to K_5 or has a $K_{3,3}$ minor. Another example of this phenomena is the GATE MATRIX LAYOUT problem. For a layout cost of $k = 2$ (pathwidth 1), there are two obstructions, for $k = 3$ (pathwidth 2), there are 110 obstructions, and for $k = 4$, there are at least 6×10^7 obstructions in the minor order. For VLSI applications, it seems that the single obstruction K_4 provides almost all of the useful information for $k = 3$. It is conceivable that finite approximate obstruction sets might be a useful algorithmic approach even for some lower ideals of non-WQO's.

Exercises 7.8

- 7.8.1.** Verify that for each fixed k , the family of graphs having cutwidth at most k is closed under the operation of fracturing a vertex.
- 7.8.2.** Prove that the minor ordering and the topological ordering coincide on the family of graphs of maximum degree 3.

¹¹We remark that the problem is equivalent to GRAPH PATHWIDTH.

- 7.8.3.** The complete graph K_5 has an interesting “Ramsey theoretic” property: No matter how the edges are labeled with positive integers, there is a cycle whose label sum is $0 \bmod 3$. Generalizing this, consider the following problem:

UNIT CYCLE AVOIDANCE

Instance: A graph G and an abelian group A .
Parameter: A
Question: Can the edges of G be labeled by elements of A so that no cycle in G has label sum equal to the identity element of A ?

Prove that **UNIT CYCLE AVOIDANCE** is nonuniformly *FPT* if A is restricted to be a group of the form $(\mathbb{Z}_2)^m$, or if G is restricted to graphs of maximum degree 3. Is this problem in *NP*? (See Arkin[36] and Barefoot, Clark, Douthett, Entringer [49]).

- 7.8.4.** (M.R. Fellows, J. Kratochvíl, M. Middendorf and F. Pfeiffer) Show that the following problem is linear *FPT*.

INDUCED MINOR TESTING FOR PLANAR GRAPHS

Instance: Planar graphs G and H .
Parameter: H .
Question: Is $G \geq H$ in the induced minor order?

(Hint: First, use Theorem 6.88 to show that for each fixed planar graph H there is a constant C_H such that if the answer is “yes,” then the treewidth of G is at most C_H .)

- 7.8.5.** (Dejter and Fellows [153]) Show that the following problems are linear *FPT*.

PLANAR DIAMETER IMPROVEMENT

Instance: A planar graph G and a positive integer k .
Parameter: k .
Question: Is there a planar graph G' with $G \subseteq G'$, such that the diameter of G' is at most k ? (The *diameter* of a graph is the maximum distance between a pair of vertices in the graph.)

PLANAR DOMINATION IMPROVEMENT NUMBER

Instance: A planar graph G and a positive integer k .
Parameter: k .
Question: Is there a planar graph G' with $G \subseteq G'$, such that G' has a k -element dominating set?

Generalizing from these two examples, define an operator that takes as arguments a minor order ideal $\mathcal{I}$ and an ideal $\mathcal{J}$ in the order generated by the set of local operations $\{C, S\}$ and produces a minor ideal.

- 7.8.6.** Robertson and Seymour have proved even stronger results concerning graph minors than we have discussed. They have proved that finitely edge-colored, directed graphs are an RS-poset in the minor order extended to these “annotated” graphs. (The local operations V , E , and C are obviously still well defined.) Use this result and Method A to prove that k -boundaried graphs are an RS poset under k -boundaried minors, where the minor order is extended to boundaried graphs by requiring that the boundary is held fixed while applying the operations that define the minor order.
- 7.8.7.** Verify that Method B applies as indicated for the **GATE MATRIX LAYOUT** problem. Prove that the image under the map f of the “yes” instances for k -GML consists of precisely the graphs of pathwidth at most $k - 1$.

7.9 Effectivizations of Obstruction-Based Methods

The techniques of **WQO**’s provided by the deep results of Robertson and Seymour have posed a number of profound philosophical challenges to some of the prevailing habits of algorithm design and complexity analysis. For example, consider the problem defined as follows.

GENUS k -COVER

- Instance:* A graph G .
Parameter: A positive integer k .
Question: Is there a finite graph H of genus at most k that is a cover of G ? [H is a *cover* of G if there is a projection map $p : V(H) \rightarrow V(G)$ such that (1) p is onto and (2) if $p(u) = x$ and $xy \in E(G)$, then there is a unique vertex $v \in V(H)$ with $uv \in E(H)$ and $p(v) = y$.]

This is nonuniformly fixed-parameter tractable by the Graph Minor Theorem, and this is the only way we currently know how to prove that the problem is decidable. It represents well the situation we are in when no other information is available.

We are confronted by two different and basic kinds of non-constructivity:

- (1) We know that for each k , there is an $O(n^3)$ algorithm for the k -th slice of the problem based on the Obstruction Principle—but we have no idea what this algorithm is, since we only know that a finite obstruction set for the k -th slice exists, and have no means of identifying it. Let us call this the problem of *algorithm nonconstructivity*.
- (2) Second, even if we knew the obstruction set, we would have a very peculiar algorithm. It would correctly answer the yes/no question, “Does a finite cover H of genus at most k exist?” but would give us no clue

about how to efficiently find such an H nor guarantee that this is even possible. Let us call this the problem of *solution nonconstructivity*.

In this section, we look at two general methods of dealing with these constructivity issues. These are sufficient to deal with almost all of the algorithmic applications of WQO's that have been considered. The first and most general method is based on *self-reduction* algorithms. The second method is a systematic means of computing obstruction sets in favorable circumstances.

7.9.1 Effectivization by Self-Reduction

The natural situation of most computations is most fully modeled in terms of relations. Let $\Pi \subseteq \Sigma^* \times \Sigma^*$ be a binary relation. Typically, we are thinking of something like the following.

Example 7.78 (The HAMILTON CIRCUIT Relation) The relation Π_{HC} consists of all the pairs (G, σ) , where G is a graph and σ is an ordering of the vertices of G that constitutes a Hamilton circuit in G .

Definition 7.79 (Polynomially Balanced) A relation Π is *polynomially balanced* if there is a polynomial q such that $\forall (x, y) \in \Pi: |x| \leq q(|y|)$ and $|y| \leq q(|x|)$.

Given such a relation Π , we can describe three basic computational problems:

- *The Checking Problem* is that of determining, for input (x, y) , whether $(x, y) \in \Pi$.
- *The Domain Decision Problem* is that of determining, for input x , whether there exists a y such that $(x, y) \in \Pi$.
- *The Search Problem* is the problem of computing a search function for Π , where such a function

$$f : \Sigma^* \rightarrow \Sigma^* \cup \{F\}$$

satisfies:

- (1) $f(x) = y$ implies $(x, y) \in \Pi$, and
- (2) $f(x) = F$ implies that there is no y with $(x, y) \in \Pi$.

A standard characterizations of *NP* is that it consists of those languages $L \subseteq \Sigma^*$ that are the domain of a polynomially balanced relation for which the checking problem can be solved in polynomial time.

In many ways, it is more natural to discuss the objectives of computation in terms of search problems. The original article by Levin [330] that introduced the notion of *NP-hardness* independently of Cook was formulated in terms of relations and search problems. The relational point of view is capable of supporting interesting systematic development, especially since most known reductions between computational problems show how to translate between solutions as much as between decision instances. (See, for instance, Arora [37] or Abrahamson, Fellows, Langston and Moret [8].)

Given that our natural concern in computer applications is almost always with search problems, it should be somewhat surprising that the theory of computational complexity has been developed almost entirely in terms of decision problems. The justification that is usually given is that there is no substantial loss of generality in the restriction of the basic theory to decision problems, since there is almost always an efficient reduction of a search problem to its naturally associated decision problem. **WQO**'s may be a source of new reasons to question this assumption.

In any case, the reduction of search to decision is generally termed *self-reducibility*, and this is the sense in which we use the word here ¹².

The following is an example of a “typical” polynomial-time self-reduction of a Π -search problem to the domain decision problem for Π .

Example 7.80 (A Self-Reduction for **HAMILTON CIRCUIT**) Using an algorithm for the domain decision from for Π_{HC} , we can compute a search function for Π_{HC} as follows. First, use the domain decision “oracle” to determine if the output should be F in case G does not have a Hamilton circuit. If the oracle answers “yes”— G is Hamiltonian—then remove any edge uv from G to obtain G' and use the domain decision oracle to determine if G' is Hamiltonian. There are two possible cases:

- (1) G' is Hamiltonian. In this case, uv is “unnecessary” and we can consider it permanently removed.
- (2) G' is not Hamiltonian. This implies that every Hamilton circuit in G traverses uv and we can mentally mark uv as “never to be removed.”

We use the domain decision oracle in this way to progressively simplify G , each round of consultation with the oracle resulting either in the removal of an edge or the identification of an essential edge. After $|E| - |V|$ consultations, the only edges remaining are those of a Hamilton circuit.

Finding an efficient self-reduction algorithm is the obvious way to deal with the problem of solution nonconstructivity that arises in applications of **WQO**'s. As an example of how parameters may interact with self-reduction, a self-reduction algorithm for **CUTWIDTH** that uses $O(n)$ calls to a decision oracle for $2k$ -**CUTWIDTH** in order to solve the search problem for k -**CUTWIDTH** is described in Fellows and Langston [222]. The important point is that to efficiently solve the search problem for a given parameter value, we might be able to make effective use of a decision algorithm for a different value of the parameter.

Surprisingly, self-reduction *also* gives us a general way to deal with the problem of algorithm nonconstructivity. The technique that we next describe will allow us to write down, for almost all known **WQO** applications, a correct algorithm to compute a search function in polynomial time—but without ever identifying the complete obstruction set, and without knowing the polynomial that bounds the running time of the algorithm!

¹²We remark that there are various formalizations of the basic idea that have been made purely within the framework of decision problem complexity theory that are not equivalent, for reasons of apples and oranges, to a relational formulation of the issue (Schnor [410]).

Definition 7.81 (Uniformly Enumerable) A quasi-order $(R, \leq)$ is *uniformly enumerable* if there is a recursive enumeration $(r_0, r_1, r_2, \dots)$ of R with the property that $r_i \leq r_j$ implies $i \leq j$.

In the minor and immersion orders, for example, a natural uniform enumeration is to generate all finite graphs based on a monotonic nondecreasing sequence of number of vertices, with graphs having the same number of vertices generated based on a monotonic nondecreasing sequence of number of edges, with ties (graphs with the same number of vertices and the same number of edges) broken arbitrarily.

Definition 7.82 (Uniform Self-Reduction) Under a uniform enumeration of the elements of $\text{domain}(\Pi)$, we say that a self-reduction algorithm for Π is *uniform* if, on input r_j , the oracle for $\text{domain}(\Pi)$ is consulted concerning r_i only for $i \leq j$.

Definition 7.83 (Honest) An oracle algorithm is *honest* if, on inputs of size n , its oracle is consulted concerning only instances of size $O(n)$.

Definition 7.84 (Robust) An oracle algorithm A with overhead bounded by $T(n)$ is *robust* [with respect to $T(n)$] if A is guaranteed to halt within $T(n)$ steps for any oracle language.

Theorem 7.85 (Fellows and Langston [218]) *Let $F = \text{domain}(\Pi)$ be an ideal in a uniformly enumerable WQO, and suppose the following are known:*

- (1) *an algorithm that solves the checking problem for Π in $O(T_1(n))$ time,*
- (2) *order tests that require $O(T_2(n))$ time,*
- (3) *a uniform self-reduction algorithm (its time bound is immaterial),*
- (4) *an honest robust self-reduction algorithm that requires $O(T_3(n))$ overhead.*

Then an algorithm requiring $O(\max\{T_1(n), T_2(n) \cdot T_3(n)\})$ time is known that solves the search problem for Π .

Proof. Let I denote an arbitrary input instance and let K denote the known elements of the (finite) obstruction set for F . (Initially, K can be empty.) We treat K as if it were the correct obstruction set, that is, as if we *did* know a correct decision algorithm. Since the elements of K will always be obstructions, if we find that $H \leq I$ for some $H \in K$, then our algorithm reports “no” and halts. Otherwise, after checking each element of K to confirm that it is not less than or equal to I , we attempt to self-reduce and check the solution so obtained. If the solution is correct, then our algorithm reports “yes,” outputs the solution, and halts.

In general, however, it may turn out that our check of the solution reveals that we have self-reduced to a nonsolution. This can only mean that there is at least one obstruction $H \notin K$. In this event, we proceed by generating the elements of R in uniform order until we find a new obstruction (an element that properly contains no other obstruction but that cannot be uniformly self-reduced to a solution). When

such an obstruction is encountered, we need only augment K with it and start over on I .

Let C_1 denote the cardinality of the correct obstruction set and let C_2 denote the largest number of vertices in any of its elements. Then, for some suitably chosen function f , the total time spent by this algorithm is bounded by $O(C_1(T_1(n) + T_2(n) \cdot T_3(n)) + f(C_2))$.

In the statement of Theorem 7.85, it is possible to replace the hypothesis that a uniform self-reduction is known with the alternate hypothesis that a domain decision algorithm is known.

Although, at first blush, any attempt to implement the algorithm used in the proof of Theorem 7.85 appears to be completely out of the question, closer inspection reveals that it may in fact provide the basis for novel approaches in algorithm design. Note that this scheme can be viewed as a “learning algorithm” that gradually accumulates a useful subset of the obstruction set, and invokes its exhaustive “learning component” only when forced to do so by input it cannot handle with this subset. Furthermore, some obstructions are already known or are easy to identify for most problems, so that we need not start with the empty set. We give an example of an application.

Example 7.86 (PLANAR DIAMETER IMPROVEMENT) Given a planar graph G and a positive integer parameter k , the question is whether there is a planar supergraph G' of G of diameter at most k (Exercise 4). It is not hard to see that for each k there is a bound on the treewidth of the yes instances, so this problem is linear time *FPT*, because for each fixed k the yes instances are an ideal in the minor order. For this problem (and others like it), there is no obvious way to express the property in MS_2 logic, since the natural definition quantifies over sets of edges that are *not* in the graph. In the absence of any further information, we know only that a linear time *FPT* algorithm *exists* for this problem and have no idea what it is. The problem is easy to self-reduce, however, just by progressively adding edges to the graph. By Theorem 7.85, we can therefore write down an *FPT* algorithm that runs in time $f(k)n^3$ for some unknown function $f(k)$.

7.9.2 Effectivization by Obstruction Set Computation

The second general method of constructively producing the *FPT* and polynomial-time algorithms that are proven to exist by applications of **WQO** theorems is based on computing the relevant obstruction sets.

We have seen that knowing only a decision algorithm for a minor order lower ideal $\mathcal{F}$ is not enough to allow us to compute the obstruction set for $\mathcal{F}$. The next theorem is in some sense complementary in that it establishes not too difficult conditions under which we *can* compute the obstruction set for $\mathcal{F}$.

Theorem 7.87 (Fellows and Langston [219]) *Suppose that $\mathcal{F}$ is an ideal in the minor order of finite graphs and that we have the following three pieces of information about $\mathcal{F}$:*

- (1) *An algorithm to decide membership in $\mathcal{F}$ (of any time complexity).*
- (2) *A bound B on the maximum treewidth of the obstructions for $\mathcal{F}$.*
- (3) *For $t = 1, \dots, B + 1$, a decision algorithm for a finite index right congruence $\sim$ on t -boundaried graphs that refines the canonical congruence for $\mathcal{F}$ on the small universe.*

Then, we can effectively compute the obstruction set $\mathcal{O}$ for $\mathcal{F}$.

Proof. The algorithm is outlined as follows. For $t = 1, \dots, B + 1$, we generate in a systematic way, using the standard set of parsing operators (see Theorem 5.72), the t -boundaried graphs of $\mathcal{U}_t^{\text{small}}$ until a certain *stop signal* is detected. At this point, for a given t , we will have generated a finite set of graphs $\mathcal{G}_t$. Of particular interest among these are the graphs $\mathcal{M}_t \subseteq \mathcal{G}_t$ that are minimal with respect to a certain partial order $\leq$ on t -boundaried graphs. We will prove that $\mathcal{O}$ is a subset of

$$\mathcal{M} = \bigcup_{t=1}^{B+1} \mathcal{M}_t$$

considering the graph of $\mathcal{M}$ with the boundaries forgotten.

There are three items to be clarified:

- (1) how the graphs of the small universe are generated,
- (2) the *search ordering* $\leq$,
- (3) the nature of the stop signal for width t .

- (1) *The order of generation of $\mathcal{U}_t^{\text{small}}$.*

Suppose X is a t -boundaried graph, $X \in \mathcal{U}_t^{\text{small}}$. By the *size* of X , we will refer to the number of nodes in a smallest possible parse tree for X over the standard set of parsing operators given by the Parsing Theorem (Theorem 6.72). For a given t , we generate the t -boundaried graphs of $\mathcal{U}_t^{\text{small}}$ in order of increasing size. By the j -th generation, we refer to all of those graphs of size j in this process.

- (2) *The search ordering $\leq$.*

To define $\leq$, we first extend the minor ordering of ordinary graphs to t -boundaried graphs in the natural way by holding the boundary fixed. In other words, the boundaried minor order is defined by the same local operations as the minor order, except that we are not allowed to delete boundary vertices or to contract edges between two boundary vertices. This can be shown to be a WQO on $\mathcal{U}_t^{\text{large}}$ by using the Graph Minor Theorem for edge-colored graphs (see Exercise 6). Let $\leq_m$ denote the minor order on ordinary graphs and let $\leq_{\partial m}$ denote the boundaried minor order (see Exercise 7.8.6).

For $X, Y \in \mathcal{U}_t^{\text{small}}$, define $X \leq Y$ if and only if $X \leq_{\partial m} Y$ and $X \sim Y$. This is a WQO since there are only finitely many equivalence classes of $\sim$ on $\mathcal{U}_t^{\text{small}}$.

(3) *The stop signal.*

The graphs of $\mathcal{U}_t^{\text{small}}$ are generated by size, one generation at a time (where the j th generation consists of all those of size j). We say that there is *nothing new at time j* if none of the t -boundaried graphs of the j th generation are minimal with respect to the search order $\leq$.

A *stop signal is detected at time $2j$* if there is nothing new at time i for $i = j, \dots, 2j - 1$.

We have now completely described the algorithm. For $t = 1, \dots, B + 1$ we generate the t -boundaried graphs in the manner described until a stop signal is detected. We form the set $\mathcal{M}$ and output the list of elements of $\mathcal{M}$ (with boundaries forgotten) that are obstructions for $\mathcal{F}$. Note that having a decision algorithm for $\mathcal{F}$ is sufficient to determine if any particular graph H is an obstruction, just by checking that $H \notin \mathcal{F}$ while each minor of H is in $\mathcal{F}$. This same procedure and the decision algorithm for $\sim$ allow us to compute whether it is time to stop.

The correctness of the algorithm is established by the following claims.

Claim 1. For each value of t a stop signal is eventually detected.

This follows immediately from the fact that $\leq$ is a **WQO** on $\mathcal{U}_t^{\text{small}}$ and, therefore, there are only a finite number of minimal elements.

Claim 2. Suppose that for a given t a stop signal is detected at time $2j$. Then, no obstruction for $\mathcal{F}$ that can be parsed with the t -boundaried set of operators has size greater than $2j$.

If T is rooted tree, then by a *rooted subtree* T' of T we mean a subtree that is generated by some vertex r of T (the root of T'), together with all of the vertices descended from r in T . For t -boundaried graphs X and Y , we say that X is a *prefix* of Y if, in a parse tree T for Y , X is parsed by a rooted subtree T' of T . To denote that X is a prefix of Y , we write $X \prec Y$.

Now suppose that T is a parse tree of minimum size for a counterexample H to Claim 2. Since all of the operators in the standard set are either binary or unary, there must be a prefix H' of H of size at least j . Since there is nothing new during the times when H' would have been generated, Claim 2 follows from Claim 3.

Claim 3. A prefix of a graph that is minimal with respect to $\leq$ must also be minimal.

If X is a prefix of Y and X is not minimal, then $X \geq_{\text{am}} X'$ with $X \neq X'$ and $X \sim X'$. Since $\sim$ is a right congruence, $Y \sim Y'$, where Y' is obtained from Y by substituting a parse tree for X' for the subtree that parses X in a parse tree for Y . Using the Parsing Replacement Property (see Observation 6.74) and since X' is a proper boundaried minor of X , Y' is a proper boundaried minor of Y . This implies that Y is not minimal with respect to $\leq$.

Claim 4. If $X \in \mathcal{O}$, then for some $t \leq B + 1$, $X \in \mathcal{M}_t$.

Since the treewidth of X is at most B , X can be parsed by the standard set of t -boundaried operators for some $t \leq B + 1$ by the Parsing Theorem, Theorem 6.72. It remains to argue that X (with the boundary supplied by this parse) is $\leq$ minimal. But this is obvious, since any proper minor is in $\mathcal{F}$ and since $\sim$ refines the canonical $\mathcal{F}$ -congruence.

The above theorem really tightens the connections between finite index congruences, WQO's, finite obstruction sets, and finite-state recognizability that we have seen emerging from a variety of different angles. We have the following notable corollary.

Corollary 7.88 *If $\mathcal{F}$ is a minor ideal of bounded treewidth, then the following sets of information about $\mathcal{F}$ are effectively interchangeable:*

- (1) *A decision algorithm for $\mathcal{F}$ and (for all t) a finite index right congruence on U_t^{small} that refines the canonical congruence for $\mathcal{F}$.*
- (2) *For all t , a finite automaton that recognizes the graphs of $\mathcal{F}$ from parse trees.*
- (3) *The obstruction set for $\mathcal{F}$.*

Proof. The information (2) immediately gives us (1). Starting from (1), (2) can be computed by Theorem 6.28 and the subsequent discussion. From the obstruction set (3), we can compute the information (2) by the methods of Chapter 6. Theorem 7.87 completes the picture.

Jens Lagergren [323] has shown that the use of the Graph Minor Theorem in proving that the algorithm of Theorem 7.87 terminates can be replaced by an explicit calculation of a “stopping time” computable from the index of the congruence $\sim$ [322].

Perhaps surprisingly, Theorem 7.87 can be implemented and a number of previously unknown obstruction sets have been mechanically computed (Cattell and Dinneen [115] Cattell, Dinneen, and Fellows [116]). The “Holy Grail” of such efforts would be a computation of the obstruction set for torus embedding, which probably contains about 2000 graphs.

Theorem 7.87 can be adapted to other partial orders, including those such as the topological order, that are not WQO's. It can be shown in this case that the (adapted) algorithm will correctly terminate if and only if the ideal $\mathcal{F}$ has a finite obstruction set—thus providing a potentially interesting way to mechanically prove the *existence* of a finite basis for particular ideals.

Exercises 7.9

- 7.9.1.** Find an *FPT* self-reduction algorithm for the **MINIMUM DEGREE GRAPH PARTITION** problem (see Exercise 5).
- 7.9.2.** Find a self-reduction for the **HAMILTON CIRCUIT** problem that makes $O(n \log n)$ calls to the decision problem oracle.
- 7.9.3.** Suppose that we have a linear *FPT* “black box” decision algorithm for the parameterized problem:

LONGEST CYCLE

- Instance:* A graph G and a positive integer k .
Parameter: k .
Question: Does G have a cycle of length at least k ?

By means of an efficient self-reduction algorithm, show that we can solve the corresponding search problem (i.e., find a cycle of length $\geq k$ when one exists) in time $f(k)n^2 \log n$.

- 7.9.4.** It is possible to formalize search problems for bounded pathwidth computations as follows (this can be generalized to bounded treewidth). A *balanced string relation* is a set R of pairs of strings of equal length: $R \subseteq \Sigma^* \times \Gamma^*$ where $(x, y) \in R$ implies $|x| = |y|$. Because the paired strings of R have equal length, we can equivalently view (x, y) as a string over the alphabet Δ , where $\Delta = \Sigma \times \Gamma$. The search problem for R is: Given $x \in \Sigma^*$, compute $y \in \Gamma^*$ such that $(x, y) \in R$ (if such a y exists). In Bodlaender, Evans and Fellows [72] it is shown that this search problem can be solved by a two-way finite-state computation (R is *finite-state searchable*) if R , when viewed as a language over the alphabet of symbol-pairs Δ , is finite-state recognizable. Fix t and let Σ be the standard set of parsing operators for pathwidth t . Let Γ be a set of color symbols that can be used to indicate a 3-coloring of a graph by pairing a color symbol to a parse symbol that introduces a new vertex. Prove that this R is finite-state recognizable and therefore finite-state searchable.

8

Miscellaneous Techniques

In this chapter, we explore three additional methods for proving fixed-parameter tractability results. Two of these build upon and extend the bounded-treewidth ideas of Chapter 6. The first is the basic graph-theoretic algorithmic technique of depth-first search. The second involves a clever utilization of tree decompositions in attacking subgraph problems. The third technique employs a family of number-theoretic hash functions to address parameterized problems in the combinatorics of finite sets. In particular, we will utilize k -perfect families of hash functions.

8.1 Depth-First Search

Depth-first search is the basis of a number of fundamental polynomial-time complexity results, including the linear-time algorithm to determine whether a graph is planar, due to Hopcroft and Tarjan [271, 439]. It is not surprising that it has proven to be a powerful method in devising *FPT* algorithms as well.

A depth-first search of a graph $G = (V, E)$, where $|V| = n$ and $|E| = e$, allows us to compute, in time $O(n + e)$, a rooted spanning tree T of G having the useful property that for every edge uv that is *not* an edge of T , either u is an ancestor of v with respect to T , or v is an ancestor of u . This is illustrated in Figure 8.1.

Such a spanning tree is termed a *depth-first spanning tree* of G , and the edges of the graph that do not belong to the spanning tree are referred to as *back edges*. The following theorem of Mader is occasionally useful in this context.

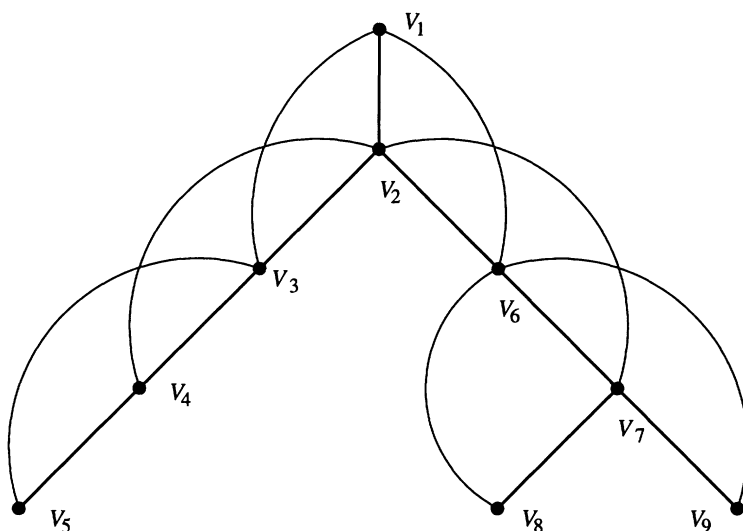


FIGURE 8.1. An Example of a Depth-First Spanning Tree

Theorem 8.1 (Mader [340]) *For every fixed graph H , there is a constant c_H , so that if $G = (V, E)$ is any graph with $e \geq c_H n$, then G necessarily contains a subdivision of H .*

The following is perhaps one of the simplest possible illustrations of how depth-first search and bounded treewidth techniques can be combined to yield interesting *FPT* results. It is, in some sense, a paradigmatic result in this area.

LONG CYCLE

Instance: A graph G and a positive integer k .

Parameter: A positive integer k .

Question: Does G have a cycle of length at least k ?

Theorem 8.2 (Fellows and Langston [218]) *LONG CYCLE can be solved for a graph G of order n in time $f(k)n$.*

Proof. We can first apply Mader's theorem to find out if there is an easy reason to answer "yes" just by counting to see if there are enough edges in the graph to conclude that G contains a subdivision of the graph $H = C_k$, the cycle of length k . If not, then e is bounded by a constant times n , and so in time $O(n)$, we can compute a depth-first spanning tree T of G .

In time $O(n)$, we can examine this depth-first spanning tree to see if any back edge "stretches" a distance greater than or equal to $k - 1$, that is, whether the path between its endpoints in the spanning tree has length at least $k - 1$. If so, then the back edge together with this path forms a cycle of length at least k , and we are done.

Otherwise, we can read off a tree decomposition of width $k - 2$ for G from the spanning tree, and solve the problem in time $O(n)$ using the techniques of Section 6.5, since the property of having a cycle of length at least k can be expressed in monadic second-order logic.

The tree that indexes the vertex sets of the tree decomposition can be identified with T . The set S_u associated to a vertex u of T consists of u together with its $k - 2$ immediate ancestors (or as many as there are, in the case of vertices close to the root of T). Given that no back edge stretches a distance greater than $k - 2$, it is easy to verify that the conditions for a tree decomposition are satisfied.

We remark that it is not really necessary to use Mader's result for the theorem above. We can avoid this by doing a depth-first traversal of G and checking for stretched back edges as we go, progressively computing a tree decomposition or stopping because a long cycle has been found.

It is interesting to note that the seemingly similar problem, **EXACT LONG CYCLE**, of determining whether G has a cycle of length *exactly* k cannot be handled by any easy (linear-time) modification of the above algorithm. In fact, this problem is only known to be *FPT* by the very different methods of the next section.

The approach embodied in the above (easy) theorem has been generalized by Bodlaender [64, 68]. The basic idea of the above theorem is to perform a depth-first search of G , and by studying the structure of the resulting depth-first spanning tree T , either identify an interesting substructure (e.g., a long cycle) or output a bounded-width tree decomposition based on T (in the sense that T provides the index structure for the decomposition). In the theorem above, the interesting structure was provided by a "stretched" edge. By considering the number of back edges that are *around* a vertex v with respect to T (either joining v to a predecessor of v or joining a predecessor to a descendant), a similar (although much more intricate) argument can be carried forward, a result due to Bodlaender [64].

Theorem 8.3 (Bodlaender [64]) *Let k and l be constants. There is an algorithm that in time $O(n)$ either:*

- (1) *Identifies in G a subgraph H that is contractible to the $2 \times k$ grid, or*
- (2) *Identifies in G a subgraph contractible to the l th circus graph, or*
- (3) *Outputs a tree decomposition of G of width at most $2(k - 1)^2(l - 1) + 1$.*

An example of a circus graph is shown in Figure 8.2.

Some applications of this theorem are explored in the exercises.

Exercises 8.1

8.1.1. Prove that there is a linear-time algorithm which takes as input a graph G and outputs a depth-first spanning tree for G .

(Hint: Suppose that $G = (V, E)$ is connected. Let $V = \{v_0, \dots, v_n\}$ list the vertices of G . Construct T in n steps. Let the root of T be v_0 , and $V_0 = \{v_0\}$ and $E_0 = \emptyset$. Now in step $i + 1$, let v_j be the largest number with v_j adjacent in G to some $y \notin V_i$. Let $v_{i+1} = y$, $V_{i+1} = V_i \cup \{y\}$ and $E_{i+1} = E_i \cup \{v_j y\}$.

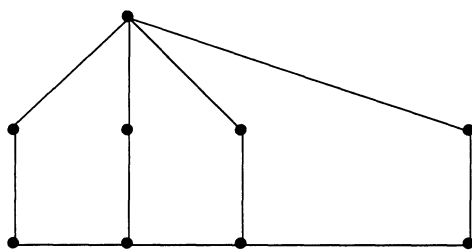


FIGURE 8.2. A Circus Graph of Size 4

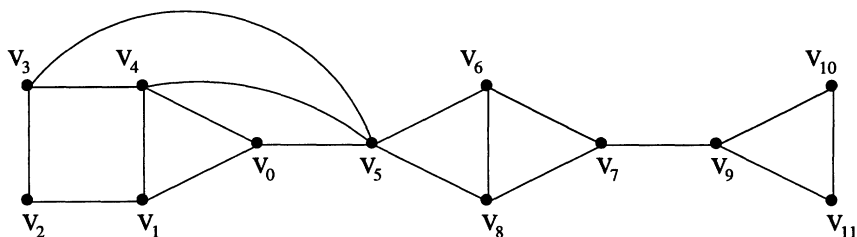


FIGURE 8.3. Graph for Exercise 2

- 8.1.2.** Compute a depth-first spanning tree of the graph of Figure 8.3, with root v_0 .
- 8.1.3.** Use depth-first search to construct an algorithm which determines if a graph has a vertex whose removal disconnects the graph.
- 8.1.4.** Use the theorem of Bodlaender [64] discussed in this section to prove that the problem of determining whether a connected graph G has a spanning tree with at least k leaves (**MAX LEAF SPANNING TREE**) is solvable in time $f(k)n$.
- 8.1.5.** The *cycle cover number* $\gamma(G)$ is the minimum number of vertices in set $V' \subseteq V$ such that $G - V'$ is acyclic. The **FEEDBACK VERTEX SET** problem asks, for a graph G and integer parameter k , whether $\gamma(G) \leq k$. Use Exercise 6 below to show that the **FEEDBACK VERTEX SET** problem is solvable in time $f(k)n$.
- 8.1.6.** Use the theorem of Bodlaender (Theorem 8.3) to prove that in time $f(k)n$, we can accomplish one of the following:
- (1) Find a tree decomposition of width at most $g(k)$.
 - (2) Find $k + 1$ vertex disjoint cycles in G .
 - (3) Find a vertex u with the property that if $V' \subseteq V$ is any k -element feedback vertex set in G , then $u \in V'$.
- 8.1.4.** (Downey and Fellows [165]) A theorem of Erdős and Posa [197] shows that if $\gamma(G) \geq C \cdot k \log k$, then G has more than k disjoint cycles. Use this and Exercise 6 to show that the problem **DISJOINT CYCLES** that

seeks to determine for a graph G and integer parameter k whether G contains k vertex disjoint cycles can be solved in time $f(k)n$.

- 8.1.5.** For any fixed k , the graphs that do not contain more than k disjoint cycles are well-quasi-ordered by topological containment [90]. Argue that if $\mathcal{F}$ is a family of graphs that is a topological lower ideal and for which there is a bound on $\gamma(G)$ for $G \in \mathcal{F}$, then $\mathcal{F}$ can be recognized in linear time.

8.2 Bounded-Width Subgraphs, the Plehn-Voigt Theorem, and Induced Subgraphs

A useful result in the toolkit of fixed-parameter tractability is the theorem of Plehn and Voigt that for subgraphs restricted by a fixed bound on treewidth, the **SUBGRAPH ISOMORPHISM** problem is *FPT*. This problem takes two graphs G and H as input, and asks whether H is a subgraph of G . They proved a related result for the **INDUCED SUBGRAPH ISOMORPHISM** problem when the maximum degree of G is bounded.

An (ordinary) subgraph of a graph G can be viewed as constituted by some subset of the vertices of the G together with some subset of the edges of G that are between these vertices. An *induced* subgraph of G , by contrast, is constituted by some subset of the vertex set of G together with *all* of the edges of G between these vertices.

Formally, a graph $H = (V_H, E_H)$ is a *subgraph (induced subgraph)* of a graph $G = (V_G, E_G)$ if there is a 1:1 map $f : V_H \rightarrow V_G$ such that $uv \in E_H$ implies $f(u)f(v) \in E_G$ [$uv \in E_H$ if and only if $f(u)f(v) \in E_G$]. A map f satisfying the appropriate condition is termed an *embedding* of H in G (as a subgraph or induced subgraph).

Plehn and Voigt [373] proved the following result by an elegant argument.

Theorem 8.4 [373] *For graphs H of treewidth at most w , it can be determined in time $f(|H|)|G|^{w+1}$ whether H is a subgraph of G .*

Proof. We prove the theorem for the case of trees ($w = 1$) and indicate how this can be generalized. This special case encompasses all of the main ideas and the intuition for the general result, but is much less notationally cumbersome and is free of some tedious details concerning parsing schemes.

Let H be a fixed forest (i.e., a graph of treewidth 1) and suppose that H is rooted (and thus the root can be viewed as the boundary of H , of size $t = 1$). It is easy to establish that a complete set of 1-boundary operators for the parsing of graphs of treewidth at most 1 is provided by the binary $\oplus$ operator and by the unary operators $\otimes_0$ and $\otimes_1$, where $\otimes_0$ adds a new isolated vertex that becomes the new root (i.e., the boundary of the resulting graph), and $\otimes_1$ adds a new vertex that becomes the new root, together with an edge between the old root and the new root. We will assume that H is given by parse tree P over this operator set in the

manner of Section 6.4. Since $\otimes_0$ and $\otimes_1$ are handled in essentially the same way in the algorithm, we will conveniently assume that H is connected (i.e., a tree) and parsed by the operator subset consisting of $\oplus$ and $\otimes_1$. We will assume also that each leaf of P represents a single edge between two vertices, one of which is the boundary.

Let $k = |H|$ denote the number of vertices of H and let $n = |G|$ be the order of G .

The algorithm proceeds by working leaf-to-root in the parse tree P . Suppose that the current node in P is labeled $\oplus$. The algorithm will receive from the argument branches two sets of embeddings (in G) S_1 and S_2 of the two rooted subtrees T_1 and T_2 of H (respectively) whose roots are identified (by the $\oplus$ operator) at this step of the parse. The goal of the algorithm at this point is to calculate a set S of embeddings of $T = T_1 \oplus T_2$ sufficient to allow a determination, recursively, when the root of H is reached, of whether there are any embeddings of H in G , by just checking whether the set S is nonempty.

If S_i consisted of *all possible* embeddings of T_i , then S could be calculated by combining pairs of embeddings (of T_1 and T_2 , respectively) that are disjoint except for their common root (in $T_1 \oplus T_2$). However, a moment's thought shows that we would then have sets S_i of order n^k . The algorithm is based on a neat trick that allows us to get away with sets S_i of size bounded by only k^k , even though the calculation of S consists of combining interior-disjoint root-compatible pairs as suggested. The trick is originally due to Monien [356] and depends on the data structure of *representation systems* that we next describe.

For each vertex u of the parse tree P , let P_u denote the subparse tree rooted at u and let T_u denote the rooted subtree of H parsed by P_u . Let T'_u denote the interior vertices of T_u , and let r_u denote the root vertex of T_u . Let $k_1(u) = |T'_u|$ and let $k_2(u) = k - k_1(u) - 1$. Thus, $k_2(u)$ is the number of vertices of H that are not in the subtree T_u of H .

Definition 8.5 (Representative System) Let u be a vertex of the parse tree P , and let v be a vertex of G . A *representation system* $S_u(v)$ for P_u at v consists of the following information, subject to various compatibility conditions:

- (1) A rooted *representation tree* $\Gamma_u(v)$ of depth at most $k_2(u)$, with each vertex that is not a leaf having $k_1(u)$ children.
- (2) To each node α of $\Gamma_u(v)$ is associated a subset of the vertices of G , $A_\alpha \subseteq V_G$, that we will term the *avoidance set* for the node α .
- (3) To each node α of $\Gamma_u(v)$ is associated either:
 - an embedding h_α of T_u in $G - A_\alpha$ that maps the root of T_u to v , $h_\alpha(r_u) = v$, or
 - the label λ .
- (4) Each edge of $\Gamma_u(v)$ is labeled with a vertex of G .

The compatibility conditions of Definition 8.5 are as follows:

- (1) If α is the root of $\Gamma_u(v)$, then $A_\alpha = \emptyset$.

- (2) If α is a node of $\Gamma_u(v)$ and the set of vertices of G occurring as labels on the path from the root of $\Gamma_u(v)$ to α is the set A , then the associated avoidance set is $A_\alpha = A$.
- (3) If λ is associated to a node α of $\Gamma_u(v)$ then:
 - α is a leaf, and
 - it is not possible to embed T_u in $G - A_\alpha$ so that r_u is mapped to v .
- (4) If an embedding h_α is associated to a node α of $\Gamma_u(v)$ and if J is the set of image vertices of $h_\alpha(T'_u)$, then the vertices in J each occur once as a label on an edge in $\Gamma_u(v)$ between α and a child of α . For a vertex u of the parse tree P , a *representation system* S_u is defined to be a collection of representation systems $S_u(v)$, one for each vertex $v \in V_G$. We will refer to the $S_u(v)$ as *subsystems* of S_u .

Intuitively, one can think about the situation addressed by a representation system in the following way. Imagine that we wish to run the algorithm sketched above, but instead of keeping *all possible* embeddings of T_i in G , we decide to keep just one (or more precisely, at most one) for each possible vertex $u \in V_G$ to which we could conceivably map the root of T_i . (This will not work, but carry on.)

In processing the current node u of the parse tree P that we assume is labeled with $\oplus$, we could see if there is a vertex $v \in V_G$ such that the (single) embeddings h_1 in $S_1(v)$ (for T_1) and h_2 in $S_2(v)$ (for T_2) are compatible; i.e., the interior vertices are mapped disjointly into G , so that h_1 and h_2 can be combined into an embedding h of $T_1 \oplus T_2$. This might “usually” be the case, since the tree H that we are trying to embed is fixed (as the parameter) and therefore “small.” If we are unlucky, we might want to inquire if we can have an embedding that is *like* h_1 , except that there is just one vertex that we now need to avoid in order to have h_1 and h_2 with disjoint images of the interiors T'_1 and T'_2 (so that they can be combined). It is the job of the representation systems to supply these “variations” on the (one) embedding that is associated to the root of a representation tree, if they exist. Note that the branching factor in a representation tree is the number of vertices in the interior of the subtree that is embedded, and thus the number of different “single vertex errors” that might occur and need to be repaired by a variation. All of this discussion is just motivational; we now describe the algorithm.

The Algorithm. For each vertex u of the parse tree P , compute (from leaf-to-root) a representation system S_u . Answer “yes” if and only if for the root r of P , one of the subsystems $S_r(v)$ is nonempty in the sense that the root of the representation tree $\Gamma_r(v)$ is not labeled by λ . To complete the description, we must describe how the computation is performed at each vertex u of P . There are three cases: (1) u is a leaf, (2) u is labeled $\oplus$, and (3) u is labeled $\otimes_1$.

(1) u is a leaf.

In this case, u represents a single edge, one endpoint of which is the boundary. For a vertex $v \in V_G$, the representation tree $\Gamma_u(v)$ will have a branching factor of 1, since there is a single internal vertex of T_u . The representation system $S_u(v)$ is

easily computed in time $O(n)$. The main task is checking, at most $k_2(u) = k - 2$ times, whether there is any edge in G that avoids a set of at most $k - 2$ vertices.

(2) u is labeled $\oplus$.

Let u' and u'' denote the children of u in the parse tree P . We assume that we have already computed representation systems $\mathcal{S}_{u'}$ and $\mathcal{S}_{u''}$. For each vertex $v \in V_G$, we compute $\mathcal{S}_u(v)$ by starting with the root of the representation tree $\mathcal{S}_u(v)$ and proceeding toward the leaves.

Let α be a node of $\Gamma_u(v)$ for which we wish to compute a label. To do this, try all possible pairs of embeddings (h', h'') where h' is an embedding of $T_{u'}$ that occurs as a label in $\Gamma_{u'}(v)$, and h'' is an embedding of $T_{u''}$ that occurs as a label in $\Gamma_{u''}(v)$. Given such a pair, check:

- whether $h'(T'_{u'}) \cap h''(T'_{u''}) = \emptyset$, i.e., the embeddings of the interiors of the subtrees joined by the $\oplus$ operator are disjoint, and
- whether $A_\alpha \cap (h'(T_{u'}) \cup h''(T_{u''})) = \emptyset$, i.e., the embedding of T_u obtained by combining h' and h'' avoids A_α .

If both of these conditions are met, then label α with the combined embedding. Otherwise, label α with the “impossibility” label λ .

(3) u is labeled $\otimes_1$.

Let u' denote the child of u in the parse tree P . We assume that a representation system $\mathcal{S}_{u'}$ has already been computed. Let $v \in V_G$ and let α be a node of $\Gamma_u(v)$ for which we wish to compute a label. For every edge $vw \in E_G$, see if there is an embedding h used as a label in $\Gamma_{u'}(w)$ that meets the conditions:

- $v \notin h(T'_{u'})$
- $A_\alpha \cap (\{v\} \cup h(T_{u'})) = \emptyset$

If both of these conditions are met for some edge uw , then label α with the “extension” of h that sends r_u to v and otherwise coincides with h on $T_{u'}$. Otherwise, label α with λ .

The operator $\otimes_0$ would be handled similarly by considering all $w \neq v$ (i.e., not necessarily adjacent vertices in G).

We now argue that the algorithm is correct. Note that by compatibility condition (3) in the definition of a representation system (Definition 8.5), the only issue is whether the above procedure correctly computes representation systems. We argue by structural induction on the parse tree P . The representation systems computed for the leaves of P are trivially valid. The crucial point is that the computation procedure will find a valid embedding to associate to a node α if one exists. We consider the cases of $\oplus$ and $\otimes_1$ separately.

For the case of $\oplus$, suppose there is some embedding h of $T_u = T_{u'} \oplus T_{u''}$ that maps r_u to v and is disjoint from A_α . By the induction hypothesis, there must be some h' in $\Gamma_{u'}(v)$ that avoids $A' = A_\alpha \cup h(T'_{u''})$, noting that the height of $\Gamma_{u'}(v)$ is $k_2(u') \geq |A_\alpha| + |T'_{u''}|$. Similarly, there must be some h'' in $\Gamma_{u''}(v)$ that avoids $A'' = A_\alpha \cup h(T'_{u'})$. In order to locate such an h' that avoids A' in $\Gamma_{u'}(v)$, start at the root of the representation tree and follow an edge to a child that is labeled by

a vertex $a \in A'$ (if none exists, then we are done). This puts a into the avoidance sets of this subtree, by the compatibility conditions (2) and (4). Such a descent needs to be made at most $|A'|$ times until a suitable h' is located; similarly for h'' .

For the case of $\otimes_1$, correctness is obvious.

The running time of the algorithm can be roughly bounded as follows. The size of the parse tree P can be bounded by $C \cdot |H|$ for a small constant C . For each vertex u of P , we must compute a representation system S_u that consists of n representation systems $S_u(v)$, one for each vertex $v \in V_G$. To compute $S_u(v)$ takes time $C(k^k)^2$ in the case that the vertex u of the parse tree P represents the $\oplus$ operator and requires time Cnk^k in the case that u represents the $\otimes_1$ operator. The total runtime is therefore at most $Ck^{2k+1}n^2$.

We next describe how the above algorithm can be generalized for $w \geq 2$.

We make use of the fact that there is a complete set of parsing operators for graphs of treewidth at most w over the set of graphs of boundary size w (where one of these operators $\otimes$ has arity w , and the graph of $\otimes$ has order $w + 1$). The details concerning this parsing scheme are not too important.

The modifications to the algorithm described above for $w = 1$ are straightforward. First of all, each subtree of the parse tree P for H now represents a graph with boundary size w ; so, instead of having one representation system $S_u(v)$ for each vertex v of G that might be the boundary of an embedding in G of T_u , the subgraph of H parsed by P_u , where u is a vertex of P , we need to maintain a representation system $S_u(B)$ for each w -element subset $B \subseteq V_G$ that might be the boundary of the subgraph of H parsed by P_u in this generalized situation. The internal vertices T'_u of the subgraph of H parsed by P_u are defined in the obvious way, as is the definition of a representation system.

Because the size of the graph of $\otimes$ is $w + 1$ (and there seems to be no way to avoid this for treewidth w while maintaining boundary size w), we have to consider all subsets of V_G of size $w + 1$ in computing the representation systems $S_u(B)$ when u represents the operator $\otimes$. Consequently, the running time of the generalized algorithm is $O(n^{w+1})$.

The generalization to graphs that are edge- and vertex-colored with a fixed finite set of colors can be observed to be straightforward also. In Plehn and Voigt [373] the extension to *minimally weighted* subgraphs is also proved.

The technique of representation systems can be applied to the **INDUCED SUBGRAPH ISOMORPHISM** problem when the maximum degree of G is bounded.

Theorem 8.6 (Plehn and Voigt [373]) *For graphs H of treewidth at most w , it can be determined in time $g(|H|, d)|G|^{w+1}$ whether H is an induced subgraph of G for graphs G of maximum degree d .*

Note that the above theorem is “best possible” at least in the sense that we would not expect to be able to get rid of the bound on the maximum degree of G , since determining whether there is an induced subgraph consisting of k isolated vertices (a graph of width $w = 0$) is equivalent to determining whether G has a k -element independent set, a problem we do not expect to be able to solve in linear time

for fixed k . (In fact, we see that this problem is one of “basic hardness results” in Part II.)

Exercises 8.2

- 8.2.1. Give a parsing scheme for graphs of treewidth at most w that fulfills the requirements of the proof in this section. What is the minimum number of operators in such a scheme?
- 8.2.2. Use the theorem of this section to show that the problem **EXACT LONG CYCLE** of determining whether a graph G has a cycle of length exactly equal to k can be solved in time $f(k)n^3$.
- 8.2.3. Use the 3-edge-colored version of the theorem of this section to show that the parameterized **3-DIMENSIONAL MATCHING (SHORT 3-DIMENSIONAL MATCHING)** problem is *FPT*. This problem takes as input a set $M \subseteq W \times X \times Y$, where W , X , and Y are disjoint sets. The question is whether there is a subset $M' \subseteq M$ of size k such that no two elements of M' agree in any coordinate. Can this trick be extended to r -dimensional matching?

8.3 Hashing

In this section, we describe a method for designing *FPT* algorithms that is based on families of hash functions. The method was first described by Alon, Yuster and Zwick [17] who called it the method of “color-coding.” It can be used to address a wide variety of problems, including the problem of **MULTIDIMENSIONAL MATCHING** that occurs as an important subroutine in other *FPT* algorithms.

There are several ways to construct the families of hash functions used in this approach. The most “efficient” families have been described by Alon, Yuster and Zwick in [17] by means of sophisticated probabilistic methods that are beyond the scope of this book, although we will describe their results below. We next describe a simpler but less efficient construction due to Dennenberg, Gurevich and Shelah [154] and improved slightly by Koblitiz [302], whose proof we give here. This is sufficient to provide a basis for the method, although “better” *FPT* algorithms are obtained by using more efficient families of hash functions.

Let $l_1(x) = \log_2(x)$, and $l_r(x) = \log_2(l_{r-1}(x))$ for $r > 1$, be the r -th repeated log function. Let $e_1(x) = 2^x$ and $e_r(x) = 2^{e_{r-1}(x)}$ be the r -th repeated exponential functions, i.e., the inverse functions of $l_1(x)$ and $l_r(x)$, respectively. For integers $u_1 > u_2 > \dots > u_r > 0$, let $\text{mod}(u_1, \dots, u_r)$ be the function defined by reduction mod u_1 followed by reduction mod u_2 followed by $\dots$ followed by reduction mod u_r .

Lemma 8.7 (Dennenberg, Gurevich and Shelah [154], Koblitiz [302]) *Let r be an arbitrary positive integer and let $n_0 = e_r(8)$. Let $X \subset \{1, \dots, n\}$ where $n \geq n_0$*

and $|X| = k$. Then, there exist integers $n > u_1 > u_2 > \dots > u_r > 0$ such that $u_r < \frac{1}{2}k^2 l_r(n) + k^{2+\epsilon}$ and the function $\text{mod}(u_1, \dots, u_r)$ is an injective map on X .

Proof. Let $B(u)$ denote the least common multiple of all positive integers less than u : $B(u) = \text{l.c.m.}(1, 2, \dots, u-1)$. We use the fact that if u is an integer ≥ 8 , then $B(u) > 2^u$. To see this, note that the Chebyshev ψ -function is related to $B(u)$ as follows: $\exp(\psi(x)) = B([x]+1)$, and so it suffices to know that $\psi(x) > (\ln 2)x$ for $x \geq 7$. By Theorem 8* of [411] we have $\psi(x) > \left(1 - \frac{8.0720}{\ln^2 x}\right)x$, and this is greater than $(\ln 2)x$ for $x \geq 175$. For $175 \geq u \geq 8$, the inequality $B(u) > 2^u$ is quickly checked by hand.

Define u_1 as the smallest positive integer such that $\text{mod}(u_1)$ is an injective map on X . Let A be the least common multiple of all differences $|x - y|$ as x and y range over pairs of distinct elements of X . Then,

$$A \leq \prod_{x, y \in X, x > y} (x - y) < n^{k(k-1)/2}.$$

But $B(u_1)$ must divide A , since otherwise there would be a $j < u_1$ that does not divide A , and hence does not divide any of the differences $x - y$, contrary to the definition of u_1 . Thus, $B(u_1) \leq A$, and so $2^{u_1} < n^{k(k-1)/2}$. Taking $\log_2$ of both sides gives

$$u_1 < k(k-1)/2 \log_2(n) < \frac{1}{2}k^2 \log_2(n).$$

Now repeat the argument in the last paragraph with the set X replaced by its image X_1 under reduction modulo u_1 , with n replaced by u_1 , etc. One finds u_2 such that $\text{mod}(u_1, u_2)$ is an injective map on X and such that

$$u_2 < \frac{1}{2}k^2 \log_2(u_1) < \frac{1}{2}k^2 \log_2(k^2 \log_2(n)) = \frac{1}{2}k^2 \log_2(\log_2(n)) + k^2 \log_2(k).$$

The lemma then follows by induction on r .

Corollary 8.8 Choose r to be the maximum possible value such that $n \geq e_r(8)$. Then, $\text{mod}(u_1, \dots, u_r)$ is a function from $\{1, \dots, n\}$ to the set $J(k)$ consisting of the first $Ck^{2+\epsilon}$ non-negative integers, where C is independent of k and n .

Let $\mathcal{H}(n, k)$ denote the family of hash functions $h : \{1, \dots, n\} \rightarrow J(k)$ based on all possible sequences $(u_1, \dots, u_r)$. The size of this collection can be bounded:

$$|\mathcal{H}(n, k)| \leq k^{2 \log^* n} (\log n)^2 \leq Ck^{\log k} (\log n)^3$$

for some constant C independent of k and n .

Note that for $h \in \mathcal{H}(n, k)$ and $x \in \{1, \dots, n\}$, the value of $h(x)$ can be computed in time $O(\log n)^3$.

The problem is defined as follows.

MULTIDIMENSIONAL MATCHING

- Input:** A set $M \subseteq X_1 \times \cdots \times X_r$, where the X_i are disjoint sets.
Parameter: The positive integer r and a positive integer k .
Question: Is there a subset $M' \subseteq M$ with $|M'| = k$, such that no two elements of M' agree in any coordinate?

Theorem 8.9 (Downey, Fellows and Koblitz [180]) **MULTIDIMENSIONAL MATCHING** is fixed-parameter tractable.

Proof. Let N denote the number of distinct coordinates that appear in the r -tuples of M , and assume that these coordinates are $\{1, \dots, N\}$. Let $K = kr$. Let n denote the total size of the description of M .

Define a *solution schema* S for the problem to be a set of k r -tuples with all coordinates distinct, and with the coordinates chosen from $J(K)$. If α is an r -tuple in M , $\alpha = (x_1, \dots, x_r)$, let $h(\alpha)$ denote the image r -tuple, $h(\alpha) = (h(x_1), \dots, h(x_r))$.

The Algorithm.

- For each $h \in \mathcal{H}(N, K)$ and for each solution schema S :
 - (1) Compute $h(\alpha)$ for each $\alpha \in M$.
 - (2) Determine whether the solution schema S is *realized*, meaning that every r -tuple in S occurs as the image $h(\alpha)$ of some $\alpha \in M$. If so, answer “yes.”
- If no hash function h realizes a solution schema, then answer “no.”

The correctness of the algorithm is nearly immediate. If M has a k -matching, then this involves $K = kr$ distinct coordinates, and by the lemma, there must be some $h \in \mathcal{H}(N, K)$ that maps these injectively to $J(K)$, and the image under h is the solution schema that causes the algorithm to output “yes.” Conversely, if there is an h that realizes a solution schema S , then choosing one r -tuple in each of the k preimages yields a matching.

The running time of the algorithm is bounded by $O(K!K^{3K+1}n(\log n)^6)$.

The running time for the algorithm above can be improved simply by using a better family of hash functions.

Definition 8.10 (*k*-Perfect Hash Functions) A family $\mathcal{F}$ of functions from $\{1, \dots, n\}$ to $\{1, \dots, m\}$ is termed *k-perfect* if for every subset $S \subseteq \{1, \dots, n\}$ of size k , there is some function $f \in \mathcal{F}$ that is injective on S .

By means of a variety of sophisticated methods, Alon, Yuster and Zwick [17] have proved the following two theorems.

Theorem 8.11 [17] *There is a k-perfect family of hash functions from $\{1, \dots, n\}$ to $\{1, \dots, k\}$ of size $2^{O(k)} \log n$.*

Theorem 8.12 [17] *There is a k-perfect family of hash functions from $\{1, \dots, n\}$ to $\{1, \dots, k^2\}$ of size $k^{O(1)} \log n$.*

Exercises 8.3

- 8.3.1.** Use the hashing technique to show that **DISJOINT r -SUBSETS** is fixed-parameter tractable. This problem takes as input a collection $\mathcal{F}$ of r subsets of a set X and a positive integer k and must answer whether there are k subsets in $\mathcal{F}$ that are disjoint. The parameter is the pair (r, k) .
- 8.3.2.** Use the hashing technique to show that **ANTICHAIN OF r -SUBSETS** is fixed-parameter tractable. This problem takes as input a collection $\mathcal{F}$ of r subsets of a set X and a positive integer k and seeks to determine if there are k subsets $S_1, \dots, S_k \in \mathcal{F}$ such that $\forall i, j \in \{1, \dots, k\}$ with $i \neq j$, both $S_i - S_j$ and $S_j - S_i$ are nonempty. The parameter is the pair (r, k) .
- 8.3.3.** Define **POLYCHROME MATCHING** to be the problem that takes as input an r -edge-colored graph G and must answer whether there is a partial matching in G consisting of r edges, with one edge of each color. Use the hashing technique to show that this problem is fixed-parameter tractable.
- 8.3.4.** Define a *pseudo-homomorphism* from a graph G to a graph H to be a map $f : V(G) \rightarrow V(H)$ that is onto and that has the property that whenever xy is an edge of H , then there is a vertex $u \in f^{-1}(x)$ and there is a vertex $v \in f^{-1}(y)$ such that $uv \in E(G)$. Use the method of hashing to show that the problem that takes as input G and H and determines whether there is a pseudo-homomorphism from G to H is fixed-parameter tractable for the parameter H .

Part II

Parameterized Intractability

9

Reductions

In Part I, we looked at a number of methods of establishing fixed-parameter tractability. Part II is devoted to problems where there is no known method which enables us to demonstrate fixed-parameter tractability. We are concerned with the situation where we can prove, or at least have good evidence to believe, that no such method exists. As we will see, for some combinatorial problems, we can actually *prove* intractability. However, as with classical *NP* and *PSPACE* “intractability,” for most interesting natural problems we can only offer a completeness theory.

It is important to note that, although we share some *methodology* with classical *NP*-completeness, our theory is quite different. However, to motivate the methodology, we will begin by reviewing classical *NP*-completeness. As we see it, the basic idea behind virtually all completeness results is the notion of a *reduction*. For an appropriate mode of computation, a reduction of a language L to a language L' is a method that computes L from L' in this mode. The best known classical examples are *polynomial-time m -reductions* ($\leq_m^P$) and *polynomial time T -reductions* ($\leq_T^P$.) For languages L and L' , we recall that $L \leq_T^P L'$ iff there is a polynomial-time Oracle Turing Machine Φ such that for all $x \in \Sigma^*$, $x \in L$ iff $\Phi^{L'}(x) = 1$. Polynomial-time T -reductions are sometimes called Cook reductions in honor of Stephen Cook who first studied them [131]. Polynomial-time m -reductions, or Karp reductions, were introduced by Richard Karp in his classic article [291]. They are often called polynomial-time *transformations* and are a special case of polynomial-time T -reductions. We recall that $L \leq_m^P L'$ iff there is a polynomial-time computable function $f : \Sigma^* \rightarrow \Sigma^*$, such that for all $x \in \Sigma^*$, $x \in L$ iff $f(x) \in L'$. Karp demonstrated the importance of polynomial time m -reductions by constructing transformations between many natural combinatorial problems in a wide variety of diverse areas, thereby revealing how widespread the *NP*-completeness phenomenon is.

Example 9.1 A polynomial-time m -reduction. Consider the following classical problems.

CNF SATISFIABILITY

- Input:* A formula X of propositional logic in conjunctive normal form (i.e., a collection of clauses).
Question: Is X satisfiable? That is, is there an assignment of the variables that makes X true?

3SAT

- Input:* A formula X of propositional logic in conjunctive normal form with maximum clause size 3.
Question: Is X satisfiable?

The classical result of Karp [291] is that $\text{CNF SATISFIABILITY} \leq_m^P \text{3SAT}$ and hence $\text{CNF SATISFIABILITY} \equiv_m^P \text{3SAT}$. The transformation is the following. Let X be an instance of **CNF SATISFIABILITY**. We compute in polynomial time $f(X)$ an instance of **3SAT** such that X is satisfiable iff $f(X)$ is satisfiable. Without loss of generality, we may assume that each clause of X has ≥ 3 literals. Let $U = \{u_1, \dots, u_m\}$ be a clause of X . We replace this clause by the following clauses:

$$U' = \{u_1, u_2, z_1\}, \{\bar{z}_1, u_3, z_2\}, \{\bar{z}_2, u_4, z_3\}, \dots, \{\bar{z}_{m-3}, u_{m-1}, u_m\},$$

where $\{z_1, \dots, z_{m-3}\}$ are new variables, chosen for this clause. Let $f(X)$ be the result obtained from X by performing this local replacement for each clause of X . Notice that the size of $f(X)$ is bounded by $3mq$, where m is the size of the largest clause and q is the number of clauses. Also, notice that any satisfying assignment for the collection U' must make U true, and hence X is satisfiable iff $f(X)$ is satisfiable.

Example 9.2 A polynomial-time T -reduction. Nice examples of this sort of thing are often provided by showing that a *search* problem reduces to a *decision* problem. One such example is provided by

HAMILTON PATH

- Input:* A graph G .
Question: Does G have a path which travels through every vertex of G exactly once?

Although it is not a decision problem, we are often interested in the associated search problem: If G has a Hamilton path, *find* one or else return that G has no Hamilton path. Of course, if we can do the search problem in polynomial time, we can do the decision problem. The algorithm that reduces the search problem to the decision one is the following. Given an oracle Ω for the decision problem and a graph G , first use Ω to see if G has a Hamilton path. If the answer is no, then output that G has no Hamilton path. If Ω tells us that G does have a Hamilton path, then choose any edge e of G and form the graph $G' = G - e$. Now ask Ω if G' has a

Hamilton path. If the answer is no, then we can conclude that every Hamilton path in G includes the edge e , and we should resolve never again to delete e . However, we can try again with a different edge e' , forming $G' = G - e'$. It is obvious that if G is not simply a path (in which case we are done) then by repeatedly trying different edges we can eventually find an edge e such that for $G' = G - e$ we will be informed by Ω that the answer for G' is yes. By making at most one query for each edge in the graph, we have thus narrowed our search to a subgraph G' of G with one fewer edge. Repeating this process, we will eventually identify a subgraph G' that is a Hamilton path in G , having made (in view of our resolutions along the way) at most one query to Ω for each edge in G .

Notice that the key difference between the T - and the m -reductions pointed out in the above examples is that in the m -reduction we compute $f(x)$ and then ask a single question of Ω , namely if $f(x)$ is in L' , whereas in the T -reduction, *many* questions need to be asked of the oracle Ω to get the result. Also the reduction is adaptive in the sense that answers from earlier questions affect the next question. Even the two examples above suggest many natural questions. For instance, is the solution of the search problem from the decision problem *intrinsically sequential*? It has been shown that if we can solve the search problem from the decision problem without the use of sequential questions then many unlikely collapses occur for complexity theory. (See, e.g., [9] and [97].)

Reductions allow us to partially order languages in terms of their computational complexity. Two languages, L and L' , are taken to have the same complexity from the point of view of the given reduction iff they are in the same *degree*; that is, there is a reduction from L to L' and vice versa. We will need to create reductions which express the fact that two languages have the same *parameterized* complexity. What is needed are reductions that ensure that if L' is computable in time $f(k)n^c$ “by the slice,” then there is a parameterized algorithm for L running in time $g(k)n^{c'}$ “by the slice.” After a moment’s thought, we realize that we can achieve such reductions only if we allow each slice of L to reduce to a finite number of slices of L' . The easiest way to make such a reduction is to reduce, in parameterized polynomial time, the k -th slice of L to the k' -th slice of L' . This leads us to the *working definition*, Definition 9.3 below, of a parameterized reduction for combinatorial reductions. From the point of view of our hardness results, this is the most important definition in the book.

Definition 9.3 (Basic Working Definition) We say that L reduces to L' by a *standard* parameterized m -reduction if there are functions $k \mapsto k'$ and $k \mapsto k''$ from $\mathbb{N}$ to $\mathbb{N}$, and a function $\langle x, k \rangle \mapsto x'$ from $\Sigma^* \times \mathbb{N}$ to Σ^* , such that

$$\begin{aligned} \langle x, k \rangle \mapsto x' \text{ is computable in time } k''|x|^c, \text{ and} \\ \langle x, k \rangle \in L \text{ iff } \langle x', k' \rangle \in L'. \end{aligned}$$

We remark that usually one can take $k' = k''$.

Example 9.4 The following is a simple example of a standard parameterized reduction. It comes from an observation about a classical reduction of Karp. Define a formula (of propositional logic) to be *monotone* if it contains no negation symbols, and *antimonotone* if it contains no negation symbols except for the literals which are *all* negated. Consider the following problem:

WEIGHTED CNF SATISFIABILITY

Input: A propositional formula X in conjunctive normal form.
Parameter: A positive integer k .
Question: Does X have a satisfying assignment of Hamming weight k ?
 (An assignment has *weight* k iff it has exactly k variables set to be *true*.)

WEIGHTED CNF SATISFIABILITY gives us a parameterized language which is a natural parameterized version of **CNF SATISFIABILITY**. We can similarly define a parameterized version of **3SAT**, **WEIGHTED 3SAT** by considering weight k solutions of problems in **3CNF** form. Finally, we can define **WEIGHTED MONOTONE CNF SATISFIABILITY** and **WEIGHTED ANTIMONOTONE CNF SATISFIABILITY** by considering only monotone and antimonotone formulas.

We also need another problem.

WEIGHTED BINARY INTEGER PROGRAMMING

Instance: A binary matrix A and a binary vector $\mathbf{b}$.
Parameter: A positive integer k .
Question: Does $A \cdot \mathbf{x} \geq \mathbf{b}$ have a binary solution of weight k ?

We have the following:

Theorem 9.5 **WEIGHTED MONOTONE CNF SATISFIABILITY** *reduces to WEIGHTED BINARY INTEGER PROGRAMMING via a standard reduction.*

Proof. Let (X, k) be an instance of **MONOTONE WEIGHTED CNF SAT**. Let $C_1, \dots, C_p$ list the clauses of X and $x_1, \dots, x_m$ list the variables. Let A be the matrix $\{a_{i,j} : i = 1, \dots, p, j = 1, \dots, m\}$ with $a_{i,j} = 1$ if x_j is present in C_i , and $a_{i,j} = 0$ otherwise. Let $\mathbf{b}$ be the vector with 1 in the j -th position for $j = 1, \dots, p$. It is easy to see that $A \cdot \mathbf{x} \geq \mathbf{b}$ has a solution of weight k iff X has a satisfying assignment of weight k (and the reasoning is reversible).

Actually, the situation in Theorem 9.5 is unusual. It is generally not the case that a classical reduction turns out to be *also* a parameterized reduction for natural parameterized versions of the problems. Classical reductions usually do not carry enough structure. This lack of structure in the classical reductions causes us to lose control of the parameter. For instance, consider the situation of Example 9.1 where **CNF SATISFIABILITY** is classically reduced to **3SAT**. There, we replace a clause with m literals by $m - 2$, clauses each consisting of 3 literals. For each clause, we introduced $m - 3$ new variables. Now, suppose the original formula has a weight k satisfying assignment that makes exactly one variable u_j in clause

U of X true. Then, to make $f(X)$ true, we would need to make $z_1, \dots, z_{j-2}$ all true. So the weight of the satisfying assignment of $f(X)$ would not only depend upon k but would depend on j , and hence the clause size of X . So this reduction from CNF SATISFIABILITY to 3SAT does *not* give a *parameterized* reduction from WEIGHTED CNF SATISFIABILITY to WEIGHTED 3SAT. It is not known if there is any such parameterized reduction. We will soon see that in fact, there is some evidence that there really is no such reduction, and hence from a parameterized point of view CNF SATISFIABILITY and 3SAT are *not* the same. We remark that if there are NP-complete problems that are not fixed parameter tractable, then it must be that some of the classical reductions do not lift to parameterized ones since we have seen that some NP-complete problems are fixed-parameter tractable.

Although Definition 9.3 is satisfactory as a working definition, for the general situation we need to be more precise. In Definition 2.4, we have seen that there are three basic flavors of fixed-parameter tractability, depending on the level of uniformity. Naturally, the three types of tractability will relativize to three flavors of reducibility. We get the following technical definitions. **[We remark, however, that if the reader is only interested in the concrete hardness results of Part II, then they can ignore the definitions to follow and take Definition 9.3 as the actual definition. We will not be needing the definitions below until Part III, where we look at the fine structure of problems under parameterized reducibilities.]**

Definition 9.6 (Uniform Fixed-Parameter Reducibility) Let A and B be parameterized problems. We say that A is *uniformly fixed-parameter-reducible* to B if there is an oracle procedure Φ , a constant α , and an arbitrary function $f : \mathbb{N} \mapsto \mathbb{N}$ such that

- (a) the running time of $\Phi(B; \langle x, k \rangle)$ is at most $f(k)|x|^\alpha$,
- (b) on input $\langle x, k \rangle$, Φ only asks oracle questions of $B^{(f(k))}$, where

$$B^{(f(k))} = \bigcup_{j \leq f(k)} B_j = \{\langle x, j \rangle : j \leq f(k) \& \langle x, j \rangle \in B\},$$

- (c) $\Phi(B) = A$.

If A is uniformly P -reducible to B , we write $A \leq_T^u B$. Where appropriate we may say that $A \leq_T^u B$ *via* f . If the reduction is many:1 (an m -reduction), we will write $A \leq_m^u B$.

Definition 9.7 (Strongly Uniform Reducibility) Let A and B be parameterized problems. We say that A is *strongly uniformly fixed-parameter-reducible* to B if $A \leq_T^u B$ via f where f is recursive. We write $A \leq_T^s B$ in this case.

Definition 9.8 (Nonuniform Reducibility) Let A and B be parameterized problems. We say that A is *nonuniformly fixed-parameter-reducible* to B if there is a constant α , a function $f : \mathbb{N} \mapsto \mathbb{N}$, and a collection of procedures $\{\Phi_k : k \in \mathbb{N}\}$ such that $\Phi_k(B^{(f(k))}) = A_k$ for each $k \in \mathbb{N}$, and the running time of Φ_k is $f(k)|x|^\alpha$. Here, we write $A \leq_T^n B$.

Note that the above are good definitions since whenever $A \leq B$ with $\leq$ any of the reducibilities, if B is fixed-parameter tractable, so too is A . Note also that the above definitions allow us to specify the notions of fixed-parameter tractability we had before. Now, nonuniform fixed-parameter tractability corresponds to being $\leq_T^n \emptyset$. We will henceforth write $FPT(\leq)$ as the fixed-parameter-tractable class corresponding to the reducibility $\leq$.

If $A \leq B$ via a standard reduction in the sense of Definition 9.3, then $A \leq_m^u B$. Furthermore, in most cases, we will look at $A \leq_m^s B$ since the function $k \mapsto k'$ will be recursive. Finally, we remark that the standard definition works for natural languages since they all have the following property: for all k , L_k is “directly coded into” (or “trivially represented in”) L_{k+1} . For instance, for any formula φ in variables $x_1, \dots, x_n$, φ has a weight k satisfying assignment iff $\varphi \wedge x_{n+1}$ has a weight $k + 1$ satisfying assignment. Thus, in a “natural” language L if $L' \leq L$, then L' reduces to a single slice of L because of this upward encoding of the slices.

Historical Notes

The notions of reducibilities and computability were developed in the 1930s in the setting of classical recursion theory by many authors, notably Post, Kleene, and Church. Of course, the notion of a reduction is in some sense much older: Consider the reduction that $a \leq b$ iff a is, say, algebraic over b . The recognition of the difficulty of NP -complete problems has a very interesting history, and the reader is referred to the illuminating article of Hartmanis [261]. The groundbreaking articles here are Karp [291], Cook [131], and Levin [330]. The notions of reducibilities we have used were developed by the authors in Downey-Fellows [165], [166], and [167]. We give fuller historical remarks on parameterized complexity at the end of the next chapter.

Exercises 9.0

- 9.0.1.** Prove that $\text{INDEPENDENT SET} \leq_m^s \text{MAXIMAL IRREDUNDANT SET}$ via a standard reduction. These problems are defined as follows. (Bodlaender and Kratsch [88] have proven the apparently stronger result that $\text{CNF SATISFIABILITY} \leq_m^s \text{MAXIMAL IRREDUNDANT SET}$. The Bodlaender-Kratsch result is apparently stronger because we will soon show that $\text{INDEPENDENT SET} \equiv_m^s \text{WEIGHTED 3SAT}$, and we believe that $\text{WEIGHTED 3SAT} <_m^s \text{WEIGHTED CNF SATISFIABILITY}$.)

$(k\text{-})\text{INDEPENDENT SET}$

Input: A graph G .
Parameter: A positive integer k .
Question: Does G have a set of k vectors $\{x_1, \dots, x_k\}$ such that for all $i \neq j$, x_i is not adjacent to x_j ?

(k -)MAXIMAL IRREDUNDANT SET

Input: A graph G .
Parameter: A positive integer k .
Question: Does G have a set X of k vertices such that for each member x of X , there is a $y \in V(G)$ such that either $y = x$ or (x, y) is an edge, and y is not adjacent to nor a member of S , and, furthermore, S is maximal with this property?

[Hint. Given a graph G and a k , we construct a graph H and a k' so that G has a k -element independent set iff H has a maximal irredundant set of size k' . H is constructed as follows. Construct k blocks consisting of $k + 1$ columns, each consisting of $n = |G|$ points. So column i in block j consists of n points $x(j, i, t)$ for $t = 1, \dots, n$. The idea is that a block represents a choice for a vertex in the independent set. Now, within a block, join all vertices except those with the same last coordinate. (We refer to this as a *row*.) The first column is not connected to anything else and is called the *dummy column*. Now, for each block i pick a column in each block $i' \neq i$ to identify with block i , and similarly identify a corresponding column i'' in block i . Do this in such a way that for each pair (i, j) , there is exactly one column $c_i(i, j)$ identified with exactly one column $c_j(i, j)$ in block j , and these columns are used for no other identifications, for any other pair except (i, j) . Let (q, r) be some such pair of columns. Now, connect each $x(i, q, t)$ to $x(j, r, s)$ iff t and s are adjacent vertices in G .

Now, let $k' = k(k + 1)$. First, if G has an independent set of size k , let $\{t_1, \dots, t_{k-1}\}$ be this independent set. The relevant irredundant set for H is obtained by taking the union of vertices of the t_i -th row of block i , for $i = 1, \dots, k$. They are each their own private neighbors, as the corresponding set in G is independent, and it is maximal by the fact that the blocks are almost cliques. Conversely, take S to be a size k' maximal irredundant set in H . Prove that it must be of the form above and, hence, corresponds to an independent set in G of size k . There are two cases depending on whether some block B has more than k vertices of S . Remember that a maximal irredundant set must also be a dominating set.]

9.0.2. Prove that **WEIGHTED ANTIMONOTONE 2SAT** (see Example 9.4) reduces to **COLORED AUTOMORPHISM** (below) by a standard reduction.

COLORED AUTOMORPHISM

Instance: A 2-colored (bipartite) graph G .
Parameter: A positive integer k .
Question: Is there an automorphism preserving colors moving exactly k blue vertices?

[Hint: Let X be an instance of **WEIGHTED ANTIMONOTONE 2-CNFSAT** with parameter k . Let $C_1, \dots, C_m$ be the clauses of X and $u_1, \dots, u_n$ the variables which will all appear in the clauses as negated since X is antimonotone. The graph $G(X)$ is constructed as follows:

For each u_i , we have two blue nodes $\overline{u(i)}$ and $u(i)$. For each clause C_j , we have three red nodes $C(j, k) : k = 1, 2, 3$. For each i , append to each $u(i)$ and $\overline{u(i)}$ a rigid tree T_i so that additionally for all $i \neq j$, T_i is not isomorphic to T_j . All the nodes of the T_i are red. (This will force $u(i)$ to go to $\overline{u(i)}$ if it moves at all.) Finally, we need some further edges. Let $C_j = \{\overline{u_p}, \overline{u_q}\}$. Add the edges $\overline{u(p)}C(j, 1)$, $\overline{u(q)}C(j, 1)$, $\overline{u(p)}C(j, 2)$, $\overline{u(q)}C(j, 2)$, and $u(p)C(j, 3)$, $u(q)C(j, 3)$.

Prove that X has a weight k satisfying assignment iff $G(X)$ has an automorphism that moves exactly $2k$ blue vertices.]

9.0.3. (The Advice View.) In Part I, Chapter 5, we saw that an alternative view of parameterized tractability is provided by considering FPT to be the class of parameterized problems that are solvable in time $c \cdot n^\alpha$ by an oracle Turing machine Φ with a finite piece of advice $f(k)$. We can define $A \leq_{\text{advice}} B$ via $f : \mathbb{N} \mapsto \Sigma^*$ if there exists an oracle Turing machine Φ running in time n^α such that

$$\langle x, k \rangle \in A \text{ iff } \Phi^{B^{(f(k)) \oplus f(k)}}(\langle x, k \rangle) = 1.$$

Prove that $A \leq_T^n B$ iff $A \leq_{\text{advice}} B$.

- 9.0.4.** (i) Prove that there exists a language $L \in FPT(\leq_m^u)$ such that $L \notin FPT(\leq_T^s)$.
(ii) Prove that there exists a recursive language $L' \in FPT(\leq_m^n)$ with $L' \notin FPT(\leq_T^u)$.

[Hint: (i) Construct a language $L = \cup_s L_s$ in stages to diagonalize against all possible $\leq_T^s$ reductions. The reader should see Theorem 19.21 in Chapter 19.]

10

The Basic Class $W[1]$ and an Analog of Cook's Theorem

In the last chapter, we looked at *one* of the ingredients of classical hardness results: the notion of a reducibility. The *other* basic ingredient is the identification of classes of intractable problems. In the case of NP -completeness, recall that a language L is a member of the class NP iff there is a polynomial-time relation R and a polynomial p such that

$$x \in L \text{ iff } \exists y[|y| \leq p(|x|) \wedge R(x, y) \text{ holds}]$$

For instance, if L is the collection of satisfiable formulas, then a formula x is satisfiable iff there is some satisfying assignment y of the variables. Recall that L is NP -complete iff $L \in NP$, and for any language $L' \in NP$, $L' \leq_m^p L$. The hidden beauty and power of these definitions comes from the profound discovery that there are literally thousands of important and natural NP -complete problems. If *any* of them were solvable in polynomial-time, then *all* of them would be. Moreover, other than a few exceptions, if a natural problem is in NP and is apparently not in P , then the problem seems to always be NP -complete.

Again, following this methodology, we will need to identify a class of problems that we feel are fixed-parameter intractable. We would then need to show that this class corresponds to a wide collection of problems. For this class, if any of the problems were *fixed-parameter* tractable, then all of the problems in the class would be *fixed-parameter* tractable.

An important element of the classical theories of, for instance, NP - and $PSPACE$ -completeness is the identification of a basic generic problem. The classical problem identified by Cook-Levin is the following:

TURING MACHINE ACCEPTANCE

Input: A nondeterministic Turing machine M , a string x , and a number n (in unary).

Question: Is there a computation path of M accepting x in $\leq n$ steps?

It is clear that **TURING MACHINE ACCEPTANCE** is NP -complete, since $\exists y(|y| \leq p(|x|) \wedge R(x, y))$ holds can be modeled by a nondeterministic Turing machine M that guesses y and checks if $R(x, y)$ holds. Thus, $x \in L$ iff $\langle M, x, 1^{p(|x|)} \rangle$ is in **TURING MACHINE ACCEPTANCE**. So, assuming the polynomial-time intractability of **TURING MACHINE ACCEPTANCE**, we now have a way of demonstrating polynomial-time intractability of another problem X simply by showing that **TURING MACHINE ACCEPTANCE** $\leq_P^p X$. Based upon the NP -completeness of **TURING MACHINE ACCEPTANCE**, strong intuitive and philosophical arguments can be proposed for the thesis that $P \neq NP$. Many NP -complete combinatorial problems such as **INTEGER PROGRAMMING**, say, have some sort of natural algebraic structure. On the face of it, one might believe that with sufficiently clever algebraic insight, we might just be able to exploit this algebraic structure to prove that **INTEGER PROGRAMMING** is in P and hence is theoretically tractable. But **INTEGER PROGRAMMING** is NP -complete, so if we can prove it to be in P , we *must* also have **TURING MACHINE ACCEPTANCE** in P . However, in the case of **TURING MACHINE ACCEPTANCE**, the point is that a nondeterministic Turing machine is such an opaque and generic object that it simply does not seem reasonable that we should be able to decide in polynomial time whether a given Turing machine on a given input has some accepting path. Cook's theorem therefore gives powerful evidence that $P \neq NP$.

In the present chapter, we will prove our own analog of Cook's theorem by proving that a number of combinatorial problems are of the same fixed-parameter complexity as a generic problem about nondeterministic Turing machines. The key underlying problem is the following.

SHORT TURING MACHINE ACCEPTANCE

Input: A nondeterministic Turing machine M and a string x .

Parameter: A positive integer k .

Question: Does M have a computation path accepting x in $\leq k$ steps?

It seems to us that if one accepts the philosophical argument that **TURING MACHINE ACCEPTANCE** is intractable, then the same reasoning would suggest that **SHORT TURING MACHINE ACCEPTANCE** is *fixed-parameter* intractable. We will soon establish that there are a large number of problems of the same fixed-parameter complexity as **SHORT TURING MACHINE ACCEPTANCE**. We believe that the existence of such a large number of problems of the same fixed-parameter complexity as **SHORT TURING MACHINE ACCEPTANCE** gives further weight to our thesis that **SHORT TURING MACHINE ACCEPTANCE** is not fixed-parameter tractable.

The main theorem of this chapter is an analog of Cook's theorem. We will first establish some preliminary results. We will need some definitions. It is convenient

to consider a 3CNF formula as a (boolean) *circuit*. Thus, a 3CNF formula is considered as a circuit consisting of one input (of unbounded fanout) for each variable, possibly inverters below the variable, and structurally a large *and* of small *or*'s (of size 3) with a single output line. The reader should refer to Figure 10.1. We can similarly consider a 4CNF formula to be a large *and* of small *or*'s, where "small" is defined to be 4. More generally, it is convenient to consider the model of a *decision circuit*. This is a circuit consisting of large and small gates with a single output line, and no restriction on the fanout of gates. For such a circuit, the *depth* is the maximum number of gates on any path from the input variables to the output line, and the *weft* is the "large gate depth." More formally, we define the weft of a circuit as follows.

Definition 10.1 (Weft) Let C be a decision circuit. The *weft* of C is defined to be the maximum number of large gates on any path from the input variables to the output line. (A gate is called *large* if its fanin exceeds some preagreed bound. The reader should note that *none* of our results depend upon what the bound actually is.)

In Figure 10.1, we have an example of a weft 2 depth 5 decision circuit.

Let $\mathcal{F} = \{C_1, \dots, C_n, \dots\}$ be a family of decision circuits. Associated with $\mathcal{F}$ is a basic parameterized language

$$L_{\mathcal{F}} = \{\langle C_i, k \rangle : C_i \text{ has a weight } k \text{ satisfying assignment}\}.$$

For instance, if $\mathcal{F}$ is the family of boolean circuits corresponding to propositional formulas in 3CNF form, then $L_{\mathcal{F}}$ corresponds to **WEIGHTED 3CNF SATISFIABILITY**. A generalization of the class of circuits corresponding to 3CNF formulas is provided by the following.

WEIGHTED WEFT t DEPTH h CIRCUIT SATISFIABILITY (WCS(t, h))

Input: A weft t depth h decision circuit C .
Parameter: A positive integer k .
Question: Does C have a weight k satisfying assignment?

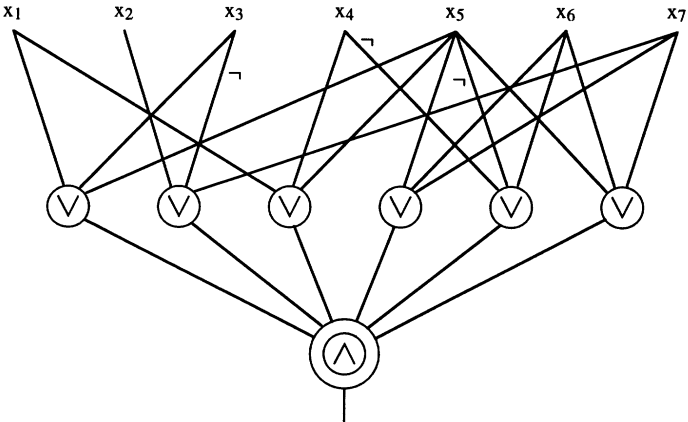
Note that **WEIGHTED 3CNF SATISFIABILITY** is subsumed by **WEFT 1 DEPTH 2 DECISION CIRCUIT SATISFIABILITY**.

Notation: We will denote by $L_{\mathcal{F}(t,h)}$ the parameterized language associated with the family of weft t depth h decision circuits.

Definition 10.2 (Basic Hardness Class) We define a language L to be in the class $W[t]$ iff L is fixed parameter reducible to $L_{\mathcal{F}(t,h)}$ for some h .

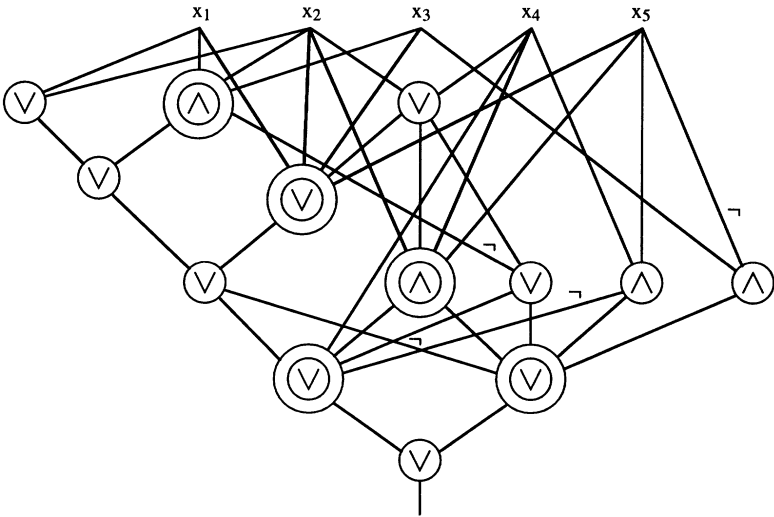
We remark that we have not specified the type of parameterized reduction in Definition 10.2. As we said earlier, in all natural instances one may take it to be the one from Definition 9.3. As we noted earlier, in all cases of hardness results of

Large gates (unbounded fanin) represented by  Small gates (bounded fanin) represented by 



A 3CNF Formula is a large *and* of small *or*'s.

$$\varphi = (x_1 + x_3 + x_5)(x_2 + \neg x_3 + x_7)(x_1 + x_4 + x_5)(x_5 + x_6 + x_7)(\neg x_4 + \neg x_5 + x_6)(x_5 + x_6 + x_7)$$



A Weft 2 Depth 5 Decision Circuit.

FIGURE 10.1. Examples of Circuits

which we are aware, the reduction is a standard one and hence the reduction is a $\leq_m^s$ reduction. (Note the parallel with the classical NP -completeness results where all known hardness results turn out to be via Karp reductions.)

Theorem 10.3 (Analog of Cook's Theorem) *The following are complete for $W[1]$:*

- (i) **WEIGHTED n -SATISFIABILITY** for any fixed $n \geq 2$.
- (ii) **SHORT TURING MACHINE ACCEPTANCE**.

The remainder of the chapter is devoted to proving Theorem 10.3. The proof is quite technical and involves a number of intermediate steps. These steps are of independent interest and establish a number of other $W[1]$ -completeness results. We begin with some preliminary *normalization* results about weft 1 circuits.

By definition, a parameterized problem $L \in W[1]$ reduces to $L_{\mathcal{F}(1,h)}$ for the family $\mathcal{F}(1, h)$ of weft 1 circuits of depth bounded by h , for some h . The following argument shows that we may assume the circuits in F to have depth 2 and a particularly simple form, consisting of a single output *and* gate which receives arguments from *or* gates having fanin bounded by a constant h' . Thus, each such circuit is isomorphically represented by a boolean expression in conjunctive normal form having clauses with at most h' literals. We will say that a family of circuits having this form is *normalized*. With this in mind, we have the following definition.

Definition 10.4 The family of parameterized problems $W[1, s]$ is defined to be those parameterized problems in $W[1]$ reducible to $L_{\mathcal{F}(s)}$ for the family $\mathcal{F}(s)$ of depth 2 weft 1 normalized circuits, with the *or* gates on level 1 having fanin bounded by s .

Lemma 10.5 *Let $\mathcal{F}$ be a family of weft 1 circuits of depth bounded by a constant h . Then, $L_{\mathcal{F}}$ is reducible (by a standard reduction) to $L_{\mathcal{F}(s)}$ for $s = 2^h + 1$, and hence $L_{\mathcal{F}} \in W[1, s]$.*

Proof. Let $C \in \mathcal{F}$ and let k be a positive integer. We describe how to produce a circuit $C' \in \mathcal{F}(s)$ and an integer k' such that C accepts a weight k input if and only if C' accepts an input of weight k' .

Step 1. The reduction to tree circuits.

The first step is to transform C into an equivalent weft 1 *tree circuit* C' of depth at most h . In a tree circuit, every logic gate has fanout one, and thus the circuit can be viewed as equivalent to a boolean formula. The transformation can be accomplished by replicating the portion of the circuit above a gate as many times as the fanout of the gate, beginning with the top level of logic gates, and proceeding downward level by level. The creation of C' from C may require time $O(|C|^{O(h)})$ and involve a similar blowup in the size of the circuit. This is permitted since h is a fixed constant independent of k and $|C|$.

Step 2. Moving the *not* gates to the top of the circuit.

Let C denote the circuit we receive from the previous step (we will use this notational convention throughout the proof). Transform C into an equivalent circuit C' by commuting the *not* gates to the top, using DeMorgan's laws. This may

increase the size of the circuit by at most a constant factor. The tree circuit C' thus consists (from the top) of the input nodes, with *not* gates on some of the lines fanning out from the inputs. In counting levels, we consider all of this as level 0.

Step 3. A preliminary depth 4 normalization.

The goal of this step is to produce a tree circuit C' of depth 4 that corresponds to a boolean expression E in the following form. (For convenience, we use product notation to denote logical $\wedge$ and sum notation to denote logical $\vee$.)

$$E = \prod_{i=1}^m \sum_{j=1}^{m_i} E_{ij},$$

where

- (1) m is bounded by a function of h ,
- (2) for all i , m_i is bounded by a function of h ,
- (3) for all i, j , E_{ij} is either

$$E_{ij} = \prod_{k=1}^{m_{ij}} \sum_{l=1}^{m_{ijk}} x[i, j, k, l]$$

or

$$E_{ij} = \sum_{k=1}^{m_{ij}} \prod_{l=1}^{m_{ijk}} x[i, j, k, l],$$

where the $x[i, j, k, l]$ are literals (i.e., input boolean variables or their negations) and for all i, j, k , m_{ijk} is bounded by a function of h . The family of circuits corresponding to these expressions has weft 1, with the large gates corresponding to the E_{ij} . (In particular, the m_{ij} are *not* bounded by a function of h .)

To achieve this form, let g denote a large gate in C . An input to g is computed by a subcircuit of depth bounded by h consisting only of small gates, and so is a function of at most 2^h literals. This subcircuit can thus be replaced, at constant cost, by either a product-of-sums expression (if g is a large $\wedge$ gate) or a sum-of-products expression (if g is a large $\vee$ gate). In the first case, the product of these replacements over all inputs to g yields the subexpression E_{ij} corresponding to g . In the second case, the sum of these replacements yields the corresponding E_{ij} .

The output of C is a function of the outputs of at most 2^h large gates. This function can be expressed as a product-of-sums expression of size at most 2^{2^h} . At the cost of possibly duplicating some of the large gate subcircuits at most 2^{2^h} times, we can achieve the desired normal form with the bounds $m \leq 2^{2^h}$, $m_i \leq 2^h$, and $m_{ijk} \leq 2^h$.

Step 4. Employing additional nondeterminism.

Let C denote the normalized depth 4 circuit received from Step 3 and corresponding to the boolean expression E described above. For convenience, assume that the

E_{ij} for $j = 1, \dots, m'_i$ are sums-of-products and the E_{ij} for $j = m'_i + 1, \dots, m_i$ are products-of-sums. Let $V_0 = \{x_1, \dots, x_n\}$ denote the variables of E .

In this step, we produce an expression E' in product-of-sums form with the size of the sums bounded by $2^h + 1$ that has a satisfying truth assignment of weight

$$k' = 2k + k(1 + 2^h)2^{2^h} + m + \sum_{i=1}^m m'_i$$

if and only if C has a satisfying truth assignment of weight k . The main idea is to use additional (bounded weight) nondeterminism to guess both (1) a weight k input x for C and (2) additional information that will allow us to check that $C(x) = 1$ with a $W[1, s]$ circuit, $s = 2^h + 1$.

The set V of variables of E' is $V = V_0 \cup V_1 \cup V_2 \cup V_3$, where

$$\begin{aligned} V_1 &= \{x[i, j] : 1 \leq i \leq n, 0 \leq j \leq (1 + 2^h)2^{2^h}\}, \\ V_2 &= \{u[i, j] : 1 \leq i \leq m, 1 \leq j \leq m_i\}, \\ V_3 &= \{w[i, j, k] : 1 \leq i \leq m, 1 \leq j \leq m'_i, 0 \leq k \leq m_{ij}\}. \end{aligned}$$

The expression E' is a product of subexpressions $E' = E_1 \wedge \dots \wedge E_8$ described by

$$\begin{aligned} E_1 &= \prod_{i=1}^n (\neg x_i + x[i, 0])(\neg x[i, 0] + x_i), \\ E_2 &= \prod_{i=1}^n \prod_{j=0}^{2^{2^h+1}} (\neg x[i, j] + x[i, j + 1 \pmod{r}]), \quad r = 1 + (1 + 2^h)2^{2^h}, \\ E_3 &= \prod_{i=1}^m \sum_{j=1}^{m_i} u[i, j], \\ E_4 &= \prod_{i=1}^m \prod_{j=1}^{m_i-1} \prod_{j'=j+1}^{m_i} (\neg u[i, j] + \neg u[i, j']), \\ E_5 &= \prod_{i=1}^m \prod_{j=1}^{m'_i} \prod_{k=0}^{m_{ij}-1} \prod_{k'=k+1}^{m_{ij}} (\neg w[i, j, k] + \neg w[i, j, k']), \\ E_6 &= \prod_{i=1}^m \prod_{j=1}^{m'_i} (\neg u[i, j] + \neg w[i, j, 0]), \\ E_7 &= \prod_{i=1}^m \prod_{j=1}^{m'_i} \prod_{k=1}^{m_{ij}} \prod_{l=1}^{m_{ijk}} (\neg w[i, j, k] + x[i, j, k, l]), \\ E_8 &= \prod_{i=1}^m \prod_{j=m'_i+1}^{m_i} \prod_{k=1}^{m_{ij}} \left(\neg u[i, j] + \sum_{l=1}^{m_{ijk}} x[i, j, k, l] \right). \end{aligned}$$

To see that Step 4 works correctly, suppose τ is a weight k truth assignment to V_0 that satisfies E . We describe how to extend τ to weight k' truth assignment τ' to the variables V that satisfies E' as follows:

- (1) For each i such that $\tau(x_i) = 1$ and for $j = 0, \dots, (1 + 2^h)2^{2h}$, set $\tau'(x[i, j]) = 1$.
- (2) For each $i = 1, \dots, m$, choose an index j_i such that E_{i, j_i} evaluates to 1 (this is possible, since τ satisfies E) and set $\tau'(u[i, j_i]) = 1$.
- (3) If, in (2), E_{i, j_i} is a sum-of-products, then choose an index k_i such that

$$\prod_{l=1}^{m_{i, j_i, k_i}} x[i, j, k, l]$$

evaluates to 1, and correspondingly set $\tau'(w[i, j, k_i]) = 1$.

- (4) For $i = 1, \dots, m$ and $j = 1, \dots, m'_i$ such that $j \neq j_i$, set $\tau'(w[i, j, 0]) = 1$.

It is straightforward to check that the above described weight k' extension τ' satisfies E' .

Conversely, suppose ν' is a weight k' truth assignment to the variables of V that satisfies E' . We argue that the restriction ν of ν' to V_0 is a weight k truth assignment that satisfies E .

Claim 1. ν sets at most k variables of V_0 to 1.

If this were not so, then the clauses in E_1 and E_2 would together force at least $(k+1)(2+(1+2^h)2^{2h})$ variables to be 1 in order for ν' to satisfy E' , a contradiction as this is more than k' .

Claim 2. ν sets at least k variables of V_0 to 1.

The clauses of E_4 ensure that ν' sets at most m variables of V_2 to 1. The clauses of E_5 ensure that ν' sets at most $\sum_{i=1}^m m'_i$ variables of V_3 to 1. If Claim 2 were false, then for ν' to have weight k' , there must be more than k indices j for which some variable $x[i, j]$ of V_1 has the value 1, a contradiction in consideration of E_1 and E_2 .

The clauses of E_3 and the arguments above show that ν' necessarily has the following restricted form:

- (1) Exactly k variables of V_0 are set to 1.
- (2) For each of the k in (1), the corresponding $(1 + 2^h)2^{2h} + 1$ variables of V_1 are set to 1.
- (3) For each $i = 1, \dots, m$, there is exactly one j_i for which $u[i, j_i] \in V_2$ is set to 1.
- (4) For each $i = 1, \dots, m$ and $j = 1, \dots, m'_i$, there is exactly one k_i for which $w[i, j, k_i] \in V_3$ is set to 1.

To argue that ν satisfies E , it suffices to argue that ν satisfies every E_{i, j_i} for $i = 1, \dots, m$.

The clauses of E_6 ensure that if $v'(u[i, j]) = 1$, then $k_i \neq 0$. This being the case, the clauses of E_7 force the literals in a subexpression of E_{i,j_i} to evaluate in a way that shows E_{i,j_i} to evaluate to 1. The clauses of E_8 enforce that E_{i,j_i} evaluates to 1 for $j_i > m'_i$.

Thus, we have the following stratification of $W[1]$ that will be useful to our arguments:

$$W[1] = \bigcup_{s=1}^{\infty} W[1, s].$$

Our next preliminary result is to show that $W[1]$ collapses to $W[1, 2]$. Recall that a circuit C is termed *monotone* if it does not have any *not* gates. Equivalently, C corresponds to boolean expressions having only positive literals. If we restrict the definition of $W[t]$ and $W[1, s]$ to families of monotone circuits, we obtain the family of parameterized problems **MONOTONE** $W[t]$ and **MONOTONE** $W[1, s]$.

Recall also, we define a circuit to be **ANTIMONOTONE** if all the input variables are negated and the circuit has no other inverters. Thus, in an antimonotone circuit, each fanout line from an input node goes to a *not* gate (and in the remainder of the circuit, there are *no* other *not* gates). The restriction to families of antimonotone circuits yields the classes of parameterized problems **ANTIMONOTONE** $W[t]$ and **ANTIMONOTONE** $W[1, s]$.

Theorem 10.6 $W[1, s] = \text{ANTIMONOTONE } W[1, s]$ for all $s \geq 2$.

Proof. The plan of our argument is to identify a problem (***s*-RED/BLUE NONBLOCKER**) that belongs to antimonotone $W[1, s]$, and then show that the problem is hard for $W[1, s]$. **RED/BLUE NONBLOCKER** is the following parameterized problem.

RED/BLUE NONBLOCKER

- Input:* A graph $G = (V, E)$ where V is partitioned into two color classes $V = V_{\text{red}} \cup V_{\text{blue}}$.
- Parameter:* A positive integer k .
- Question:* Is there is a set of red vertices $V' \subseteq V_{\text{red}}$ of cardinality k such that every blue vertex has at least one neighbor that does not belong to V' .

The *closed neighborhood* of a vertex $u \in V$ is the set of vertices $N[u] = \{x : x \in V \text{ and } x = u \text{ or } xu \in E\}$.

It is easy to see that the restriction of **RED/BLUE NONBLOCKER** to graphs G of maximum degree s belongs to antimonotone $W[1, s]$ since the product-of-sums boolean expression

$$\prod_{u \in V_{\text{blue}}} \sum_{x_i \in N[u] \cap V_{\text{red}}} \neg x_i$$

has a weight k truth assignment if and only if G has size k nonblocking set.

Such an expression corresponds directly to a circuit meeting the defining conditions for antimonotone $W[1, s]$. We will refer to the restriction of **RED/BLUE**

NONBLOCKER to graphs of maximum degree bounded by s as s -RED/BLUE **NONBLOCKER**. We next argue that s -RED/BLUE **NONBLOCKER** is complete for $W[1, s]$.

By Lemma 10.5, we can assume that we are given a normalized boolean expression. Thus, let X be a boolean expression in conjunctive normal form with clauses of size bounded by s . Suppose X consists of m clauses $C_1, \dots, C_m$ over the set of n variables $x_0, \dots, x_{n-1}$. We show how to produce in polynomial time by local replacement, a graph $G = (V_{\text{red}}, V_{\text{blue}}, E)$ that has a nonblocking set of size $2k$ if and only if X is satisfied by a truth assignment of weight k .

Before we give any details, we give a brief overview of the construction, whose component design is outlined in Figure 10.2. There are $2k$ “red” components arranged in a line. These are alternately grouped as blocks from $V_{\text{red}} = V_1 \cup V_2$ ($V_1 \cap V_2 = \emptyset$), with a block of vertices from V_1 and then a block of vertices from V_2 in a manner to be precisely described below. The idea is that the V_1 blocks should represent a positive choice (corresponding to a literal being true) and the V_2 blocks should correspond to the “gap” until the next positive choice. We think of the V_1 blocks as $A(0), \dots, A(k-1)$. (With a V_2 block $B(i)$ between successive $A(i), A(i+1)$ blocks, the last following $A(k-1)$.) We will ensure that for each pair of red vertices in a block, there will be a blue vertex connected to the pair and nowhere else (these are the sets V_3 and V_5). This device ensures that at most one red vertex from each block can be chosen, and since we must choose $2k$, this ensures that we choose *exactly* one red vertex from each block. The reader should think of the V_2 blocks as arranged in k columns. Now, if i is chosen from a V_1 block, we will ensure that column i gets to select the next gap. To ensure this we connect a blue degree 2 vertex to i and each vertex not in the i -th column of the next V_2 block. Of course, this means that if i is chosen, since these blue vertices must have an unchosen red neighbor, we must choose from the i -th column. The final part of the component design is to enforce consistency in the next V_1 block; that is, if we choose i and have a gap choice in the next V_2 block, column i , of j , then the next chosen variable should be $i + j + 1$. Again, we can enforce this by using many degree 2 blue vertices to block any other choice. (These are the V_6 vertices.) Also, we keep the k choices for V_1 blocks in ascending order $1 \leq c_1 \leq \dots \leq c_k \leq n$ with c_i in $A(i)$ by the use of extra blue enforcers, the blue vertices V_8 . The idea here is that we ensure that for each j in $A(i)$ and each $j' \leq j$ in $A(q)$ with ($q > i$), there is a blue vertex v in V_8 adjacent to both j and j' . This ensures that if j is chosen in block $A(i)$, then we cannot choose any $j' \leq j$ in any subsequent V_1 block. The last part of the construction is to force consistency with the clauses. We do this as follows. For each way a nonblocking set can correspond to making a clause false, we make a blue vertex and join it up to the s relevant vertices. This ensures that they cannot *all* be chosen. (This is the purpose of the V_7 vertices.) We now turn to the formal details.

The red vertex set V_{red} of G is the union of the following sets of vertices:

$$V_1 = \{a[r_1, r_2] : 0 \leq r_1 \leq k-1, 0 \leq r_2 \leq n-1\},$$

$$V_2 = \{b[r_1, r_2, r_3] : 0 \leq r_1 \leq k-1, 0 \leq r_2 \leq n-1, 1 \leq r_3 \leq n-k+1\}.$$

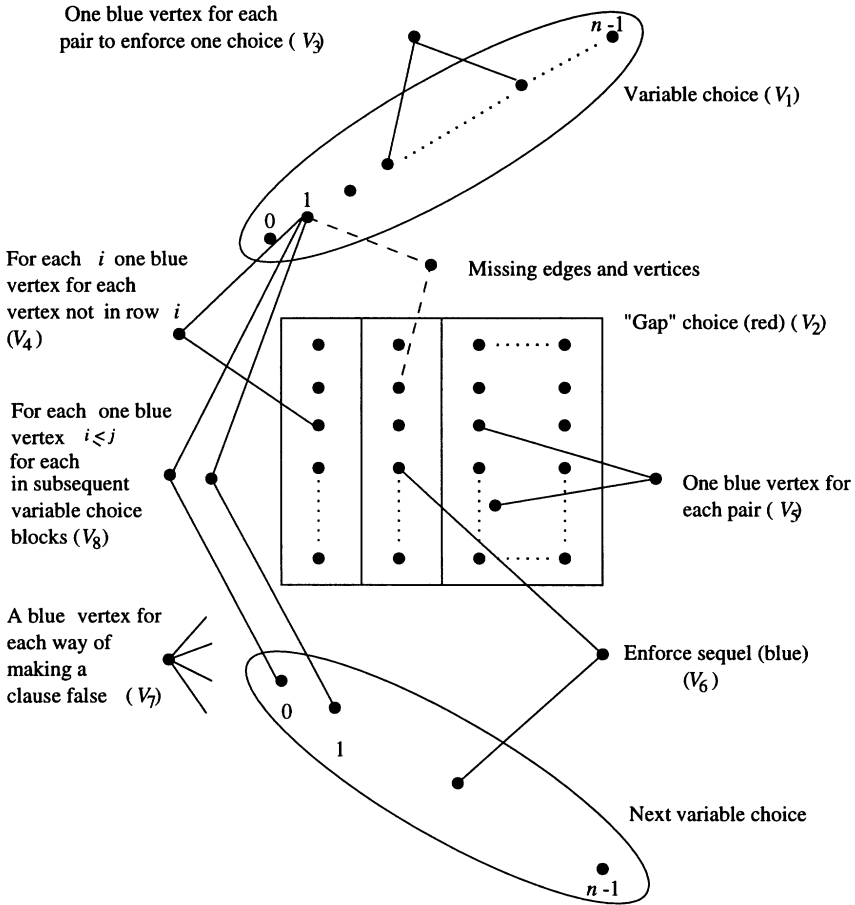


FIGURE 10.2. Gadget for RED/BLUE NONBLOCKER

The blue vertex set V_{blue} of G is the union of the following sets of vertices:

$$\begin{aligned}
 V_3 &= \{c[r_1, r_2, r'_2] : 0 \leq r_1 \leq k-1, 0 \leq r_2 < r'_2 \leq n-1\}, \\
 V_4 &= \{d[r_1, r_2, r'_2, r_3, r'_3] : 0 \leq r_1 \leq k-1, 0 \leq r_2, r'_2 \leq n-1, 0 \leq r_3, \\
 &\quad r'_3 \leq n-1 \text{ and either } r_2 \neq r'_2 \text{ or } r_3 \neq r'_3\}, \\
 V_5 &= \{e[r_1, r_2, r'_2, r_3] : 0 \leq r_1 \leq k-1, 0 \leq r_2, r'_2 \leq n-1, r_2 \neq r'_2, \\
 &\quad 1 \leq r_3 \leq n-k+1\}, \\
 V_6 &= \{f[r_1, r'_1, r_2, r_3] : 0 \leq r_1, r'_1 \leq k-1, 0 \leq r_2 \leq n-1, \\
 &\quad 1 \leq r_3 \leq n-k+1, r'_1 \neq r_2 + r_3\}, \\
 V_7 &= \{g[j, j'] : 1 \leq j \leq m, 1 \leq j' \leq m_j\}, \\
 V_8 &= \{h[r_1, r'_1, j, j'] : 0 \leq r_1 < r'_1 \leq k-1 \text{ and } j \geq j'\}.
 \end{aligned}$$

In the description of V_7 , the integers m_j are bounded by a polynomial in n and k whose degree is a function of s , which will be described below. Note that since s

is a fixed constant independent of k , this is allowed by our definition of reduction for parameterized problems.

For convenience, we distinguish the following sets of vertices:

$$A(r_1) = \{a[r_1, r_2] : 0 \leq r_2 \leq n - 1\},$$

$$B(r_1) = \{b[r_1, r_2, r_3] : 0 \leq r_2 \leq n - 1, 1 \leq r_3 \leq n - k + 1\},$$

$$B(r_1, r_2) = \{b[r_1, r_2, r_3] : 1 \leq r_3 \leq n - k + 1\}.$$

The edge set E of G is the union of the following sets of edges. In these descriptions, we implicitly quantify over all possible indices for the vertex sets $V_1, \dots, V_8$.

$$E_1 = \{a[r_1, q]c[r_1, r_2, r'_2] : q = r_2 \text{ or } q = r'_2\},$$

$$E_2 = \{b[r_1, q_2, q_3]d[r_1, r_2, r'_2, r_3, r'_3] : \text{either } (q_2 = r_2 \text{ and } q_3 = r_3) \\ \text{or } (q_2 = r'_2 \text{ and } q_3 = r'_3)\},$$

$$E_3 = \{a[r_1, r_2]e[r_1, r_2, q, q']\},$$

$$E_4 = \{b[r_1, q, q']e[r_1, r_2, q, q']\},$$

$$E_5 = \{b[r_1, r_2, r_3]f[r_1, r'_1, r_2, r_3]\},$$

$$E_6 = \{a[r_1 + 1 \bmod n, r'_1]f[r_1, r'_1, r_2, r_3]\},$$

$$E_7 = \{a[r_1, j]h[r_1, r'_1, j, j']\},$$

$$E_8 = \{a[r'_1, j']h[r_1, r'_1, j, j']\}.$$

We say that a red vertex $a[r_1, r_2]$ *represents the possibility* that the boolean variable x_{r_2} may evaluate to *true* (corresponding to the possibility that $a[r_1, r_2]$ may belong to a $2k$ -element nonblocking set V' in G). Because of the vertices V_8 and the edge sets E_7 and E_8 , any k nonblocking elements one from each V_1 block, $\{a[r_j, r_{ij}] : j = 0, \dots, k - 1\}$, $i_0 < i_1 < \dots < i_k \leq k - 1$. We say that a red vertex $b[r_1, r_2, r_3]$ *represents the possibility* that the boolean variables $x_{r_2+1}, \dots, x_{r_2+r_3-1}$ (with indices reduced mod n) may evaluate to *false*.

Suppose C is a clause of X having s literals. There are $O(n^{2s})$ distinct ways of choosing, for each literal $l \in C$, a single vertex representative of the possibility that $l = x_i$ may evaluate to *false*, in the case that l is a positive literal or in the case that l is a negative literal $l = \neg x_i$, a representative of the possibility that x_i may evaluate to *true*. For each clause C_j of X , $j = 1, \dots, m$, let $R(j, 1), R(j, 2), \dots, R(j, m_j)$ be an enumeration of the distinct possibilities for such a set of representatives. We have the additional sets of edges for the clause components of G :

$$E_9 = \{a[r_1, r_2]g[j, j'] : a[r_1, r_2] \in R(j, j')\},$$

$$E_{10} = \{b[r_1, r_2, r_3]g[j, j'] : b[r_1, r_2, r_3] \in R(j, j')\}.$$

Suppose X has a satisfying truth assignment τ of weight k , with variables $x_{i_0}, x_{i_1}, \dots, x_{i_{k-1}}$ assigned the value *true*. Suppose $i_0 < i_1 < \dots < i_{k-1}$. Let $d_r = i_{r+1 \bmod k} - i_r \pmod n$ for $r = 0, \dots, k - 1$. It is straightforward to verify that the set of $2k$ vertices

$$N = \{a[r, i_r] : 0 \leq r \leq k - 1\} \cup \{b[r, i_r, d_r] : 0 \leq r \leq k - 1\}$$

is a nonblocking set in G .

Conversely, suppose N is a $2k$ -element nonblocking set in G . It is straightforward to check that a truth assignment for X of weight k is described by setting those variables *true* for which a vertex representative of this possibility belongs to N , and by setting all other variables to *false*.

Note that the edges of the sets E_1 (E_2) which connect pairs of distinct vertices of $A(r_1)$ [$B(r_1)$] to blue vertices of degree 2 enforce that any $2k$ -element nonblocking set must contain exactly one vertex from each of the sets $A(0), B(0), A(1), B(1), \dots, A(k-1), B(k-1)$. The edges of E_3 and E_4 enforce (again, by connections to blue vertices of degree 2) that if a representative of the possibility that x_i evaluates to *true* is selected for a nonblocking set from $A(r_1)$, then a vertex in the i -th row of $B(r_1)$ must be selected as well, representing (consistently) the interval of variables set false (by increasing index because of the E_7 and E_8 edges) until the “next” variable selected to be *true*. The edges of E_5 and E_6 ensure consistency between the selection in $A(r_1)$ and the selection in $A(r_1 + 1 \bmod n)$. The edges of E_9 and E_{10} ensure that a consistent selection can be nonblocking if and only if it does not happen that there is a set of representatives for a clause witnessing that every literal in the clause evaluates to *false*. (There is a blue vertex for every such possible set of representatives.)

Theorem 10.6 provides the starting point for demonstrating the following dramatic collapse of the $W[1]$ stratification.

Theorem 10.7 $W[1] = W[1, 2]$.

Proof. It suffices to argue that for all $s \geq 2$, $\text{ANTIMONOTONE } W[1, s] = W[1, 2]$. The argument here consists of another change of variables. Let C be an antimonotone $W[1, s]$ circuit for which we wish to determine whether a weight k input vector is accepted. We show how to produce a circuit C' corresponding to an expression in conjunctive normal form with clause size 2, that accepts an input vector of weight

$$k' = k2^s + \sum_{i=2}^s \binom{k}{i}$$

if and only if C accepts an input vector of weight k . (The circuit C' will, in general, not be antimonotone, but this is immaterial by Theorem 10.6. Actually, in [163] we use another reduction that only needs $k' = k^{s+1} + \sum_{i=2}^s \binom{k}{i}$ and is hence polynomial in k for a fixed s . See Exercise 4.)

Let $x[j]$ for $j = 1, \dots, n$ be the input variables to C . The idea is to create new variables representing all possible sets of at most s and at least two of the variables $x[j]$. Let $A_1, \dots, A_p$ be an enumeration of all such subsets of the input variables $x[j]$ to C . The inputs to each *or* gate g in C (all negated, since C is antimonotone) are precisely the elements of some A_i . The new input corresponding to A_i represents that all of the variables whose negations are inputs to the gate g have the value *true*. Thus, in the construction of C' , the *or* gate g is replaced by the negation of the corresponding new “collective” input variable.

We introduce new input variables of the following kinds:

- (1) One new input variable $v[i]$ for each set A_i for $i = 1, \dots, p$ to be used as above.
- (2) For each $x[j]$, we introduce 2^k copies $x[j, 0], x[j, 1], x[j, 2], \dots, x[j, 2^k - 1]$.

In addition to replacing the *or* gates of C as described above, we add to the circuit additional *or* gates of fanin 2 that provide an enforcement mechanism for the change of variables. The necessary requirements can be easily expressed in conjunctive normal form with clause size 2, and thus can be incorporated into a $W[1, 2]$ circuit.

We require the following implications concerning the new variables:

- (1) The $n \cdot 2^k$ implications, for $j = 1, \dots, n$ and $r = 0, \dots, 2^k - 1$,

$$x[j, r] \Rightarrow x[j, r + 1 \pmod{2^k}].$$

- (2) For each containment $A_i \subseteq A_{i'}$, the implication

$$v[i'] \Rightarrow v[i].$$

- (3) For each membership $x[j] \in A_i$, the implication

$$v[i] \Rightarrow x[j, 0].$$

It may be seen that this transformation may increase the size of the circuit by a linear factor exponential in k . We make the following argument for the correctness of the transformation.

If C accepts a weight k input vector, then setting the corresponding copies $x[i, j]$ among the new input variables accordingly, together with appropriate settings for the new “collective” variables $v[i]$, yields a vector of weight k' that is accepted by C' .

For the other direction, suppose C' accepts a vector of weight k' . Because of the implications in (1) above, exactly k sets of copies of inputs to C must have value 1 in the accepted input vector (since there are 2^k copies in each set). Because of the implications described in (2) and (3) above, the variables $v[i]$ must have values in the accepted input vector compatible with the values of the sets of copies. By the construction of C' , this implies there is a weight k input vector accepted by C .

We have now done most of the work required to show that the following well-known problems are complete for $W[1]$:

INDEPENDENT SET

Instance: A graph $G = (V, E)$.

Parameter: A positive integer k .

Question: Is there a set $V' \subseteq V$ of cardinality k , such that for $\forall u, v \in V'$, $uv \notin E$?

CLIQUE

Instance: A graph $G = (V, E)$.

Parameter: A positive integer k .

Question: Is there a set of k vertices $V' \subseteq V$ that forms a complete subgraph of G (that is, a clique of size k)?

Theorem 10.8

- (i) **INDEPENDENT SET** is complete for $W[1]$.
- (ii) **CLIQUE** is complete for $W[1]$.

Proof. It is easy to observe that **INDEPENDENT SET** belongs to $W[1]$. By Theorems 10.6 and 10.7, it is enough to argue that **INDEPENDENT SET** is hard for **ANTIMONOTONE** $W[1, 2]$. Given a boolean expression X in conjunctive normal form (product of sums) with clause size 2 and all literals negated, we may form a graph G_X with one vertex for each variable of X and having an edge between each pair of vertices corresponding to variables in a clause. The graph G_X has an independent set of size k if and only if X has a weight k truth assignment.

(ii) This follows immediately by considering the complement of a given graph. The complement has an independent set of size k if and only if the graph has a clique of size k .

Finally, we are in a position to establish our first analog of Cook's theorem, which we restate for the reader's convenience. In fact, we can now state an extended form.

Theorem 10.9 (Parameterized Cook's Theorem) *The following are complete for $W[1]$:*

- (i) (with Liming Cai and J. Chen) **SHORT TURING MACHINE ACCEPTANCE**.
- (ii) **WEIGHTED 2-SATISFIABILITY**.
- (iii) s -**RED/BLUE NONBLOCKER** for $s \geq 2$.
- (iv) **INDEPENDENT SET**.
- (v) **CLIQUE**.

Proof. We need only prove that **SHORT TURING MACHINE ACCEPTANCE** is $W[1]$ -complete to finish the result. To show hardness for $W[1]$, we reduce from **CLIQUE**. $W[1]$ -hardness will then follow by Theorem 10.8(ii). Let $G = (V, E)$ be a graph for which we wish to determine whether it contains a k -clique. We have shown how to construct a nondeterministic Turing machine M that can reach an accept state in $k' = f(k)$ moves if and only if G contains a k -clique. The Turing machine M is designed so that any accepting computation consists of two phases. In the first phase, M nondeterministically writes k symbols representing vertices of G in the first k tape squares. (There are enough symbols so that each vertex of G is represented by a symbol.) The second phase consists of making $\binom{k}{2}$ scans of the k tape squares, each scan devoted to checking, for a pair of positions i and j , that the vertices represented by the symbols in these positions are adjacent in G . Each such pass can be accomplished by employing $O(|V|)$ states in M dedicated to the ij -th scan.

In order to show membership in $W[1]$, it suffices to show how the **SHORT TURING MACHINE ACCEPTANCE** problem for a Turing machine $M = (\Sigma, Q, q_0, \delta, F)$ and positive integer k can be translated into one about whether a circuit C accepts a weight k' input vector, where C has depth bounded by some t (independent of k and the Turing machine M) and has only a single large (output) *and* gate, with all

other gates small. We arrange the circuit so that the k' inputs to be chosen to be set to 1 in a weight k' input vector represent the various data: (1) the i -th transition of M , for $i = 1, \dots, k$, (2) the head position at time i , (3) the state of M at time i , and (4) the symbol in square j at time i for $1 \leq i, j \leq k$. Thus, we may take $k' = k^2 + 3k$. In order to force exactly one input to be set equal to 1 among a pool of input variables (for representing one of the above choices), we can add to the circuit for each such pool of input variables, and for each pair of variables x and y in the pool, a small “not both” circuit representing $(\neg x \vee \neg y)$. It might seem that we must also enforce (e.g., with a large *or* gate) the condition “at least one variable in each such pool is set true”—but this is actually unnecessary, since in the presence of the “not both” conditions on each pair of input variables in each pool, an accepted weight k' input vector *must have* exactly one variable set true in each of the k' pools. Let n denote the total number of input variables in this construction. We have $n = O(k|\delta| + k^2 + k|Q| + k^2|\Sigma|)$ in any case.

The remainder of the circuit encodes various checks on the consistency of the above choices. These consistency checks conjunctively determine whether the choices represent an accepting k -step computation by M , much as in the proof of Cook's theorem. These consistency checks can be implemented so that each involves only a bounded number b of the input variables. For example, we will want to enforce that if five variables are set true indicating particular values of the tape head position at time $i + 1$ and the head position, state, scanned symbol, and machine transition at time i , then the values are consistent with δ . Thus, $O(n^5)$ small “checking” circuits of bounded depth are sufficient to make these consistency checks; in general, we will have $O(n^b)$ “checking” circuits for consistency checks involving b values. All of the small “not both” and “checking” circuits feed into the single large output *and* gate of C . The formal description of all this is laborious but straightforward.

We remark that Theorem 10.3 depends crucially on there being no bound on the size of the Turing machine alphabets in the definition of the problem. If we restrict **SHORT TURING MACHINE ACCEPTANCE** to Turing machines with $|\Sigma|$ bounded by some constant b , then the number of configurations is bounded by $b^k|Q|k$ and the problem becomes fixed-parameter tractable (see Exercise 1).

Historical Notes

A number of authors have looked at (machine dependent) complexity classes within P . These include the Kintala-Fischer β -hierarchy [297], the work of Geske [243], and the work of Buss and Goldsmith [101]. These and similar investigations were mainly technical structural investigations and looked at problems for a *single* k rather than examining the asymptotic behavior of the parameterized problem as $k \rightarrow \infty$. This forces one to look at reductions such as “quasi-linear time reducibility” and the like (see, e.g., Naik, Regan and Sivakumar [359] and Gurevich-Shelah [253]). We will discuss some further connections in Part III, Chapter 17.

All of these investigations could be looked at as “miniaturizations” of classical complexity.

As best the authors can tell, the first author to note that the asymptotic behavior of fixed-parameter languages might be of interest was Ken Regan in [382]. Regan did so in some comments where he suggested that it might be important that **DOMINATING SET** might require time $\Omega(n^{k+1})$. The first article to attempt to study the asymptotic complexity of parameterized problems and the issue of n^α vs $n^{f(k)}$ with $f(k) \rightarrow \infty$ was by Abrahamson, Ellis, Fellows and Mata [5], with some of those results recast in Buss-Goldsmith [101]. In that article, the objects were P -checkable, P -indexed relations, called (polynomial time) *generator tester* pairs. For each k , one needed to be able to generate a P -time list of potential candidates for solutions that were easy to test. The actual definition is very involved, but the following example gives the flavor. If we consider the problem **VERTEX COVER**, then for an instance G and a parameter k , the potential witnesses would be pairs consisting of G and a vertex cover V with the index of $V \leq j = \binom{n}{0} + \cdots + \binom{n}{k}$. The ideas only seemed to apply to relations in NP . Although there is a notion of parameterized tractability in [5], it is roughly equivalent to our notion of nonuniform fixed-parameter tractability and hence suffers from the problem that the tractable classes can be nonrecursive. The real problems with the article [5] are that (1) the notion of reducibility, which is defined on relations rather than parameterized languages, is rather unnatural and very unwieldy and (2) the notion of intractability is that of (essentially) being P -complete (or “dual P -complete”) under *logspace* reducibility “by the slice”; that is, in [5], problems are “PGT-complete” (in their notation) only when for each k they are more or less P -complete. Of course, this means that the [5] ideas cannot address things such as **DOMINATING SET** and **INDEPENDENT SET**, nor apparently anything in the $W[t]$ classes of the next chapters. As we will see in Chapter 14 the [5] results can be easily placed in our setup, as they give $W[P]$ -completeness results.

In this context, we see our major contributions as being the identification of the correct notions of reducibility, identifying the correct setting for the study of parameterized intractability, and, finally, identifying some “good” problems with which to measure hardness. The number of problems that have now been identified as $W[1]$ -hard, and hence apparently intractable, would seem to support our claims.

All of the results of this chapter are taken from Downey-Fellows [170], except for the reduction from **CLIQUE** to **SHORT TURING MACHINE ACCEPTANCE** and the proof that **SHORT TURING MACHINE ACCEPTANCE** is in $W[1]$. These last two results are due to Liming, Cai, Chen, Downey, Fellows [109]. The reduction for **RED/BLUE NONBLOCKER** from Downey-Fellows [170] contained a flaw that was spotted by Alexander Vardy and is corrected here.

Exercises 10.0

- 10.0.1.** Prove that the problem **RESTRICTED SHORT TURING MACHINE ACCEPTANCE** is strongly *FPT*. This is the restriction of **SHORT TURING**

MACHINE ACCEPTANCE to machines with $|\Sigma|$ (the size of the alphabet) bounded by some fixed constant b .

10.0.2. Prove that the **INDEPENDENT SET** is *FPT* when restricted to:

- (i) graphs with fixed maximum degree m ,
- (ii) planar graphs.

10.0.3. (Some $W[1]$ -complete problems related to algebraic structures.)
Prove that the following problems are $W[1]$ complete.

(i) **SEMIGROUP EMBEDDING**

Input: A semigroup $S = (S, \cdot)$.
Parameter: A semigroup $G = (G, \times)$.
Question: Is $(G, \times)$ isomorphic to a subsemigroup of $(S, \cdot)$?

(ii) **SEMILATTICE EMBEDDING**

Input: An (upper) semilattice $(X, \leq_X)$.
Parameter: A semilattice $(Y, \leq_Y)$.
Question: Is $(Y, \leq_Y)$ a subsemilattice of $(X, \leq_X)$?

(iii) **BIPARTITE GRAPH EMBEDDING**

Input: A bipartite graph G .
Parameter: A bipartite graph H .
Question: Can H be embedded into G ?

[Hint. (i) First, we need to observe that **SEMIGROUP EMBEDDING** is in $W[1]$. Let $G = \{g_1, \dots, g_k\}$ be the parameter semigroup, $G = (G, \times)$ with the appropriate multiplication table $M(G)$, and let S be the given semigroup $S = (S, \cdot)$ with $S = \{x_1, \dots, x_n\}$. The weft 1 circuit consists of kn input variables labeled $\{(x_i, g_j) := 1, \dots, n, j = 1, \dots, k\}$. The circuit needs a small *or* gate for each i, j , and p for $j \neq p$ to express that if (x_i, g_j) , then $\neg(x_i, g_p)$. This needs $\leq k^2n$ small *or* gates. We also need $\leq k^3n^2$ many small *or* and *and* gates to express the fact that the tables are compatible; that is, we express the implication “if (s_i, g_j) and (s_p, g_q) , then (s_i, g_m) ” where $s_i \cdot s_p = s_i$ and $g_j \times g_q = g_m$ is read off from, respectively, $M(G)$ and $M(S)$. All of these gates are then joined by a large *and* gate. Note that if the circuit has a weight k accepting input, then exactly one g_j can be chosen for an S_i , and hence G_j is a subsemigroup of S_i . (This technique works for many embedding results, and, in particular it works for (ii) and (iii).)

As with all of (i) to (iii), the proof that they are $W[1]$ -hard uses reductions from the literature, and, in particular Power [377] and Booth [92]. We reduce **CLIQUE** to **SEMIGROUP EMBEDDING**. Given a graph $G = (V, E)$ and parameter k , we use Booth's semigroup defined as $S_G = V \cup E \cup \{0\}$, where

$$\begin{aligned} y \cdot x &= x \cdot y = x \text{ if } x = y \\ &= y \text{ if } x \in y \text{ and } x \in V \text{ and } y \in E \\ &= \{x, y\} \text{ if } \{x, y\} \in E \\ &= 0 \text{ otherwise.} \end{aligned}$$

One must observe that if G contains a k clique with $\varphi : K_k \rightarrow G$ then $\tilde{\varphi} : K_k \rightarrow S_G$ defined via $0 \rightarrow 0$ and $x \rightarrow \varphi(x)$ for $x \in V \cup E$ induces an embedding of S_{K_k} into S_G . Similarly, if $\varphi : K_k \rightarrow S_G$ and $k \geq 3$, then since 0_{S_G} is the only element y of S_G for which there are at least 4 elements x with $x \times y = y$, we must have that φ takes 0 to 0. φ maps edges to edges since if y is an edge then there exist at least 3 elements x with $x \cdot y = y$ yet if y is a vertex there is exactly one such x , and $\varphi(x) \neq 0$ by the facts that $0 \rightarrow 0$ and φ is 1-1. By the homomorphism property, and the definition of φ , φ takes vertices to vertices and hence induces an injection from K_k to G . So G contains a k -clique.

Cases (ii) and (iii) are rather similar, but we give some details since Power [377] is not published. For **SEMILATTICE EMBEDDING**, on a derived Booth semigroup S_H , if we define $\leq_H$ on S_H via $a \leq_H b$ iff $a \cdot b$, then $(S_H, \leq_H)$ is a lower semilattice (the meet of a and b is $a \cdot b$ and since all elements of S_G are idempotent, this follows by some standard semigroup theory e.g. Howie [272]). It is straightforward to show that S_{K_k} is embeddable into S_G if $(S_{K_k}, \leq_{K_k})$ is a sublattice of $(S_G, \leq_G)$.

Finally we consider (iii). To see that **BIPARTITE GRAPH EMBEDDING** is $W[1]$ -complete, the Hasse Diagram of $(S_{K_k}, \leq_{K_k})$ and $(S_G, \leq_G)$ are both bipartite graphs and one can simply argue that the former is embeddable in the latter iff the first poset is embeddable in the latter.]

10.0.4. Prove that **MONOTONE** $W[1] \subseteq FPT$. [Hint: First prove that if C is a $W[1]$ circuit of depth d then C has $\leq 2^d$ many large gates. (That is, assuming that “small” means fanin 2.) Now, normalize the circuit to a large *and* of mostly small *or*’s but with possibly $\leq 2^d$ large *or*’s. Now argue by the method of search trees. See Corollary 18.2]

10.0.5. Using the classical proof of Cook’s Theorem as a model, carefully complete the details of the proof of Theorem 10.3 (i).

10.0.6. Prove that the following problem **IRREDUNDANT SET** is in $W[1]$. (Downey, Fellows and Raman [173] have proven that **IRREDUNDANT SET** is $W[1]$ -hard.) The problem **MAXIMAL IRREDUNDANT SET** (see Exercise 9.0.1) is known to be $W[2]$ -hard.)

IRREDUNDANT SET

Instance: A graph $G = (V, E)$, a positive integer k .

Parameter: A positive integer k .

Question: Is there a set $V' \subseteq V$ of cardinality k having the property that each vertex $u \in V'$ has a private neighbor? (A *private neighbor* of a vertex $u \in V'$ is a vertex u' (possibly $u' = u$) with the property that for every vertex $v \in V'$, $u \neq v$, $u' \notin N[v]$.)

[Hint: Let $G = (V, E)$ be a graph for which we wish to determine if G has a k -element irredundant set. We construct the circuit C corresponding to the following boolean expression E . The variables of E are

$$\{p[i, x, y] : 1 \leq i \leq k, x, y \in V\} \cup \{c[i, x] : 1 \leq i \leq k, x \in V\}$$

The expression E has the clauses:

- (1) $(\neg p[i, x, y] + \neg p[i, x', y'])$ for $1 \leq i \leq k$ and either $x \neq x'$ or $y \neq y'$, $x, x', y, y' \in V$.
- (2) $(\neg c[i, x] + \neg c[i, x'])$ for $1 \leq i \leq k$ and $x \neq x', x, x' \in V$.
- (3) $(c[i, x] + \neg p[i, x, y])$ for $1 \leq i \leq k$ and $x, y \in V$.
- (4) $(\neg c[j, u] + p[i, x, y])$ for $1 \leq i, j \leq k, i \neq j$ and $u \in N[y]$.

Now argue that the circuit C accepts a weight $2k$ input vector if and only if G has a k -element irredundant set.]

11

Some Other $W[1]$ -Hardness Results

In this chapter, we will explore some further $W[1]$ -hardness and completeness results. These examples are taken from logic, formal language theory, and from combinatorial problems arising from molecular biology. We also look at further applications of the theory to arenas such as algebra and number theory in the Exercises, where problems such as the following are proven to be $W[1]$ -hard (and the first two are in $W[2]$):

PERFECT CODE

Instance: A graph $G = (V, E)$.

Parameter: A positive integer k .

Question: Does G have a k -element perfect code? A *perfect code* is a set of vertices $V' \subseteq V$ with the property that for each vertex $v \in V$, there is precisely one vertex in $N[v] \cap V'$.

SIZED SUBSET SUM

Input: A set $X = \{x_1, \dots, x_n\}$ of integers and a positive integer S .

Parameter: A positive integer k .

Question: Do there exist k elements of X that sum to S ?

PERMUTATION GROUP FACTORIZATION

Instance: A set A of permutations $A \subseteq S_n$, and $x \in S_n$.

Parameter: A positive integer k .

Question: Does x have a factorization of length k over A ?

All of these areas are very good arenas for the study of parameterized complexity since one is often concerned with small parameters but large objects. We begin by looking at a problem arising from logic, or more properly, computational learning theory. The *Vapnik–Chervonenkis Dimension* is the central parameter in computational learning theory both in the PAC model and in the Angluin-type exact learning model. (See Exercise 4.) It has proved to be especially important in the determination of lower bounds in computational learning theory.

Definition 11.1 (Vapnik–Chervonenkis Dimension) The *Vapnik–Chervonenkis dimension* (VC dimension) of a family of subsets F of a base set X is the maximum cardinality of a set $S \subseteq X$ such that for every subset $S' \subseteq S$, $\exists Y \in F$ such that $S \cap Y = S'$. In general, we say that such a subset S' of S is *generated* in S by F . The VC dimension of F is thus the largest cardinality of $S \subseteq X$ such that every subset of S is generated by F .

We can naturally associate a (parameterized) language with the concept of the VC dimension.

VAPNIK-CHERVONENKIS DIMENSION

Instance: A family of subsets F of a base set X .

Parameter: A positive integer k .

Question: Is the VC dimension of F at least k ?

The VC dimension provides a very nice concrete example of the situation where problems with very different classical complexities can have the same parameterized ones. The VC dimension of a family of sets $\mathcal{F}$ over a base set X of cardinality n can be shown to be at most $\log n$. Consequently, the above problem is unlikely to be NP -complete [368]. It is, in fact, LOGNP -complete and hence can only be NP -complete in the unlikely event that $NP \subseteq \text{DTIME}(n^{\log n})$. However, the following theorem shows that **VAPNIK-CHERVONENKIS DIMENSION** is $W[1]$ -complete.¹ *In the following theorem, both membership and hardness for the parameterized complexity class $W[1]$ involve reductions that are exponential in the parameter k .*

Theorem 11.2 ([169], [163]) **VAPNIK-CHERVONENKIS DIMENSION** is complete for $W[1]$.

¹Notice that parameterized complexity analysis thus provides an indirect method of demonstrating *practical* intractability of problems either not likely to be NP -complete, or problems whose complexity is currently unclassified. One example of this is the proof by Bodlaender et al. [79] that **PRECEDENCE CONSTRAINED SCHEDULING** is $W[1]$ -hard and hence will not have practical algorithms for reasonable k . This is one of the open problems from Garey and Johnson [239]. A really nice candidate for such analysis is **GRAPH ISOMORPHISM**.

Proof. *Membership in $W[1]$* (Downey and Fellows [169]).

Let $\mathcal{F} \subseteq 2^U$ denote the family of sets under consideration. Suppose $\mathcal{F} = \{X_j : 1 \leq j \leq m\}$ and $U = \{1, \dots, n\}$. It suffices to show that in time $f(k) \cdot (mn)^\alpha$, we can produce a product-of-sums boolean expression E in which the clauses have size bounded by some constant, and such that E has a satisfying truth assignment of weight $k' = g(k)$ if and only if the VC dimension of $\mathcal{F}$ is at least k .

The set of variables for E is $V = V_1 \cup V_2$, where
 $V_1 = \{a[i, j] : 1 \leq i \leq 2^k, 1 \leq j \leq m\}$,
 $V_2 = \{b[r, s] : 1 \leq r \leq k, 1 \leq s \leq n\}$.

The variables of V_1 serve to indicate which sets in $\mathcal{F}$ witness the shattering of the k -element subset of U indicated by the variables of V_2 . Let γ be a fixed 1:1 correspondence between the integers i in the range $1 \leq i \leq 2^k$ and the length k 0-1 vectors, and write $\gamma_i(l) \in \{0, 1\}$ to indicate the value of the l -th component of the vector associated to i .

We will take $k' = k + 2^k$ and $E = E_1 \cdot E_2$ as follows.

E_1 is a product of small clauses that enforces the conditions:

(1) For each index value of i (r) in the definition of V_1 (V_2), at most one variable of V_1 (V_2) is set to *true*.

(2) If $b[i', j]$ and $b[i', j']$ are set to *true*, with $i < i'$, then $j < j'$.

This can be accomplished with clauses of size 2.

Note that any satisfying truth assignment to E_1 of weight $k + 2^k$ must set *exactly* one variable of V_1 (V_2) *true* for each index value of i (r).

E_2 is the product of clauses expressing the implications: $a[i, j] \Rightarrow \neg b[r, s]$ for all *incompatible* pairs of indices (i, j) and (r, s) , where such a pair is defined to be *incompatible* if and only if either (1) $\gamma_i(r) = 1$ and $s \notin X_j$ or (2) $\gamma_i(r) = 0$ and $s \in X_j$.

The verification that this works correctly is straightforward.

$W[1]$ -Hardness (Downey, Evans, and Fellows[163]).

We use a reduction from CLIQUE. Given a graph $G = (V, E)$ and a positive integer k , we describe how to compute a family of sets F over a base set X , so that F has VC dimension k if and only if G has a k -clique. The cardinality of the family F that we will describe is $O(k^2 n^2 + 2^k)$, and the size of the base set X is kn , where n is the order of G . For convenience, we will assume that $V = \{1, \dots, n\}$. We write $[m]$ to denote the set $\{1, \dots, m\}$.

The base set X is simply

$$X = \{(u, i) : u \in V, i \in [k]\}.$$

The family F consists of four subfamilies, $F = F_0 \cup F_1 \cup F_2 \cup F_3$ which are described as follows. (These correspond, roughly, to the cardinality of the sets in the subfamilies.)

$$F_0 = \{\emptyset\},$$

$$F_1 = \{(u, i) : u \in V, i \in [k]\},$$

$$F_2 = \{(u, i), (v, j) : uv \in E, i, j \in [k]\},$$

$$F_3 = \{(u, i) : i \in S\} : S \subseteq [k], \#(S) \geq 3\}.$$

To see this that this construction works, let C be the clique in G and let f be any 1:1 map from C to $\{1, \dots, k\}$. Consider the set $S \subseteq X$ of cardinality k :

$$S = \{(u, f(u)) : u \in C\}.$$

If $S' \subseteq S$ has cardinality at least 3, then it is generated by the corresponding set in F_3 . It is straightforward to verify that subsets of S of cardinality smaller than 3 are generated by $F_0 \cup F_1 \cup F_2$.

Conversely, suppose S is a k -element subset of X , every subset of which is generated by F . For each subset $S' \subseteq S$, choose a witness $W \in F$ with $W \cap S = S'$. If S' has cardinality at least 3, then its witness must be chosen from F_3 . But this implies that every set in F_3 must serve as a witness for some $S' \subseteq S$ of cardinality at least 3.

The witnesses for sets $S' \subseteq S$ of cardinality 2 must therefore belong to F_2 . We cannot have both (u, i) and (u, j) in S for $i \neq j$, else there is no witness possible for the 2-element set consisting of these (by the definition of F_2). Consequently, S must range over k different vertex indices. The fact that there are witnesses for all of the 2-element subsets implies that there is a corresponding k -clique in G .

Turning from computational learning theory, we look at some problems from logic and formal language theory. Specifically, we consider the following two problems.

SHORT DERIVATION (FOR UNRESTRICTED GRAMMARS)

Instance: The two pieces of information:

- (1) A grammar $G = (N, \Sigma, \Pi, S)$, where N is a finite set of "nonterminal symbols," Σ is a finite set of "terminal symbols," Π is a finite set of "production rules" of the form $(\alpha \rightarrow \beta)$ with $\alpha, \beta \in (N \cup \Sigma)^*$, and $S \in N$ is the "start symbol."
- (2) A word $x \in \Sigma^*$.

Parameter: A positive integer k .

Question: Is there a G -derivation of x of length k ? That is, is there a sequence of words $x_0, \dots, x_k$ with $x_i \in (N \cup \Sigma)^*$ for $i = 0, \dots, k$, that satisfies the requirements:

- (1) $x_0 = S$,
- (2) $x_k = x$, and
- (3) for each $i = 1, \dots, k$ there is a production rule $\pi_i = (\beta \rightarrow \gamma) \in \Pi$ such that $x_{i-1} = \alpha\beta\delta$ and $x_i = \alpha\gamma\delta$?

We note in passing that if the above problem is restricted to grammars having production rules in which the left-hand side always consists of a single nonterminal symbol (termed a *context-free grammar*), then it is fixed-parameter tractable (Exercise 1).

SHORT POST CORRESPONDENCE

Instance: A Post system Π where a *Post system* consists of a finite alphabet Σ and two sequences $\alpha = (\alpha_1, \dots, \alpha_n)$ and $\beta = (\beta_1, \dots, \beta_n)$.

Parameter: A positive integer k .

Question: Is there a length k solution for Π ? That is, is there a sequence of integers $i_1, \dots, i_k$ (not necessarily distinct) such that

$$\alpha_{i_1} \cdots \alpha_{i_k} = \beta_{i_1} \cdots \beta_{i_k}?$$

Lemma 11.3 (Cai, Chen, Downey and Fellows) **CLIQUE reduces to SHORT DERIVATION.**

Proof. Let $G = (V, E)$ be a simple graph, and k a positive integer. We describe a grammar $\mathcal{G} = (N, \Sigma, \Gamma, S)$, a word $x \in \Sigma^*$, and a positive integer $k' = f(k)$ such that the start symbol S derives x in k' steps if and only if G has a k -clique.

The terminal symbols of the grammar are

$$\Sigma = \{\#\} \cup \{u : u \in V\} \cup \{l\}.$$

The nonterminal symbols are

$$N = N_1 \cup N_2 \cup N_3 \cup N_4.$$

where

$$\begin{aligned} N_1 &= \{S, S', M, W, Z\}, \\ N_2 &= \{X_i : 1 \leq i \leq k\}, \\ N_3 &= \{R[i, j] : 1 \leq i \leq k, 0 \leq j \leq k-1\}, \\ N_4 &= \left\{ T[i, u, j] : 1 \leq i \leq k, u \in V, 1 \leq j \leq \binom{k}{2} \right\}. \end{aligned}$$

The start symbol is S .

The target string $x \in \Sigma^*$ is

$$x = \#^{3\binom{k}{2}} l^{k+2}.$$

where the exponents indicate symbol repetition.

The set of productions Γ is

$$\Gamma = \Gamma_1 \cup \Gamma_2 \cup \dots \cup \Gamma_{12}.$$

In the formal description of some of these production rule sets (in particular, Γ_6 , Γ_7 , and Γ_8), we make reference to the following sets of indices. Let σ denote the sequence of symbols in the set $S = \{\sigma[i, j] : 1 \leq i < j \leq k\}$ in increasing lexicographic order. Thus, the length of the sequence σ is $\binom{k}{2}$. Let s_r denote the

r -th symbol of the sequence σ . For $i = 1, \dots, k$, define

$$\begin{aligned} J_i &= \{r : s_r = \sigma[i, j], 1 \leq j \leq k\}, \\ J'_i &= \{r : s_r = \sigma[j, i], 1 \leq j \leq k\}, \\ J''_i &= \left\{1, \dots, \binom{k}{2}\right\} - (J_i \cup J'_i). \end{aligned}$$

Thus, for $k = 4$, we have $\sigma = \sigma[1, 2]\sigma[1, 3]\sigma[1, 4]\sigma[2, 3]\sigma[2, 4]\sigma[3, 4]$, $J_1 = \{1, 2, 3\}$, $J'_1 = \emptyset$, $J''_1 = \{4, 5, 6\}$, $J_2 = \{4, 5\}$, $J'_2 = \{1\}$, and $J''_2 = \{2, 3, 6\}$.

Intuitive Overview. Let S denote the starting symbol for the grammar. The initial production rule (the only one that can be applied) is

$$S \rightarrow S'R[k, 1] \cdots R[1, 1]\#X[1]X[2]\#X[1]X[3]\# \cdots \#X[k-1]X[k]Z.$$

There are productions by which the pairs of symbols $X[i]X[j]$ may produce a pair of terminal symbols representing an edge of the graph; this represents in some sense a “guess” about a k -clique in G . Note that there are $\binom{k}{2}$ pairs corresponding to the $\binom{k}{2}$ edges of a k -clique. For such a guess to be consistent, it must be verified that each occurrence of the symbol $X[i]$ produces the same guessed vertex (as the endpoint of a guessed edge). The R and T symbols make this consistency check by “commuting across” the intermediate string. The last index of these symbols functions as a counter that keeps track of which edge of the potential clique is being checked. The $R[i, *]$ (“read”) symbols commute until the first occurrence of the i -th vertex (produced by $X[i]$) of the potential k -clique is encountered. At this point, the symbol $R[i, *]$ is commuted past the edge of this first occurrence, but transformed into a T (“test”) symbol that records the identity of the i -th vertex (as the second component in the indexing of the symbol). As this symbol continues to commute across the portion of the string corresponding to the guessed edges, there are three possibilities that arise with respect to next guessed edge of the commute: (1) the first endpoint of the guessed edge should correspond to the recorded vertex of the symbol, (2) the second endpoint of the guessed edge should correspond to the recorded vertex of the symbol, or (3) neither endpoint should correspond to the recorded vertex of the symbol. The positions in the sequence of $\binom{k}{2}$ guessed edges corresponding to these three possibilities are indexed by J_i , J'_i , and J''_i , respectively, and the corresponding sets of production rules are Γ_6 , Γ_7 , and Γ_8 . The “read and test” nonterminal symbols that implement the consistency checking can only be eliminated from the string by successfully commuting to the Z symbol on the right end of the string. In the final phase of a successful derivation of the target string x , the symbol S' must also commute across in order to be eliminated at the right end. As it commutes, it replaces the guessed vertices with the place-holding symbol $\#$.

The details of the production rules of the grammar are as follows:

$$\begin{aligned} \Gamma_1 &= \{S \rightarrow S'R[k, 1] \cdots R[1, 1]\#X_1X_2\#X_1X_3\# \cdots \\ &\quad \cdots \#X_1X_k\#X_2X_3\#X_2X_4\# \cdots \#X_{k-1}X_kZ\}, \\ \Gamma_2 &= \{X_iX_j \rightarrow uv : 1 \leq i < j \leq k, uv \in E\}, \end{aligned}$$

$$\begin{aligned}
\Gamma_3 &= \{R[1, 1]\#uv \rightarrow \#uvT[1, u, 2] : uv \in E\}, \\
\Gamma_4 &= \{R[i, j]\#uv \rightarrow \#uvR[i, j + 1] : 2 \leq i \leq k, 1 \leq j \leq i - 2, uv \in E\}, \\
\Gamma_5 &= \{R[i, j]\#uv \rightarrow \#uvT[i, v, j + 1] : 2 \leq i \leq k, j = i - 1, uv \in E\}, \\
\Gamma_6 &= \{T[i, u, j]\#uv \rightarrow \#uvT[i, u, j + 1] : 1 \leq i \leq k, j \in J_i, uv \in E\}, \\
\Gamma_7 &= \{T[i, v, j]\#uv \rightarrow \#uvT[i, v, j + 1] : 1 \leq i \leq k, j \in J'_i, uv \in E\}, \\
\Gamma_8 &= \{T[i, u, j]\#xy \rightarrow \#xyT[i, u, j + 1] : 1 \leq i \leq k, j \in J''_i\}, \\
\Gamma_9 &= \left\{ T \left[i, u, \binom{k}{2} + 1 \right] Z \rightarrow ZM : 1 \leq i \leq k - 1, u \in V \right\}, \\
\Gamma_{10} &= \left\{ T \left[k, u, \binom{k}{2} + 1 \right] Z \rightarrow WM \right\}, \\
\Gamma_{11} &= \{S'\#uv \rightarrow \###S' : uv \in E\}, \\
\Gamma_{12} &= \{S'WM^k \rightarrow l^{k+2}\}.
\end{aligned}$$

It remains only to specify the number of steps for the derivation:

$$k' = 1 + \binom{k}{2} + (k + 1) \left[\binom{k}{2} + 1 \right].$$

Half of the correctness argument for the reduction is straightforward. If G has a k -clique, then a derivation of x of length k' can be written down by following the sketch above, noting that all of the necessary means for commuting symbols from left to right are available. For the other half of the argument, it is easy to see that the necessary first step of the derivation (since it is the only one possible) creates a situation where the R symbols (or the T symbols into which they are transformed) must be commuted across the string. The key point is that this is possible only if the guessed edges are consistent in providing evidence of a k -clique in G .

Corollary 11.4 **SHORT DERIVATION** is $W[1]$ -hard.

Proof. By Theorem 10.9, **CLIQUE** is $W[1]$ -complete.

Recall that a *context-sensitive* grammar is one where the productions $\alpha \rightarrow \beta$ satisfy the length restriction $|\alpha| \leq |\beta|$ and $\alpha, \beta \neq \epsilon$. Our argument for Lemma 11.3 actually shows that **SHORT DERIVATION** remains $W[1]$ -hard for this special case. We next establish membership in $W[1]$.

Lemma 11.5 (Liming Cai, Chen, Downey, and Fellows) **SHORT DERIVATION** is in $W[1]$.

Proof. To show that **SHORT DERIVATION** belongs to $W[1]$, we will use Theorem 10.3; that is, we will describe (at a high level) how to reduce the problem of determining whether a word x can be generated in k steps from a given (unrestricted) grammar $G = (N, \Sigma, \Pi, S)$, to the problem of determining whether a nondeterministic Turing machine can reach a halting configuration in $f(k)$ steps.

The computation of the Turing machine is organized in two phases. The first phase nondeterministically guesses a description of the k -step derivation of x . [This

description can be recorded by $O(k^2)$ symbols written in as many tape squares; we describe how this can be accomplished below.] The second phase consists of a deterministic computation that checks whether the guessed derivation of x is valid.

Suppose the sequence $x_0, \dots, x_k$ corresponds to a k -step derivation of x in the grammar G in the sense that (1) $x_0 = S$ is the start symbol of G , (2) $x_k = x$, and (3) for each $i, i = 1, \dots, k$, x_i can be obtained from x_{i-1} by the application of the production rule $\alpha_i \rightarrow \beta_i$. For any word $w \in (N \cup \Sigma)^*$, let $w[j]$ denote the j -th symbol of w , and for $s \leq t$ let $w[s, t]$ denote the substring of w consisting of the symbols $w[s] \cdots w[t]$. Thus, for the production rule $\alpha_r \rightarrow \beta_r$, we write $\beta_r[s, t]$ to denote the substring of the yield β_r consisting of the s -th through t -th symbols.

We employ an alphabet with a distinct symbol for each possible substring $\beta_r[s, t]$ of the yield of a production rule. Note that the number of symbols required is bounded by a polynomial in the size of the description of the grammar G . Let Γ denote this alphabet.

The description of the k -step derivation of x for $i = 1, \dots, k$ consists of the following:

(1) A factorization of x_i

$$x_i = \beta_{i_1}[s_{i_1}, t_{i_1}] \cdots \beta_{i_m}[s_{i_m}, t_{i_m}]$$

represented by symbols of Γ .

(2) The production rule $\alpha_i \rightarrow \beta_i$ that yields x_i from x_{i-1} .

(3) The substring of symbols of (1) for x_{i-1} that represents the symbols consumed in the application of the production rule identified in (2).

(4) The substring of symbols of (1) for x_i that represents the symbols yielded by the application of the production rule identified in (2).

We can describe appropriate factorizations for the strings $x_0, \dots, x_k$ in more detail as follows. To each symbol of each of the strings $x_0, \dots, x_k \in (N \cup \Sigma)^*$, we can associate a *time of production*—the index i of the string x_i in which the symbol first appeared (i.e., was in the yield of the application of the production rule that produced x_i from x_{i-1})—and a *time of consumption*—the index j (if one exists, otherwise say ∞) such that the symbol is consumed by the application of the production rule that yields x_{j+1} from x_j . Say that two symbols in x_i are *equivalent* if (1) they are adjacent in x_i and (2) they have the same times of production and consumption. The relevant factorization of x_i is then given by the equivalence classes. Each factor can be expressed by a symbol $\beta_r[s, t] \in \Gamma$. By an easy induction, the factorization for each x_i has a representation over the symbols of Γ of length at most $1 + 2(k - 1)$.

It follows from the definition of the equivalence relation that if x_i is produced from x_{i-1} by the production rule $\alpha_i \rightarrow \beta_i$, then α_i is represented by a substring of the symbols of Γ that represent the factorization of x_{i-1} , and β_i is represented by a substring of the symbols of Γ that represent the factorization of x_i .

The second computational phase for the Turing machine consists of k deterministic checks, with the i -th check verifying that the guessed information of the first phase relevant to the derivation of x_i from x_{i-1} is consistent. It is easy to see

that each such check can be accomplished by $g(k)$ moves through a state space of polynomial size, for an appropriately chosen $g(k)$.

As a consequence of Corollary 11.4 and Lemma 11.5, we have proved the following.

Theorem 11.6 (Liming Cai, Chen, Downey, and Fellows) *SHORT DERIVATION is complete for $W[1]$.*

We next consider the complexity of **SHORT POST CORRESPONDENCE**.

Lemma 11.7 (Liming Cai, Chen, Downey, and Fellows) *SHORT POST CORRESPONDENCE reduces to SHORT TURING MACHINE ACCEPTANCE.*

Proof. Given an instance (Π, k) of the **SHORT POST CORRESPONDENCE** problem, we can easily express the question in terms of whether a particular Turing machine has a k' -step accepting computation for k' determined by an appropriate function of k , as follows. Consider a machine M that computes in two phases, over an alphabet that includes one symbol for each pair of strings in the Post system, and one symbol for each of the positive integers $1, \dots, m$, where m is a bound on the total number of symbols (in the strings of the Post system) for any solution. [Clearly, $m \leq k|\Pi|$.] In the first phase, M writes on $3k$ tape squares, indicating (nondeterministically) a *guessed solution* including the information: (1) what Post pair is the i -th factor for $i = 1, \dots, k$, (2) on what symbol positions the i -th factors begin (in the two concatenated strings). In the second phase, M conducts a number of “checks,” each consisting of (1) a scan of the *guess*, recording in the resulting state q , e.g., that the i -th factor is (x_j, y_j) and begins in the first component in symbol position r , and that the next factor is due to begin in symbol position s . There is a transition out of q in the transition table for M if and only if the information recorded in the state q is “valid,” i.e., $s = r + |x_j|$. The number of states required for this check is $O(|\Pi|^3)$. It is not too hard to see that $2k$ checks of this sort are enough to insure that the guessed starting positions of the factors are consistent. Similarly, it is necessary to make k^2 checks that each guessed (factor + starting position) in the first component is compatible with each guessed (factor + starting position) in the second component; that is, we must ensure that this information does not imply any mismatched symbols in the two solution strings for the Post problem. A successful check will make $f(k)$ moves, where f is an appropriately chosen function of k .

Theorem 11.8 (Liming Cai, Chen, Downey, and Fellows) *SHORT POST CORRESPONDENCE is complete for $W[1]$.*

Proof. Lemma 11.7 shows that the problem belongs to $W[1]$ via the analog of Cook’s Theorem (Theorem 10.9). To show that it is hard for $W[1]$, we compose the reduction of Lemma 11.3 (from **CLIQUE**) with the reduction of Theorem 4.2 of Davis [148] (attributed to an unpublished manuscript of Floyd); that is, we demonstrate that **SHORT DERIVATION** in k steps reduces to **SHORT POST CORRESPONDENCE**.

Actually, Floyd's reduction applies to what are called "Semi-Thue" processes. A semi-Thue process Π consists of a finite alphabet $A = \{a_1, \dots, a_n\}$ together with a set of productions of the form $g \rightarrow \bar{g}$. We will write $u_1 \xRightarrow{\Pi} u_2$ to mean that there is a production $g \rightarrow \bar{g}$ and strings a and b such that $u_1 = agb$ and $u_2 = a\bar{g}b$. Similarly, we will, as usual, denote the transitive closure by $\Rightarrow_{\Pi}^*$. The parametric problem at hand is

SHORT SEMI-THUE PROCESS

Input: A semi-Thue process Π , and words w and v .
Parameter: A positive integer k .
Question: Does $w \Rightarrow_{\Pi}^* v$ in $\leq k$ steps?

In general, Floyd's reduction below does *not* constitute a *parameterized* reduction of **SHORT DERIVATION** to **SHORT POST CORRESPONDENCE**. However, the image of the reduction from **CLIQUE** described by Lemma 11.3 consists of instances of the **SHORT DERIVATION** problem where (1) the grammar is context sensitive and is, indeed a semi-Thue process and (2) the word to be derived in k steps has length bounded by a function of k . Thus, to complete our proof, we need only describe a parameterized reduction from **SHORT SEMI-THUE PROCESS** to **SHORT POST CORRESPONDENCE**. As we see below, Floyd's reduction does the trick. Here is Floyd's reduction according to Davis [148].

Let Π be a semi-Thue process on the alphabet $A = \{a_1, \dots, a_n\}$, and let u and v be words on A . We shall construct a Post correspondence problem P on the alphabet

$$B = \{a_1, \dots, a_n, a'_1, \dots, a'_n, [,], *, *'\},$$

consisting of $2n + 4$ symbols. For any word w on A , we write w' for the word obtained from w by placing a prime after each symbol of w .

Let the productions of Π be $g_i \rightarrow \bar{g}_i$, $i = 1, 2, \dots, k$. We shall assume that included among these productions are the n "identity" productions $a_i \rightarrow a_i$, $i = 1, 2, \dots, n$. This last assumption does not restrict generality (i.e., the class of pairs $\langle r, s \rangle$ for which $r \Rightarrow_{\Pi}^* s$ is in no way changed by including the identity productions). However, we may now state that $u \Rightarrow_{\Pi}^* v$ if and only if we can write

$$u = u_1 \xRightarrow{\Pi} u_2 \xRightarrow{\Pi} \dots \xRightarrow{\Pi} u_m = v,$$

where m is *odd*. The Post correspondence system P on the alphabet B is then to consist of the pairs:

$$\left. \begin{array}{l} \langle [u*, [], \langle *, *' \rangle, \langle *', * \rangle, \\ \langle \bar{g}_j, g'_j \rangle \\ \langle \bar{g}'_j, g_j \rangle \end{array} \right\} j = 1, 2, \dots, k, \langle [], *'v \rangle.$$

Now, let

$$u = u_1 \xRightarrow{\Pi} u_2 \xRightarrow{\Pi} \dots \xRightarrow{\Pi} u_m = v,$$

where m is odd. Then, the word

$$w = [u_1 * u'_2 *' u_3 * \cdots * u'_{m-1} *' u_m]$$

is a solution of P , as is obvious from the two decompositions

$$\begin{aligned} w &= [u_1 * |u'_2 *' | u_3 * | \cdots |] \\ &= [|u_1 * | u'_2 *' | \cdots | *' u_m]. \end{aligned}$$

To see, for example, that $u'_2 *'$ corresponds to $u_1 *$, we note that we can write $u_1 = r g_j s$ and $u_2 = r \bar{g}_j s$ for some $j = 1, 2, \dots, k$. Then, $u'_2 = r' \bar{g}'_j s'$ and the correspondence is obvious (recalling that the productions $a_1 \rightarrow a_1, \dots, a_n \rightarrow a_n$ are present in Π).

Conversely, if $\bar{w}$ is a solution, then to avoid mismatches on the extreme left and right (between primed and unprimed symbols), $\bar{w}$ must begin $[$ and end $]$. Hence, letting w be the portion of $\bar{w}$ up to the first $]$,

$$w = [u * \cdots *' v],$$

where, to begin with, we have the correspondences

$$\begin{aligned} w &= [u * | \cdots *' v], \\ w &= [|u * \cdots | *' v]. \end{aligned}$$

Hence, we must have $u *$ corresponding to some $r' *'$ and $*' v$ to some $* s'$, where $u \Rightarrow_{\Pi}^* r$ and $s \Rightarrow_{\Pi}^* v$. Continuing this procedure, we see that $u \Rightarrow_{\Pi}^* v$. This completes the construction and one easily sees that this is, indeed, a parameterized polynomial time reduction from **SHORT SEMI-THUE PROCESS** to **SHORT POST CORRESPONDENCE**.

Corollary 11.9 **SHORT SEMI-THUE PROCESS** is $W[1]$ -complete.

As is well known, there is an intimate connection between logical decision problems and tiling problems. We consider the complexity of tiling small regions; the problem is defined as follows (see [239]).

SQUARE TILING

- Instance:* Set C of colors, collection $T \subseteq C^4$ of tiles (where $\langle a, b, c, d \rangle$ denotes a tile whose top, right, bottom, and left sides are colored a, b, c , and d , respectively).
- Parameter:* A positive integer k .
- Question:* Is there a tiling of a $k \times k$ square using the tiles in T , i.e., an assignment f of a tile $A(i, j) \in T$ to each ordered pair i, j , $1 \leq i \leq k$, $1 \leq j \leq k$, such that (1) if $f(i, j) = \langle a, b, c, d \rangle$ and $f(i+1, j) = \langle a', b', c', d' \rangle$, then $a' = c$, and (2) if $f(i, j+1) = \langle a', b', c', d' \rangle$, then $b = d'$?

Theorem 11.10 (Liming Cai, Chen, Downey, and Fellows) **SQUARE TILING** is complete for $W[1]$.

Proof. Membership in $W[1]$ is straightforward, again using **SHORT TURING MACHINE ACCEPTANCE** and Theorem 10.9. Given an instance of the tiling problem as defined above, we can create a Turing machine M that reaches a halting configuration in k' steps (where k' is an appropriate function of k) as follows. In the first phase of computation, M writes onto k^2 tape squares nondeterministically a choice of tiles. In the second phase of computation, M makes $2k(k-1)$ passes, each pass dedicated to checking the compatibility of the tiles chosen for two adjacent positions in the $k \times k$ square. In any given pass, the first recorded tile of the pair is “remembered” as state information.

To show that the problem is hard for $W[1]$, we reduce from the $W[1]$ -complete problem **CLIQUE** in two steps. The most important step is the reduction of **CLIQUE** to the problem of tiling a $k_1 \times k_2$ rectangular region. The second and easier step is to reduce the rectangular tiling problem to **SQUARE TILING**. We address the easier step first. Suppose $k_1 \leq k_2$.

Let T denote the set of tiles over the set of colors C . We describe how to construct a set of tiles T' over a set of colors C' such that a $k_1 \times k_2$ region can be tiled from T if and only if a $k' = k_2$ square region can be tiled from T' .

Suppose $z \notin C$. We may take $C' = C_1 \cup C_2$, where

$$\begin{aligned} C_1 &= C \times \{0, \dots, k_2\}, \\ C_2 &= \{z\} \times \{0, \dots, k_2\}. \end{aligned}$$

The set of tiles T' is the union $T' = T_1 \cup T_2 \cup T_3$, where

$$\begin{aligned} T_1 &= \{ \langle (a, i), (b, j+1), (c, i+1), (d, j) \rangle : \\ &\quad 0 \leq i \leq k_1 - 1, 0 \leq j \leq k_2 - 1, \langle a, b, c, d \rangle \in T \}, \\ T_2 &= \{ \langle (a, k_1), (z, j+1), (z, k_1+1), (z, j) \rangle : 0 \leq j \leq k_2 - 1, a \in C \}, \\ T_3 &= \{ \langle (z, i), (z, j+1), (z, i+1), (z, j) \rangle : k_1 + 1 \leq i \leq k_2 - 1, \\ &\quad 0 \leq j \leq k_2 - 1 \}. \end{aligned}$$

The basic idea is that the tiles of T_2 and T_3 provide for additional rows of padding to fill out the $k_2 \times k_2$ square. The verification that a $k_1 \times k_2$ rectangular tiling with tiles from T yields a $k_2 \times k_2$ square tiling with tiles from T' is straightforward and left to the reader.

Now, suppose there is a $k_2 \times k_2$ square tiling f with tiles from T' . The second components of the C' colors of the tiles in T' force the tile $A(i, j) = f(i, j)$ to have the form $\langle (x_1, i-1), (x_2, j), (x_3, i), (x_4, j-1) \rangle$, for $1 \leq i, j \leq k_2$. By the definition of T_1 , and forgetting the second components of the colors, f must also describe a tiling of the $k_1 \times k_2$ rectangle in the first k_1 rows with tiles from T .

We next argue that **CLIQUE** can be reduced to the rectangular tiling problem. Let $G = (V, E)$ and k be an instance of **CLIQUE**. We will describe a set of colors C and a set of tiles T that can tile a $k_1 \times k_2$ rectangle if and only if G

has a k -clique, where $k_1 = k$ and $k_2 = \binom{k}{2}$. We will consider the columns of the $k_1 \times k_2$ rectangle as indexed by J_k in lexicographic order. Let J_k denote the set of $\binom{k}{2}$ ordered pairs (r, s) with $1 \leq r < s \leq k$. Let J_k^+ denote J_k augmented with the single additional element 0. Consider J_k^+ to be linearly ordered by the lexicographic ordering inherited from J_k together with taking 0 to be the minimal element. For $\alpha \in J_k^+$, let $pre(\alpha)$ denote the immediate predecessor of α in the ordering of J_k^+ .

The set of colors is $C = C_1 \cup C_2$, where

$$\begin{aligned} C_1 &= J_k^+ \times V, \\ C_2 &= \{0, \dots, k\} \times E. \end{aligned}$$

The set of tiles is

$$\begin{aligned} T = \{ \langle (i-1, uv), (pre(\alpha), w), (i, uv), (\alpha, w) \rangle : 1 \leq i \leq k, uv \in E, w \in V, \\ \alpha = (r, s) \in J_k, \text{ with } (w = u \text{ if } i = r) \text{ and } (w = v \text{ if } i = s) \}. \end{aligned}$$

In discussing tiling, it is useful to have the notion of the sequence of colors in a row or column. Given a valid tiling f of the $k_1 \times k_2$ rectangle, write

$$f(i, j) = \langle f_1(i, j), f_2(i, j), f_3(i, j), f_4(i, j) \rangle;$$

that is, f_1 describes the top color assignment, f_2 describes the right-side color assignment, etc. Define the *sequence of colors of the i -th row* to be the sequence of length $k_1 + 1$:

$$\begin{aligned} f_4(i, 1), f_2(i, 1) &= f_4(i, 2), f_2(i, 2) \\ &= f_4(i, 3), \dots, f_2(i, k_1 - 1) = f_4(i, k_1), f_2(i, k_1). \end{aligned}$$

Similarly, define the sequence of colors of a column.

Claim 1. *If there is a k -clique in G , then the $k_1 \times k_2$ rectangle can be tiled from T .*

Proof. Let $v_1, \dots, v_k$ be the vertices of the clique. We describe a tiling by describing the row and column color sequences. In the i -th row the sequence of colors is

$$(0, v_i), ((1, 2), v_i), ((1, 3), v_i), \dots, ((1, k), v_i), ((2, 3), v_i), \dots, ((k-1, k), v_i).$$

In the column indexed by $\alpha = (r, s)$, the sequence of colors is

$$(0, v_r v_s), (1, v_r v_s), (2, v_r v_s), \dots, (k, v_r v_s).$$

From this point, it remains only to check that these sequences can be realized by tiles of T , which we leave to the reader.

Claim 2. *If the $k_1 \times k_2$ rectangle can be tiled from T , then G has a k -clique.*

Proof. Let f denote the tiling. By forgetting the first components, we may refer to the colors of C_1 as *representing* a vertex, and we may similarly refer to the colors of C_2 as representing an edge. First, note that in any valid tiling from T , we must have the same vertex represented in all of the colors of a row sequence, simply because every tile has the same vertex represented on its sides. Similarly, the same edge is represented in the sequence of colors of any column. Furthermore, the first components of the sequence of colors in a column must be increasing and is thus forced to be $0, 1, 2, \dots, k$. A similar statement holds for the sequence of colors in a row. Write v_i to denote the vertex represented in row i . Let $i < j$ and consider the column indexed by (i, j) . It is straightforward to check that the definition of T forces the edge represented by this column to be $v_i v_j$, which implies that the v_i are distinct and pairwise adjacent in G .

Claims 1 and 2 complete the proof of the theorem.

The last problem we will study in this chapter is the very important **LONGEST COMMON SUBSEQUENCE (LCS)** problem. The computational problem of finding the longest common subsequence of a set of k strings has been studied extensively over the last twenty years (see Hirschberg [269], and Irving and Fraser [277] and references therein). This problem has many applications. When $k = 2$, the longest common subsequence is a measure of the similarity of two strings and is thus useful in molecular biology, pattern recognition, and text compression (Lu and Fu [337], Maier [343], and Sankoff [405]). The version of LCS in which the number of strings is unrestricted is also useful in text compression [343], and is a special case of the multiple sequence alignment and consensus subsequence discovery problems in molecular biology (see, for instance, Pevzner [372], Day and McMorris [149, 150]).

To date, most research has focused on deriving efficient algorithms for the LCS problem when $k = 2$ (see Hirschberg [269] and Irving and Fraser [277] and references therein). Most of these algorithms are based on the dynamic programming approach of Pearson and Miller [371] and require quadratic time. Although the k -unrestricted LCS problem is *NP*-complete ([343]), certain of the algorithms for the $k = 2$ case have been extended to yield algorithms that require $O(n^{(k-1)})$ time and space, where n is the length of the longest of the k strings (see Irving and C. B. Fraser [277] and Baeza-Yates [43]).

In this book, we will study several variations of the LCS problem. The differing incarnations of the problem arise as we vary the parameters.

LONGEST COMMON SUBSEQUENCE (LCS-1, LCS-2, and LCS-3)

Input: A set of k strings $X_1, \dots, X_k$ over an alphabet Σ , and a positive integer m .

Parameter 1: k . (We refer to this problem as **LCS-1**.)

Parameter 2: m . (We refer to this problem as **LCS-2**.)

Parameter 3: (k, m) . (We refer to this problem as **LCS-3**.)

Question: Is there a string $X \in \Sigma^*$ of length at least m that is a subsequence of X_i for $i = 1, \dots, k$?

TABLE 11.1. The Fixed-Parameter Complexity of the LCS Problem

Problem	Parameter	Alphabet Size $ \Sigma $	
		Unbounded	Fixed
LCS-1	k	$W[t]$ -hard, $t \geq 1$	?
LCS-2	m	$W[2]$ -hard	FPT
LCS-3	k, m	$W[1]$ -complete	FPT

The parameterized complexity of the LCS problem is summarized in Table 11.1. The results are all due to Bodlaender, et al. [75, 76].

We will delay the analysis of LCS-1 and LCS-2 until Chapter 12. We remark that LCS-1 is particularly important from our point of view since it has been used as the basis of many W -hardness results concerning problems arising from molecular biology. The reader is referred to Bodlaender, Fellows, and Hallett [79], Kaplan, Shamir, and Tarjan [289], and Bodlaender, et al. [75, 76, 73].

Theorem 11.11 *LCS-3 is complete for $W[1]$.*

Proof. • *Membership in $W[1]$.* Membership in $W[1]$ can be seen by a reduction to **WEIGHTED s -CNF SATISFIABILITY** for expressions having bounded clause size. By padding with new symbols or by repeating some of the X_i , we can assume for convenience (with polynomially bounded blowup) that $k = m$. The idea is to use a truth assignment of weight k^2 to indicate the k positions in each of the k strings of an instance of LCS-3 that yield a common subsequence of length k .

The details are as follows. Let $X_1, \dots, X_k$ be an instance of LCS-3. By a trivial padding with symbols having only a single occurrence, we may assume that the strings X_i are all of length n . Let $a[i, j]$ denote the j -th symbol of X_i . Let $B = \{b[i, j, r] : 1 \leq i \leq k, 1 \leq j \leq n, 1 \leq r \leq k\}$ be a set of boolean variables. The interpretation we intend for the variable $b[i, j, r]$ is that the r -th symbol $x[r]$ of a length k common subsequence $X = x[1] \cdots x[k]$ occurs as the symbol $a[i, j]$ in the string X_i ; that is, $x[j] = a[i, j]$. Let B_i be the set of elements $b[i, j, r]$ with first index i .

Let $E = E_1 E_2 E_3$ be the boolean expression over the set of variables B , where

$$\begin{aligned}
 E_1 &= \prod_{i=1}^k \prod_{r=1}^k \prod_{1 \leq j < j' \leq n} (\neg b[i, j, r] + \neg b[i, j', r]), \\
 E_2 &= \prod_{i=1}^k \prod_{j=1}^n \prod_{1 \leq r < r' \leq k} (\neg b[i, j, r] + \neg b[i, j, r']), \\
 E_3 &= \prod_{r=1}^k \prod_{1 \leq i < i' \leq k} \prod_{1 \leq j \leq j' \leq n} \prod_{a[i, j] \neq a[i', j']} (\neg b[i, j, r] + \neg b[i', j', r]).
 \end{aligned}$$

We claim that E has a weight k^2 truth assignment if and only if the X_i have a common subsequence of length k . It is easy to verify that a truth assignment corresponding to a length k common subsequence according to our intended interpretation of the boolean variables satisfies E . For the converse direction, suppose τ is a weight k^2 truth assignment that satisfies E . The clauses of E_1 ensure (by the Pigeonhole Principle) that no more than k variables of B_i are set *true* for $i = 1, \dots, k$. Consequently, there must be exactly k variables set to *true* in each B_i , and since E_2 is satisfied, these must indicate k distinct positions in X_i according to our interpretation. The clauses of E_3 ensure that the corresponding subsequence symbols in the k strings are the same.

W[1]-hardness.

To show $W[1]$ -hardness, we reduce from **CLIQUE**. Let $G = (V, E)$ be a graph for which we wish to determine whether G has a k -clique. We show how to construct a family $\mathcal{F}_G$ of $k' = f(k)$ sequences over an alphabet Σ that have a common subsequence of length $k'' = g(k)$ if and only if G contains a k -clique. Assume for convenience that the vertex set of G is $V = \{1, \dots, n\}$.

The Alphabet. We first describe the alphabet $\Sigma = \Sigma_1 \cup \Sigma_2 \cup \Sigma_3 \cup \Sigma_4$. We refer to these as *vertex symbols* (Σ_1), *edge symbols* (Σ_2), *vertex position symbols* (Σ_3), and *edge position symbols* (Σ_4).

$$\Sigma_1 = \{\alpha[p, q, r] : 1 \leq p \leq k, 0 \leq q \leq 1, 1 \leq r \leq n\},$$

$$\Sigma_2 = \{\beta[i, j, q, u, v] : 1 \leq i < j \leq k, 0 \leq q \leq 1, 1 \leq u < v \leq n, uv \in E\},$$

$$\Sigma_3 = \{\gamma[p, q, b] : 1 \leq p \leq k, 0 \leq q \leq 1, 0 \leq b \leq 1\},$$

$$\Sigma_4 = \{\delta[i, j, q, b] : 1 \leq i < j \leq k, 0 \leq q \leq 1, 0 \leq b \leq 1\}.$$

We will use the following shorthand notation to refer to various subsets of Σ . The notation indicates which indices are held fixed to some value, with “*” indicating that the index should vary over its range of definition in building the set. For example, $\Sigma_1[p, *, r] = \{\alpha[p, q, r] : 0 \leq q \leq 1\}$ is the set of two elements with the first and third indices fixed at p and r , respectively.

An Example of a Clique Representation. The sequences in $\mathcal{F}$ are constructed in such a way that the k -cliques in G (considered with vertices in ascending order) are in 1:1 correspondence with the common subsequences of length k'' . It will be useful in motivating the construction to consider an example of this intended correspondence. Consider a graph having a 3-clique on the vertices $\{a, b, c\}$.

This 3-clique would be represented by the following common subsequence $\sigma(a, b, c)$, which we describe according to a hierarchy of factorizations. (Exponential notation indicates repetition of a symbol.)

$$\sigma(a, b, c) = \langle \text{first vertex} \rangle \langle \text{second vertex} \rangle \langle \text{third vertex} \rangle,$$

where

$$\langle \text{first vertex} \rangle = \langle \text{vertex 1} \rangle \langle \text{edge (1,2)} \rangle \langle \text{edge (1,3)} \rangle \langle \text{vertex 1 echo} \rangle,$$

$$\langle \text{second vertex} \rangle = \langle \text{vertex 2} \rangle \langle \text{edge (1,2) echo} \rangle \langle \text{edge (2,3)} \rangle \langle \text{vertex 2 echo} \rangle,$$

$$\langle \text{third vertex} \rangle = \langle \text{vertex 3} \rangle \langle \text{edge (1,3) echo} \rangle \langle \text{edge (2,3) echo} \rangle \langle \text{vertex 3 echo} \rangle,$$

and where the constituent subsequences over Σ are

$$\begin{aligned}
 \langle \text{vertex 1} \rangle &= \gamma[1, 0, 0]^w \alpha[1, 0, a] \gamma[1, 0, 1]^w, \\
 \langle \text{edge (1,2)} \rangle &= \delta[1, 2, 0, 0]^w \beta[1, 2, 0, a, b] \delta[1, 2, 0, 1]^w, \\
 \langle \text{edge (1,3)} \rangle &= \delta[1, 3, 0, 0]^w \beta[1, 3, 0, a, c] \delta[1, 3, 0, 1]^w, \\
 \langle \text{vertex 1 echo} \rangle &= \gamma[1, 1, 0]^w \alpha[1, 1, a] \gamma[1, 1, 1]^w, \\
 \langle \text{vertex 2} \rangle &= \gamma[2, 0, 0]^w \alpha[2, 0, b] \gamma[2, 0, 1]^w, \\
 \langle \text{edge (1,2) echo} \rangle &= \delta[1, 2, 1, 0]^w \beta[1, 2, 1, a, b] \delta[1, 2, 1, 1]^w, \\
 \langle \text{edge (2,3)} \rangle &= \delta[2, 3, 0, 0]^w \beta[2, 3, 0, b, c] \delta[2, 3, 0, 1]^w, \\
 \langle \text{vertex 2 echo} \rangle &= \gamma[2, 1, 0]^w \alpha[2, 1, b] \gamma[2, 1, 1]^w, \\
 \langle \text{vertex 3} \rangle &= \gamma[3, 0, 0]^w \alpha[3, 0, c] \gamma[3, 0, 1]^w, \\
 \langle \text{edge (1,3) echo} \rangle &= \delta[1, 3, 1, 0]^w \beta[1, 3, 1, a, c] \delta[1, 3, 1, 1]^w, \\
 \langle \text{edge (2,3) echo} \rangle &= \delta[2, 3, 1, 0]^w \beta[2, 3, 1, b, c] \delta[2, 3, 1, 1]^w, \\
 \langle \text{vertex 3 echo} \rangle &= \gamma[3, 1, 0]^w \alpha[3, 1, c] \gamma[3, 1, 1]^w.
 \end{aligned}$$

In the above, the position symbols are repeated $w = w(k)$ times for reasons useful for the correctness argument concerning the reduction.

The Target Parameters. There are $f_1(k) = 2k + k(k - 1) = k^2 + k$ position symbols (in Σ_3 and Σ_4). We take $w = f_1(k)^2 + 1$, $k' = f_1(k) + 2$, and $k'' = (w + 1)f_1(k)$.

Symbol Subsets and Operations. It is convenient to introduce a linear ordering on Σ that corresponds to the “natural” order in which the various symbols occur, as illustrated by the example above. We can achieve this by defining a “weight” on the symbols of Σ and then ordering the symbols by weight.

Let $N = 2kn$ (a value conveniently larger than k and n). Define the *weight* $\|a\|$ of a symbol $a \in \Sigma$ by

$$\|a\| = \begin{cases} pN^6 + qN^5 + r & \text{if } a = \alpha[p, q, r] \in \Sigma_1, \\ q'iN^6 + qjN^6 + q'N^4 + q'jN^3 + qiN^3 + uN + v & \text{if } a = \beta[i, j, q, u, v] \in \Sigma_2, \\ pN^6 + qN^5 + bN^2 & \text{if } a = \gamma[p, q, b] \in \Sigma_3, \\ q'iN^6 + qjN^6 + q'N^4 + q'jN^3 + qiN^3 + bN^2 & \text{if } a = \delta[i, j, q, b] \in \Sigma_4, \end{cases}$$

where $q' = (q - 1)^2$.

Define a linear order on Σ by $a < b$ if and only if $\|a\| < \|b\|$. The reader can verify that, assuming $a < b < c$, the symbols of the example sequence $\sigma(a, b, c)$ described above occur in ascending order.

For $a, b \in \Sigma$, $a < b$, we define the *segment* $\Sigma(a, b)$ to be $\Sigma(a, b) = \{e \in \Sigma : a \leq e \leq b\}$, and we define the segments $\Sigma_i(a, b)$ similarly.

If Γ is a finite set of symbols, then it is easy to see that there is a “universal” string $(m\Gamma) \in \Gamma^*$ of length $m|\Gamma|$ that contains as a subsequence every string of

length at most m over Γ , for example, by running through the symbols in Γ m times. We will use the notation $(m\Gamma)$ to refer to any choice of such a string. Where m is unimportant except that it be “large enough” (with the understanding that this means also “not too large”), we may write $(*\Gamma)$ for convenience.

If $\Gamma \subseteq \Sigma$, let $(\uparrow \Gamma)$ be the string of length $|\Gamma|$ which consists of one occurrence of each symbol in Γ in ascending order, and let $(\downarrow \Gamma)$ be the string of length $|\Gamma|$ which consists of one occurrence of each symbol in Γ in descending order.

String Gadgets. We next describe some “high level” component subsequences for the construction. In the following, let $\updownarrow$ denote either $\uparrow$ or $\downarrow$. Product notation is interpreted as referring to concatenation.

Vertex and Edge Selection Gadgets

$$\begin{aligned} \langle \updownarrow \text{ vertex } p \rangle &= \gamma[p, 0, 0]^w (\updownarrow \Sigma_1[p, 0, *]) \gamma[p, 0, 1]^w, \\ \langle \updownarrow \text{ vertex } p \text{ echo} \rangle &= \gamma[p, 1, 0]^w (\updownarrow \Sigma_1[p, 1, *]) \gamma[p, 1, 1]^w, \\ \langle \updownarrow \text{ edge } (i, j) \rangle &= \delta[i, j, 0, 0]^w (\updownarrow \Sigma_2[i, j, 0, *, *]) \delta[i, j, 0, 1]^w, \\ \langle \updownarrow \text{ edge } (i, j) \text{ echo} \rangle &= \delta[i, j, 1, 0]^w (\updownarrow \Sigma_2[i, j, 1, *, *]) \delta[i, j, 1, 1]^w, \\ \langle \updownarrow \text{ edge } (i, j) \text{ from } u \rangle &= \delta[i, j, 0, 0]^w (\updownarrow \Sigma_2[i, j, 0, u, *]) \delta[i, j, 0, 1]^w, \\ \langle \updownarrow \text{ edge } (i, j) \text{ to } v \rangle &= \delta[i, j, 1, 0]^w (\updownarrow \Sigma_2[i, j, 1, *, v]) \delta[i, j, 1, 1]^w. \end{aligned}$$

Control and Selection Assemblies

$$\begin{aligned} \langle \updownarrow \text{ control } p \rangle &= \langle \updownarrow \text{ vertex } p \rangle \left(\prod_{s=1}^{p-1} \langle \updownarrow \text{ edge } (s, p) \text{ echo} \rangle \right) \\ &\quad \times \left(\prod_{s=p+1}^k \langle \updownarrow \text{ edge } (p, s) \rangle \right) \langle \updownarrow \text{ vertex } p \text{ echo} \rangle, \\ \langle \uparrow \text{ choice } p \rangle &= \prod_{x=1}^n \left(\gamma[p, 0, 0]^w \alpha[p, 0, x] \gamma[p, 0, 1]^w \prod_{t=1}^{p-1} \langle \uparrow \text{ edge } (t, p) \text{ to } x \rangle \right. \\ &\quad \times \left. \prod_{t=p+1}^k \langle \uparrow \text{ edge } (p, t) \text{ from } x \rangle \gamma[p, 1, 0]^w \alpha[p, 1, x] \gamma[p, 1, 1]^w \right), \\ \langle \downarrow \text{ choice } p \rangle &= \prod_{x=n}^{\text{down to } 1} \left(\gamma[p, 0, 0]^w \alpha[p, 0, x] \gamma[p, 0, 1]^w \prod_{t=1}^{p-1} \langle \downarrow \text{ edge } (t, p) \text{ to } x \rangle \right. \\ &\quad \times \left. \prod_{t=p+1}^k \langle \downarrow \text{ edge } (p, t) \text{ from } x \rangle \gamma[p, 1, 0]^w \alpha[p, 1, x] \gamma[p, 1, 1]^w \right). \end{aligned}$$

Edge Symbol Pairing Gadget

$$\begin{aligned} \langle \text{edge } (i, j) \text{ from } u \text{ to } v \rangle \\ = \beta[i, j, 0, u, v] (*\Sigma(\delta[i, j, 0, 1], \delta[i, j, 1, 0])) \beta[i, j, 1, u, v] \end{aligned}$$

The Reduction. We may now describe the reduction. The instance of LCS-3 consists of strings which we may consider as belonging to three subsets: *Control*,

Selection, and *Check*. The two strings in the *Control* set are

$$X_1 = \prod_{t=1}^k \langle \uparrow \text{ control } t \rangle,$$

$$X_2 = \prod_{t=1}^k \langle \downarrow \text{ control } t \rangle.$$

The $2k$ strings in the *Selection* set are, for $p = 1, \dots, k$,

$$Y_p = \left(\prod_{t=1}^{p-1} \langle \uparrow \text{ control } t \rangle \right) \langle \uparrow \text{ choice } p \rangle \left(\prod_{t=p+1}^k \langle \uparrow \text{ control } t \rangle \right),$$

$$Y'_p = \left(\prod_{t=1}^{p-1} \langle \downarrow \text{ control } t \rangle \right) \langle \downarrow \text{ choice } p \rangle \left(\prod_{t=p+1}^k \langle \uparrow \text{ control } t \rangle \right).$$

The $2\binom{k}{2} = k(k-1)$ strings in the *Check* set are, for $1 \leq i < j \leq k$,

$$\begin{aligned} Z_{i,j} = & \left(\prod_{t=1}^{i-1} \langle \uparrow \text{ control } t \rangle \right) \langle \uparrow \text{ vertex } i \rangle \left(\prod_{s=1}^{i-1} \langle \uparrow \text{ edge } (s, i) \text{ echo} \rangle \right) \\ & \times \left(\prod_{s=i+1}^{j-1} \langle \uparrow \text{ edge } (i, s) \rangle \right) \\ & \times \delta[i, j, 0, 0]^w \prod_{\substack{1 \leq u < v \leq n \\ uv \in E}}^{\text{lex } \uparrow} \langle \text{edge } (i, j) \text{ from } u \text{ to } v \rangle \\ & \times \delta[i, j, 1, 1]^w \left(\prod_{s=i+1}^{j-1} \langle \uparrow \text{ edge } (s, j) \text{ echo} \rangle \right) \left(\prod_{s=j+1}^k \langle \uparrow \text{ edge } (j, s) \rangle \right) \\ & \times \langle \uparrow \text{ vertex } j \text{ echo} \rangle \left(\prod_{t=j+1}^k \langle \uparrow \text{ control } t \rangle \right), \end{aligned}$$

$$\begin{aligned} Z'_{i,j} = & \left(\prod_{t=1}^{i-1} \langle \downarrow \text{ control } t \rangle \right) \langle \downarrow \text{ vertex } i \rangle \left(\prod_{s=1}^{i-1} \langle \downarrow \text{ edge } (s, i) \text{ echo} \rangle \right) \\ & \times \left(\prod_{s=i+1}^{j-1} \langle \downarrow \text{ edge } (i, s) \rangle \right) \\ & \times \delta[i, j, 0, 0]^w \prod_{\substack{1 \leq u < v \leq n \\ uv \in E}}^{\text{lex } \downarrow} \langle \text{edge } (i, j) \text{ from } u \text{ to } v \rangle \end{aligned}$$

$$\begin{aligned} & \times \delta[i, j, 1, 1]^w \left(\prod_{s=i+1}^{j-1} \langle \downarrow \text{edge } (s, j) \text{ echo} \rangle \right) \left(\prod_{s=j+1}^k \langle \downarrow \text{edge } (j, s) \rangle \right) \\ & \times \langle \downarrow \text{vertex } j \text{ echo} \rangle \left(\prod_{t=j+1}^k \langle \downarrow \text{control } t \rangle \right). \end{aligned}$$

We comment that the key difference between $Z_{i,j}$ and $Z'_{i,j}$ is that in $Z_{i,j}$, the edge symbol pairing gadgets occur in increasing lexicographic order, and in $Z'_{i,j}$, the gadgets are in decreasing lexicographic order.

Proof of Correctness. Where S_1 and S_2 are strings of symbols, let $l(S_1, S_2)$ denote the maximum length of a common subsequence of S_1 and S_2 .

In the Control Strings X_1 and X_2 , we distinguish certain substrings that we term *positions*. Note that both of these strings are formed as the concatenation of four different kinds of substrings: $\langle \text{vertex} \rangle$, $\langle \text{vertex echo} \rangle$, $\langle \text{edge} \rangle$, and $\langle \text{edge echo} \rangle$, and that each of these “vertex and edge selection” substrings begins and ends with a matched pair of substrings of repeated symbols from Σ_3 (in the case of vertex selection), or from Σ_4 (in the case of edge selection). These matched pairs of position symbol substrings determine a *position*—note that these position symbol substrings (and therefore the positions defined) occur in the same order in X_1 and X_2 . Thus, there are $k(2 + k - 1) = k^2 + k$ positions.

Between a matched pair of position symbol substrings in X_1 there is a set of symbols in increasing order that we will term a *set of (vertex or edge) stairs*, and in X_2 in the corresponding position, there occurs the same set of symbols in decreasing order. The proof of the following claim is trivial.

Claim 1. Suppose Σ is a linearly ordered finite alphabet, that $S \uparrow$ is the string consisting of the symbols of Σ in increasing order, and that $S \downarrow$ is the string consisting of the symbols of Σ in decreasing order. Then $l(S \uparrow, S \downarrow) = 1$.

Claim 2. A common subsequence C of the control sequences X_1 and X_2 of maximum length l satisfies the conditions (1) $l = k''$ and (2) C consists of the position symbol substrings (common to X_1 and X_2) together with one symbol in each position defined by these substrings.

Proof. It is clear that $l \geq k''$ because there are many different common subsequences of length k'' consisting of all the position symbol substrings (which are the same in X_1 and X_2) together with a single choice of vertex or edge symbol in each position. Now, suppose there is a common subsequence C of length greater than k'' and fix attention on subsequences C_1 of X_1 and C_2 of X_2 that are isomorphic to C (for the reason that C might occur in more than one way as a subsequence). Then, C_1 must contain two vertex or edge symbols ϵ_1 and ϵ_2 that occur on the same set of stairs in X_1 . By Claim 1, these two symbols, considered now in C_2 , cannot occur on the same set of stairs in X_2 . This implies that any position symbols between ϵ_1 and ϵ_2 in X_2 do not belong to C_2 . Consequently, there are at least $2w$ position symbols of X_2 that do not occur in $C = C_2$. But in order for the length of C to be at least k'' , this means that C must contain more than $f_1(k)^2$ vertex and edge symbols. By the Pigeonhole Principle, there must, therefore, be a set of stairs in X_1

that contains $m > f_1(k)$ vertex or edge symbols of C_1 . By Claim 1, no more than one of the corresponding symbols in C_2 can occur on any set of stairs in X_2 , and, therefore, X_2 must have at least m sets of stairs, a contradiction. This establishes (1), and, furthermore, shows that no two symbols of a common subsequence of length k'' can occur on the same set of stairs. Thus, (2) may also be concluded by observing that there must be at least one vertex or edge symbol from each set of stairs, else the length of C would be less than k'' .

By Claim 2, if C is a common subsequence of X_1 and X_2 of length k'' , we may refer unambiguously to the vertices and edges represented in the various positions of C . In particular, note that these positions occur in k vertex units, each of which consists of an *initial vertex position*, followed by $k - 1$ *edge and edge echo positions* and concluding with a *terminal vertex echo position*. If uv is an edge of the graph with $u < v$, then we refer to u as the *initial vertex* and to v as the *terminal vertex* of the edge.

Claim 3. If C is a subsequence of length k'' common to the Control and Selection sets, then in each vertex unit, (1) the vertex u represented in the initial vertex position is also represented in the terminal vertex echo position, (2) each edge represented in an edge echo position has terminal vertex u , and (3) each edge represented in an edge position has initial vertex u .

Proof. Suppose C is a subsequence of length k'' common to X_1 and X_2 . We argue that if C is also common to Y_p and Y'_p then the statements of the Claim are satisfied for the p -th vertex unit. Let C_p and C'_p denote specific subsequences of Y_p and Y'_p , respectively, with $C = C_p = C'_p$.

The strings Y_p and Y'_p differ from the Control strings X_1 and X_2 , respectively, only in the replacement of a $\langle \downarrow$ control $p \rangle$ gadget with a $\langle \downarrow$ choice $p \rangle$ gadget. In particular, the position symbols in the other constituent substrings occur in the same way in all four strings, and so by Claim 2, C_p (C'_p) must include all of the length w position symbol substrings in Y_p (Y'_p) occurring outside of $\langle \uparrow$ choice $p \rangle$ ($\langle \downarrow$ choice $p \rangle$). Furthermore, C_p must contain precisely two vertex symbols α and α' , appropriately positioned, from $\langle \uparrow$ choice $p \rangle$, and C'_p must contain the same two (and no other) vertex symbols from $\langle \downarrow$ choice $p \rangle$.

The subsequence of Y_p consisting of all the vertex symbols in $\langle \uparrow$ choice $p \rangle$ is the vertex index increasing sequence

$$S = \prod_{x=1}^n (\alpha[p, 0, x] \alpha[p, 1, x])$$

and the subsequence of Y'_p consisting of all the vertex symbols in $\langle \downarrow$ choice $p \rangle$ is the vertex index decreasing sequence

$$S' = \prod_{x=n}^1 (\alpha[p, 0, x] \alpha[p, 1, x]).$$

The only possibility for α and α' to be common to S and S' is for α and α' to represent the same vertex u ; that is, $\alpha = \alpha[p, 0, u]$ and $\alpha' = \alpha[p, 1, u]$. This establishes (1).

Consider the position symbols occurring in Y_p between α and α' in C_p , and occurring in Y'_p between α and α' in C'_p . Since these must occur in C (by Claim 2) and this can happen in only one way, all of these position symbols must belong to C_p and C'_p , respectively. This ensures (2) and (3).

The length w substrings of the position symbols $\delta[i, j, 0, 0]$ and $\delta[i, j, 0, 1]$ in C define the (i, j) -th edge position in the i -th vertex unit and the length w substrings of the position symbols $\delta[i, j, 1, 0]$ and $\delta[i, j, 1, 1]$ in C define the (i, j) -th edge echo position in the j -th vertex unit. We term these a *corresponding pair* of edge and edge echo positions.

Claim 4. If C is a subsequence of length k'' common to the Control, Selection, and Check sets, then for each corresponding pair of an edge position and an edge echo position, the same edge must be represented in the two positions.

Proof. Suppose C is a subsequence of length k'' common to the Control and Selection sets. We argue that if C is also common to $Z_{i,j}$ and $Z'_{i,j}$, then the claim holds for the (i, j) -th corresponding pair of positions. Let $C_{i,j}$ and $C'_{i,j}$ denote specific subsequences of $Z_{i,j}$ and $Z'_{i,j}$ isomorphic to C .

It is convenient to consider $Z_{i,j}$ (and similarly $Z'_{i,j}$) under the factorization $Z_{i,j} = Z_{i,j}(1)Z_{i,j}(2)Z_{i,j}(3)$ where

$$Z_{i,j}(2) = \prod_{\substack{1 \leq u < v \leq n \\ uv \in E}}^{\text{lex } \uparrow} \langle \text{edge } (i, j) \text{ from } u \text{ to } v \rangle$$

and where $Z_{i,j}(1)$ and $Z_{i,j}(3)$ are the appropriately defined prefix and suffix, respectively, of $Z_{i,j}$.

Since none of the position symbols in $Z_{i,j}(1)$ or $Z_{i,j}(3)$ occur in $Z_{i,j}(2)$, all of the position symbols in $Z_{i,j}(1)$ and $Z_{i,j}(3)$ must belong to $C_{i,j}$. Similarly, all of the position symbols in $Z'_{i,j}(1)$ and $Z'_{i,j}(3)$ must belong to $C'_{i,j}$. This implies, by Claim 2, that $C_{i,j} \cup Z_{i,j}(2) = C'_{i,j} \cup Z'_{i,j}(2)$ begins with a symbol $\beta[i, j, 0, u, v]$ and ends with a symbol $\beta[i, j, 0, x, y]$. We argue that necessarily $u = x$ and $v = y$.

From the fact that $\beta[i, j, 1, x, y]$ follows $\beta[i, j, 0, u, v]$ in $Z_{i,j}(2)$, and from the construction of the latter in increasing lexicographic order, we may deduce that (u, v) precedes (x, y) lexicographically. Similarly, since $Z'_{i,j}(2)$ is constructed in decreasing lexicographic order, we deduce that (x, y) precedes (u, v) , and therefore $(x, y) = (u, v)$.

We now argue the correctness of the reduction as follows. If G has a k -clique, then it is easily seen that there is a common subsequence of length k'' in which the k vertex units represent the vertices of the clique, and the edge and edge echo positions within each vertex unit represent the edges incident on the represented vertex of the unit in increasing lexicographic order. (Each edge is thus represented

twice, in the vertex units corresponding to its endpoints, first in an edge position in the initial vertex unit, and second in an edge echo position in the terminal vertex unit.)

Conversely, suppose there is a common subsequence C of length k'' . By Claims 2 and 3, C represents a sequence of k vertices of G . That these must be a clique in G follows from Claim 4 and the definition of the “edge from” and “edge to” gadgets, which restrict the edges represented in a vertex unit to those present in the graph and for which the vertex is, respectively, initial or terminal. That completes the proof.

Exercises 11.0

- 11.0.1.** Prove that **SHORT DERIVATION** is in *FPT* if the left-hand side always consists of a single nonterminal symbol.
- 11.0.2.** Prove that **LCS-2** and **LCS-3** are in *FPT* if $|\Sigma|$ is fixed.
- 11.0.3.** (Some classes suspected to be between $W[1]$ and $W[2]$.) (Downey and Fellows [170]) Recall that $W[2]$ denotes the parameterized languages reducible to **WEIGHTED CNF SATISFIABILITY**. Consider the following problems:

PERFECT CODE

- Instance:* A graph $G = (V, E)$.
Parameter: A positive integer k .
Question: Does G have a k -element perfect code? A *perfect code* is a set of vertices $V' \subseteq V$ with the property that for each vertex $v \in V$ there is precisely one vertex in $N[v] \cap V'$.

WEIGHTED EXACT CNF SATISFIABILITY

- Instance:* A boolean expression E in conjunctive normal form.
Parameter: A positive integer k .
Question: Is there a truth assignment of weight k to the variables of E that makes exactly one literal in each clause of E true?

Prove the following:

- (i) **PERFECT CODE** $\in W[2]$.
 (ii) **PERFECT CODE** $\equiv_m^s$ **WEIGHTED EXACT SATISFIABILITY**.
 (iii) **PERFECT CODE** is $W[1]$ -hard.

[Hint: (i) Let $G = (V, E)$ be a graph for which we wish to determine whether G has a k -element perfect code. It suffices to show how to efficiently construct a boolean expression E_G in product-of-sums form that has a weight k truth assignment if and only if the graph G has a k -element perfect code. One can take the expression E_G to be the expression $E = E_0 E_1 E_2$ where the variables

of E_G are in one-to-one correspondence with vertices of G and

$$\begin{aligned} E_0 &= \prod_{u \in V} \sum_{x \in N[u]}, \\ E_1 &= \prod_{uv \in E} (\neg u + \neg v), \\ E_2 &= \prod_{uv, vw \in E} (\neg u + \neg w). \end{aligned}$$

(ii) To see that **WEIGHTED EXACT CNF SATISFIABILITY** $\leq_m^s$ **PERFECT CODE**, a reduction can be demonstrated using the transformation used in the proof of the key combinatorial lemma, Lemma 12.4. The other direction uses the expression of part (i).

(iii) We reduce from **INDEPENDENT SET**. Let $G = (V, E)$ be a graph. We show how to produce a graph $H = (V', E')$ that has a perfect code of size $k' = \binom{k}{2} + k + 1$ if and only if G has a k -element independent set. The vertex set V' of H is the union of the sets of vertices:

$$V_1 = \{a[s] : 0 \leq s \leq 2\},$$

$$V_2 = \{b[i] : 1 \leq i \leq k\},$$

$$V_3 = \{c[i] : 1 \leq i \leq k\},$$

$$V_4 = \{d[i, u] : 1 \leq i \leq k, u \in V\},$$

$$V_5 = \{e[i, j, u] : 1 \leq i < j \leq k, u \in V\},$$

$$V_6 = \{f[i, j, u, v] : 1 \leq i < j \leq k, u, v \in V\}.$$

The edge set E' of H is the union of the sets of edges:

$$E_1 = \{a[0]a[i] : i = 1, 2\},$$

$$E_2 = \{a[0]b[i] : 1 \leq i \leq k\},$$

$$E_3 = \{b[i]c[i] : 1 \leq i \leq k\},$$

$$E_4 = \{c[i]d[i, u] : 1 \leq i \leq k, u \in V\},$$

$$E_5 = \{d[i, u]d[i, v] : 1 \leq i \leq k, u, v \in V\},$$

$$E_6 = \{d[i, u]e[i, j, u] : 1 \leq i < j \leq k, u \in V\},$$

$$E_7 = \{d[j, v]e[i, j, u] : 1 \leq i < j \leq k, v \in N[u]\},$$

$$E_8 = \{e[i, j, x]f[i, j, u, v] : 1 \leq i < j \leq k, x \neq u, x \notin N[v]\},$$

$$E_9 = \{f[i, j, u, v]f[i, j, x, y] : 1 \leq i < j \leq k, u \neq x \text{ or } v \neq y\}.$$

An overview of this construction and a (partial) example are given in Figures 11.1 and 11.2.

Suppose C is a perfect code of size k' in H . Since $a[1]$ and $a[2]$ are pendant vertices attached to $a[0]$, neither vertex belongs to C because both cannot belong to C , and if only one belongs to C , then C fails to be a dominating set. It follows that $a[0] \in C$. This implies that none of the vertices in V_2 and V_3 belong to C (V_3 would kill V_2), and it implies also that exactly one vertex in each of the cliques formed by the edges of E_5 belongs to C (to cover V_3). Note that each of these k -cliques has n vertices indexed by V , the vertex set of G (this is the *selection* gadget). Let I be the set of vertices of G corresponding to the elements of C in these cliques. We argue that I is an independent set of order k in G .

Suppose $u, v \in I$ and that $uv \in E$. Then, there are indices $i < j$ between 1 and k such that (without loss of generality) $d[i, u] \in C$ and $d[j, v] \in C$. By the definition of E_6 and E_7 , each of these vertices is adjacent to $e[i, j, u]$,

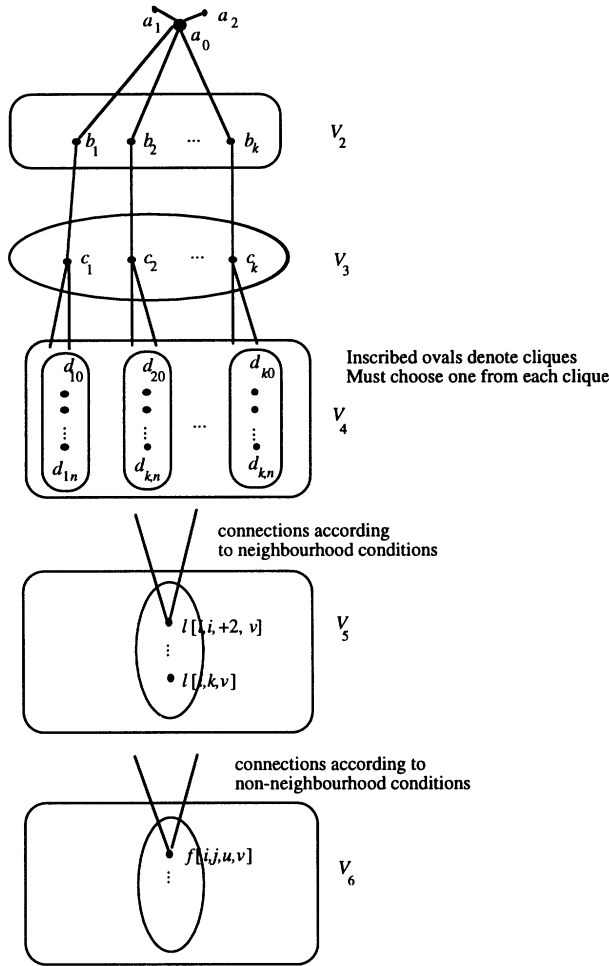


FIGURE 11.1. Overview of **INDEPENDENT SET** $\leq_m^s$ **PERFECT CODE**

which contradicts that C is a perfect code in H . Thus, I is an independent set in G .

Conversely, we argue that if $J = \{u_1, \dots, u_k\}$ is a k -element independent set in G , then H has a perfect code C_J of size k' . We may take C_J to be the following set of vertices:

$$C_J = \{a[0]\} \cup \{d[i, u_i] : 1 \leq i \leq k\} \cup \{f[i, j, u_i, u_j] : 1 \leq i, j \leq k\}.$$

That C_J is a perfect code can be verified directly from the definition of H .

11.0.4. (Downey and Fellows [170]) Prove that **PERFECT CODE** $\leq_m^s$ **SIZED SUBSET SUM** and, hence, **SIZED SUBSET SUM** is hard for $W[1]$. (See Exercise 3.)

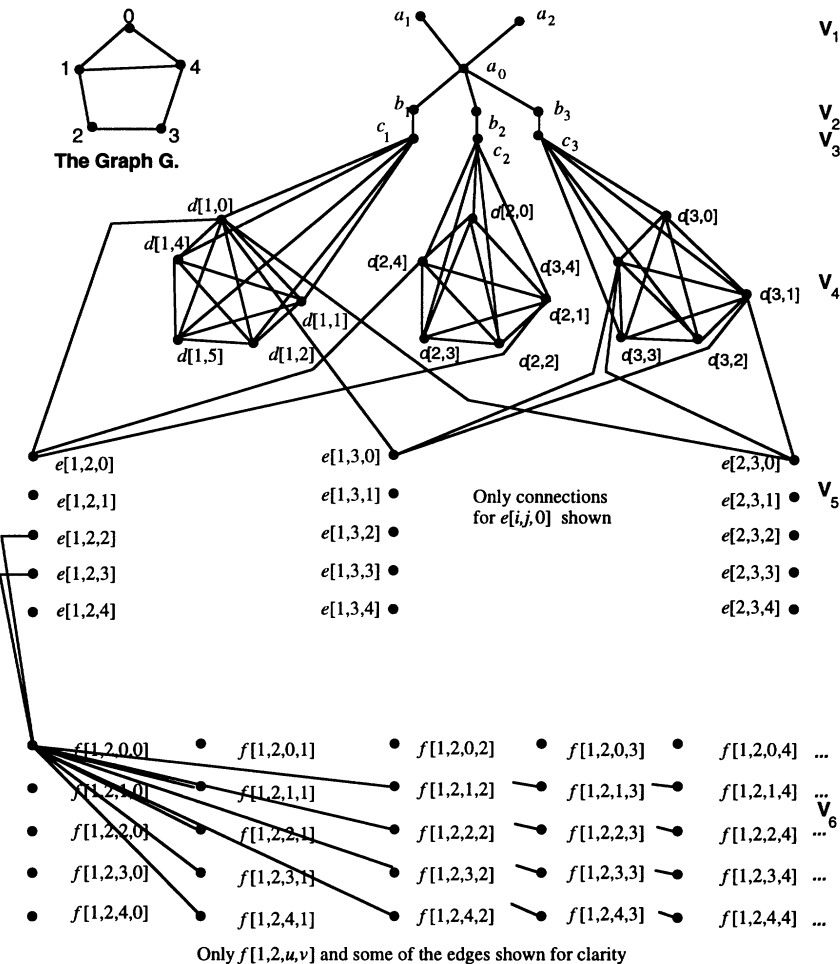


FIGURE 11.2. Example of $\text{INDEPENDENT SET} \leq_m^s \text{PERFECT CODE}$

SIZED SUBSET SUM

- Input:* A set $X = \{x_1, \dots, x_n\}$ of integers and a positive integer S .
- Parameter:* A positive integer k .
- Question:* Do there exist k elements of X that sum to S ?

[Hint: Let $G = (V, E)$ be a graph for which we wish to determine whether G has a perfect code of size k . Suppose for convenience that the vertex set of the graph $V = \{0, \dots, n - 1\}$. We can easily compute the list of positive integers $L = (x[i, j] : 1 \leq i \leq k, 0 \leq j \leq n - 1)$ and the positive integer

M , where

$$x[i, j] = (k+1)^{n+k-i} + \sum_{u \in N[j]} (k+1)^u,$$

$$M = \sum_{t=0}^{n+k-1} (k+1)^t$$

such that L has a sublist of size k summing to M if and only if G has a k -element perfect code. The correctness of this transformation is easily observed if the numbers of L are represented in base $k+1$, and it is noted that there can be no carries in a sum of k integers from L expressed in this way.]

11.0.5. (Downey and Fellows [170]) Consider the problem

SHORT CHEAP TOUR

- Instance:* A weighted graph and a positive integer S .
Parameter: A positive integer k .
Question: Is there a tour through at least k vertices of cost at most S ?

The precise difficulty of this problem is, at present, open but a variation is hard for $W[1]$. Let **SHORT EXACT TOUR** be the same as **SHORT CHEAP TOUR** except that we ask that the tour costs *exactly* S . Prove that **EXACT CHEAP TOUR** is hard for $W[1]$.

[Hint: Reduce **SIZED SUBSET SUM** to **EXACT CHEAP TOUR**.]

11.0.6. (Short Algebraic Factorizations) (Cai, Chen, Downey, and Fellows [109]) Prove that the following problem is hard for $W[1]$ via a reduction from **PERFECT CODE** above.

PERMUTATION GROUP FACTORIZATION

- Instance:* A set A of permutations $A \subseteq S_n$, and $x \in S_n$.
Parameter: A positive integer k .
Question: Does x have a factorization of length k over A ?

[Hint: Let $G = (V, E)$ be a graph of order n for which we wish to determine whether there is a perfect code of size k . Let $n' = (k+1)n$. We show how to produce an equivalent factorization problem instance for $S_{n'}$. View n' as divided into n blocks of size $k+1$, with these blocks in 1:1 correspondence with the vertices of G . Let γ denote a cyclic permutation of the elements of a block. Our set A consists of n permutations, one for each vertex of G . For a vertex u and with $N[u]$ denoting the solid neighborhood of u in G , let a_u denote the element of $S_{n'}$ which acts on the blocks corresponding to $v \in N[u]$ according to γ , and which is the identity map on all other blocks. The permutation x to be factored consists of the permutation γ on each of the n blocks.]

12

The W -Hierarchy

The defining problem for the “ W -Hierarchy” introduced in the present chapter, is the **WEIGHTED WEFT t DEPTH h CIRCUIT SATISFIABILITY** problem, $WCS(t, h)$ (Definition 10.1). The reader should recall that this problem takes as input a decision circuit C with large gate depth t and bounded depth h and asks if C accepts a weight k input vector. This problem was introduced as a generalization of the problem **WEIGHTED 3SAT**. Indeed, in Chapter 10, we proved that in the case $t = 1$, there is no difference between the parameterized power of weft t circuits and boolean expressions of logical depth t , in the sense that we proved that **WEIGHTED 3SAT** is $W[1]$ -complete.

Now, as noted in Example 9.1, *classically*, there is no difference in the computational complexity of determining satisfiability of classes of propositional formula no matter what their form (3CNF, CNF, or polynomial sized). However, *parameterized* reductions tend to be much more *structure preserving* than classical reductions, and certainly most classical reductions, such as the reduction between **CNF SATISFIABILITY** and **3SAT**, are definitely not parameterized reductions. In fact, we believe that there is *no* parameterized reduction to **WEIGHTED 3SAT** from **WEIGHTED CNF SATISFIABILITY**. We offer some evidence for this claim in the next chapter. It seems to us that the situation is the following. Classical reductions are so coarse that most natural problems in, say, NP that are apparently not in P seem to be NP -complete. Parameterized reductions are sufficiently refined that instead of one large class of naturally intractable problems all of the same complexity, there seem to be many sets of natural combinatorial problems, all intractable in the parameterized sense, and yet of differing parameterized complexity.

In this chapter, we will describe a hierarchy of classes of problems of increasing complexity. This hierarchy is based on the belief that **WEIGHTED CNF SATISFIABIL-**

ITY and WEIGHTED 3SAT are of differing complexity, and the underlying reason for this differing complexity is the difference in logical depth of the formula. Thus, if CNF is thought of as products-of-sums of literals, then we can define a propositional formula to be t -normalized as follows.

Definition 12.1 We say that a propositional formula X is t -normalized if X is of the form products-of-sums-of-products . . . of literals with t -alternations.

Thus, we see, according to Definition 12.1, 2-normalized is the same as CNF. Now we believe that for all t , WEIGHTED t -NORMALIZED SATISFIABILITY, that is, satisfiability for t -normalized formulas, is strictly easier than WEIGHTED $t + 1$ -NORMALIZED SATISFIABILITY. This suggests a hierarchy based on the complexity of t -normalized formulas. As in Chapter 10, we can consider t -normalized formulas as special examples of weft t circuits, namely the weft t boolean circuits. Again, we are able to prove that weft is, indeed, the key notion that determines computational power. The reader should recall from Definition 10.2 that

$$L \in W[t] \text{ iff } L \text{ reduces to } \text{WCS}(t, h).$$

Definition 12.2 (The W -Hierarchy) We term the union of these $W[t]$ classes together with two other classes $W[\text{SAT}] \subseteq W[P]$, the W -Hierarchy. Here, $W[P]$ denotes the class obtained by having no restriction on depth, i.e., P -size circuits, and $W[\text{SAT}]$ denotes the restriction to boolean circuits of P -size. Hence, the W -hierarchy is

$$FPT \subseteq W[1] \subseteq W[2] \subseteq \cdots \subseteq W[\text{SAT}] \subseteq W[P].$$

We conjecture that each of the containments is proper.

We will explore $W[\text{SAT}]$ and $W[P]$ in Chapter 13. The remainder of the present chapter will be devoted to establishing some results about the $W[t]$ classes for $t \geq 2$. The first is a higher-level version of Cook's theorem, the *Normalization Theorem* below. Armed with the Normalization Theorem, we will then establish some concrete hardness results.

Theorem 12.3 (The Normalization Theorem, Downey and Fellows [165, 166, 170]) For all $t \geq 1$, WEIGHTED t -NORMALIZED SATISFIABILITY is complete for $W[t]$.

We remark that the Normalization Theorem plays a role in our theory analogous to Cook's Theorem for NP -completeness, in the following sense. Usually, membership of a particular $W[t]$ class is easy so the circuit definition is easy to reduce to, whereas the t -normalized formulas provide problems that are easy to reduce from to establish hardness. The other view of the Cook(-Levin) Theorem is that it connects a generic problem from Turing machines to SATISFIABILITY; hence, SAT-

ISFIABILITY is very unlikely to be tractable. This other view of the Cook(-Levin) Theorem was realized in Theorem 10.3.

A very important part of the proof of the Normalization Theorem consists of the “hardwiring” of the gadget in the following key combinatorial reduction (Lemma 12.4). (None of the techniques used in the case $t = 1$ of Chapter 2 pertain to $t \geq 2$ or conversely.) The natural parameterized version of the **DOMINATING SET** problem is the following.

DOMINATING SET

Input: A graph G .
Parameter: A positive integer k .
Question: Does G have a dominating set of size k ? A dominating set for G is a set $X \subseteq V(G)$ such that for all $y \in V(G)$, there is an $x \in X$ with $\langle x, y \rangle \in E(G)$.

Lemma 12.4 **WEIGHTED CNF SATISFIABILITY** $\equiv_m^s$ **DOMINATING SET**.

Proof. Let X be a boolean expression in conjunctive normal form consisting of m clauses $C_1, \dots, C_m$ over the set of n variables $x_0, \dots, x_{n-1}$. We show how to produce in polynomial time by local replacement, a graph $G = (V, E)$ that has a dominating set of size $2k$ if and only if X is satisfied by a truth assignment of weight k .

A diagram of the gadget used in the reduction is given in Figure 12.1. The idea of the proof is as follows. There are k of the gadgets arranged in a vertical line. Each of the gadgets has three main parts. Taken from top to bottom, these are variable selection, gap selection, and gap and order enforcement. The variable selection component $A(r)$ is a clique and the gap selection component $B(r)$ consists of n -cliques which we call columns. Our first action is to ensure that in *any* dominating set of $2k$ elements, we must pick one vertex from each of these two components. This goal is achieved by the $2k$ sets of $2k + 1$ enforcers, vertices from V_4 and V_5 . (The names refer to the sets below.) Take the V_4 , for instance. For a fixed r , these $2k + 1$ vertices are connected to all of the variable selection vertices in the component $A(r)$, and nowhere else. Thus, if they are to be dominated by a $2k$ dominating set, then we must choose *some* element in the set $A(r)$, and similarly, we must choose an element in the set $B(r)$ by virtue of the V_5 enforcers. Since we will need exactly $2k$ (or even $\leq 2k$) dominating elements, it follows that we must pick *exactly* one from each of the $A(r)$ and $B(r)$ for $r = 1, \dots, k$.

As the name suggests these will be picked by the variable selection components, $A(r), r = 0, \dots, k-1$. Each of these k components consists of a clique of n vertices labeled $0, \dots, n-1$. The intention being that the vertex labeled i represents a choice of variable i being made true in the formula X . Correspondingly, in the next $B(r)$, we have columns (cliques) $i = 0, \dots, n-1$. The intention is that column i corresponds to the choice of variable i in the preceding $A(r)$. The idea then is the following. We join the vertex $a[r, i]$ corresponding to variable i , in $A(r)$, to all vertices in $B(r)$ *except* those in column i . This means that the choice of i in $A(r)$ will cover all vertices of $B(r)$ except those in this column. It follows

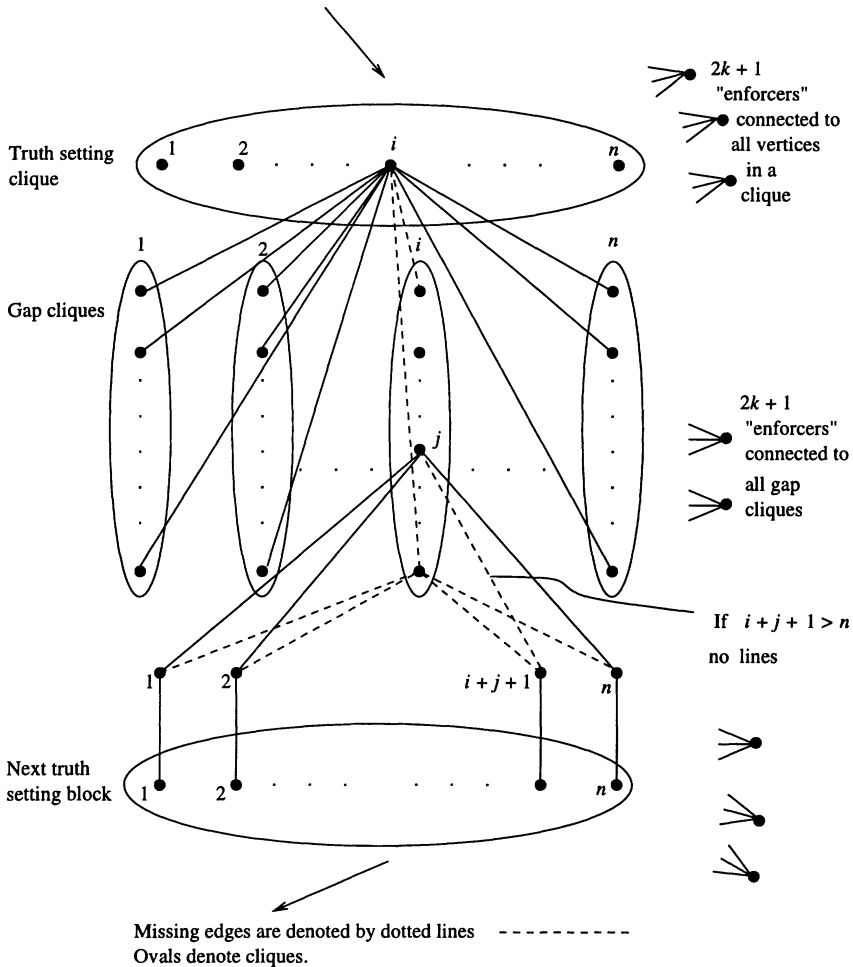


FIGURE 12.1. Gadget for CNFSAT Reducing to DOMINATING SET

that we *must* choose the dominating element from this column and nowhere else. (There are no connections from column to column.) The columns are meant to be the gap selection saying how many 0's there will be until the next positive choice for a variable. We finally need to ensure that (i) if we choose variable i in $A(r)$ and gap j in column i of $B(r)$ then we need to pick $i + j + 1$ in $A(r + 1)$ and (ii) that the selections are in order. This is the role of the gap and order enforcement component which consists of a set of n vertices (in V_6 .) For $r < k - 1$, the method is to connect vertex j in column i of $B(r)$ to all of the n vertices $d[r, s]$ except to $d[r, i + j + 1]$ provided that $i + j + 1 \leq n$. (For $r = k - 1$, we simply connect vertex j in column i of $B(r)$ to all of the n vertices $d[r, s]$ except to $d[r, i + j + 1]$, since this will need to "wrap around" to $A(0)$.) The first point of this process is that if

we choose vertex j in column i with $i + j + 1 > n$, then *none* of the vertices of the enforcement line are dominated. Since the only other edge from a gap enforcement vertex is the single edge to the corresponding vertex in $A(r + 1)$, this means that no size $2k$ dominating set is compatible with such a choice. It follows that we must choose some vertex j with $i + j + 1 \leq n$ from column i . The point of this is that if we choose vertex j in column i , then we will dominate all of the $d[r, s]$ except $d[r, i + j + 1]$. Since we will only connect $d[r, s]$ additionally to $a[r + 1, s]$ and nowhere else, to choose an element of $A[r + 1]$ and still dominate all of the $d[r, s]$ we must actually choose $a[r + 1, i + j + 1]$.

Thus, the above provides a selection gadget that chooses k true variables with the gaps representing false ones. We enforce that the selection is consistent with the clauses of X via the clause vertices V_3 . These are connected in the obvious ways. One connects a choice in $A[r]$ or $B[r]$ corresponding to making a clause C_q true to the vertex $c[q]$. Then, if we dominate all the clause vertices too, we must have either chosen in some $A[r]$, a positive occurrence of a variable in C_q or we must have chosen in $B[r]$, a gap corresponding to a negative occurrence of a variable in C_q , and conversely. We now turn to the formal details.

The vertex set V of G is the union of the following sets of vertices:

$$\begin{aligned} V_1 &= \{a[r, s] : 0 \leq r \leq k - 1, 0 \leq s \leq n - 1\}, \\ V_2 &= \{b[r, s, t] : 0 \leq r \leq k - 1, 0 \leq s \leq n - 1, 1 \leq t \leq n - k + 1\}, \\ V_3 &= \{c[j] : 1 \leq j \leq m\}, \\ V_4 &= \{a'[r, u] : 0 \leq r \leq k - 1, 1 \leq u \leq 2k + 1\}, \\ V_5 &= \{b'[r, u] : 0 \leq r \leq k - 1, 1 \leq u \leq 2k + 1\}, \\ V_6 &= \{d[r, s] : 0 \leq r \leq k - 1, 0 \leq s \leq n - 1\}. \end{aligned}$$

For convenience, we introduce the following notation for important subsets of some of the vertex sets above. Let

$$\begin{aligned} A(r) &= \{a[r, s] : 0 \leq s \leq n - 1\}, \\ B(r) &= \{b[r, s, t] : 0 \leq s \leq n - 1, 1 \leq t \leq n - k + 1\}, \\ B(r, s) &= \{b[r, s, t] : 1 \leq t \leq n - k + 1\}. \end{aligned}$$

The edge set E of G is the union of the following sets of edges. In these descriptions, we implicitly quantify over all possible indices.

$$\begin{aligned} E_1 &= \{c[j]a[r, s] : x_s \in C_j\}, \\ E_2 &= \{a[r, s]a[r, s'] : s \neq s'\}, \\ E_3 &= \{b[r, s, t]b[r, s, t'] : t \neq t'\}, \\ E_4 &= \{a[r, s]b[r, s', t] : s \neq s'\}, \\ E_5 &= \{b[r, s, t]d[r, s'] : s + t + 1 \leq n \wedge s' \neq s + t\} \cup \{b[k - 1, s, t]d[k - 1, s'] : \\ &\quad s' \neq s + t \pmod{n}\}, \\ E_6 &= \{a[r, s]a'[r, u]\}, \\ E_7 &= \{b[r, s, t]b'[r, u]\}, \\ E_8 &= \{c[j]b[r, s, t] : \exists i \, \overline{x_i} \in C_j, s < i < s + t\}, \\ E_9 &= \{d[r, s]a[r', s] : r' = r + 1 \pmod{k}\}. \end{aligned}$$

Suppose X has a satisfying truth assignment τ of weight k , with variables $x_{i_0}, x_{i_1}, \dots, x_{i_{k-1}}$ assigned the value *true*. Suppose $i_0 < i_1 < \dots < i_{k-1}$. Let $d_r = i_{r+1 \pmod k} - i_r \pmod n$ for $r = 0, \dots, k-1$. It is straightforward to verify that the set of $2k$ vertices

$$D = \{a[r, i_r] : 0 \leq r \leq k-1\} \cup \{b[r, i_r, d_r] : 0 \leq r \leq k-1\}$$

is a dominating set in G .

Conversely, suppose D is a dominating set of $2k$ vertices in G . The closed neighborhoods of the $2k$ vertices $a'[0, 1], \dots, a'[k-1, 1], b'[0, 1], \dots, b'[k-1, 1]$ are disjoint, so D must consist of exactly $2k$ vertices, one in each of these closed neighborhoods. Also, none of the vertices of $V_4 \cup V_5$ are in D , since if $a'[r, u] \in D$, then necessarily $a'[r, u'] \in D$ for $1 < u' < 2k+1$ (otherwise D fails to be dominating), which contradicts that D contains exactly $2k$ vertices. It follows that D contains exactly one vertex from each of the sets $A(r)$ and $B(r)$ for $0 \leq r \leq k-1$.

The possibilities for D are further constrained by the edges of E_4 , E_5 , and E_9 . The vertices of D in V_1 represent the variables set to *true* in a satisfying truth assignment for X , and the vertices of D in V_2 represent intervals of variables set to *false*. Since there are k variables to be set to *true*, there are, considering the indices of the variables mod n , also k intervals of variables to be set to *false*. Furthermore, the set E_5 forces the chosen variables to be chosen so that if $r < r'$ and we choose $a[r, q]$ and $a[r', q']$, then $q < q'$.

The edges of E_4 , E_5 , and E_9 enforce that the $2k$ vertices in D must represent such a choice consistently. To see how this enforcement works, suppose $a[3, 4] \in D$. This represents that the third of k distinct choices of variables to be given the value *true* is the variable x_4 . The edges of E_4 force the unique vertex of D in the set $B(3)$ to belong to the subset $B(3, 4)$. The index of the vertex of D in the subset $B(3, 4)$ represents the difference (mod n) between the indices of the third and fourth choices of a variable to receive the value *true*, and thus the vertex represents a range of variables to receive the value *false*. The edges of E_5 and E_9 enforce that the index t of the vertex of D in the subset $B(3, 4)$ represents the “distance” to the next variable to be set *true*, as it is represented by the unique vertex of D in the set $A(4)$.

It remains only to check that the fact that D is a dominating set ensures that the truth assignment represented by D satisfies X . This follows by the definition of the edge sets E_1 and E_8 .

Remark. A very important fact about the above proof is the following. For our fixed k , the enforcement gadgetry causes us to choose the $2k$ vertices, k from the $A(r)$ ’s and k from the $B(r)$ ’s, and there is a 1-1 correspondence between weight k assignments to X and size $2k$ sets that *dominate the graph G' which denotes the gadget, that is, G without the clause connections and vertices*. Define a *weak dominating set* to be a set of $(2k)$ vertices that dominates the gadget of G . The fact we will use in the proof of the normalization theorem is that under the 1-1 correspondence between weight k assignments to X and weak dominating sets, if clauses $C_{i_1}, \dots, C_{i_p}$ are the subset of the set of clauses satisfied by some weight

k assignment, then $c_{i_1}, \dots, c_{i_p}$ are exactly the clause vertices dominated by the corresponding size $2k$ weak dominating set in G . That is, not only is there a correspondence between weight k satisfying assignments for X and weight $2k$ dominating sets in G , but in fact there is an exact correspondence between *all* weight k assignments together with the clauses they satisfy, and with *all* weak dominating sets and the clause vertices they dominate.

Corollary 12.5 (i) **MONOTONE CNF SATISFIABILITY** $\equiv_m^s$ **CNF SATISFIABILITY**.
(ii) **WEIGHTED CNF SATISFIABILITY** $\leq_m^s$ **WEIGHTED BINARY INTEGER PROGRAMMING**

Proof. (i) See Exercise 1. (ii) See Theorem 9.5.

Proof of the Normalization Theorem. Let $L \in W[t]$. By Theorem 10.3, we can suppose that $t \geq 2$. Let $\mathcal{F}$ be the family of circuits of depth bounded by h and weft bounded by t to which L reduces. It suffices to reduce $L_{\mathcal{F}}$ to **WEIGHTED t -NORMALIZED SATISFIABILITY**. An instance of the latter problem may be viewed as a pair consisting of a positive integer k and a circuit having t alternating layers of *And* and *Or* gates corresponding to the t -normalized expression structure P-o-S-o-P-... and having a single output *And* gate. Thus, the argument essentially shows how to “normalize” the circuits in $\mathcal{F}$.

Let (C, k) be an instance of $L_{\mathcal{F}}$. We show how to determine whether C accepts a weight k input vector, by consulting an oracle for **WEIGHTED t -NORMALIZED SATISFIABILITY** (viewed as a problem about circuits) for finitely many weights k' . The algorithm for this determination will be uniform in k and run in time $f(k)n^\alpha$, where n is the size of the circuit C . The exponent α will be a (possibly exponential) function of h and t . This is permissible, since every circuit in $\mathcal{F}$ observes these bounds on depth and weft.

Step 1. The reduction to tree circuits.

The first step is to transform C into a *tree circuit* C' (or *formula*) of depth and weft bounded by h and t , respectively. In a tree circuit, every logic gate has fanout one. (The input nodes may have large fanout.) The transformation is accomplished by replicating the portion of the circuit above a gate as many times as the fanout of the gate, beginning with the top level of logic gates, and proceeding downward level by level. (We regard a decision circuit as arranged with the inputs on top and the output on the bottom.) The creation of C' from C may require time $O(n^{O(h)})$ and involve a similar blowup in the size of the circuit. The tree circuit C' accepts a weight k input vector if and only if the original circuit C accepts a weight k input vector.

Step 2. Moving the *not* gates to the top of the circuit.

Let C denote the circuit we receive from the previous step (we will use this notational convention throughout the proof). Transform C into an equivalent circuit C' by commuting the *not* gates to the top, using DeMorgan's laws. This may increase the size of the circuit by at most a constant factor. The tree circuit C' thus consists (from the top) of the input nodes, with *not* gates on some of the lines fanning out from the inputs. In counting levels, we consider all of this as level 0

and may refer to negated fanout lines from the input nodes as negated inputs. Next, there are levels consisting only of large and small *and* and *or* gates, with a single output gate (which may be of either principal logical denomination at this point).

Step 3. Homogenizing the layers.

The goal of this step is to reduce to the situation where all of the large gates are at the bottom of the circuit, in alternating layers of large *And* and *Or* gates. To achieve this, we work from the bottom up, with the first task being to arrange for the output gate to be large.

Let C denote the circuit received from the previous step. Suppose the output gate z is small. Let $C[z]$ denote the connected component of C including z that is induced by the set of small gates. Thus, all gates providing input to $C[z]$ are either large or are input gates of C . Because of the bound h on the depth of C , there are at most 2^h inputs to $C[z]$. The function of these inputs computed by $C[z]$ is equivalent to a product-of-sums expression E_z having at most 2^h sums, with each sum a product of at most 2^h inputs. Let C' denote the circuit equivalent to C obtained by replacing the small gate output component $C[z]$ with E_z , duplicating subcircuits of C as necessary to provide the inputs to the depth 2 circuit representing E_z . (The “product” gate of E_z is now the output gate of C' .) This entails a blowup in size by a factor bounded by 2^{2^h} . Since h is an absolutely fixed constant (not dependent on n or k), this blowup is “linear” and permitted. Note that E_z and therefore C' are easily computed in a similar amount of time to this size blowup.

Let p denote the output *and* gate of C' (corresponding to the product in E_z). Let $s_1, \dots, s_m$ denote the *or* gates of C' corresponding to the sums of E_z . We consider all of these gates to be *small*, since the number of inputs to them does not depend on n or k . (Equivalently, if the gates of these two levels were replaced by binary input gates, we would see that the reduction of C to C' has increased circuit depth from h to 2^h .)

Each *or* gate s_i of C' has three kinds of input lines: those coming from large *And* gates, those coming from large *Or* gates, and those coming from input gates of C' . We will use the same symbol to denote an input line, the subcircuit of C' that computes the value on that line, or the boolean expression corresponding to the subcircuit (since C' is a tree circuit, it is equivalent to a boolean expression). Let these three groups of inputs be denoted

$$\begin{aligned} S_{i,\wedge} &= \{s_i[\wedge, j] : j = 1, \dots, m_{i,\wedge}\}, \\ S_{i,\vee} &= \{s_i[\vee, j] : j = 1, \dots, m_{i,\vee}\}, \\ S_{i,\top} &= \{s_i[\top, j] : j = 1, \dots, m_{i,\top}\}, \end{aligned}$$

and define

$$S_i = S_{i,\wedge} \cup S_{i,\vee} \cup S_{i,\top},$$

For each line $s_i[\vee, j]$ of C' coming from a large *Or* gate u , let

$$S_{i,\vee,j} = \{s_i[\vee, j, k] : k = 1, \dots, m_{i,\vee,j}\}$$

denote the set of input lines to u in C' . Similarly, for each line $s_i[\wedge, j]$ of C' coming from a large *And* gate v , let

$$S_{i,\wedge,j} = \{s_i[\wedge, j, k] : k = 1, \dots, m_{i,\wedge,j}\}$$

denote the set of input lines to v in C' .

Let

$$k' = \sum_{i=1}^m (1 + m_{i,\vee}).$$

The integer k' is the number of *or* gates (counting both large and small gates) that are either part of $C[z]$ or directly supply input to $C[z]$. Note that k' is bounded above by $2^h \cdot 2^{2^h}$.

We describe how to produce a weft t circuit C'' from C' that accepts an input vector of weight $k'' = k + k'$ if and only if C' (and therefore C) accepts an input vector of weight k . The tree circuit C'' will have a large *And* gate giving the output.

Let $x_1, \dots, x_n$ denote the inputs to C' . The circuit C'' has additional input variables that, for the most part, correspond to the input lines to the *or* gates singled out for attention above. The set V of new input variables is the union of the following groups of variables:

$$V = \left(\bigcup_{i=1}^m V_i \right) \cup \left(\bigcup_{i=1}^m \bigcup_{j=1}^{m_{i,\vee}} V_{i,j} \right),$$

where

$$V_i = \{v_i[\wedge, j] : 1 \leq j \leq m_{i,\wedge}\} \cup \{v_i[\vee, j] : 1 \leq j \leq m_{i,\vee}\} \cup \{v_i[\top, j] : 1 \leq j \leq m_{i,\top}\}$$

and

$$V_{i,j} = \{v_i[\vee, j, k] : 1 \leq k \leq m_{i,\vee,j}\} \cup \{n[i, j]\},$$

The circuit C'' is represented by the boolean expression

$$C'' = E_1 \cdot E_2 \cdot E_3 \cdot E_4 \cdot E_5 \cdot E_6 \cdot E_7,$$

where

$$\begin{aligned} E_1 &= \prod_{i=1}^m \left(\sum_{u \in V_i} u \right), \\ E_2 &= \prod_{i=1}^m \prod_{u \neq v, u, v \in V_i} (\neg u + \neg v), \\ E_3 &= \prod_{i=1}^m \prod_{j=1}^{m_{i,\vee}} \left(\sum_{u \in V_{i,j}} u \right), \end{aligned}$$

$$\begin{aligned}
E_4 &= \prod_{i=1}^m \prod_{j=1}^{m_{i,\vee}} \prod_{\substack{u,v \in V_{i,j} \\ u \neq v}} (\neg u + \neg v), \\
E_5 &= \prod_{i=1}^m \prod_{j=1}^{m_{i,\wedge}} \prod_{k=1}^{m_{i,\wedge,j}} (s_i[\wedge, j, k] + \neg v_i[\wedge, j]), \\
E_6 &= \prod_{i=1}^m \prod_{j=1}^{m_{i,\vee}} (\neg v_i[\vee, j] + \neg n[i, j]), \\
E_7 &= \prod_{i=1}^m \prod_{j=1}^{m_{i,\vee}} \prod_{k=1}^{m_{i,\vee,j}} (s_i[\vee, j, k] + \neg v_i[\vee, j, k]).
\end{aligned}$$

The size of C'' is bounded by $|C'|^2$.

Claim 1. The circuit C'' has weft t .

To see this, note first of all that since $t \geq 2$, any input-output path beginning from a new input variable (in V) has at most two large gates as the expression for C'' is essentially a product-of-sums. In E_5 and E_7 , the sums involve subexpressions of C' ; any input-output path from an original input variable (of C') passes through one of these *or* gates. Observe that in C' , these subexpressions have weft at most $t - 1$. The sums of E_5 and E_7 are small, so these paths further encounter only the bottommost large *And* gate.

Claim 2. The circuit C'' accepts an input vector of weight k'' if and only if C' accepts an input vector of weight k .

First, note that any input vector of weight k'' accepted by C'' must set exactly one variable in each of the sets of variables V_i (for $i = 1, \dots, m$) and $V_{i,j}$ (for $i = 1, \dots, m$ and $j = 1, \dots, m_{i,\vee}$) to 1 and all of the others in the set to 0 in order to satisfy $E_1 \cdot E_2 \cdot E_3 \cdot E_4$. It follows that any such accepted input must set exactly k of the original variables of C' to 1, by the definition of k'' .

The role of the (new) variables set to 1 in the sets of variables that represent inputs to *or* gates is to indicate an accepting computation of C' on the weight k input of old variables. The expressions $E_5, \dots, E_7$ enforce the correctness of this representation in C'' of the computation of C' .

The expression E_5 ensures that if the new variable $v_i[\wedge, j]$ is set to 1, indicating that the subexpression $s_i[\wedge, j]$ of C' evaluates to 1, then every argument $s_i[\wedge, j, k]$ must evaluate to 1. (Note that subexpressions $s_i[\wedge, j, k]$ appear in C'' whereas the subexpressions $s_i[\wedge, j]$ do not. The computations performed in C' by the latter are simply represented by the values of the input variables in V .) The role of the variables $n[i, j]$ is to represent that “none of the inputs” to the *or* gate has the value 1. The expression E_6 enforces that if this situation is represented, then the output of the gate is not represented as having the value 1. The expression E_7 ensures that if the new variable $v_i[\vee, j, k]$ has the value 1, indicating that the subexpression $s_i[\vee, j, k]$ of C' evaluates to 1, then this subexpression must, in fact, evaluate to 1.

By the above, we may now assume that the circuit with which we are working has a large output gate (which may be of either denomination). Renaming

for convenience, let C denote the circuit with which we are working under this assumption.

If g and g' are gates in C of the same logical character ($\wedge$ or $\vee$) with the output of g going to g' , then they can be consolidated into a single gate without increasing weft if g is small and g' is large. We term this a *permitted contraction*. Note that if g is large and g' is small, then contraction may not preserve weft. We will assume that permitted contractions are performed whenever possible, interleaved with the following two operations.

(1) *Replacement of bottommost small gate components.*

As at the beginning of Step 3, let $C_1, \dots, C_m$ denote the bottommost connected components of C induced by the set of small gates, and having at least one large gate input. Since the output gate of C is large, each C_i gives output to a large gate g_i . If g_i is an *And* gate, then C_i should be replaced with product-of-sums circuitry equivalent to C_i . If g_i is an *Or* gate, then C_i should be replaced with equivalent sum-of-products circuitry. Note that in either case, this immediately creates the opportunity for a permitted contraction. As per the discussion at the beginning of Step 3, this replacement circuitry is small, and this operation may increase the size of the circuit by a factor of 2^{2^h} . This step will be repeated at most h times, as we are working from the bottom up in transforming C .

(2) *Commuting small gates upward.*

After (1), and after the permitted contractions, the bottommost small gate components are each represented in the modified circuit C' by a single small gate h_i giving output to g_i . Without loss of generality, all of the arguments to h_i may be considered to come from large gates. (The only other possibility is that an input argument to h_i may be from the input level of the circuit—but there is no increase in weft in treating this as an honorary large gate for convenience.) Suppose that g_i is an *And* gate and that h_i is an *or* gate (the other possibility is handled dually).

There are three possible cases:

- (i) All of the arguments to h_i are from *Or* gates.
- (ii) All of the arguments to h_i are from *And* gates.
- (iii) The arguments to h_i include both large *Or* gates and large *And* gates.

In case (i), we may consolidate h_i and all of the gates giving input to h_i into a single large *Or* gate without increasing weft.

In case (ii), we replace the small $\vee$ (h_i) of large $\wedge$'s with the equivalent (by distribution) large $\wedge$ of small $\vee$'s. Since h_i may have 2^h inputs, this may entail a blowup in the size of the circuit from n to n^{2^h} . This does not increase weft, and creates the opportunity for a permitted contraction.

In case (iii), we similarly replace h_i and its argument gates with circuitry representing a product-of-sums of the inputs to the arguments of h_i . The difference is that in this case, the replacement is a large $\wedge$ of large (rather than small) $\vee$ gates. Weft is preserved when we take advantage of the contraction now permitted between the large $\wedge$ gate and g_i .

We may achieve our purpose in this step by repeating the cycle of (1) and (2). At most, h repetitions are required. The total blowup in the size of the circuit in this step is crudely bounded by $n^{2^{h^2}}$.

Step 4. Removing a bottommost *Or* gate.

By a Turing reduction, we can determine whether a tree circuit giving output from an *Or* gate accepts a weight k input vector, by simply making the same determination for each of the input branches (subformulas) to the gate.

In order to accomplish this step by a many:1 reduction, we do the following. Let b be the number of branches of the circuit C with bottommost *Or* gate that we receive at the beginning of this step. We modify C by creating new inputs $x[1], \dots, x[b]$. The purpose of these input variables is to indicate which branch of C accepts a weight k input vector. Let $C_1, \dots, C_b$ be the branches of C , so that C is represented by the expression $C_1 + \dots + C_b$. The modified circuit C' is represented by the expression

$$C' = (x[1] + \dots + x[b]) \cdot \prod_{1 \leq i < j \leq b} (\neg x[i] + \neg x[j]) \cdot \prod_{1 \leq i \leq b} (C_i + \neg x[i]).$$

The first two product terms of the above expression ensure that exactly one of the new variables must have value 1 in an accepted input vector. The modified circuit C' accepts a weight $k + 1$ input vector if and only if C accepts a weight k input vector. For w at least 2, the transformation is *weight-preserving*, and yields a circuit C' with bottommost *And* gate, but possibly with *not* gates at the lower levels. Thus, it may be necessary to repeat Steps 2 and 3 to obtain a homogenized circuit with bottommost *And* gate.

Step 5. Organizing the small gates.

The tree circuit C received from the previous step has the properties that (i) the output gate is an *And* gate, (ii) from the bottom, the circuit consists of layers which alternately consist of only *And* gates or only *Or* gates, for up to t layers, and (iii) above this, there are branches B of height $h' = h - t$ consisting only of small gates. Since a small gate branch B has bounded depth, it has at most $2^{h'}$ gates, and thus in constant time (since h is fixed), we can find either (1) an equivalent sum-of-products circuit with which to replace B , as required by Case 1 of Step 6 below, or (2) an equivalent product-of-sums circuit, as required by Case 2 of Step 6.

In this step, all such small gate branches B of C are replaced in this way, appropriate for the relevant case of Step 6. In Case 1, the depth 2 sum-of-products circuits replacing the small gate branches B have a bottommost *or* gate g_B of fanin at most $2^{2^{h'}}$, and the *and* gates feeding into g_B have fanin at most $2^{h'}$, so the *weight* of the circuit has been preserved by this transformation, which may increase the size of C by the constant factor $2^{2^{h'}}$. The topmost level of large gates (to which the branches B are attached in C) consists of *Or* gates, in Case 1, so that the gates g_B can be merged into this topmost level. Merging is performed similarly in Case 2, where the replacing circuits are products-of-sums, and the topmost level of large gates consists of *And* gates. For the next step, we consider two cases, depending

on whether the topmost level of large gates consists of *Or* gates or *And* gates. (Essentially, this corresponds to whether weft t is even or odd.)

Step 6. A monotone change of variables (two cases).

In this step (in both cases), we employ a “change of variables” based on the combinatorial reduction of Lemma 12.4. The goal is to obtain an equivalent circuit that has the property that either all the inputs are **MONOTONE** (case 1) (i.e., no inverters in the circuit) or all the inputs are negated with no other inverters in the circuit, that is, **ANTIMONOTONE**. (Actually, in this case we will have *some* of the inputs positive, but these will only be *enforcers*, as we will see. So we should call this case **NEARLY ANTIMONOTONE**) (case 2). The point of this step becomes apparent in the next step when we use the special character of the circuit thus constructed to enable us to eliminate the small gates.

Consider the reduction of Lemma 12.4. This reduction consists of two parts. The first is the sequence of selection gadgets which allow variable choice, gap choice, and then gap enforcement, and the second part is consistency obtained by clause wiring. The idea is to “hard wire” the selection and consistency parts of the construction into the circuit. The point being that we can replace positive instances of variable fanout in the original circuit by outputs corresponding to choice of that variable in the positive selection component. We can replace negative fanout in the original circuit by the appropriate sets of gap variables. Finally, we can wire in the fact that we need a dominating set and other enforcements by using the facts that we will only look at a weight $2k$ input, and an *And* of *Or*’s, which will not add to the weft of the circuit. We argue more precisely below, and also prove in two parts that the whole process can be accomplished without increasing weft, given that the weft is ≥ 2 .

Suppose the inputs to the circuit C received at the beginning of this step are $x[1], \dots, x[n]$, and suppose that the output gate of C is an *And* gate. Let Y denote the boolean expression having $2n$ clauses, with each clause consisting of a single literal and with one clause for each of the $2n$ literals of the n input variables. The reduction of Lemma 12.4 allows us to translate Y into a monotone formula via a dominating set, thus capturing monotonically all the relevant input settings. Let G_Y be the graph constructed for this expression as in the proof of Lemma 12.4. Note that only part of G_Y will actually be wired into C .

Keeping this in mind and using the variable (vertex) set obtained from G_Y , the change of variables is implemented for C as follows; (1) Create a new input for each vertex of G_Y that is not a clause vertex. (2) Replace each positive input fanout of $x[i]$ in C with an *Or* gate having k new input variable arguments corresponding to the vertices to which the clause vertex for the clause $(x[i])$ of Y is adjacent in G_Y . (3) Replace each negated fanout line of $x[i]$ with an *Or* gate having $O(n^2)$ new input variable arguments corresponding to the vertices to which the clause vertex for the clause $(\neg x[i])$ of Y is adjacent in G_Y . (4) Merge with the output *And* gate of C a new circuit branch corresponding to the product-of-sums expression, where the product is taken over all nonclause vertices of G_Y , and the sum for a vertex u is the sum of the new inputs corresponding to the nonclause vertices in $N[u]$ (this is the dominating set and other enforcements).

The modified circuit C' obtained in this way accepts a weight $2k$ input vector if and only if the original circuit C accepts a weight k input vector. The proof of this is essentially the same as for Lemma 12.4. (This is the whole point of the remark following the proof of Lemma 12.4.) If all of the *not* gates of C are at the top, then the circuit C' will be **MONOTONE**. However, to see that this change of variables can be employed to obtain a monotone or nearly antimonotone circuit *without* increasing weft, we must consider two cases.

Case 1. The topmost large-gate level consists of *Or* gates.

Let C denote the circuit obtained from Step 5 and perform a change of variables as described above. The sequence of transformations of C for this step is shown schematically in Figure 12.2.

The result is a circuit C' with no *not* gates. The input weight we are now concerned with is $2k$, and the construction of C' from C may involve quadratic blowup.

Next, we move the small *and* gates on the second level upward past the *Or* gates introduced by the change of variables, and then merge the *Or* gates down to the topmost large layer (of *Or* gates).

Case 2. The topmost large-gate level consists of *And* gates.

Here, we use a similar argument, beginning with a trick. Below each gate of the topmost large-gate level (of *And* gates), a double negation is introduced (equivalently). One of the *not* gates is moved to the top of the circuit (by DeMorgan's identities). This is followed by a change of variables based on Lemma 12.4, as in Case 1. The second-level *and* gates are commuted upward, and the *Or* gates of the second and third levels are merged, as in Case 1. So now, the circuit has no negated inputs and no inverters except the residual ones below the top layer of *Or* gates. Finally, these remaining *not* gates are commuted to the top. Note that this means that all fanouts are negated except the ones to the enforcement *Or* gate added during Step (4).

We are now in position for the last step.

Step 7. Eliminating the remaining small gates.

If we regard the inputs to C as variables, this step consists of another "change of variables." Let k be the relevant weight parameter value supplied by the last transformation. In this step, we will produce a circuit C' corresponding directly to a t -normalized boolean expression (that is, consisting only of t alternating layers of *And* and *Or* gates) such that C accepts a weight k input vector if and only if C' accepts a vector of weight $k' = k \cdot 2^{k+1} + 2^k$.

Suppose that C has m remaining small gates. In Case 1, these are *and* gates, and the inputs are all positive. In Case 2, these are *or* gates, and the inputs to these gates are all negated. For $i = 1, \dots, m$, we define the sets A_i of the inputs to C to be the sets of input variables to these small gates. The central idea for this step is to create new inputs representing the sets A_i of inputs to C .

For example, suppose (Case 1) that the output of the small *and* gate g_i in C is the boolean product ($abcd$) of the inputs a, b, c , and d to C . Thus, $A_i = \{a, b, c, d\}$. The gate g_i can be eliminated by replacing it with an input line from a new variable $v[i]$ which represents the predicate $a = b = c = d = 1$. (This representation, of

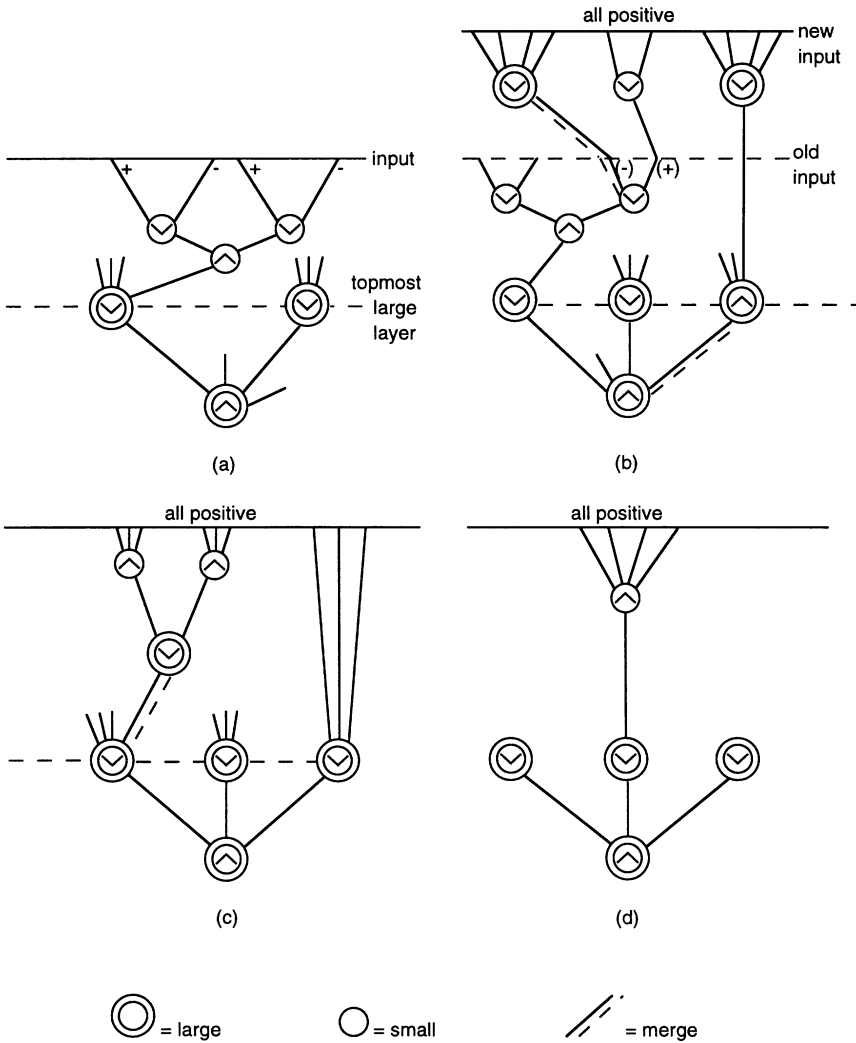


FIGURE 12.2. A Topmost Layer of Large Or Gates.

course, will need to be enforced by additional circuit structure.) Similarly (Case 2), if g_i computes the value $(\bar{a} + \bar{b} + \bar{c} + \bar{d})$, then g_i can be replaced by a negated input line from $v[i]$.

Let $x[j]$ for $j = 1, \dots, s$ be the input variables to C . We introduce new input variables of the following kinds:

- (1) One new variable $v[i]$ for each set A_i for $i = 1, \dots, m$ to be used as indicated above.
- (2) For each $x[j]$, we introduce 2^{k+1} copies $x[j, 0], x[j, 1], x[j, 2], \dots, x[j, 2^{k+1} - 1]$.
- (3) “Padding” consisting of 2^k meaningless variables (inputs not supplying output to any gates) $z[1], \dots, z[2^k]$.

We add to the circuit an enforcement mechanism for the change of variables. The necessary requirements can be easily expressed in P-o-S form, and, thus, can be incorporated into the bottom two levels of the circuit as additional *Or* gates attached to the bottommost (output) *And* gate of the circuit.

We require the following implications concerning the new variables:

- (1) The $s \cdot 2^{k+1}$ implications, for $j = 1, \dots, s$ and $r = 0, \dots, 2^{k+1} - 1$,

$$x[j, r] \Rightarrow x[j, r + 1 \pmod{2^{k+1}}]$$

- (2) For each containment $A_i \subseteq A_{i'}$, the implication

$$v[i'] \Rightarrow v[i].$$

- (3) For each membership $x[j] \in A_i$, the implication

$$v[i] \Rightarrow x[j, 0].$$

- (4) For $i = 1, \dots, m$, the implication

$$\left(\prod_{x[j] \in A_i} x[j, 0] \right) \Rightarrow v[i].$$

It may be seen that this transformation may increase the size of the circuit by a linear factor exponential in k . We make the following argument for the correctness of the transformation.

If C accepts a weight k input vector, then setting the corresponding copies $x[i, j]$ among the new input variables accordingly, together with appropriate settings for the new “collective” variables $v[i]$, yields a vector of weight between $k \cdot 2^{k+1}$ and $k \cdot 2^{k+1} + 2^k$ that is accepted by C' . The reason the weight of this corresponding vector may fall short of $k' = k \cdot 2^{k+1} + 2^k$ is that not all of the subsets of the k input variables to C having value 1 may occur among the sets A_i . An accepted vector of weight exactly k' can be obtained by employing some of the “padding” input variables $z[i]$ to C' .

Note that the seemingly simpler strategy of creating a new input variable for each set of at most k inputs to C would not serve our purposes, since it would involve increasing the size n of the circuit to possibly n^k . (We are limited in our computational resources for the reduction to $f(k)n^\alpha$. The constant α can be an arbitrary function of the depth and width bounds h and t , but not k .)

For the other direction, suppose C' accepts a vector of weight k' . Because of the implications in (1) above, exactly k sets of copies of inputs to C must have value 1 in the accepted input vector. Because of the implications (2)–(4), the variables

$v[i]$ must have values in the accepted input vector compatible with the values of the sets of copies. By the construction of C' , this implies there is a weight k input vector accepted by C . This concludes the proof of the Normalization Theorem.

The proof of the Normalization Theorem allows us to establish some technical results on monotone and antimonotone formulae.

Theorem 12.6 ([165], [166], [170])

(i) For $t > 0$, and t even,

(a) **MONOTONE** $W[t] = W[t]$.

(b) **WEIGHTED MONOTONE t -NORMALIZED SATISFIABILITY** is $W[t]$ -complete.

(c) **WEIGHTED MONOTONE $t + 1$ -NORMALIZED SATISFIABILITY** is in $W[t]$.

(ii) For $t > 0$, and t odd,

(a) **ANTIMONOTONE** $W[t] = W[t]$.

(b) **WEIGHTED ANTIMONOTONE t -NORMALIZED SATISFIABILITY** is $W[t]$ -complete.

(c) If, additionally, $t \geq 3$, **WEIGHTED ANTIMONOTONE $t + 1$ -NORMALIZED SATISFIABILITY** is in $W[t]$.

Proof. (i) Let t be even. The proof of (a) comes from the analysis of Step 6, Case 1. For (b) we can apply Step 6, Case 1, to a t -normalized formula. The result is a t -normalized monotone formula. For (c), the result follows by the transformations of Step 7, applied to a monotone $t + 1$ -normalized formula.

(ii) (a) Again the proof will consist of hardwiring a gadget. This time, the gadget is the one concerning the problem **RED/BLUE NONBLOCKER** from the proof of Theorem 10.9. The case $t = 1$ was dealt with in Theorem 10.3. Let C be a circuit of weft t for t odd, $t \geq 3$. By The Normalization Theorem, we may assume that C is represented by a boolean expression E_0 that is in (alternating) product-of-sums-of-products . . . form (for t alternations). The first level of the circuit below the inputs consists of *And* gates (since t is odd).

Suppose the inputs to C are $x_1, \dots, x_n$. Let X_1 be the boolean expression with single-literal clauses $X_1 = (x_1)(x_2) \cdots (x_n)$ and let G be the graph constructed from X_1 by the reduction in the proof of Theorem 10.3. Let $y_1, \dots, y_z$ be new variables, one for each red vertex in G .

Let E_1 be the boolean expression

$$E_1 = \prod_{u \in (V_{\text{blue}} - V_7)} \sum_{y_i \in N[u]} \neg y_i$$

and let C_1 be a circuit realizing E_1 .

We modify C in the following ways:

(1) Each positive fanout from an input x_i to C is replaced by an *And* gate receiving negated inputs from all of the new input variables y_j for which the corresponding red vertices of G represent the possibility that x_i evaluates to *false*.

(2) Each negated fanout from an input x_i to C is replaced by an *And* gate receiving negated inputs from all of the new input variables y_j for which the corresponding

red vertices of G represent the possibility that x_i evaluates to *true*.

(3) The circuit C_1 is conjunctively combined with C at the bottommost (output) *And* gate.

The circuit C' obtained in this way accepts a weight $2k$ input vector if and only if C accepts a weight k input vector. The argument for correctness is essentially the same as for Theorem 10.3. The circuit C' has weight t after the *And* gates, replacing the former inputs, are coalesced with the *And* gates of the topmost large gate level (this is feasible, since t is odd). All of the input fanout lines of C' are negated. (b) and (c) are similar.

Note that (i)(c) and (ii)(b) suggest the following structural collapses for all t :

MONOTONE $W[2t + 1] = W[2t]$.

ANTIMONOTONE $W[2t + 2] = W[2t + 1]$.

As we will see in Chapter 18, these collapses do indeed occur; hence, we finish up with the following picture of the W -hierarchy.

$$\begin{aligned}
 & \mathbf{FPT} = \mathbf{MONOTONE} \, W[1] \\
 & \subseteq W[1] = \mathbf{ANTIMONOTONE} \, W[1] = \mathbf{ANTIMONOTONE} \, W[2] \\
 & \subseteq W[2] = \mathbf{MONOTONE} \, W[2] = \mathbf{MONOTONE} \, W[3] \subseteq \\
 & \quad \quad \quad \vdots \\
 & \subseteq W[\mathbf{SAT}] \subseteq W[P].
 \end{aligned}$$

We have already seen that there are some natural graph-theoretic problems such as **DOMINATING SET** which live naturally at the $W[2]$ level of the W -Hierarchy. The restriction of dominating set to tournaments (**TOURNAMENT DOMINATING SET**) is also complete for $W[2]$, as we will see in Exercise 3. [A *tournament* is a directed graph $G = (V, A)$ such that for each pair of vertices $u, v \in V$, exactly one of uv or vu is in A .] The fact that the parameterized versions of **DOMINATING SET** and **TOURNAMENT DOMINATING SET** are of the same complexity provides another illustration of the fact that problems with unparameterized versions of apparently quite different complexities can have the same parameterized complexity. In this case, Papadimitriou and Yannakakis [368] have shown that **TOURNAMENT DOMINATING SET** is complete for the class **LOGNP**, whereas **DOMINATING SET** is **NP**-complete and hence **TOURNAMENT DOMINATING SET** is of the same classical polynomial time degree as **DOMINATING SET** only in the unlikely event that $NP \subseteq DTIME(n^{\log n})$.

We conclude this chapter by looking at one important example of a problem that is $W[t]$ -hard for all t . The problem is again the **LONGEST COMMON SUBSEQUENCE** problem which is important from the point of view of applications to computational molecular biology, and it is also important from our point of view since **LCS-1** has proved to be the key to many hardness results such as **BANDWIDTH**, **INTERVALIZING COLORED GRAPHS**, **TRIANGULATING COLORED GRAPHS**, and **COLORING CUTWIDTH**, etc. (See Refs. [79], [75, 76], and [289].)

From Chapter 11, we recall the three versions of the **LONGEST COMMON SUBSEQUENCE** problem.

TABLE 12.1. The Fixed-Parameter Complexity of the LCS Problem

Problem	Parameter	Alphabet Size $ \Sigma $	
		Unbounded	Fixed
LCS-1	k	$W[t]$ -hard, $t \geq 1$	?
LCS-2	m	$W[2]$ -hard	<i>FPT</i>
LCS-3	k, m	$W[1]$ -complete	<i>FPT</i>

LONGEST COMMON SUBSEQUENCE

Input: A set of k strings $X_1, \dots, X_k$ over an alphabet Σ , and a positive integer m .

Parameter 1: k . (We refer to this problem as LCS-1.)

Parameter 2: m . (We refer to this problem as LCS-2.)

Parameter 3: (k, m) . (We refer to this problem as LCS-3.)

Question: Is there a string $X \in \Sigma^*$ of length at least m that is a subsequence of X_i for $i = 1, \dots, k$?

The reader should also recall Table 12.1, the table of hardness results of the LCS problem. All of the results are due to Bodlaender, et al. [75, 76].

We proved that LCS-3 was $W[1]$ -complete in Theorem 11.11. The reader is invited to prove that LCS-2 is $W[2]$ -hard (Exercise 6). We conclude this chapter by proving the following result.

Theorem 12.7 *LCS-1 is hard for $W[t]$ for all t .*

Proof. We reduce from **MONOTONE t -NORMALIZED SATISFIABILITY**. Let E be a t -normalized monotone expression and k be the parameter. Let n denote the number of variables of E . By simple padding, we may assume that E has the form

$$E = \bigwedge_{i_1=1}^n \bigvee_{i_2=1}^n \cdots \bigwedge_{i_{t-1}=1}^n \bigvee_{i_t=1}^n l[i_1, \dots, i_t].$$

Let $V = \{u_1, \dots, u_n\}$ denote the set of variables of E . Thus, in the above expression for E , $l[i_1, \dots, i_t]$ is always a positive literal, that is, an element of V . We show how to produce an instance of LCS-1 consisting of $\binom{k}{2} + 2k + 2$ strings that have a common subsequence of length m if and only if E has a weight k truth assignment, with m described by

$$m = 3k + 3n^{t/2} + 2 \sum_{j=0}^t c(j)w(j, t),$$

where

$$c(j) = n^{\lceil j/2 \rceil}$$

and

$$w(j, t) = n^{2t(t-j)}.$$

We will use the following notation for indexing. The set $\{1, \dots, n\}$ is denoted as $[n]$. By $[n]^r$ we mean the set $\{\alpha = (a_1, \dots, a_r) : 1 \leq a_i \leq n \text{ for } 1 \leq i \leq r\}$. By $[n]^0$ we denote the singleton set $\{\epsilon\}$ where ϵ denotes the unique vector of length 0 over $[n]$. If $\alpha \in [n]^s$ and $b \in [n]$ with $\alpha = (a_1, \dots, a_s)$, then we write $\alpha.b$ to denote $(a_1, \dots, a_s, b) \in [n]^{s+1}$. We consider that $[n]^r$ is ordered lexicographically and will use $\uparrow_{\text{lex}}$ to denote increasing lexicographic order and $\downarrow_{\text{lex}}$ to denote decreasing lexicographic order. We make use of the index set I defined by

$$I = \bigcup_{r=1}^i [n]^r,$$

We say that $\alpha \in I$ is *even* if $\alpha \in [n]^r$ for r even, otherwise α is termed *odd*. If $\alpha \in [n]^r$, then we write $|\alpha| = r$ and term this the *rank* of α . If $\alpha, \beta \in I$ and α is a proper prefix of β , then we write $\alpha \prec \beta$.

The Alphabet

The alphabet Σ for target instance of LCS-1 can be expressed as the union

$$\Sigma = \Sigma_1 \cup \Sigma_2 \cup \Sigma_3 \cup \Sigma_4 \cup \Sigma_5,$$

where

$$\begin{aligned} \Sigma_1 &= \{c[j], c'[j] : 1 \leq j \leq k\}, \\ \Sigma_2 &= \{v[i, j] : 1 \leq i \leq n, 1 \leq j \leq k\}, \\ \Sigma_3 &= \{p[\alpha], p'[\alpha] : \alpha \in I\}, \\ \Sigma_4 &= \{q[\alpha, j], q'[\alpha, j] : \alpha \in [n]^i, 1 \leq j \leq k\}, \\ \Sigma_5 &= \{u[\alpha, i, j] : \alpha \in [n]^i, 1 \leq i \leq n, 1 \leq j \leq k\}. \end{aligned}$$

Symbol Subsets, Order, and Rank. Let Σ' denote $\Sigma_1 \cup \Sigma_2 \cup \Sigma_4 \cup \Sigma_5$. If S_1 and S_2 are sets of symbols of an ordered alphabet, then $S_1 < S_2$ denotes that for all $a \in S_1$ and $b \in S_2$, $a < b$. We consider that Σ' is linearly ordered in the unique way consistent with the following requirements:

- $(\Sigma_1 \cup \Sigma_2) < (\Sigma_4 \cup \Sigma_5)$.
- Σ_2 is ordered lexicographically by symbol index.
- For all $i \in [n]$ and $j \in [k]$, $c[j] < v[i, j] < c'[j]$.
- For $1 \leq i < j \leq k$, $\{c[i], c'[i]\} < \{c[j], c'[j]\}$.
- If (α, j) precedes (β, h) lexicographically, then $\{q[\alpha, j], q'[\alpha, j]\} < \{q[\beta, h], q'[\beta, h]\}$.
- Σ_5 is ordered lexicographically by symbol index.
- $q[\alpha, j] < u[\alpha, i, j] < q'[\alpha, j]$ for all $\alpha \in [n]^i$, $i \in [n]$ and $j \in [k]$.

By $\Sigma'[a, b]$ we denote the set of symbols $\{s \in \Sigma' : a \leq s \leq b\}$ in the above linear ordering.

Each of the symbols in $\Sigma'' = \Sigma_3 \cup \Sigma_4 \cup \Sigma_5$ is (partially) indexed by some $\alpha \in I$. We term the *rank* of a symbol s in this set, denoted $|s|$, to be the rank $|\alpha|$ of

the index α . If $\Gamma \subseteq \Sigma$ is a set of symbols, then $\Sigma''[r]$ denotes the set of symbols in Γ of rank r , $0 \leq r \leq t$.

In discussing strings over the alphabet Σ , if $\Gamma \subseteq \Sigma$ is a symbol subset and $S \in \Sigma^*$, then by $S \cap \Gamma$ we denote the subsequence of S consisting of all symbols in Γ . We write $|S|$ to denote the length of a string S .

Substring Gadgets. Product notation in the description of these components refers to string concatenation. Where s is a symbol, the notation s^w denotes the symbol s repeated w times. Note that in some cases products are formed in decreasing order according to some index, which is indicated by notation such as

$$\prod_{i=n}^1 \dots$$

The following strings provide gadgets for our reduction:

$$\begin{aligned} \langle \uparrow \text{ selection } j \rangle &= c[j] \left(\prod_{i=1}^n v[i, j] \right) c'[j], \\ \langle \downarrow \text{ selection } j \rangle &= c[j] \left(\prod_{i=n}^1 v[i, j] \right) c'[j], \\ \langle \uparrow \text{ select} \rangle &= \prod_{j=1}^k \langle \uparrow \text{ selection } j \rangle, \\ \langle \downarrow \text{ select} \rangle &= \prod_{j=1}^k \langle \downarrow \text{ selection } j \rangle. \end{aligned}$$

Where Γ is a set of symbols, we use $\langle * \Gamma * \rangle$ to denote an arbitrary string which contains as a subsequence every string of length m over Γ (such as a string which simply runs through Γ , m times in any order). As a notational convenience, we write $\langle * s \dots t * \rangle$ for $\langle * \Sigma_3 \cup \Sigma'[s, t] * \rangle$.

Recursively, we define $\langle \uparrow \alpha \rangle$ and $\langle \downarrow \alpha \rangle$ for $\alpha \in I$.

For $\alpha \in [n]^t$ and $l[\alpha] = u_i \in V$:

$$\begin{aligned} \langle \uparrow \alpha.i \rangle &= p[\alpha] \left(\prod_{j=1}^k q[\alpha, j] u[\alpha, i, j] q'[\alpha, j] \right) p'[\alpha], \\ \langle \downarrow \alpha.i \rangle &= p[\alpha] \left(\prod_{j=k}^1 q[\alpha, j] u[\alpha, i, j] q'[\alpha, j] \right) p'[\alpha]. \end{aligned}$$

In general:

$$\langle \uparrow \alpha \rangle = p[\alpha]^{w(|\alpha|, t)} \prod_{i=1}^n \langle \uparrow \alpha.i \rangle p'[\alpha]^{w(|\alpha|, t)},$$

$$\begin{aligned} \langle \downarrow \alpha \rangle &= p[\alpha]^{w(|\alpha|, t)} \prod_{i=1}^n \langle \downarrow \alpha.i \rangle p'[\alpha]^{w(|\alpha|, t)} && \text{for } \alpha \text{ even,} \\ \langle \downarrow \alpha \rangle &= p[\alpha]^{w(|\alpha|, t)} \prod_{i=n}^1 \langle \downarrow \alpha.i \rangle p'[\alpha]^{w(|\alpha|, t)} && \text{for } \alpha \text{ odd.} \end{aligned}$$

The Reduction. We may now describe the reduction. The instance of **LCS-1** to which we reduce consists of three sets of strings: the *Control Strings*, the *Quorum Strings*, and the *Consistency Strings*.

The two *Control Strings* are

$$\begin{aligned} X_1 &= \langle \uparrow \text{ select} \rangle \langle \uparrow \epsilon \rangle, \\ X_2 &= \langle \downarrow \text{ select} \rangle \langle \downarrow \epsilon \rangle. \end{aligned}$$

Let $\Delta = \{(r, s) : 1 \leq r < s \leq n\}$ ordered lexicographically. The $\binom{k}{2}$ *Quorum Strings* are, for $1 \leq i < j \leq n$,

$$Y_{ij} = \left(\prod_{(r,s) \in \Delta}^{\uparrow \text{lex}} \langle *c[1] \cdots c[i] * \rangle v[r, i] \langle *c'[i] \cdots c[j] * \rangle v[s, j] \langle *c'[j] \cdots c'[k] * \rangle \right) \langle * \Sigma'' * \rangle.$$

The $2k$ *Consistency Strings* are, for $j = 1, \dots, k$,

$$\begin{aligned} Z_j &= \prod_{r=1}^n \langle \text{selection } j \text{ is variable } r \rangle, \\ Z'_j &= \prod_{r=n}^1 \langle \text{selection } j \text{ is variable } r \rangle, \end{aligned}$$

where

$$\langle \text{selection } j \text{ is variable } r \rangle = \langle *c[1] \cdots c[j] * \rangle v[r, j] \langle *c'[j] \cdots c'[k] * \rangle \left(\prod_{\alpha \in [n]^t}^{\uparrow \text{lex}} \langle *q[\alpha, 1] \cdots q[\alpha, j] * \rangle u[\alpha, r, j] \langle *q'[\alpha, j] \cdots q'[\alpha, k] * \rangle \right)$$

Proof of Correctness. The following general ideas are useful to our arguments. To the expression E there naturally corresponds a boolean tree circuit $C = C_E$. A truth assignment τ to the variables V of E may be considered as an input vector x_τ to the circuit C , with $C(x_\tau) = 1$ if and only if τ satisfies E . The circuit C may be described as follows

- (1) For each $\alpha \in I$, there is a gate g_α of C (of rank $|\alpha|$).
- (2) g_α is an *and* gate if α is even, and an *or* gate if α is odd.
- (3) The output gate of C is g_ϵ .
- (4) For $|\alpha| < t$, the gate g_α takes input from the gates $g_{\alpha.i}$ for $i = 1, \dots, n$.
- (5) The inputs to C are in 1:1 correspondence with V .
- (6) For $|\alpha| = t$, the gate g_α takes the single input $u_i \in V$ such that $l[\alpha] = u_i$ in E .

A subcircuit C' of C is a *witnessing subcircuit* if it satisfies the conditions:

- (1) $g_\epsilon \in C'$.
- (2) For each even $\alpha \in I$, $|\alpha| < t$, if $g_\alpha \in C'$, then for all $i \in [n]$, $g_{\alpha.i} \in C'$.
- (3) For each odd $\alpha \in I$, if $g_\alpha \in C'$, then there is a unique $i \in I$ such that $g_{\alpha.i} \in C'$.

The following observations about witnessing subcircuits are useful.

Claim 1. $C(x) = 1$ if and only if there is a witnessing subcircuit C' of C such that $C'(x) = 1$ and each gate of C' evaluates to 1.

Claim 1 follows trivially from the monotonicity of C .

Claim 2. If C' is a witnessing subcircuit of C , then the number of gates of rank r , for $r = 0, \dots, t$, is given by the function

$$c(j) = n^{\lceil j/2 \rceil}.$$

Claim 2 follows by an elementary induction, noting the special structure of C .

The following fact about the “weighting function” $w(j, t)$ will be useful.

Claim 3. For $0 \leq r \leq t - 1$,

$$w(r, t) > \sum_{j=r+1}^t |X_1 \cap \Sigma''[j]|.$$

Claim 3 is easily verified from the definitions.

Claim 4. In the Control Strings X_1 and X_2 :

- (1) Each symbol in Σ_3 occurs as a *block*; that is, the symbol occurs only in a substring consisting of some number of repetitions of the symbol.
- (2) If $\alpha < \beta$, then all symbols with index β occur between the block of symbols $p[\alpha]^{w(|\alpha|, t)}$ and the block of symbols $p'[\alpha]^{w(|\alpha|, t)}$.
- (3) If β is an index of a symbol occurring between the symbol blocks $p[\alpha]^{w(|\alpha|, t)}$ and $p'[\alpha]^{w(|\alpha|, t)}$, then $\alpha \leq \beta$, with $\alpha < \beta$ properly if the symbol is in Σ_3 .

Claim 4 is readily observed from the definition of $\langle \uparrow \alpha \rangle$ and $\langle \downarrow \alpha \rangle$.

In one direction, the argument for the correctness of the reduction is relatively easy. Given a satisfying weight k truth assignment $\tau : V \rightarrow \{0, 1\}$ for E , we describe a common subsequence of length m in the following way. Let C' be a witnessing subcircuit of C for the input vector corresponding to τ . Let I' denote the set of indices of the logic gates of C'

$$I' = \{\alpha : g_\alpha \in C'\}$$

and suppose the variables set to 1 by τ are $v_{i_1}, \dots, v_{i_k}$, with $v_{i_1} < v_{i_2} < \dots < v_{i_k}$.

Let Γ denote the set of symbols

$$\begin{aligned} \Gamma = & \Sigma_1 \cup \{v[i_j, j] : 1 \leq j \leq k\} \cup \{p[\alpha], p'[\alpha] : \alpha \in I'\} \\ & \cup \{q[\alpha, j], q'[\alpha, j] : \alpha \in [n]^t \cap I', l[\alpha] = v_{i_j}, 1 \leq j \leq k\} \\ & \cup \{u[\alpha, i_j, j] : \alpha \in [n]^t \cap I', l[\alpha] = v_{i_j}, 1 \leq j \leq k\}. \end{aligned}$$

Claim 5. The string $S = X_1 \cap \Gamma$ is a common subsequence of the Control and Consistency Strings of length m .

Proof of Claim 5. First note that $S = S_1 S_2$, where $S_1 \in (\Sigma_1 \cup \Sigma_2)^*$ and $S_2 \in (\Sigma'')^*$ and that similar factorizations hold for X_1 and X_2 . An inspection of the definition of $\langle \uparrow \text{ select} \rangle$ and $\langle \downarrow \text{ select} \rangle$ shows that S_1 is a common subsequence of the first parts of X_1 and X_2 , the main point being that between each pair of symbols $c[j]$ and $c'[j]$, there is just the single symbol $v[i_j, j]$ in S . Note also that the length of S_1 is $3k$.

Let $X'_i = X_i \cap \Sigma''$ for $i = 1, 2$. We argue that S_2 is a subsequence of X'_1 and X'_2 by induction on the rank of symbols in S_2 . Let $S_2[r]$ denote the subsequence of S_2 consisting of the symbols of rank r . By Claim 4, it is sufficient to establish that $S_2[r]$ is a subsequence of X'_1 and X'_2 for $r = 0, \dots, t$. The base step of the induction, $r = 0$, is trivial. For the induction step, by Claim 4, it suffices to show that the subsequence of S consisting of symbols with index $\beta = \alpha.i$ (for some i) having rank $r + 1$ is a subsequence of X'_i (for $i = 1, 2$) occurring between the symbol blocks $p[\alpha]^{w(|\alpha|, t)}$ and $p'[\alpha]^{w(|\alpha|, t)}$. If α is even, then this follows from the fact that the blocks of the symbols $p[\alpha.i]$ and $p'[\alpha.i]$ occur in ascending order in both X'_1 and X'_2 . If α is odd, then this follows trivially because there is only one relevant index $\alpha.i \in I'$. Note that in S there are precisely three symbols between each pair of symbol blocks $p[\alpha]^{w(|\alpha|, t)}$ and $p[\alpha]^{w(|\alpha|, t)}$, where $\alpha \in [n]^t \cap I'$, and that there are $n^{t/2}$ such pairs. From this, it is easy to verify that S has length m .

The above arguments establish that S is a subsequence of the Control Strings. By essentially the same inductive argument, the symbols $p[\alpha]$ must occur in S in lexicographically increasing order. Using this fact, it is straightforward to verify that S is a subsequence of $\langle \text{selection } j \text{ is variable } i_j \rangle$ and thus S is a subsequence of Z_j and Z'_j for $j = 1, \dots, k$. S_2 is trivially a subsequence of $\langle * \Sigma'' * \rangle$. For $p < q$, $v_{i_p} < v_{i_q}$, so S is a subsequence of Y_{pq} .

To complete the proof of correctness for the reduction, we argue that if T is a common subsequence of the Control and Consistency Strings of length m , then E is satisfied by a weight k truth assignment. In particular, we argue that T must correspond to a weight k input vector and a witnessing subcircuit of $C = C_E$ with respect to this vector.

Because the Control Strings can be factored in a similar way, we may factor T as $T = T_1 T_2$ with $T_1 \in (\Sigma_1 \cup \Sigma_2)^*$ and $T_2 \in (\Sigma'')^*$.

Claim 6. The length of T_1 is at most $3k$.

Claim 6 follows simply from the fact that for any fixed index j , the symbols $v[i, j]$ occur in X_1 in increasing order with respect to i and they occur in X_2 in decreasing order with respect to i .

Say that an index $\alpha \in I$ is *represented* in T if both of the symbols $p[\alpha]$ and $p'[\alpha]$ occur in T . Say that an index $\alpha \in I$ is *forbidden* in T if for all indices β with $\alpha \preceq \beta$, no symbol with index β occurs in T . The following is an immediate consequence of the definition.

Claim 7. If α is forbidden in T , then for all $i \in [n]$, $\alpha.i$ is forbidden.

Claim 8. If $\alpha \in I$ is odd, then there is at most one $i \in [n]$ with $\alpha.i$ represented in T . Furthermore, if $\alpha.i$ is represented in T , then for all $j \neq i$, $\alpha.j$ is forbidden in T .

Proof of Claim 8. Suppose $i < j$ with $\alpha.i$ represented in T . By the definition of X_1 and X_2 , all of the symbols with index $\alpha.i$ precede all of the symbols with index $\alpha.j$ in X_1 , and all of the symbols with index $\alpha.i$ succeed all of the symbols with index $\alpha.j$ in X_2 . Consequently, no symbol with index $\alpha.j$ can occur in the common subsequence T . Furthermore, if β is an index with $\alpha.j \preceq \beta$, then by Claim 4, all symbols with index β occur in X_1 and X_2 between blocks of symbols with index $\alpha.j$, so the above argument applies as well to symbols with these indices, so that $\alpha.j$ is forbidden. The case of $j < i$ is symmetric.

Let $s(r)$ denote the number of indices $\alpha \in I$ of rank r that are represented in T .

Claim 9. For $r = 0, \dots, t$

(1) $s(r) = c(r)$.

(2) Every index of rank r is either represented or forbidden.

Proof of Claim 9. By induction on r . For $r = 0$, if either $p[\epsilon]$ or $p'[\epsilon]$ fails to occur in T , then necessarily $|T \cap \Sigma''[0]| \leq w(0, t)$, so T must contain at least $w(0, t)$ symbols of rank at least 1, a contradiction of Claim 3. This establishes both (1) and (2).

For the induction step, if $s(r+1) < c(r+1)$, then the induction hypothesis and the definition of m imply that T must contain more than $w(r+1, t)$ symbols of rank greater than $r+1$, which contradicts Claim 3. Suppose $s(r+1) > c(r+1)$.

Case 1: r is even. Then $c(r+1) = n \cdot c(r)$. By (1) of the induction hypothesis, there are precisely $s(r) = c(r)$ indices of rank r represented in T , and all other indices of rank r are forbidden. Since each represented index of rank r has only n extensions to an index of rank $r+1$, there must be some rank $r+1$ index $\alpha.i$ represented in T for which α is not represented in T . By (2) of the induction hypothesis, α is forbidden in T , a contradiction. Thus, (1) must hold, and by the same argument, if α of rank r is represented, then for all $i \in [n]$, $\alpha.i$ is represented. If α of rank r is forbidden in T , then by Claim 7, $\alpha.i$ is forbidden in T for all $i \in [n]$. This establishes (2).

Case 2: r is odd. Then $c(r+1) = c(r)$. By (1) of the induction hypothesis, there are precisely $s(r) = c(r)$ indices of rank r represented in T , and all other indices of rank r are forbidden. There cannot be an index $\alpha.i$ represented in T with α not represented, as this would contradict (2) of the induction hypothesis. By the Pigeonhole Principle, there must be an index α represented in T and $i \neq j$ with both $\alpha.i$ and $\alpha.j$ represented in T , a contradiction of Claim 8. Thus, (1) must hold, and by the same arguments, we see that for each represented α of rank r , there is a unique $i \in [n]$ with $\alpha.i$ represented. By Claim 8, we get (2).

One can observe from the definition of $\langle \uparrow \alpha \rangle$ and $\langle \downarrow \alpha \rangle$ that there can be at most three symbols in $T \cap (\Sigma_4 \cup \Sigma_5)$ with a given index α of rank t , and that these must occur between $p[\alpha]$ and $p'[\alpha]$ and must occur in a substring of the form $q[\alpha, j]u[\alpha, i, j]q'[\alpha, j]$. By this observation and Claims 6 and 9, we can conclude:

Claim 10. The length of T_1 is precisely $3k$ and the length of T_2 is precisely

$$3n^{t/2} + 2 \sum_{j=0}^t c(j)w(j, t).$$

On the basis of Claim 10, we may associate to T a well-defined truth assignment τ to the variables of the expression E : $\tau(u_i) = 1$ if and only if for some j , $1 \leq j \leq k$, the symbol $v[i, j]$ occurs in T_1 . By Claim 10, there are exactly k symbols of Σ_2 in T_1 . However, we must argue that for $j < j'$, only one of $v[i, j]$ and $v[i, j']$ occurs in T_1 , thus ensuring that τ has weight k . To see this, note that T must be a subsequence of $Y_{jj'}$. Suppose $v[i, j]$ occurs in T (necessarily in T_1). The only symbols $v[i', j']$ occurring in $Y_{jj'}$ after $v[i, j]$ satisfy $i < i'$, by the definition of $Y_{jj'}$. Thus, to T we may associate a truth assignment of weight k .

Claim 11. If $\alpha \in [n]^t$ is represented in T , with $u[\alpha, i, j]$ occurring between $p[\alpha]$ and $p'[\alpha]$, then $v[i, j]$ occurs in T_1 (and τ assigns u_i the value 1).

Proof of Claim 11. By Claim 10, there must be a symbol $v[p, j]$ in T_1 . The symbol $u[\alpha, i, j]$ occurs only once in Z_j and in Z'_j . The symbols $v[p, j]$ preceding the occurrence of $u[\alpha, i, j]$ in Z_j satisfy $p \leq i$. The symbols $v[p, j]$ preceding the occurrence of $u[\alpha, i, j]$ in Z'_j satisfy $p \geq i$. Thus, the only possibility is $v[i, j]$.

Claim 12. The indices $\alpha \in I$ represented in T are those of a witnessing subcircuit C' of C that accepts the input vector x_τ corresponding to the truth assignment τ .

Proof of Claim 12. That the indices represented in T form a witnessing subcircuit C' of C follows from Claims 8 and 9. Since C' is monotone, it suffices to establish that all gates of rank t evaluate to 1 in order to conclude that $C'(x_\tau) = 1$. This follows from Claim 11, noting that if α of rank t is represented, then $u[\alpha, i, j]$ occurs between $p[\alpha]$ and $p'[\alpha]$ if and only if $l[\alpha] = u_i$, by the definition of $\langle \uparrow \alpha \rangle$ and Claim 4.

By the correspondence between C and E , we conclude that τ is a weight k truth assignment for E , which completes the proof of the theorem.

We remark that we do not know if LCS-1 is in $W[P]$.

Further Notes on the W -Hierarchy and Generalized Normalization to $W^[t]$*

Recently, Downey et al. [177] studied the following parametric problem in relational database queries: It is well known that for a fixed relational database query of ϕ in m free variables, it can be determined in time polynomial in the size n of the database whether there exists an m -tuple x that belongs to the relation defined by the query. For the best known algorithms, however, the exponent of the polynomial is proportional to the size of the query. Downey et al. studied the data complexity of this problem parameterized by the size $k = |\phi|$ of the query, and answer a question raised by Yannakakis at STOC 1995. The main results from Ref. [177]

show that (1) the general problem is complete for the parametric complexity class $AW[*]$ (of Chapter 13), and (2) when restricted to monotone queries, the problem is complete for the fundamental parametric complexity class $W[1]$.

An important consequence of the proof of (2) was a significantly improved characterization of the parameterized complexity class $W[1]$. Our basic definition was for families of circuits that satisfy that (1) the depth of the circuits is bounded by a constant c and (2) on any input-output path there is at most one gate having unbounded fan in (termed a *large* gate), with all other gates having fan in bounded by c (that is, *small* gates). Downey et al. proved that the definition can be broadened by allowing circuits of depth bounded by an arbitrary function $f(k)$. If we denote this parameterized complexity class $W^*[1]$, then the improved characterization is:

$$W^*[1] = W[1].$$

As a consequence, for instance, the k -WEIGHTED SATISFIABILITY problem for boolean expressions that are products of k -sums is complete for $W[1]$. Downey et al. also proved a general normalization lemma for $W^*[t]$ for all t .

Although the Downey-Fellows-Taylor result fell short of the desired “ $W[t] = W^*[t]$,” which is still conjectural, in a subsequent article, Downey and Fellows [172] proved that $W[2] = W^*[2]$.

The significance of all of these results come from the fact that the generalized normalization lemmata allow for “easier” membership arguments. The arguments for the cases $t = 1, 2$ above are quite ad hoc and it seems that the general “ $W[t] = W^*[t]$?” question is hard.

Exercises 12.0

- 12.0.1.** Prove Corollary 12.5(ii); that is, prove that **DOMINATING SET** is in *MONOTONE* $W[2]$.

[Hint: The reduction is straightforward. Let $\{x_1, \dots, x_n\}$ denote the set of vertices of the given graph G , and we will interpret them as input variables for our circuit $C(G)$. Have a layer of *Or* gates directly below the variables. These are also one per input variable and we will label them $g_1, \dots, g_n$. Make x_i and input to the *Or* gate g_j precisely in the case that (x_i, x_j) is an edge of G . Now to complete the circuit, we have one large *And* gate with inputs from each of the *Or* gates. It is easy to see that satisfying assignments correspond directly to dominating sets and conversely.]

- 12.0.2.** The reduction of Theorem 12.3 proves also that the problem of determining if a CNF formula has a satisfying assignment of weight $\leq k$ is $W[2]$ -complete. Prove, however, that the problems of whether determining if a $W[1]$ circuit has a weight $\leq k$ satisfying assignment, is in *FPT*.

- 12.0.3.** Recall that a tournament is a directed graph $G = (V, A)$ such that for all $x, y \in V$, exactly one of xy or yx is in A . Prove that **TOURNAMENT DOMINATING SET** is $W[2]$ -complete [169]).

[Hint: To show that it is hard for $W[2]$, reduce from **DOMINATING SET**. Let $G = (V, E)$ be an undirected graph for which we wish to determine whether G has a dominating set of size k . We describe how a tournament T can be constructed that has a dominating set of size $k + 1$ if and only if G has a dominating set of size k . The size of T is $O(2^k \cdot n)$, where n is the number of vertices of G , and T can be constructed in time polynomial in n and 2^k .]

The vertex set of the tournament T is partitioned into three sets: V_A , V_B , and V_C . The vertices in the set V_A are in 1:1 correspondence with the vertices of G , and we write $V_A = \{a[u] : u \in V(G)\}$. The set of vertices V_B of T corresponds to m copies of the vertex set of G and we may write this as $V_B = \{b[i, u] : 1 \leq i \leq m, u \in V(G)\}$. (The appropriate cardinality for m is determined below.) V_C consists of just a single vertex which we will denote c . The construction of T must ensure that for every pair of vertices x, y of T , one of the arcs xy or yx is present. Let T_0 be any tournament on n vertices (we will use T_0 as “filler”). Include arcs in T to make a copy of T_0 between the vertices of each of the n -element sets V_A and $V_B(i) = \{b[i, u] : u \in V(G)\}$ for $i = 1, \dots, m$.

Construct a tournament T_1 on $m = O(2^{k+1})$ vertices which has no dominating set of size $k + 1$. Consider that the vertex set of T_1 is $V(T_1) = \{1, \dots, m\}$. For each arc ij in T_1 , include in T an arc from each vertex of $V_B(i)$ to each vertex of $V_B(j)$.

The adjacency structure of G is represented in T in the following way: For each vertex $u \in V(G)$ include arcs from the vertex $a[u]$ to the vertices $b[i, v]$ for every $v \in N_G[u]$ and for each i , $1 \leq i \leq m$, and from every other vertex in V_B include an arc to $a[u]$. Thus, the neighborhood structure of G is represented in arcs from V_A to V_B , otherwise there are arcs from V_B to V_A .

Finally, there are arcs in T from c to every vertex in V_A and from every vertex in V_B to c .

Now, argue that G has a k element dominating set iff T has a $k + 1$ element dominating set.]

12.0.4. (*Blind Reductions and Learning by Equivalence and Membership Queries.*) The vertex set of the tournament $T(G)$ that is constructed in the previous question depends only on the size of G , and furthermore, the correspondence between the solutions to the two problems is such that for a pair of corresponding problem instances $(G, T(G))$:

- (1) A solution for $T(G)$ can be computed from a solution for G , even if G is *invisible*, that is, knowing only the size of G .
- (2) A solution for G can be similarly computed from a solution for $T(G)$, even if $T(G)$ is *invisible*.

A moment's reflection on familiar reductions in the theory of NP -completeness will reveal that the reductions there almost never have this property. Reductions in fixed-parameter complexity theory often do possess this property, which provides for interesting applications in computational learning theory. We briefly describe how these applications come about.

In the model of exact learning by membership and (extended) equivalence queries, we study the situation where the Teacher possesses (for

example) a graph on n vertices that is hidden from the Learner. We assume that the Learner initially knows *nothing* about the graph, not even its size. The goal of the Learner is to produce “knowledge” of, for example, the dominating sets in the graph. Such knowledge can be represented by an algorithm (e.g., a circuit) which decides whether a given set of vertices is a dominating set in the graph being taught.

There are two ways in which the Learner may interact with the Teacher:

(1) By presenting a knowledge representation to the Teacher and asking whether it is correct. The Teacher responds either *yes* (in which case the Learner is done) or *provides a counterexample* to the correctness of the knowledge representation. This interaction is termed an (extended) *equivalence query*.

(2) By asking whether a particular set of vertices is a dominating set in the graph being taught. The Teacher responds either *yes* or *no*. This interaction is called a *membership query*.

A central question that is studied is whether there is an algorithm that the Learner can execute in polynomial time (in the size of the graph to be learned) which will yield a complete and correct knowledge representation in interaction with any Teacher who responds correctly (but not necessarily judiciously) to the queries.

Prove the following:

(i) (Downey and Fellows [169]) If the k -element dominating sets of a tournament can be learned in fixed-parameter polynomial time, then so can the k -element dominating sets in an arbitrary graph.

[Hint. Use the blindness of the reduction of the last question. The Learner is interacting with a Teacher of k -element dominating sets in a graph of unknown size. By diagonalization, however, the Learner can be presumed to have knowledge of the size of the graph. The Learner forms a mental model of a tournament as described by the reduction of the previous question. Since the graph is not visible to the Learner, this model is not complete. The Learner pretends to learn the dominating sets in the tournament. Because of the blindness of the reduction, queries that the Learner wishes to make about the tournament can be translated into queries about dominating sets in the graph that the Teacher actually present is teaching, and similarly, counterexamples provided by the Teacher can be translated into counterexamples concerning the (imaginary) tournament. When the Learner has a correct knowledge representation of the dominating sets in the tournament, this provides (by composing with the blind translation of solutions between the problems) a correct representation of the dominating sets in the graph.]

(ii) (Downey, Evans, and Fellows [163]) Arbitrary DNF formulas can be exactly learned in polynomial time by extended equivalence queries (only) if and only if monotone DNF formulas can be so learned.

[Hint. Use similar reasoning applied to the reduction of Lemma 12.4.]

- 12.0.5. (Monoid Factorization)** (Cai, Chen, Downey, and Fellows [109])
 Prove that the following problem is hard for $W[2]$ using a reduction from **DOMINATING SET**.

H_n FACTORIZATION

Instance: A set A of self-maps on $[n]$, and a self-map h .

Parameter: A positive integer k .

Question: Is there a factorization of h of length k over A ?

[Hint: Let $G = (V, E)$ be a graph for which we are to determine whether there is a k -element dominating set. Let n be the order of G . As in Exercise 6, we construct a set A of self-maps on $[n']$ where we view $[n']$ as consisting of n blocks. Here, we have $n' = 2n$ (the blocks have size 2). Let α denote the self-map on $\{1, 2\}$ that maps both elements to 2. For each vertex u of G , we construct a map a_u that consists of α in each block corresponding to a vertex $v \in N[u]$, and that is the identity map on all other blocks. The self-map h to be factored consists of α in each block. Note that $\alpha = \alpha^i$ for any number of compositions of the block map α .]

- 12.0.6. (Bodlaender, Downey, Fellows, and Wareham [76])** Prove that **LCS-2** is in $W[P]$ and hard for $W[2]$.

[Hint: Membership in $W[P]$ is easy. For $W[2]$ -hardness, reduce from **DOMINATING SET**. Let $G = (V, E)$ be a graph with $V = \{1, \dots, n\}$. We will construct a set S of strings that have a common subsequence of length k if and only if G has a k -element dominating set.

The alphabet for the construction is

$$\Sigma = \{\alpha[i, j] : 1 \leq i \leq k, 1 \leq j \leq n\}$$

with

$$\Sigma_i = \{\alpha[i, j] : 1 \leq j \leq n\},$$

$$\Sigma[t, u] = \{\alpha[i, j] : (i \neq t) \text{ or } (i = t \text{ and } j \in N[u])\}.$$

The set S consists of the following strings.

Control Strings

$$X_1 = \prod_{i=1}^k (\uparrow \Sigma_i),$$

$$X_2 = \prod_{i=1}^k (\downarrow \Sigma_i).$$

Check Strings

For $u = 1, \dots, n$:

$$X_u = \prod_{i=1}^k (\uparrow \Sigma[i, u]).$$

To see that the construction works correctly, first note that by Claim 1 of the proof of Theorem 12.7, it follows easily that any sequence C of length k common to both control strings must consist of exactly one symbol from

each Σ_i in ascending order. Thus, with such a sequence C , we may associate the set V_C of vertices represented by C : if $C = \alpha[1, u_1] \cdots \alpha[k, u_k]$, then $V_C = \{u_i : 1 \leq i \leq k\} = \{x : \exists i \alpha[i, x] \in C\}$.

We argue that if C is also a subsequence of the Check Strings $\{X_u\}$, then V_C is a dominating set in G . To this end, let $u \in V(G)$ and fix a substring C_u of X_u with $C_u = C$.

Claim. For some index j , $1 \leq j \leq k$, the symbol $\alpha[j, u_j]$ occurs in the $(\uparrow \Sigma[j, u])$ portion of X_u , and thus $u_j \in N[u]$ by the definition of $\Sigma[j, u]$.

We argue by induction on k . The case of $k = 1$ is clear. For the induction step, there are two cases: (1) the first $k - 1$ symbols of C_u occur in the prefix $(\uparrow \Sigma[1, u]) \cdots (\uparrow \Sigma[k - 1, u])$ of X_u , and the induction hypothesis immediately yields the Claim, or (2) the symbol $\alpha[k - 1, u_{k-1}]$ occurs in the $(\uparrow \Sigma[k, u])$ portion of $C_u \cap X_u$. In case (2), this implies that the symbol $\alpha[k, u_k]$ of $C = C_u$ also occurs in the $(\uparrow \Sigma[k, u])$ part of X_u .

By the Claim, if C is a subsequence of the *Control* and *Check* strings, then every vertex of G has a neighbor in V_C , that is, V_C is a dominating set in G .

Conversely, if $D = \{u_1, \dots, u_k\}$ is a k -element dominating set in G with $u_1 < \cdots < u_k$, then the sequence $C = \alpha[1, u_1] \cdots \alpha[k, u_k]$ is easily seen to be common to the strings of $\mathcal{S}$.]

13

Beyond $W[t]$ -Hardness

The $W[t]$ -classes reflect the intrinsic difficulty of solution checking formulas of bounded logical depth. Naturally, the question arises as to what happens if we have no bound on the depth and simply look at parameterized problems of “polynomial size.” When we do this, we arrive at classes hard for $\cup_t W[t]$. The study of such classes is the theme of the present chapter. Two important classes immediately suggest themselves. These are the classes $W[SAT]$ and $W[P]$ generated by the following problems.

WEIGHTED SATISFIABILITY

Input: A (boolean) formula X .
Parameter: A positive integer k .
Question: Does X have a weight k satisfying assignment?

WEIGHTED CIRCUIT SATISFIABILITY

Input: A decision circuit C .
Parameter: A positive integer k .
Question: Does C have a weight k satisfying assignment?

The classes are then the following:

$W[SAT]$, the class of problems reducible to **WEIGHTED SATISFIABILITY**.

$W[P]$, the class of problems reducible to **WEIGHTED CIRCUIT SATISFIABILITY**.

The standard translation of Turing machines into circuits shows that **WEIGHTED CIRCUIT SATISFIABILITY** is the same as the problem of ascertaining if a deterministic Turing machine accepts an input of weight k . In the same way that we have seen

that **WEIGHTED CNF SATISFIABILITY** and **WEIGHTED 3SAT** seem to have different parameterized complexities, it seems that the containment $W[\text{SAT}] \subset W[P]$ is proper. $W[\text{SAT}]$ does not seem to be well populated, but $W[P]$ contains many languages, most particularly those that are P -complete “by the slice.”

For the remainder of the chapter we will identify some $W[P]$ -complete languages. We begin with some technical results. Consider the following problem.

SHORT CIRCUIT SATISFIABILITY

- Instance:* A decision circuit C with at most n gates and $\leq k \log n$ inputs.
Parameter: A positive integer k .
Question: Is there any setting of the inputs that causes C to output 1?

Lemma 13.1 [4] **SHORT CIRCUIT SATISFIABILITY** is $W[P]$ -complete.

Proof. To see that it is $W[P]$ -hard, take an instance C of **WEIGHTED CIRCUIT SATISFIABILITY**, with parameter k and inputs $x_1, \dots, x_n$. Let $z_1, \dots, z_{k \log n}$ be new variables. Using lexicographic order in polynomial time (independent of k), we can have a surjection from $Z = \{z_1, \dots, z_{k \log n}\}$ to the (characteristic function of the) k -element subsets of $X = \{x_1, \dots, x_n\}$. Representing this as a circuit with inputs Z and outputs X , we can put this circuit on top of C to form C' so that C accepts a weight k input iff C' accepts some input.

That $W[P]$ contains **SHORT CIRCUIT SATISFIABILITY** is equally easy, just use the polynomial embedding of the $k \log n$ into the $k + 1$ -element subsets of n for n sufficiently large.

We remark that the $k \log n$ trick above is very useful in many arguments. For the following result, we shall employ ideas from Ref. [5]. Let φ be a formula of propositional logic (or a circuit). Suppose we wish to extend a partial truth assignment t of a (circuit or) formula φ to a satisfying assignment t' . If there is a clause, say, all but one of whose literals are forced to be false by t , then t implicitly causes the remaining literal to be made *true*. Repeat this process until we can go no further. If this process generates t' , a satisfying assignment of φ all of whose literals are specified, we say that t causes φ to *unravel*. Following Ref. [5], we have the problem definition.

SHORT SATISFIABILITY

- Input:* A formula φ of n variables and a list of at most $k \log n$ distinguished variables of φ .
Parameter: A positive integer k .
Question: Is there any setting of the distinguished variables that causes φ to unravel?

Theorem 13.2 **SHORT SATISFIABILITY** is $W[P]$ -complete.

Proof. We reduce from **SHORT CIRCUIT SATISFIABILITY**. Let $x_1, \dots, x_m, C$ be an instance of **SHORT CIRCUIT SATISFIABILITY** with parameter k and $m \leq k \log n$. These variables will be the distinguished variables for the input

$\varphi = \varphi(C)$. We introduce new variables and define φ by locally replacing gates of C as follows. We replace a *not* gate $y = \bar{x}$ by the formula $(x \vee y) \wedge (\bar{x} \vee \bar{y})$ and an *and* gate $z = x \wedge y$ by $(x \vee \bar{z}) \wedge (y \vee \bar{z}) \wedge (\bar{x} \vee \bar{y} \vee z)$. The formula generated by the circuit is at most quadratically larger than the circuit, and clearly, a setting of $x_1, \dots, x_m$ satisfies C iff it unravels φ .

The basis for many of our reductions is the $W[P]$ -completeness of **WEIGHTED MONOTONE CIRCUIT SATISFIABILITY**, which is the restriction of the problem **WEIGHTED CIRCUIT SATISFIABILITY** to circuits with no inverters. This result was first announced in Downey et al. [179] in which they analyzed the parameterized tractability of logical problems stemming from logic and linguistics. In the present chapter, the key problem identified by those authors is the following.

MINIMUM AXIOM SET

- Input:* A finite set S of “sentences,” an “implication relation” R consisting of pairs (A, t) where $A \subseteq S$ and $t \in S$.
- Parameter:* A positive integer k .
- Question:* Is there a set $S_0 \subseteq S$ with $|S_0| \leq k$ and a positive integer n such that if we define S_i , $1 \leq i \leq n$, to consist of exactly those $t \in S$ for which either $t \in S_{i-1}$ or there exists a set $U \subseteq S_{i-1}$ such that $(U, t) \in R$, then $S_n = S$?

The decisive result is the following, announced in Ref. [179], but whose full proof only appeared in Ref. [4].

Theorem 13.3 ([179][4]) **MINIMUM AXIOM SET** is $W[P]$ -complete.

Lemma 13.4 **MINIMUM AXIOM SET** belongs to **MONOTONE** $W[P]$.

Proof. We describe a parameterized polynomial-time transformation of an instance (S, R) of **MINIMUM AXIOM SET** to a circuit C that accepts a weight k vector if and only if (S, R) has an axiom set of size k . For convenience of description, we view C as a directed acyclic graph for which (1) each vertex is assigned a logic function in the set $\{\wedge, \vee, 1\}$, where 1 denotes the identity function, and (2) some (appropriate) vertices have been designated as inputs.

Suppose $|S| = n$. The vertex set of C is $V(C) = V_1 \cup V_2$, where

$$V_1 = \{t[u, i] : u \in S, 0 \leq i \leq n\}.$$

Each vertex of V_1 is assigned the identity logic function. The inputs to C are the vertices $\{t[u, 0] : u \in S\}$. We may think of each vertex of V_1 in the circuit C as representing a boolean variable, the meaning of which is, “statement u is true at time i .” Thus, a weight k input to C represents k statements (the axioms) being true at time 0.

The vertices of V_2 are the union of n^2 sets of vertices

$$V_2 = \bigcup \{V[u, i] : u \in S, 1 \leq i \leq n\},$$

where the vertices of $V[u, i]$ implement a monotone circuit $C(u, i)$ that computes the value of $t[u, i]$ from the value of variables at the $i - 1$ level of V_1 , in accordance with the definition of the sets S_i in the description of the **MINIMUM AXIOM SET** problem. We may describe $C(u, i)$ by the sum-of-products expression

$$C(u, i) = t[u, i - 1] + \Sigma_{(A, u) \in R} \left(\prod_{v \in A} t[v, i - 1] \right).$$

The correctness of this construction is straightforward, noting that by time n , all of the statements deducible from the axioms will have been deduced.

Lemma 13.5 *MINIMUM AXIOM SET is hard for $W[P]$.*

Proof. Let C be a decision circuit for which we wish to determine whether there is a boolean vector of weight k accepted by C . We describe how C can be transformed in parameterized polynomial time into an instance (S, R) of the **MINIMUM AXIOM SET** problem that has an axiom set of size $2k$ if and only if C accepts a weight k input vector.

We view the circuit C as a directed acyclic graph $C = (V, A)$. We may assume that the circuit C is *leveled* in the sense that the vertices of V are partitioned into m sets $V[1], \dots, V[t]$ (“levels”), with the inputs constituting level 1, and such that for every arc $uv \in A$, if u is in level i , then v is in level $i + 1$. Since C is a general circuit, each vertex of $V - V[1]$ is assigned a logic function from the set $\{\wedge, \vee, \neg\}$. Thus, $V - V[1]$ can be expressed as the disjoint union

$$V - V[1] = V_{\wedge} \cup V_{\vee} \cup V_{\neg}.$$

We assume without loss of generality that $\wedge$ and $\vee$ vertices have in-degree 2 and that $\neg$ vertices have in-degree 1. We will assume that C has n inputs $V[1] = \{u_0, u_1, \dots, u_{n-1}\}$. Let z denote the output vertex of C .

The set of “statements” S for the instance (S, R) of **MINIMUM AXIOM SET** is the union of three sets, $S = S[1] \cup S[2] \cup S[3]$. Similarly, we describe the deductive structure R as the union of several parts, $R = R[1] \cup R[2] \cup R[3] \cup R[4] \cup R[5]$. The deductive structure $R[1]$ on $S[1]$ has the primary role of simulating the logic gates of the circuit C . The deductive structure $R[2]$ on $S[1]$ and $S[2]$ handles the inputs to the simulation of C . The role of $R[3]$ and $R[4]$ is to serve as an enforcement mechanism for the input simulation. The role of $R[5]$ is in handling the output of the simulation of C :

$$S[1] = \{q[u, i] : u \in V, i \in \{0, 1\}\}$$

The simulation of C provided by the deductive structure on $S[1]$ is straightforward. Note that each vertex u of C is represented by two “statements” $q[u, 0]$ and $q[u, 1]$ in $S[1]$. The deductive structure (S, R) simulates C in the following general way. First, the structure provided by $S[2]$ enforces (in a manner explained below) that “at time 1” exactly one of $q[u, 0]$ and $q[u, 1]$ has been deduced for each input vertex $u \in V_0$ (that is, will belong to the set S_1 in the definition of

the **MINIMUM AXIOM SET** problem), for any $2k$ element axiom set with any hope of success (let us call this “meaningful input” for the moment). Inductively, the simulation of C ensures that for meaningful input, for each vertex $q \in V$ and for $i = 1, \dots, t$, exactly one of $q[u, 0]$ and $q[u, 1]$ will belong to S_i .

In describing the simulation, we interpret this in the following way: If $q[u, 0] \in S_t$, then the vertex (logic gate) q of C has logic value 0 at time t , and if $q[u, 1] \in S_t$, then q has logic value 1 at time t . It may eventually transpire (at time t , since the circuit is leveled) that for the output vertex z of C , the statement $q[z, 1]$ is deduced. The structure (S, R) is such that if $q[z, 1]$ is deduced (at time t), then “everything” in S can be deduced at time $t + 2$. The details of the circuit simulation are as follows.

For $u \in (V_\wedge \cup V_\vee)$, let u' and u'' denote the input vertices to u , that is, the two vertices for which $u'u$ and $u''u$ are in the set of arcs A describing the circuit. Similarly, for $u \in V_\neg$, let u' denote the single input vertex to u . The implications of $R[1]$ provide the circuit simulation. We have $R[1] = R_\wedge \cup R_\vee \cup R_\neg$, where

$$\begin{aligned} R_\wedge &= \bigcup_{u \in V_\wedge} \{(\{q[u', 1], q[u'', 1]\}, q[u, 1]), (\{q[u', 0]\}, q[u, 0]), \\ &\quad (\{q[u'', 0]\}, q[u, 0])\}, \\ R_\vee &= \bigcup_{u \in V_\vee} \{(\{q[u', 0], q[u'', 0]\}, q[u, 0]), (\{q[u', 1]\}, q[u, 1]), \\ &\quad (\{q[u'', 1]\}, q[u, 1])\}, \\ R_\neg &= \bigcup_{u \in V_\neg} \{(\{q[u', 1]\}, q[u, 0]), (\{q[u', 0]\}, q[u, 1])\} \end{aligned}$$

We next describe the structure which simulates the input to the circuit. This construction is based on the ideas of Theorem 12.3. We have $S[2] = S[2, 1] \cup S[2, 2] \cup S[2, 3] \cup S[2, 4] \cup S[2, 5]$, where

$$\begin{aligned} S[2, 1] &= \{a[r, s, i] : 0 \leq r \leq k-1, 0 \leq s \leq n-1, 1 \leq i \leq 2\}, \\ S[2, 2] &= \{b[r, s, t, i] : 0 \leq r \leq k-1, 0 \leq s \leq n-1, \\ &\quad 1 \leq t \leq n-k+1, 1 \leq i \leq 2\}, \\ S[2, 3] &= \{a'[r, u, i] : 0 \leq r \leq k-1, 1 \leq u \leq 2k+1, 1 \leq i \leq 2\}, \\ S[2, 4] &= \{b'[r, u, i] : 0 \leq r \leq k-1, 1 \leq u \leq 2k+1, 1 \leq i \leq 2\}, \\ S[2, 5] &= \{d[r, s, i] : 0 \leq r \leq k-1, 0 \leq s \leq n-1, 1 \leq i \leq 2\}. \end{aligned}$$

Note that the set of statements $S[2]$ can be partitioned into two sets $S[2] = S'[2] \cup S''[2]$ according to the value of the last index of the elements; that is, let $S'[2]$ be those elements of $S[2]$ with last index $i = 1$ and let $S''[2]$ be those elements of $S[2]$ with last index $i = 2$. For $x' \in S'[2]$, let x'' be the element of $S''[2]$ that corresponds to x' by changing the last index from $i = 1$ to $i = 2$. Similarly, we define the notation $S'[2, j]$ for $j = 1, \dots, 5$.

We can now describe the second set of implications

$$R[2] = \{(\{a[r, i, 1] : 0 \leq r \leq k-1\}, q[u_i, 1]) : 0 \leq i \leq n-1\} \\ \cup \{(\{b[r, s, t, 1]\}, q[u_i, 0]) : s < i < s+t, 0 \leq i \leq n-1\},$$

We next introduce as an intermediate descriptive device, a graph $G = (V, E)$ (the *domination graph*) on the vertex set $V = S'[2]$. For convenience, we introduce notation for the following sets of vertices in this graph:

$$A(r) = \{a[r, s, 1] : 0 \leq s \leq n-1\}, \\ B(r) = \{b[r, s, t, 1] : 0 \leq s \leq n-1, 1 \leq t \leq n-k+1\}, \\ B(r, s) = \{b[r, s, t, 1] : 1 \leq t \leq n-k+1\}.$$

The edge set E of G is the union of the following sets of edges. In these descriptions we implicitly quantify over all possible indices.

$$E_1 = \{a[r, s, 1]a[r, s', 1] : s \neq s'\}, \\ E_2 = \{b[r, s, t, 1]b[r, s, t', 1] : t \neq t'\}, \\ E_3 = \{a[r, s, 1]b[r, s', t, 1] : s \neq s'\}, \\ E_4 = \{b[r, s, t, 1]d[r, s', 1] : s' \neq s+t \pmod{n}\}, \\ E_5 = \{a[r, s, 1]a'[r, u, 1]\}, \\ E_6 = \{b[r, s, t, 1]b'[r, u, 1]\}, \\ E_7 = \{d[r, s, 1]a[r', s, 1] : r' = r+1 \pmod{n}\}.$$

We use this graph in describing the third set of implications as follows. For a vertex $v \in V$, the *closed neighborhood* $N[v]$ of v is defined to be $N[v] = \{w : vw \in E\} \cup \{v\}$:

$$R[3] = \{(N[x'], x'') : x' \in V = S'[2]\},$$

The third set of statements is

$$S[3] = \{e[i] : 1 \leq i \leq 2k+1\}.$$

The final sets of implications are

$$R[4] = \{(\{S''[2] \cup \{q[z, 1]\}, e[i]) : 1 \leq i \leq 2k+1\}, \\ R[5] = \{(S[3], s) : s \in S[1] \cup S[2]\}.$$

This completes the description of the instance (S, R) of **MINIMUM AXIOM SET**. We argue the correctness of the reduction as follows.

We first describe an *interpretation* of a set U of $2k$ vertices in the domination graph, as this plays a key role in the argument. We interpret $a[r, s, 1] \in U$ as the directive, “set the input u_s of C to 1.” We interpret $b[r, s, t, 1] \in U$ as the directive,

“set the inputs u_i of C to 0, for all $i, s < i < s+t$.” If these directives are consistent for the vertices in a set U , call U a *consistent* set of vertices in the domination graph. Say that U is *total* if the directives corresponding to the vertices in U assign each input to C a value. If U is consistent and total, then the *interpretation* τ_U of U is the input to C corresponding to U according to the directives.

Claim 1. A dominating set D of $2k$ vertices in the domination graph G is consistent and total and the corresponding boolean input vector τ_D to the circuit C has weight k .

Proof of Claim 1. Suppose D is a dominating set of $2k$ vertices in G . The closed neighborhoods of the $2k$ vertices $a'[0, 1, 1], \dots, a'[k-1, 1, 1], b'[0, 1, 1], \dots, b'[k-1, 1, 1]$ are disjoint, so D must consist of exactly $2k$ vertices, one in each of these closed neighborhoods. Also, none of the vertices of $V_4 \cup V_5$ are in D , since if $a'[r, u, 1] \in D$, then necessarily $a'[r, u', 1] \in D$ for $1 < u' < 2k+1$ (otherwise D fails to be dominating), which contradicts that D contains exactly $2k$ vertices. It follows that D contains exactly one vertex from each of the sets $A(r)$ and $B(r)$ for $0 \leq r \leq k-1$.

The possibilities for D are further constrained by the edges of E_3 , E_4 , and E_7 . The vertices of D in $S'[2, 1]$ represent the inputs set to 1 in a weight k input vector for the circuit C , and the vertices of D in $S'[2, 2]$ represent “intervals” of inputs set to 0 in a weight k input vector for C . Since there are k inputs to be set to 1 (in a weight k vector), there are, considering the indices of the variables mod n , also k intervals of inputs to be set to 0.

The edges of E_3 , E_4 , and E_7 enforce that the $2k$ vertices in D must represent such a choice consistently. To see how this enforcement works, suppose $a[3, 4, 1] \in D$. This represents that the third of k distinct choices of inputs to be given the value 1 is the input u_4 . The edges of E_3 force the unique vertex of D in the set $B(3)$ to belong to the subset $B(3, 4)$. The index of the vertex of D in the subset $B(3, 4)$ represents the difference (mod n) between the indices of the third and fourth choices of an input to receive the value 1, and thus the vertex represents a range of inputs to receive the value 0. The edges of E_4 and E_7 enforce that the index t of the vertex of D in the subset $B(3, 4)$ represents the “distance” to the next input to receive the value 1, as it is represented by the unique vertex of D in the set $A(4)$.

In this way, a dominating set D of size $2k$ is forced to be both consistent and total, and we may note also that τ_D necessarily has weight k .

Claim 2. If C accepts a weight k input vector, then (S, R) is a yes instance of MINIMUM AXIOM SET.

Proof of Claim 2. Let τ denote an input boolean vector of weight k accepted by C , that assigns the inputs $V_\tau = \{u_{i_0}, u_{i_1}, \dots, u_{i_{k-1}}\} \subseteq V[1]$ the value 1. Suppose $i_0 < i_2 < \dots < i_{k-1}$. Let $d_r = i_{r+1 \pmod k} - i_r \pmod n$ for $r = 0, \dots, k-1$. It is straightforward to verify that the set of $2k$ vertices

$$D = \{a[r, i_r, 1] : 0 \leq r \leq k-1\} \cup \{b[r, i_r, d_r] : 0 \leq r \leq k-1\}$$

is a dominating set in the domination graph G and that $\tau_D = \tau$.

We take the initial set of axioms $S_0 = D$. We will argue that $S_{t+2} = S$. Since D is a dominating set in the graph on $S'[2]$, by the implications in $R[3]$ (based on closed neighborhoods in the domination graph), we may deduce that $S''[2] \subseteq S_1$.

The implications in $R[2]$ yield that

$$S[1] \cap S_1 = \{q[u, 1] : u \in V_\tau\} \cup \{q[u, 0] : u \in V[1] - V_\tau\},$$

consistent with our description of the circuit simulation (at time 1).

By induction on i , noting the form of the implications in $R[1]$, it is straightforward that the circuit simulation is correct; that is, for $i = 2, \dots, t$, we have

$$\begin{aligned} S[1] \cap S_i &= \{q[u, 1] : u \text{ evaluates to 1 in } C(\tau)\} \\ &\cup \{q[u, 0] : u \text{ evaluates to 0 in } C(\tau)\} \end{aligned}$$

Since C accepts τ , $q[z, 1] \in S_t$, and consequently by the implications in $R[4]$, $S[3] \subset S_{t+1}$. By the implications in $R[5]$, this yields $S \subseteq S_{t+2}$.

Claim 3. If $(S, R, 2k)$ is a yes instance of **MINIMUM AXIOM SET**, then C accepts a weight k input vector.

Proof of Claim 3. Let S_0 be the set of $2k$ axioms. If S_0 does not contain $2k$ statements in $S'[2]$, then by Claim 1, S_0 is not a dominating set in the domination graph, and, consequently, there is at least one statement $s \in S''[2]$ that can only be deduced if every statement in $S[3]$ is first deduced (or is an axiom). Not every statement in $S[3]$ can be an axiom (since the cardinality of $S[3]$ is $2k + 1$), so there must be some statement $s' \in S[3]$ that is properly deduced, necessarily by an implication in $R[4]$, prior to the deduction of s . An examination of $R[4]$ shows that s must be deduced prior to s' , a contradiction. Thus, $S_0 \subseteq S'[2]$, and furthermore, S_0 must be a dominating set in G .

By Claim 1, S_0 must be consistent and total, and we may therefore consider the interpretation τ_{S_0} , which we argue is accepted by C . The only possibility for statements in $S'[2] - S_0$ to be deduced is by the implications in $R[5]$, and these can only be applied if $q[z, 1]$ has been deduced. By the inductive argument for the correctness of the circuit simulation, this can only happen if C accepts the weight k input vector τ_{S_0} .

That completes the proof of the lemma and hence of the theorem.

Corollary 13.6 **WEIGHTED MONOTONE CIRCUIT SATISFIABILITY** is $W[P]$ -complete; that is, $MONOTONE W[P] = W[P]$.

Proof. By Lemma 13.4 **MINIMUM AXIOM SET** is a special case of **WEIGHTED MONOTONE CIRCUIT SATISFIABILITY**.

The reader might wonder if instead of the intricate argument above, we could simply use the reasoning of the Normalization Theorem as follows. First, move all the inverters to the top of the circuit. Now, consider each line as a “gate” with fanin

1 and fanout 1. Hardwire the **DOMINATING SET** gadget from the Normalization Theorem (more correctly, the gadget from Lemma 12.4). This would increase the weft by 1 but make the circuit monotone, and since we do not care about controlling weft—only size—this would seem to give an easy construction of a circuit where the original circuit has a weight k satisfying assignment iff the new one has a weight $2k$ satisfying assignment.

The problem is that *the action of moving the inverters to the top can increase the size of the circuit exponentially in the height*. This is because of the fact that the circuits are not *tree circuits*, they are only *decision circuits*, and gates can, thus, have *unbounded* fanout. However, the reasoning here can clearly be applied to tree circuits; hence, we can prove a similar result to Theorem 13.6 for $W[SAT]$. [Theorem 13.7 (ii) follows by similarly hardwiring the gadget from Theorem 10.3.]

Theorem 13.7 ([4]) *The following are complete for $W[SAT]$.*

(i) **MONOTONE $W[SAT]$** .

(ii) **ANTIMONOTONE $W[SAT]$** .

The basic results above form a basis for many of the reductions for various concrete problems. The next theorem gives some illustrations. More can be found in the Exercises. We will need the following definitions:

DEGREE 3 SUBGRAPH ANNIHILATOR

Instance: A graph G .

Parameter: A positive integer k .

Question: Is there a set V' of at most k vertices of G , such that $G - V'$ has no minimum degree 3 subgraph?

k -LINEAR INEQUALITIES

Instance: A system of linear inequalities.

Parameter: A positive integer k .

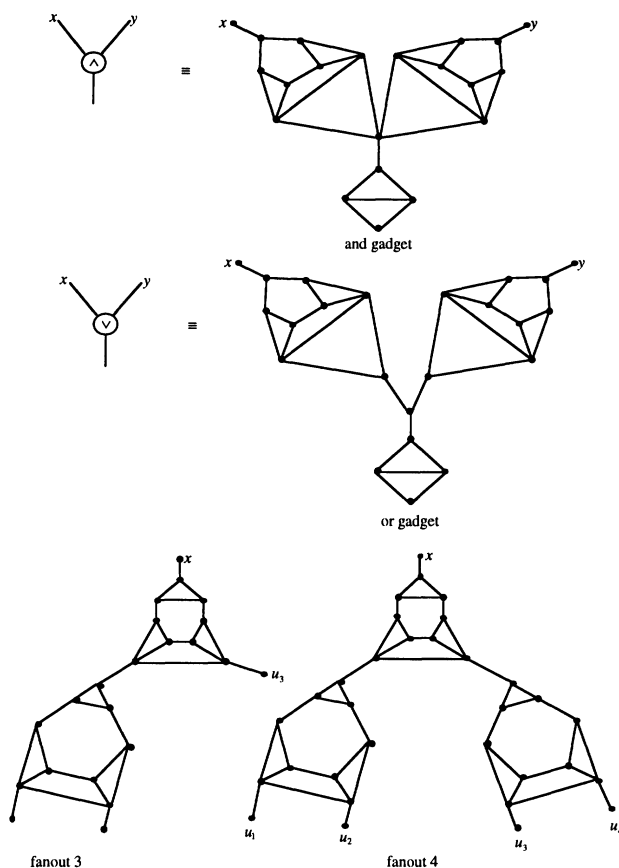
Question: Can the system be made consistent over the rationals by deleting at most k of the inequalities?

Theorem 13.8 ([3, 4]) *The following problems are $W[P]$ -complete.*

(i) **DEGREE 3 SUBGRAPH ANNIHILATOR**,

(ii) **k -LINEAR INEQUALITIES**,

Proof. (i) We reduce from **WEIGHTED MONOTONE CIRCUIT SATISFIABILITY**. Let C be an instance of **WEIGHTED MONOTONE CIRCUIT SATISFIABILITY** with parameter k . We can regard all gates to have only two inputs with only a quadratic increase in the size of the circuit. We shall use local replacement. We replace the *and* and *or* gates by the gadgets given in Figure 13.1. Formally, we can define the *and* gadget $AND(x, y, z)$ for a gate $x \wedge y$ with output z via the vertices $v(x, j)$ and $v(y, j)$ for $1 \leq j \leq 7$ together with vertex $q(x, y)$ and “pendant vertices” $l(x, y, p)$ for $p = 1, 2, 3, 4$.

FIGURE 13.1. Gadgets for **DEGREE 3 SUBGRAPH ANNIHILATOR**

The edges are for $r \in \{x, y\}$, $rv(r, 1), v(r, 1)v(r, 2), v(r, 1)v(r, 3), v(r, 2)v(r, 4), v(r, 3)v(r, 5), v(r, 4)v(r, 5)$. (These are the “pentagonal” edges of the gadget.) $v(r, 2)v(r, 6), v(r, 4)v(r, 6), v(r, 6)v(r, 7), v(r, 3)v(r, 7), v(r, 5)v(r, 7)$. (These are the edges connected to the pentagon.) $v(r, 6)q(x, y), v(r, 7)q(x, y), q(x, y)l(x, y, 1)$. (These are the connecting edges.) $l(x, y, 1)l(x, y, 2), l(x, y, 1)l(x, y, 3), (x, y, 2)l(x, y, 3), l(x, y, 2)l(x, y, 4), l(x, y, 3)l(x, y, 4)$. (These are the edges of the “pendant diamond.”)

The vertices for the *or* gadget $OR(x, y, z)$ are $w(x, j), w(y, j)$ for $1 \leq j \leq 7$ and three additional vertices $m(x), m(y), n(x, y)$ and $g(x, y, p)$ for $p = 1, 2, 3, 4$. The vertices $w(r, j)$ are connected among themselves exactly as we did for the $v(r, j)$ above, as are the pendant $g(x, y, p)$. We also have edges $w(r, 6)m(r), w(r, 7)m(r), m(r)n(x, y)$, and $n(x, y)z$.

Additionally, we shall need another sort of gadget, a *fanout* gadget which is used to cope with the fanout occurring at gates and in the construction. This gadget is also indicated in Figure 13.1 for fanouts 3 and 4.

Let the variables of C be $x[1], \dots, x[n]$. For the construction of the graph $G(C)$, we first replace all the gates of C and their fanouts by the gadgets of Figure 13.1. Call the result $I(C)$. Using $I(G)$, we make another intermediate graph $I'(G)$ as follows. For each $x[i] \in \{x[1], \dots, x[n]\}$, create a subgraph consisting of the (new) vertices $v(i, j) : j = 1, \dots, 4$ and edges $v(i, 1)v(i, 2)$, $v(i, 1)v(i, 3)$, $v(i, 2)v(i, 3)$, $v(i, 2)v(i, 4)$, and $v(i, 3), v(i, 4)$. Form $2k + 1$ copies of the $I(C)$, except that in place of the $k + 1$ versions of the input variable $x[i]$, we connect all relevant gates to the *single* vertex $v(i, 4)$. [Thus, if in the original circuit the variable $x[i]$ had fanout m , then in the graph $I'(C)$, $v(i, 4)$ will have degree $m(k + 1) + 2$.] Finally, we form the final graph $I''(C)$ by taking the vertex $u(i)$ representing the output of the i -th version of $I(C)$ used in $I'(C)$, and for each $j = 1, \dots, n$, join $o(i)$ to $v(j, 1)$ via one of the outputs u_j of a fanout gadget of fanout n . This concludes the construction.

To see that the construction succeeds, we show that C has a weight k accepting input iff $I''(C)$ has no k -element degree 3 subgraph annihilator. First, suppose that C has a weight k accepting input. Let $x[i_1], \dots, x[i_k]$ be the true variables in some satisfying assignment. Remove the vertices $v[q, 4]$ for $q \in \{i_1, \dots, i_k\}$ from $I''(C)$. We claim that the result has no min degree 3 subgraph. We shall describe a process that removes edges that are useless to a min degree 3 subgraph and eventually kills all of $I''(C)$. The reader should note the following guiding principle to our proof. Given any graph G , the following simple process determines if G has a min degree 3 subgraph.

1. Repeat until there are no degree ≤ 2 vertices:
Find any vertex v of degree < 3 . Let $G \leftarrow G - v$.
2. G has a min degree 3 subgraph iff the result of 1 is nonempty.

In our construction, the interpretation is that “ x removed” equates to “ x is true.” Suppose vertex x of the and gadget $AND(x, y, z)$ is removed. This will cause the vertex $v(x, 1)$ at the top of the gadget that is joined to x to have degree 2. Thus, we may remove $v(x, 1)$ from $I''(C)$ since it cannot be part of a min degree 3 subgraph. In turn, this decreases the degree of $v(x, 2)$ and $v(x, 3)$ to 2. Arguing in this way, we see that all of the vertices $v(x, j)$ will need to be removed. Note that this causes $l(x, y)$ to have degree 3. Thus, if one input of *and* gate is removed, then the effect is to remove that half of the gadget.

In the case of an *or* gate, we can similarly argue that the removal of either of the inputs will cause the whole gadget to be removed, and the vertices corresponding to the other input and the one corresponding to z to have their degree reduced by 1.

Translating the gadget observations back to C , it follows that if a gate is made true, then the corresponding gadget can be removed from $I''(C)$. Since the given assignment is one that satisfies C , it follows that any vertex u representing the output of a copy of C can be erased. Consequently, using the same reasoning, we can erase all the vertices of the fanout gadgets, thus dropping the degree of the vertices $v[i, 1]$ by 1. This then allows us to erase the vertices $v[i, j]$ for all i and j and thus erase all of $I''(C)$. Thus, $I''(C) - \{v[i_1, 4], \dots, v[i_k, 4]\}$ has no min degree 3 subgraph.

Conversely, suppose that $I''(C)$ has a collection V' of k vertices such that $G = I''(C) - V'$ has no min degree 3 subgraph. It follows that except for variable components, V' must be disjoint from $k + 1$ of the circuit copies. As the circuit copies are identical, we may suppose without loss of generality that V' has no internal circuit vertices. Now, the removal of V' must cause the graph to be erased as above. This means that the removal of V' and the process above must cause *all* of the output nodes u to be removed, or else we can add the nine nodes below the fanout immediately below u and have a min degree 3 subgraph. Since all the output vertices must be removed by V' , and not all can be in V' , it follows that the circuit above at least one output vertex u must be removed. (Fan out gadgets can only be removed from above, not below.) Since there are $2k + 1$ copies of the circuit, it follows that for one such u , its removal can only be generated by the removal of a set of variables, which we can take to be $\{v[i_1, 4], \dots, v[i_m, 4]\}$. Now, we can argue that since u is removed, if u corresponds to an *and* gate, then *both* of its inputs must be removed, and if u is an *or* gate, then at least one of its inputs must be removed. Chasing the removed inputs up the circuit, we can argue that eventually a set of vertices must be removed, and we see that this set of $\leq k$ vertices cause the circuit to be erased. The reason these $\leq k$ vertices are able to cause this removal is the following. Apply the removal process to the initial set V' . As the removal process discharges through the circuit the “first” time, only vertices corresponding to variables can cause the removal of u . But if u is not removed on the first sweep, then it will never be removed. As above, they must correspond to a satisfying assignment for C . Since C is monotone, they can be extended to a weight k satisfying assignment for C . Thus, C is satisfiable by a weight k assignment iff $I''(C)$ has a size k degree 3 subgraph annihilator, and hence **DEGREE 3 SUBGRAPH ANNIHILATOR** is $W[P]$ -hard.

(ii) This time we employ the reduction from Ref. [5]. Cook observed that if C is a circuit with input variables $x = (x_1, \dots, x_n)$ and output variables $y = (y_1, \dots, y_m)$, then there is a *LOGSPACE* procedure that, on input C , produces a set L of linear inequalities with the properties below.

- (a) The inequalities include variables $X = (X_1, \dots, X_n)$ and $Y = (Y_1, \dots, Y_m)$.
- (b) For all binary vectors $b = (b_1, \dots, b_n)$, there is exactly one point that satisfies all the inequalities of L and additionally satisfies $X = b$.
- (c) If c is the output of C on input b , then the unique assignment that satisfies L and $X = b$ also satisfies $Y = c$.

L is said to simulate C . (The proof of this result is the following: represent *not* gates $a = \bar{b}$ by $a = 1 - b$ and $0 \leq a \leq 1$, and *and* gates $a = b \wedge c$ by $0 \leq a \leq 1$, $a \leq b$, $a \leq c$, and $b + c - 1 \leq a$.) We reduce from **WEIGHTED CIRCUIT SATISFIABILITY**. Let C be an instance of **WEIGHTED CIRCUIT SATISFIABILITY** with q input variables $x_1, \dots, x_q$. Let $X_1, \dots, X_q$ be rational variables. Consider the inequalities

$$X_i \leq 0, \text{ and } k + 1 \text{ copies of both } X_1 + \dots + X_q \geq k, \text{ and } 0 \leq X_i \leq 1.$$

To this system we add $k + 1$ copies of the inequalities representing C with $X_1, \dots, X_q$ also representing the input variables. We also add a variable Z to

represent the output variable and add the inequality $Z \geq 1 - k + 1$ times. Clearly, this system is satisfiable by the deletion of k of the inequalities (which must be the $X_i \leq 0$ type) iff C has a weight k satisfying assignment.

Exercises 13.0

13.0.1. (Abrahamson, Downey, and Fellows [3, 4]) Prove that the following problems are complete for $W[P]$.

(i) **t -THRESHOLD STARTING SET**

Instance: A digraph D .

Parameter: A positive integer k .

Question: Is there a set of k vertices of D with the property that it will pebble D under the rule that a vertex can be pebbled iff at least t incoming vertices are pebbled?

(ii) **CHAIN REACTION CLOSURE**

Instance: A directed graph D .

Parameter: k .

Question: Does there exist a set V' of k vertices of D whose chain reaction closure is D . Here the *chain reaction closure* of V' is the smallest superset S of V' such that if $u, u' \in S$ and arcs $ux, u'x$ are in D then $x \in S$.

(iii) **k -INDUCED SATISFIABILITY**

Input: A formula φ .

Parameter: A positive integer k .

Question: Is there a set of k variables, and a truth table assignment to those variables that causes φ to unravel?

[Hint: We look at (i); (ii) and (iii) are similar. Use the technique of Theorem 13.8 and reduce from **WEIGHTED MONOTONE CIRCUIT SATISFIABILITY**. This time we use the *and*, *or*, and *fanout* gates of Figure 13.2, where the arcs go *down*. In the construction, this time we represent the variables by pairs of vertices $x[i], y[i]$. Again, we form $I''(C)$ by replacing the gates in the circuit C by gadgets as above and then taking $2k+1$ copies of the for $i = 1, \dots, n$. Again, each *pair* that represents an output for a circuit fans out to *and* gates above each of the variable pairs (or more precisely, to collections of gadgets corresponding to large *and* gates with outputs the variables). Note, for instance, that it requires both of the input pairs of the *and* gadget to be in the closure before the output pair will be included. Thus, for the variable pairs to be included, we need all of the output pairs corresponding to the circuits to be included. Thus, as in the previous argument, the original collection of $\leq 2k$ variable pairs included in V' cannot cause any more variable pairs to be included unless they first cause the

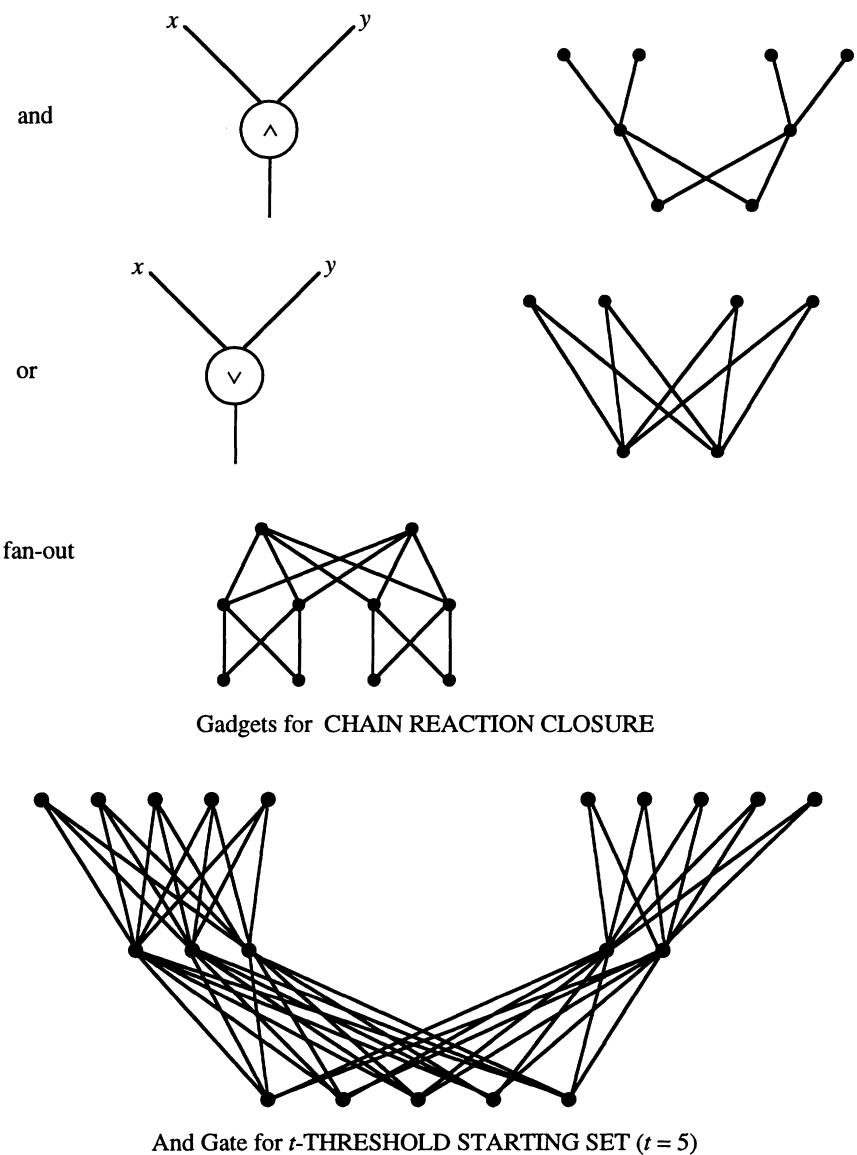
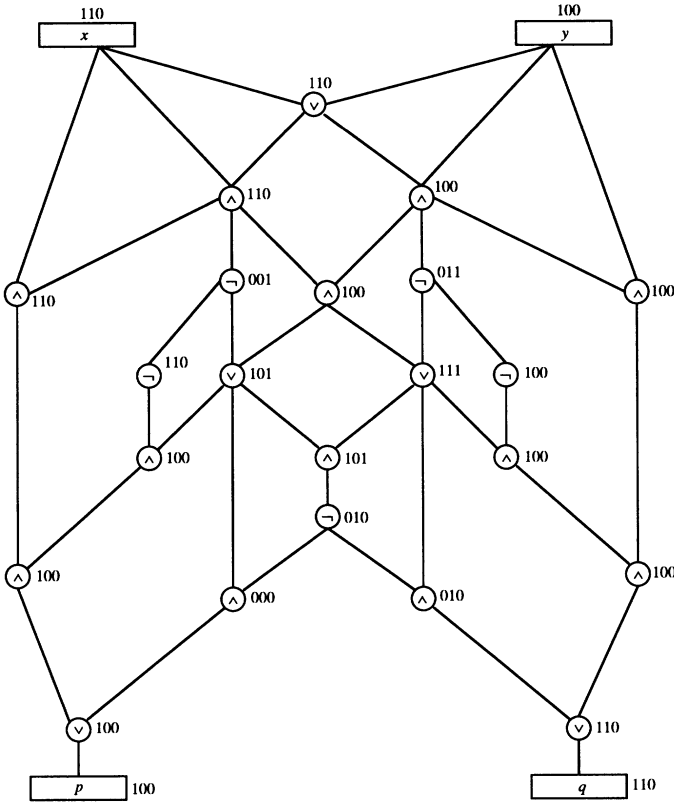


FIGURE 13.2. Gadgets for Exercise 1

inclusion of all of the circuit output pairs *first*. But as in the previous argument, this corresponds to a satisfying assignment of C .]

13.0.2. (Abrahamson, Downey, and Fellows [4]) Prove that **WEIGHTED PLANAR CIRCUIT SATISFIABILITY**, the restriction of **WEIGHTED**

FIGURE 13.3. Gadget for **WEIGHTED PLANAR CIRCUIT SATISFIABILITY**

CIRCUIT SATISFIABILITY to planar circuits (i.e., can be drawn on the plane with no edges crossing), is complete for $W[P]$.

[Hint: Reduce **WEIGHTED CIRCUIT SATISFIABILITY** to the planar version. This involves no more than the use of a crossover gadget along the lines of Lichtenstein [332] or Garey and Johnson [239]. Take an instance φ of **WEIGHTED CIRCUIT SATISFIABILITY** with parameter k . Suppose a wire from x (a variable *or* gate) is crossing one from y . We replace this crossing by the gadget of Figure 13.3. In this figure we have listed the values at each gate as a triple corresponding to the settings of x and y . Note that we need only specify the values for a triple by symmetry. Since p agrees with y and q agrees with x , we have achieved the desired crossing.]

- 13.0.3.** (Abrahamson, Downey, and Fellows [4]) Prove that **WEIGHTED MONOTONE SATISFIABILITY** (the restriction of **WEIGHTED SATISFIABILITY** to monotone formulae) and **WEIGHTED ANTIMONOTONE SATISFIABILITY** (the restriction of **WEIGHTED SATISFIABILITY** to antimonotone formulas) are both $W[P]$ complete.

[Hint. Hardwire the gadgets used in the Normalization Theorem.]

13.0.4. (Downey and Fellows) Prove that the following problem is complete for $W[P]$.

k -BASED TILING

Instance: A tiling system with distinguished tiles.

Parameter: A positive integer k .

Question: Is there a tiling of the $n \times n$ plane using the tiling system and starting with exactly k distinguished tiles in a line ?

(Hint: Use a generic simulation of a Turing machine.)

14

Fixed Parameter Analogs of *PSPACE* and *k*-Move Games

As we have seen, classical time complexity classes such as *NP* seem to split into many parameterized classes when natural parameterized versions of the problems are considered. So too, the same situation seems to occur when we look at parameterized *space*. Of course, a natural parameterized space class suggests itself if we wish to consider fixed-parameter space complexity.

Definition 14.1 We say that a parameterized language L is in *SLICEWISE PSPACE* if there is a procedure Φ , a function f , and a constant α such that for all k ,

$$\langle x, k \rangle \in L \text{ iff } \Phi(\langle x, k \rangle) \text{ accepts, and the space bound on } \Phi(\langle x, k \rangle) \text{ is } f(k)|x|^\alpha.$$

As the Liming, Cai, Chen, Downey, and Fellows [110] observed, one can similarly define a parameterized class *SLICEWISE C* for any classical complexity class C . This was the basis for the “advice view” of Chapter 5 and Exercises 5. The reader should note that since there is a set in $DSPACE(|x|^{c+1})$ which in linear time can compute any language in $DSPACE(|x|^c)$, it follows that:

Observation. If $P = PSPACE$, then *SLICEWISE PSPACE* = *FPT*.

In this chapter, the main goal is to point out a very interesting connection between parameterized versions of space complexity and the complexity of k -move games. Of course, many classical game problems are known to be *PSPACE*-complete. Typically, such problems ask whether the first player to move has a winning strategy. The reason that the classical problems are *PSPACE*-complete stems from the following facts:

QBFSAT is *PSPACE*-complete.

Games are naturally in **QBF** form since the alternating plays can be thought of as alternating quantifiers.

A natural parameterized version of classical games is whether the first player has a strategy that wins within at most k moves. This, in turn, leads to natural parameterized versions of **QBFSAT** (**QUANTIFIED BOOLEAN FORMULA SATISFIABILITY**), but, unfortunately, it seems again that there is no coincidence between *SLICEWISE PSPACE* and any parameterized form of **QBFSAT**. (The problem is that the classical proofs that **QBFSAT** is *PSPACE*-complete uses alternations of quantifiers to emulate *PSPACE* by guessing intermediate steps along the lines of Savitch's theorem. Unfortunately, this process does not seem to translate well into the parameterized setting since the intermediate steps in a *SLICEWISE PSPACE* computation can well lose weight control. In fact, the appropriate analog of Savitch's theorem is not known in the parameterized setting.)

Parameterized versions of some hard game problems are fixed-parameter tractable. An example is the **ALTERNATING HITTING SET** game (Garey and Johnson [239], Schaefer [409]) restricted to sets of any fixed size $t \geq 2$, which is *PSPACE*-complete. We consider the problem:

RESTRICTED ALTERNATING HITTING SET

Instance: A collection C of subsets of a set B with $|S| \leq k_1$ for all $S \in C$.
Parameter: $\langle k_1, k_2 \rangle$.
Question: Does player I have a win in $\leq k_2$ moves in the following game? Players play alternatively and choose unchosen elements, until, for each $S \in C$, some member of S has been chosen. The player whose choice this happens to be wins.

The reader can prove the following easy result in Exercise 1.

Theorem 14.2 (Abrahamson, Downey and Fellows [3][4]) **RESTRICTED ALTERNATING HITTING SET** is (strongly) fixed-parameter tractable.

On the other hand, some naturally parameterized game problems appear not to be fixed-parameter tractable. **GENERALIZED GEOGRAPHY** is a game played on a directed graph G with a distinguished start vertex [239, 409]. Players alternate choosing vertices, starting at the start vertex v_1 , in such a way that the chosen vertices form, in sequence, a simple directed path in G . The first player who is unable to choose a vertex loses. Determining whether player 1 has a winning strategy in a **GENERALIZED GEOGRAPHY** game is *PSPACE*-complete. A good candidate for a game problem that is not fixed-parameter tractable is **SHORT GENERALIZED GEOGRAPHY**, in which it is asked whether player 1 has a strategy that wins a given game of **GENERALIZED GEOGRAPHY** in at most k moves.

In order to address such questions, we introduce the classes $AW[P]$, $AW[SAT]$, and $AW[*]$, which plausibly contain problems that are not in *FPT*. The classes can be viewed as generalizations of the polynomial-Time Hierarchy and **QBFSAT**. We show that **SHORT GEOGRAPHY** is $AW[*]$ -complete.

Like $W[SAT]$, $AW[SAT]$ is the closure under parameterized reductions of a kernel problem of such a general nature that it appears not to be fixed-parameter tractable. This problem is a parameterized version of **QBF**, defined as follows.

PARAMETERIZED QBFSAT

- Instance:* A sequence $s_1, \dots, s_r$ of pairwise disjoint sets of boolean variables, and a boolean formula X involving the variables in $s_1 \cup \dots \cup s_r$.
- Parameters:* $r, k_1, \dots, k_r$.
- Question:* Is it the case that there exists a size k_1 subset t_1 of s_1 such that for every size k_2 subset t_2 of s_2 there exists a size k_3 subset t_3 of s_3 such that ... (alternating quantifiers) such that, when the variables in $t_1 \cup \dots \cup t_r$ are made true, and all other variables are made false, formula X is true?

Definition 14.3 $AW[SAT]$ is the set of all problems that fp -reduce to **PARAMETERIZED QBFSAT**.

Clearly, an equivalent formulation of this problem is

- Instance:* A quantified boolean formula $Q_1 x_1 \dots Q_n x_n X$.
- Parameter:* $k = \langle k_1, \dots, k_n \rangle$.
- Question:* Is $Q_1^{k_1} x_1 Q_2^{k_2} x_2 \dots Q_n^{k_n} x_n X$ true? (Here, $\exists^k x$ is interpreted to mean “does there exist a *weight* k x such that ...” and $\forall^k x$ is interpreted to mean “for all *weight* k x ...”)

Similarly, we can define the problem of **PARAMETERIZED QUANTIFIED CIRCUIT SATISFIABILITY** (**PARAMETERIZED QCSAT**), specified by replacing the X by a circuit with the variables x as the input. So with this interpretation, **WEIGHTED CIRCUIT SATISFIABILITY** is in **PARAMETERIZED** Σ_1 . We have a natural result which is a partial analog to the classical result that **QBFSAT** is $PSPACE$ -complete. We need the following problem definition.

COMPACT TM COMPUTATION

- Instance:* A nondeterministic Turing machine M and a word x .
- Parameter:* A positive integer k .
- Question:* Is there an accepting computation of M on input x that visits at most k work tape squares?

The following result improves the earlier work of Cai et al.[109] who proved that **COMPACT TM COMPUTATION** is $W[P]$ -hard.

Theorem 14.4 (Abrahamson, Downey and Fellows [4]) **COMPACT TM COMPUTATION** is $AW[P]$ -hard.

Proof. Let $X, s_1, \dots, s_n$ be an instance of **PARAMETERIZED QCSAT** with parameter $k = \langle k_1, \dots, k_n \rangle$. We shall use as a subroutine the method developed by Cai et al.[110] to prove $W[P]$ -hardness. This proof went as follows. The reduction is from **WEIGHTED MONOTONE CIRCUIT SATISFIABILITY**. Let C be a circuit for which we wish to determine whether there is an input vector of weight k accepted

by C . We may assume that each logic gate g of C has two inputs. In time polynomial in $|C|$, we can describe a Turing machine M sketched as follows.

M has an alphabet consisting of one letter for each input to C , and the operation of M consists of two phases. In the first phase, M makes k moves nondeterministically, writing down in the first k tape squares k symbols which represent k inputs to C set to 1. In the second phase (and visiting no other tape squares), M checks whether the guess made in the first phase represents a vector accepted by the circuit C .

The key point is that we can structure the transition table of M to accomplish this, with the size of the table polynomial in $|C|$. To do this, we make two states q_{up}^l and q_{down}^l for each connection (or *line*) l of the circuit C . Let g be an *and* gate of C , let l be an output line of g , and suppose the input lines to the gate g are l_1 and l_2 . We include in the transition table for M transitions from $q_{up}^{l_1}$ to q_{up}^l , from $q_{down}^{l_1}$ to q_{up}^l , and from $q_{down}^{l_2}$ to q_{down}^l . The significance of being a state q_{down}^l is that this represents a value of 1 for the line l as computed by C on the input guessed in the first phase. The state q_{up}^l might be viewed as a state of *query* about the value of l for the circuit C on the input guessed in the first phase. Note that the three transitions described above for the *and* gate g thus enforce that q_{down}^l can be reached only if $q_{down}^{l_1}$ and $q_{down}^{l_2}$ can be reached. The appropriate transitions for an *or* gate will differ in the obvious way; i.e., we arrange that the state q_{down}^l can be reached if either of $q_{down}^{l_1}$ or $q_{down}^{l_2}$ can be reached.

If l is an input line to the circuit C , then we encode in the state table for M a “check” (involving a scan of the k tape squares) to see if the corresponding input symbol was written during the first phase of computation. The second phase begins in the state $q_{up}^{l_{out}}$, where l_{out} is the output line of C , and the only accept state is $q_{down}^{l_{out}}$.

Note that the proof above gives a canonical way of going from a proposed collection t_i of true input variables to a Turing acceptance. Furthermore, note that there is no problem in considering only monotone circuits for this proof since the proof that **WEIGHTED MONOTONE CIRCUIT SATISFIABILITY** is $W[P]$ -complete lifts to one that proves that **PARAMETERIZED MONOTONE QCSAT** is $AW[P]$ -complete.

Thus, to complete the proof, we need to state how to introduce layers of quantifiers. Without loss of generality, we can suppose that Q_1 is existential. What we do is break the work tape into n cells of size $k_1, \dots, k_n$. Our first action is to write a guess for t_1 in the first k_1 squares. We build a recursive algorithm that accepts only if t_1 can be extended to a satisfying pattern according to the quantifier structure. For cell $i \geq 2$, if the quantifier corresponding to k_i is an existential one, then on each sweep, in the k_i squares of *cell* i which we will write a guess for the variables t_i from s_i we will be assigning true for this sweep. (We shall process the guesses in lexicographic ordering.) If the Q_i corresponding to cell i is a universal quantifier, then for each setting of the cells $1, \dots, i-1$, we will cycle lexicographically through all the possibilities for t_i , that is, all the k_i element subsets of s_i . As above, for each sweep, we will get a setting for cells corresponding to k true variables, and we can see if M accepts. Fix a setting of $1, \dots, t_{n-1}$. Recursively, note that if the last Q_n is universal, then cell n will pass back a confirmation of this setting only if it successfully cycles through all possible t_n . Similarly, if Q_n is existential, cell

n can pass a yes for this setting only if it finds a t_n . This process can be continued inductively using at most k counters; hence, we can make a machine that uses at most $2k$ squares and accepts iff $Q_1x_1 \dots Q_nx_nC$ is true.

Even $AW[P]$ appears to be too large a class for our purposes, again because of the bound on the logical depth of the statements of the problems. Instead, we concentrate on the class $AW[*]$ defined below.

Definition 14.5 **PARAMETERIZED QBFSAT_t** is the restriction of **PARAMETERIZED QBFSAT** in which the formula part is t -normalized. $AW[t]$ is the set of all parameterized problems that f p-reduce to **PARAMETERIZED QBFSAT_t**, and $AW[*] = \bigcup_t AW[t]$.

Lemma 14.6 (Abrahamson, Downey and Fellows [4]) **SHORT GEOGRAPHY** is in $AW[2]$.

Proof. We use a generic reduction. Let D be a digraph with distinguished vertex v_0 upon which we shall play **SHORT GEOGRAPHY** with parameter k . Let $\{v_0, \dots, v_n\}$ list the vertices of D and E the edge set. We shall have variables $\{p_{i,j} : 1 \leq i \leq k \wedge 0 \leq j \leq n\}$. We think of the game as a pebbling game with $p_{i,j}$ denoting that the i -th pebble is on vertex v_j . We need clauses as follows:

- (1) $p_{1,0}$. [Pebble 1 is on vertex v_0 .]
- (2) $\bigwedge_{1 \leq i \neq j \leq k, 0 \leq k \leq n} (p_{i,q} \rightarrow \overline{p_{j,q}})$ [Only one pebble per vertex tex.]
- (3) $\bigwedge_{v_i, v_j \notin E, 1 \leq q \leq k} (p_{q,i} \rightarrow (\overline{p_{q+1,j}} \wedge \overline{p_{q-1,j}}) \vee (\bigvee_{q' \leq q, q' \text{ odd}, 0 \leq t \leq n} p_{q',t} \wedge (\bigwedge_{v_i, v_r \in E} (\bigvee_{q'' \leq q'} p_{q'',r}))))$. [If $v_i v_j$, not an edge, then for any pebble placed on v_i , the preceding pebble and the next pebble must not be on v_j unless player 1 has already won on some vertex v_t pebbled with some $q' \leq q$.]
- (4) $\bigwedge_{2 \leq j \leq k, 0 \leq i \leq n} (p_{j,i} \rightarrow (\bigvee_{v_i v_q \in E} p_{j-1,q}) \vee (\bigvee_{q' \leq j, q' \text{ odd}, 0 \leq t \leq n} p_{q',t} \wedge (\bigwedge_{v_i, v_r \in E} (\bigvee_{q'' \leq q'} p_{q'',r}))))$. [If v_i is pebbled by pebble $j \geq 2$, then some vertex adjacent to v_i must be pebbled by pebble $j - 1$ unless player 1 has already won on some v_t as in (3).]
- (5) $\bigwedge_{1 \leq j \leq k-2, j \text{ odd}, 0 \leq i \leq n} (p_{j,i} \rightarrow (\bigvee (\bigvee_{q' \leq j, q' \text{ odd}, 0 \leq t \leq n} p_{q',t} \wedge (\bigwedge_{v_i, v_r \in E} (\bigvee_{q'' \leq q'} p_{q'',r})))) \vee ((\bigwedge_{v_i v_j \in E} \bigvee_{1 \leq q \leq j} p_{j,q}) \vee (\bigvee_{\{q,r: v_i v_q, v_q v_r \in E\}} (p_{j+1,q} \wedge p_{j+2,r}))))$. [If player 1 pebbles v_i , then either she wins at this play, has won at a preceding play, or player 2 pebbles a vertex v_q adjacent to v_i with the next pebble and player 1 pebbles a vertex adjacent to v_q .]

Let $P(p_{i,j} : 1 \leq i \leq k, 0 \leq j \leq n)$ denote the conjunction of (1)-(5) expressed in CNF form. Note that this expression has length polynomial in $\langle |D|, k \rangle$. It is by definition in $W[2]$. The expression we then need is the following.

$$\forall y_2 \in \{p_{2,0}, \dots, p_{2,n}\} \exists y_3 \in \{p_{3,0}, \dots, p_{3,n}\} \dots (k \text{ alternations}) P$$

This is interpreted as making the chosen variable from the set $\{p_{j,0}, \dots, p_{j,n}\}$ true and the others false. Note that the form of the expression P ensures that the formula is true iff player 1 has a $\leq k$ -move winning strategy.

We prove that **PARAMETERIZED QBFSAT** $_t$ fp -reduces to **SHORT GEOGRAPHY** for every t . The reduction is actually from a restricted form of **PARAMETERIZED QBFSAT** $_t$.

Definition 14.7 **UNITARY PARAMETERIZED QBFSAT** $_t$ is the restriction of **PARAMETERIZED QBFSAT** $_t$ in which the parameters $k_1, \dots, k_r$ are all 1.

Lemma 14.8 For $t > 0$, **UNITARY PARAMETERIZED QBFSAT** $_t$ is $AW[t]$ -complete.

Proof. The method is quite simple. A quantifier: $\exists k$ members of $(S = \{s_1, \dots, s_n\})(\dots)$, can be replaced by $2k$ quantifiers:

$$\exists x_1 \in S \forall y \in \emptyset \exists x_2 \in S \forall y \in \emptyset \dots \exists x_k \in S \forall y \in \emptyset (\wedge_{i \neq j} (x_i \neq x_j) \wedge \dots)$$

(Treat universal quantifiers similarly.) Note that the overall parameter is doubled. We also only add an additional large *and* of *or*'s to the circuit. This expression, in turn, can be put into standard form by replacing the i -th occurrence of S by $S_i = \{s_1^i, \dots, s_n^i\}$ and adding the expression $\wedge_{1 \leq i \leq k, j \in \{1, \dots, n\}} (s_j \equiv s_j^i)$. Again, there is no increase in the weft of the circuit.

Theorem 14.9 (Abrahamson, Downey and Fellows [3, 4]) **SHORT GEOGRAPHY** is $AW[*]$ -complete. Hence, $AW[*] = AW[2]$.

Proof. We reduce **PARAMETERIZED QBFSAT** $_{2t}$ to **SHORT GEOGRAPHY** for an arbitrary $t > 0$. Let $I = (r, k_1, \dots, k_r, s_1, \dots, s_r, X)$ be an instance of **PARAMETERIZED QBFSAT** $_{2t}$, and assume that r is odd and the leading quantifier is existential. The reduction uses ideas from Schaefer's polynomial-time reduction from **QBF** to **GENERALIZED GEOGRAPHY** [409]. The graph on which the geography game is played has three parts: the choice component, the formula testing component, and the literal testing component.

The choice component is similar to Schaefer's and is designed so that player 1 chooses a member of s_1 , then player 2 chooses a member of s_2 , then player 1 chooses a member of s_3 , etc. The choice testing gadget is given in Figure 14.1 for successive quantifier pairs $Q_{t_i} Q_{t_{i+1}}$. The gadget for Q_{t_i} which asks us to pick *one* member from s_i , consists of vertices v_i, w_i , and x_j for each $x_j \in s_i$. The edges are $v_i x_j$ and $x_j w_i$ for each j . A total of $3r$ moves are made through this component, and we add two additional edges to ensure that it is player 2's move at the end, where the game enters the formula-testing component.

In the formula-testing component, we use player 1's moves to simulate disjunctions and player 2's moves to simulate conjunctions. A total of $2t$ moves are made through this component. Let y be bottom vertex of the choice component. Let C be a circuit representing X . Reversing the arrows, we create a tree representing C with root y . We can assume that the circuit is $2t$ layers, beginning with layers of, alternatively, disjunctions and then conjunctions via the Normalization Theorem. This is why when the arrows are reversed, since y will be representing the output of a conjunction and it will be player 2's turn, player 2 will always be playing a conjunction and player 1 a disjunction. Play ends at a *literal vertex* v , corresponding to a literal in formula X , with the move being player 2's.

A literal vertex v corresponding to a positive literal x has an edge to the vertex v_x in the choice component t_i that corresponds to x . If v_x was chosen (variable x is true), then player 2 has no move, and player 1 wins. If v_x was not chosen (so x is false), then player 2 moves to v_x and wins.

If literal vertex v corresponds to a negative literal $\bar{x}$, then the literal testing component has edges (v, u_x) and (u_x, v_x) , where u_x is a new vertex and v_x is the vertex corresponding to x in the choice component. Vertex u_x switches the initiative and causes player 1 to win if v_x was not chosen, and player 2 to win if v_x was chosen.

In Figure 14.1, we have given an example of this construction for the formula

$$\exists x \in \{x_1, x_2, x_3\} \forall y \in \{y_1, y_2, y_3, y_4\} [(x_1 \vee y_1) \wedge (\bar{x}_1 \vee \bar{y}_2 \vee \bar{y}_3)].$$

It is easy to see that the parity of the choices allow player 1 to win iff the formula is true, since all plays begin at v_1 . The total number of moves is at most $3r + 2t + 4$. Since t is fixed, the conditions of an fp -reduction are met.

Remark. Note that the above says that there is a trade-off between quantification and weft. It would be interesting to understand exactly why this occurs. Of course, *weft* and *quantifier complexity* are both measures of logical alternation, so perhaps this phenomenon is not too surprising. We will improve upon this result to establish the following in Chapter 16.

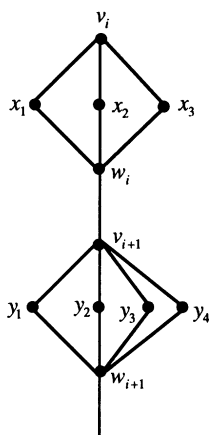
Theorem 14.10 (The Trade-off Theorem, Downey, Fellows, and Regan [181])
 $AW[*] = AW[1]$.

The theorem above suggests that $AW[*]$ is perhaps the natural home of k -move games. Other examples of this phenomenon are given in the exercises. We remark that we do not know if the analog of $PSPACE$ having $QBFSAT$ as a complete problem holds (i.e., $COMPACT TM COMPUTATION \equiv_m^s AW[P]$). Parameterized space seems a very interesting arena of investigation. For instance, as we mentioned earlier, we do not know if a parameterized version of Savitch's theorem holds.

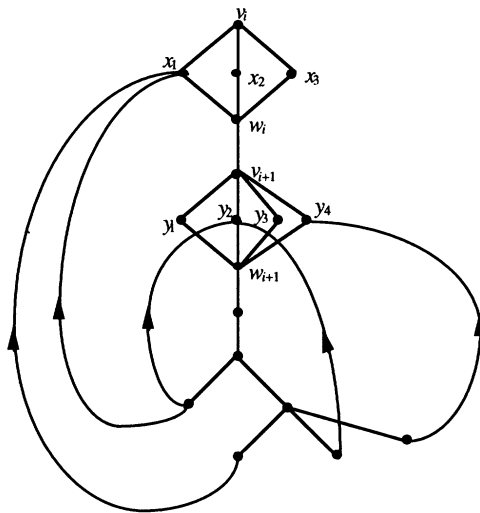
Obviously, there are a number of other k -move games that are natural candidates for $AW[*]$ -completeness. We mention two. The first is **ALTERNATING HITTING SET** with no vertex degree restrictions. The other is

SHORT GENERALIZED HEX (Even and Tarjan [200])

- Instance:* A graph G with two distinguished vertices v_1 and v_2 .
Parameter: A positive integer k .
Question: Does player I have a winning strategy in the following game. Player I plays with white pebbles and player II with black ones. Pebbles are to be placed on nondistinguished vertices alternately by Player I and then Player II. Player I wins if he can construct a path of white vertices from v_1 to v_2 , in $\leq k$ moves.



Choice components for Q_t, Q_{t+1} for $t_i = t_{i+1} = 1$, $s_i = \{x_1, x_2, x_3\}$, $s_{i+1} = \{y_1, y_2, y_3, y_4\}$.



Example for $\exists x \in \{x_1, x_2, x_3\} \forall y \in \{y_1, y_2, y_3, y_4\} ((x_1 \vee y_2) \wedge (\bar{x}_1 \vee \bar{y}_1, \vee \bar{y}_3))$.

[All edges are directed down unless otherwise indicated.]

FIGURE 14.1. Gadgets for **SHORT GEOGRAPHY**

Exercises 14.0

14.0.1. (Abrahamson, Downey, and Fellows [3][4]) Prove that **RESTRICTED ALTERNATING HITTING SET** is strongly *FPT* (Theorem 14.2). [Hint: It is simplest to consider $k_1 = 2$, the analog of the *PSPACE* complete problem **ALTERNATING VERTEX COVER**. Take an edge (x, y) . All vertex covers must include x or y . Try each, generating the tree of possibilities. Terminate a branch and put the cover at the leaf if a branch achieves a vertex cover. This gives a tree with at most $k_1^{k_2} = 2^{k_2}$ leaves (corresponding to possible candidates for vertex covers), at most $k_2 2^{k_2}$ vertices, and all size $\leq k_2$ covers must contain a subset occurring at one of the leaves.

Now, we select k_2 additional vertices of G , not occurring at any of the leaves of the tree (we can assume V is large compared to k_2 , else the problem is easily done). Consider all possible strategies played on the subgraph induced by these at most $k_2 + k_2 2^{k_2}$ vertices. It is easy to see that player I has a winning strategy in $\leq k_2$ moves in G iff he has one in this set of strategies.]

14.0.2. (Abrahamson, Downey, and Fellows [4]) Prove that the following problem is $AW[*]$ -complete.

SHORT NODE KAYLES

Instance: A graph G .

Parameter: k

Question: Does player I have a winning k -move strategy in the following game? Players pebble a vertex not adjacent to any pebbled vertex. The first player with no play loses. Player I plays first.

Remark. We have stuck with the terminology of Schaefer, although the reader should think of the above as k -move **ALTERNATING DOMINATING SET**.

[Hint: First, we show that the problem is $AW[*]$ -hard. In view of Theorem 14.9, we only need show that the problem is $AW[2]$ -hard. Let $\varphi = \exists x_k \in S_k \forall x_{k-1} \in S_{k-1} \dots \exists x_1 \in S_1 (C_1 \wedge \dots \wedge C_m)$ be an instance of unitary $AW[2]$. Here, we shall assume that k is odd, and $S_i = \{x_{i,1}, \dots, x_{i,n_i}\}$ with $S_1 = \{x_{1,1}, x_{1,2}\}$ and $B_1 = x_{1,2} \vee x_{1,2} \vee \overline{x_{1,1}} \vee \overline{x} - 1, 2$. We need the vertex sets:

$$V = \cup_i S_i,$$

$$S_0 = \{z_{0,q} : 1 \leq q \leq m\},$$

$$Y_i = \{y_{i,j} : 0 \leq j \leq i-1\} \text{ for } 1 \leq i \leq k.$$

Now we need the edge sets:

$$D = \{xz_{0,q} : x \text{ occurs in clause } C_q\},$$

$$F = \{xz_{0,q} : y \neq x, x, y \in S_i \text{ and } \overline{y} \text{ occurs in clause } C_q\},$$

$$G = \{y_{i,j}w : w \in ((\cup_{0 \leq p < i, p \neq j} S_p) \cup (\cup_{1 \leq r < i, r \neq j} Y_r))\}.$$

Following, ideas of Schaefer, we say that the game is played *legitimately* if the node played at move i is an element of S_{k-i+1} . We claim that if at move i a player does not play legitimately, then the other player wins at the next move. Suppose that the first $k-i$ moves have been legitimate. If the player then plays illegitimately, note that he cannot have played any node from $\cup_{j \leq i+1} (S_{k-j} \cup Y_{k-j})$, as these are already dominated by pebbled vertices. Now,

if he plays a vertex in $S_j \cup Y_j$ for some $j < i$, then his opponent can win by playing $y_{i,j}$. (Every vertex in $S_j \cup Y_j$ is adjacent to the illegal vertex; all the rest are adjacent to either a previously played vertex or $y_{i,j}$.) If the illegitimate play is in Y_i , the only remaining possibility, it must be a $y_{i,j}$, and then the opponent can win by playing a member of S_j if $j > 0$, and either $z_{0,1}$ or $z_{0,2}$ if $j = 0$. This enforcement gadgetry clearly now ensures that player I has a winning strategy of k moves iff φ is true, as the reader can readily check. To complete the proof, we need to establish membership of $AW[*]$. But this is essentially the same as Theorem 14.6.]

15

Provable Intractability: The Class XP

We have seen many classes that are apparently different from FPT , but as with classical complexity theory, we have no proof of this fact. Moving further along, we might seek classes that are *provably* distinct from FPT , along the lines that $EXP \neq P$. We will briefly look at one such class in this section.

In Chapter 5, we met the notion of the class $SLICEWISE P$. Indeed, recall that for any complexity class C , we defined the class of $SLICEWISE C$ as those languages L accepted in time (space) $f(k)g(|x|)$ for a arbitrary function f and a P -time function g . As with the other definitions, we appended the adjective “uniform” if, additionally, f is recursive. Under this definition, recall that FPT is $SLICEWISE P$. We generalize this definition as follows.

Definition 15.1 (X Classes) Let C be a (classical) complexity class. We say that a parameterized language L is in XC iff $L_k \in C$ for all k . If the index for membership of C can be found uniformly as a computable function of k , then we say that $L \in \text{uniform } XC$.

For instance, the class XP consists in those parameterized languages L such that $L_k \in P$ for each k . L would be in uniform XP if there are computable functions f and g such that for each k , $\Phi_{f(k)}$ accepts L_k and has running time $|x|^{g(k)}$ upon input $\langle x, k \rangle$.

For the purposes of clarity of this chapter, we will only look at XP and remark that all the results hold with no change for uniform XP as well. By standard diagonalization, we have the following.

Observation 15.2 $FPT \subset XP$.

There are a number of problems that are complete for XP , which seems to be the parameterized class corresponding to EXP . We briefly look at some such results drawn from the literature.

Definition 15.3 (Pebble Game) A *pebble game* is a quadruple (N, R, S, T) consisting of

- (i) N , a finite set of vertices (or nodes).
- (ii) $R \subseteq \{(x, y, z) : x, y, z \in N, x \neq y \neq z\}$, called the set of *rules*.
- (iii) $S \subset N$, the start set. $|S|$ is called the *rank* of the game.
- (iv) $T \in N$, called the terminal node of the game.

The pebble game is played by two players I and II, who alternatively move pebbles on the nodes. At the beginning of the game, pebbles are placed on all the start nodes. If $(x, y, z) \in R$ and there are pebbles upon x and y but not on z , then a Player can move a pebble from x to z . The winner is the first Player to put a pebble on T or can force the other Player into a position where he cannot move.

PEBBLE GAME PROBLEM

- Input:* A pebble game (N, R, S, T) .
Parameter: A positive integer k .
Question: If $|S| = k$ determine if Player I has a winning strategy.

We reinterpret an old result of Adachi et al. [10, 11] to establish the following.

Theorem 15.4 ([10, 11]) *The PEBBLE GAME PROBLEM is XP -complete.*

In the proof, we begin with proving hardness. Actually, Adachi et al. establish the following sharper characterization for the single-tape Turing machine model.

Theorem 15.5 ([10, 11]) *Let $L \in \mathbf{DTIME}(n^k)$. Then L is $O(n \log n)$ -reducible using space $O(n \log n)$ to the $(2k + 1)$ -pebble game problem.*

Proof. We follow Adachi et al. [10, 11]. The proof uses a generic reduction. Since $L \in \mathbf{DTIME}(n^k)$, there is a single-tape Turing machine M which accepts L within n^k time. We will assume that M has a set of states Q , a set of symbols Γ with initial symbols $\Sigma \subseteq \Gamma$, a blank symbol b , initial state q_0 and final state q_f , and a transition function $\delta : Q \times \Gamma \mapsto Q \times \Gamma \times \{-1, 0, 1\}$. Suppose that $\delta(q, X) = (\hat{q}, \hat{X}, j)$. This would have the interpretation that the machine, on reading some symbol X in state q , will either move left ($j = -1$), right ($j = 1$), or remain where it is ($j = 0$), move to some state $\hat{q}$, and (over-)print the symbol X by symbol $\hat{X}$. Without loss of generality, we make some assumptions on M .

We use the symbol $\#$ as a left endmarker. A *configuration* shown is, as usual, represented by

$$\#X_1X_2 \cdots X_{i-1}(qX_i)X_{i+1} \cdots X_{n^k},$$

where $X_j \in \Gamma$ for all j ($1 \leq j \leq n^k$). We treat an element of $Q \times \Gamma$ as an abstract symbol. Hence, (qX_i) is the i -th symbol ($\#$ is the 0-th symbol) of the configuration. We also assume that the head never scans the endmarker. The initial configuration

for an input $w = w_1 w_2 \cdots w_n$ is $\#(q_0 w_1) w_2 \cdots w_n b^{n^k - n}$. Whenever M enters the final state q_f , the configuration of M should be of the form

$$\#(q_f b) X_2 X_3 \cdots X_{n^k},$$

where $X_j \in \Gamma$ for all i ($2 \leq i \leq n^k$). Once M enters q_f , M stays in the state q_f without changing the position of the tape head or the contents of the tape; that is, if M accepts w within time n^k , then M is in q_f at time n^k .

We construct the $(2k + 1)$ -pebble game G from M and its input w such that (1) M accepts w within time $|w|^k$ if and only if the first Player has a forced win in the $(2k + 1)$ -pebble game and (2) the construction is performed within time $O(n \log n)$.

Throughout this proof, we use two global variables t and i for explanation, where $0 \leq t \leq n^k$ and $1 \leq i \leq n^k$. The variable t represents the current time of the Turing machine and i represents the position of the symbol of the configuration. We prepare nodes in the pebble game of the form $[qX]$, $[X]$ for every state $q \in Q$ and every tape symbol $X \in \Gamma$ of the Turing machine. We also use nodes of the form $[X_{-1}, X_0, X_1]$, where $X_{-1}, X_0, X_1 \in (Q \times \Gamma) \cup \Gamma \cup \{\#\}$. Let $F(X_{-1}, X_0, X_1)$ be the symbol in $(Q \times \Gamma) \cup \Gamma$ such that if $\alpha X_{-1} X_0 X_1 \beta$ is a configuration of M for some α, β , then one move of M changes the symbol X_0 to $F(X_{-1}, X_0, X_1)$. When $X_{-1} X_0 X_1$ is not part of any configuration, then $F(X_{-1}, X_0, X_1)$ is not defined. We note that $F(X_{-1}, X_0, X_1)$ is uniquely defined and is independent of t and i since M is deterministic.

The game uses $(2k + 1)$ pebbles. One pebble is placed to represent a symbol of a configuration; k pebbles are used for **TCOUNTER**, a counter for t ; and another k pebbles are for **ICOUNTER**, a counter for i . Each value is represented by k , n -ary counters, and the j -th pebble is placed to represent the j -th n -ary digit for $1 \leq j \leq k$.

The pebble game will simulate M backward. At the beginning of the pebble game, a pebble is placed on $[q_f b]$ and $2k$ pebbles are placed in order to represent $t = n^k$ and $i = 1$. This corresponds to $q_f b$ being the first symbol of the configuration at time n^k . The rules of the pebble game are constructed in such a way that moves of pebbles are restricted as follows:

- (1) Only pebble moves shown in Figure 15.1 are allowed; that is, both players have to be forced to move pebbles along the edges in Figure 15.1.
- (2) When a pebble is on $[X]$, the first Player can move the pebble to a node of the form $[X_{-1}, X_0, X_1]$ such that $X = F(X_{-1}, X_0, X_1)$, and when a pebble is on $[X_{-1}, X_0, X_1]$, the second Player can choose d from $\{-1, 0, 1\}$ and can move the pebble to $[X_d]$.
- (3) Before and after the renewals of **TCOUNTER** and **ICOUNTER**, k pebbles are used to represent the current value of t and another k pebbles are used for i . During these renewal procedures, each Player is forced to cooperate with another Player to update **TCOUNTER** and **ICOUNTER**.

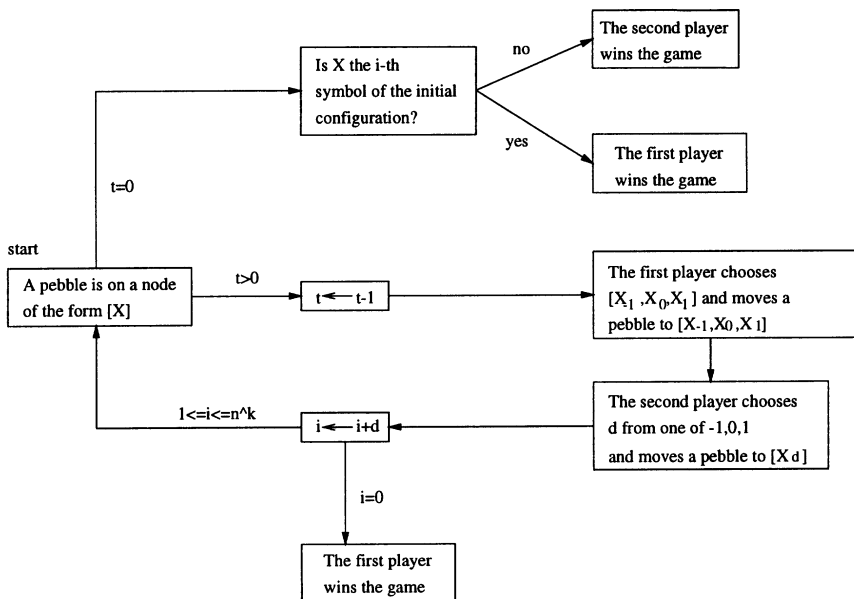


FIGURE 15.1. Pebble Moves Where $X = F(X_{-1}, X_0, X_1)$, $X \in (Q \times \Gamma) \cup \Gamma$, $d \in \{-1, 0, 1\}$. Initially a Pebble Is on $[q_f b]$ and $t = n^k$, $i = 1$

Typical moves in the game are (1) at time $t > 0$ to iterate those pebble moves that are described in the boxes along the cycle in Figure 15.1 and (2) at time $t = 0$ to check whether X is the i -th symbol of the initial configuration to determine which Player wins the game. The alternation inherent in the definition of the pebble game allows us to get by without explicitly storing the contents of the tape inside the game. If the first Player ever tries to cheat by modifying the initial contents of a tape square illegally, the second Player can catch him by forcing a win by staying on this square.

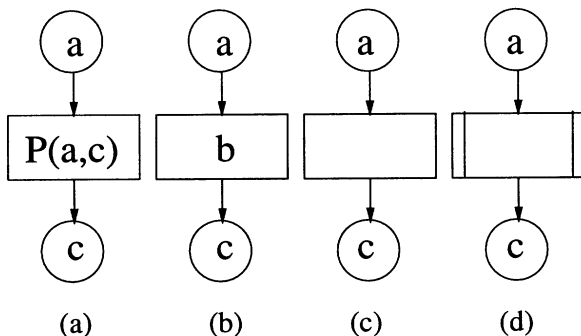


FIGURE 15.2. A Set of Rules, Where a , b , and c Are Nodes and $P(a, c)$ Is a Condition

Now we define the rules of the pebble game. For our convenience, let Figure 15.2(a) denote the set of rules

$$\{(a, b, c) \mid P(a, c), b \text{ is a node in } \mathbf{ICOUNTER}\}$$

where a , b , and c are nodes and $P(a, c)$ is a condition on a and c . Thus, if a pebble is on a , no pebble is on c , and $P(a, c)$ holds, then a Player can move the pebble from a to c in his turn since there is some pebble on nodes in $\mathbf{ICOUNTER}$. Let Figure 15.2(b) denote the rule (a, b, c) , and let Figure 15.2(c) denote the set of rules

$$\{(a, b, c) \mid b \text{ is in the set of the nodes of the game}\}$$

where the box in the figure is left blank. We let Figure 15.2(d) denote a predefined set of rules. We use two predefined set of rules, $\mathbf{TCOUNTER}$ and $\mathbf{ICOUNTER}$.

The rules of the pebble game are shown in Figures 15.3 and 15.4. Consider Figure 15.3. The first Player moves a pebble from $[X]$ to $[w1, X]$. For every element $[X]$ in $(Q \times \Gamma) \cup \Gamma$, $\mathbf{TCOUNTER}$ uses nodes of the form $[w1, X]$ and $[w2, X]$. Once a pebble is on $[w1, X]$, two players of the game are forced to cooperate for decreasing a value of t by one until the second Player puts a pebble on $[w2, X]$ to end the procedure for $\mathbf{TCOUNTER}$. The value of t is represented by the content of $\mathbf{TCOUNTER}$. When $t = 0$, one of the k pebbles of $\mathbf{TCOUNTER}$ is placed on a distinguished node [time-zero]. If a pebble is already on [time-zero] and if the first Player moves a pebble to $[w1, X]$, then the second Player wins the game.

When a pebble is on $[w2, X]$, the first Player moves the pebble to $[X_{-1}, X_0, X_1]$ such that $X = F(X_{-1}, X_0, X_1)$. Then, the second Player selects one of the three nodes [dec, X_{-1}], [X_0], and [inc, X_1], and moves the pebble from $[X_{-1}, X_0, X_1]$ to the selected node. If a pebble is on a node of the form [dec, Y] or [inc, Y], the two players are forced to operate with $\mathbf{ICOUNTER}$. The contents of $\mathbf{ICOUNTER}$ represent the value of i , that is, the symbol position of a configuration. If a pebble is on [dec, Y], then the contents of $\mathbf{ICOUNTER}$ are decreased by one, and if the resulting value is zero, then one of the k pebbles of $\mathbf{ICOUNTER}$ is moved to a distinguished node [leftend]. If a pebble is on [inc, Y], then the contents of $\mathbf{ICOUNTER}$ are increased by one. We do not have to check the right end of the tape, since the value of $\mathbf{TCOUNTER}$ would be 0 before the contents of $\mathbf{ICOUNTER}$ will be $n^k + 1$. The first Player moves a pebble to $[w3, X_1]$ or $[w3, X_{-1}]$ to end the procedure for $\mathbf{ICOUNTER}$. If a pebble is on [leftend], then the first Player wins.

Figure 15.4 is a set of rules when $t = 0$. Note that a pebble is on [time-zero] when and only when $t = 0$. Thus, the first Player can move a pebble to $[w4, X]$ only when $t = 0$. Assume that the first Player places a pebble on $[w4, X]$. If X is the i -th symbol of the initial configuration, then the first Player wins the game; otherwise the second Player wins, where i is the contents of $\mathbf{ICOUNTER}$.

Detailed construction of $\mathbf{TCOUNTER}$, $\mathbf{ICOUNTER}$, and Figure 15.4 is straightforward and left to the reader. We shall prove the correctness of the construction; that is, M accepts w if and only if the first Player has a forced win in pebble game G . We call a position of pebble game G a *disposition* $D(X, t, i)$ of G if

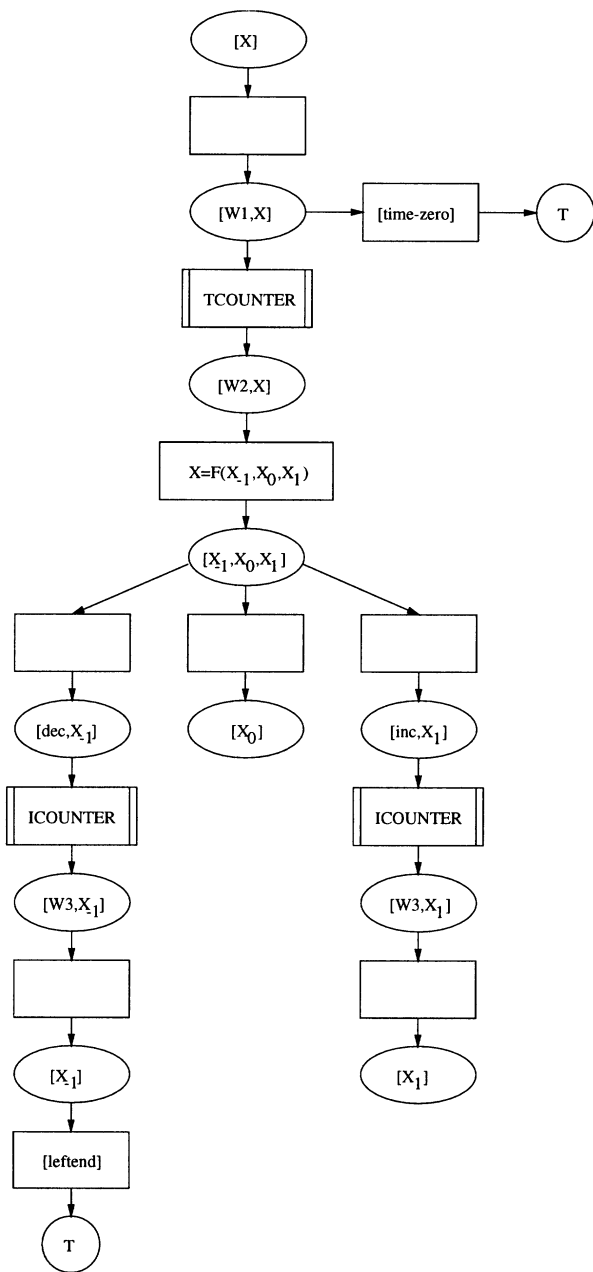


FIGURE 15.3. Rules for the Pebble Game

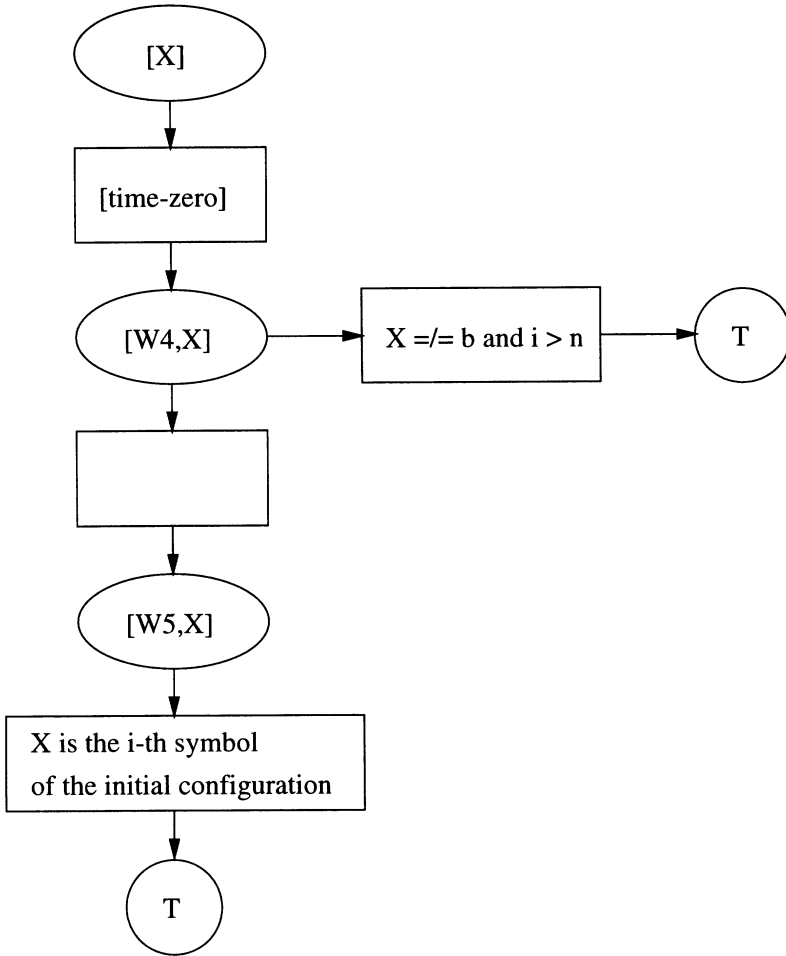


FIGURE 15.4. More Rules for the Pebble Game

- (i) A pebble is on $[X]$, $X \in (Q \times \Gamma) \cup \Gamma$.
- (ii) The contents of **TCOUNTER** is t .
- (iii) The contents of **ICOUNTER** is i .

Note that k pebbles are used for **TCOUNTER**, and another k pebbles for **ICOUNTER**.

We now show that X is the i -th symbol of the configuration at time t of M if and only if the first Player has a forced win from $D(X, t, i)$.

We prove this by induction on t .

Basis. When $t = 0$, it is obvious that the first Player has a forced win if and only if X is the i -th symbol of the initial configuration. This is illustrated in Figures 15.1 and 15.4.

Induction Step. Assume that the induction hypothesis holds at time $t - 1$. We consider the case at time t ($1 \leq t \leq n^k$).

Assume that the first Player has a forced win from the disposition $D(X, t, i)$ of the game. Thus, there is a way to win the game for the first Player even if the second Player may select any node to move a pebble. After the first Player moves a pebble from $[w2, X]$ to $[X_{-1}, X_0, X_1]$, where $X = F(X_{-1}, X_0, X_1)$, the first Player then has a forced win from $D(X_d, t - 1, i + d)$ for all $d, d \in \{-1, 0, 1\}$. By the induction hypothesis, X_d is the $(i + d)$ -th symbol of the configuration at time $t - 1$, for each $d, d \in \{-1, 0, 1\}$. Thus, X is the i -th symbol of the configuration at time t , since $X = F(X_{-1}, X_0, X_1)$.

Conversely, assume that X is the symbol of the configuration at time t ($1 \leq t \leq n^k$) and position i ($1 \leq t \leq n^k$). Then, there are symbols X_{-1}, X_0 , and X_1 such that $X = F(X_{-1}, X_0, X_1)$, and these symbols occupy positions $i - 1, i, i + 1$ of the configuration at time $t - 1$, respectively. By the induction hypothesis, the first Player has a forced win for $D(X_d, t - 1, i + d)$ of the game for all d ($d \in \{-1, 0, 1\}$). Thus, the first Player has a forced win from $D(X, t, i)$. Note that there is nondeterminism here, since the discussion is made for all d ($d \in \{-1, 0, 1\}$).

This completes the proof of the induction.

We note that at the beginning of the game, pebbles are placed as $D(q_f b, n^k, 1)$, and that the symbol at the first position of the configuration at time n^k is $(q_f b)$ if M accepts w within time n^k . Therefore, we have shown that M accepts w within time n^k if and only if the first Player has a forced win in the pebble game G .

Note that the number of the nodes and the rules of G is $O(n)$, and that the construction of G is performed within $O(n \log n)$ time.

To complete the proof of Theorem 15.4, we need membership in XP .

Lemma 15.6 *The PEBBLE GAME PROBLEM is in XP .*

Proof. To prove Lemma 15.6, we need to show that the k -pebble game is in P . Note that since the allowed moves preserve the fact that there are k pebbles placed on the graph, there are at most $\binom{n}{k}$ many configurations. We build a graph G , whose vertices are all possible ways that k pebbles can be placed on the set of nodes of a pebble game. We make a directed edge from configuration u to configuration v whenever there is an allowed move that would transform u in one step to v . There is a single vertex S that represents the starting configuration, and there are $\binom{n-1}{k-1}$ many configurations that would represent a win (that is, with the one remaining pebble on the terminal node T).

The lemma immediately follows from the following general lemma about a game played with one marker. We need a definition. Define the **BASIC ONE-PEBBLE GAME** to be a directed graph $D = (V, A)$ with a single start vertex S and a set of final vertices F contained in V . Play begins with a single marker on S . A move consists of moving the marker along an arc of D . The first Player to move the marker to a vertex of F wins.

Lemma 15.7 *It can be determined in time $O(|D|)$ whether the first Player has a winning strategy for the BASIC ONE-PEBBLE GAME.*

Proof. Let D' be the digraph obtained from D in the following way:

- (1) Compute the set of winning configurations W (for Player 1)—these are the vertices w for which there is an arc from w to a vertex of F . Thus, if the marker is on w and it is Player 1's turn, then Player 1 can play and win. We assume that there are no out-arcs from vertices in F , since they represent "game over."
- (2) Compute the set of vertices F' defined to be the set of all vertices x such that every arc xy leads to vertex y in W .

Then, we define D' to be the directed graph with vertex set $V - F - W$, with s inherited from D , with arcs inherited from D (except delete arcs between F' vertices), and with the set F' of final vertices.

Equivalently, D' is obtained from D by deleting F and W , i.e., deleting arcs between F' vertices. And s is still there if it has not been removed.

Note that Player 1 has a winning strategy for D iff Player 1 has a winning strategy for D' . (One has a strategy to win the **ONE-PEBBLE GAME** iff one has a strategy to get the marker moved to a vertex from which every out-arc—your possible moves—leads to a vertex from which one can win.)

The algorithm is just to compute the sequence of **BASIC ONE-PEBBLE GAMES** $D, D', D'' \dots$

Corollary 15.8 ([11]) (i) *The $(2k + 1)$ -pebble game problem requires $\Omega(n^{k-\epsilon})$ time on STM for any $\epsilon > 0$ and $k > 2$.*

(ii) *Hence, for a fixed k , the **PEBBLE GAME PROBLEM** of rank k requires time $\Omega(n^{\frac{k-1}{4-\epsilon}})$ for any $\epsilon > 0$ and $k > 5$ and is hard for $DTIME(n^k)$.*

Proof. Exercise 1.

There are several other XP completeness results which can also be extracted from Refs. [10] and [11]. They are proved by the construction of combinatorial reductions from the **PEBBLE GAME PROBLEM**. We will merely state them here.

The **CAT AND MOUSE GAME** is a quintuple $G = (X, E, c, M, v)$ with X a set of vertices, E a set of edges, $c \in X$, $M \subseteq X$, and $v \in X$. In the game, Player I begins with his token on c and Player II begins with tokens on each member of M . Players play alternatively and can move tokens from vertex x to vertex y provided $xy \in E$. Two tokens of Player II cannot occupy the same vertex. Player I wins if he can place his token on a vertex with one of Player II's tokens. Player II wins if she can place one of her tokens on vertex v even if it is occupied by Player I's token. Player I plays first.

CAT AND MOUSE PROBLEM

Input: A **CAT AND MOUSE GAME** $G = (X, E, c, M, v)$.

Parameter: A positive integer k .

Question: If $|M| = k$, does Player I have a winning strategy?

Let Q be a set of integers. The **PEG GAME** is a triple $G = (V, k, l)$ with $k, l \in Q$ and $V \subseteq V^l$ such that $(v_1, \dots, v_l) \subseteq V$ implies that $v_1 + \dots + v_l = 0$. A play of the **PEG GAME** is as follows. There are l pegs on the board and k rings. The interpretation of the vector $(v_1, \dots, v_l) \in V$ is that for each i , if $v_i \geq 0$, then we

put v_i rings on the i -th peg, and if $v_i \leq 0$, we remove v_i rings from the i -th peg. Initially, all k rings are on the first peg. Players play alternatively according to the rules V of the game. The Player who wins is the one who places all k rings on the last peg, or who forces the opponent into a position where the opponent has no valid play.

PEG GAME PROBLEM

Input: A PEG GAME $G = (V, k, l)$.
Parameter: A positive integer k .
Question: Does Player I have a winning strategy?

Historical Remarks

The diagonalization method produces languages L in $\text{DTIME}(n^{k+1})$ - $\text{DTIME}(n^k)$, and shows that exponential time differs from P . This fact was realized very early. The results of this section are drawn from Refs. [10] and [11], which gave the first examples of natural combinatorial problems separating the polynomial-time classes. We remark that, in some sense, their results fit more easily into our setting since XP -completeness seems to be machine independent, whereas the class $\text{DTIME}(n^k)$ is not.

Exercises 15.0

15.0.1. Prove Corollary 15.8.

[Hint: By diagonalization, there exists a language L in $\text{DTIME}(n^k)$ which is not in $\text{DTIME}(n^{k-\epsilon})$ for any $\epsilon > 0$. By Theorem 15.5, L is $O(n \log n)$ -reducible to the $(2k+1)$ -pebble game problem. Suppose that the $(2k+1)$ -pebble game problem is $n^{k-\epsilon_1}$ time computable on an STM for some $\epsilon_1 > 0$. If $n^2 \log^2 n \ll n^{k-\epsilon_1} \log^{k-\epsilon_1} n$, then establish that that L is $O(n^{k-\epsilon_1} \log^{k-\epsilon_1} n)$ time computable on a Turing machine. Otherwise L is $O(n^2 \log^2 n)$ time computable on a Turing machine.]

15.0.2. (Chandra and Stockmeyer [436]) Prove that the CAT AND MOUSE GAME PROBLEM is in XP .

(Hint: Use the method of Lemma 15.6.)

15.0.3. Prove that the PEG GAME PROBLEM is in XP .

(Hint: Use the method of Lemma 15.6.)

Part III

Structural and Other Results

16

Another Basis for the W -Hierarchy, the Tradeoff-Theorem, and Randomized Reductions

The fundamental separation hypothesis upon which our theory of parameterized intractability is based is

$$W[1] \neq FPT.$$

A weaker hypothesis might be that for some t , $W[t] \neq FPT$ or even that, say, $W[P] \neq FPT$. A more subtle hypothesis is that the W -hierarchy is proper, and, in particular, $W[1] \neq W[2]$. In the present chapter, we will look at some evidence as to why one or all of these hypotheses might seem reasonable. We begin by concentrating upon the $W[1]$ vs. $W[2]$ question. The quintessential complete problems here are **CLIQUE** for $W[1]$ and **DOMINATING SET** for $W[2]$.

For each k , the language of graphs with a clique of size k is defined by the existential formula

$$\varphi_k := (\exists u_1 \dots u_k) : \wedge_{i,j \leq k} E(u_i, u_j),$$

where $E(\cdot, \cdot)$ formalizes the adjacency relation for graphs. By contrast, the language of graphs with a dominating set of size k is known to require two blocks of like quantifiers to define in first-order logic, such as by the Σ_2 formula

$$\psi_k := (\exists u_1 \dots u_k) : (\forall v) \vee_{i \leq k} (v = u_i \vee E(v, u_i)).$$

Both problems are about searching for a set of vertices of size k that satisfy the condition following the “:”, but in ψ_k , the condition is more complex because it has the extra quantifier over vertices v . Put another way, once candidate vertices have been assigned to $u_1, \dots, u_k$, the condition for **CLIQUE** is entirely “local” in a

sense studied for parameterized languages in [370], whereas that for **DOMINATING SET** requires a “global” reference to other parts of the graph. Some parameterized problems on graphs have conditions that make several alternating first-order quantifications over the graph, and are known to belong to $W[t]$ only for higher values of t . As we have seen in Part II, other problems have conditions that are not first-order definable at all, and some of these are complete for $W[P]$ (see [76]). Intuitively, the question “Does $W[1] = W[2]$?” asks whether a local check of a fixed-size substructure can do the same work as a global check. The question “For all t , does $W[t] = W[2]$?” asks whether the simple check over vertices v in the $W[2]$ -complete **DOMINATING SET** problem suffices to verify any condition that is definable by circuits of bounded weft t . Similarly, if $W[P] = W[2]$, then fixed-parameter many-one reductions have an enormous power to simplify the checking of properties. We must be careful not to push this reasoning too far. For instance the “ k slices” of **VERTEX COVER** have logical definitions of form similar to that of ψ_k and yet are fixed-parameter tractable. Part of the difficulty is that the “logical complexity” of the *reductions* can be more complex than the logical depth of the circuits used in the definition. More precisely, compare the W situation with the classical NP situation. If $L \leq_m^P Q$ and $Q \in NP$, then L can be put in the NP logical form: for some P -time relation R ,

$$x \in L \text{ iff } \exists y(R(x, y)).$$

However if $L' \leq_m^s Q'$ and $Q' \in W[t]$, we have no guarantee that we can actually *express* L' as the weight k' solutions to some collection of weft t circuits. Indeed, we can show that in general this is not true.

The goal of the present chapter is to tie the classes in the W -Hierarchy to classes of (parameterized) languages that are much *smaller* than P , and where complexity-based hierarchies are *known* to be proper. Our results bring out the *computational nature* of the W -Hierarchy and provide some evidence for a positive answer to the question, “Are all classes in the W -Hierarchy distinct?” One way we do this is to prove that the basic defining weft classes (i.e., without the reductions) do, in fact, generate provably distinct classes. We also have a second agenda in that we wish to demonstrate that the notion of weft is, itself, of independent interest even to classical complexity theorists.

As an overview, each class $W[t]$ is shown to be definable via existential quantification on the class of parameterized languages recognizable by polynomial-sized circuits of constant weft t , analogous to the way NP is defined by existential quantification on P . The circuits we obtain are actually AC^0 circuits of depth t except for extra layers of gates of fanin 2, and providing also for parameterization, we call them $G[t]$ circuits. In symbols, we have $W[t] = \langle N[t] \rangle$, where $N[t] = \exists \cdot G[t]$. $\langle F \rangle$ denotes the closure of the class F under parametric reductions. Also, $W[P] = N[P] =_{\text{def}} \exists \cdot G[P]$, without the closure notation. Then, we show that not only is the $G[t]$ -hierarchy proper, but, more importantly, the $N[t]$ -hierarchy is proper. Thus, among the three “elements” of the W -hierarchy, namely parameterized languages, circuit weft, and f.p. reductions, only the last can be responsible for any collapse. We explain how these results rule out any “nor-

mal" argument for collapse of the W -hierarchy and give this as evidence that the hierarchy does not collapse. One other result for this chapter is to further explore the analogy with the polynomial-time hierarchy and to see the effect of adding further quantifiers to the G -hierarchy. Thus, we define for each t , a hierarchy $H[t]$ by alternating $\forall$ - and $\exists$ -quantification over $G[t]$. In the notation of Chapter 14, we will have that $AW[t] = \langle H[t] \rangle$. Improving Theorem 14.9, we will prove the full Trade-off Theorem, $H[t] = H[1]$.

Definition 16.1 (Parametric Connection) A *parametric connection* is a function $\alpha : (\mathbb{N} \times \mathbb{N}) \rightarrow (\mathbb{N} \times \mathbb{N}) : (n, k) \mapsto (n', k')$, a polynomial q , and arbitrary functions $f, g : \mathbb{N} \rightarrow \mathbb{N}$ with $n' = f(k)q(n)$ and $k' = g(k)$. A parametric connection is *nice* if it can be computed in time n'' described by another parametric connection.

To economize on notation, we write $n, k, n', k', n'', k'', \dots$ to indicate that the first four quantities represent one parametric connection, the third through sixth another, and so on. The connection relation is transitive. This notion enables us to define circuit complexity directly for parameterized problems:

Definition 16.2 (Parameterized Family of Circuits) A *parameterized family* of circuits is a bi-indexed family of circuits $\mathcal{F} = \{C_{n,k} : n, k \in \mathbb{N}\}$ such that each $C_{n,k}$ has n inputs and size at most n' , where n' is part of a connection with n, k . We say that such a family is *FPT-uniform* if there is an algorithm to produce the circuit $C_{n,k}$ in time $O(n')$.

Definition 16.3 $G[t]$ (Uniform $G[t]$) is the class of parameterized languages $L \subseteq \Sigma^* \times \mathbb{N}$ for which there is a parameterized (uniform) family of weft t circuits $\mathcal{F} = \{C_{n,k}\}$ such that for all x and k , with $n = |x|$, $\langle x, k \rangle \in L \iff C_{n,k}(x) = 1$. If there is no restriction on the circuit weft, then we obtain the class of parameterized languages $G[P]$, and if the circuits are boolean, then we get $G[SAT]$.

It is not difficult to see that $\text{Uniform } G[P] = \text{FPT}$ (Exercise 1). Thus, the classes $G[t]$ provide a stratification of the problems that are fixed-parameter tractable. Now we can build upon them in much the same way that NP is definable by bounded existential quantification over P . NP uses a polynomial length bound, whereas our classes $N[t]$ use bounds on Hamming weight.

Definition 16.4

- (a) For any class $\mathcal{F}$ of parameterized languages, $\exists \cdot \mathcal{F}$ stands for the class of parameterized languages A such that for some $B \in \mathcal{F}$, there are nice parametric connections (n, k, n', k', n'', k'') , giving for all $\langle x, k \rangle$, $\langle x, k \rangle \in A \iff (\exists y \in \Sigma^{n'}) [\text{weight}(y) = k' \wedge (xy, k'') \in B]$. (Here, $n = |x|$, $n' = |y|$, and $n'' = n + n'$.)
- (b) For all $t \geq 1$, $N[t]$ stands for $\exists \cdot \text{Uniform } G[t]$, and $N[P]$ stands for $\exists \cdot \text{Uniform } G[P]$.

In a corresponding way, we can define “bounded weight” versions of the other familiar class operators $\forall$, $\oplus$, and BP . Combining the latter two formally, we have that a language A belongs to $BP \cdot \oplus \cdot \mathcal{F}$ if there exists $B \in \mathcal{F}$ and nice connections giving for all $\langle x, k \rangle$,

$$\begin{aligned} \langle x, k \rangle \in A \rightarrow \Pr_{y \in \{0,1\}^{n'}} [\text{weight}(y) = k' \wedge (|z \in \{0,1\}^{n''} : \text{weight}(z) \\ = k'' \wedge (xyz, k'') \in B| \text{ is odd})] > 3/4, \end{aligned}$$

whereas $\langle x, k \rangle \notin A \rightarrow \Pr[\dots] < 1/4$. If the latter probability is zero (i.e., we have one-sided error), then we write $A \in RP \cdot \oplus \cdot G[t]$.

The next theorem provides a powerful characterization of the $W[t]$ classes.

Theorem 16.5 (The Replacement Theorem [174]) *For all $t \geq 1$, $W[t] = \langle N[t] \rangle$.*

To see what is interesting about this theorem, consider the special case of $t = 2$ and the $W[2]$ -complete parameterized problem **DOMINATING SET**. The original criterion for showing **DOMINATING SET** to be in $W[2]$ requires constructing, for each graph G and positive integer k , a weft 2 circuit C_G that accepts a weight k input vector iff G has a k -element dominating set. The point is that for each graph G , we construct a *different* circuit, thus perhaps $2^{\binom{n}{k}}$ different circuits for graphs of order n for a fixed value of k . By contrast, to show that **DOMINATING SET** belongs to the FPT -closure of $N[t]$, we must refer all of the graphs of order n to a *single* circuit $C_{n,k}$. The input to $C_{n,k}$ consists of the concatenation xy of a string x representing G and a string y representing the $k \log n$ bits of nondeterminism. For this particular instance, our proof must devise a bi-indexed family of weft 2 circuits, each circuit $C_{n,k}$ of which is “universal” for the dominating set problem for graphs of order n and for the parameter k . These “universal circuits” will resemble programmable logic arrays.

Proof. Assume first that $t \geq 2$ and that t is even. Let L be a parameterized language in $W[t]$. We can assume without loss of generality that the reduction showing membership of L in $W[t]$ maps $\langle x, k \rangle$ to (C_x, k') , where

- (i) C_x is a t -normalized circuit.
- (ii) C_x has n' inputs.

- (iii) C_x has exactly n'' gates on each level other than the input and output levels (achievable by padding).
- (iv) k' , n' , and n'' are described by nice parametric connections.

Let the gates (including inputs) of C_x be described by the set $\{g[s, i] : 0 \leq s \leq t, 1 \leq i \leq n'\}$. Here, the level of the gate is indicated by the first index. Note that on level t , only one gate (the output) is important (the padding is just a notational convenience). We may assume the output gate is $g[t, 1]$.

We consider the following uniform circuit family $\mathcal{F}_L = \{C_{m,k'}\}$, $m = t(n'')^2 + n'$. (To arrange for $\mathcal{F}_L$ to have one circuit for each possible pair of indices, simply pad with nonaccepting empty circuits for index pairs not of the indicated form.)

The circuit $C_{m,k'}$ is described as follows. There are $2t + 1$ levels of gates $L_0, \dots, L_{2t}$. The inputs to the circuit constitute level 0. The gate sets are described as follows:

$$L_0 = \{a_X[s, i, j] : 1 \leq s \leq t, 1 \leq i \leq n'', 1 \leq j \leq n''\} \cup \{a_Y[i] : 1 \leq i \leq n'\},$$

and for $s = 1, \dots, t$,

$$\begin{aligned} L_{2s} &= \{c[2s, i] : 1 \leq i \leq n''\}, \\ L_{2s-1} &= \{b[2s-1, i, j] : 1 \leq i \leq n'', 1 \leq j \leq n''\}. \end{aligned}$$

According to our assumption that t is even, we assign the following logic functions to these gates: for $s = 1, \dots, 2t$, the gates of L_s are $\wedge$ gates if s is congruent to 0 or 1 mod 4; the other levels are $\vee$ gates. The $a_X[*, *, *]$ inputs to the circuit have the role of describing the circuit C_x . The $a_Y[*]$ inputs represent the (non-deterministic) inputs to C_x . The gates on even-indexed levels L_{2s} provide a PLA-type template on which to simulate the circuit C_x . Note that these are large gates of the same logical character as the gates on level s of C_x . The gates on odd-indexed levels L_{2s-1} are small gates whose function is to interpret the description of C_x so that C_x can be simulated.

The $a_X[*, *, *]$ inputs describe C_x in the following way. Set $a_X[s, i, j] = 1$ if and only if, in C_x , the gate $g[s, i]$ takes input from $g[s-1, j]$. Let $\chi(C_x)$ denote the length $t(n'')^2$ 0-1 vector that describes C_x in this way.

The gates in the circuit $C_{m,k'}$ are connected as follows:

- (i) For $s = 1, \dots, t$, the gate $c[2s, i]$ receives input from each of the gates $b[2s-1, j]$ for $j = 1, \dots, n''$.
- (ii) For s odd, $2 \leq s \leq t$, the gate $b[2s-1, i, j]$ computes the boolean expression $c[2(s-1), j] \wedge a_X[s, i, j]$.
- (iii) For s even, $2 \leq s \leq t$, the gate $b[2s-1, i, j]$ computes the boolean expression $c[2(s-1), j] \vee \neg a_X[s, i, j]$.
- (iv) The gate $b[1, i, j]$ computes the boolean expression $a_Y[j] \wedge a_X[1, i, j]$.

Note that since the gates $b[*, *, *]$ are small, the simulation preserves weft. The following claim establishes that the circuit $C_{m,k'}$ works correctly.

Claim 1. For all $y \in \Sigma^{n'}$ of weight k' , $C_{m,k}(\chi(C_x) \cdot y) = 1$ if and only if $C_x(y) = 1$.

Claim 1 is easily proved by induction on the levels of the circuit simulation.

An essentially identical argument handles t odd, $t \geq 3$. The case of $t = 1$ presents additional difficulties and must be handled as a special case. (The simulation above would result in universal circuits of weft 2.)

It suffices to show a “universal” family of circuits for the $W[1]$ -complete problem **INDEPENDENT SET**. What we want is a weft 1 circuit that takes as input the concatenation of two strings x and y , where x describes a graph of order n and y represents the candidate k -element independent set. We can accomplish this by having the first part of the input $x = (x[1, 2], x[1, 3], \dots, x[n-1, n])$ represent the adjacencies of G as a 0-1 string of length $\binom{n}{2}$, and letting $y = (y[1], \dots, y[n])$ (the nondeterministic part of the input) have length n and weight specification k . The circuit can simply represent the boolean expression

$$C = \prod_{1 \leq i < j \leq n} (\neg x[i, j] \vee \neg y[i] \vee \neg y[j]).$$

The above arguments show that $W[t] \subseteq N[t]$. For the reverse inclusion, suppose L is a parameterized language in $N[t]$. Then, $\langle x, k \rangle \in L$, $|x| = n$, if and only if $\exists y \in \Sigma^{n'}$ of weight k' , such that a nicely produced weft t circuit $C_{n, "k"}$ accepts xy . To exhibit a reduction from L to **WEIGHTED CIRCUIT SATISFIABILITY** for weft t , we may just take the image of the reduction to be $C_{n, "k"}$ with the first $n'' - n'$ inputs “removed” by being fixed to the value of x .

In a similar way, we can prove the following characterization of $W[P]$ (Exercise 2).

Theorem 16.6 ([174]) $W[P] = \langle N[P] \rangle$.

Let Σ_d^P stand for the class of languages recognized by depth- d unbounded fanin boolean circuits of polynomial size having a single *or* gate at the output, as described in the survey by Boppana and Sipser [93]. Let Π_d^P stand for the complements of these languages, which are recognized by depth- d circuits with an *and* gate at the output. Sipser [426] showed that for all $d \geq 1$, $\Sigma_d^P \neq \Pi_d^P$. It is not surprising that this carries over to the parameterized setting to show that the $G[t]$ -hierarchy is proper. We prove that this carries over to the nondeterministic case.

Theorem 16.7 (Separation Theorem [174]) For all $t \geq 1$, $N[t] \subset N[t+1]$.

Proof. Suppose $N[t] = N[t+1]$, and let A_0 be a language in Π_{t+1}^{poly} . Define a simple parameterized language A by $A = \{\langle x, k \rangle : x \in A_0\}$. Then, $A \in G[t+1] \subseteq N[t+1]$. By our supposition, $A \in N[t]$. By the definition of $N[t]$, there exists a parameterized language $B \in G[t]$ accepted by a bi-indexed family of circuits $\mathcal{F} = \{C_{n,k}\}$ such that we have for all x and any fixed integer k_0 ,

$$\begin{aligned}
x \in A_0 &\iff (x, k_0) \in A \iff (\exists y \in \{0, 1\}^{n'})[\text{weight}(y) \\
&= k'_0 \wedge (xy, k''_0) \in B] \text{ iff } (\exists y \in \{0, 1\}^{n'})[\text{weight}(y) = k'_0 \wedge C_{n, n''_0}(xy)] = 1.
\end{aligned}$$

Here, again, the priming indicates that n and the fixed k_0 are part of nice parametric connections, with $n'' = |xy| = n + n'$.

Using C_{n, n''_0} as a building block, we can create a circuit $\tilde{C}_{n, n''_0}$ that evaluates $C_{n, n''_0}(xy)$ for all possible y , with an output $\vee$ gate on all these possibilities. There are $\binom{n'}{k'_0}$ possible y , but this is permitted since k'_0 is a constant. The family of circuits constructed from $\mathcal{F}$ in this way over all n show that $A_0 \in \Sigma_{t+1}^{\text{poly}}$, contradicting the fact that Π_d^{poly} is not contained in Σ_d^{poly} , for all d .

The above theorem does not prove, of course, that the $W[t]$ -hierarchy is proper. If we could prove that, then we would have $P \neq NP$. We would need something like $N[t] \not\subseteq G[P]$ for that. What it does show is that any “normal” approach of the kind often employed in the study of the W classes, namely the use of additional (bounded-weight) nondeterminism, will necessarily fail. For example, to show that $W[t+1]$ collapses to $W[t]$, we might hope to design some sort of gadgetry whose operation can be described by a weft t circuit C' , that would correctly verify that a circuit C of weft $t+1$ accepts a particular weight k input vector x on the basis of some additional $k' \log n$ bits of nondeterministic information. Collapse would then follow by using C' to process *two* guesses: the input x to C and the “proof” that $C(x) = 1$. Since x has bounded weight and the size of C' can involve a blowup in size of $f(k)n^{g(t)}$ for $|C| = n$ and arbitrary functions f and g , we might well believe that there is some hope for this project. However, if this program were to succeed, then we would, in fact, have shown that $G[t+1] \subseteq N[t]$. By the following easy but important proposition, in which the transitivity of parametric connections enables us to “coalesce” two like quantifiers into one, we would then have $N[t+1] \subseteq N[t]$, contradicting the Separation Theorem.

Lemma 16.8 *Let $\mathcal{F}$ be any class of parameterized languages. Then, $\exists \cdot \exists \cdot \mathcal{F} = \exists \cdot \mathcal{F}$.*

Although the parameterization of A_0 in the proof of the Separation Theorem is trivial by itself, the manner in which the parameter interacts with the definition of $\exists \cdot \text{Uniform } G[t]$ and with the switch between Σ_d and Π_d circuits is noteworthy, and, overall, the information in the theorem seems surprisingly good. It lends support to the conjecture that the $W[t]$ -hierarchy is proper.

The classes $N[t]$ are defined by a single bounded-weight existential quantification. It is natural to consider corresponding classes defined by universal and by alternating bounded-weight quantification. In this way, we would expect to obtain analogs of the classes $AW[t]$ of Chapter 14.

Definition 16.9 For each $t \geq 1$, define $\Sigma_1[t] = W[t] = \langle \exists \cdot \text{Uniform } G[t] \rangle$. Correspondingly, define $\Pi_1[t] = \langle \forall \cdot \text{Uniform } G[t] \rangle$. For $i \geq 2$, define $\Sigma_i[t] = \langle \exists \cdot \Pi_{i-1}[t] \rangle$ and $\Pi_i[t] = \langle \forall \cdot \Sigma_{i-1}[t] \rangle$. Define $\Sigma_0[t] = \Pi_0[t] = \langle G[t] \rangle = \text{FPT}$.

Finally, for each t , define $H[t]$ to be the union of these classes, viz.

$$H[t] = \bigcup_{i=0}^{\infty} \Sigma_i[t] \cup \Pi_i[t].$$

As one would expect, the $\Pi_i[t]$ classes consist of the complements of parameterized languages in the $\Sigma_i[t]$ classes. Moreover, by the methods of Theorem 12.3 and induction on i , it follows that the Σ_i -quantified analog of **WEIGHTED t -NORMALIZED SATISFIABILITY** is complete for $\Sigma_i[t]$. The proof of Theorem 16.6 can easily be modified to prove the following (Exercise 3):

Theorem 16.10 $AW[t] = \langle G[t] \rangle$.

The next theorem shows that in contrast to the proper inclusions of the $N[t]$ -hierarchy, the $H[t]$ -hierarchy collapses to $H[1]$. This result constitutes a significant improvement of Theorem 14.9, where it is shown that $AW[*] = AW[2]$. From the following, we have as an immediate corollary that $AW[*] = AW[1]$.

Theorem 16.11 (Trade-off Theorem [181]) *For all $t \geq 1$, $H[t] = H[1]$.*

Proof. By induction, it suffices to show that $H[t] \subseteq H[t-2]$, for t odd. Let $L \in \Sigma_s[t]$. We argue that $L \in \Sigma_{s+2}[t-2]$. By the above remarks, L is reducible to the Σ_s -quantified version of **WEIGHTED t -NORMALIZED SATISFIABILITY**. Accordingly, let E be a boolean expression over a set of variables V that is partitioned into sets $V = V_1 \cup \dots \cup V_s$ with $V_i \cap V_j = \emptyset$ for $1 \leq i < j \leq s$, such that E has the form

$$E = \prod_{i=1}^m \sum_{j=1}^{m_i} \prod_{k=1}^{m_{ij}} E[i, j, k],$$

where $E[i, j, k]$ is a literal if $t = 3$, and is otherwise a weft $t-3$ expression that is a large $\vee$ of weft $t-2$ subexpressions. Let $(k_1, \dots, k_s)$ be a sequence of positive integers. The quantified satisfiability question for E is whether

$\exists$ a weight k_1 truth assignment to the variables of V_1 , such that

$\forall$ weight k_2 assignments to the variables of V_2 ,

$\dots$,

such that E is satisfied. Now we describe an expression E' over a set of variables

$$V' = V \cup V_{\forall} \cup V_{\exists},$$

where

$$V_{\forall} = \{a[i] : 1 \leq i \leq m\}$$

and

$$V_{\exists} = \{e[i, j] : 1 \leq i \leq m, 1 \leq j \leq m_i\},$$

such that the answer to the quantified satisfiability question for E is “yes” if and only if the answer to the quantified satisfiability question for E' is “yes.” The latter is defined to hold iff

$\exists$ a weight k_1 assignment to V_1 , such that
 $\forall$ weight k_2 assignments to V_2 ,
 $\dots$,
 $\forall$ weight 1 assignments to $V_\forall$,
 $\exists$ a weight 1 assignment to $V_\exists$,
 such that E' is satisfied.

The expression E' is described by $E' = E'_1 \cdot E'_2$, where the two factors are

$$E'_1 = \prod_{i=1}^m \prod_{j=1}^{m_i} (e[i, j] \rightarrow a[i])$$

and

$$E'_2 = \prod_{i=1}^m \prod_{j=1}^{m_i} \prod_{k=1}^{m_{ij}} (E[i, j, k] \vee \neg e[i, j]).$$

For $t > 3$, since $E[i, j, k]$ is a large logical sum of subexpressions and has weight $t - 3$, the same is true for $(E[i, j, k] \vee \neg e[i, j])$, and therefore E' has weight $t - 2$. If $t = 3$, then E' is a product of sums of size 2 and, thus, has weight 1. The verification that the construction works correctly is straightforward and is left to the reader.

This proof does not tell us whether $\Sigma_s[t]$ is equal to $\Sigma_{s+2}[t - 2]$, and, in general, we do not know exactly how the hierarchies $H[t]$ intercalate for different t .

We can easily follow the analogy with NP and PH for other complexity results and classes. One such class could be generated by the result of Valiant and Vazirani: for randomized reductions, NP is the same detecting unique solutions. We could then look at analogs of this result. Downey et al. proved one such result in Ref. [174]. One might then analyze the question of whether the W -hierarchy might collapse under *randomized* reductions.

Historical Notes

All of the results of this chapter are taken from Ref. [174].

Exercises 16.0

16.0.1. Prove that $\text{Uniform } G[P] = FPT$.

[Hint: If a parameterized language L is in $\text{Uniform } G[P]$, then membership of $\langle x, k \rangle$ in L , $|x| = n$, can be decided in the right amount of time $O(n')$ by generating the circuit $C_{n,k}$ and evaluating it on input x . The converse also holds by imitating the usual proof that languages in P have polynomial-sized circuits.]

16.0.2. Prove that $W[P] = \langle N[P] \rangle$.

[Hint: Consider the defining problem **WEIGHTED CIRCUIT SATISFIABILITY** and the fact that reductions may be folded into circuits.]

16.0.3. Prove $AW[t] = \langle G[t] \rangle$.

16.0.4. (Downey and Fellows [166]) As another analog to classical notions, we define the D -hierarchy, $D_p[t]$, as the languages that can be expressed as the intersection of a language in $W[t]$ and one whose complement is in $W[t]$. Prove that the following problem is in $D_p[2]$.

UNIQUE WEIGHTED CNF SATISFIABILITY

Instance: A boolean CNF formula X and an integer k .

Parameter: A positive integer k .

Question: Is there a unique weight k satisfying assignment for X ?

[Hint: It suffices to describe how to say that a CNF expression has at least two satisfying expressions in $W[2]$. Let C be the circuit corresponding to X . Take two copies of C . Add $O|X|$ many gates to express the fact that the first copy of C has a satisfying assignment different from the second. Now, for k , choose q and r appropriately and take q copies of the left circuit and r copies of the right. Add new gates to express the fact that the inputs of the left q must all be equal and the fact that the inputs of the right r must all be equal. Now, accept C if the new circuit has a weight $(q + r)k$ accepting input. Then, for the correct choice of (q, r) , depending only on k , C has two or more accepting inputs iff the new circuit has one of weight $(q + r)k$.]

Relationships with Classical Complexity and Limited Nondeterminism

17.1 Classical Complexity

In the previous chapter, we looked at some evidence that the W -hierarchy does not collapse. Here, we will begin by looking at further results that may support this hypothesis. We analyze the following question:

“Does W -hierarchy collapse imply anything unreasonable in the classical complexity classes such as NP ?”

Now, one might hope that $W[P] = FPT$ might imply $P = NP$, which would be very strong evidence that $W[P] \neq FPT$. Unfortunately, we cannot prove this implication, and in the next section, we construct an oracle A relative to which $P \neq NP$ and yet $W[P] = FPT$. This oracle construction would seem to give at least some evidence that the implication does not hold, and in fact, if it does, then it will require nonrelativising techniques to prove it.

In any case, purely on general grounds we believe that all we could hope for is some sort of “smaller” collapse than $P = NP$. To express our results, we use the *Limited Nondeterminism* classes introduced by Kintala and Fischer [297]. The computation model is a Turing machine.

Definition 17.1 (i) (Kintala and Fischer [297]) Let $NP[f(n)]$ denote the class of decision problems that are solvable in P -time with an algorithm using only $f(n)$ bits of nondeterminism.

(ii) (Abrahamson, Downey, and Fellows [3, 4]) We define the class $SUBEXP TIME(f(n))$ to be the set of languages L for which there is a polynomial $p(n)$ such that L is accepted in $DTIME(p(n)2^{g(n)})$ for some function g in $o(f(n))$ (that is, $\lim_{n \rightarrow \infty} f(n)/g(n) = \infty$).

We have the following result connecting parameterized and classical complexity.

Theorem 17.2 ([4]) *For the reducibility $\leq_T^s$, and hence for the standard reducibility, $W[P] = FPT$ iff for every P -time function f with $f(n) \geq \log n$, there is a recursive function h such that for every $L \in NP[f(n)]$, $h(L)$ computes machine M which witnesses that $L \in SUBEXP TIME(f(n))$. [We say that $NP[f(n)]$ is constructively a subset of $SUBEXP TIME(f(n))$.]*

Corollary 17.3 ([4]) *For $\leq_T^s$, $W[P] = FPT$ implies that for all P -time f , $NP[f(n)] \subseteq SUBEXP TIME(f(n))$.*

Proof. ($\Rightarrow$) Suppose $W[P] = FPT$. Then **WEIGHTED CIRCUIT SATISFIABILITY** is solvable by some procedure Φ in time $g(k)p(n)$, where p is a polynomial and g is some function. We can assume that g is strictly increasing and that it grows rapidly, since any function can be bounded above by such a function.

Let $f(n) \geq \log n$ be polynomial-time computable, and let α be a nondeterministic polynomial-time algorithm that uses at most $f(n)$ bits of nondeterminism on inputs of length n . The following algorithm β simulates α , and we will show that β runs in time $q(n)2^{o(f(n))}$ for a polynomial q . For $k < m$, let $E_{m,k}$ be a canonical list of the size k subsets of $\{1, \dots, m\}$. If s is member of $E_{m,k}$, let $N(s)$ be the index of s in list $E_{m,k}$. List $E_{m,k}$ can be chosen so that $N(s)$ can be computed in polynomial time in m . Let $[N(s)]_d$ be the d least significant bits of the binary representation of $N(s)$.

Algorithm $\beta(x)$:

$n \leftarrow |x|$.

$k \leftarrow f(n)/\log \log n$.

loop

$m \leftarrow \lceil k2^{f(n)/k} \rceil$.

$C \leftarrow$ a circuit with m inputs, and the following behavior.

On an input vector v of weight k , C considers v to encode a size k subset s_v of $\{1, \dots, m\}$, and computes $n_v = [N(s_v)]_{f(n)}$.

C then simulates α on input x with guess n_v , and produces the same output as α does with that guess.

Run Φ on input $\langle C, k \rangle$ for at most $p(n)2^{f(n)/\log \log n}$ steps.

If Φ terminates within the allotted time, stop and give the same answer that it gave.

$k \leftarrow k - 1$

end loop

Since $\binom{m}{k} \geq (m/k)^k = \lceil k2^{f(n)/k} / k \rceil^k \geq 2^{f(n)}$ at each iteration, all $2^{f(n)}$ binary strings of length $f(n)$ can be passed as guesses to α , when all weight k inputs are tried. It is evident, therefore, that if $\beta(x)$ terminates, then it gives the same answer that $\alpha(x)$ does. We show that $\beta(x)$ terminates within time $q(n)2^{o(f(n))}$.

Variable m is largest for small values of k , so the last loop iteration costs the most. Since Φ terminates in time $g(k)p(n)$, the last iteration is the first one where $g(k) \leq 2^{f(n)/\log \log n}$; that is, in the last iteration, $k = g^{-1}(2^{f(n)/\log \log n})$. [Here, for an increasing function $z(n)$, we define $z^{-1}(n)$ to be the largest m with $z(n) \leq m$.]

(Note that the initial value of k is larger than that for sufficiently rapidly growing functions g .) Considering this smallest value of k to be a function $k(n)$ of n , observe that $\lim_{n \rightarrow \infty} k(n) = \infty$.

The initial value of k is computable in polynomial time in n . The cost of producing circuit C is polynomial in m and n . But $m(n) = k(n)2^{f(n)/k(n)}$ is in $2^{o(f(n))}$, so any polynomial of m is also $2^{o(f(n))}$. The simulation of Φ is bounded in time by $p(n)2^{o(f(n))}$, so each iteration of the loop uses time at most $q(n)2^{o(f(n))}$ for some polynomial q . Since there are at most $f(n)/\log \log n$ iterations, the entire algorithm runs within time $q(n)2^{o(f(n))}$.

To complete this half of the proof, note that β is computable from α .

($\Leftarrow$) Suppose that $NP[f(n)]$ is constructively a subset of $SUBEXP TIME(f(n))$ for every polynomial time function $f(n) \geq \log n$. Let $f_k(n) = k \log n$.

For each k , let WCS_k be the restriction of **WEIGHTED CIRCUIT SATISFIABILITY** to inputs of the form (x, k) . Then, WCS_k is in $NP[f_k(n)]$. Let α_k be a deterministic algorithm that solves WCS_k in time $p(n)2^{f_k(n)/g(n)} = p(n)n^{k/g(n)}$, where p is a polynomial and $\lim_{n \rightarrow \infty} g(n) = \infty$. By the constructive nature of the supposition, α_k is a recursive function of k . So, to solve **WEIGHTED CIRCUIT SATISFIABILITY** on input (x, k) , compute α_k , and run it on input (x, k) . Call the resulting algorithm β .

Algorithm β solves **WEIGHTED CIRCUIT SATISFIABILITY** in time $h(k)p(n)n^{k/g(n)}$ for some function h . For $k < g(n)$, this time is $h(k)q(n)$ for a polynomial q . For $k \geq g(n)$, the time is bounded by a function of k . So, there is a function h' such that algorithm β runs in time $h'(k)q(n)$.

Notice that we only explicitly use the fact that we can compute $k(n)$ as a function of n to get the constructivity conclusion in the $\rightarrow$ part and similarly only use constructivity in the other half to get *strong* uniformity. In both cases, the same proof will show that

Corollary 17.4 *For the reducibility $\leq_T^u$, $W[P] = FPT$ iff for all P -time $f \geq \log n$, $NP[f(n)] \subseteq SUBEXP TIME(f(n))$.*

The theorems above say nothing about the classes $W[t]$, and, in particular, the class $W[1]$ which is our current measure of intractability. We will see in the next section that this class and “log”alternating Turing machines are closely related. In this section, we offer one result. We remark that all of the results we have seen regarding $W[t]$ collapse seem to be model dependent since the definition of the class $W[t]$ itself comes from restricted logical depth of formulas.

Definition 17.5 ([4]) Let $\mathcal{C}$ be a class of decision circuit problems. We shall say that $\mathcal{C}$ is *nearly polynomial time* if there is an algorithm Φ such that for each $C \in \mathcal{C}$, Φ solves $\langle C, k \rangle$ in time $p(n)2^{o(v)}$, where $p(n)$ is a polynomial and v is the collection of input variables to C .

Theorem 17.6 (a) (Abrahamson, Downey and Fellows [4]) for t even, and Downey and Fellows for t odd.) *For $t > 0$, $W[t] \subseteq FPT$ implies that $SAT[t]$ (where $SAT[t]$ denotes (unweighted) t -normalized satisfiability) is nearly polynomial time.*

(b) (Abrahamson, Downey and Fellows [4])

(i) $W[\text{SAT}] \subseteq \text{FPT}$ implies SAT is nearly polynomial time.

(ii) $W[P] \subseteq \text{FPT}$ implies **WEIGHTED CIRCUIT SATISFIABILITY** is nearly polynomial time.

Proof. We prove (a), the others follow by the same method (Exercise 1).

Case 1. Even Weft.

Suppose that $W[2t]$ is FPT . Then, there is a $f(k)p(n)$ time algorithm for solving **WEIGHTED MONOTONE $2t$ -NORMALIZED SATISFIABILITY** on instances $\langle x, k \rangle$ with $|x| = n$, for some polynomial $p(n)$ and arbitrary recursive function $f(k)$. We assume that $f(k) > k$, and as in the previous proof, we let $f^{-1}(k)$ denote the largest y with $f(y) \leq k$. Furthermore, we can assume that $f(k)$ is computable in time $f(k)$, so that we can easily see if $f(k) \geq y$.

Now, let X be a member of $\text{SAT}[2t]$, that is, a $2t$ -normalized boolean formula, and consider it as a circuit with $2v$ input lines for the variables and their complements. We construct a pair $\langle C, k \rangle$ with C a circuit and k an integer to be chosen later. We ensure that C is a $2t$ -normalized monotone circuit and that X is satisfiable iff C has a weight k satisfying assignment. The reader should note that the reduction will *not* be a parameterized one. Let $s = 2^{\lceil v/k \rceil}$. The circuit C has four distinct parts:

- (1) A decoding circuit E with ks input lines from variables $\{z_1, \dots, z_{ks}\}$.
- (2) A circuit X' emulating the action of X .
- (3) An enforcement circuit connected to the ks inputs.
- (4) An additional *and* gate taking the outputs of X' and the enforcement circuit. The single output of this *and* gate is the output for the circuit. The reader is referred to Figure 17.1.

The ks input lines of the encoding circuit can be considered a $k \times s$ matrix. The encoding circuit translates each row of the input into two binary numbers. Each binary number represents the position of the *single* 1 in that row. (The enforcement circuit will be weft 1 and will express the fact that *the input matrix has at most one 1 per row*. Since we will be looking for weight k inputs to C , this will ensure that there is exactly one 1 per row.) Each binary number has length $\log s$ and the collection of possible binary strings of length $\log s$ will be in exact one-to-one correspondence with the position of the 1 on the row. The binary numbers are complements of each other (meaning that whenever one is 1, the other is 0). One corresponds to negated variables and one corresponds to variables. This can all be easily achieved disjunctively using $2 \log s$ large *or* gates. For instance, for the positive variables, suppose the binary number is to be represented by $x_1 \dots x_{\log s}$. We then have one *or* gate corresponding to x_1 . This is connected to every second position on the row. Thus, x_1 will be 1 iff the 1 on the row falls on an even square. (Similarly, if we want the negation of x_1 in the other binary string, we use all the odd positions.) For x_2 , we have a large *or* attached to positions $4n + 3$ and $4n + 4$ of the row for each n . In general, we use $2^j n + 2^{j-1} n + 1, \dots, 2^{j+1} n$ for each n for x_j . The idea here is that the $2v$ inputs to X are chopped into k equal pieces (sequentially), each of length $\log s$. We note that each row has length s , so we will

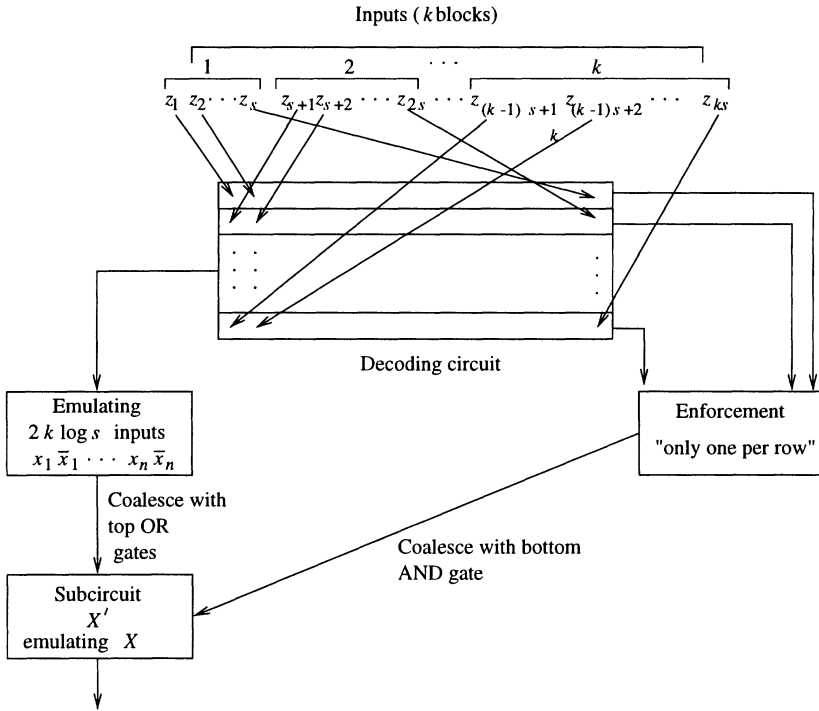


FIGURE 17.1. The Even Case of Theorem 17.6(a)

indeed be able to create a total of $2k \log s = 2k \times \log 2^{\lceil v/k \rceil} \geq 2v$ lines in this way. It is clear that we can use these lines as the inputs to the circuit X' . Circuit X' is the same as X , except that each literal is replaced by one of the $2v$ output lines. Clearly, the circuit can be massaged to be $2t$ -normalized by coalescing the conjunction at the output of X' with the bottom *and*, and the disjunctions of the encoding circuit with those of the upper level of X' .

Now, to complete the proof, for a given input with v variables, we choose $k = f^{-1}(2^{v/\log v})$. Note that $f(k) < o(v)$. This means that the $f(k)$ factor in the solution size is acceptable. The size of C is P -time in X and ks . As k is an unbounded function of v , we see that $s < 2^{o(v)}$. The running time of Φ on C is $f(k)p(|C|)$. This is $2^{o(v)}p(|X| \cdot 2^{o(v)})$. But this quantity is $p(|X|) \cdot 2^{o(v)}$, giving the desired result.

Case 2. Odd Weft ≥ 3 .

For this case, we use the trick from Step 6, Case 2 of the Normalization Theorem, Theorem 12.3. Let $t \geq 1$, and this time, let X be a weft $2t + 1$ -normalized circuit and let $A_1, \dots, A_m$ list the *And* gates at the top layer, below the inputs. For each output line from the A_i , introduce two inverters, and using De Morgan's laws, draw the top inverter past A_i to make it a (large) *Or* gate. Now, apply the process of Case 1 to the resulting equivalent circuit.

Case 3. Weft 1.

This time let X be a weft 1 circuit representing a 3CNF formula. The trouble with the above is that the enforcement gadget is a collection of *large Or* gates. If we use the process of Case 3, we would convert a weft 1 circuit into a weft 2 circuit. To get around this difficulty, we replace the large gates by a collection of small gates using the fact that the clause size is 3.

Again, we will have ks input variables in k blocks of size s . Again we will have the enforcement gadget of Cases 1 and 2. (This was in normalized weft 1.) Again, we will think of positions of the (blocks) rows of the input as encoding input variables or their negations to the circuit X .

However, now we will not explicitly have the circuit X' emulating X directly. Rather, we will have a new circuit or weft 1 taking the ks inputs directly. At the bottom, we will have a large *And* gate A . Suppose that $C = \{x_i, x_j, x_k\}$ is a clause of X . Now, we wish to wire into the circuit all the possibilities that make C *false*. Note that the literals can correspond to at most three blocks of input variables. If all the literals of C come from the same block B , then for each z_m in block B which corresponds to the placement of a 1 in the block corresponding to negating all the literals in clause C , we simply have a line passing through an inverter. This line directly inputs to gate A . Suppose that the literals come from more than one block, and, without loss of generality, from three blocks, B_i , B_j , and B_k . (Here we suppose that x_d is set in block B_d .) Then, for each setting of three input variables z_{d_p} from block B_p , $d_p \in \{i, j, k\}$, one z_{d_p} per block, which make the clause *false*, create a gate expressing the fact that $\overline{z_{d_i} \wedge z_{d_j} \wedge z_{d_k}}$. Run the gate's output directly to gate A . This is a small *or* gate. We create at most $O(|X|^3)$ gates in this way. The circuit we get is satisfiable by a weight k input iff X is satisfiable. The remaining reasoning goes through as before.

Since $v < |X|$, the reader should note that the above implies the following corollary:

Corollary 17.7 (Downey and Fellows) *If $W[1] = FPT$, then*

$$3SAT \subseteq DTIME(2^{o(n)}).$$

It would seem unlikely that $3CNFSAT \subseteq DTIME(2^{o(n)})$ if $P \neq NP$, and so Corollary 17.7 seems to provide some evidence that $W[1] \neq FPT$.

Historical Notes

The material in this section is from Refs. [3] and [4], or is unpublished work of the authors.

Exercises 17.1

- 17.1.1.** (Abrahamson, Downey, and Fellows [4]) Prove (ii) and (iii) of Theorem 17.6.

17.2 Nondeterminism in P , $LOGNP$, and the Cai-Chen Model and Other Models

Ever since the Kintala-Fischer [297] article, authors have looked at limiting the amount of nondeterminism allowed in an oracle computation. We have already seen that our work has impact in these smaller classes. Up to the work of Abrahamson et al. [5], authors had always studied these smaller classes “by themselves.” In some ways, this is akin to our studying a single k -slice of a problem rather than looking at the asymptotic behavior as $k \rightarrow \infty$. Immediately, one is faced with the problem of finding good reducibilities. The trouble is that we are now looking at “nondeterminism in P ,” and, hence, the reducibilities will need to be bounded by some polynomial and hence be machine dependent. Typically, these studies fix a model. This model is usually a multitape Turing machine, so that the Linear Speedup Theorem works, and we do not need worry about multiplicative constants. Such studies into limited nondeterminism in P include Álvarez, Díaz, J. Torán [19], Buss and Goldsmith [101], Gurevich and Shelah [253], Kintala and Fischer [297], Joseph, Pruim, and Young [284], Kaye [290], and Regan [385]. All of these studies attempt to study the problems by limiting the number of nondeterministic moves available and keeping the complexity of the reductions down. The Kintala–Fischer [297] article is rather technical. As we have seen in a Section 17.1, Kintala and Fischer define a class we denote by $NP[f(n)]$. This class is the collection of languages accepted by nondeterministic (multitape) Turing machines allowing only $f(n)$ c -ary nondeterministic moves. Concentrating on $NP[\log^k n]$ for $k > 0$, they use a Baker–Gill–Solovay [48] type construction (see Theorem 19.6) to get an oracle separation of the hierarchy $NP[\log n] \subseteq NP[\log^2 n] \cdots \subseteq NP[n]$.

Geske [243] similarly studies the relationship between the classes $DTIME(n^q)$ and $NTIME(n^q)$. The classes in P looked at by Buss and Goldsmith [101] are rather more elaborate, and, furthermore, they define reduction procedures to enable them to address completeness issues. Fix the model as a multitape machine. They define

$$P_q = \cup_{d \in \mathbb{N}} DTIME[n^q \log^d n]$$

and $N^m P_q$ to be the collection of languages accepted by nondeterministic machines running in time $O(n^q \log^d n)$ (for some d), making at most $m \log n$ nondeterministic binary moves in the computation. In the Buss–Goldsmith article, it is shown that the language

$$G_q^m = \{m\#x\#1^j : M \text{ accepts } x \text{ within } j|x|^{q-1} \text{ steps} \\ \text{using at most } m \log |x| \text{ many choices}\}$$

is complete for $N^m P_q$ under the quasi-linear reductions. Here, a reduction is called *quasi-linear* if it runs in time $O(n \log^d n)$ for some d . (The notion of quasi-linear time also occurs in Ref. [253].) The reader should note that the Buss–Goldsmith article clearly demonstrates the problems which arise when one tries to study the types of computational problems that are the basis of this book, by only studying a *single* slice of a parametrized language. One is forced to only consider reductions

that are naturally small with respect to the possible amount of nondeterminism in that particular slice. Not only does this fact make the setting rather sensitive to the model chosen, but, as with the Abrahamson et al. article [5], one then needs the languages themselves to be complete for P , subject to the refined reductions, and hence there seems to be no way to address the apparent parametric intractability of **3SAT**, **DOMINATING SET**, **CLIQUE**, or many other natural problems. Buss and Goldsmith do show that the Abrahamson et al. [5] completeness results and hence some of our $W[P]$ -completeness results can be recast in their setting. To do this, Buss and Goldsmith observe that the reductions of Abrahamson et al. [5] can be made to be quasi-linear in the following sense: For a parameterized set C , it becomes what they call N^*P_q -complete if for each k , there is a $k' \geq k$ such that $C_{k'}$ is hard for N^kP_q with respect to quasi-linear reductions. In this setting, for instance, we discover that **WEIGHTED SATISFIABILITY** is N^*P_2 -complete. In many ways, these results can be viewed as an analysis of the complexity of the reductions used in the Abrahamson et al. [5] article, completeness results from the Buss-Goldsmith setting will lift to ones for W .

The article of Regan [385] takes a rather different tack. Regan noted in passing that the question of the existence of an n^c algorithm to determine if a k -clique existed in a graph G of size n was nontrivial (this seems to be the first place this observation is mentioned). This observation led him to study the structure of the k -th slice of what we call parameterized problems. Again, he only considers the local rather than the global behavior, concentrating on a fixed k . In Ref. [385], he analyses the structure of such sets by analogy with sparse sets. He studies the density of hard instances and the extent to which an analog of Mahaney's theorem [342] (the existence of sparse NP -complete sets implies $P = NP$) holds. He shows that some differences do exist and leaves open several interesting questions. Liming Cai and Chen [108, 105] look at another variation of the Kintala-Fischer-type model which they refer to as the *Guess and Check* model.

Definition 17.8 (Guess and Check [105]) Let $s(n)$ be a function and let $\mathcal{C}$ be a complexity class. A language L is in the class $GC(s(n), \mathcal{C})$ if there are a language $A \in \mathcal{C}$ and an integer $c > 0$ such that for all x , $x \in L$ if and only if $\exists y \in \{0, 1\}^*$, $|y| \leq c \cdot s(|x|)$, and $(x, y) \in A$.

Intuitively, the first component $s(n)$ in the $GC(s(n), \mathcal{C})$ model specifies the length of the guessed string y , which is the amount of nondeterminism allowed to be made in the computation, whereas the second component $\mathcal{C}$ specifies the verifying power of the computation.

The classes relevant to the W -hierarchy are those of the form $GC(s(n), \Pi_h^B)$, where $s(n) = \omega(\log n)$, and Π_h^B is the class of languages accepted by $O(\log n)$ time alternating Turing machines that make at most h alternations and always begin with universal states (such alternating Turing machines will be called " Π_h^B -ATMs"). Cai and Chen note that, to our current knowledge, a deterministic polynomial-time computation can only "guess" a string of length at most $\Theta(\log n)$ by exhaustively enumerating all strings of length $\Theta(\log n)$. Moreover, Yao [468] has proven that the

class Π_h^B is a proper subclass of the class P . It seems a reasonable conclusion that the model $GC(s(n), \Pi_h^B)$ has a (presumably) stronger guessing ability but (provably) weaker computational power than deterministic polynomial-time computations.

Theorem 17.9 ([103]) *For any nondecreasing function $t(n)$ constructible in polynomial time and for any integer $h > 1$, the following language $\text{CSAT}(t, h)$ is complete for the class $GC(t(n) \log(n), \Pi_h^B)$ under the polynomial-time reduction.*

$\text{CSAT}(t, h)$: the set of all strings of the form $x = \alpha\#w$, where w is an integer not larger than $t(|x|)$, and α is a Π_h -circuit that accepts an input of weight w .

Proof. In [103], Theorem 4.7 (Exercise 1) Liming Cai and Chen proved that for any nondecreasing function $t(n)$ constructible in polynomial time, the language $\text{CSAT}(t, h)$ is hard for the class $GC(t(n) \log(n), \Pi_h^B)$ under the polynomial-time reduction, for all $h > 1$. Therefore, to prove the theorem, we only have to show that the language $\text{CSAT}(t, h)$ is in the class $GC(t(n) \log(n), \Pi_h^B)$. Consider the algorithm $\text{CSAT}(t, h)$ -SIMULATOR in Figure 17.2. We shall show that the algorithm $\text{CSAT}(t, h)$ -SIMULATOR can be implemented by a Π_h^B -ATM such that for all $x = \alpha\#w$, $x \in \text{CSAT}(t, h)$ if and only if there is a string $y \in \{0, 1\}^*$, $|y| \leq 2t(|x|) \log(|x|)$ such that the algorithm accepts (x, y) .

The binary string y of length $2w \log(|x|)$ encodes a weight w input v_y of the circuit α as follows. The first $(w + 1) \log(|x|)$ bits of y are interpreted as $w + 1$ pairs $(p_0, p_1), (p_1, p_2), \dots, (p_{w-1}, p_w), (p_w, p_{w+1})$, with $0 = p_0 < p_1 < p_2 < \dots < p_w < p_{w+1} = m + 1$, such that the j -th bit of v_y is 1 if and only if $j = p_i$ for some i , $0 < i < w + 1$. Here, without loss of generality, we assume that $\log(|x|) \geq 2|m|$, so that a pair (p_i, p_{i+1}) of integers with $0 \leq p_i, p_{i+1} \leq m + 1$ can be encoded into $\log(|x|)$ bits. The rest part of the string y will be ignored in the simulation.

It is not very difficult to verify that the step “Checking” of the algorithm can be implemented by a Π_2^B -ATM. Now, we analyze the step “Simulation.”

Since the output gate of α is an $\wedge$ gate, the first execution of the loop body in Step 2 is a universal branch, which is combined with the starting universal branching of the algorithm to form the first phase of the algorithm.

The loop execution of Step 2 simply simulates the computation of the circuit α . After h executions of the loop body in Step 2, the algorithm is in its h -th phase, which is an existential phase if h is even, or a universal phase if h is odd. Therefore, the branching in Step 3 can be combined with the last phase in Step 2 to form the last phase of the algorithm. Therefore, the algorithm $\text{CSAT}(t, h)$ -SIMULATOR makes at most h alternations. Moreover, it is obvious that each computation path of the algorithm takes time $O(\log n)$. This concludes that the algorithm $\text{CSAT}(t, h)$ -SIMULATOR can be implemented by a Π_h^B -ATM.

Recall that v_y is the weight w boolean string of length m such that the j -th bit of v_y is 1 if and only if there is a pair (p_i, p_{i+1}) in y , $i > 0$ and $j = p_i$. We show that the algorithm $\text{CSAT}(t, h)$ -SIMULATOR on input (x, y) correctly simulates the circuit α on input v_y , where $x = \alpha\#w$. First, assume that h is an even number.

Algorithm CSAT(t, h)-SIMULATOR

INPUT: (x, y) , where $x = \alpha \# w$ and α is a circuit with input $v_1 \cdots v_m$.

Universally branch into step Checking and step Simulation.

Checking {Syntax Checking}

Reject if any of the following conditions is not satisfied.

- $w \leq m$ and α is a Π_h -circuit;
- $|y| = 2w \log(|x|)$ and the first $(w + 1) \log(|x|)$ bits of y encode $w + 1$ pairs $(p_0, p_1), (p_1, p_2), \dots, (p_w, p_{w+1})$, where $0 = p_0 < p_1 < p_2 < \dots < p_w < p_{w+1} = m + 1$ and each pair (p_i, p_{i+1}) consists of exactly $\log(|x|)$ bits.

Simulation {Circuit Simulation}

- Let g be the output gate of α ;
- Repeat the following loop h times:
 - if g is an $\wedge$ -gate, then universally guess an input g' of g . Let $g = g'$;
 - if g is an $\vee$ -gate, then existentially guess an input g' of g . Let $g = g'$;
- {At this point, g is an input node of the circuit α . This step is different for the case h is even and for the case h is odd.}
 - [In case h is even]
 - Existentially read a pair (p_i, p_{i+1}) from the string y ;
 - (a) If g is a positive literal v_j , then accept if and only if $p_i = j$;
 - (b) If g is a negative literal $\bar{v}_j$, then accept if and only if $p_i < j < p_{i+1}$.
 - [In case h is odd]
 - Universally read a pair (p_i, p_{i+1}) from the string y ;
 - (a) If g is a positive literal v_j , then reject if and only if $p_i < j < p_{i+1}$;
 - (b) If g is a negative literal $\bar{v}_j$, then reject if and only if $p_i = j$.

FIGURE 17.2. The Algorithm CSAT(t, h)-SIMULATOR

Let g_0 be a gate at input level of α , which must be an $\vee$ gate. Therefore, the gate g_0 gets value 1 on the input v_y if and only if either an input of g_0 is a positive literal v_j and there is a pair (p_i, p_{i+1}) in y such that $j = p_i$, or an input of g_0 is a negative literal $\bar{v}_j$ and j is not any p_i contained in y ; i.e., there is a pair (p_i, p_{i+1}) in y such that $p_i < j < p_{i+1}$. This process is correctly simulated by a last phase of the algorithm CSAT(t, h)-SIMULATOR, which guesses an input of g_0 in the last phase of Step 2 and a pair (p_i, p_{i+1}) in Step 3 and checks the above conditions. The case that $h > 1$ is odd can be verified similarly.

This concludes that the algorithm CSAT(t, h)-SIMULATOR is a Π_h^B -ATM, which accepts (x, y) , where $x = \alpha \# w$, if and only if y is a string of length $2w \log(|x|)$ that encodes a weight w boolean string v_y of length m as described above, and the circuit α accepts the input v_y .

For any $x = \alpha\#w$, if $x \in \text{CSAT}(t, h)$, then $w \leq t(|x|)$, and α accepts an input v of weight w . Let y_v be the binary string of length $2w \log(|x|)$ that encodes v as described above. Then, the pair (x, y_v) is accepted by the algorithm $\text{CSAT}(t, h)$ -SIMULATOR, and $|y_v| = 2w \log(|x|) \leq 2t(|x|) \log(|x|)$. On the other hand, if $x \notin \text{CSAT}(t(n), h)$, then the pair (x, y) is not accepted by the algorithm $\text{CSAT}(t, h)$ -SIMULATOR for any y of length at most $2t(|x|) \log(|x|)$. [Note that by the algorithm, a string of length at most $2t(|x|) \log(|x|)$ cannot be used to encode an input of α that has weight larger than $t(|x|)$.]

Thus, the language $\text{CSAT}(t, h)$ is in the class $GC(t(n) \log(n), \Pi_h^B)$ and consequently, is complete for the class $GC(t(n) \log(n), \Pi_h^B)$ under the polynomial-time reduction.

Cai and Chen have the following necessary and sufficient condition for $W[h]$ to collapse to FPT .

Theorem 17.10 ([105, 106]) *For all $h > 1$, $W[h] = FPT$ if and only if there is an unbounded, nondecreasing function $t(n) \leq n$ computable in polynomial time such that the class $GC(t(n) \log(n), \Pi_h^B)$ is a subclass of the class P .*

Proof. Suppose that $GC(t(n) \log(n), \Pi_h^B)$ is a subclass of the class P , then by Theorem 17.9, the problem $\text{CSAT}(t, h)$ is in P . An algorithm h -SOLVER solving the parameterized problem **WEIGHTED h -NORMALIZED SATISFIABILITY** can be described as follows: Given an input $\langle \beta, k \rangle$, first compare the values k and $t(|\beta|)$. If $k > t(|\beta|)$, then the algorithm tries all possible assignments of weight k to the expression β to see if any of these assignments satisfies the expression β . In the case $k \leq t(|\beta|)$, the algorithm first converts the expression β into a Π_h circuit α_β , then solves the problem $\text{CSAT}(t, h)$ on input $x = \alpha_\beta\#k$.

We analyze the algorithm h -Solver above. Define

$$\tau(k) = \min\{m \mid \text{for all } n \geq m, t(n) \geq k\}.$$

Since $t(n)$ is unbounded, nondecreasing, and computable in polynomial time, the function $\tau(k)$ is well defined and recursive. In the case $k > t(|\beta|)$, the size $|\beta|$ of the boolean expression β is bounded by $\tau(k)$. In particular, the number of variables in the Boolean expression β is bounded by $\tau(k)$. Thus, the number of assignments of β that have weight k is bounded by $(\tau(k))^k$. Therefore, in this case, the algorithm h -Solver takes time $O((\tau(k))^k |\beta|^2)$ to examine all possible assignments of weight k to the expression β , assuming we are using a straightforward quadratic-time algorithm to evaluate a boolean expression given an assignment. On the other hand, if $k \leq t(|\beta|)$, then the algorithm h -Solver takes time $O(|\alpha_\beta|^c) = O(|\beta|^c)$ for some constant c independent of the parameter k because of our assumption that the problem $\text{CSAT}(t, h)$ is in P . In conclusion, the algorithm h -SOLVER runs in time $O((\tau(k))^k |\beta|^d)$, where $d \geq 2$ is a constant independent of the parameter k ; that is, the problem **WEIGHTED h -NORMALIZED SATISFIABILITY** is fixed-parameter tractable. Therefore, by the Normalization Theorem, $W[h] = FPT$.

Conversely, suppose that $W[h] = FPT$, where $h > 1$. By the Normalization Theorem, the parameterized problem **WEIGHTED h -NORMALIZED SATISFIABILITY**

is fixed-parameter tractable. Suppose that h -SETTLER is an algorithm of running time $O(f(k)n^c)$ which solves the problem **WEIGHTED h -NORMALIZED SATISFIABILITY**, where f can be assumed to be an unbounded nondecreasing recursive function and $c \geq 1$ is a constant. Let t be the inverse function of f defined by

$$t(m) = \max\{n \mid f(n) \leq m\}.$$

Without loss of generality, we can assume that $t(n) \leq n$ and that $t(n)$ is unbounded, nondecreasing, and computable in time polynomial in n (as in Ref. [105]). Now an algorithm $\text{CSAT}(t, h)$ -SETTLER solving the problem $\text{CSAT}(t, h)$ can be described as follows: On input $x = \alpha\#w$, where α is a Π_h circuit, and $w \leq t(|x|)$, first convert α into an equivalent h -normalized boolean expression β_α , then call the algorithm h -Settler on input $\langle \beta_\alpha, w \rangle$.

Since the value $t(|x|)$ can be computed in time bounded by a polynomial in $|x|$, the condition $w \leq t(|x|)$ can be checked in polynomial time. The h -normalized boolean expression β_α equivalent to the Π_h circuit α has size $|\beta_\alpha|$ bounded by $|\alpha|^h$ because the depth of the circuit α is bounded by h . Moreover, it is easy to see that the h -normalized boolean expression β_α can be constructed from the Π_h circuit α in time $O(|\beta_\alpha|^2)$. Now by our assumption, the running time of the algorithm h -SETTLER on input $\langle \beta_\alpha, w \rangle$ is bounded by $O(f(w)|\beta_\alpha|^c)$. Since $w \leq t(|x|)$ and the function f is nondecreasing, we have $f(w) \leq f(t(|x|)) \leq |x| = O(|\beta_\alpha|)$, where the second inequality comes from the fact that the function t is the inverse function of the function f . Therefore, the time complexity of the algorithm $\text{CSAT}(t, h)$ -SETTLER is bounded by $O(|\beta_\alpha|^2) + O(|\beta_\alpha|^{c+1}) = O(|\alpha|^{h(c+1)}) = O(|x|^{h(c+1)})$.

This shows that the problem $\text{CSAT}(t, h)$ can be solved in polynomial time. By Theorem 17.9, the language $\text{CSAT}(t, h)$ is complete for the class $GC(t(n) \log(n), \Pi_h^B)$ under the polynomial-time reduction. Therefore, the class $GC(t(n) \log(n), \Pi_h^B)$ is a subclass of the class P .

Liming Cai and Chen argue that the above theorem gives strong evidence that the W -hierarchy does not collapse. Their argument is as follows. A deterministic polynomial-time computation can enumerate all strings of length $c \log n$ for a fixed constant c . However, it is unknown whether a deterministic polynomial-time computation can “guess” a string of length larger than $\Theta(\log n)$. Note that for any unbounded nondecreasing function $t(n)$, the function $t'(n) = t(n) \log(n)$ is of order $\omega(\log n)$. Thus, the model $GC(t(n) \log n, \Pi_h^B)$ seems to have a stronger ability of guessing than a deterministic polynomial-time computation. Theorem 17.10 basically says that the W -hierarchy does not collapse unless a deterministic polynomial-time computation can guess a string of length larger than $\Theta(\log n)$, which seems unlikely based on our current understanding of nondeterminism.

As a final note for this chapter, we recall Fagin’s characterization of NP introduced in Chapter 4, Section 4.3. NP is the class of all problems that can be put in the form

$$\{I : \exists S \in 2^{[n]^k} \forall x \in [n]^p \exists y \in [n]^q \phi(I, S, x, y)\}.$$

Here, $I \subseteq [n]^m$ is the input relation, x and y are tuples ranging over $[n] = \{1, 2, \dots, n\}$, and ϕ is quantifier-free (Theorem 4.7). Papadimitriou and Yannakakis [368] use this definition as an inspiration for limiting nondeterminism. In [356], in the same spirit as Liming Cai and Chen, they propose the class **LOGNP** as the class of all problems that can be polynomial-time-reducible to ones that can be put in the form

$$\{I : \exists S \in [n]^{\log n} \forall x \in [n]^p \exists y \in [n]^q \forall j \in [\log n] \phi(I, s_j, x, y, J)\},$$

where $S = \{s_1, \dots, s_{\log n}\}$ is an ordered subset of $[n]$ (and, similarly, other classes such as **LOGSNP** obtained by dropping the “ $\exists y \in [n]^p$ ”). They demonstrate that, for instance, the **VC DIMENSION** (of Chapter 11) is **LOGNP**-complete and **LOG DOMINATING SET** is **LOGSNP**-complete. One can view limiting nondeterminism via limiting the scope of the the second-order quantifier in the syntactic definition of **NP** in Fagin’s theorem.

Historical Notes

The results of this section are mainly due to Liming Cai and Chen [108]. Papadimitriou and Yannakakis note that it is open if **LOG CLIQUE** (i.e., is there a clique of size $\log n$?) is **LOGSNP**-complete. Perhaps the answer lies with a collapse of the W -hierarchy. They also suggest that one could approach parametric problems by looking at Fagin’s definition, but limiting the search size of S to a fixed k . Some of these ideas have been taken up by Cai and Chen.

Exercises 17.2

- 17.2.1. (Liming Cai and Chen [105]) Prove that for any nondecreasing function $t(n)$ constructible in polynomial time and for any integer $h > 1$, the following language **CSAT**(t, h) is complete for the class $GC(t(n) \log(n), \Pi_h^B)$ under the polynomial-time reduction.
- 17.2.2. (Papadimitriou and Yannakakis [368]) Prove that the following are **LOGSNP**-complete: **LOG DOMINATING SET** (i.e., is there a dominating set of size $\leq \log n$?) and **TOURNAMENT DOMINATING SET**.
- 17.2.3. (Papadimitriou and Yannakakis [368]) Prove that **LOG CLIQUE** is in **LOGSNP**.

The Monotone and Antimonotone

Collapse Theorems: **MONOTONE**

$$W[2t + 1] = W[2t] \text{ and } \mathbf{ANTIMONOTONE}$$

$$W[2t + 2] = W[2t + 1]$$

In this chapter, we will give proofs of the two structural collapses noted in Chapter 12.3 after Theorem 12.6. The techniques are similar to the proof of the Normalization Theorem.

Theorem 18.1 (Monotone Collapse Theorem) *For all t , **MONOTONE** $W[2t + 1] = W[2t]$.*

Proof. By Theorem 12.6(i)(a) and the definition of weft we have already shown

$$W[2t] \subseteq \mathbf{MONOTONE} \, W[2t] \subseteq \mathbf{MONOTONE} \, W[2t + 1]$$

For the reverse inclusion, we must start again “from the beginning” since the Normalization Theorem, Theorem 12.3, does not preserve monotonicity.

Let $L \in \mathbf{MONOTONE} \, W[2t + 1]$. Let $\mathcal{F}$ denote a family of monotone circuits of depth bounded by h and weft bounded by $2t + 1$ such that L reduces to the **WEIGHTED CIRCUIT SATISFIABILITY** problem $L_{\mathcal{F}}$. We will show that we can reduce this problem to **WEIGHTED CIRCUIT SATISFIABILITY** for nonmonotone weft $2t$ circuits.

The fact that $\mathbf{MONOTONE} \, W[1]$ is fixed-parameter tractable plays a central role in our argument. (See Corollary 18.2 and Exercise 4.) Our algorithm for this problem is based on the method of bounded search trees discussed in Chapter 3. For the moment, consider the special case of **WEIGHTED MONOTONE q -CNF SATISFIABILITY** (cf. Exercise 3). Suppose E is a monotone CNF formula with maximum clause size q for which we wish to determine if there is a weight k satisfying truth assignment. We can easily build a q -branching tree T of height k with a truth assignment of weight k corresponding to each leaf, such that E is

weight k satisfiable if and only if one of these leaf assignments does the job. (The key idea is that for a clause to be satisfied, there are q possibilities, each requiring a variable to be set to 1. The branches of a node in the tree T correspond to these possibilities.) Note that T has at most q^k leaves.

Let (C, k) be an instance of $L_{\mathcal{F}}$. Our proof consists of two parts. The first part transforms C in various ways that consist of replacing or replicating parts of C in monotonicity-preserving ways; the input variables remain the same as does the parameter k and the weft of the circuit. We will term this process *semi-normalization*. This transformation will be useful in proving the other main theorem of this chapter. In the second part of the reduction, the input variables are changed in a way that decreases the weft, but results in a nonmonotone circuit.

Part 1. Semi-normalization.

Step 1.1: Transformation to a tree circuit.

We first transform C into a tree circuit (equivalently represented as a boolean formula) in the manner of Step 1 of the Normalization Theorem (Theorem 12.3).

Step 1.2: Normalization of small gate components.

Viewing the circuit as graph, we are concerned next with the connected components induced by the small gates. Each such component is necessarily a tree subcircuit which has a single (small) output gate. Suppose $C(g)$ is such a component, with output gate g . Because of the bound h on the depth of C , there are at most 2^h inputs to $C(g)$. The function of these inputs computed by $C(g)$ is equivalent to a product-of-sums expression $E_{\wedge}(g)$ having at most 2^{2^h} sums, and to a similar sum-of-products expression $E_{\vee}(g)$. We now consider three cases.

If the output of g goes to a large $\wedge$ gate g' , then replace $C(g)$ in C with $E_{\wedge}(g)$, replicating subcircuits of C as necessary to provide the inputs to the subcircuit corresponding to $E_{\wedge}(g)$. The output $\wedge$ gate of the $E_{\wedge}(g)$ subcircuit can now be coalesced with g' .

If the output of g goes to a large $\vee$ gate g' , then replace $C(g)$ in C with $E_{\vee}(g)$ similarly, and coalesce the output gate of the replacing subcircuit with g' .

If g is the output gate of C , then replace $C(g)$ with $E_{\wedge}(g)$.

If there is a small gate g anywhere in the resulting circuit that gives output to a large gate g' of the same logical character, then g should be coalesced with g' . This coalescing of “small down to large” does not increase the weft of the circuit. (Note that “large down to small” coalescing may increase weft.)

The above transformations result in a tree circuit C' for which every small-gate-connected component (*small* now means fanin bounded by 2^{2^h}) is a single gate, with the possible exception of the component containing the output gate of C' .

Step 1.3: Elimination of internal small gates.

By an *internal small gate* of C' we mean a small gate g that gives its output to a large gate of C' and the arguments to which are either (i) all large gates or (ii) a mixture of large gates and inputs to C' that includes at least one large gate. (Thus, an internal small gate is a small gate that is *not* either at the top of the circuit with inputs that are entirely inputs to C' or part of small gate component that includes the output gate of C' .) Case (ii) can be handled in the same way as

case (i) by considering the inputs to g that are inputs to the circuit C' as the outputs of (“honorary”) large gates having a single input.

Let g' denote a large gate receiving the output of an internal small gate g . There are two cases.

The gate g' is a large $\vee$ gate. Necessarily, g is a $\wedge$ gate, else g and g' would have been coalesced. Note that all of the large $\wedge$ gates g_i giving input to g can be combined by replacing them with a single large $\wedge$ gate on all of their inputs. So we may assume that there is at most one large $\wedge$ gate supplying an argument to g . Let g_1 denote this input to g . The remaining inputs $g_2, \dots, g_s$ to g are large $\vee$ gates. Note that g is small, so $s \leq 2^{2^h}$. For $i = 1, \dots, s$, let m_i denote the number of inputs to g_i and write $E(i, j)$ to denote the j -th input to g_i . The boolean function computed by g can be written

$$E_g = \prod_{j=1}^{m_1} E(1, j) \cdot \prod_{i=2}^s \prod_{l=1}^{m_i} E(i, l).$$

By distributing, this can be rewritten

$$E_g = \sum_{l_2=1}^{m_2} \sum_{l_3=1}^{m_3} \cdots \sum_{l_s=1}^{m_s} \left(E(2, l_2) \cdot E(3, l_3) \cdots E(s, l_s) \cdot \prod_{j=1}^{m_1} E(1, j) \right).$$

Note that this is (potentially) a large sum of large products. We replace g according to this distribution, replicating subcircuits to compute the expressions $E(i, j)$ as needed. This may entail a blowup in the size of C' of $O(n^s)$, which is permitted, since s is small. Weft is preserved by coalescing the new gate g (which is now a large $\vee$ gate) with g' .

The gate g' is a large $\wedge$ gate. Then, necessarily, g is a large $\vee$ gate. In this case, we make the dual replacement.

We now have a monotone tree circuit C'' that is semi-normalized. By trivial modifications such as inserting unary $\wedge$ and $\vee$ gates where appropriate, we may assume that C'' has the following form, where the levels are indexed from 0 to $2t + 4$ and the inputs to the circuit constitute level 0:

- (1) Level 1 consists of small gates, each of opposite character from the large gate on level 2 to which it gives its output.
- (2) On any input-output path through the circuit, the gates from levels 2 to $2t + 2$ alternate in logical character and are all large. (Note that there are two possible alternation patterns: beginning with $\wedge$ or beginning with $\vee$.)
- (3) The lowest 2 levels implement an all-small product-of-sums expression.

Let s denote the bound on small gate arity, and let $w = 2t + 1$. We can describe C'' by the following canonical expression E . This will be our starting point for the second part of the reduction:

$$E = \prod_{i=1}^s \left(\sum_{j=1}^{s_1} E_1(i, j) + \sum_{j=1}^{s_2} E_2(i, j) \right),$$

where for $i = 1, \dots, s$ and $j = 1, \dots, s_1$, the subexpressions $E_1(i, j)$ have the form

$$E_1(i, j) = \prod_{i_1=1}^{m_1} \sum_{i_2=1}^{m_2} \cdots \prod_{i_w=1}^{m_w} \sum_{l=1}^s v(1, i, j, i_1, \dots, i_w, l),$$

and for $i = 1, \dots, s$ and $j = 1, \dots, s_2$, the subexpressions $E_2(i, j)$ have the form

$$E_2(i, j) = \sum_{i_1=1}^{n_1} \prod_{i_2=1}^{n_2} \cdots \sum_{i_w=1}^{n_w} \prod_{l=1}^s v(2, i, j, i_1, \dots, i_w, l),$$

and the $v(\dots)$ are input variables of the circuit. Note that the weft of the circuit corresponding to E is $w = 2t + 1$. We will refer to the subexpressions $E_1(i, j)$ as *type 1*, and the subexpressions $E_2(i, j)$ as *type 2*.

Part 2. Decreasing the weft.

Step 2.1: Replacement of the top small gates of type 1 subexpressions.

We next make use of the fact that **MONOTONE q -CNF SATISFIABILITY** is fixed-parameter tractable to replace the tops of the type 1 subexpressions. Let g be a gate at level 2 in a subcircuit corresponding to one of the subexpressions $E_1(i, j)$. The output of g is described by a monotone product-of-sums expression having maximum clause size s . This is equivalent to a logical $\vee$ of at most s^k products of size k that can be computed by the *FPT* algorithm for the bounded clause size satisfiability problem. In this manner, we replace each top large $\wedge$ gate for each subexpression $E_1(i, j)$. Since the new gate g gives its output to a large $\vee$ gate g' at level 3, we can coalesce g and g' . Note that this “almost” decreases the weft relative to these subexpressions; the hitch is that the top gates now have k inputs, and k is not small, although we are making some sort of progress. The type 1 subexpressions now have the form

$$E_1(i, j) = \prod_{i_1=1}^{m_1} \sum_{i_2=1}^{m_2} \cdots \sum_{i_{w-1}=1}^{m_{w-1}} \prod_{i_w=1}^k u(1, i, j, i_1, \dots, i_w).$$

Step 2.2: Elimination of the topmost conjuncts.

Our strategy here is similar to Step 7 in the proof of the Normalization Theorem. The main idea is to introduce new variables, each of which represents a predicate indicating that every variable in a set of original variables is 1. Let $x(i)$ for $i = 1, \dots, n$ denote the input variables for the circuit we are now considering (received from Step 2.1). Let $A(i, j, i_1, \dots, i_{w-1})$ denote the set of variables that are input to the (appropriately indexed) topmost $\wedge$ gate of a type 1 subcircuit:

$$A(1, i, j, i_1, \dots, i_{w-1}) = \bigcup_{i_w=1}^k \{u(1, i, j, i_1, \dots, i_{w-1}, i_w)\} \subseteq \{x(1), \dots, x(n)\}.$$

Similarly for the type 2 subcircuits, let

$$A(2, i, j, i_1, \dots, i_w) = \bigcup_{l=1}^s \{v(2, i, j, i_1, \dots, i_w, l)\}.$$

We will refer to these as the *macro sets* of the variables of E . We may suppose without loss of generality that $k \geq s$, since a topmost $\wedge$ gate with $s > k$ inputs will necessarily evaluate to false. Let $\mathcal{A}$ denote the family of distinct macro sets

$$\mathcal{A} = \bigcup_{i=1}^s \bigcup_{j=1}^{s_1} \bigcup_{i_1=1}^{m_1} \cdots \bigcup_{i_{w-1}=1}^{m_{w-1}} \{A(i, j, i_1, \dots, i_{w-1})\}.$$

Each $A \in \mathcal{A}$ represents the argument set of a topmost conjunct.

Our construction is based on an augmentation $\mathcal{A}'$ of $\mathcal{A}$.

Let A denote an arbitrary finite set. Say that a tower of sets

$$A_1 \subset A_2 \subset \cdots \subset A_s$$

is a *trail* for A if

- (i) $|A_1| = 1$,
- (ii) $A_s = A$, and
- (iii) for $i = 1, \dots, s$, $|A_i - A_{i-1}| = 1$.

For each $A \in \mathcal{A}$, choose (arbitrarily) a trail for A . Note that the length of each such trail will be at most k . Let $\Omega(A)$ denote the family of sets in the trail for A , that is,

$$\Omega(A) = \{A_1, \dots, A_s\}.$$

We define $\mathcal{A}'$ to be the union of the trails:

$$\mathcal{A}' = \bigcup_{A \in \mathcal{A}} \Omega(A).$$

We now introduce new variables of the following kinds:

- (1) One new variable $y(A)$ for each distinct macro set $A \in \mathcal{A}'$.
- (2) For each original variable $x(i)$, we introduce $k2^{k+1}$ new variable “copies”

$$x(i, 0), \dots, x(i, k2^{k+1} - 1).$$

- (3) “Padding” consisting of $k2^k$ meaningless variables (inputs not supplying arguments to any gates) $z(1), \dots, z(k2^k)$.

The change of variables requires some enforcement that can be expressed as a weft 1 subcircuit, detailed below. The expression E for the circuit is transformed to the following expression E' over the set of new variables. We will argue below that E' has a satisfying truth assignment of weight $k' = k2^{2k+1} + k2^k$ if and only if E has a satisfying truth assignment of weight k .

$$E' = \prod_{i=1}^s \left(\sum_{j=1}^{s_1} E'_1(i, j) + \sum_{j=1}^{s_2} E'_2(i, j) \right) \cdot E'_3,$$

where

$$E'_1(i, j) = \prod_{i_1=1}^{m_1} \sum_{i_2=1}^{m_2} \cdots \sum_{i_{w-1}=1}^{m_{w-1}} y(A(1, i, j, i_1, \dots, i_{w-1})),$$

$$E'_2(i, j) = \sum_{i_1=1}^{n_1} \prod_{i_2=1}^{n_2} \cdots \sum_{i_w=1}^{n_w} y(A(2, i, j, i_1, \dots, i_w)),$$

and E'_3 is a product of small sums expression recording the following “enforcement” implications concerning the new variables:

(1) For $i = 1, \dots, n$ and $r = 0, \dots, k2^{k+1} - 1$,

$$x(i, r) \Rightarrow x(i, r + 1 \pmod{k2^{k+1}}).$$

(2) For each containment $A \subseteq A'$ for $A, A' \in \mathcal{A}'$,

$$y(A') \Rightarrow y(A).$$

(3) For each membership $x(i) \in A$ for $A \in \mathcal{A}'$,

$$y(A) \Rightarrow x(i, 0).$$

(4) For each $A \in \mathcal{A}$ and for each $A_i \in \Omega(A)$,

$$(y(A_i) \wedge x(i)) \Rightarrow y(A_{i+1}),$$

where $x(i) = A_{i+1} - A_i$.

E' may be larger than E by a linear factor exponential in k . Correctness for the transformation can be argued as follows.

If E is satisfied by a weight k truth assignment τ , then a weight k' satisfying truth assignment τ' for E' can be had by the following:

Setting $\tau'(x(i, 0)) = \tau'(x(i, 1)) = \dots = \tau'(x(i, k2^{k+1} - 1)) = 1$ for any i for which $\tau(x(i)) = 1$.

Setting $\tau'(y(A)) = 1$ for any set $A \in \mathcal{A}'$ for which $\tau(x(i)) = 1$ for all $x(i) \in A$.

Setting $\tau'(z(i)) = 1$ for some number of the padding variables, in order to bring the weight of τ' to exactly k' . (The reason for this is that not all of the subsets of the k variables set to 1 by τ may occur in $\mathcal{A}$.)

Setting everything else to 0.

Conversely, suppose E' is satisfied by a weight k' truth assignment σ' . Because the implications of (1) are satisfied, exactly k sets of copies of the original variables must have value 1 for σ' . Suppose the indexes of these sets of copies are $l_1, \dots, l_k$ and let $K = \{x(l_1), \dots, x(l_k)\}$. We will argue that the truth assignment σ that sets the variables of K to 1 and all else to 0 is a satisfying truth assignment for E .

The implications of (4) ensure that $\sigma'(y(A)) = 1$ for any $A \subseteq K$, $A \in \mathcal{A}$.

The implications of (2), (3), and (4) together imply that the settings of the macro variables by σ' must be exactly consistent with the values of the sets of copies of the original variables. By the construction of E' , we see that σ is a weight k satisfying truth assignment for E .

We are definitely making progress. Note that E' would now have weft $w - 1 = 2t$ except for the subexpressions of type 2. The main idea of our next step is (roughly) to remove the bottom most large $\vee$ gates of the type 2 subexpressions by introducing additional nondeterminism.

Step 2.3: *More nondeterminism, less weft.*

Let C' now denote the tree circuit corresponding to the expression E' received from Step 2.2. The bottom of the circuit is a small $\wedge$ of small $\vee$ gates. The arguments for these small $\vee$ gates are supplied from large $\wedge$ gates in the case of type

1 subexpressions, and large $\vee$ gates in the case of type 2 subexpressions. Furthermore, all of the small gates are at the bottom of the circuit. We begin by coalescing the small $\vee$ gates with the bottommost large $\vee$ gates of the type 2 subexpressions.

After renaming and reindexing, the current form of the circuit can be described by the weft $2t + 1$ expression

$$F = \prod_{i_1=1}^{s_1} \sum_{i_2=1}^{n_2} \prod_{i_3=1}^{n_3} F(i_1, i_2, i_3),$$

where s_1 is a constant independent of k and n . Let k_F denote the relevant weight parameter; that is, we have completed a reduction to the situation where we are concerned with whether F has a truth assignment of weight k_F , where k_F is purely a function of k .

We next introduce new variables $v(i_1, i_2)$ for $i_1 = 1, \dots, s_1$ and $i_2 = 1, \dots, n_2$ to indicate how subexpressions of F are satisfied. Let

$$k'_F = k_F + s_1$$

and consider

$$F' = F'_1 \cdot F'_2 \cdot F'_3,$$

where

$$\begin{aligned} F'_1 &= \prod_{i_1=1}^{s_1} \sum_{i_2=1}^{n_2} v(i_1, i_2), \\ F'_2 &= \prod_{i_1=1}^{s_1} \prod_{1 \leq j < j' \leq n_2} (\neg v(i_1, j) + \neg v(i_1, j')), \\ F'_3 &= \prod_{i_1=1}^{s_1} \prod_{i_2=1}^{n_2} \prod_{i_3=1}^{n_3} (\neg v(i_1, i_2) + F(i_1, i_2, i_3)). \end{aligned}$$

The weft of F' is $2t$. We claim that F' has a truth assignment of weight k'_F if and only if F has a truth assignment of weight k_F .

A truth assignment of weight k'_F that satisfies F' must set exactly s_1 of the new variables $v(i_1, i_2)$ to 1, by the restrictions imposed by F'_1 and F'_2 . Consequently, exactly k_F of the original variables of F must also be set to 1, in order for the total weight of the assignment to be k'_F . Moreover, these must be one variable for each of the s_1 branches of F . Since F'_3 is satisfied, F is satisfied by this weight k_F assignment. In the other direction, the conversion of a weight k_F satisfying assignment for F into a weight k'_F satisfying assignment for F' follows the same correspondence. This completes the theorem.

Corollary 18.2 *MONOTONE $W[1]$ is fixed-parameter tractable.*

Proof. (See also Exercise 4.) In the proof of the theorem we have used the fact that the special case of **WEIGHTED MONOTONE q -CNF SATISFIABILITY** is fixed-parameter tractable. The steps of the proof through Step 2.1 all preserve both

monotonicity and the original input variables, and consist only in rewriting the boolean function computed by the circuit. Thus, it is enough to argue that it is fixed-parameter tractable to determine if an expression in the form resulting from Step 2.1 has a weight k truth assignment. This form is a monotone product-of-sums-of- k -products

$$E = \prod_{i=1}^s \sum_{j=1}^m \prod_{l=1}^k x(i, j, l),$$

where s is a constant independent of k and the size of the expression.

We can simply try all m^s possibilities for choosing branches of E to be satisfied.

Theorem 18.3 (Antimonotone Collapse Theorem) *For all t , $ANTIMONOTONE$ $W[2t + 2] = W[2t + 1]$.*

Proof. Half of the theorem is easy. For the other half, we may make use of the same initial transformations to semi-normalized form as in the first part of the proof of Theorem 18.1, since these are based on reorganizations of the circuit computations that have no effect on monotonicity or antimonotonicity. After these transformations (and some trivial replications of parts of the circuit to simplify the indexing), our starting point is a circuit C of size n corresponding to an expression E in the form

$$E = \prod_{i=1}^s \left(\sum_{j=1}^s E_1(i, j) + \sum_{i=1}^s E_2(i, j) \right),$$

where s is a constant independent of k and n , and where the subexpressions have the form

$$E_1(i, j) = \prod_{i_1=1}^m \sum_{i_2=1}^m \cdots \sum_{i_w=1}^m \prod_{l=1}^s \neg v(1, i, j, i_1, \dots, i_w, l),$$

$$E_2(i, j) = \sum_{i_1=1}^m \prod_{i_2=1}^m \cdots \prod_{i_w=1}^m \sum_{l=1}^s \neg v(2, i, j, i_1, \dots, i_w, l),$$

with the $v(\cdot \cdot \cdot)$ being input variables of the circuit and $w = 2t + 2$. We wish to determine whether there is a weight k input accepted by C . The remaining steps of the transformation parallel the steps of Part 2 of the proof of Theorem 18.1, although there are a few differences.

Step 2.1: *Replacement of the top small gates of type 1 subexpressions.*

The top three levels of a type 1 subexpression have the form

$$F = \prod_{i=1}^m \sum_{j=1}^m \prod_{l=1}^s \neg x(i, j, l).$$

We have the logical equivalences

$$F = \neg \neg F = \prod_{i=1}^m \left(\neg \prod_{j=1}^m \sum_{l=1}^s x(i, j, l) \right).$$

The top two levels of the above expression are monotone products-of-sums, so as in the proof of Theorem 18.1, we can write

$$F = \prod_{i=1}^m \left(\neg \sum_{j=1}^{s^k} \prod_{l=1}^k y(i, j, l) \right),$$

where the $y(\dots)$ are appropriate circuit inputs. Commuting the negations to the top of the circuit gives

$$F = \prod_{i=1}^m \prod_{j=1}^{s^k} \sum_{l=1}^k \neg y(i, j, l).$$

By coalescing the $\wedge$ gates corresponding to the products, we have “almost” decreased the weft of these subexpressions by 1, except that the topmost sums have size k , which is not small.

Step 2.2: *Elimination of the topmost disjuncts.*

We now have an antimonotone circuit, the first two levels of which are various products-of-sums of negated inputs, with the sums having size k in the case of the type 1 subexpressions and size s in the case of the type 2 subexpressions. This is the dual of our situation for Step 2.2 of the proof of Theorem 18.1.

Note that if $v(a, b, c)$ is a new variable that represents the predicate $a = b = c = 1$, then we have the logical equivalence

$$(\neg a) \vee (\neg b) \vee (\neg c) = \neg v(a, b, c).$$

This provides the basis for our transformation. A complication arises because we are unable to argue for well-behavedness as we did for Theorem 18.1, so that our argument must be modified slightly.

As in Step 2.2 of the proof of Theorem 18.1, let $\mathcal{A}$ denote the family of sets of variables that, negated, are the inputs to the topmost $\vee$ gates. For each $A \in \mathcal{A}$, choose a trail $\Omega(A)$ for A and let $\mathcal{A}'$ be the union of the trails.

We introduce new variables of the following kinds:

- (1) One new variable $y(A)$ for each $A \in \mathcal{A}'$.
- (2) For each original variable $x(i)$, the copy variables $x(i, 0), \dots, x(i, k2^{k+1} - 1)$.
- (3) Padding variables $z(1), \dots, z(k2^k)$.

Each top level disjunct of the negations of the variables in a set $A \in \mathcal{A}$ is replaced by $\neg y(A)$; this is essentially the dual of the substitution made in Step 2.2 of Theorem 18.1.

The remainder of the details of this step are left to the reader, as from this point on they are essentially identical to the previous argument.

Step 2.3: *More nondeterminism, less weft.*

Because of the weft 1 enforcements introduced in Step 2.2, we are no longer dealing with an antimonotone circuit, but the same was true in the proof of Theorem 18.1. In fact, we are now in essentially the same position and can apply *the same* transformation as in the earlier argument, with one complication.

The problem is that this works if $t \geq 1$, but not for $t = 0$, since the enforcement expressions for the change of variables have weight 2. As is so often the case, we are forced into making a special argument to fit our conclusions to weight 1.

For the case of $t = 0$, we may take our current starting point to be described by the weight 2 expression

$$F = \prod_{i=1}^s \sum_{j=1}^m \prod_{l=1}^m \lambda(i, j, l),$$

where $\lambda(i, j, l)$ is a literal, that is, either an input variable or the negation of an input variable. Assume that we are interested in whether there is a weight k_F truth assignment for F , and note that s is small. The basic idea of our transformation for this case is the same; we are simply forced to do a little more work to obtain weight 1 enforcement for our change of variables.

Let $x(1), \dots, x(n)$ denote the variables of F . We introduce new variables $V = V_1 \cup V_2$, where

$$\begin{aligned} V_1 &= \{v(i, j) : 1 \leq i \leq s, 1 \leq j \leq m\}, \\ V_2 &= \{r(i, j) : 1 \leq i \leq n, 0 \leq j \leq s-1\}. \end{aligned}$$

The transformed expression is $F' = F'_1 \cdot F'_2 \cdot F'_3 \cdot F'_4$, where

$$\begin{aligned} F'_1 &= \prod_{i=1}^s \prod_{j=1}^m \prod_{l=1}^m (\neg v(i, j) + \lambda(i, j, l)), \\ F'_2 &= \prod_{i=1}^s \prod_{1 \leq j < j' \leq m} (\neg v(i, j) + \neg v(i, j')), \\ F'_3 &= \prod_{i=1}^n (\neg x(i) + r(i, 0)) \cdot (x(i) + \neg r(i, 0)), \\ F'_4 &= \prod_{i=1}^n \prod_{j=0}^{s-1} (\neg r(i, j) + r(i, j+1 \pmod{s})). \end{aligned}$$

The transformed weight parameter is

$$k'_F = s + k_F(s+1).$$

Suppose τ is a weight k_F truth assignment that satisfies F . Then, for each i , $1 \leq i \leq s$, we can identify an argument with index j to the i -th large $\vee$ gate that evaluates to 1. Correspondingly, set $\tau'(v(i, j)) = 1$. Let τ' correspond to τ on $\{x(1), \dots, x(n)\}$, and for each i such that $\tau(x(i)) = 1$, let $\tau'(r(i, j)) = 1$ for $j = 0, \dots, s-1$. Then, τ' is a truth assignment to the variables of F' of weight k'_F and it is easy to check that τ' satisfies F' .

Conversely, suppose σ' is a truth assignment of weight k'_F that satisfies F' . Because of the implications of F'_3 and F'_4 , the number p of original variables $x(i)$ and variables of V_2 set to 1 by σ' must be a multiple of $s+1$, and if $p = q(s+1)$,

then precisely q of the variables $x(i)$ are set to 1. The definition of k'_F forces $p = k_F$. Thus, the restriction σ of σ' to the variables $x(i)$ constitutes a weight k_F truth assignment for the variables of F . We argue that σ satisfies F . Since σ' has weight $k_F(s + 1)$ on the original variables together with V_2 , it must have weight s on V_1 . The expression F'_2 ensures that for each i , $1 \leq i \leq s$, there is exactly one j such that $\sigma'(v(i, j)) = 1$. It is easy to see that these $v(i, j)$ indicate how F is satisfied by σ .

Since F' is an expression of weight 1, we are done.

19

The Structure of Languages Under Parameterized Reducibilities

19.1 Some Tools

In this rather technical chapter, we will develop techniques which will enable us to have some insight into the structure of the recursive languages under the various reducibilities. The techniques and results of this chapter have a strongly recursion-theoretic flavor and require some rather sophisticated techniques from classical recursion theory. We will assume that the reader is noddingly familiar with the basic notions from classical recursion theory, such as Kleene's Arithmetical Hierarchy and the like, but not familiar with priority arguments.

Briefly, we remind the reader that a language is called recursively enumerable if it is empty or the range of a recursive function, and the language is called recursive if both it and its complement are recursively enumerable. Of course, recursive languages correspond to decidable problems, and recursively enumerable ones correspond to "semidecidable" problems such as the halting problem. After Kleene we know that a language L is recursively enumerable iff it is Σ_1 ; that is, there is a recursive relation R such that for all x ,

$$x \in L \text{ iff } \exists y R(x, y).$$

A language is called Π_n if it is the complement of a Σ_n language, and, hence, it is of the form $x \in L \text{ iff } \forall y R(x, y)$. More generally, we can define languages to be Σ_n (resp. Π_n) iff there is a recursive relation R and n alternating quantifiers $Q_1, \dots, Q_n$ with Q_1 existential (universal) such that

$$x \in L \text{ iff } Q_1 x_1 Q_2 x_2 \cdots Q_n x_n R(x, x_1, \dots, x_n).$$

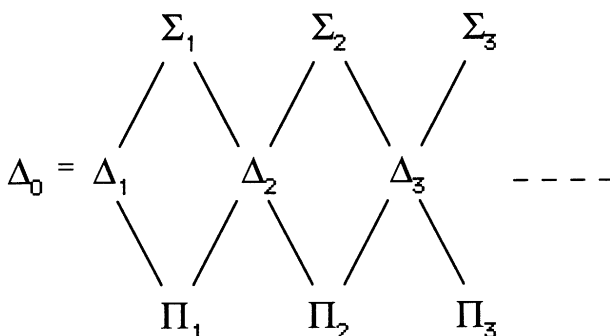


FIGURE 19.1. The Arithmetical Hierarchy

Finally, a language is called Δ_n iff it is both Σ_n and Π_n . All of this leads to Kleene's arithmetical hierarchy in Figure 19.1.

Here, inclusions are from left to right, and via relativizations of the halting problem, all of the inclusions are proper. This is all connected with computation via the following. Let $\{\Phi_e : e \in \mathbb{N}\}$ list all Turing procedures. For any set A , we can define the *jump* of A to be $A' = \{e : \Phi_e(A; e) \downarrow\}$. The halting problem is coded by $\emptyset'$. The proof of the halting problem shows that for any set A , $A <_T A'$. The "spine" of the arithmetical hierarchy is provided by the languages $\emptyset', \emptyset'' [= (\emptyset')']$, $\emptyset', \dots$. The *degree* of a language is the equivalence class of the language under Turing reducibility. Hence, the degree of A would be $\{B : B \equiv_T A\}$. We denote the degree of the recursive languages by $\mathbf{0}$, the degree of $\emptyset'$ by $\mathbf{0}'$, etc. The principal fact we need is the following:

Lemma 19.1 *A language L is Δ_{n+1} iff $L \leq_T \mathbf{0}^{(n)}$.*

Thus, there is a natural connection between syntactic descriptions of a language and their computational complexity. For our purposes, we will also need the following.

Lemma 19.2 (Shoenfield Limit Lemma) *A language L is Δ_2 iff there is a $\{0, 1\}$ -valued recursive function f such that:*

- (i) *for all x , $\lim_s f(x, s) =_{\text{def}} g(x)$ exists (i.e., $|\{s : f(x, s) \neq f(x, s+1)\}|$ is finite) and,*
- (ii) *$g(x) = L(x)$, where we identify languages with their characteristic functions [meaning that $g(x) = 1$ iff $x \in L$].*

Proof. The proof is straightforward but instructive. Suppose L is Δ_2 . Thus, there is a procedure Ψ reducing L to $\emptyset'$. Thus, for all x , $x \in L$ iff $\Psi(\emptyset'; x) = 1$. We approximate $\Psi(\emptyset'; x)$. Let $\emptyset'_s = \{e : e \leq s \wedge \Phi_{e,s}(\emptyset; e) \downarrow\}$. Define $f(x, s) = 1$ if $\Psi_s(\emptyset'_s; x) \downarrow = 1$, and $f(x, s) = 0$ otherwise. Then, clearly, f so construed does the job, since the amount of information used in a computation is finite. Specifically, we know that for all x , $\Psi(\emptyset'; x) = L(x)$. Now there is a largest element of $\emptyset'$ queried in the $\Psi(\emptyset'; x)$ computation. Call this $u(x)$. At some stage s for all $t \geq s$

and all $u' \leq u(x)$,

$$u' \in \emptyset' \text{ iff } u' \in \emptyset'_t.$$

Furthermore, for some $s_1 \geq s$, we know that

$$\Psi_{s_1}(\emptyset'_{s_1}; x) \downarrow = L(x).$$

But then for all $t \geq s_1$, it must be that

$$\Psi_t(\emptyset'; x) = L(x) = \Psi(\emptyset'; x).$$

Note that as f changes whenever Ψ does, we also know that $f(x, t) = L(x)$.

For the reverse direction, suppose f satisfying (i) and (ii) exists. We define a procedure Γ and a recursively enumerable language M so that $\Gamma(M) = L$. Since M is recursively enumerable, it will then follow that $M \leq_T \emptyset'$ and hence $L \in \Delta_2$. We define M and Γ in stages.

Stage 0. Define $M_0 = \emptyset$ and $\Gamma_0(M_0; x) = f(x, 0)$ and define $\gamma(x, 0) = \langle x, 0 \rangle$ for all $x \leq 0$.

Stage $s + 1$. Define $\gamma(s + 1, y) = \langle s + 1, s + 1 \rangle$ for all $y \leq s + 1$. Now we consider only $x \leq s$. If $f(x, s) = f(x, s + 1)$, change nothing for x ; that is, set $\Gamma(M_{s+1}; x) = \Gamma(M_s; x)$ via the same computation, and keep $\gamma(x, s + 1) = \gamma(x, s)$.

If $f(x, s + 1) \neq f(x, s)$, enumerate $\gamma(x, s)$ into M_{s+1} allowing us the change $\Gamma_{s+1}(M_{s+1}; x)$ to be $f(x, s + 1)$. Set $\gamma(x, s + 1) = \langle x, s + 1 \rangle$.

It is not difficult to see that $\lim_s \gamma(x, s)$ exists and that, hence, Γ describes a procedure reducing L to M .

The proof of the above result has much in common with many proofs in classical recursion theory. One performs a *construction* where some object is built in *stages*. Typically, one has some overall *goal* that one breaks down into smaller subgoals for which it is argued that they are all eventually met in the limit. As an archetype for such proofs, think of Cantor's proof that the collection of all infinite binary sequences is uncountable. One can conceive of the proof as follows.

Suppose we could list the sequences as $\mathcal{S} = \{S_0, S_1, \dots\}$ with $S_e = s_{e,0}s_{e,1}\dots$. It is *our* job to construct a binary sequence $U = u_0u_1\dots$ that is not on the list $\mathcal{S}$. This should be thought of as a game against our opponent who must supply us with $\mathcal{S}$. We shall construct u in stages, at stage t specifying only $u_0 \dots u_t$ the initial segment of U of length $t + 1$. Our *requirements* are the decomposition of the overall goal into subgoals of the form

$$R_e : U \neq S_e,$$

one for each $e \in \mathbb{N}$. Of course, we know how to satisfy these requirements. At stage e , we simply make $u_e \neq s_{e,e}$ by setting $u_e = 1$ or 0, and making $u_e = 1$ iff $s_{e,e} = 0$. Hence, for all e , $U \neq S_e$; all the requirements are met. This is a contradiction to the fact that $\mathcal{S}$ supposedly lists all infinite binary sequences, as U is a binary sequence.

Notice that if we define a real to be *recursive* if it has an infinite recursive binary (or decimal) expansion, then the above proof shows that there is no recursive enumeration of all the recursive reals.

Our proofs and constructions will be rather more complex than this easy result, but readers should keep the overall *structure* of the above in their minds when looking at the results to follow. Our construction will be in finite steps where an object is constructed stage by stage in finite pieces. The object will be constructed to satisfy a list of requirements $\{R_e; e \in \mathbb{N}\}$. In our constructions, we will need one further obvious but important result. We need the following definition.

Definition 19.3 (Use) In a computation $\Phi(A; x)$ from an oracle A , we define the *use of the computation upon input x* to be the collection of strings z queried of A during the course of the computation. (If the computation does not halt, then we take the use to be undefined.) We denote the use of the computation to be $u(\Phi(A; x))$.^a

^aThis definition of *use* is slightly nonstandard in the sense that in classical recursion theory, the use of a computation is usually taken to be the largest element queried during the computation. The reader may decide to adopt this definition without much loss of readability.

Lemma 19.4 (Use Principle) *Suppose that A and B are languages and $\Phi(A; x) \downarrow$. Suppose that for all strings $z \in u(\Phi(A; x))$, $A(z) = B(z)$. Then, $\Phi(B; x) = \Phi(A; x)$ via identical computations.*

The proof is obvious. Since the languages are the same, that is, $A(z) = B(z)$, for all queried strings z , they give the same answer, and hence the computations will be the same. Why is the use principle so important? The point is that often we wish to construct objects to beat certain oracle computations. If we “preserve the use” of the said computations, we will preserve the computation. But the use is only a finite portion and we will still have almost all the rest of the language to meet the other requirements. Here is a famous example using the “finite extension method,” which is a primitive form of Cohen Forcing, and is the mainstay of much of traditional structural complexity theory.

Theorem 19.5 (Kleene and Post [300]) *There exist degrees $\mathbf{a}$ and $\mathbf{b}$ both below $0'$ and such that $\mathbf{a} \upharpoonright_T \mathbf{b}$; that is, they are incomparable under Turing reductions.*

Proof. We construct $A = \lim_s A_s$ and $B = \lim_s B_s$ in stages, to meet the requirements below for all $e \in \mathbb{N}$:

$$\begin{aligned} R_{2e} : \Phi_e(A) &\neq B, \\ R_{2e+1} : \Phi_e(B) &\neq A. \end{aligned}$$

Note that if we so construe A and B , then $A \not\leq_T B$ since we meet all the R_{2e+1} 's, and $B \not\leq_T A$ since we meet all the R_{2e} 's. Hence, A and B will have incomparable

T degrees. The fact that $A, B \leq_T \emptyset'$ will come from the construction and will be observed at the end.

The argument is a finite extension in the sense that at each stage s , we specify a finite portion of A , namely A_s , and a finite portion B_s of B . They will be specified for all strings of length $\leq t_s$. *The key invariant is that for all stages $u \geq s$, for all z if $|z| \leq t_s$, then $A(z) = A_s(z) = A_u(z)$, and $B(z) = B_s(z) = B_u(z)$.* Thus, in a finite extension argument, after stage s , we can only *extend* the portion of A (B) we have specified so far. [Here we abuse notation by saying that a *set of strings* C extends another D ($C > D$) iff for all z with $|z| \leq \max\{|z'| : D(z') \text{ defined}\}$, $C(z) = D(z)$; that is, the characteristic function of C extends that of D .] To put it another way, we cannot at a later stage change the sets on anything we have specified so far.

Construction.

Stage 0. Set $A_0 = B_0 = \lambda$ (the empty string). Set $t_0 = 0$.

Stage $2e+1$. (Attend R_{2e}) We will have specified A_{2e} , B_{2e} , and t_{2e} , at stage $2e$. Pick some string x , called a *witness*, of length $> t_{2e}$, and see if there is a string τ extending A_{2e} such that

$$\Phi_e(\tau; x) \downarrow.$$

If such a τ exists, choose the first such τ and set $A_{2e+1} = \tau$ [i.e., for all z with $|z| \leq |\tau|$, let $A(z) = \tau(z)$]. For all q with $|q| \leq |x|$, set $B_{2e+1}(q)$ to be the following:

- (i) $B_{2e}(q)$ if $|q| \leq |B_{2e}|$,
- (ii) $1 - \Phi_e(\tau; x)$ if $q = x$,
- (iii) 0, otherwise.

Finally, set t_{2e+1} to be so long that it exceeds t_{2e} , $|\tau|$, and $|x|$.

If no such τ exists, then set $A_{2e+1} = A_{2e}$, $B_{2e+1} = B_{2e}$, and $t_{2e+1} = |x|$ (which is $> t_{2e} + 1$).

Stage $2e+2$. (Attend R_{2e+1}) Proceed as we did in stage $2e+1$ except with the roles of A and B reversed.

End of Construction.

Verification. To verify the construction, we prove that we meet R_j for all j , and in fact, we meet R_j at stage $j+1$. This is proven by induction on j . First, note that for all n , $t_{n+1} > t_n$. We suppose that for an induction, we have met R_j for all $j < n$ by stage n , and the parameter t_n is so large that it will protect all the computations for $j < n$. Without loss of generality, $n = 2e$. Now at stage $n+1$, there are two cases to consider. If there is a τ extending A_n with $\Phi_e(\tau, x) \downarrow$, our action is to adopt τ as our next A_{n+1} and cause $\Phi_e(A_{n+1}; x) \neq B_{n+1}(x)$ by (ii). We will then set t_{n+1} to be large enough that for all $n' \geq n$, for all z with $z \in u(\Phi(A_{n+1}; x))$, $A_{n'}(z) = A_n(z)$, and hence $\Phi(A_{n+1}; x) = \Phi(A; x) \neq B_{n+1}(x) = B(x)$. The other case is that no such τ exists. Then, since A is an extension of A_n , it can only be that $\Phi(A; x) \uparrow$, and, hence, in either case, we meet R_n .

Finally, we argue that $A, B \leq_T \emptyset'$. Notice that the construction is, in fact, fully recursive except for the decision as to which case we are in at stage n . There, we must decide if there is some τ with a convergent computation. For instance, at

stage $2e + 1$, we must decide

$$\exists \tau, s[\tau > A_{2e} \wedge \Phi_{e,s}(\tau; x) \downarrow].$$

This is a Σ_1 question and can be decided by $\mathcal{O}'$. [Specifically, use the s-m-n theorem to construct a recursive function $\varphi_{s(2e)}$ such that for all z , $\varphi_{s(2e)}(z) = 1$ if $\exists \tau, s[\tau > A_{2e} \wedge \Phi_{e,s}(\tau; x) \downarrow]$, and $\varphi_{s(2e)}(z) \uparrow$ otherwise. Then $\exists \tau, s[\tau > A_{2e} \wedge \Phi_{e,s}(\tau; x) \downarrow]$ iff $s(2e) \in \mathcal{O}'$.]

Remark. We remark that the reasoning at the end of the above proof is quite common: $\mathcal{O}'$ can answer any Δ_2 and hence any Σ_1 and Π_1 question.

Turning now to classical complexity theory, a famous application of the finite extension method is due to Baker et al. [48]. We need a technical tool.

Notation. [Listing Polynomial Time Procedures] Notice that there is an effective listing of all *polynomial*-time Turing procedures obtained by listing $\Gamma_e, : e \in \mathbb{N}$ with Γ_e a version of Φ_e running in time $|x|^e + e$; that is, for $s \leq |x|^e + e$, we define $\Gamma_{e,s}(D; x) = \Phi_{e,s}(D; x)$, and for $s > |x|^e + e$, we let $\Gamma_{e,s}(D; x) = \Phi_e(D; x)$ only if $\Phi_{e,t}(D; x) \downarrow$, for some $t \leq |x|^e + e$; and otherwise, we let $\Gamma_{e,s}(D; x) = \Gamma_{e,|x|^e+e}(D; x) = 0$. Notice that Γ_e has the property that for all e , for all x and oracles D ,

$$\Gamma_e(D; x) \downarrow \text{ in } \leq |x|^e + e \text{ steps.}$$

Theorem 19.6 ([48]) *There exists a recursive set A such that $NP^A \neq P^A$.*

Proof. We construct A via the finite extension method in stages. We also construct an auxiliary set B defined via

$$x \in B \text{ iff } \exists y[|y| = |x| \wedge y \in A].$$

Clearly, $B \in NP^A$. Let Γ_e for $e \in \mathbb{N}$, list all polynomial-time procedures as above. We meet the requirements

$$R_e : \Gamma_e(A) \neq B.$$

Construction.

Stage 0. Define $A_0 = \lambda$ and $t_0 = 0$.

Stage $e+1$ (Attend R_e) We have specified A_e and t_e . Find some $n = n(e)$ sufficiently large so that $2^n > n^e + e$ and $n > t_e$. Choose $x = x(e) = 0^n$ as a *witness* for R_e . Let A'_e denote the *empty* extension of A_e meaning that $A'_e(x) = A_e(x)$ for x with $|x| \leq t_e$, and $A'_e(x) = 0$ otherwise. There are two cases.

Case 1. $\Gamma_e(A'_e; x) = 1$.

Action. Choose t_{e+1} to be sufficiently large that it exceeds T_e and n and the length of all strings in $u(\Gamma_e(A'_e; x))$. Now, define $A_{e+1} = A'_e$ for all strings of length $\leq t_{e+1}$.

Case 2. $\Gamma_e(A'_e; x) = 0$.

Action. Now, since $|x| = n$ and $2^n > n^e + e$, there is some string z of length n not

in $u(\Gamma_e(A'_e; x))$ that is not addressed during the course of the computation. Our action is to choose t_{e+1} large, as in Case 1, and then set $A_{e+1}(y) = A'_e(y)$ for all strings of length $\leq t_{e+1}$ *except* for z . We put z into A_{e+1} ; that is $A_{e+1}(z) = 1$.

End of Construction.

Verification. Again, we work by induction on e . So at Stage $e + 1$, we choose an n so large that $2^n > |x|^n + e$. Now, in the first case, notice that by the choice of t_{e+1} and hence t_q for $q \geq e + 1$, we have

$$\Gamma_e(A; x) = \Gamma_e(A_{e+1}; x) = 1 \neq 0 = B_{e+1}(x) = B(x).$$

[The fact that $B(x) = 0$ follows from the observation that no string of length $n = |x|$ is ever put into A .]

In Case 2, we have $\Gamma(A'_e; x) = 0$. Now, since the only difference between A'_e and A_{e+1} on strings of length $\leq t_{e+1}$ is on z and we choose z , a string of length $|x|$ *not addressed in the $\Gamma_e(A'_e; x) = 0$ computation, by the use principle, we must have*

$$\Gamma_e(A; x) = \Gamma_e(A_{e+1}; x) = \Gamma_e(A'_e; x) = 0 \neq 1 = B_{e+1}(x) = B(x).$$

The last equality follows since we have put into A a string z of the same length as x which puts x into B . In either case, we meet the R_e .

A more subtle generalization of the finite extension method is the *priority method*. We begin by looking at the simplest incarnation of this elegant technique, the *finite injury priority method*. This method is somewhat like the finite extension method but with backtracking. It also bears some resemblance to the Gurevich-Harrington [252] game of **LAR** in their proof of Rabin's result that S2S has a decidable theory.

In any case, the idea is the following. Suppose that now we must again satisfy requirements $R_0, R_1, \dots$, but this time we are constrained to some sort of effective construction, so *we are not allowed oracle-type questions in the construction*. As an illustration, in the result below, we will construct *recursively enumerable* A and B with incomparable Turing degrees. Now the Kleene-Post method constructs languages with incomparable degrees below $0'$, but they are *not* recursively enumerable. The reason that A and B are not recursively enumerable is that we satisfy the requirements *in order*. To do this, we are using an $0'$ oracle question at each stage. To make A and B recursively enumerable, somehow we must have a recursive construction where elements go into the sets A and B and never leave them. The key idea discovered independently by Friedberg [233] and Muchnik [357] is to pursue *two* or more strategies for the R_e in the following sense. It seems that we need to know which option "Does τ exist or not?" to know which strategy to pursue. But the idea is that we first *guess* that no such τ exists for our witness x . This means that nothing is really done for the sake of R_e (except to keep $x \notin B_s$) *unless* we see a stage where some $\tau \subseteq A_s$ exists. If such a stage occurs, then we will try to make A extend τ and win, as before, by putting x into B if necessary. So whatever case occurs, we will win.

The only problem with all of this is that this action will probably change B on x . This action may upset, “injure” in the standard terminology, some *other* requirement trying to preserve B . To make sure that everyone gets to be met, we put a “priority ordering” on all the R_e and only allow R_j to injure R_i if R_j has higher priority than R_i . If R_j is injured at stage s , then we will “initialize” (meaning restart with a new large follower) the requirement R_j .

In general, in a finite injury priority argument, one has a list of requirements in some priority ordering. There are several different ways to meet individual requirements R_e . Exactly which way depends upon information that is not available to us but is “revealed” to us during the construction. The problem is that actions by one requirement can injure others. We must construe things so that only requirements of higher priority can injure ones of lower priority, and we can always restart the ones of lower priority once they are injured. In a finite injury argument, any R_e *requires attention* only finitely often and we argue by induction that all the requirements get an environment wherein they can each be met. We remark that there are much more complex infinite injury arguments where R_i might injure some R_j infinitely often. But the key there is that the injury is somehow controlled so that the *coherence criterion* is satisfied. All requirements eventually get an environment where they can be met (Harrington’s “golden rule.”). The reader is referred to Soare [431] for an account of modern recursion theory and, in particular, an account of these beautiful techniques.

We now turn to the formal proof of the following theorem of Friedberg and Muchnik which solved the famous problem of Post, which asked if $\mathbf{0}$ and $\mathbf{0}'$ were the only recursively enumerable degrees. (And if they were, it would mean that all semidecidable problems were either decidable or the “halting problem in disguise.”) Historically, the Friedberg–Muchnik Theorem was the first place where the priority method was used. We remark in passing that it is possible to solve Post’s problem without a priority argument but with much more difficult techniques. (See Kucera [314].)

Theorem 19.7 ([233, 357]) *There exist recursively enumerable languages A and B such that A and B have incomparable Turing degrees.*

Proof. We will build $A = \cup_s A_s$ and $B = \cup_s B_s$ in stages to satisfy the same requirements as the Kleene–Post Theorem; that is, we make recursively enumerable A and B to meet the requirements

$$\begin{aligned} R_{2e} : \Phi_e(A) &\neq B, \\ R_{2e+1} : \Phi_e(B) &\neq A. \end{aligned}$$

The strategy for a single R_j . We begin by looking at the strategy for a single R_j . Without loss of generality, let $j = 2e$.

(i) Initially, we will pick a new fresh number $x = x(j)$ to follow R_j . This number is targeted for B , and, of course, we have $x \notin B_s$.

(ii) We wait for a stage $t > s$ to occur with $\Phi_{e,t}(A_t; x) \downarrow = 0 = B_t(x)$. [**Comment:** If stage t does not occur, then we must have $\Phi_e(A; x) \neq B(x)$. For suppose

otherwise. Then, if $\Phi_e(A; x) = B(x)$, since we never put x into B , it must be that $\Phi_e(A; x) \downarrow = 0$. Now, since all computations halt in a finite number of steps and some stage n must occur with $A_{n'}(z) = A(z)$ for all $n' > n$ and z with $|z| \leq \max\{|p| : p \in u(\Phi_e(A; x))\}$, we must have a stage where the computation $\Phi_{e,t'}(A'; x) = \Phi_e(A; x)$ is final. One can take $t = t'$.]

(iii) Should stage t occur, we will let R_j require attention by performing the Kleene-Post action with A_t taking the role of τ ; that is, we will set $A_{t+1} = A_t$, but put x into $B_{t+1} - B_s$, causing

$$\Phi_{e,t+1}(A_{t+1}; x) = 0 \neq 1 = B_{t+1}(x) = B(x).$$

In the construction below, we will then act to protect this with priority e .

Note that when we take action (iii), we might injure other R_k trying to preserve the use of $\Phi_{e',t}(B_t; x')$ because $x \in u(\Phi_{e',t}(B_t; x'))$, and hence, the $\Phi_{e',t}(B_t; x')$ computation will be using the fact that " $x \notin B$ " which R_j has made false at stage $t + 1$.

Definition. We say that R_j requires attention at stage s if j is least such that one of the following pertains.

- (i) R_j has no follower at stage s .
- (ii) R_j has a follower $x(j, s)$ at stage s and it is *waiting* and, furthermore, supposing that $j = 2e$,

$$\Phi_{e,s}(A_s; x(j, s)) \downarrow = 0 = B_s(x(j, s)).$$

Construction.

Stage 0. Set $A_0 = B_0 = \emptyset$.

Stage $s > 0$. Find the least j with R_j requiring attention. Adopt the appropriate case below.

Case 1. (i) Pertains.

Action. Find a large fresh number x (i.e., exceeding all numbers computations, etc. previously seen) and appoint $x(j, s) = x$ as a waiting follower for R_j . Initialize all $R_{j'}$ with $j' > j$. (That is, cancel all followers associated with $R_{j'}$.) Do nothing else. (So $A_s = A_{s-1}$ and $B_s = B_{s-1}$, etc.)

Case 2. (ii) Pertains.

Action. Initialize all $R_{j'}$ for $j' > j$. Set $A_s = A_{s-1}$, but set $B_s = B_{s-1} \cup \{x(j, s)\}$. Declare $x(j, s)$ to be no longer waiting.

End of construction.

Verification. We prove by induction on j that:

- (a) Each R_j receives attention only finitely often.
- (b) $\lim_s x(j, s) = x(j)$ exists.
- (c) R_j is met.

For an induction, suppose (a), (b), and (c) for all $j' < j$. Let s_0 be a stage good for j ; that is, for all $s \geq s_0$, for all $j' > j$,

- (a') $R_{j'}$ does not require attention at stage s .
- (b') $x(j', s) = x(j', s_0)$.
- (c') $R_{j'}$ is met at stage s .

If we choose s_0 to be minimal, then it is clear that since no requirements have followers at the beginning of the construction, and lose them whenever higher priority ones receive attention, it can only be that R_j receives attention via (i) at stage $s_0 + 1$, and is appointed a large fresh follower $x = x(j, s_0 + 1)$. By choice of s_0 , x is never cancelled: The only requirements that could cancel x are $R_{j'}$ for $j' < j$. There are two possibilities.

The first is that x is waiting forever. In this case, there is no stage t with $\Phi_{e,t}(A_t; x) \downarrow = 0 = B_t(x)$. This means that either $\Phi_t(A; x) \uparrow$ or $\Phi_e(A; x) = 1 \neq 0 = B(x)$. In either case, we win.

The second case is that (ii) pertains to R_j at some stage $s > s_0 + 1$. In this latter case, we act to cause a disagreement at stage s , namely

$$\Phi_{e,s}(A_s; x) = 0 \neq 1 = B_s(x).$$

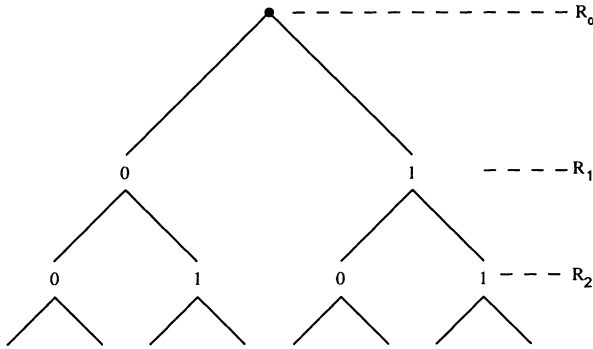
Now, since we initialize all $R_{j'}$ for $j' > j$, and new followers are always appointed large and fresh, it follows that this stage s disagreement is immortal.

In either case, R_j only receives attention at most twice more after stage s_0 and is met. Furthermore, one sees that $x(j, s_0 + 1) = x(j, t) = x(j)$ for all $t > s_0$. This concludes the induction and hence the proof of the Friedberg–Muchnik Theorem.

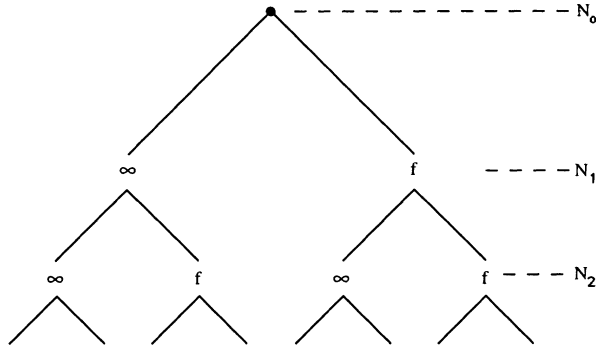
We remark that the above is the very simplest of all finite injury arguments since it is what is called “bounded injury” in the sense that we can put a bound *in advance* on the number of injuries that R_j will have. In this case, the bound is 2^j . There are examples such as Sack’s splitting theorem (see Exercises 12) where we have no such bound. For one of the results of the next section, we will also need the infinite injury method, or, more precisely, the Π_2 version of the infinite injury method. The Friedberg–Muchnik construction above can be viewed another way. We can attach versions of requirements to a “priority tree,” in this case the tree Σ^* , the tree of all binary strings. The idea is the following.

Attached to the empty string λ is R_0 . More generally, each node σ has a “version” of R_e where $|\sigma| = e$. Thus, there are 2^e versions of R_e . In this context, we refer to σ on the tree as *guesses*. Each node σ has two *outcomes* 0 and 1. The outcome 0 indicates that this version of R_e appoints a follower but never enumerates the follower into its target set (i.e., Case 2 does not hold for this version of R_e). The outcome 1 indicates that the version of R_e at guess σ actually enumerates its follower into its target set. See Figure 19.2 for this setup.

Now, the idea is the following. Consider R_1 . The version guessing 0 believes that “ R_0 will appoint a follower then never again act.” Thus, its action is to simply act *immediately* after its guess “appears correct”; that is, once it sees R_0 appoint its follower, it can act immediately to appoint its follower. Its belief of the actual universe will turn out to be false only if R_0 does, in fact, later act to enumerate its follower. At the outcome 1 of R_0 , we will have a “backup” strategy which believes “ R_0 will in fact enumerate its follower.” The backup strategy for R_1 , this *version* of R_1 , will only act if its guess appears correct. Thus, it only acts *after* R_0 enumerates its follower. Then and only then, this version of R_1 wakes up and



For the Friedberg-Muchnik Theorem



For the Minimal Pair Theorem

FIGURE 19.2. The Assignment of Priorities and the Outcomes

appoints its follower. The version guessing outcome 0 is now known to be wrong and would be canceled.

All of the above is simply extended inductively in the tree. This is a version of R_4 with guess $\sigma = 1\hat{0}\hat{0}\hat{1}$ would have as its strategy, “the σ strategy,” that it wait for R_0 to enumerate, the version of R_1 guessing 1 to appoint a follower, the version of R_2 guessing $1\hat{0}$ to appoint a follower, and the version of R_3 guessing $1\hat{0}\hat{0}$ to enumerate its follower before it acts.

The *True Path (TP)* of the construction is the actual way the construction is satisfied as seen by the tree. Thus, *TP* is the leftmost path visited infinitely often,

where “leftmost” is measured in terms of lexicographic ordering (that is, $\sigma \leq_L \tau$ means that σ is lexicographically less than τ).

The reader might wonder what all this terminology is about. For the Friedberg–Muchnik Theorem, the payoff of new insight provided by this reorganization of the construction does not really balance the additional notational and conceptual burden. However, all of the above concepts were invented in the last 20 years to make infinite injury arguments comprehensible. The key difference is the following. In the above argument, the apparent true path TP_s , the guess of length s that appears correct at stage s , only moves left, since R_e can only go from outcome 0 at σ to outcome 1 at σ . Hence, by the Limit Lemma (Lemma 19.2), the true path is recursive in $\emptyset'$. In infinite injury arguments, TP_s can move both left and right. Assuming that the tree is finitely branching, this means that TP is only recursive in $\emptyset''$.

In an infinite injury argument, again we have requirements $\{R_e : e \in \mathbb{N}\}$. But now, the action of R_e can be infinitary. Obviously, we cannot just initialize R_{e+1} whenever R_e acts, for then it would never be met. What happens is that we will have a version that guesses R_e acts infinitely often. This backup strategy gets to act just in the stages when R_e looks like it will be acting infinitarily. The requirements must be such that the golden rule will again be satisfied: There must be a version of R_{e+1} which has a strategy that can live with R_e acting infinitarily, and, conversely, this version of R_{e+1} must not in the limit stop R_e from being met.

We give one example and refer the reader to the Exercises for others (Exercises 13, 14, and 15).

Theorem 19.8 ([317, 470]) *There exist nonrecursive recursively enumerable sets A and B such that for all sets C if $C \leq_T A, B$, then C is recursive. The degrees of these sets are said to form a Minimal Pair.*

Proof. We construct $A = \cup_s A_s$ and $B = \cup_s B_s$ in stages to satisfy the following requirements for $e \in \mathbb{N}$:

$$\begin{aligned} R_e &: \overline{A} \neq W_e. \\ Q_e &: \overline{B} \neq W_e. \\ N_{i,j} &: \Phi_i(A) = \Phi_j(B) = f \text{ and } f \text{ is total, implies } f \text{ recursive.} \end{aligned}$$

Here, the reader is reminded that $W_e = \text{dom } \varphi_e$ is the e -th recursively enumerable set. We meet the R_e and the Q_e by a Friedberg–Muchnik-type strategy; that is, we shall pick a follower x , targeted for A in the case of R_e , and wait until x enters $W_{e,s}$. Of course, should x never enter $W_{e,s}$ for any s , then $x \notin (W_e \cup A)$ and hence $A \neq \overline{W_e}$. Should x enter $W_{e,s}$ for some stage s , then we can win forever by putting x into A_t at some $t \geq s$.

The tricky requirements are the $N_{i,j}$. We will first discuss how to meet a single $N_{i,j}$ in isolation and then look at the coherence problems between the various requirements and the solution to these provided by the use of a tree of strategies.

For a single $N_{i,j}$, we will need the auxiliary functions

$$\ell(i, j, s) = \max\{x : \forall y < x (\Phi_{i,s}(A_s; y) = \Phi_{j,s}(B_s; y))\}.$$

$$m\ell(i, j, s) = \max\{\ell(i, j, t) : t < s\}.$$

We call $\ell(i, j, s)$ the *length of agreement function* and call $m\ell(i, j, s)$ the *maximum length of agreement function*. The maximum length of agreement function is a sort of “high-water mark” for lengths of agreement seen so far. We shall call a stage i, j -*expansionary* if the current length of agreement exceeds the previous high-water mark; that is, s will be called i, j -expansionary if $\ell(i, j, s) > m\ell(i, j, s)$. Also, we let $mu(i, j, s)$ denote the maximum element used in any computation below $\ell(i, j, s)$. *Following standard conventions, we will always suppose that the procedures are given sufficiently slowly that all uses are bounded by s at stage s .*

The key idea for a single $N_{i,j}$ is the following. Suppose that $\ell(i, j, s) > x$. Suppose that we allow some element y to enter A_{s+1} below $u(\Phi_i(A_s; x))$, but nothing else to enter A or B at Stage $s + 1$. Then, if we do not allow any further elements $\leq mu(i, j, s)$ to enter A or B until the next i, j -expansionary stage $t > s + 1$, it can only be that $B_t(z) = B_s(z)$ for all $z \leq mu(i, j, s)$, and hence $\Phi_{j,s}(B_s; x) = \Phi_{j,t}(B_t; x)$. But since t is expansionary, this means that

$$\Phi_{i,s}(A_{i,s}; x) = \Phi_{i,t}(A_t; x) [= \Phi_{j,t}(B_t; x) = \Phi_{j,s}(B_s; x)].$$

That is, even though the $\Phi_{i,t}(A_t; x)$ -computations might have changed because y entered A_{s+1} , the value of the computation on x remains the same, since the B side has not changed and hence must be giving the same answer as it did at stage s .

So, in summary, the idea is to only change one side of a computation between expansionary stages and then argue that the “other side” will “hold” the computation at its current value until the next expansionary stage is found, when we will be free to enumerate into either side again. Thus, the Friedberg strategies *guessing that* $\ell(i, j, s) \rightarrow \infty$ must only put numbers into A or B at expansionary stages. Finally, we remark that if no expansionary stage $t > s$ is ever found, then $\ell(i, j, s) \nrightarrow \infty$ and hence $\Phi_i(A) \neq \Phi_j(B)$.

Coherence. There is a problem with all of this. Consider two N -type requirements N and N' . Now suppose that N has higher priority than N' . Now, N requests us to only put numbers into A or B during its expansion stages $\{s_1, s_2, \dots\}$. Similarly, N' might request us to only put numbers in during stages $\{t_1, t_2, \dots\}$. The problem is that these sets of stages might be *disjoint*. Then, N blocks us from putting numbers into A or B stages t_i and N' blocks us from putting numbers in during stages s_i . Hence, *collectively*, the pair block us from *ever* putting numbers into A or B .

This problem is overcome by the following observation. Given that this is a version of N' guessing that N has infinitely many expansionary stages, this version of N' can be such that it only *acts during* N 's expansionary stages. This will force N' 's expansionary stages to be *nested* within N 's expansionary stages. In this way, the requirements are forced to cohere.

We now turn to the formal details.

The Priority Tree. We use the tree $\mathcal{T} = \{\infty, f\}^*$ (see Figure 19.2). We assign $N_{i,j}$ to σ on $\mathcal{T}$ iff $|\sigma| = \langle i, j \rangle$. Also, if $|\sigma| = 2e$, then we also assign R_e to σ and if $|\sigma| = 2e + 1$ we assign Q_e to σ . For a requirement M we will write M_σ for the version of M at guess σ . Again, we use lexicographical ordering with $\infty <_L f$.

Definition 19.9 (a) We define the notions σ -stage, $m\ell(\sigma, s)$, and σ -expansionary by induction on $|\sigma|$.

- (i) Every stage s is a λ -stage.
- (ii) Suppose that s is a τ -stage with $|\tau| = \langle i, j \rangle$. Let $\ell(\tau, s) = \ell(i, j, s)$. Define

$$m\ell(\tau, s) = \max\{0, \ell(\tau, t) : t \text{ is a } \tau\text{-stage} < s\}.$$

We say that s is τ -expansionary if $\ell(\tau, s) > m\ell(\tau, s)$ and declare s to be a τ^∞ -stage. If $\ell(\tau, s) \leq m\ell(\tau, s)$, declare that s is a τ^f -stage.

- (b) We define TP_s to be the unique σ of length s with s a σ -stage.

Definition 19.10 (a) We say that R_σ *requires attention* at stage s if $W_{e,s} \cap A_s = \emptyset$ (where $2e = |\sigma|$), s is a σ -stage, and one of the following holds.

- (i) R_σ currently has no follower.
- (ii) R_σ has a follower $x \in W_{e,s}$.
- (b) We similarly define Q_σ to require attention.

The Construction.

Step 1. Compute TP_s . Initialize all versions of requirements at guesses $\tau \not\leq_L TP_s$.

Step 2. Find the R_σ or Q_σ of highest priority that requires attention at stage s . Without loss of generality, we will suppose this to be R_σ . Initialize all requirements at guesses, τ with $\tau \not\leq \sigma$. Adopt the appropriate case below.

Case 1. Definition 19.10 (i) holds.

Action. Appoint $x(\sigma, s) = s$ to follow R_σ . (Remember that s is larger than all computations seen so far by convention.)

Case 2. Definition 19.10 (ii) holds.

Action. Enumerate x into A_{s+1} .

End of Construction.

Verification. Let TP be the leftmost path visited infinitely often; that is, $\lambda \subset TP$, and for all τ , if $\tau \subset TP$, then $\tau^\infty \subset TP$ iff $\exists^\infty s(\tau^\infty \subset TP_s)$. Otherwise, $\tau^f \subset TP$.

Lemma 19.11 *All the R_e and Q_e have versions that are met, and for all $\tau \leq TP$, R_τ (or Q_τ) acts only finitely often.*

Proof. This lemma is proven by induction on e . We consider R_e . Let $\sigma \subset TP$ with $|\sigma| = 2e$. Go to a stage s_0 where for all $\tau <_L \sigma$ and $s > s_0$:

- (i) if $\tau \not\subset \sigma$, s is not a τ -stage;
- (ii) if M is a Q or R requirement assigned to τ , then M will not act at stage s .

Assuming s_0 to be least and a σ -stage, we can assume that either $W_{e,s_0} \cap A_s \neq \emptyset$ (in which case we are done), or R_σ receives attention via Case 1 getting a follower x at stage s_0 . This follower is immortal by choice of s_0 and the induction hypothesis. It will succeed in meeting R_e as in the basic module since it has priority at each σ -stage.

Lemma 19.12 *All the $N_{i,j}$ have versions that are met.*

Proof. Again, we prove this by induction. Let $\sigma \subset TP$ with $|\sigma| = \langle i, j \rangle$. Choose s_0 as in Lemma 19.11, so that no higher-priority action can cause grief to N_σ . Now, if $\sigma \hat{\infty} f \subset TP$, we are done since $\liminf \ell(i, j, s) < \infty$ and hence $\Phi_i(A) \neq \Phi_j(B)$.

So, we suppose that $\sigma \hat{\infty} \infty \subset TP$. To compute $\Phi_i(A; x)$ recursively, find the least $\sigma \hat{\infty}$ -stage $s = s(x) > s_0$ such that $\ell(\sigma, s) > x$. Note that this can be computed recursively from the parameters s_0 and σ . We claim that $\Phi_i(A; x) = \Phi_{i,s}(A; x)$. To see this, note that by Step 1 of the construction, we will initialize all $\tau \not\subseteq TP_s$ at stage s . In particular, at stage s by choice of s_0 and since we appoint *new* followers to be *large*, and we are never above or left of $\sigma \hat{\infty}$ after stage s_0 , the only numbers which are *below* s and can enter A or B *after* stage s are followers associated with $\gamma \supseteq \sigma \hat{\infty}$. Such followers can only enter their target sets at $\sigma \hat{\infty}$ stages $s \geq s_0$; that is, they can only enter at σ -expansionary stages. In particular, as with the basic module, we can argue that for any $\sigma \hat{\infty}$ -stage $t \geq s$, at most one number $\leq mu(\sigma, t)$ can enter A or B before the next $\sigma \hat{\infty}$ -stage $t' > t$.

Thus, exactly as in the basic module, we have that

$$\begin{aligned} \Phi_{i,s}(A_s; x) &= \Phi_{i,t}(A_t; x) = \Phi_{j,s}(B_s; x) \\ &= \Phi_{j,t}(B_t; x) = \Phi_i(A; x) = \Phi_j(B; x), \end{aligned}$$

for all $\sigma \hat{\infty}$ -stages $t > s$.

Before we look at the structure of parameterized reducibilities, we need one further technique: *Delayed Diagonalization*. This technique was developed in complexity theory, although it has found uses in classical recursion theory such as in Downey and Shore [184]. The technique works roughly as follows. We aim to satisfy some requirement R_e , but are constrained in the construction so that we are not able to use an oracle to allow us to decide how to diagonalize at the time we wish. So what we do is to set things up so that if we pursue some strategy long enough, eventually we will meet our objectives. For example, we might want to diagonalize against some requirement and we know that we will do so on *one* of an exponential number of strings. Now, maybe the action we need to do for R_{e+1} depends upon exactly *which* string achieved the diagonalization. The trouble is, we might be constrained to work in polynomial time, so at the current stage, we can not know which string achieved the diagonalization. We can only look at polynomially many of the exponentially many strings. The idea is that we “delay” the construction in the sense that we simply keep extending the set marking time until a stage is found where, “looking back,” we can see which string did the construction. For example, although at stage s , there are exponentially many strings of length s , at stage $t = 2^s$ there are only polynomially many strings of *length* s .

We give the classical example: Ladner’s Density Theorem. We will need the following notation.

Notation. (i) We denote by z_n the n -th string in the length/lexicographic ordering of Σ^* .

(ii) We let $A \oplus B = \{1 \hat{\infty} x : x \in A\} \cup \{0 \hat{\infty} y : y \in B\}$.

We also need the following easy technical lemma.

Lemma 19.13 (Slow Enumeration Lemma) (i) *Let D be a recursive language. There exists a polynomial time computable function f such that the range of f is D .*

(ii) *Furthermore, we can ask that for all z , if $y \in \{f(z_0), \dots, f(z_s)\}$ and $|f(z_{s+1})| > |y|$, then for all $s' > s$, $|f(z_{s'})| > |y|$ (The Length Increasing Property).*

Proof. As D is recursive, there is an injective recursive function g with range D . We will have f as a slowed-down version of g . We define g in stages.

Stage 0. (i) Let $g(\lambda) = f(\lambda)$. Let $n(0) = 1$. Declare $z_{n(0)}$ to be the waiting string.

Stage $s + 1$. We will have a “waiting” string $z_n = z_{n(s)}$. See if $g(z_n) \downarrow$ in $\leq s + 1$ many steps. If not, then set $n(s + 1) = n(s)$ and define $f(p) = f(1^s)$ for all strings p of length $s + 1$. If $g(z_n) \downarrow$ in $\leq s + 1$ many steps, then for all strings p of length $s + 1$, define $f(p) = g(z_n)$. In this latter case, set $n(s + 1) = n(s) + 1$.

Clearly, the construction of f gives a polynomial-time computable function such that the range of f is D .

(ii) First arrange that g has the length increasing property, and then use the construction of (i).

Theorem 19.14 (Ladner’s Density Theorem [321]) *The polynomial-time degrees of recursive languages are dense; that is, let A and B be recursive languages with $A <_q^p B$ for $q \in \{T, m\}$. Then there exists a recursive language C such that $C \leq_m B$ and $A <_q^p A \oplus C <_q^p B$.*

Proof. We verify only the case $q = T$. The other one is virtually the same. We must build $C \leq_m B$ so that we satisfy the requirements below:

$$\begin{aligned} R_e : \Gamma_e(A \oplus C) &\neq B, \\ Q_e : \Gamma_e(A) &\neq C. \end{aligned}$$

Actually, the strategy for meeting all the R_e is obvious. Simply let $C = \emptyset$. It cannot be that $\Gamma_e(A \oplus \emptyset) = B$ lest $B \leq_T^p A$. Similarly, the strategy for meeting all the Q_e is again obvious. This time, simply set $C = B$ and then we must have $\Gamma_e(A) \neq C$. The difficulty is in the resolution of these two ideas, since making $C = \emptyset$ is patently incompatible with making $C = B$.

Ladner’s idea is based on the following observation: We know that if we make $C = \emptyset$, then it must be that for some x ,

$$\Gamma_e(A \oplus C; x) \neq B(x).$$

The idea is that we will set $C(x) = 0 = \emptyset(x)$ **long enough** so that some x with $\Gamma_e(A \oplus C; x) \neq B(x)$ appears, and then switch to making $C = B$ on longer strings for the next requirement which will be of the Q_e type. So the set C will resemble, in the phrase of Wolfgang Maass, B with holes in it; that is, it will appear as B for long intervals and then as $\emptyset$ for long intervals. Which it currently emulates is decided by a polynomial-time relation $D(x)$, which, in turn, is decided by the

delayed diagonalization “looking back” technique. Thus, we will define C via D via

$$x \in C \text{ iff } [x \in C \wedge D(|x|) = 1].$$

Clearly then, if D is polynomial-time-computable, then $C \leq_m^p B$ since to decide if $x \in C$, we first compute $D(|x|)$. If this is 1, we have $x \in C$ iff $x \in B$ and if this is 0, we have $x \notin C$.

Now by the Slow Enumeration Theorem, we will have A and B given as the range of polynomial-time-computable functions f and g , respectively. For convenience, we will write $A_s = \{f(0), \dots, f(s)\}$ and $B_s = \{g(0), \dots, g(s)\}$. Here, we are writing $z_i = i$ for convenience, and remind the reader of the length increasing property of these enumerations. Assuming that D is polynomial-time-computable, this implicitly gives C_s via B_s .

Definition 19.15 (Certified Computations) For a polynomial-time procedure Γ , we will say that a computation of the form

$$\Gamma_s(A_s \oplus C_s; x) = \text{ or } \neq B_s(x)$$

or of the form

$$\Gamma_s(A_s; x) = \text{ or } \neq C_s(x)$$

(which halts in $\leq s$ steps) is *certified* at stage s if we have that the latest elements of A_s and B_s (and hence all future elements of $A - A_s$ and $B - B_s$) have lengths exceeding the use of the relevant computation.

The crucial point of *certified* computations is that they are *final*. In particular, they are not giving us the wrong information purely because of the slowness of the enumerations of A or B . The above definition is essential to virtually all applications of delayed diagonalization and should be digested before proceeding.

Construction. The construction is specified by the definition of D below. For convenience we will also have another parameter, namely $\text{req}(s)$, the current requirement we are meeting.

Stage 0. Set $\text{req}(0) = R_0$, and $D(0) = 0$ (so that C is emulating $\emptyset$ locally).

Stage $s + 1$. There are two cases.

Case 1. $\text{req}(s) = R_e$ for some e [and hence it will be that $D(s) = 0$].

Action. See if there is a z_i with $i \leq s$ such that

$$\Gamma_{e,s}(A_s \oplus C_s; z_i) \neq B_s(z_i)$$

with certified computations.

Subcase 1.1 Some such z_i exists.

Action. Set $\text{req}(s + 1) = Q_e$ and set $D(s + 1) = 1$.

Subcase 1.2 No such z_i exists.

Action. Do nothing; that is, keep $\text{req}(s + 1) = R_e$ and $D(s + 1) = 0$.

Case 2. $\text{req}(s) = Q_e$ for some e .

Action. See if there is some z_i for $i \leq s$ such that

$$\Gamma_{e,s}(A_s; z_i) \neq C_s(z_i)$$

with certified computations.

Subcase 2.1 Some such z_i exists.

Action. Set $\text{req}(s+1) = R_{e+1}$ and $D(s+1) = 0$.

Subcase 2.2 No such z_i exists.

Action. Do nothing; that is, keep $\text{req}(s+1) = Q_e$ and $D(s+1) = 1$.

End of Construction.

Verification. We argue that all the requirements are met in order and $\text{req}(s)$ runs through all the requirements. We do so by induction. So suppose we have met all requirements of higher priority than R_e , and at stage s_0 , we set $\text{req}(s_0) = R_e$. Thus, we would set $D(s_0) = 0$. Now, at each stage $s \geq s_0$ we are keeping $\text{req}(s) = R_e$ and $D(s) = 0$ while we cannot see some z_i with $i \leq s$ and $\Gamma_{e,s}(A_s \oplus C_s; z_i) \neq B_s(z_i)$ with certified computations. By the nature of certified computations, if we see some such z_i , then we will know that, in fact,

$$\Gamma_e(A \oplus C; z_i) \neq B(z_i).$$

Furthermore, we will move on to Q_e and never set $\text{req}(s') = R_e$ at any stage $s' \geq s$. Thus, it suffices to argue that some such stage s with $\Gamma_{e,s}(A_s \oplus C_s; z_i) \neq B_s(z_i)$ via certified computations exists. If we suppose not, then we must have that $\Gamma_e(A \oplus C) = B$. But the key thing to note is that $C =^* \emptyset$; that is, except for a finite piece, C is $\emptyset$. Because of this, we can put the nonempty finite piece in a table lookup and then get $\Gamma'_e(A \oplus \emptyset) = B$ where Γ' emulates Γ except on the finite portion where C is not $\emptyset$. There, Γ' uses the table lookup instead. But then we have just described a reduction reducing B to A via Γ' , and that is a contradiction. Thus, we must have some stage s with $\Gamma_{e,s}(A_s \oplus C_s; z_i) \neq B_s(z_i)$ via certified computations.

The case $\text{req}(s_0) = Q_e$ is essentially the same and is left as an easy exercise for the reader (Exercise 9).

Finally, note that $C \leq B$ since D is computable in quadratic time. All the searches are bounded by s and there are s of them.

We remark that although proving that the polynomial-time degrees of recursive sets are dense only uses a (delayed) diagonalization argument, the T -degrees of recursively enumerable sets are also dense, but this argument is a very interesting infinite injury argument. (See Refs. [402] and [430], Chap. VIII.)

It is also true that the polynomial-time degrees of *all* sets are dense, this being a theorem of Juichi Shinoda [420]. To establish Shinoda's density theorem, we need to use the so-called "speedup" technique. We will not need this technique for the purposes of the present book, although it is used in many powerful degree constructions. However, for completeness we will briefly treat this technique, and the reader is referred to the exercises of this and the next section for some typical applications.

The technique is best thought of as a miniaturization of the finite extension technique in classical degree arguments. The best arena to demonstrate the technique is the construction of a minimal pair of polynomial-time degrees. We remark that it is easy to prove this theorem using delayed diagonalization (see Exercise 16).

Theorem 19.16 (Ladner [321]) *There exists a minimal pair of polynomial-time degrees; that is, there exist recursive languages A and B such that if $A, B \notin P$ and for all $C \leq_T^P A, B$, $C \in P$.*

Proof. We must meet the following requirements:

$$P_e : \Gamma_e(\emptyset) \neq A.$$

$$Q_e : \Gamma_e(\emptyset) \neq B.$$

$$N_e : \Delta_e(t) = \Lambda_e(B) = f \Rightarrow f \in P.$$

Here, $\langle \Delta_e, \Lambda_e \rangle_{e \in \mathbb{N}}$ is an enumeration of all pairs of P -time procedures.

Now, first, pretend that we were actually using the finite extension method to build a minimal pair of T -degrees below $\mathbf{0}'$, as in Theorem 19.5. (Thus, the Γ, Λ, Δ are now classical Turing procedures.) To meet the P_e at a stage s would be easy. Given A_s and B_s , we would pick a large follower x and compute $\Gamma_e(\emptyset; x)$, and then make $A(x) \neq \Gamma(\emptyset; x)$. The Q_e are met analogously. To meet the N_e , at some stage t we would ask the following:

$$\exists x, \sigma, \tau (\sigma \supset A_t \wedge \tau \supset B_t \wedge \Delta_e(\sigma; x) \neq \Lambda_e(\tau; x)).$$

If the answer is “yes,” then we would extend A_t to be σ and extend B_t to be τ , causing $\Delta_e(A)(x) \neq \Lambda_e(B)(x)$ since A extends $A_{t+1} = \sigma$ and B extends $B_{t+1} = \tau$. If no such σ, τ , and x exist, then we can effectively compute $\Delta_e(A)(x)$ by simply computing $\Delta_e(\beta; x)$ on any β extending A_t .

To miniaturize the above to recursive languages, we must bound the searches for σ, τ , and x . This bounding causes the argument to actually become a finite injury argument, as we will now see.

The overall structure of the argument is to consider the requirements in cycles. Initially, the cycle consists of one requirement, the one of highest priority. Arrange the requirements in some descending priority ordering $R(0), R(1), \dots$ (i.e., so that $R(i) \in \{P_j, Q_j, N_j\}$). At any stage s , the cycle will be a finite collection of requirements $\mathcal{R} = R(n_s), \dots, R(0)$, arranged in *ascending* order of priority. [Thus, $R(i+1)$ has *lower* priority than $R(i)$ and $R(0)$ has highest priority for the construction.] For the $\mathcal{R}$ -cycle, we will consider the requirements $R(n_s)$, then $R(n_{s-1})$, etc. in order, and only finish the $\mathcal{R}$ -cycle when we finish considering $R(0)$. At the next stage, we would begin the $\mathcal{R}^+ = R(n_s + 1), R(n_s), \dots, R(0)$ -cycle. *It is crucial to the construction that while we are in the $\mathcal{R}$ -cycle we will consider the requirements of the $\mathcal{R}$ -cycle one at a time in reverse order of priority and consider no other requirements.*

Construction. Suppose that we are in the $\mathcal{R}$ -cycle. Let $\text{Begin}(s)$ be the initial stage where the current $\mathcal{R}$ -cycle began.

At stage s , we will have the parameter $p(s)$ and be considering a requirement $R(j)$, and have a *pending* requirement $R(s)$ with a pending action $\text{act}(s)$. We

say that $R(j)$ *asserts control* of the construction. It does so until a stage $p'(s)$, as described below. $p'(s)$ is determined by what type of requirement $R(j)$ is and by the construction.

At the end of stage $p'(s)$, we will pass control of the construction to $R(e - 1)$. We will make a decision to either make $R(j)$ the *new* pending requirement with a new pending action $act(p'(s))$, or keep the old pending action and requirement. If it is the case that $R(j) = R(0)$, we actually *perform* the pending action. Since things only change when $s = p(s)$, we will assume that this is the case.

Case 1. $R(j) = P_e$ for some e . [Or $R(j) = Q_e$ dually.] If $R(j)$ is already met, replace $R(j)$ by $R(j - 1)$, and the pending action and requirement both remain the same.

Assuming that $R(j)$ is not as yet met, we wish to know the result of

$$\Gamma_e(\emptyset; 1^{p(s)}).$$

We let $p'(s)$ be sufficiently large so that $\Gamma_e(\emptyset; 1^{p(s)})$ halts in fewer than $p'(s)$ steps. (For instance $p'(s) = 2^{\gamma_e(p(s))}$, where γ_e is the running time of Γ_e , would be more than sufficient.)

At stage $p'(s)$, $R(j)$ will become the pending requirement, and declare $R(j)$'s pending action, $act(p'(s))$, to be

“Extend $A_{Begin(s)}$ and $B_{Begin(s)}$ via the empty extension to length $p(s)$ except possibly for A and $1^{p(s)}$ which will be assigned as $A(1^{p(s)}) = 1 - \Gamma_e(\emptyset; 1^{p(s)})$.”

Now replace $R(j)$ by $R(j - 1)$. (The reader should notice that if the requirement is of the P_e or Q_e form, then either it will have been met or will become the pending requirement. This fact is *not* true of an N_e which may *never* become the pending requirement.)

Case 2. $R(j) = Q_e$ for some e . Again, if $R(j)$ is already met, replace $R(j)$ by $R(j - 1)$, and the pending action and requirement both remain the same.

Assuming that $R(j)$ is not as yet met, choose $p'(s)$ to be sufficiently large so that we can see in $p'(s)$ many steps if there is any extensions σ of $A_{Begin(s)}$ and τ of $B_{Begin(s)}$ of length $2^{\max\{\delta_e(p(s)), \lambda_e(p(s))\}}$ (the uses of Δ_e and Λ_e , respectively), such that for some z with $|z| \leq p(s)$,

$$\Delta_e(\sigma; z) \neq \Lambda_e(\tau; z).$$

[Notice that since the reductions are polynomially time bounded, we can keep $R(j)$ in control until such a stage is found. The stage will be essentially double exponential in the uses of the reductions.]

Subcase 2a. No such σ, τ, z exist.

Action. Do nothing: The pending requirement and action remain the same, and we replace $R(j)$ by $R(j - 1)$.

Subcase 2b. We find such a triple σ, τ, z .

Action. $R(j)$ becomes the pending requirement, and the pending action becomes

“Extend $A_{Begin(s)}$ to σ and $B_{Begin(s)}$ to τ .”

At the End of the $\mathcal{R}$ -Cycle. At stage t , we reach the end of the $\mathcal{R}$ -cycle, we actually *perform* the pending action, and we make $Begin(t + 1) = t + 1$.

End of Construction.

It is now routine to verify the construction. If $R(j)$ is the pending requirement at the end of some $\mathcal{R}$ -cycle, then its pending action will be done. Once the requirement is acted for, it is met and, hence, will never again have an effect on a cycle. Therefore, we can see that all the P_e and Q_e are met.

Finally, to see that all the N_e are met, first note that if ever we act for N_e at the end of a cycle, we must meet it by creating a disagreement. Assume that $\Delta(A) = \Lambda(B)$. Go to a stage s_0 where for all $s \geq s_0$ and $j < e$, N_j is never the pending requirement at stage s . It follows that N_j for $j < e$ have *no* further effect on the construction. (This is similar to the proof of Blum's Speedup Theorem [163].) Furthermore, at all stages $t > s_0$, if t is the beginning of a new cycle, then $\text{Begin}(t) = t$. But, of course, $\text{Begin}(s) < p(s)$ for such stages. Since we are assuming that N_e does not become a pending action, we know that for all possible incarnations σ and τ of A_t and B_t and all z with $|z| \leq \text{Begin}(t)$, it is the case that

$$\Gamma_e(\sigma; z) = \lambda_e(\tau; z).$$

But then as $\text{Begin}(t) \rightarrow \infty$, and at stage t , we can deduce that $\Gamma_e(A) \in P$ since to compute $\Gamma_e(A; z)$ simply go to the first stage $q > s_0$ and $q > z$. The value of $\Gamma_e(A; z)$ is then $\Gamma_e(\sigma; z)$, where σ is the empty extension of $A_{\text{Begin}(q)}$.

We remark that the speedup technique has the advantage over the delayed diagonalization technique, that it can be used to embed nondistributive lattices into degree structures (Ambros-Spres [20]). It does have the disadvantage that it produces languages that are not elementarily recursive, since there are many nested iterations needed to figure out the final incarnation of the initial segment of a language. Below are some exercises to sharpen the reader's skills in the new techniques introduced in this chapter. The reader unfamiliar with the techniques above should attempt to complete these before tackling the more intricate arguments in the next section.

Historical Notes

The basic diagonalization argument is attributed to Cantor [114]. The applications of this method to classical recursion theory was known to workers in the field in the 1930s such as Post, Kleene, Church, Rosser, Turing, and Gödel. The basic halting problem is often attributed to Turing [449], although it is essentially present in Gödel [245]. The finite extension method is a refinement of the diagonalization technique which grew out of Post's classic article [375] and is apparently due to Post who was terminally ill at the time. The final article [300] was prepared by Kleene. These ideas are now known to be precursors of Cohen's set-theoretic forcing [127]. There are other precursors such as the work of Nerode [362]. The reader is referred to Kunen [315] as a basic text on forcing. Forcing arguments have recently found many applications to structural complexity (see Fortnow [228]). Within a short time after the publication of the ground-breaking Kleene-Post article [300], Post's problem was more or less simultaneously solved by Friedberg in the

United States and Muchnik in Russia [233, 357]. Both authors were students at the time. Almost immediately, people realized the power and fundamental nature of the priority technique these authors introduced. Later, Shoenfield and Sacks [403] introduced the infinite injury method via intricate combinatorial methods rather like infinite “pinball machines.” This model was later suggested by Lerman [329] (and found to be very valuable in certain circumstances). The tree of strategies method has its roots in the seminal articles of Lachlan [317] and Yates [470], where the idea of nested strategies is introduced. The idea has earlier precursors in Friedberg’s e -state construction of a Maximal Set (Friedberg [234]). Tree of strategy arguments are now ubiquitous in modern recursion theory and seem to have first been explicitly introduced in Ladner [320]. They were the key to the $0'''$ so-called “monstrous injury” method of Lachlan [318, 319]. This method was later developed and refined by Harrington and is the mainstay of many modern arguments as the construction “lives” at the level of $\leq_T$. The reader is referred to Soare [431] for more details and for an account of the main lines of development at least until 1987. A recent and very interesting application of the tree of strategy technique in classical complexity theory can be found in [284].

Ladner [321] seems to be the first author to study the structure of recursive sets under reducibilities of bounded complexity. That article introduced many of the basic techniques such as delayed diagonalization. (In fact, one can also find there a primitive form of the speedup technique.) The speedup technique at least in its modern incarnation has its roots in (Blum [62]), but its explicit development is due to Ambos-Spies [20]. (See Exercise 17.) We remark that the speedup technique has been used for infinite injury arguments in [126], [161], and [421]. Of course, there had been earlier studies in structural complexity such as those by Savitch [408], Hartmanis and Stearns [264], and Blum [62]. These articles studied complexity classes and established many of the fundamental results concerning these objects. We refer the reader to Odifreddi [365]. We do not study abstract complexity classes here. Baker et al. introduced the controversial subject of oracles in [48]. Oracles have had an uneasy relationship with complexity theorists ever since their introduction. The primary use of oracles comes from the observation that they show us that “techniques that *relativise* are not sufficient to solve problem Q ” where Q for instance, is $P = NP$. The trouble is that we find it hard to know what the statement *actually means*. This is especially true in view of the recent nonrelativising results such as $IP = PSPACE$ of Shamir [418]. We refer the reader to [124] and [224] for discussions on the role of relativisation in complexity theory. We also refer the reader to [45] for an excellent introduction to structural complexity.

Exercises 19.1

- 19.1.1.** (Folklore, after Post) Prove that a language L is recursively enumerable if L is the domain of a partial recursive function if L is finite or the range of an injective partial recursive function.

19.1.2. (Rice [386], Rice's Theorem) An *index set* I is a set such that for all partial recursive functions φ_e and φ_j , if $\varphi_e = \varphi_j$ and $e \in I$ then $j \in I$. Prove Rice's Theorem: An index set is recursive iff $I = \emptyset$ or $I = \Sigma^*$.

[Hint: Suppose not. Let I be a nontrivial index set. Without loss of generality, we may assume that I contains a function φ_e that is not everywhere undefined and that the everywhere undefined function is a member of $\bar{I}$. Consider the partial recursive function defined via $\psi(x, y) = \varphi_e(y)$ if $\varphi_x(x) \downarrow$ and $\psi(x, y) \uparrow$ otherwise. Now by the s-m-n theorem, there is a recursive family of partial recursive functions $\varphi_{s(x)}$ such that for all y , $\varphi_{s(x)}(y) = \psi(x, y)$. Now, use the fact that I is recursive to get a decision procedure for the halting problem using the family $\varphi_{s(x)}$.]

19.1.3. (Folklore) (i) Prove that the index set (see the question above) $\text{Tot} = \{e : \varphi_e \text{ is total}\}$ is not recursively enumerable.

(ii) Prove that the index set $\text{Inf} = \{e : \varphi_e \text{ has infinite domain}\}$ is neither recursively enumerable nor corecursively enumerable.

(iii) Prove that Tot and Inf are Π_2 complete.

[Hint: (i) and (ii) use diagonalization. For (iii), for Tot , say, let L be a Π_2 language. Thus, there is a recursive relation such that for all x , $x \in L$ iff $\forall y \exists t R(x, y, t)$. Build a recursive family of partial recursive functions $f_e : e \in \mathbb{N}$, (via, say, the s-m-n theorem) such that for all x , f_x is total iff $x \in L$. To do this, allow $f_x(y) \downarrow$ only when some t is found so that $R(x, y, t)$ holds.]

19.1.4. (Folklore) Use the finite extension method to construct languages A and B below $\mathbf{0}'$ such that $A \equiv_T B$ yet $A \not\leq_m B$ and $B \not\leq_m A$.

[Hint: To get $A \equiv_T B$, use the following reduction. For x a power of 4, we have $x \in A$ iff $x \in B$ or $2x \in B$ and for x a power of 25, we have $x \in B$ iff $x \in A$ or $5x \in A$. To make $A \not\leq_m B$, meet the requirements

$$R_e : \varphi_e \text{ is not a reduction with } A \leq_m B \text{ via } \varphi_e.$$

We meet R_e by using some witness x which is a power of 4 and seeing if $\varphi_e(x) \downarrow$. Then, one of x or $2x$ will be used to diagonalize against φ_e since at most one of them can be $\varphi_e(x)$.]

19.1.5. (Kleene and Post, more or less) A language A is called *low* if $A' (= \{e : \Phi_e(A; e) \downarrow\})$ is recursive in $\mathbf{0}'$. Construct a nonrecursive low language.

[Hint: Use the finite extension method. Build A in stages to meet the requirements below.

$$R_e : A \neq W_e (\text{where } W_e \text{ denotes the } e\text{-th recursively enumerable set})$$

at odd stages. At even stages $2e$, try to force $\Phi_e(A_{2e}; e) \downarrow$ by searching for some τ extending A_{2e-1} with $\Phi_e(\tau; e) \downarrow$. Keep the construction recursive in $\mathbf{0}'$.]

19.1.6. (Jockusch and Posner) A language L is called *1-generic* if for each recursively enumerable set of strings V_e , one of the following pertains:

(i) There is some $\sigma \in V_e$ which is an initial segment of L . (Here, V_e

denotes the e -th recursively enumerable set of strings.)

(ii) There is some initial segment τ of L such that for all $\gamma \in V_e$, τ is not an initial segment of γ .

Use the finite extension method to construct a 1-generic language below $\mathbf{0}'$.

[Hint: Meet the requirements

$$R_e : (\exists \sigma \in V_e)[\sigma < L] \vee (\exists \tau < L)(\forall \gamma \in V_e)[\tau \not\prec \gamma].$$

Do so by attending R_e at stage e . See if there is a σ in V_e with σ extending L_e .

If not, then use L_e to play the role of τ .]

19.1.7. (Baker, Gill, and Solovay [48]) Construct a recursive oracle A with $NP^A \cap coNP^A \neq P^A$.

[Hint: Construct A and an auxiliary set B where $y \in B$ iff $\forall z[|z| = 2|y| \rightarrow z \in A]$ iff $\exists q[|q| = 2|y| + 1 \wedge q \in A]$. Then, $B \in NP^A \cap coNP^A$. Now, meet the same diagonalization requirements as in the proof that there is an oracle with $NP^A \neq P^A$. Care must be exercised to make the two iff statements compatible. Note that if $|y|$ is chosen to be long enough whenever z with $|z| = 2|y|$ is queried, we can add z to an incarnation of A we build without adding *all* the length $2|y|$ strings.]

19.1.8. (Baker, Gill, and Solovay [48]) Construct an oracle A with $NP^A = P^A$.

[Hint: Consider the language

$$K_p^A = \{ \langle x, e, 1^n \rangle : \text{some computation of } M_e \text{ (the } e\text{th nondeterministic Turing machine) halts in } \leq n \text{ steps.} \}$$

Then, K_p^A is automatically NP^A -complete for any A . Now, construct recursively any language A so that $A = K_p^A$.]

19.1.9. Provide the details of the case that $\text{req}(s_0) = Q_e$ in the proof of Ladner's Density Theorem.

19.1.10. (Friedberg, essentially) Use the priority method to construct a low nonrecursive recursively enumerable set.

[Hint: Construct $A = \cup_s A_s$ in stages to meet

$$P_e : \overline{A} \neq W_e.$$

Here, W_e denotes the e -th recursively enumerable set.

$$N_e : (\exists^\infty s)(\Phi_{e,s}(A_s : e) \downarrow) \Rightarrow \Phi_e(A; e) \downarrow.]$$

19.1.11. (Trakhtenbröt, see Soare [431], VII.2.5) A recursively enumerable language L is called *autoreducible* if there is a Turing procedure Φ such that for all x

$$L(x) = \Phi(L \cup \{x\}; x).$$

That is, determining if x is a member of L can be ascertained from L *without* asking directly “Is $x \in L$?” Examples include complete theories where one asks if “ $\bar{x} \in L$.” The concept is due to Trakhtenbröt. Use a finite injury priority argument to prove that there is a recursively enumerable *nonautoreducible* language.

[Hint: Build L to meet the requirements

$$R_e : (\exists x)[\Phi_e(L \cup \{x\}; x) \neq L(x)].$$

Use followers. For a follower x , wait until

$$\Phi_{e,s}(L_s \cup \{x\}; x) \downarrow = 0 = L_s(x).$$

If such a stage s occurs, put x into L_{s+1} and initialize all lower-priority requirements.]

19.1.12. (Sacks [402]) The following exercise is an example of a finite injury argument of the unbounded type.

(i) Let A be a recursively enumerable nonrecursive set. Prove that there exist recursively enumerable disjoint sets A_1 and A_2 with $A = A_1 \cup A_2$ with A_1 Turing incomparable with A_2 . Indeed, given any nonrecursive recursively enumerable language C , we can ensure that $A_i \not\leq_T C$ for $i = 1, 2$.

(ii) Deduce the following result of Friedberg [233]: There is no minimal recursively enumerable degree.

(Hint: Let $C = \cup_s C_s$. Build $A_i = \cup A_{i,s}$ in stages by a priority argument to meet the requirements below:

$$R_{i,e} : \Phi_e(A_i) \neq C.$$

At every stage, we must put any number y entering $A_{s+1} - A_s$ into exactly one of $A_{1,s+1} - A_{1,s}$ or $A_{2,s+1} - A_{2,s}$. This causes $A = A_1 \cup A_2$ with A_1 and A_2 disjoint. To meet $R_{i,e}$, we define the *length of agreement function*

$$\ell^i(e, s) = \max\{x : \forall y < x [\Phi_{e,s}(A_{i,s}; y) = C(y)]\}.$$

The idea is that once $\ell^i(e, s) > t$ for any t , we *try to preserve* $\ell^i(e, s') > t$ for all stages $s' > s$. The way we do this preservation is to put numbers entering $A_{s+1} - A_s$ below the use of the $\ell^i(e, s)$ computations into $A_{i-1,s+1}$ and *not* into $A_{i,s}$. Exactly into which set we put y entering $A_{s+1} - A_s$ is determined by priorities. Argue that once $R_{i,e}$ has priority, it will direct all elements y below the use of $\ell^i(e, s)$ into $A_{1-i,s}$; hence, if we suppose that we *fail* to meet $R_{e,i}$ so that $\ell^i(e, s) \rightarrow \infty$, we can deduce that C is recursive, a contradiction.]

19.1.13. (Downey and Welch [185], Ambos-Spies [21]) Modify the construction of a minimal pair (Theorem 19.8) to construct a recursively enumerable nonrecursive set A , such that for all recursively enumerable sets A_1 and A_2 , if A_1 and A_2 are disjoint and $A_1 \cup A_2 = A$, then the degrees of A_1 and A_2 form a minimal pair.

19.1.14. (Thickness Lemma, Shoenfield) A recursively enumerable set A is called *piecewise recursive* if $A = \bigcup_e A^{(e)}$ with $A^{(e)} \subseteq N^{(e)} =_{\text{def}} \{\langle x, e \rangle : e \in \mathbb{N}\}$, and such that for all e , $A^{(e)}$ is recursive. Let A be a recursively enumerable piecewise recursive set, and let C be a nonrecursive recursively enumerable set. Furthermore, suppose that for all e , $A^{(e)}$ is an initial segment of $N^{(e)}$, so that it is either $\{\langle 0, e \rangle, \dots, \langle n(e), e \rangle\}$, or $N^{(e)}$. Use the infinite injury method to construct a recursively enumerable subset B of A such that for all e , $|A^{(e)} - B^{(e)}| < \infty$, and $C \not\leq_T B$. B is called a *thick* subset of A .

[Hint: Meet the requirements $R_e : |A^{(e)} - B^{(e)}| < \infty$ and $N_e : \Phi(B) \neq C$. To meet the R_e , you must make sure that almost all of the e -th column of A gets into B . To meet the N_e , use the Sacks strategy of Exercise 12. The only problem is that infinitely often one can try to preserve a computation $\Phi_{e,s}(B_s; x) = C_s(x)$ only later to have some $\langle z, i \rangle$ with $i < e$ and $\langle z, i \rangle < u(\Phi_{e,s}(B_s; x))$ enter $A_i - A_s$ and hence get into B if we are to meet R_e . (It is only reasonable by priorities that N_e can control $A^{(i)}$ iff $i > e$.) The solution to this dilemma is to have *two* or more versions of N_e *guessing* whether $A^{(i)}$ is finite or not (i.e., for each $i \leq e$), and “not believing” a computation unless it looks correct according to the relevant guess.]

19.1.15. (Downey and Jockusch [183]) Use the infinite injury method to construct an m -topped incomplete nonrecursive recursively enumerable degree; that is, construct a recursively enumerable nonrecursive set A such that for all recursively enumerable sets $C \leq_T A$, $C \leq_m A$.

[Hint: Construct A and an auxiliary recursively enumerable set C . Meet the requirements $P_e : \bar{A} \neq W_e$, $Q_e : \Phi_e(A) \neq C$, and

$$R_{i,j} : \Phi_i(A) = W_j \Rightarrow W_j \leq_m A.$$

We meet P_e and Q_e by Friedberg Strategies. To meet the $R_{i,j}$, proceed as follows. Because we are only interested in $\langle i, j \rangle$ for which $\Phi_i(A) = W_j$, we will suppose that once $\ell(i, j, s) > x$ (where $\ell(i, j, s) = \max\{x : \forall y < x (\Phi_{i,s}(A_s; y) = W_{j,s}(y))\}$), for the first time we will define a value $f(i, j, x)$, a large fresh number, and promise that

$$x \in W_j \text{ iff } f(i, j, x) \in A.$$

Thus, at expansionary stages, if we see $x \in W_{j,s}$, we will put $f(i, j, s)$ into A_t at some stage $t \geq s$. This interferes with the meeting of the Q_e because we might try to preserve $\Phi_{e,s}(A_s; z)$ to cause a disagreement $\Phi_{e,s}(A_s; z) \neq C_s(z)$, only to later be forced to kill this disagreement by putting some $f(i, j, x)$ into A below $u(\Phi_{e,s}(A_s; z))$ (since $\langle i, j \rangle < e$). As usual, since R_e is infinitary, this cycle could reoccur infinitely often, causing us to *never* meet Q_e . The solution is for the requirements to guess whether there are infinitely many $\langle i, j \rangle$ expansionary stages, and if Q_e is guessing that R_e is infinitary, it should only act when it gets a “believable” computation: one where all the coding markers $f(\sigma, x)$ of higher priority are in their final positions.]

19.1.16. (Ladner [321]) Use the delayed diagonalization method to prove that there exist *minimal pairs* of polynomial time languages.

[Hint: Meet the requirements

$$P_e : B \neq \Gamma_e(\emptyset),$$

$$R_e : A \neq \Gamma_e(\emptyset),$$

$$N_e : \Delta_e(A) = \Lambda_e(B) = f \Rightarrow f \in P.$$

One can meet P_e (and R_e) by followers. Pick a follower x and wait for $\Gamma_e(\emptyset; x)$ to give a value; then make B different from this value. In conjunction with the N_e , this uses delayed diagonalization, as we will see. The interesting requirements are the N_e . The key idea is that if A is empty, then $\Delta_e(A) = \Lambda_e(\emptyset)$ must be in P . So we can make $B \notin P$ and still meet all the N_e at the expense of making $A = \emptyset$. The same is true with the roles of A and B reversed. So, the idea is that while we want to meet a P_e , we make A empty *long enough* that we get to win P_e . Note that once we put x into B , say, if we wait for an exponential number of steps, making both A and B empty extensions, we will be able to switch from making A empty to making B empty and move on to the next R_e .]

19.1.17. (Ambos-Spies [20]) Historically, this is Ambos-Spies' first application of the speedup technique. Use the speedup technique to show that if A is a recursive language not in P , then there is a recursive language B not in P forming a minimal pair with A .

19.2 Results

Now, we turn to the analysis of our parameterized languages. So far, our results have only needed the working definition of parameterized reducibility, Definition 9.3, which was of the form

$$\langle x, k \rangle \in L \text{ iff } \langle x', k' \rangle \in L'.$$

Now, we will actually need to carefully differentiate between the technical differences in the various flavors of uniformity of the various reducibilities. Thus, we recall the relevant definitions below.

Definition 19.17 (Uniform Fixed Parameter Reducibility) Let A and B be parameterized problems. We say that A is *uniformly fixed parameter-reducible* to B if there is an oracle procedure Φ , a constant α , and an arbitrary function $f : \mathbb{N} \mapsto \mathbb{N}$ such that

- (a) the running time of $\Phi(B; \langle x, k \rangle)$ is at most $f(k)|x|^\alpha$,
- (b) on input $\langle x, k \rangle$, Φ only asks oracle questions of $B^{(f(k))}$, where

$$B^{(f(k))} = \bigcup_{j \leq f(k)} B_j = \{\langle x, j \rangle : j \leq f(k) \& \langle x, j \rangle \in B\},$$

- (c) $\Phi(B) = A$.

If A is uniformly fixed parameter reducible to B , we write $A \leq_T^u B$. Where appropriate we may say that $A \leq_T^u B$ via f . If the reduction is many:1 (an *m-reduction*), we will write $A \leq_m^u B$.

Definition 19.18 (Strongly Uniform Reducibility) Let A and B be parameterized problems. We say that A is *strongly uniformly fixed-parameter-reducible* to B if $A \leq_T^s B$ via f where f is recursive. We write $A \leq_T^s B$ in this case.

Definition 19.19 (Nonuniform Reducibility) Let A and B be parameterized problems. We say that A is *nonuniformly fixed-parameter-reducible* to B if there is a constant α , a function $f : \mathbb{N} \mapsto \mathbb{N}$, and a collection of procedures $\{\Phi_k : k \in \mathbb{N}\}$ such that $\Phi_k(B^{(f(k))}) = A_k$ for each $k \in \mathbb{N}$, and the running time of Φ_k is at most $f(k)|x|^\alpha$. Here, we write $A \leq_T^n B$.

We will need the following technical result.

Lemma 19.20 ([168]) (i) Suppose that $A \leq_T^u B$ (or $A \leq_m^u B$) with A and B recursive. Then, there exists a recursively enumerable [i.e., Δ_2 with $f(x, s+1) \neq f(x, s)$ implying that $f(x, s) \neq f(x, t)$ for all $t > s$] function f such that $A \leq_T^u B$ (resp. $A \leq_m^u B$) via f .

(ii) Suppose that $A \leq_T^n B$ (or $A \leq_m^n B$) with A and B recursive. Then there exists a recursively enumerable function f such that $A \leq_T^n B$ (resp. $A \leq_m^n B$) via f .

Proof. We do (i) for $\leq_T^u$, the others being essentially similar and left to the reader (Exercise 2). Suppose that A and B are recursive and $A \leq_T^u B$. Then, there is a procedure Φ , a constant α , and a function g so that for all k

(*) $(\forall z)(\langle z, k \rangle \in A \text{ iff } \Phi(B^{(g(k))}; \langle z, k \rangle) = 1 \text{ and runs in time } \leq g(k)|z|^\alpha)$.

We claim that $\emptyset'$ can compute a value that works in place of $g(k)$ in the above; that is, for each k , $\emptyset'$ can compute $m = m(k)$ satisfying (*) with m in place of g . Call this (*). The reason is that the expression in the scope of the universal quantifier is recursive and hence the whole expression is $\leq_T \emptyset$. This uses essentially the same reasoning as the proof of Theorem 19.5. For completeness, we remind the reader that for each pair $\langle n, k \rangle$, we can enumerate a partial recursive function $\psi_{\langle n, k \rangle} = \varphi_{h(n, k)}$ whose index $h(n, k)$ is given by the s-m-n theorem with $\text{dom } \psi_{\langle n, k \rangle}$ equal to $\mathbb{N}$ if there is some z with $\langle z, k \rangle \notin A$ but $\Phi(B^{(n)}; \langle z, k \rangle) = 1$, or $\langle z, k \rangle \in A$ and $\Phi(B^{(n)}; \langle z, k \rangle) = 0$, or $\Phi(B^{(n)}; \langle z, k \rangle)$ not running in time $n|z|^\alpha$; and we have $\psi_{\langle n, k \rangle}$, the empty function otherwise. Now, $\emptyset'$ can decide if $\langle h(n, k), h(n, k) \rangle \in \emptyset'$ and, hence, can compute the least n such that $\text{dom } \psi_{\langle n, k \rangle} = \emptyset$. For such an n , we have that

$$A_k = \Phi(B^{(n)}) \text{ in running time } n|z|^\alpha.$$

Now, it is clear that we can approximate such an $n(k)$ via a function where values only increase and, hence, we can take an m to perform the role of g that is recursively enumerable.

Using Lemma 19.20, it is not difficult to prove that the three definitions above define differing reducibilities.

Theorem 19.21 ([168]) (i) $FPT(\leq_T^u) \subset FPT(\leq_T^n)$ even for recursive sets.
(ii) $FPT(\leq_T^s) \subset FPT(\leq_T^u)$.

Proof. (ii) We prove this by a simple diagonalization argument. Let $\{(\Phi_e, \varphi_e) : e \in \mathbb{N}\}$ denote an enumeration of all pairs consisting of a procedure and a partial recursive function. We shall satisfy the requirements for $e \in \mathbb{N}$:

$R_{\langle e, n \rangle}$: Either φ_e is not total, or
for some k, x , $\Phi_e(\emptyset; \langle x, k \rangle) \neq A(\langle x, k \rangle)$, or
 $\Phi_e(\emptyset; \langle x, k \rangle)$ does not run in time $\varphi_e(|k|)|x|^n$.

Additionally, we must ensure that $A \in FPT(\leq_T^u)$. We devote $A_{\langle e, n \rangle}$ to meeting $R_{\langle e, n \rangle}$. We ensure that at most one element of row $\langle e, n \rangle$ enters A , and if z enters $A_{\langle e, n \rangle}$, then z is of the form $\langle 1^m, \langle e, n \rangle \rangle$ for some m .

We shall build A in stages. At stage s , we decide the fate of $\langle 1^s, k \rangle$ for all $k \in \mathbb{N}$. At stage s , the construction runs as follows:

For each $\langle e, n \rangle \leq s$, if $R_{\langle e, n \rangle}$ is not yet declared satisfied, compute s steps in the computation of $\varphi_e(\langle e, n \rangle)$. (Call this $\varphi_{e,s}(\langle e, n \rangle)$.) If $\varphi_{e,s}(\langle e, n \rangle) \uparrow$, do nothing for $\langle e, n \rangle$ at this stage, keeping $m(\langle e, n \rangle, s) = m(\langle e, n \rangle, s-1)$. If $\varphi_{e,s}(\langle e, n \rangle) \downarrow$, declare $R_{\langle e, n \rangle}$ as satisfied and perform the following diagonalization for $\langle e, n \rangle$. Run $\Phi_e(\emptyset; \langle 1^s, \langle e, n \rangle \rangle)$ for $\varphi_e(\langle e, n \rangle)s^n$ many steps. If the procedure does not halt in this many steps, we need do nothing, since the running time is wrong. If $\Phi_e(\emptyset; \langle 1^s, \langle e, n \rangle \rangle) \downarrow$ in $\varphi_e(\langle e, n \rangle)s^n$ many steps, set

$$A(\langle 1^s, \langle e, n \rangle \rangle) = 1 - \Phi_e(\emptyset; \langle 1^s, \langle e, n \rangle \rangle).$$

In either case, set $m(\langle e, n \rangle, 1^s) = 2\varphi_e(\langle e, n \rangle)(s+1)$. It is clear that the diagonalization succeeds, ensuring that $A \neq \Phi_e(\emptyset)$. Note that $A \in FPT(\leq_T^u)$ since for any $k, \langle z, k \rangle \in A$ iff z is of the form 1^t and $\langle 1^t, k \rangle$ is put into A at stage t . This can be decided in time $m(k, t) \cdot t$, and since $m(k, t) \neq m(k, t+1)$ at most once, we see that $A \in FPT(\leq_T^u)$.

(i) Again, we use a simple diagonalization argument. Now, we need a family of reductions $\{\Delta_k : k \in \mathbb{N}\}$ with $\Delta_k(\emptyset)$ computing A_k . By the Limit Lemma we need to meet the following requirements:

$R_{\langle e, n, c \rangle}$: Either $\lim_s \varphi(\langle e, n \rangle, s)$ fails to exist, or
 $\Phi_e(\emptyset) \neq A_{\langle e, n \rangle}$, or
it does not run in time $\varphi_e(\langle e, n \rangle)|z|^c$.

Here, we are working with pairs consisting of a procedure and a binary recursive function. We are denoting by $\varphi(p)$ the value of $\lim_s \varphi(p, s)$ if it exists. We shall additionally, and without loss of generality, assume that φ_e is nondecreasing in both variables where defined. (This follows by the proof of Lemma 19.20.) In the construction to follow, a value can be used for $\langle e, n, c \rangle$.

At stage s , if $R_{\langle e, n, c \rangle}$ is not as yet declared satisfied and $\langle e, n, c \rangle \leq s$, find the least unused $j \leq s$, if any, such that $j = \varphi_{e,s}(\langle e, n, c \rangle, t) \downarrow$ for some $t \leq s$. If either $\varphi_{e,s}(\langle e, n, c \rangle, t) \uparrow$ for all $t \leq s$ or there is no unused j , do nothing. If j and, hence, t exist, declare j as used. Now, compute js^n steps in the computation of $\Phi_e(\emptyset; \langle 1^s, \langle e, n, c \rangle \rangle)$. If this does not halt, do nothing else. If $\Phi_j(\emptyset; \langle 1^s, \langle j, n \rangle \rangle)$

halts in $\leq js^c$ steps, win by setting

$$A(\langle 1^s, \langle j, n, c \rangle \rangle) = 1 - \Phi_j(\emptyset; \langle 1^s, \langle j, n, c \rangle \rangle).$$

It is clear that A is recursive. Now Δ_k is one of the following two reductions: Either $\Delta_k = \Psi$ which, on input $\langle x, k \rangle$ says that $\langle x, k \rangle \notin A$, or $\Delta_k = \Delta$ which, on input $\langle x, k \rangle$ computes s where R_k is satisfied, and then has $\langle y, k \rangle \in A$ iff y is of the form 1^s and $1 - \Phi_{e(k)}(\emptyset; \langle 1^s, k \rangle) = 1$ and $\langle y, k \rangle \notin A$ otherwise. Note that the algorithm runs in constant time, so that $A \in FPT(\leq_T^n)$.

We can immediately see a difference between the arithmetical complexities of the parameterized and classical reducibilities. Classical polynomial time T - and m -reducibilities are both Σ_2 since, for instance, $L \leq_T^p L'$ iff

$$\exists e \forall x [x \in L \text{ iff } \Gamma_e(L; x) = 1].$$

In our situation, we have the following.

Theorem 19.22 (Cholak and Downey [126]) *Let A be a computable set. Then for $q \in \{m, T\}$,*

- (i) $\{e : W_e \leq_q^s A\}$ is a Σ_3^0 set.
- (ii) $\{e : W_e \leq_q^u A\}$ is a Σ_4^0 set.
- (iii) $\{e : W_e \text{ computable and } W_e \leq_q^n A\}$ is a Σ_4^0 set.

Proof. We prove (ii). The others are left to the reader. (Exercise 5) By the definition, we see that

$$W \leq_q^u A \text{ iff } \exists e \forall k \exists u \forall z (\Phi_e(A^{(u)}; \langle z, k \rangle) = W(\langle z, k \rangle) \text{ in time } u|z|^n).$$

This is clearly Σ_4^0 . □.

We can improve on this result as follows by showing that the relevant index sets are as complicated as they can be.

Theorem 19.23 *Suppose that A is computable. Let $q \in \{m, T\}$. Then the index sets*

- (i) $\{e : W_e \leq_q^s A\}$ and $\{e : W_e \equiv_q^s A\}$ are Σ_3^0 complete.
- (ii) $\{e : W_e \leq_q^u A\}$, $\{e : W_e \equiv_q^u A\}$, $\{e : W_e \text{ computable and } \leq_q^n A\}$ and $\{e : W_e \text{ computable and } \equiv_q^n A\}$ are Σ_4^0 complete.

Proof (i). We prove that $\{e : W_e \leq_q^s A\}$ is Σ_3^0 complete, the remainder of (i) being left to the reader. (Exercise 6.) For ease of notation we shall first prove that $FPT(\leq_T^s)$ is Σ_3^0 complete. We will then point out the easy modifications to obtain the result for A computable. Thus let B be a Σ_3^0 set. By definition, there is a computable relation Q such that for all e

$$e \in B \text{ iff } \exists x \forall y \exists z Q(x, y, z, e).$$

We will build a collection $\mathcal{C}$ of sets $V_e = W_{f(e)}$ whose indices are given by the s-m-n theorem, such that we meet the requirements

$$R_e : V_e \leq_T^s \emptyset \text{ iff } e \in B.$$

Actually the fundamental strategy is to try to ensure that $V_e \not\leq_T^s \emptyset$, and only *fail* to achieve this goal if $e \in B$.

Now, $V_e \leq_T^s \emptyset$ means that there should be some reduction triple consisting of a reduction (Φ_m) , a total computable function φ_m and a positive integer n , such that for all $\langle w, k \rangle$ $\Phi_m(\emptyset; \langle w, k \rangle) = V_e(\langle w, k \rangle)$, and the running time of the computation is bounded by $\varphi(k)|w|^n$. Hence, provided that $e \notin B$, we will meet all the “subrequirements” below.

$R_{e,m,n}$: If $e \notin B$ then

either φ_m is not total, or

for some k, w , $\Phi_m(\emptyset; \langle w, k \rangle) \neq V_e(\langle w, k \rangle)$, or

$\Phi_m(\emptyset; \langle w, k \rangle)$ does not run in time $\leq \varphi_m(k)|w|^n$.

The reader should compare the requirements above with the $R_{\langle e, n \rangle}$ of Theorem 19.21. We will suppose that each requirement occurs infinitely often. It is easiest to conceive of the construction as follows. For each potential “witness” x to the outer existential quantifier used to show that $e \in B$, we will have a *Control Device* $\mathcal{C}(x)$. The idea of the control device is that it tells us when to attempt to meet the subrequirements.

The fundamental plan is to pursue an elaboration of the strategy of Theorem 19.21. For the present situation, our strategy is the following:

We build V_e in stages. At stage s we decide the fate of $\langle 1^s, k \rangle$ for all $k \in \mathbb{N}$. (If $k > s$ then $\langle 1^s, k \rangle \notin V_e$ by fiat.) The reader should recall that in Theorem 19.21, we devoted the j -th slice of A to meeting R_j . Here V_e has the role of A and where j will be some $\langle e, m, n \rangle$. In the present proof, this assignment will be replaced by a more flexible arrangement where *slice*(j, s) denotes the current slice devoted to satisfying R_j . Modulo the control device (to be described) (so that we can attack the requirement freely at each stage), at stage s , the construction of V_e runs as follows:

For each $\langle e, m, n \rangle \leq s$, if $R_{e,m,n}$ is not yet declared satisfied, compute s steps in the computation of $\varphi_m(\text{slice}(\langle e, m, n \rangle, s))$. (Call this $\varphi_{m,s}(\text{slice}(\langle e, m, n \rangle, s))$.) If $\varphi_{m,s}(\text{slice}(\langle e, m, n \rangle, s)) \uparrow$ do nothing for $\langle e, m, n \rangle$ at this stage. If $\varphi_{m,s}(\text{slice}(\langle e, m, n \rangle, s)) \downarrow$ declare $R_{e,m,n}$ as satisfied and perform the following diagonalization for $\langle e, m, n \rangle$. Run $\Phi_m(\emptyset; \langle 1^s, \text{slice}(\langle e, m, n \rangle, s) \rangle)$ for $\varphi_m(\text{slice}(\langle e, m, n \rangle, s))s^n$ many steps. If this does not halt in this many steps we need do nothing since the running time is wrong. If $\Phi_m(\emptyset; \langle 1^s, \text{slice}(\langle e, m, n \rangle, s) \rangle) \downarrow$ in $\varphi_m(\text{slice}(\langle e, i, n \rangle, s))s^n$ or fewer steps set

$$V_e(\langle 1^s, \text{slice}(\langle e, i, n \rangle, s) \rangle) = 1 - \Phi_m(\emptyset; \langle 1^s, \text{slice}(\langle e, m, n \rangle, s) \rangle).$$

The enumeration of B controls the above in the following way. We say that x is *s-confirmed* (for e) at stage t if

$$\forall y \leq s \exists z \leq t Q(e, x, y, z).$$

We say that $\mathcal{C}(x)$ asserts control of $R_{e,m,n}$ at stage t if

- (i) x is s -confirmed at stage t ,
- (ii) x was not s confirmed at stage $t - 1$, and
- (ii) $x < \langle e, m, n \rangle < s$.

We remark that the idea here is that if the x is a least correct witness to e being in B then the inner Π_2 part of the definition of B (namely $\forall y \exists z Q(x, y, z, e)$) will “appear correct” (i.e. be confirmed) infinitely often. The principle idea is that such confirmation will allow us to move the slices we are dynamically assigning to the satisfaction of the requirements into higher slices. If e is really in B then, for all but a finite number of rows, we will see that we will kick all of the witness rows to infinity. On the other hand, if $e \notin B$ then eventually each slice, for example, $slice(j, s)$, comes to a resting place, and this is where we will get to meet R_j . More formally, we have the following.

The Full Construction: Incorporation of x -Control. Now we will modify the construction to incorporate $\mathcal{C}(x)$. At each stage t for each $x, e \leq t$, we first see if x is s confirmed for e for some $s \leq t$. If this is the case, we say that $\mathcal{C}(x)$ asserts control of various $R_{e,m,n}$. In particular, for each such $\langle e, m, n \rangle$, we will reset $slice(\langle e, m, n \rangle, t + 1)$ to be large and fresh. Specifically, for all $\langle m', n' \rangle \geq \langle m, n \rangle$, set $slice(\langle e, m', n' \rangle, t + 1) = slice(\langle e, m', n' \rangle, t) + t + 1$. Also we initialize $R_{e,m',n'}$.

End of Construction

Now the point of all the above is the following. One can easily establish by induction that

- (i) $e \in B$ implies that for all but finitely many m, n , $R_{e,m,n}$ is initialized infinitely often and hence $slice(\langle e, m, n \rangle, s) \rightarrow \infty$.
- (ii) $e \notin B$ implies that for all m, n , $\lim_s slice(\langle e, m, n \rangle, s)$ exists.

To see that (i) holds, if $e \in B$ there is some x with $\forall y \exists z Q(e, x, y, z)$. Now such an x will be confirmed for e infinitely often. Hence it will assert control of any $R_{e,m,n}$ for $\langle m, n \rangle > x$ infinitely often, driving $slice(\langle e, m, n \rangle, s)$ to ∞ . On the other hand, if $e \notin B$, then for each x , there is some y such that for all z , it is never the case that $Q(e, x, y, z)$. It follows that x will only be confirmed for e until it gets stuck on y . Thereafter, it can no longer initialize any $R_{e,m,n}$. Since there are only j numbers $x < j$ and only $x < \langle m, n \rangle$ can initialize $R_{e,m,n}$, it follows that there is a stage $t_{m,n}$ such that for all $t \geq t_{m,n}$, $R_{e,m,n}$ is not initialized at stage t . Since $slice(\langle e, m, n \rangle, t)$ is only reset when $R_{e,m,n}$ is initialized, it follows that $slice(\langle e, m, n \rangle, t)$ comes to a limit.

Finally we see that we meet all the requirements. If $e \notin B$ then a simple induction shows that, for $j = \langle e, m, n \rangle$, we will meet R_j in row $slice(j) = \lim_s slice(j, s)$.

If $e \in B$ then we argue as follows. Let x be least such that $\mathcal{C}(x)$ asserts control of some $R_{e,m,n}$ infinitely often. Let s_0 be a stage such that for all $t > s_0$, for all $y < x$, $\mathcal{C}(y)$ does not assert control of $R_{e,m,n}$ at stage t . Thus for all $\langle y, d \rangle < x$ for all $slice(\langle e, y, d \rangle, s_0) = slice(\langle e, y, d \rangle)$. For all j we know that R_j acts on row z at most once. Let $s_1 > s_0$ be a stage such that for all $\langle m', n' \rangle < x$, and for all $t > s_1$, $R_{e,m',n'}$ does not act at stage t . Then we know that for all $j < slice(x - 1)$, and all p ,

$$V_{e,s}(\langle p, j \rangle) = V_e(\langle p, j \rangle).$$

(Indeed we remark that for all p with $|p| > s_1$, $V_e(p) = 0$.) We now give a $FPT(\leq_T^s)$ decision procedure for determining if $u = \langle v, k \rangle \in V_e$.

If $k < \text{slice}(x - 1)$ go to stage s_1 and see if $u \in V_e$. If $k \geq \text{slice}(x - 1) + 1$ go to the stage $s = s(k) > s_1$ where $C(x)$ has asserted control of e k times. Then for all $t > s$, for all $\langle m, n \rangle \geq x$, $\text{slice}(\langle e, m, n \rangle, t) > k$. Hence for all $t > s$ and for all $\langle e, m, n \rangle$, $\text{slice}(\langle e, m, n \rangle, t) \neq k$. It follows that $u \in V_e$ iff $u \in V_{e,s}$. Notice that the computation of the stage $s(k)$ depends on k alone, and hence the procedure is independent of v . This is why the procedure is FPT in constant time by the slice. This concludes our proof that $FPT(\leq_T^s)$ is Σ_3^0 complete.

To modify the construction for the more general case proving that $\{e : W_e \leq_T^s\}$ is also Σ_3^0 complete, the control device is the same but we must modify the diagonalization. Basically, one replaces the oracle $\emptyset$ by $A^{(\varphi_{m,s}(\text{slice}(\langle e, m, n \rangle, s)))}$. Since A is computable, the construction either make V_e in $FPT(\leq_T^s)$, or produce a language $V_e \not\leq_T^s A$.

(ii) The other result we will prove is that $\{e : W_e \leq_T^u A\}$ is Σ_4^0 complete. The remaining completeness results are totally analogous and are left to the reader (Exercise 7). Again we take $A = \emptyset$. This proof is considerably more complex than the previous one and we need to take some care. Actually, this will need the infinite injury method, and uses a priority tree. We use the following representation theorem for Σ_4^0 sets:

Lemma 19.24 (Yates [471]) *The index set $\text{Comp} = \{e : \emptyset \leq_T W_e\}$ of complete computably enumerable sets, is Σ_4^0 complete*

For a proof of Lemma 19.24 see Soare [431], XII §2. In the construction to follow, we will need to approximate Comp as our control device. Thus, we will need to discover which W_e are T -complete. To do this we will need to guess the index of a procedure Φ_i with $\Phi_i(W_e) = K$, with $K = \emptyset'$. To this end, let

$$\ell(e, i, s) = \max\{x : \forall y < x (\Phi_i(W_{e,s}; y) = K_s(y))\}.$$

Now $\ell(e, i, s)$ can exhibit the following behaviour: We can have $\limsup_s \ell(e, i, s) \not\rightarrow \infty$, or we can have $\limsup_s \ell(e, i, s) \rightarrow \infty$. The latter can either mean that $\liminf \ell(e, i, s) \rightarrow \infty$ (so that $\Phi_i(W_e) = K$), or that for some x , $\liminf \ell(e, i, s) = x$, and hence $\Phi_i(W_e; x) \uparrow$. Here we will use the so-called “hat” convention: We assume that if for a computably enumerable set M and procedure Φ_i if $\Phi_{i,s}(M_s; z) \downarrow$, but $z' \leq u(\Phi_{i,s}(M_s; z))$ enters $M_{s+1} - M_s$, then $\Phi_{i,s}(M_{s+1}; z) \uparrow$. The hat convention ensures that if $\Phi_i(M; z) \uparrow$ then $\exists^\infty s (\Phi_{i,s}(M_s; z'') \uparrow)$ for all $z'' \geq z$. The trick is due to Lachlan. (See Soare [431], Ch. VIII.) On the priority tree we will need to guess if $\ell(e, i, s) \rightarrow \infty$. This will be measured at nodes τ , which we refer to as *top* nodes. Such τ nodes have outcomes (i, ∞) and (i, f) as in the minimal pair theorem, Theorem 19.8. The trouble is that we also need to differentiate between infinite and finite $\liminf$ behaviour. This is very difficult to achieve at a single node, and in fact we will decompose the behaviour into infinitely many guesses for the $\liminf$ below the (i, ∞) outcome of the τ node. At such a node trying to figure out such behaviour,

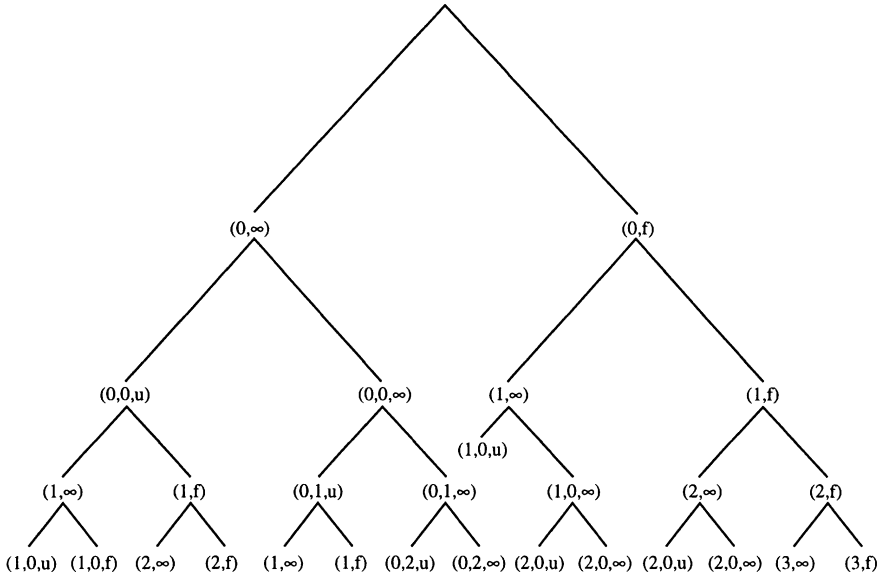


FIGURE 19.3. The Priority Tree for Theorem 19.23

which we refer to as a σ node, we will be considering e, i, x . We will have two outcomes (i, x, ∞) and (i, x, u) , with (i, x, u) to the left of (i, x, ∞) . Thus we have the priority tree given in Figure 19.3.

The outcome (i, x, ∞) is meant to represent the fact that $\exists s \forall t \geq s (\ell(e, i, s) > x)$, whereas the outcome (i, x, u) represents the fact that infinitely often (i, x, ∞) looks correct, and yet later $\ell(e, i, s) \leq x$. (The “u” means “unbounded” here.) Thus this is the outcome that $\limsup \ell(e, i, s) \rightarrow \infty$, and yet (if x is least), $\liminf \ell(e, i, s) = x$.

In the same way we used $\mathcal{C}$ to control the strategies above, here we will use the Priority Tree PT to control our strategies. Note that we will build one version of PT for each e below. (The formal definition can be found in Definition 19.25, following some further motivational comments.)

Before we explain exactly how this control is achieved, let us turn to the actual requirements we will meet. For each e we will again build an auxiliary computable set V_e to try to meet the requirements below.

$R_e: V_e \in FPT(\leq_u^m)$ iff $e \in Comp$.

Since the V_e are all built separately, we will drop the “ e ” when the context is clear. Actually, as with the previous construction we will meet the following requirements.

R : Either $e \in Comp$ and $V \in FPT$ via some witness i
 (Here, of course, by witness i we mean that $\Phi_i(W_e) = K$.)
 or $\forall m, n R_{(e, m, n)}$, where

$R_{\langle e, m, n \rangle}$: Either there exists an m', n' such that
 $\lim_s \varphi_m(\langle e, m', n' \rangle, s) =_{\text{def}} \varphi_m(e, m', n')$ fails to exist, or
 there exist p, w , with
 $\Phi_m(\emptyset, \langle p, w \rangle) \neq V(\langle p, w \rangle)$, or
 it does not run in time $\leq \varphi_m(w)|p|^n$.

Notice that in the above, we have made the replacement of the unary partial computable function $\varphi_m(\cdot)$ by a binary partial computable function $\varphi_m(\cdot, \cdot)$ since in the uniform case we will need to guess the use and constant which will filter through the limit lemma. In fact we will assume that such φ_m are monotone in both variables. Again, we are using the requirements of Theorem 19.21 (ii) in modified form. We attempt to meet the $R_{\langle e, m, n \rangle}$ as we did there, but again modulo the external control this time driven by the Priority Tree. The basic strategy used to meet the $R_{\langle e, m, n \rangle}$ is the following.

At each stage s , if $R_{\langle e, m, n \rangle}$ is not as yet declared satisfied and $\langle e, m, n \rangle \leq s$ find the least unused $p \leq s$, if any, such that $p = \varphi_{m,s}(\langle e, m, n \rangle, t) \downarrow$ for some $t \leq s$. If either $\varphi_{m,s}(\langle e, m, n \rangle, t) \uparrow$ for all $t \leq s$ or there is no unused p do nothing. If p , and hence t , exist declare p as used. Now compute ps^n steps in the computation of $\Phi_m(\emptyset; \langle 1^s, \langle e, m, n \rangle \rangle)$. If this computation does not halt, do nothing else. If $\Phi_m(\emptyset; \langle 1^s, \langle e, m, n \rangle \rangle)$ halts in $\leq ps^n$ steps, win by setting

$$V_e(\langle 1^s, \langle e, m, n \rangle \rangle) = 1 - \Phi_m(\emptyset; \langle 1^s, \langle e, m, n \rangle \rangle).$$

Now again we will need to modify the above so that a version R_σ of $R_{\langle e, m, n \rangle}$ and guessing σ will work on a slice $\text{slice}(\sigma, s)$ instead of a fixed slice $\langle e, m, n \rangle$. Again we will need to argue that if there is no witness i to $e \in \text{Comp}$ and hence $e \notin \text{Comp}$, then for all σ on TP , $\lim_s \text{slice}(\sigma, s) = \text{slice}(\sigma)$ exists. Additionally, if $e \notin \text{Comp}$, we will need to argue that for all $\langle e, m, n \rangle$ there is some $\sigma \subset TP$ devoted to solving $R_{\langle e, m, n \rangle}$. This is done as follows.

Definition 19.25 (The Priority Tree and Attachments)

Step 1. We formally define the priority tree and the attachments as follows. If α is on PT then α is of the form $\widehat{v}(i, x, \infty)$, $\widehat{v}(i, x, u)$, λ , $\widehat{v}(i, f)$, or $\widehat{v}(i, \infty)$. If α is of the form λ , $\widehat{v}(i, f)$, or $\widehat{v}(i, x, u)$ we say that α is a τ node. If α is of the form $\widehat{v}(i, x, \infty)$ or $\widehat{v}(i, \infty)$, we say that α is a σ node. Now we construct PT by induction on the length of nodes.

Case 1. If the node is a τ node then it will form the “top” of an i tree. Let $i(\tau)$ be the least i not yet assigned to any $\tau' \subset \tau$ and assign $i(\tau)$ to τ . (Hence $i(\tau) = i - 1$ in all cases except $\tau = \lambda$.) The outcomes of τ will be $(i(\tau), \infty)$ and $(i(\tau), f)$ with $(i(\tau), \infty) <_L (i(\tau), f)$. Put $\tau \widehat{O}$ on PT for the outcomes O of τ .

Case 2. If the node is a σ node so that it will be devoted to some x for some i , find the longest τ node $\subseteq \sigma$ and define $i(\sigma) = i(\tau)$ and $\tau(\sigma) = \tau$. (τ is called σ 's *top*. Note that $i(\sigma) = i$ in all cases above.) It will be the case that $\tau \widehat{\infty} \subseteq \sigma$. For all γ with $\tau \widehat{\infty} \subseteq \gamma \subset \sigma$ will have $\gamma \widehat{v}(i, x', \infty) \subseteq \sigma$. Let

$$x(\sigma) = \max\{0, x' + 1 : \tau \widehat{\infty} \subseteq \gamma \widehat{v}(i, x', \infty) \subseteq \sigma\}.$$

(In all the specifying cases above it will be the case that $x = \max\{0, x + 1\}$.) Let σ have outcomes $(i, x, u) <_L (i, x, \infty)$. Put $\sigma \hat{=} O$ on PT for each outcome O of σ .

Step 2. Now assign by induction on $\langle e, m, n \rangle$ versions of $R_{\langle e, m, n \rangle}$ to nodes of the form $\gamma = \sigma \hat{=}(i, x, u)$ and $\gamma = \tau \hat{=} f$ on PT . Do this in the obvious way. For each such node γ find the least $\langle e, m, n \rangle$ not attached to any node $\mu \subset \gamma$, and attach $R_{\langle e, m, n \rangle}$ to γ .

As in Theorem 19.8 we will need the notion of an α -stage.

Definition 19.26 (a) We define the notions α -stage, $m\ell(\alpha, s)$, and α -expansionary by induction on $|\alpha|$.

(i) Every stage s is a λ -stage.

(ii) **Case 1.** Suppose that s is a β -stage with β a top node devoted to solving the problem of whether $\ell(\beta, s) = \ell(i(\beta), s) \rightarrow \infty$. Define

$$m\ell(\beta, s) = \max\{0, \ell(\beta, t) : t \text{ is a } \beta\text{-stage} < s\}.$$

We say that s is β -expansionary if $\ell(\beta, s) > m\ell(\beta, s)$ and declare s to be a $\beta \hat{=} \infty$ -stage.

If $\ell(\beta, s) \leq m\ell(\beta, s)$, declare that s is a $\beta \hat{=} f$ -stage.

Case 2. Suppose that β is a σ node devoted to (i, x) . If there has been a previous β -stage t let s' denote the largest β -stage $\leq s$. If $\ell(\tau(\beta), s') > x$ and there has been a stage m with $s' < m < s$ with $\ell(\tau(\beta), m) \leq x$, declare s to be a $\beta \hat{=}(i, x, u)$ -stage. In any other case, declare s to be a $\beta \hat{=}(i, x, \infty)$ -stage.

(b) As in Theorem 19.8, we define TP_s , the apparent true path at stage s , to be the unique α of length s with s an α -stage.

Definition 19.27 Suppose that R_α is a version of $R_{\langle e, m, n \rangle}$ attached to α . We say that R_α *requires attention* at stage s if s is an α -stage, R_α is not yet declared satisfied, and there is some least unused $p \leq s$ such that $p = \varphi_{m,s}(\text{slice}(\alpha, s), t) \downarrow$ for some $t \leq s$.

The Construction. Having gone to all the effort above, the construction actually becomes rather easy.

Stage 0. Define $\text{slice}(\alpha, s) = \alpha$ for all α on PT with R_α attached.

Stage $s + 1$. Compute TP_{s+1} . For each γ with $\gamma \not\leq_L TP_{s+1}$, initialize γ and in particular reset $\text{slice}(\gamma, s)$ to be new and large. ($> s$). For each $\alpha \leq TP_{s+1}$, if R_α requires attention via p , declare p as used. Now compute ps^n steps in the computation of $\Phi_m(\emptyset; \langle 1^s, \text{slice}(\alpha, s) \rangle)$. If this does not halt do nothing, do nothing else. If $\Phi_m(\emptyset; \langle 1^s, \text{slice}(\alpha, s) \rangle)$ halts in $\leq ps^n$ steps, win by setting

$$V(\langle 1^s, \text{slice}(\alpha, s) \rangle) = 1 - \Phi_m(\emptyset; \langle 1^s, \text{slice}(\alpha, s) \rangle),$$

and declare R_α as satisfied.

End of Construction.

The verification is straightforward modulo all the definitions etc. Let TP be the true path. Notice that the construction ensures that R_α only can receive attention at α -stages. (See Definition 19.27.) The initialization of γ right of TP_s means that the following is easily seen by induction:

- (i) If $TP <_L \gamma$ then $\text{slice}(\gamma, s) \rightarrow \infty$.
- (ii) If $\gamma <_L TP$ and $\gamma \not\subseteq TP$ then γ is visited only finitely often, and hence there is some stage s_0 beyond which R_α will never receive attention.
- (iii) If $TP \not\subseteq_L \gamma$ then $\lim_s \text{slice}(\gamma, s) = \text{slice}(\gamma)$ exists.

Now if $e \in \text{Comp}$ then we know that there is some (least) i such that i is the witness to the fact that $e \in \text{Comp}$ and hence $\liminf_s \ell(e, i, s) \rightarrow \infty$. Let τ be the top node on TP with $i(\tau) = i$. Clearly there are infinitely many $\tau \hat{\ } (i, \infty)$ -stages. Now for each x , we see that $(i, x, \infty) \subseteq TP$. We show that $V \in \text{FPT}(\leq_T^s)$ as follows. Let s_0 be a stage beyond which we are never left of $\tau \hat{\ } (i, \infty)$. Since we ensure that at most finitely many elements ever get into any slice, we can by fiat write in all the slices associated with nodes left or above $\tau \hat{\ } (i, \infty)$. We consider slices V_y for y beyond such slices. Let $x > y$ and $s \geq s_0$. We know that for all nodes γ with $\gamma \not\subseteq_L \tau \hat{\ } (i, 0, \infty) \hat{\ } (i, 1, \infty) \hat{\ } \dots \hat{\ } (i, x, \infty)$, either γ is above or left of τ and so we can suppose that all elements of $\text{slice}(\gamma, s)$ to ever enter V have already done so by stage s_0 , or we know that $\text{slice}(\gamma, s) > x > y$. The procedure to decide if $\langle v, y \rangle \in V$ is that $\langle v, y \rangle \in V$ iff $\langle v, y \rangle \in V$ by stage t' where $t = t(y)$ is a constant computed by a $\mathbf{0}'$ -oracle. We compute $t > s_0$ by the notion of TP_s . Each time we play $\tau \hat{\ } (i, 0, \infty) \hat{\ } (i, 1, \infty) \hat{\ } \dots \hat{\ } (i, x, \infty)$ immediately after playing $\tau \hat{\ } (i, 0, \infty) \hat{\ } (i, 1, \infty) \hat{\ } \dots \hat{\ } (i - 1, x, \infty) \hat{\ } (i, x, \infty)$, or simply play $\tau \hat{\ } (i, 0, \infty) \hat{\ } (i, 1, \infty) \hat{\ } \dots \hat{\ } (i, z, u)$ for any $u \leq x$, we reset $t(y, s)$ to be the current stage number. Because of the choice of i we know that $\lim_s (t(x), s)$ exists and hence we have $t(y) \leq_T \mathbf{0}'$. Finally note that the choice of t means that the construction is respected, in the sense that we have enough time to emulate that stage of the construction, and hence the answers must be in agreement with the construction.

Finally if $e \notin \text{Comp}$ then by the way we attach $R_{(e, m, n)}$ to α on TP (in order) since there are infinitely many $\tau \hat{\ } (i, \infty)$ or infinitely many $\gamma \hat{\ } (i, x, u)$ on the TP each $R_{(e, m, n)}$ has a version on the TP that must be met, since its $\text{slice}(\alpha, s)$ must come to a limit.

To complete the proof note that the one can replace the oracle by A and the diagonalization similarly to get the relativized version, as in the $\leq_T^s$ case. $\square$

The structure of $(\text{Rec}, \leq)$, the recursive languages under a parameterized reduction $\leq$ is very interesting and fairly open at this stage. Some further results can be found in Exercises 19 and in Refs. [118], [126], and [168]. Much work remains. We give only one further result: We look at whether the analog of Ladner's Density Theorem (Theorem 19.14) holds. We remark that at present we *do not know* if the result holds for any of the uniform or nonuniform reducibilities. We can solve this problem for the most commonly met reducibility: strong uniform reducibility.

Theorem 19.28 ([168]) *Let $q \in \{m, T\}$. If A and B are recursive with $A <_q^s B$, then there exists a set C with $A <_q^s A \oplus C <_q^s B$.*

Proof. We begin by briefly recalling the construction of Theorem 19.14. Recall that this worked as follows. There were given recursive sets $A < B$ (working with $\leq_m^p$, say). By the Slow Enumeration Theorem, we assumed that A and B were given as the range of p -time functions. We built C to satisfy the requirements

$$\begin{aligned} R'_{2e} : \Gamma_e(A \oplus C) &\neq B, \\ R'_{2e+1} : \Gamma_e(A) &\neq C, \end{aligned}$$

additionally ensuring that $C \leq_m^p B$. For the sake of the R'_j , we defined a polynomial-time relation $D(n)$. Then, we declared that $x \in C$ iff $D(|x|) = 0$ and $x \in B$, making $C \leq_m^p B$.

As the reader will recall, the key idea used to meet the R'_j was to meet them *in order* by “delayed” diagonalization. So we began with R'_0 . We set at each stage s , $D(s) = 1$ until a stage t was found where

$$\Gamma_{0,t}(A_t \oplus C_t; z_n) \neq B(z_n)$$

via *certified* computations. Then, at stage t , we saw that we had diagonalized R'_0 at z_n , this being found by *looking back for a certified disagreement*. The idea was then to move to R'_1 and then to R'_2 , etc. For R'_1 , we set $D(t+1) = 0$, causing C to look like B locally. So we kept $D(u)$ for $u > t$ equal to zero until a stage v is found with some $m \leq v$ and

$$\Gamma_{0,v}(A_v; z_m) \neq C_v(z_m)$$

via certified computations. We then move to R'_2 , setting $D(v+1)$ to be 1 again. Thus, the set C so constructed looked like B with “holes” in it.

Keeping the above ideas in mind, we turn to the result at hand. Now, we are given $A < B$ with $\leq$ either $\leq_T^s$ or $\leq_m^s$. Again, we must construct C , now to meet the following requirements

$$\begin{aligned} R_{2\langle e,n \rangle} : & \text{Either } \varphi_e \text{ is not total,} \\ & \text{or } (\exists k)(B_k \neq \Phi_e(A \oplus C^{(\varphi_e(k))})) \\ & \text{or } (\exists x, k)(\Phi_e(A \oplus C^{(\varphi_e(k))}; \langle x, k \rangle) \text{ does not run in time} \\ & \quad \varphi_e(k)|x|^n). \\ R_{2\langle e,n+1 \rangle} : & \text{Either } \varphi_e \text{ is not total,} \\ & \text{or } (\exists k)(C_k \neq \Phi_e(A^{(\varphi_e(k))})) \\ & \text{or } (\exists x, k)(\Phi_e(A^{(\varphi_e(k))}; \langle x, k \rangle) \text{ does not run in time} \\ & \quad \varphi_e(k)|x|^n). \end{aligned}$$

To aid the discussion, we will use several conventions. First, if $\varphi_{e,s}(k) \downarrow$, then the computation $\Phi_e(E^{(\varphi_e(k))}; \langle x, k \rangle)$ cannot call any y of the form (k', z) for $k' > \varphi_e(k)$. Also, since we get a win for free if $\varphi_{e,s}(k) \downarrow$ and the running time of $\Phi_e(E^{(\varphi_e(k))}; \langle x, k \rangle)$ exceeds $\varphi_e(k)|x|^n$, we shall assume that in the above, the third option does not pertain to R_j and concentrate on the first two. This is because if the running time exceeds the bounds during the construction, we can *cancel* the relevant requirement. The argument to follow is a finite injury priority one with the Ladner strategy embedded.

Without loss of generality, we can take φ_e to be strictly increasing. Again, there will be long intervals with $C(\langle x, k \rangle)$ equal to $\emptyset$ and long intervals where it looks like B , for “many” k . We have problems, since, for instance, we cannot decide if φ_e is total. We first focus on the satisfaction of a single $R_0 = R_{2(e,n)}$. We then describe the basic module for an odd-type requirement, and finally describe the coherence mechanism whereby we combine strategies.

The Basic R_0 -Module

To meet R_0 above, we perform the following cycle. We have a parameter $k(0, s)$ that is nondecreasing in s and such that $\lim_s k(0, s) = k(0)$ exists. This is meant to be the number of “rows” devoted to R_0 . It remains constant until we change it.

1. (*Initialization*) Pick $k(0, 0) = 1$.
2. Wait until a stage s occurs with one of the following holding:
 - 2(a). (*Win*) “Looking back,” we see a disagreement; that is, as with the Ladner argument, we see an $n < s$ with $z_n \in \{\langle x, j \rangle : j < k(0, s)\}$.

$$\Phi_{e,s}(A \oplus C^{(\varphi_e(k(0,s)-1))}; z_n) \neq B(z_n)$$

via certified computations, or

- 2(b). Not 2(a) and $\varphi_{e,s}(k(0, s)) \downarrow$.

Comment. If s does not occur, then $\varphi_e(k(0, s)) \uparrow$ and, hence, φ_e is not total. In this case, we call $k(0, s)$ a *witness to the nontotality* of φ_e .

If 2(a) pertains, we declare R_0 to be *satisfied* (forever) and end its effect (forever). If 2(b) pertains, then we perform the following action.

3. R_0 asserts control of $C^{(\varphi_e(k(0,s)))}$; that is, R_0 asks that for all $t \geq s$, until 2(b) pertains, we promise to set $C^{(\varphi_e(k(0,s)))}(y) = 0$ for all y with $|y| = t$ and $y \in (\Sigma^*)^{(\leq \varphi_e(k(0,s)))}$. This can be achieved via a restraint $r(n, k)$.
4. Reset $k(0, s+1) = k(0, s) + 1$ and go to 2.

The Outcomes of the Basic R_0 Module

We claim that 2(b) cannot occur infinitely often and, hence, $\lim_s k(0, s) = k(0)$ exists. Note that we have only reset $k(0, s)$ if 2(b) pertains in step 3. So suppose $k(0, s) \rightarrow \infty$ and hence $\varphi_e(k(0, s)) \rightarrow \infty$. Then, for each q and almost all y , we have $C(\langle q, y \rangle) = 0$.

We write $A =^* B$ to denote that the symmetric difference of A and B is finite. So $C_q =^* \emptyset$ for all q . Furthermore, for all q , we can compute a stage $h(q)$ where

$$[\forall t > h(q)](C_q(\langle y, q \rangle) = 0 \quad \text{for all } y \text{ with } |y| > h(q)),$$

where $h(q)$ is the stage where R_0 asserts control of row q .

Finally, we know that for all k ,

$$\Phi_e((A \oplus C)^{(\varphi_e(k))}) = B_k.$$

This allows us to get a reduction $\Delta(A) = B$. For each input $\langle y, k \rangle$, Δ simply computes $B(\langle y, k \rangle)$ for all y with $|y| \leq h(k)$, and $C(\langle z, k' \rangle)$ for all k', z with $k' \leq \varphi_e(k)$ and $|z| \leq h(k)$. Then, Δ simulates $\Phi_e(A^{(\varphi_e(k))}; \langle y, k \rangle)$ if $|y| > h(k)$ with the exception that, if Φ_e calls some $\langle r, k' \rangle$ with $|r| \leq h(k)$ [and necessarily $k' \leq \varphi_e(k)$], then Δ uses the table of values for C to provide the answer.

Note that the computations of $\Delta(A; \langle x, k \rangle)$ and $\Phi_e(C; \langle x, k \rangle)$ must agree and, hence, $\Delta(A) = B$, a contradiction. Thus, 2(b) can pertain only finitely often. It follows that there are two outcomes.

Outcome $(0, f)$: 2(a) occurs for some t . Then, we win R_0 with finite effect. (**Comment:** Once R_0 is met in this way, say at stage t , then we are completely free to do what we like with all y for which $|y| > t$ without injuring R_0 .)

Outcome $(0, \infty)$: 2(a) does not occur. Then, φ_e is not total. Note that the effect of R_0 is in this case infinite and for some $k = \lim_s k(0, s) - 1$, we will have

$$C^{(\varphi_e(k))} =^* \emptyset$$

and, furthermore, there is a reduction Δ_0 with time bound $\varphi_e(k)|x|^n$ for which

$$\Delta_0(A^{(\varphi_e(k))}) = B^{(k)}$$

Note that for the basic module, Δ_0 is simply Φ_e .

The Basic Module for R_1

This is essentially the same as for R_0 except that for R_1 we wish to set $C(\langle x, k \rangle) = B(\langle x, k \rangle)$. Herein is the basic conflict: An even-indexed requirement R_j asks that lots of rows look like $\emptyset$ and an odd-indexed R_j asks for them to look like B .

Combining Strategies

We cannot perform a delayed diagonalization as in the proof of Ladner's theorem, since we cannot know if $\varphi_e(k)$ is defined. The combination of strategies needs the priority method. Let us consider a module for R_1 that works in the outcomes of R_0 . We cannot know if this outcome is $(0, f)$ or $(0, \infty)$. Instead we have a strategy based on a guess as to R_0 's behavior. Basically R_0 always believes that $k(0, s)$ is $k(0)$; that is, that the current value is the final one. Let $e = e(0)$, $n = n(0)$, $f = e(1)$, and $m = n(1)$.

Although R_1 believes that $\varphi_e(k(0, 0)) \uparrow$, R_1 acts as if R_0 is not there. So, if $k(0, 0) = k(0)$ and $\varphi_e(k(0, 0)) \uparrow$, then we win R_1 for the same reasons as we did for R_0 . On the other hand, if $\varphi_e(0) \downarrow$ for some least stage s , then R_0 will assert control of $C^{(\varphi_e(k(0, 0)))}$. For the sake of R_1 , we have probably been setting $C(0, x) = B(0, x)$ for all x with $|x| < s$. Since R_0 has higher priority than R_1 , R_1 must release its control of C_0 [and indeed of C_j for $j \leq \varphi_e(k(0))$] until a stage, if any, occurs where 2(a) pertains to R_0 so that R_0 is satisfied and releases control forever (or it becomes inactive because of a time bound being exceeded). Note that if 2(a) pertains at t , then R_1 is free to reassert control of C_0 for all y of the form $\langle y, 0 \rangle$ with $|y| > t$. Also, in this case, as R_1 is the requirement of highest overall priority remaining, its control cannot be violated and hence it will be met.

On the other hand, while R_0 can hope that 2(a) will pertain to R_1 , R_0 may have outcome $(0, \infty)$ and R_0 will never release control of C_0 . The key idea at this point is that we begin anew with a version of R_1 , believing that $k(0, s+1) = k(0)$; that is, R_0 will never again act.

This version of R_1 can only work with C_q for $q > \varphi_e(k(0, s)) = \varphi_e(k(0, 0))$. Some care is needed since potentially we need all of B to meet R_1 .

An elegant solution to this difficulty is to *shift* B into C above $\varphi_e(k(0, s))$. Thus, R_1 will ask that

$$C(\langle x, q \rangle) = B(\langle x, q - \varphi_e(k(0, s)) - 1 \rangle)$$

for $q > \varphi_e(k(0, s))$. It does so until either $k(0, t)$ is reset again, or 2(a) pertains, or the time bounds are exceeded. In the latter cases, it reverts to the $(0, f)$ -strategy. In the first case, it begins anew on $q > \varphi_e(k(0, t))$. Since this restart process only occurs finitely often, it follows that we eventually get a *final* version of R_1 whose actions will not be disturbed.

Thus, there is a final version of R_1 that is met as follows. As $\lim_s k(0, s) = k(0)$ exists, there is a value r and a stage s_0 so that for $q \geq r$ and $s > s_0$, R_1 is not initialized at stage s and can assert control on C_q if it so desires. If R_0 has outcome $(0, f)$, then $r = 0$, otherwise $r = \varphi_e(k(0) - 1) + 1$. So, we know that if R_1 fails, then for all j , there is a stage $h(j)$ (computable from the parameters r and s_0) where for y with $|y| > h(j)$,

$$C(\langle y, r + j \rangle) = B(\langle y, j \rangle)$$

and

$$\Phi_f(A; \langle y, r + j \rangle) = C(\langle y, r + j \rangle).$$

Thus, if R_1 fails again, we can prove there is a reduction $\Delta(A) = B$ with running time $O(|z|^m)$ and computable constants. This is a contradiction.

The outcomes for R_1 are thus $(1, \infty)$ and $(1, f)$. In the former case, we know that for a finite number of rows j and for almost all y , $C(\langle y, j \rangle) = B(\langle y, j \rangle)$. But we also know that for such rows, there is a reduction Δ_f such that

$$\Delta_f(A; \langle y, j \rangle) = C(\langle y, j \rangle) \text{ in time } O(|y|^m) \text{ and computable constants.}$$

We continue in the obvious way with the inductive strategies. Consider, for example, R_2 . It is confronted with, at worst, a finite number of rows permanently controlled by R_0 and a finite number by R_1 . However, in each case we know that there is a reduction from a computable number of rows of A to these rows, and hence a reduction

$$\Psi_2(A; \langle y, j \rangle) = C(\langle y, j \rangle)$$

for all j cofinally under the control of either R_0 or R_1 . Therefore, to argue that R_2 is met, we get to use Ψ_2 to help construct a reduction from A to B ; that is, for R_i , let $e = e(i)$ and $n = n(i)$. Then, inductively, we have a reduction and constants $p(2)$, $m(2)$, and $r(2)$ with

$$\Psi_2(A^{m(2)}; \langle x, j \rangle) = C(\langle x, j \rangle)$$

for all $j \leq p(q)$ running in time $m(2)|x|^{r(2)}$. Furthermore, we have a stage s_2 such that for all $k < 3$, R_k ceases further activity.

Thereafter, R_2 is free to assert control over any row q of C for $q > p(2)$. If we suppose that R_2 fails, then for each such q , R_2 will eventually assert control of C_q

at some stage $h(q)$ to make $C(\langle x, q \rangle) = 0$ for all x with $|x| > h_2(q)$ and we have $\Phi_{e(2)}(C) = B$.

Now, to get a reduction Δ from A to B , we argue as for R_0 except that now if $\Phi_{e(2)}$ makes an oracle question of $\langle y, j \rangle$ for $j \leq p(2)$, we use Ψ_2 to answer this question. Thus, we get a reduction Δ_2 that runs in time $O(|x|^{r(2)+n(2)})$, with computable constants and correct use. Thus, again, $B \leq A$, a contradiction.

To complete the proof, one generalizes the above directly into the relevant Depth n strategies, which are given in the obvious way inductively.

To conclude this chapter, we will briefly mention some oracle results. Although there has always been much debate about the value of oracle results, particularly in view of the $IP = PSPACE$ nonrelativizing result of Shamir [418], nevertheless, it is felt that oracle results do give some insight into the difficulty and independence of various open questions.

Since there is an oracle A such that $P^A = NP^A$, there is an oracle with $W[P]^A = FPT^A$. We offer two additional oracle results. Before stating these results, we remark that we do not know of an oracle relative to which the W -hierarchy is infinite.

Theorem 19.29 ([168]) *There exist oracles A and B such that*

- (i) $FPT^A \neq W[1]^A$,
- (ii) $P^B \neq NP^B$ yet $W[P]^B = FPT^B$.

The Oracle Model. Before we turn to the proof we should mention that it is not totally clear exactly what is wanted from an oracle in the parameterized setting. The natural thing is to only allow parameterized queries during the course of an oracle computation. This is the view suggested by looking at $\leq_T^q$ for $q \in \{s, u, n\}$, and is the one we adopt. Similarly, the classes we study are defined by circuits and, again, it seems reasonable to adopt the Wilson [463] view of using oracle gates again modified for the parameterized setting; that is, a parameterized oracle gate with parameter k and oracle $\mathcal{O}$ will output 1 on input $\sigma = x_1 \dots x_n$ ($x_i \in \{0, 1\}$) with the inputs read left to right, iff $\sigma \in \mathcal{O}$ and σ has weight k .

Proof of Theorem 19.29. (i) (For $FPT(\leq_T^s)$ and for other reducibilities, see Exercise 10.) This is a minor variation of the Baker-Gill-Solovay construction. We use the set C defined as follows:

$$\langle z, k \rangle \in C \text{ iff } (\exists y(|y|) = |x| \text{ and } y \in A \text{ and } y \text{ has weight } k).$$

Now, C is in $W[1]^A$ via the circuit such that, on input $\langle z, k \rangle$, the circuit to accept $\langle z, k \rangle$ consists of a single oracle gate with inputs $z_1, \dots, z_{|z|}$ ordered left to right. Then, $\langle z, k \rangle \in C$ iff there is a weight k work accepted by the gate (i.e., in A). It is routine then to build A to meet

- $R_{e,n}$. Either φ_e is not total, or
- there is an $\langle x, k \rangle$ such that $\Phi_e(A; \langle x, k \rangle) \neq C(\langle x, k \rangle)$, or
- the running time exceeds $\varphi_e(k)|x|^n$.

To do this, we assign $R_{e,n}$ some row $k = k(n)$ with $m^k > \Omega(m^n)$. Then, we meet $R_{e,n}$ as follows. Wait until $\varphi_e(k) \downarrow$. Then, find an x of sufficient length so as not to injure other requirements, and so that $|x|^k$ exceeds $\varphi_e(k)|x|^n$. We can then diagonalize via $\langle x, k \rangle$ in the standard way, using the string not addressed in the A computation of length $|x|$ (if it is the case that Φ_e is outputting 0), or doing nothing as the case may be but then restraining the result so as not to be disturbed by future actions. The result is an easy finite extension argument.

(ii) Again, we do this for $\leq_T^s$ and observe that the obvious modifications work for the other reducibilities. Let Q_e denote the e -th P -time relation. Define K^F via

$$\langle \langle x, e, 0^n \rangle, k \rangle \in K^F \text{ for some } y \text{ with } |y| = |x|, \text{ } y \text{ has weight } k \\ \text{and } Q_e^F(y) \text{ holds in } n \text{ steps.}$$

In view of the direct relationship between circuits and relations, it is clear that K^F is $W[P]^F$ -complete.

Now, let f be any recursive function from $\mathbb{N}$ to $\mathbb{N}$. Suppose we build a recursive set B such that for each k and all x with $|x| \leq f(k)$, we have $B(\langle x, k \rangle)$ which can be computed in $g(k)$ many steps, and for all y with $|y| > f(k)$, we have $B(\langle y, k \rangle) = B(\langle y, k \rangle)$. We claim that $B \equiv_m^s F$, so that $W[P]^F = W[P]^B$. To see this for the k -th row for the reduction from B to F , say, we first compute $g(k)$ and $f(k)$. As B and F are recursive, we can write the corresponding initial segments in a table. Otherwise, $\langle x, k \rangle \in B$ iff $\langle x, k \rangle \in F$, and hence $B \equiv_m^s F$.

Now take any F with $W[P]^F = F$; that is, define F via

$$\langle \langle x, e, 0^n \rangle, k \rangle \in F \text{ iff for some } y \text{ with } |y| = |x|, \text{ } y \text{ has weight } k \\ \text{and } Q_e(y) \text{ holds in } n \text{ steps.}$$

Now, it will suffice to define f , g , and B as above and ensure that $P^B \neq NP^B$. We do this as follows. We must meet the requirements

$$R_k : \Gamma_k(B) \neq C,$$

where $C \in NP^B$ and Γ_k denotes the k -th P -time procedure with use q_k , say. We meet R_k via row $k + 1$. We define C so that

$$\langle x, k \rangle \in C \text{ iff } (\exists y)[|y| = |x| \text{ and } \langle y, k \rangle \in C], \text{ and hence } C \in NP^B.$$

At stage k , we will have defined $f(i)$ and $g(i)$ for $i \leq k$ and a restraint $r(k)$. Choose x so that $2^{|x|}$ exceeds $q_k(\langle x, k \rangle)$, and $|x| > r(k)$. Now, compute $\Gamma_k(B_k^*; \langle x, k+1 \rangle)$, where B_k^* is the result of setting B equal to F on all $\langle x, j \rangle$ for $j \leq k$ and $|x| > f(j)$, and setting $B(\langle x, k \rangle)$ equal to what we have decided at stages $\leq k$ for y with $|y| \leq f(j)$. (So, basically we have *decided* at stage k the precise contents of B on $\langle x, j \rangle$ for $j < k$.) Set $B_k^*(\langle z, r \rangle) = 0$ for $r \geq k + 1$.

Now, if $\Gamma_k(B_k^*; \langle x, k+1 \rangle) = 1$, define $\langle x, k+1 \rangle = 0$ for all z with $|z| \leq r(k+1)$ with $r(k+1) \geq r(k)$ and also exceeding all uses seen so far. (This means that future actions will not affect these protected computations.) If $\Gamma_k(B_k^*; \langle x, k+1 \rangle) = 0$, then for some y with $|y| = |x|$, we have that $\langle y, k+1 \rangle$ is not queried during

the computation of $\Gamma_k(B_k^*; \langle x, k+1 \rangle)$ computation because $2^{|x|}$ exceeds $q_k(|x|)$. Now, put $\langle y, k+1 \rangle$ into B and otherwise set $B(\langle z, k+1 \rangle) = 0$ for all z with $|z| \leq r(k+1)$. Set $f(k+1) = r(k+1)$ and define $g(k+1)$ appropriately.

This ensures that $\Gamma_k(B) \neq C$ via the witness $\langle x, k+1 \rangle$ since inductively all previous restraints are maintained, and hence all previous disagreements are also preserved. Thus, $NP^B \neq P^B$ and yet $FPT^B(\leq_m^s) = W[P]^B$.

Corollary 19.30 ([168]) *There exist recursive oracles B and F with $W[P]^B = W[P]^F = BPT^B = BPT^F$, $B \equiv_m^s F$, $NP^F = P^F$, yet $P^B \neq NP^F$.*

Proof. Take B and F as in the proof of Theorem 19.29(ii). We claim that $NP^B = P^B$. This will then give the desired result. Define D via

$$\langle x, e, 0^{p_e(|x|)} \rangle \in D \text{ iff some computation of } \Phi_e(F; x) \text{ accepts in } n \text{ steps.}$$

Then, as usual, D is NP^F complete. We claim that $D \in P^F$. To see this simply note that

$$\langle x, e, 0^{p_e(|x|)} \rangle \in D \text{ iff } (\exists j)[j \leq |x| \text{ and } \langle \langle x, e, 0^{p_{h(e)}(|x|)} \rangle, j \rangle \in F],$$

where $Q_{h(e)}$ is the relation representing Φ_e . Hence, $P^D = NP^D$.

The reader should note that Theorem 19.29(i) actually separates FPT^A from $MONOTONE W[1]^A$, which is *not* true for $A = \emptyset$. Actually, $FPT^A \neq MONOTONE W[1]^A$, can be shown to hold with probability 1 for a random oracle A (Exercise 12), so this gives yet another example of the failure of the random oracle hypothesis.

Historical Notes

Investigations into the structure of recursive languages under parameterized reducibilities began with Downey and Fellows [168], although the definitions of this section can be found in Refs. [166] and [167]. Other investigations are found in Ref. [119] and Coles, Downey, and Sorbi [118], where some undecidability results are proven for the theories of the structures. These results are beyond the scope of the chapter and need complex infinitary applications of the speedup technique. (See Exercise 9.) Theorem 19.29(i) is independently due to Abrahamson [2]. Theorem 19.23 and a number of results in the exercises are new.

Exercises 19.2

19.2.1. (Downey and Fellows [168]) Prove the following:

- (i) Suppose A and B are recursive sets with $A \leq_T^n B$ via a recursive collection $\{\Phi_{g(k)} : k \in \mathbb{N}\}$ of reductions all running in time $O(|z|^\alpha)$. Then $A \leq_T^u B$.
- (ii) Furthermore, if the running time is recursively bounded (and hence can be taken to be $g(k)|z|^\alpha$), then $A \leq_T^s B$.

19.2.2. Prove (ii) of Lemma 19.20; that is, prove that if $A \leq_T^n B$ (or $A \leq_m^n B$) with A and B recursive, then there exists a recursively enumerable function f such that $A \leq_T^n B$ (resp. $A \leq_m^n B$) via f .

19.2.3. (Cai, Chen, Downey, and Fellows [110]) Let $\mathcal{C}$ be a class of functions representing time (space) resource bounds. Recall from Part I, Chapter 5, that a (parameterized) language L is *(uniform) $\mathcal{C} + \text{advice}$* if there is a (recursive) function $g \in \mathcal{C}$, an oracle Turing machine Γ , and a function $w : N \mapsto \Sigma^*$ (the *advice* function) such that

$$\langle x, k \rangle \in L \text{ iff } \Gamma^{w(k)}(\langle x, k \rangle) \text{ accepts,}$$

and, furthermore, for all k and x , the running time (space) of $\Gamma^{w(k)}(\langle x, k \rangle)$ is $\leq g(|x|)$. Similarly, recall that *(uniform) SLICEWISE $\mathcal{C}$* as those languages L accepted in time (space) $f(k)g(|x|)$.

Verify the following fact from Chapter 5 (Exercise 7):

$$\begin{aligned} &(\text{uniform}) \text{ LOGSPACE} + \text{advice} \\ &\neq (\text{uniform}) \text{ SLICEWISE LOGSPACE}. \end{aligned}$$

[Hint: We only need to build a language whose k -th slice is accepted in space $f(k) \log |x|$, where we get to build f . We need to construct $f(k)$ so that we have enough room to diagonalize. We need the following fact.

Claim. Suppose that L is in LOGSPACE + advice. Then, there exists a witness oracle Turing machine Γ accepting L running in space $O(\log |x|)$ which has an advice function w recursive in $\mathbf{0}'$, the Turing degree of the halting problem.

First, prove the claim using the method of Lemma 19.20. Thus, it suffices to meet the following requirements.

$$\begin{aligned} R_{e,n} : & \text{Either } \exists k (\lim_s (\varphi_e(k, s)) (=_{\text{def}} \varphi_e(k)) \text{ does not exist),} \\ & \text{or } \exists x, k (\Phi_e^{\varphi_e(k)}(\langle x, k \rangle) \text{ does not run in space } n \cdot \log(|x|)) \\ & \text{or } \exists x, k (\Phi_e^{\varphi_e(k)}(\langle x, k \rangle) \neq L(\langle x, k \rangle)). \end{aligned}$$

In the above, $\langle \varphi_e(\ , \), \Phi_e \rangle_{e \in \omega}$ is a simultaneous enumeration of all pairs consisting of an oracle Turing machine and a partial recursive binary function. We devote the $\langle e, n \rangle$ -th slice of L to meeting the requirement $R_{e,n}$. We define $f(\langle e, n \rangle) = n + 1$. We can assume (see Lemma 19.20) that if $\varphi_e(\langle e, n \rangle, s) \downarrow \neq \varphi_e(\langle e, n \rangle, t) \downarrow$ and $t > s$, then for all $u \geq t$, $\varphi_e(\langle e, n \rangle, u) \neq \varphi_e(\langle e, n \rangle, s)$.

At stage s , on row $\langle e, n \rangle$ with $1^{(e,n)} \leq s$ we perform the following action. If we see $\varphi_{e,s}(\langle e, n \rangle, t) \downarrow$ in $\leq \log(|s|)$ space and $\Phi_e^{\varphi_e(\langle e, n \rangle, t)}(1^s, \langle e, n \rangle) \downarrow$ in $\leq (n + 1) \log(s)$ space, with $t \leq \log(|s|)$, for the largest such t , let

$$L(\langle 1^s, \langle e, n \rangle \rangle) = 1 + \Phi_e^{\varphi_e(\langle e, n \rangle, t)}(\langle 1^s, \langle e, n \rangle \rangle).$$

This action will succeed in diagonalizing the *possibility* of $\varphi_e(\langle e, n \rangle, t)$ as being a value for $\lim_s (\varphi_e(\langle e, n \rangle, s))$, since either we will thus diagonalize a final value for $\varphi_e(\langle e, n \rangle)$, or possible values will change infinitely often, or the running space will never be correct, or finally $\varphi_e(\langle e, n \rangle)$ will be partial. In any case, we succeed in meeting the $R_{e,n}$ result.]

19.2.4. (Cai, Chen, Downey, and Fellows [110]) Extending the notation of Exercise 3, let L_1 and L_2 be parameterized problems. We say that L_1 is (*uniformly*) *SLICEWISE LOGSPACE* reducible to L_2 iff there is a procedure M and a (recursive) function $f(k)$ such that for all $z \in \Sigma^*$, and all $k \in \omega$,

$$\langle z, k \rangle \in L_1 \text{ iff } M(\langle x, k \rangle) \text{ accepts and runs in space } f(k) \log |z|.$$

Similarly, we can define (*uniform*) *LOGSPACE + advice* reductions for M by asking for the existence of an oracle Turing machine Γ , and an *advice* function w so that for all x, k ,

$$\langle x, k \rangle \in L_1 \text{ iff } \Gamma^{w(k) \oplus L_2}(\langle x, k \rangle) \text{ accepts,}$$

and Γ is running in space $\log |x|$.

Let $A \subseteq \Sigma^*$. Define $N_1(A) = \{\langle z, 0 \rangle : z \in A\}$ and $N_2(A) = \{\langle z, k \rangle : z \in A, k \in \omega\}$. Note that $N_1(A) \equiv N_2(A)$ for either of the reductions defined above. Also, let B be a parameterized problem. We can define the *standardization* of B to be $S(B) = \{\langle \langle z, q \rangle, k \rangle : \langle z, q \rangle \in B \text{ and } q \leq k\}$. Again, note that B and $S(B)$ have the same parameterized complexity. Standardization ensures that the structure of a set is smooth in the sense that the k -th row codes all the information of the preceding rows.

Prove the following:

- (i) Let A be P -complete. Then, $N_1(A)$ and $N_2(A)$ are complete for *FPT* under (*uniform*) *LOGSPACE + advice* reductions.
- (ii) Furthermore, if B is complete for *FPT* under (*uniform*) *SLICEWISE LOGSPACE* reductions, then for some k , for all $k' \geq k$, the k' -th row of $S(B)$ is P -complete. Hence, conclude that (*strongly uniform*) *FPT* = (*uniform*) *SLICEWISE LOGSPACE* iff $P = \text{LOGSPACE}$.
- (iii) (*uniform*) *SLICEWISE LOGSPACE* has no problem complete under (*uniform*) *LOGSPACE + advice* reductions.

[Hint: This uses the technique of Exercise 3. If we had such a problem L , accepted in space $f(k) \log(|x|)$ slicewise, then we could build L' not computable by L in space $2^k f(k) \log(|x|)$ since the additional 2^k multiplicative factor gives us plenty of extra space to diagonalize and no finite advice could lift $f(k)$ to $2^k f(k)$.]

19.2.5. Prove (i) and (ii) of Theorem 19.22; that is, prove that if $q \in \{m, T\}$, then

$$\{e : W_e \leq_q^s A\} \text{ is a } \Sigma_3 \text{ set,}$$

and

$$\{e : W_e \text{ recursive and } W \leq_q^n A\} \text{ is a } \Sigma_4 \text{ set.}$$

19.2.6. Prove that $\{e : W_e \equiv_q^s \emptyset\}$ with $q \in \{m, T\}$ is Σ_3 -complete.

19.2.7. Prove that $\{e : W_e \text{ recursive and } \equiv_q^r \emptyset\}$ for $q \in \{m, T\}$ and $r \in \{u, n\}$ is Σ_4 -complete.

19.2.8. (Cholak and Downey [126]) Call a language L *row finite* if L_k contains only finitely many elements for each k . Let $q \in \{m, T\}$. Prove the following:

(i) Suppose that $\mathcal{I}$ is an ideal (i.e., $\mathcal{I}$ is closed under join and initial segments) in $(REC, \leq_q^u)$ and $\mathcal{I}$ is generated by $\{A_i : i \in Q\}$ with Q a Σ_4^0 set, and $\{A_i : i \in \mathbb{N}\}$ is a recursive collection of row finite recursive sets. Then, there is a collection $C = \{B_i : i \in M\}$ of row finite recursive languages, with M recursive and such that $\mathcal{I}$ is equal to the ideal generated by C .

(ii) Suppose that $\mathcal{I}$ is an ideal in $(REC, \leq_m^s)$ generated by $\{A_i : i \in Q\}$ with Q a Σ_3^0 set, and $\{A_i : i \in \mathbb{N}\}$ a recursive collection of row finite recursive languages. Then, there is a recursive collection C of row finite recursive sets with $\mathcal{I}$ equal to the ideal generated by C .

[Hint: Use the method of Theorem 19.23.]

19.2.9. (Cholak and Downey [126], Weak Exact Pairs) This question assumes that the reader is familiar with the speedup technique for finite and infinite injury arguments in the study of the polynomial-time degrees. (See for instance, Refs. [19, 161], and [421].) The Weak Exact Pair Theorem of this question is used by Cholak and Downey [126] to encode countable distributive lattices and, hence, deduce the undecidability of the elementary theory of the recursive languages under $\leq_q^u$ and $\leq_q^s$. This argument is along the lines but very much more difficult than the one of Ambos-Spies and Nies [22]. We remark that it is unknown if all ideals have exact pairs.

In the notation of Exercise 8, prove the following. Let $\{A_i : i \in Q\}$ be a Σ_4^0 collection of row finite recursive languages. Then, the ideal $\mathcal{I}$ of degrees in $(REC, \leq_q^u)$ generated by this collection has an “exact pair”; that is, there is a pair of recursive languages C and D such that $\mathbf{a} \in \mathcal{I}$ iff $\mathbf{a} \leq_m^u \mathbf{c}, \mathbf{d}$, where $\mathbf{c}$ denotes the uniform q -degree of C and $\mathbf{d}$ the uniform q -degree of D .

[Hint: By Exercise 8, we may assume that the collection $\{A_i : i \in Q\}$ is recursive. We meet the requirements

- $R_{e,n}$: Either $\lim_s \varphi_e(k, s) =_{\text{def}} \varphi_e(k)$ fails to exist for some k
- or $\exists x, k (\Gamma_e(C; \langle x, k \rangle) \text{ or } \Delta_e(D; \langle x, k \rangle) \text{ does not run in time } \varphi_e(k)|x|^n)$,
- or the use exceeds $C^{(\varphi_e(k))}$ or $D^{(\varphi_e(k))}$
- or $\Gamma_e(C) \neq \Delta_e(D)$,
- or for some i , $\Gamma_e(C) \leq_q^u \oplus_{j \leq i} A_j$.

This time, we shall code A_i into $C_{i,j}$ and $D_{i,j}$ for $j \in \mathbb{N}$; that is, we ensure that we meet

$$P_i : A_{i,j} =^* C_{i,j} =^* D_{i,j} \text{ for all } j.$$

Then the construction runs as follows. We can split the P_i into $P_{i,j}$ for $j \in \omega$, trying to achieve $A_{i,j} =^* C_{i,j} =^* D_{i,j}$. This is achieved by direct coding, subject to finite injury. We will have $R_{e,n}$ respect $P_{i,j}$, e.g., for $i, j < \langle e, n \rangle$.

To combine the strategies, use the “speedup technique” of Downey [161], Shinoda-Slaman [421], or Ambos-Spies [20].

19.2.10. Modify the proof of Theorem 19.29(i) to construct oracles A and B with

- (i) $FPT(\leq_T^u)^A \neq W[1]^A$,
- (ii) $FPT(\leq_T^n)^B \neq W[1]^B$.

19.2.11. (i) Prove that collapse propagates from strong to weak levels of uniformity; that is, prove that if, e.g., $W[1] = FPT$ for $\leq_T^s$, then $W[1] = FPT$ for $\leq_T^q$ with $q \in \{u, n\}$.

(ii) Construct oracles A and B with

- (a) $W[P]^A = FPT^A$ for $\leq_T^u$ yet $W[1]^A \neq FPT^A$ for $\leq_T^s$.
- (b) $W[P]^B = FPT^B$ for $\leq_T^n$ yet $W[1]^B \neq FPT^B$ for $\leq_T^u$.

[Hint: For (a), code the diagonalization slices above the use of the point of diagonalization. Use a priority argument.]

19.2.12. (For those who know the methods of Bennett and Gill [55]) Prove that $MONOTONE\ W[1]^A \neq FPT^A$ for a random oracle A .

19.2.13. (Cholak and Downey [126]). (Again, the reader will need knowledge of the speedup technique at least to be able to make minimal pairs.) A natural question is whether there is an easy transference of results from $(REC, \leq_m^P)$ to $(REC, \leq_m^q)$. Cholak and Downey [126] showed that the obvious approach does not work, and the methods shed considerable light on the standard methods for getting results in $(REC, \leq_m^P)$. The obvious approach is to take a given set A and define $A' = \{\langle x, 0 \rangle : x \in A\}$; that is, to concentrate A in row 0 of A' . The hope would be that if (e.g.) A and B form a minimal pair in $(REC, \leq_m^P)$, then in $(REC, \leq_m^u)$, A' and B' would form a minimal pair. (There are obviously many other structural properties we can attempt to transfer in this way. We concentrate here on minimal pairs as a test case.) Cholak and Downey showed that this is not true in a very strong sense, as we now see. First, note that usually when we make minimal pairs for P , we meet the requirements

$$R_e : \Phi_e(A) = \Gamma_e(B) \text{ then } \Phi_e(A) \text{ is recursive.}$$

[Here without loss of generality we let $\langle \Phi_e, \Gamma_e \rangle : e \in \mathbb{N}$ denote a listing of all pairs of P -time procedures. See Exercise 16.] Suppose the running time is bounded by $|x|^n$. What we actually do in all known constructions is to ensure that if $\Phi_e(A)$ equals $\Gamma_e(B)$, then we can compute the answer via the empty string, and hence $\Phi_e(A)$ is computable in time $|x|^n$. It follows that we would expect that $\Gamma_e(A')$ would certainly be in $FPT(\leq_m^n)$. Prove that even this is not necessarily the case; that is, prove the following.

(Cholak and Downey [126]) *There exist recursive A and B such that A and B form a PTIME minimal pair, yet A' and B' do not form a $\leq_m^n$ minimal pair.*

[(Hint. Build A and B and an auxiliary set $Q \leq_m^n A', B'$ to meet the R_e of a minimal pair requirement as well as the following.

$N_{e,n}$: Either $\varphi_e(k, s)$ has no limit for some k ,
 or the running time for some k, x exceeds $\varphi_e(k)|x|^n$,
 or the use bound is exceeded,
 or for some k, x , $Q(\langle x, k \rangle) \neq \Delta_{\varphi_e(k)}(\emptyset; \langle x, k \rangle)$.

Make A and B a minimal pair by the speedup technique. Meet the $N_{e,n}$ on row $\langle e, n \rangle$. Let $g = \langle e, n \rangle$. Split the requirement into infinitely many subrequirements that assert the above or that $\varphi_e(k, s)$ changes at least p times, but give all priority $\langle e, n \rangle$. Call this $R_{g,p}$ and give it priority $\langle g, p \rangle$. This will act if we have already seen $R_{g,p-1}$ act, and we get a new value for $\varphi_e(g, t)$ not previously seen and visible at stage s , and such that $\varphi_e(g, t)s^n \leq s^{n+1}$. At this stage when $\mathbb{N}$ asserts control, it will find a very large follower x and declare that $\langle x, g \rangle \in Q$ iff $x \in A$ iff $x \in B$. Note that this forces us with priority $\langle g, p \rangle$ to put x into A iff we put x into B . Now, we do the obvious thing. Go to an appropriately huge stage and see if $\varphi_e(g, t)$ is still the same and the running time and use are still okay, and if so, compute the value of $\Delta_e(\emptyset; \langle x, g \rangle)$. Then, we promise to diagonalize against $Q(\langle x, g \rangle)$ using x via A and B . This is done with priority $\langle e, n \rangle$. As with all speedup arguments so long as this is not injured by higher-priority action, this will be the true action. Clearly, this action will meet the $N_{e,n}$. Why can we meet the R_k ? Note that there are only a finite number of N_j of higher priority than R_k . Their only action is to declare certain configurations illegal and to make certain x enter A iff B . However, each action for this is bounded by time $|x|^{n+1}$ for the relevant n for N_j . Thus, in time polynomially determined from the stage number and j , R_j can figure out whether the higher-priority requirements will assert control of the use of a computation, and what the answer will be. Otherwise, they will diagonalize as usual. Hence, the R_k are met. Finally, note that $Q \leq_m^n A', B'$, since the action of an R_j is finite.]

Part IV

Appendix

Appendix A

A Problem Compendium and Guide to W -Hierarchy Completeness, Hardness, and Classification; and Some Research Horizons

This appendix contains problem definitions and summaries of most of the presently known completeness and hardness results, and information concerning fixed-parameter tractability. References are given where appropriate. The problems discussed are grouped (more or less) according to level in the W -hierarchy. Our list for FPT is incomplete, but the given examples should provide the reader with the flavor of such results.

A.1 In FPT

Remark. The following is only a small list of the known FPT problems. Many more can be obtained by, for instance, Courcelle's theorem applied to classes of bounded treewidth, in conjunction with Bodlaender's theorem on treewidth k recognition. (See, e.g., Refs. [6, 7, 136, 457], and this book.)

BIPARTITE MATCHING CARDINALITY

- Instance:* A bipartite graph $B = (U, V, E)$.
Parameter: An integer k .
Question: Does G have at least k matchings?

This is shown to be in FPT by Itai et al. [279]. The general version of this problem is $\#P$ -complete [452]. The bipartite case is equivalent to the problem **PERMANENT LOWER BOUND** which asks if the permanent of a given binary matrix exceeds k .

CONSECUTIVE BLOCK MINIMIZATION ([239], Problem SR17)

- Instance:* An $m \times n$ matrix M of 0's and 1's and a positive integer k .
Parameter: k .
Question: Is it possible to permute the columns of M to obtain a matrix M' that has at most k blocks of consecutive 1's? (A *block* of consecutive 1's is an interval of a row such that all entries in that interval are 1's.)

The general problem is *NP*-complete. ([239], Problem SR17). In *FPT* by the method of problem kernel (Exercise 3.2.6).

CROSSING NUMBER FOR MAX DEGREE 3 GRAPHS

- Instance:* A graph G all of whose vertices have max degree 3.
Parameter: A positive integer k .
Question: Does G have an embedding with crossing number $\leq k$?

This is $O(n^3)$ by Ref. [205] via Refs. [396] and [397].

CUTWIDTH

- Instance:* A graph $G = (V, E)$.
Parameter: A positive integer k .
Question: Is the cutwidth of $G \leq k$?

The general version of this problem is *NP*-complete by a reduction from **SIMPLE MAX CUT** (See Ref. [239], GT44). This problem is in *FPT* by Fellows and Langston [217]. Solvable in $O(n)$ time for fixed k [65].

$\Pi_{i,j,k}$ GRAPH MODIFICATION PROBLEM

- Instance:* A graph $G = (V, E)$.
Parameters: Non-negative integers i, j, k .
Question: Can we delete at most i vertices, j edges, and add at most k edges and get a Π graph?

This is *FPT* for any nontrivial hereditary property characterized by a finite set of forbidden induced subgraphs [103]).

DIAMETER IMPROVEMENT FOR PLANAR GRAPHS

- Instance:* A planar graph $G = (V, E)$.
Parameter: A positive integer k .
Question: Can G be augmented with additional edges in such a way that the resulting graph G' remains planar and the diameter of G' is at most k ?

This problem is in *FPT* by Downey and Fellows [169] after Robertson and Seymour [397]. Solvable in $O(|V|)$ time for fixed k [65].

DUAL OF COLORING

- Instance:* A graph G and a positive integer k .
Parameter: k .
Question: Can G be properly colored by $n - k$ colors, where n is the number of vertices of G ?

Classically, *NP*-complete. In *FPT* by J. Telle (Exercise 3.2.7). Also, similar methods prove that the following is in *FPT*.

DUAL OF IRREDUNDANT SET

- Instance:* A graph G and a positive integer k .
Parameter: k .
Question: Does G have an irredundant set of size $n - k$?

PLANAR DIAMETER IMPROVEMENT

- Instance:* A planar graph G and a positive integer k .
Parameter: k .
Question: Is there a planar graph G' with $G \subseteq G'$, such that the diameter of G' is at most k ? (The *diameter* of a graph is the maximum distance between a pair of vertices in the graph.)

In *FPT* by Dejter and Fellows [153].

PLANAR DOMINATION IMPROVEMENT NUMBER

- Instance:* A planar graph G and a positive integer k .
Parameter: k .
Question: Is there a planar graph G' with $G \subseteq G'$, such that G' has a k -element dominating set?

In *FPT* by Exercise 7.8.5.

DISJOINT PATHS

- Instance:* A graph $G = (V, E)$, $s_1, \dots, s_k \in V$ start vertices, $t_1, \dots, t_k \in V$ end vertices.
Parameter: A positive integer k .
Question: Do there exist vertex disjoint paths $P_1, \dots, P_k$ such that P_i starts at vertex s_i and ends at vertex t_i for $i = 1 \dots k$?

The general version of this problem is *NP*-complete by a reduction from 3SAT (See Ref. [239], ND40.) This problem is in *FPT* by Robertson and Seymour [397], who proved that the problem is solvable in $O(n^3)$ time for fixed k . This was improved to linear time by Reed et al. [383] for planar graphs.

FEEDBACK VERTEX SET

Instance: A graph $G = (V, E)$.

Parameter: A positive integer k .

Question: Is there a set U of k vertices of G such that each cycle of G passes through some vertex of U ?

The general version of this problem is *NP*-complete by a reduction from **VERTEX COVER**. (See Ref. [239], GT7.) This problem is in *FPT* by Downey and Fellows [169] and Bodlaender [64]. Solvable in $O(n)$ time for fixed k .

GATE MATRIX LAYOUT

Instance: A boolean matrix M .

Parameter: A positive integer k .

Question: Is there a permutation of the columns of M so that if in each row we change to * every 0 lying between the row's leftmost and rightmost 1, then no column contains more than k 1's and *'s?

This problem is in *FPT* by Fellows and Langston [219]. Solvable in $O(n)$ time for fixed k [65]. Equivalent to **GRAPH PATHWIDTH**.

GRAPH GENUS

Instance: A graph $G = (V, E)$.

Parameter: A positive integer k .

Question: Does G have genus k ?

The general version of this problem is *NP*-complete. This problem is in *FPT* by Fellows and Langston [216] via Robertson and Seymour [397]. Solvable in time $O(n^3)$ for fixed k by the results of Robertson and Seymour. Solvable in time $O(|G|)$ with a strongly *FPT* algorithm by Mohar [354].

GROUPING BY SWAPPING

Instance: A finite alphabet Σ , a string $x \in \Sigma^*$, and a positive integer k .

Parameter: k .

Question: Is there a sequence of k or fewer adjacent symbol interchanges that transforms x into a string x' in which all occurrences of each symbol $a \in \Sigma$ are in a single block?

The general problem is *NP* complete by Garey and Johnson [239], Problem SR21. In *FPT* by the method of problem kernel (Exercise 3.2.4).

HITTING SET FOR SIZE THREE SETS

- Instance:* A collection C of 3-element subsets of a set S and a positive integer k .
- Parameter:* k .
- Question:* Does S contain a *hitting set* for C of size at most k ; that is, a subset $S' \subseteq S$ with $|S'| \leq k$ and such that S' contains at least one element from each set in C ?

The general problem is *NP*-complete. D. Bryant proved that it is in *FPT* by the method of problem kernel (Exercise 3.2.10).

INDUCED MINOR TESTING FOR PLANAR GRAPHS

- Instance:* Planar graphs G and H .
- Parameter:* H .
- Question:* Is $G \geq H$ in the induced minor order?

Proved *FPT* by Fellows et al. [215]. The general problem is *NP*-complete.

LONG CYCLE

- Instance:* A graph $G = (V, E)$.
- Parameter:* A positive integer k .
- Question:* Does G have a cycle of length $\geq k$?

The general version of this problem is *NP*-complete by a reduction from **HAMILTON CIRCUIT** (See Ref. [239], ND28). This problem is in *FPT* by Fellows and Langston [218]. Solvable in $O(n)$ time for fixed k . The associated problem **EXACT LONG CYCLE** where we determine if there is a cycle of length *exactly* k is also *FPT*, and is due to the authors in Chapter 8.

 k -LEAF SPANNING TREE

- Instance:* A graph $G = (V, E)$.
- Parameter:* A positive integer k .
- Question:* Does G have a spanning tree with k or more leaves?

The general version of this problem is *NP*-complete by a reduction from **DOMINATING SET**. (See Ref. [239], ND2.) This problem is in *FPT* by Downey et al. (this book) correcting Downey and Fellows [169], and, by Bodlaender [64]. In $\text{LOGSPACE} + \text{advice}$ by Cai et al. [110]. Solvable in $O(n)$ for fixed k .

BOUNDED CHARACTER STATE PERFECT PHYLOGENY

- Instance:* A set $\mathcal{C} = \{1, \dots, m\}$ of characters, for each $c \in \mathcal{C}$ a set $A_c = \langle 1, \dots, r_c \rangle$ of states, and n species $s \in A_1 \times \dots \times A_m$ where $m = \max_{c \in \mathcal{C}} r_c$.
- Parameter:* A positive integer m .
- Question:* Is there a tree T with the properties
- (i) $\mathcal{S} \subseteq V(T) \subseteq A_1 \times \dots \times A_m$.
 - (ii) Every leaf in T is in $\mathcal{S}$.
 - (iii) For each $c \in \mathcal{C}$, and each $j \in A_c$, the set of vectors $v \in V(T)$ such that $v_c = j$ induces a subtree of T .

Such a tree is called a *perfect phylogeny* for $\mathcal{S}$. The unparameterized problem is *NP*-complete by Bodlaender et al. [81] and Steel [433]. McMorris et al. [347] demonstrated an $O(n^m)$ algorithm, and a linear time algorithm was found for $m = 3$ by Bodlaender and Kloks [85]. Dress and Steel [187] gave a $O(nm^2)$ algorithm for $r \leq 3$, and Kannan and Warnow [287] gave an $O(n^2m)$ algorithm for all $r \leq 4$. The *FPT* algorithm for all r was established by Agarwala and Fernandez-Baca [12] who demonstrated an $O(2^{3r}(nm^3 + m^4))$ algorithm. This algorithm was then reduced by Kannan and Warnow (see Ref. [12], section 6) by a factor of 2^r using a different data structure. Agarwala and Fernandez-Baca have since developed an $O((r - \frac{n}{m})^m rnm)$ algorithm [13].

MATRIX DOMINATION

- Instance:* An $n \times n$ matrix M with entries from $\{0, 1\}$ and a positive integer k .
- Parameter:* k .
- Question:* Is there a set C of k or fewer nonzero entries in M that dominate all others, in the sense that every nonzero entry in M is in the same row or in the same column as some element of C ?

The general problem is *NP*-complete ([239], Problem MS12). In *FPT* by the method of problem kernel (Exercise 3.2.9).

MINIMUM DISJUNCTIVE NORMAL FORM

- Instance:* A set $X = \{x_1, x_2, \dots, x_n\}$ of variables, a set $A \subseteq \{0, 1\}^n$ of implicants, a positive integer m .
- Parameter:* $|A|$.
- Question:* Is there a DNF expression E over X , having no more than m disjuncts, such that E is true for precisely those truth assignments in A and no others?

The general version of this problem is *NP*-complete by a reduction from **SET COVER** (See Ref. [239], LO9). This problem is in *FPT* by Ref. [179]. Solvable in $O(2^{|A|}n)$ time for fixed A .

MINIMUM FILL-IN

- Instance:* A graph G .
Parameter: A positive integer k .
Question: Can we add $\leq k$ edges to G and cause G to become chordal?

The general problem is *NP*-complete by Yannakakis [466]. Solvable in time $O(c^k \cdot |E|)$ and $O(k^5|E||V| + f(k))$ by Kaplan et al. [289].

MINOR ORDER TEST

- Instance:* Graphs $G = (V, E)$.
Parameter: A graph H
Question: Is $H \leq_{\text{minor}} G$?

The general version of this problem is *NP*-complete by a reduction from **HAMILTON CIRCUIT**. (See Ref. [239], OPEN2.) This problem is in *FPT* by Robertson and Seymour [397]. Solvable in $O(n^3)$ time for fixed H .

MORE THAN HALF MAX-3SAT

- Instance:* A 3 SAT expression E and a positive integer k .
Parameter: k .
Question: Is there a truth assignment that satisfies at least $\lceil n/2 \rceil + k$ clauses?

The general problem is easily seen *NP*-complete. In *FPT* by Raman (Exercise 3.2.8).

MULTIDIMENSIONAL MATCHING

- Input:* A set $M \subseteq X_1 \times \dots \times X_r$, where the X_i are disjoint sets.
Parameter: The positive integer r and a positive integer k .
Question: Is there a subset $M' \subseteq M$ with $|M'| = k$, such that no two elements of M' agree in any coordinate?

The general problem is *NP*-complete [239]. Proved to be *FPT* by Downey et al. (this book). It is used to prove that various related problems (e.g., **DISJOINT r -SUBSETS**, **ANTICHAIN OF r -SUBSETS**) are *FPT*.

 k -PERFECT MATCHINGS

- Instance:* A graph G .
Parameter: A positive integer k .
Question: Does G have at least (or exactly) k perfect matchings?

The general problem of finding the maximum number of perfect matchings is $\#P$ -complete (in the size of G) even for bipartite graphs by Valiant [452]. For $k = 1$, the problem is in *P* for bipartite graphs by the old work of Ford and Fulkerson,

and for general graphs by the work of Edmonds [191]. The problem is $O(k|E|)$ for any fixed k by Itai et al. [279]. For *weighted* graphs, finding the best k matchings is *FPT* by, for instance, Chegireddy and Hamacher [125].

NEARLY A PARTITION

- Instance:* A finite set X , a family F of subsets of X and a positive integer k .
Parameter: k .
Question: Is there a subfamily $F' \subseteq F$ with $|F'| \leq k$ such that $F - F'$ is a partition of X ?

The classical problem is easily seen to be *NP*-complete. In *FPT* by the method of problem kernel (Exercise 3.2.9).

PLANAR FACE COVER

- Instance:* A graph $G = (V, E)$.
Parameter: A positive integer k .
Question: Can G be embedded in the plane so that there are k faces which cover all vertices?

The general version of this problem is *NP*-complete by a reduction from **VERTEX COVER** by Fellows [206]. This problem is in *FPT* by Bienstock and Monma [60]. Solvable in $O(n)$ time for fixed k .

PURE IMPLICATIONAL SATISFIABILITY OF FIXED F-DEPTH

- Input:* A pure implicational formula X with at most two instances of each proposition and at most k occurrences of the propositional constant **f** (where **f** is always false).
Parameter: A positive integer k .
Question: Is X satisfiable.

Heusch [267] proved the general problem is *NP*-complete and gave an $O(n^{k+1})$ algorithm for the parametric case. Franco et al. [231] provided an $O(k^k n^2)$ algorithm.

RECTILINEAR PICTURE COMPRESSION

- Instance:* An $n \times n$ matrix of 0's and 1's and a positive integer k .
Parameter: k .
Question: Are there k rectangles that cover all the 1's?

The general problem is *NP*-complete. ([239], Problem SR25). In *FPT* by the method of problem kernel (Exercise 3.2.6).

RESTRICTED ALTERNATING HITTING SET

- Instance:* A collection C of subsets of a set B with $|c| \leq k_1$ for all $c \in C$, an integer k_2 .
- Parameter:* Positive integers k_1, k_2
- Question:* Does player one have a forced win in $\leq k_2$ moves in the following game played on C and B ? Players alternate choosing a new element of B until, for each $c \in C$, some member of c has been chosen. The player whose choice causes this to happen loses.

The general version of this problem is *PSPACE*-complete by a reduction from **QBF**. (See Ref. [239], GP7). This problem is in *FPT* by Abrahamson et al. [4]. Solvable in $O(n)$ time for fixed k_1 and k_2 .

SEARCH NUMBER

- Instance:* A graph $G = (V, E)$.
- Parameter:* A positive integer k .
- Question:* Are k searchers sufficient to ensure the capture of a fugitive who is free to move with arbitrary speed about the edges of G ?

The general version of this problem is *NP*-complete. This problem is in *FPT* by Fellows and Langston [217]. Solvable in $O(n)$ time for fixed k by Bodlaender [65].

SET BASIS

- Instance:* A collection C of subsets of a finite set S and a positive integer k .
- Parameter:* k .
- Question:* Is there a collection B of subsets of S with $|B| = k$ such that, for every set $A \in C$, there is a subcollection of B whose union is exactly A ?

The general problem is *NP*-complete. ([239], Problem SP7). It is *FPT* by the method of problem kernel (Exercise 3.2.3).

SHORT 3DIMENSIONAL MATCHING

- Instance:* Set $M \subseteq X \times Y \times Z$ where X, Y, Z are disjoint sets having the same number of elements q .
- Parameter:* A positive integer k .
- Question:* Does there exist a set $M' \subseteq M$ such that $|M'| = k$ and for all $(x, y, z) \neq (x', y', z')$ both in M' , we have $x \neq x'$, $y \neq y'$, and $z \neq z'$?

The general problem with k varying is *NP*-complete and is one of Garey and Johnson's six basic problems. For k fixed the problem is *FPT* by Downey and Fellows (this book).

SHORTEST COMMON SUPERSEQUENCE II

- Instance:* An alphabet Σ , a set of strings $\{r_1, \dots, r_k\}$ formed over alphabet Σ .
- Parameter:* M .
- Question:* Does there exist a string S such that every r_i is a subsequence of S and $|S| \leq M$?

The general version of this problem is *NP*-complete by a reduction from **VERTEX COVER** [343]. This problem is shown to belong to *FPT* by Fellows et al. [207].

STEINER TREE

- Instance:* A graph $G = (V, E)$, a set S of at most k vertices in V , an integer m .
- Parameter:* A positive integer k .
- Question:* Is there a set of vertices $T \subseteq V - S$ such that $|T| \leq m$ and $G[S \cup T]$ is connected?

The general version of this problem is *NP*-complete by a reduction from **EXACT COVER**. (See Ref. [239], ND12.) [241] This problem is in *FPT* by Dreyfus and Wagner [188]. Solvable in time $O(3^k n + 2^k n^2 + n^3)$ by the Dreyfus-Wagner algorithm [462].

STEINER PROBLEM IN HYPERCUBES

- Instance:* k length q binary sequences $X_1, \dots, X_k$ and a positive integer M (given in binary).
- Parameter:* k .
- Question:* Is there a subgraph S of the q -dimensional binary hypercube that includes the vertices $X_1, \dots, X_k$, such that S has at most M edges?

In *FPT* by the method of problem kernel (Exercise 3.2.11).

TREEWIDTH

- Instance:* A graph $G = (V, E)$.
- Parameter:* A positive integer k .
- Question:* Does G have treewidth k ?

The general version of this problem is *NP*-complete [32]. Solvable in $O(n)$ time for fixed k [71].

UNIQUE HITTING SET

- Instance:* A set X and k subsets $X_1, \dots, X_k$ of X .
- Parameter:* k .
- Question:* Is there a set $S \subseteq X$ such that for all i , $1 \leq i \leq k$, $|S \cap X_i| = 1$?

In *FPT* by the method of problem kernel (Exercise 3.2.5).

VERTEX COVER

- Instance:* A graph $G = (V, E)$.
Parameter: A positive integer k .
Question: Does G have a vertex cover of size $\leq k$?

The general version of this problem is *NP*-complete by a reduction from 3SAT. (See Ref. [239], GT1.) This problem is in *FPT* by Downey and Fellows [45, 165] and [100]; and in fact in *LOGSPACE* + *advice* by Cai et al. [110]. Solvable in time $O((53/40)^k k^2 + kn)$ by [46].

A.2 In *FPT* (Nonuniform)

GRAPH LINKING NUMBER

- Instance:* A graph $G = (V, E)$.
Parameter: A positive integer k .
Question: Can G be embedded into 3-space such that the maximum size of a collection of topologically linked disjoint cycles is bounded by k ?

This problem is in *FPT* (nonuniform) by Fellows and Langston [216] after Robertson and Seymour [397]. Solvable in $O(n^3)$ time for fixed k .

GENUS k -COVER

- Instance:* A graph G .
Parameter: k .
Question: Is there a finite graph H of genus at most k that is a cover of G ? [H is a *cover* of G if there is a projection map $p : V(H) \rightarrow V(G)$ such that: (1) p is onto, and (2) if $p(u) = x$ and $xy \in E(G)$, then there is a unique vertex $v \in V(H)$ with $uv \in E(H)$ and $p(v) = y$.]

This problem is in *FPT*(nonuniform) essentially by Robertson and Seymour [397]. Solvable in $O(n^3)$ time for fixed k . This is the only known proof of the decidability of the problem.

UNIT CYCLE AVOIDANCE

- Instance:* A graph G .
Parameter: An abelian group of the form $A = (\mathbb{Z}_2)^m$.
Question: Can the edges of G be labeled by elements of A so that no cycle in G has label sum equal to the identity element of A ?

In nonuniform *FPT* via Robertson and Seymour [397]. Also in nonuniform *FPT* for an arbitrary abelian group A provided that G is a max degree 3 graph.

A.3 In Randomized *FPT*

BOUNDED FACTOR FACTORIZATION

- Instance:* An n -bit positive integer N , a positive integer k .
Parameter: A positive integer k .
Question: Is there a prime factor p of N such that $p < n^k$?

This problem is in randomized *FPT* by Fellows and Koblitz [211, 212].

LINEAR EXTENSION COUNT

- Instance:* A poset $(P, \leq)$.
Parameter: A positive integer k .
Question: Does P have at least k linear extensions?

This problem is in randomized *FPT* by Brightwell and Winkler [95, 96]. Not known to be in *FPT*.

POLYNOMIALLY SMOOTH NUMBER

- Instance:* An n -bit positive integer N .
Parameter: A positive integer k .
Question: Is N n^k -smooth, i.e., is every prime divisor of N bounded by n^k ?

This problem is in randomized *FPT* by Fellows and Koblitz [211, 212]. The notion of n^k -smoothness of n -digit numbers arises naturally in the study of polynomial-time complexity. For example, the concept plays a central role in the proof that primality is in $UP \cap co-UP$.

SMALL PRIME DIVISOR

- Instance:* An n -bit positive integer N , a positive integer k .
Parameter: A positive integer k .
Question: Does N have a nontrivial divisor less than n^k ?

This problem is in randomized *FPT* by Fellows and Koblitz [211, 212].

A.4 $W[1]$ -Complete

ARC-PRESERVING LONGEST COMMON SUBSEQUENCE

- Instance:* Sequences A, B s.t. $|A| = n, |B| = m$, alphabet Σ , arc sets $P_A \subset n \times n, P_B \subset m \times m$.
- Parameter:* k
- Question:* Is there an arc-preserving subsequence S of A and B of length $\geq k$? A subsequence is considered “arc-preserving” if whenever both endpoints of an arc from either A or B are found in S , the corresponding symbols are joined by an arc in the other sequence.

The general version of this problem is NP -complete by a reduction from **CLIQUE** [199]. This problem is $W[1]$ -complete by the same reduction [199]. Remains hard for both $W[1]$ and NP even when arcs do not share endpoints.

BINARY CLADISTIC CHARACTER COMPATIBILITY

- Instance:* A set C of n binary cladistic characters over m objects.
- Parameter:* A positive integer k .
- Question:* Is there a subset $C' \subseteq C, |C'| = k$, such that all pairs of characters in C' are compatible?

The general version of this problem is NP -complete by a reduction from **CLIQUE** [151]. This problem is $W[1]$ -complete by the same reduction [462]. The unconstrained-character version of this problem is also $W[1]$ -complete [462]. If $k = |C|$, one obtains the problem **TRIANGULATING COLORED GRAPHS (PERFECT PHYLOGENY)**.

BINARY QUALITATIVE CHARACTER COMPATIBILITY

- Instance:* A set C of n binary qualitative characters over m objects.
- Parameter:* A positive integer k .
- Question:* Is there a subset $C' \subseteq C, |C'| = k$, such that all pairs of characters in C' are compatible?

The general version of this problem is NP -complete by a reduction from **BINARY CLADISTIC CHARACTER COMPATIBILITY** [151, 462]. This problem is $W[1]$ -complete by the same reduction. The unconstrained-character version of this problem is $W[1]$ -hard [462].

CLIQUE

- Instance:* A graph $G = (V, E)$, a positive integer k .
- Parameter:* A positive integer k .
- Question:* Is there a set of k vertices $V' \subseteq V$ that forms a complete subgraph of G (that is, a clique of size k)?

The general version of this problem is *NP*-complete by a reduction from **VERTEX COVER**. (See Ref. [239], GT19.) This problem is *W*[1]-complete by a reduction from **INDEPENDENT SET** by Downey and Fellows [170]. Fixed-parameter tractable for planar graphs, and for graphs of maximum degree $f(k)$ for any fixed function f .

INDEPENDENT SET

- Instance:* A graph $G = (V, E)$, a positive integer k .
Parameter: A positive integer k .
Question: Is there a set $V' \subseteq V$ of cardinality k , such that $\forall u, v \in V'$, $uv \notin E$?

The general version of this problem is *NP*-complete by a reduction from **VERTEX COVER**. (See Ref. [239], GT20.) This problem is *W*[1]-complete by Downey and Fellows [170]. Fixed-parameter tractable for planar graphs.

IRREDUNDANT SET

- Instance:* A graph $G = (V, E)$, a positive integer k .
Parameter: A positive integer k .
Question: Is there a set $V' \subseteq V$ of cardinality k having the property that each vertex $u \in V'$ has a private neighbor? [A *private neighbor* of a vertex $u \in V'$ is a vertex u' (possibly $u' = u$) with the property that for every vertex $v \in V'$, $u \neq v$, $u' \notin N[v]$.]

IRREDUNDANT SET is in *W*[1] by Downey and Fellows [165]. It is proven to be *W*[1]-complete in Ref. [173].

LONGEST COMMON SUBSEQUENCE (III)

- Instance:* A set of k strings $X_1, \dots, X_k$ over an alphabet Σ , a positive integer m .
Parameter: Positive integers k, m
Question: Is there a string $X \in \Sigma^*$ of length at least m that is a subsequence of X_i for $i = 1, \dots, k$?

The general version of this problem is *NP*-complete by a reduction from **VERTEX COVER**. (See Ref. [239], SR10.) This problem is *W*[1]-complete by a reduction from **CLIQUE** by Bodlaender et al. [75].

MONOTONE QUERY NONEMPTINESS

- Instance:* A database d , a query ϕ which has only existential quantifications and no negations.
Parameter: $|\phi|$
Question: Is the query relation r_ϕ nonempty?

This problem is *W*[1]-complete by Downey et al. [177].

***d*-RED/BLUE NONBLOCKER**

- Instance:* A graph $G = (V, E)$ of maximum degree d where V is partitioned into two color classes $V = V_{\text{red}} \cup V_{\text{blue}}$, a positive integer k .
- Parameter:* A positive integer k .
- Question:* Is there a set of red vertices $V' \subseteq V_{\text{red}}$ of cardinality k such that every blue vertex has at least one neighbor that does not belong to V' ?

This problem is $W[1]$ -complete by [170].

SEMIGROUP EMBEDDING

- Instance:* A semigroup $(S, \cdot)$ and a semigroup $(H, \times)$.
- Parameter:* A semigroup H .
- Question:* Can H be embedded into S ?

This problem is $W[1]$ -complete by a reduction from **CLIQUE** [Downey and Fellows (this book)].

SEMILATTICE EMBEDDING

- Instance:* A semilattice L and a semilattice H .
- Parameter:* A semilattice H .
- Question:* Is H embeddable into L ?

This problem is $W[1]$ -complete by a reduction from **CLIQUE** [Downey and Fellows (this book)].

SET PACKING

- Instance:* A finite family of sets $S = S_1, \dots, S_n$, an integer k .
- Parameter:* A positive integer k .
- Question:* Does S contain a subset R of k mutually disjoint sets?

The general version of this problem is NP -complete by a reduction from **X3C**. (See Ref. [239], SP3.) This problem is $W[1]$ -complete by a reduction from **INDEPENDENT SET** [42, 462].

SHORT CONTEXT SENSITIVE DERIVATION

- Instance:* A context-sensitive grammar $G = (N, \Sigma, \Pi, S)$, a word $x \in \Sigma^*$.
- Parameter:* A positive integer k .
- Question:* Is there a G derivation of x of length at most k ?

The general version of this problem is $PSPACE$ -complete by a reduction from **LINEAR BOUNDED AUTOMATON ACCEPTANCE**. (See Ref. [229], AL20.) This problem is $W[1]$ -complete by a reduction from **CLIQUE** [178, 179].

SHORT SEMI-THUE PROCESS

Input: A semi-Thue process Π , and words w, v .

Parameter: A positive integer k .

Question: Does $w \Rightarrow_{\Pi}^* v$ in $\leq k$ steps?

The general version is undecidable by a generic reduction as in Davis [142]. Proved $W[1]$ -complete in Chapter 11 of this book.

SHORT NONDETERMINISTIC TURING MACHINE COMPUTATION (I)

Instance: A multitape, multihead nondeterministic Turing machine M ; a word x on the input alphabet of M .

Parameter: the number of tapes t ; the number of heads per tape h ; a positive integer k .

Question: Is there a computation of M on input x that reaches a final state in at most k steps?

The general version of this problem is undecidable. (See, for example, Hopcroft and Ullman [270].) The problem restricted to *standard* Turing machines (that is, machines having one tape and one head) is $W[1]$ -complete by a reduction from **CLIQUE** [178]. The problem restricted to standard Turing machine and where there is a bound either on the size of the alphabet or on the number of nondeterministic transition possibilities out of a given state is in FPT. The general problem is $W[1]$ -complete ([119]). It remains $W[1]$ -complete if M has exactly one non-final internal state, empty input, and either two work tapes with one head per tape, or one tape with two heads ([119]). The problem is in FPT if either the size of the alphabet is a parameter, or there is at most one writable tape and the number of internal states is a parameter ([119]).

SHORT POST CORRESPONDENCE

Instance: A Post system Π , a positive integer k .

Parameter: A positive integer k .

Question: Is there a length k solution for Π ?

The classical **POST CORRESPONDENCE** problem is a well-known undecidable problem. This problem is $W[1]$ -complete by a reduction from **SHORT UNRESTRICTED GRAMMAR DERIVATION** [109].

SHORT UNRESTRICTED GRAMMAR DERIVATION

Instance: An unrestricted phrase-structure grammar G , a word x , a positive integer k .

Parameter: A positive integer k .

Question: Is there a G derivation of x of length at most k ?

The general version of this problem is undecidable. (See, for example, Ref. [270].) This problem is $W[1]$ -complete by a reduction from **CLIQUE** [109].

SMALL MINIMUM DEGREE FOUR SUBGRAPH

- Instance:* A graph $G = (V, E)$.
Parameter: k
Question: Is there a subgraph of G of minimum degree at least 4 and of cardinality at most k ?

This problem is shown W[1]-complete by Wareham [462].

SQUARE TILING

- Instance:* A set C of “colors,” a collection $T \subseteq C^4$ of “tiles” (where $\langle a, b, c, d \rangle$ denotes a tile whose top, right, bottom, and left sides are colored a, b, c , and d , respectively), a positive integer $k \leq C$.
Parameter: A positive integer k .
Question: Is there a tiling of a $k \times k$ square using the tiles in T , i.e., an assignment of a tile $A(i, j) \in T$ to each ordered pair i, j , $1 \leq i \leq k$, $1 \leq j \leq k$, such that (1) if $f(i, j) = \langle a, b, c, d \rangle$ and $f(i + 1, j) = \langle a', b', c', d' \rangle$, then $a = c'$, and (2) if $f(i, j) = \langle a, b, c, d \rangle$ and $f(i, j + 1) = \langle a', b', c', d' \rangle$, then $b = d'$.

The general version of this problem is *NP*-complete by a reduction from **DIRECTED HAMILTON PATH**. (See Ref. [239], GP13.) This problem is W[1]-complete by Cai et al. and Downey and Fellows (this book).

VAPNIK-CHEVONENKIS (VC) DIMENSION

- Instance:* A family of subsets F of a base set X , an integer k .
Parameter: A positive integer k .
Question: Is the VC dimension of F at least k ? (The VC dimension of a family of subsets F of a base set X is the maximum cardinality of a set $S \subseteq X$ such that for each subset $S' \subseteq S$, $\exists Y \in F$ such that $S \cap Y = S'$.)

The general version of this problem is *LOGSNP*-complete [368]. This problem is W[1]-complete by a reduction from **CLIQUE** [163]. Membership of W[1] is proven by a generic reduction in Ref. [169].

 t -THRESHOLD STABLE SET

- Instance:* A directed graph $G = (V, A)$.
Parameter: A positive integer k .
Question: Does G have a stable set of size k ? (A *stable set* is a set of vertices $V' \subseteq V$ such that for every vertex v of $V - V'$, there are fewer than t vertices $u \in V'$ with $uv \in A$.)

This problem is $W[1]$ -complete by a reduction from **INDEPENDENT SET**. Complete for $W[1]$ by Downey and Fellows (this book).

WEIGHTED q -CNF SATISFIABILITY

- Instance:* A q -CNF formula X , i.e., a CNF formula such that each clause has no more than q literals.
Parameter: A positive integer k .
Question: Does X have a satisfying assignment of weight k ?

This problem is $W[1]$ -complete by a reduction from **INDEPENDENT SET** [170].

A.5 $W[1]$ -Hard, in $W[2]$

PERFECT CODE

- Instance:* A graph $G = (V, E)$.
Parameter: A positive integer k .
Question: Does G have a k -element perfect code? (A *perfect code* is a set of vertices $V' \subseteq V$ with the property that for each vertex $v \in V$ there is precisely one vertex in $N[v] \cap V'$.)

This problem is $W[1]$ -hard by a reduction from **INDEPENDENT SET** and in $W[2]$ by Downey and Fellows [170]. We believe that it may be of difficulty intermediate between $W[1]$ and $W[2]$.

UNIQUE WEIGHTED CNF SATISFIABILITY

- Instance:* A boolean CNF formula X .
Parameter: A positive integer k .
Question: Is there a unique weight k satisfying assignment for X ?

There are obvious versions of the above for normalized satisfiability at any level of the W -hierarchy. For the CNF situation above, the problem is clearly in the natural analog of D_p . The reader should recall that D_p consists of the class of languages L that can be expressed as the intersection of a language in NP and one in $co - NP$ [367]. Clearly, we can similarly define $D_p[2]$ (or, more generally, $D_p[t]$) as for D_p but with $W[2]$ in place of NP . This problem is in $D_p[2]$ [166]. This class also seems to contain the problem **UNIQUE DOMINATING SET**. At present, we do not know if there are $D_p[t]$ -complete problems; possible candidates might be suitably parameterized Travelling Salesman problems. Note that by using the Valiant-Vazirani technique [456] we can show that **UNIQUE WEIGHTED SATISFIABILITY** for arbitrary boolean formulas is the same as **WEIGHTED SATISFIABILITY** for boolean formulas under randomized reductions. However, it is not clear if this is true for any finite level of the W -hierarchy since, for instance, weight is lost in

the Valiant-Vazirani [456] proof. Nevertheless, using an unpublished argument, we have shown that a weighted version of Ref. [456] holds for $t \geq 2$. This seems a very fruitful area to analyze.

WEIGHTED EXACT CNF SATISFIABILITY

- Instance:* A boolean expression E in conjunctive normal form.
Parameter: A positive integer k .
Question: Is there a truth assignment of weight k to the variables of E that makes exactly one literal in each clause of E true?

This problem is $W[1]$ -hard by a reduction from **PERFECT CODE** and in $W[2]$ by Downey and Fellows [170]. Equivalent to **PERFECT CODE** [170]. A related problem is **UNIQUE WEIGHTED CNF SATISFIABILITY** above.

MAXIMUM-LIKELIHOOD DECODING

- Instance:* A binary $m \times n$ matrix H , a target vector $s \in \mathbb{F}^m$, and an integer $k > 0$.
Parameter: A positive integer k .
Question: Is there a set of at most k columns of H that sum to s ?

Originally proved NP -complete by a reduction from **3DIMENSIONAL MATCHING** by Berlekamp, McEliece and van Tilborg [56]. $W[1]$ hardness and membership of $W[2]$ was proven by Downey, Fellows, Vardy, and Whittle [182].

WEIGHT DISTRIBUTION

- Instance:* A binary $m \times n$ matrix H , and an integer $k > 0$.
Parameter: A positive integer k .
Question: Is there a set of k columns of H that sum to the all-zero vector?

Originally proved NP -complete by a reduction from **3DIMENSIONAL MATCHING** by Berlekamp, McEliece, and van Tilborg [56]. $W[1]$ hardness and membership of $W[2]$ was proven by Downey, Fellows, Vardy, and Whittle [182].

WEIGHT DISTRIBUTION

- Instance:* A binary $m \times n$ matrix H , and an integer $k > 0$
Parameter: k
Question: Is there a set of k columns of H that sum to the all-zero vector?

Originally proved NP -complete by a reduction from **3DIMENSIONAL MATCHING** by Berlekamp, McEliece, and van Tilborg [56]. $W[1]$ hardness and membership of $W[2]$ was proven by Downey, Fellows, Vardy, and Whittle [182].

A.6 $W[1]$ -Hard, in $W[P]$

PERMUTATION GROUP FACTORIZATION

- Instance:* A set A of permutations $A \subseteq S_n, x \in S_n$.
Parameter: A positive integer k .
Question: Does x have a factorization of length k over A ?

This problem is $W[1]$ -hard by a reduction from **PERFECT CODE** and in $W[P]$ by Cai et al. [109].

SUBSET SUM

- Instance:* A set of integers $X = \{x_1, \dots, x_n\}$ and an integer s .
Parameter: A positive integer k .
Question: Is there a subset $X' \subseteq X$ of cardinality k such that the sum of the integers in X' equals s ?

The general version of this problem is NP -complete by a reduction from **PARTITION**. (See Ref. [239], SP13.) This problem is $W[1]$ -hard by a reduction from **PERFECT CODE** by Downey and Fellows [170] and in $W[P]$ by Fellows and Koblitz [212]. Not known to belong to $W[t]$ for any t .

A.7 $W[1]$ -Hard

α -BALANCED SEPARATOR

- Instance:* A graph $G = (V, E)$.
Parameter: A positive integer k .
Question: Does there exist a set of vertices $S, |S| \leq k$, such that every component of $G[V - S]$ has at most $\alpha|V|$ vertices?

This problem is $W[1]$ -hard by a reduction from **CLIQUE** by Bodlaender and Kratsch [88]. This problem is $W[1]$ -hard for every fixed α .

COLORING GRAPH AUTOMORPHISM

- Instance:* A 2-colored (bipartite) graph G .
Parameter: A positive integer k .
Question: Is there an automorphism preserving colors moving exactly k blue vertices?

$W[1]$ -hard by a reduction from **ANTIMONOTONE 2SAT**. (Downey and Fellows (this book)).

COLORED PROPER INTERVAL GRAPH COMPLETION

- Instance:* A graph $G = (V, E)$, a vertex coloring $c : V \rightarrow \{1, \dots, k\}$.
Parameter: A positive integer k .
Question: Does there exist a proper interval supergraph of G which respects c ?

This problem is $W[1]$ -hard by a reduction from **INDEPENDENT SET** by Kaplan and Shamir [288]. This problem is equivalent to **COLORED UNIT INTERVAL GRAPH COMPLETION** as the class of unit interval graphs and proper interval graphs are equivalent.

EXACT CHEAP TOUR

- Instance:* A weighted graph $G = (V, E)$, a weight function $w : E \rightarrow \mathbb{Z}$.
Parameter: A positive integer k .
Question: Is there a tour through at least k nodes of G of cost exactly S ?

The general version of this problem is NP -complete by a reduction from **HAMILTON CIRCUIT**. (See Ref. [239], ND22.) This problem is $W[1]$ -hard by Ref. [170]. See the related **SHORT CHEAP TOUR** problem.

FINITE STATE AUTOMATA INTERSECTION I

- Instance:* A set of k deterministic finite-state automata $A_1, \dots, A_k$ having the same input alphabet Σ .
Parameter: k, m
Question: Is there a string $X \in \Sigma^*$ of length at least m that is accepted by each $A_i, i = 1, \dots, k$?

The general version of this problem is $PSPACE$ -complete by a reduction from **LINEAR SPACE ACCEPTANCE** ([239], AL6.) This problem is $W[1]$ -hard by a reduction from **LONGEST COMMON SUBSEQUENCE III** [462].

INVMAX

- Instance:* A circuit C , an initial configuration $conf_0$ describing the placing of inverters on connections between gates in G .
Parameter: A positive integer k .
Question: Is there a subset A of the gates in G to which DeMorgan's rules can be applied such that the resulting circuit will have at least k gates without any inverters attached to their output lines?

The general version of this problem is NP -complete by a reduction from **INDEPENDENT SET** [462]. This problem is $W[1]$ -hard by the same reduction.

REACHABILITY DISTANCE FOR VECTOR ADDITION SYSTEMS (PETRI NETS)

Instance: A set T of m length n integer-valued vectors $T = \{x^i = (x_1^i, \dots, x_n^i) : 1, \leq i \leq m\}$, a non-negative starting vector $s = (s_1, \dots, s_n)$, a non-negative target vector $t = (t_1, \dots, t_n)$, a positive integer k .

Parameter: A positive integer k .

Question: Is there a choice of k indices $i_1, \dots, i_k$, $1 \leq i_j \leq m$ for $j = 1, \dots, k$ such that $t = s + \sum_{j=1}^k x^{i_j}$ and such that every intermediate sum is non-negative in each component, that is, $s_r + \sum_{j=1}^q x_r^{i_j} \geq 0$ for $q = 1, \dots, k$ and $r = 1, \dots, n$?

This problem is $W[1]$ -hard by a reduction from **CLIQUE** [179].

SUBSET PRODUCT

Instance: A set of integers $X = \{x_1, \dots, x_n\}$, integers a, m .

Parameter: A positive integer k .

Question: Is there a subset $X' \subseteq X$ of cardinality k such that the product of the integers in X' is congruent to $a \bmod m$?

The general version of this problem is NP -complete by a reduction from X3C. (See Ref. [239], SP14.) This problem is $W[1]$ -hard by a reduction from **PERFECT CODE** by Fellows and Koblitz [211, 212].

NEAREST VECTOR

Instance: A basis $X = \{x_1, x_2, \dots, x_n\} \subset \mathbb{Z}^n$ for a lattice $\mathbb{L}$, a target vector $s \in \mathbb{Z}^n$, and an integer $k > 0$.

Parameter: A positive integer k .

Question: Is there a vector $x \in \mathbb{L}$, such that $\|x - s\|^2 \leq k$?

Downey, Fellows, Vardy, and Whittle proved $W[1]$ hardness in [182].

THETA SERIES

Instance: A basis $X = \{x_1, x_2, \dots, x_n\} \subset \mathbb{Z}^n$ for a lattice $\mathbb{L}$, and an integer $k > 0$.

Parameter: A positive integer k .

Question: Is there a vector $x \in \mathbb{L}$, such that $\|x\|^2 = k$?

$W[1]$ hardness and NP -completeness proven by Downey, Fellows, Vardy, and Whittle in [182]. It uses a reduction to **PERFECT CODE**.

EXACT EVEN SET

Instance: A red/blue graph $G = (\mathcal{R}, \mathcal{B}, E)$ and an integer $k > 0$.

Parameter: A positive integer k .

Question: Is there a set of k vertices $R \subseteq \mathcal{R}$ each member of $\mathcal{B}$ having an even number of neighbours in R ?

Proved $W[1]$ -hard by a reduction to **PERFECT CODE** in Downey, Fellows, Vardy, and Whittle [182].

ODD SET

- Instance:* A red/blue graph $G = (\mathcal{R}, \mathcal{B}, E)$ and an integer $k > 0$.
Parameter: A positive integer k .
Question: Is there a set of at most k vertices $R \subseteq \mathcal{R}$ Is there a set of k vertices $R \subseteq \mathcal{R}$ each member of $\mathcal{B}$ having an odd number of neighbours in R ?

Proved $W[1]$ -hard by a reduction to **PERFECT CODE** in Downey, Fellows, Vardy, and Whittle [182].

EXACT ODD SET

- Instance:* A red/blue graph $G = (\mathcal{R}, \mathcal{B}, E)$ and an integer $k > 0$.
Parameter: A positive integer k .
Question: Is there a set of k vertices $R \subseteq \mathcal{R}$ each member of $\mathcal{B}$ having an odd number of neighbours in R ?

Proved $W[1]$ -hard by a reduction to **PERFECT CODE** in Downey, Fellows, Vardy, and Whittle [182].

A.8 W[2]-Complete

DOMINATING CLIQUE

- Instance:* A graph $G = (V, E)$, a positive integer k .
Parameter: A positive integer k .
Question: Is there a set of k vertices $V' \subseteq V$ that forms a complete subgraph of G and is also a dominating set for G ?

This problem is $W[2]$ -hard by a reduction from **DOMINATING SET** [88]. Problem is in *FPT* if V' is also required to be *efficient*, i.e. each vertex not in V' is dominated by exactly one vertex in V' [88].

DOMINATING SET

- Instance:* A graph $G = (V, E)$, a positive integer k .
Parameter: A positive integer k .
Question: Is there a set of k vertices $V' \subseteq V$ with the property that every vertex of G either belongs to V' or has a neighbor in V' ?

The general version of this problem is *NP*-complete by a reduction from **VERTEX COVER**. (See Ref. [239], GT2.) This problem is $W[2]$ -complete by a reduction

from **WEIGHTED CNF SATISFIABILITY** [165]. Fixed-parameter tractable for planar graphs [169]. Problem is $W[2]$ -hard if the dominating set V' is required to be either connected or *total*; i.e., for each vertex in V there is an edge to some vertex in V' [88]. Problem is harder than **PERFECT CODE**, i.e. $W[1]$ -hard, if a general, connected, or total dominating set is also required to be *efficient*, i.e. each vertex not in V' is dominated by exactly one vertex in V' [88].

DOMINATING THRESHOLD SET

- Instance:* A graph $G = (V, E)$, integers k, r .
Parameter: Positive integers k and r .
Question: Is there a set $V' \subseteq V$ of at most k vertices such that for every vertex u , $N[u]$ contains at least r elements of V' ?

This problem is $W[2]$ -complete by Downey and Fellows [172].

HITTING SET

- Instance:* A finite family of sets $S = S_1, \dots, S_n$ comprised of elements from $U = \{u_1, \dots, u_m\}$, an integer k .
Parameter: A positive integer k .
Question: Is there a subset $T \subseteq U$ of size k such that for all $S_i \in S$, $S_i \cap T \neq \emptyset$?

The general version of this problem is NP -complete by a reduction from **VERTEX COVER** (See Ref. [239], SP8). This problem is $W[2]$ -complete by a reduction from **SET COVER** [462].

INDEPENDENT DOMINATING SET

- Instance:* A graph $G = (V, E)$, a positive integer k .
Parameter: A positive integer k .
Question: Is there a set of k vertices $V' \subseteq V$ that is both an independent set and a dominating set in G ?

The general version of this problem is NP -complete by a reduction from **VERTEX COVER**. (See Ref. [239], GT2.) This problem is $W[2]$ -complete by Downey and Fellows [165]. Fixed-parameter tractable for planar graphs [169].

SET COVER

- Instance:* A finite family of sets $S = S_1, \dots, S_n$, an integer k .
Parameter: A positive integer k .
Question: Is there a subset $R \subseteq S$ whose union is all elements in the union of S ?

The general version of this problem is NP -complete by a reduction from **X3C**. (See Ref. [239], SP5.) This problem is $W[2]$ -complete by a reduction from **DOMINATING SET** by Paz and Moran [370] and Wareham [462].

TOURNAMENT DOMINATING SET

- Instance:* A tournament T , a positive integer k .
Parameter: A positive integer k .
Question: Does T have a dominating set of cardinality at most k ?

The general version of this problem is LOGSNP-complete [368]. This problem is $W[2]$ -complete by a reduction from **DOMINATING SET** [172].

WEIGHTED BINARY INTEGER PROGRAMMING

- Instance:* A binary matrix A , a binary vector $\mathbf{b}$, an integer k .
Parameter: A positive integer k .
Question: Does $A \cdot \mathbf{x} \geq \mathbf{b}$ have a binary solution of weight k ?

The general version of this problem is NP -complete by a reduction from 3SAT (See Ref. [239], MP1). This problem is $W[2]$ -complete by a reduction from **MONOTONE WEIGHTED CNF SATISFIABILITY** [166]. The problem **WEIGHTED EXACT BINARY INTEGER PROGRAMMING** which asks that equality hold is hard for $W[1]$.

A.9 $W[2]$ -Hard, in $W[P]$ **MONOCHROME CYCLE COVER**

- Instance:* An edge-colored graph $G = (V, E)$, a positive integer k .
Parameter: A positive integer k .
Question: Is there a set of k vertices $V' \subseteq V$ with the property that every monochrome cycle in G contains a vertex in V' ?

This problem is $W[2]$ -hard and in $W[P]$ by Downey and Fellows (unpubl.). Not known to belong to $W[t]$ for any t .

MONOID FACTORIZATION

- Instance:* A set A of self-maps on $[n]$, a self-map h , an integer k .
Parameter: A positive integer k .
Question: Is there a factorization of h of length k over A ?

This problem is $W[2]$ -hard by a reduction from **DOMINATING SET** and in $W[P]$ by Cai et al. [109].

A.10 $W[2]$ -Hard

LONGEST COMMON SUBSEQUENCE II

- Instance:* A set of k strings $X_1, \dots, X_k$ over an alphabet Σ , a positive integer m .
- Parameter:* A positive integers m
- Question:* Is there a string $X \in \Sigma^*$ of length at least m that is a subsequence of X_i for $i = 1, \dots, k$?

The general version of this problem is NP -complete by a reduction from **VERTEX COVER**. (See Ref. [239], SR10.) This problem is $W[2]$ -hard by a reduction from **DOMINATING SET** [76].

MAXIMAL IRREDUNDANT SET

- Instance:* A graph $G = (V, E)$, a positive integer k .
- Parameter:* A positive integer k .
- Question:* Is there a set $V' \subseteq V$ of cardinality k such that (1) each vertex $u \in V'$ has a private neighbor and (2) V' is not a proper subset of any $V'' \subseteq V$ which also has this property? (A *private neighbor* of a vertex $u \in V'$ is a vertex u' (possibly $u' = u$) with the property that for every vertex $v \in V'$, $u \neq v$, $u' \notin N[v]$.)

This problem is $W[2]$ -hard by a reduction from **DOMINATING SET** [83]. Originally proven to be $W[1]$ -hard by Downey and Fellows. (See Exercise 1.)

PRECEDENCE CONSTRAINED k -PROCESSOR SCHEDULING

- Instance:* A set T of unit-length tasks, a partial order $<$ on T , a positive integer deadline D , a number of processors k .
- Parameter:* A positive integer k .
- Question:* Is there a map $f : T \rightarrow \{1, \dots, D\}$, such that for all $t, t' \in T$, $t < t'$ implies $f(t) < f(t')$, and for all i , $1 \leq i \leq D$, $|f^{-1}(i)| \leq k$?

The general version of this problem is not known to be either NP -complete or in P ([239], OPEN8), but it is $W[2]$ -hard by a reduction from **DOMINATING SET** [79].

STEINER TREE

- Instance:* A graph $G = (V, E)$, a set S of at most k vertices in V , an integer m .
- Parameter:* Positive integers k and m .
- Question:* Is there a set of vertices $T \subseteq V - S$ such that $|T| \leq m$ and $G[S \cup T]$ is connected?

The general version of this problem is *NP*-complete by a reduction from **EXACT COVER**. (See Ref. [239], ND12; GKR.) This problem is $W[2]$ -hard by a reduction from **DOMINATING SET** [82].

SHORT NONDETERMINISTIC TURING MACHINE COMPUTATION (II)

- Instance:* A multitape, multi-head nondeterministic Turing machine M ; a word x on the input alphabet of M .
- Parameter:* A positive integer k .
- Question:* Is there a computation of M on input x that reaches a final state in at most k steps?

The general version of this problem is undecidable. (See, for example, Hopcroft and Ullman [270].) the problem is in $W[P]$ and $W[2]$ -hard by a reduction from **DOMINATING THRESHOLD SET** ([119]). It remains $W[2]$ -hard when the instances are restricted to machines having just one head per tape, empty input, exactly one non-final internal state and binary alphabet ([119]). It remains also $W[2]$ -hard when the machines have just one tape and an unbounded number of heads ([119]). Finally, the problem restricted to *total* Turing machines (that is, machines such that for any global configuration there exists an applicable transition) is $W[1]$ -complete ([119]).

A.11 $W[t]$ -Complete

WEIGHTED t -NORMALIZED SATISFIABILITY

- Instance:* A t -normalized boolean expression X , a positive integer k .
- Parameter:* A positive integer k .
- Question:* Does X have a satisfying truth assignment of weight k ?

This problem is $W[t]$ -complete by Downey and Fellows [166].

WEIGHTED t -NORMALIZED $W[*]$ SATISFIABILITY

- Instance:* A boolean expression E that is an n -product of k -sums of n -products of k -sums . . . (repeated so that the number of product terms is t).
- Parameter:* k
- Question:* Is there a weight k truth assignment that satisfies E ?

This problem is $W[t]$ -complete by Downey et al. [168]. The argument for completeness proceeds from first principles and follows the same style as the basic proofs for $W[t]$ completeness of **WEIGHTED t -NORMALIZED SATISFIABILITY**.

$\leq k$ WEIGHTED t -NORMALIZED SATISFIABILITY

Instance: A boolean formula X .

Parameter: A positive integer k .

Question: Does X have a satisfying assignment of weight $\leq k$?

For $t \geq 2$ this problem is $W[t]$ -complete by Cai and Chen [103, 104, 105]. This fact also follows by the main lemma of Downey and Fellows [166]. For $t = 1$, the problem is in FPT .

A.12 $W[t]$ -Hard, for All t , in $W[P]$

SHORT PHONOLOGICAL SEGMENTAL DECODING

Instance: An integer k , a simplified segmental grammar $s = (F, S, D, R, c_p, C)$ such that the number of mutually exclusive rule sets in R , $|R_{m.e.}|$, is at most k , string $s \in S^+$.

Parameter: A positive integer k .

Question: Is there a string $u \in D$ such that $g(u) = s$?

The general version of this problem is NP -complete by a reduction from **CLIQUE** [387]. This problem is $W[t]$ -hard by a reduction from **WEIGHTED t -NORMALIZED SATISFIABILITY** and in $W[P]$ by Downey et al. [170]. See Downey et al. [170] and Ristad [387] for definitions of simplified segmental grammars. The same proof also implies the $W[t]$ -hardness and membership in $W[P]$ of **SHORT PHONOLOGICAL SEGMENTAL ENCODING**.

A.13 $W[t]$ -Hard, for All t

BANDWIDTH

Instance: A graph $G = (V, E)$, a positive integer k .

Parameter: A positive integer k .

Question: Is there a 1:1 linear layout $f : V \rightarrow \{1, \dots, |V|\}$ such that $uv \in E$ implies $|f(u) - f(v)| \leq k$?

The general version of this problem is NP -complete by a reduction from **3PARTITION** (See Ref. [239], ND40). This problem is $W[t]$ -hard by a reduction from **UNIFORM EMULATION ON A PATH** [76]. Remains $W[t]$ -hard for all t when the given graph is directed and the layout must respect arc direction, or when the given graph is a tree [79]. The related problem **CUTWIDTH** is in FPT [205].

COLORED CUTWIDTH

- Instance:* A graph $G = (V, E)$, an edge coloring $c : E \rightarrow \{1, \dots, r\}$, a positive integer k .
- Parameter:* A positive integer k .
- Question:* Is there a 1:1 linear layout $f : V \rightarrow \{1, \dots, |V|\}$ such that for each color $j \in \{1, \dots, k\}$ and for each i , $1 \leq i \leq |V| - 1$, we have $|\{uv : c(uv) = j \text{ and } f(u) \leq i \text{ and } f(v) \geq i + 1\}| \leq r$?

This problem is $W[t]$ -hard by a reduction from **LONGEST COMMON SUBSEQUENCE I** [79, 82]. The directed version of this problem where $f(u) < f(v)$ if $(u, v) \in E$ is also hard for $W[t]$, for all t [82]. Furthermore, the **MODIFIED COLORED CUTWIDTH** of a layout $f : V \rightarrow \{1, \dots, |V|\}$ is defined as follows: For each color $j \in \{1, \dots, k\}$ and for each i , $1 \leq i \leq |V| - 1$, we have $|\{uv : c(uv) = j \text{ and } f(u) < i \text{ and } f(v) \geq i + 1\}| \leq r$. This variant also remains hard for $W[t]$, for all t [82].

DOMINO TREewidth

- Instance:* A graph $G = (V, E)$.
- Parameter:* A positive integer k .
- Question:* Is the domino treewidth of G at most k ?

The general version of this problem is NP -complete by a reduction from **LONGEST COMMON SUBSEQUENCE I** by Bodlaender and Engelfreit [77]. This problem is $W[t]$ -hard by the same reduction.

FEASIBLE REGISTER ASSIGNMENT

- Instance:* A directed acyclic graph $G = (V, E)$, a positive integer k , a register assignment $r : V \rightarrow \{R_1, \dots, R_k\}$.
- Parameter:* A positive integer k .
- Question:* Is there a linear ordering f of G , and a sequence $S_0, S_1, \dots, S_{|V|}$ of subsets of V , such that $S_0 = \emptyset$, $S_{|V|}$ contains all vertices of in-degree 0 in G , and for all i , $1 \leq i \leq |V|$, $f^{-1}(i) \in S_i$, $S_i - \{f^{-1}(i)\} \subseteq S_{i-1}$ and S_{i-1} contains all vertices u for which $(f^{-1}(i), u) \in E$, and for all j , $1 \leq j \leq k$, there is at most one vertex $u \in S_i$ with $r(u) = R_j$?

The general version of this problem is NP -complete by a reduction from **3SAT**. (See Ref. [239], PO2.) This problem is $W[t]$ -hard by a reduction from **LONGEST COMMON SUBSEQUENCE I** [79].

FINITE STATE AUTOMATA INTERSECTION II

Instance: A set of k deterministic finite-state automata $A_1, \dots, A_k$ having the same input alphabet Σ , a positive integer m .

Parameter: k .

Question: Is there a string $X \in \Sigma^*$ of length at least m that is accepted by each $A_i, i = 1, \dots, k$?

The general version of this problem is *PSPACE*-complete by a reduction from **LINEAR SPACE ACCEPTANCE**; see Ref. [239], AL6. This problem is $W[t]$ -hard by a reduction from **LONGEST COMMON SUBSEQUENCE IV** [462]. Solvable in polynomial time for any fixed k [271].

FINITE STATE AUTOMATA INTERSECTION III

Instance: A set of k deterministic finite-state automata $A_1, \dots, A_k$ having the same input alphabet Σ , a positive integer m .

Parameter: $k, |\Sigma|$.

Question: Is there a string $X \in \Sigma^*$ of length at least m that is accepted by each $A_i, i = 1, \dots, k$?

The general version of this problem is *PSPACE*-complete by a reduction from **LINEAR SPACE ACCEPTANCE**; see Refs. [239], AL6. This problem is $W[t]$ -hard by a reduction from **LONGEST COMMON SUBSEQUENCE IV** [462].

LONGEST COMMON SUBSEQUENCE I

Instance: A set of k strings $X_1, \dots, X_k$ over an alphabet Σ , a positive integer m .

Parameter: A positive integer k .

Question: Is there a string $X \in \Sigma^*$ of length at least m that is a subsequence of X_i for $i = 1, \dots, k$?

The general version of this problem is *NP*-complete by a reduction from **VERTEX COVER**. (See Ref. [239], SR10.) This problem is $W[t]$ -hard by a reduction from **MONOTONE WEIGHTED t -NORMALIZED SATISFIABILITY** [75].

LONGEST COMMON SUBSEQUENCE IV

Instance: A set of k strings $X_1, \dots, X_k$ over an alphabet Σ , a positive integer m .

Parameter: A positive integer $k, |\Sigma|$.

Question: Is there a string $X \in \Sigma^*$ of length at least m that is a subsequence of X_i for $i = 1, \dots, k$?

The general version of this problem is *NP*-complete by a reduction from **VERTEX COVER**. (See Ref. [239], SR10.) This problem is $W[t]$ -hard by a reduction from **LONGEST COMMON SUBSEQUENCE I** [75]. The reduction requires the size of Σ to

grow as a function of the parameter. The complexity when $|\Sigma|$ is a fixed constant is open.

MODULE ALLOCATION ON GRAPHS OF BOUNDED TREewidth

Instance: A set of modules $M = \{1, 2, \dots, m\}$, a set of processors $P = \{1, 2, \dots, p\}$, a cost function $e : (M \times P) \rightarrow \mathbf{R} : (x, y) \mapsto t$ where t is the cost of executing module $x \in M$ on processor $y \in P$, a communication cost function $C : (M \times P \times M \times P) \rightarrow \mathbf{R} : (x, y, x', y') \mapsto t$ where t is the communication cost when module x is assigned to processor y and module x' is assigned to processor y' , a communication graph $G = (M, E)$, and a positive real number l .

Parameter: $\text{treewidth}(G) = k$.

Question: Does there exist an assignment of modules to processors such that the total cost of execution is less than or equal to l ?

The general version of this problem is NP -hard. This problem is $W[t]$ -hard by Bodlaender et al. [78, 79].

SHORTEST COMMON SUPERSEQUENCE I

Instance: An alphabet Σ , a set of strings $\{r_1, \dots, r_k\}$ formed over Σ , an integer M .

Parameter: k .

Question: Does there exist a string S such that every r_i is a subsequence of S and $|S| \leq M$?

The general version of this problem is NP -complete by a reduction from **VERTEX COVER** ([239], SR8). This problem is $W[t]$ -hard, for all t , by a reduction from **LCS-1** Hallett [257, 258]. The reduction allows for an arbitrary alphabet size. The complexity of this problem for fixed-sized Σ remains open.

SHORTEST COMMON SUPERSEQUENCE III

Instance: An alphabet Σ , a set of strings $\{r_1, \dots, r_k\}$ formed over Σ , an integer M .

Parameter: $k, |\Sigma|$.

Question: Does there exist a string S such that every r_i is a subsequence of S and $|S| \leq M$?

The general version of this problem is NP -complete by a reduction from **VERTEX COVER** ([239], SR8). This problem is hard for $W[t]$, for all t , by Hallett [257, 258].

RESTRICTED COMPLETION TO A PROPER INTERVAL GRAPH WITH BOUNDED CLIQUE SIZE

- Instance:* A graph G together with a set E of edges prohibited from G' below.
- Parameter:* A positive integer k .
- Question:* Is there a $G' \supset G$ which is a proper interval graph and has clique size at most k , and G' has no edges from E ?

The general problem is *NP*-complete and the parameterized problem hard for $W[t]$ for all t by Kaplan et al. [288, 289]. An restricted version of this problem is **PROPER INTERVALIZING COLORED GRAPHS** where the set of prohibited edges is encoded by the vertex-coloring. This version also remains hard for $W[t]$, for all t [288]. In fact, it remains $W[t]$ hard even if $E = \emptyset$ [288].

TOPOLOGICAL BANDWIDTH

- Instance:* A connected graph $G = (V, E)$.
- Parameter:* k .
- Question:* Is the topological bandwidth of G at most k ? The *topological bandwidth* of a graph G is the minimum bandwidth over all graphs obtainable by subdividing the edges of G .

The general version of this problem is *NP*-complete. This problem is $W[t]$ -hard, for all t , by a reduction from **BANDWIDTH** [79].

TRIANGULATING COLORED GRAPHS

- Instance:* A graph $G = (V, E)$, vertex coloring $c : V \rightarrow \{1, \dots, k\}$.
- Parameter:* A positive integer k .
- Question:* Does there exist a supergraph $G' = (V, E')$ where $E \subseteq E'$ and G' is properly colored by c and G' is triangulated?

The general version of this problem is *NP*-complete by a reduction from **INDEPENDENT SET** [79]. This problem is $W[t]$ -hard by a reduction from **LONGEST COMMON SUBSEQUENCE I** [79].

UNIFORM EMULATION FOR A DIRECTED GRAPH

- Instance:* A directed acyclic graph $G = (V, E)$.
- Parameter:* An integer k .
- Question:* Does there exist a function $f : V \rightarrow \{1, \dots, |V|/k\}$ such that for all $uv \in E$ implies $f(u) - f(v) \in \{0, 1\}$ and for all i , $|f^{-1}(i)| \leq k$?

This problem is $W[t]$ -hard, for all t by a reduction from **WEIGHTED t -NORMALIZED CIRCUIT SATISFIABILITY** by Bodlaender et al. [79, 80].

UNIFORM EMULATION ON A PATH

- Instance:* A graph $G = (V, E)$.
Parameter: A positive integer k .
Question: Does there exist a function $f : V \rightarrow \{1, \dots, |V|/k\}$ such that for all $uv \in E$ implies $|f(u) - f(v)| \leq 1$ and for all i , $|f^{-1}(i)| \leq k$?

This problem is $W[t]$ -hard by a reduction from **MONOTONE WEIGHTED t -NORMALIZED SATISFIABILITY** [79]. Remains $W[t]$ -hard for all t when given graph is a tree [79].

A.14 $W[\text{SAT}]$ -Hard**3-DIMENSIONAL EUCLIDEAN GENERALIZED MOVER'S PROBLEM**

- Instance:* A set O of obstacle polyhedra, a set P of polyhedra which are freely linked together at a set of linkage vertices V such that P has k degrees of freedom of movement, and initial and final positions p_I and p_F of P in 3-dimensional Euclidean space.
Parameter: k .
Question: Is there a legal movement of P from p_I to p_F ; i.e., is there a continuous sequence of translation and rotations of the polyhedra in P such that at each point in time, no polyhedron in P intersects any polyhedron in O and the polyhedra in P intersect themselves only at the linkage vertices in V ?

The general version of this problem is $PSPACE$ -complete by a reduction from **LINEAR SPACE ACCEPTANCE** [384]. This problem is $W[\text{SAT}]$ -hard by a reduction from **COMPACT DETERMINISTIC TURING MACHINE COMPUTATION I** by Cesati and Wareham [123]. Although polynomial-time algorithms are known for this problem for very limited kinds of robots, e.g., line segments, disks, rectangles (see Ref. [401] and references), the best known algorithm for arbitrarily complex robots requires $O(n^k(\log n \cdot d^{O(k)} + 2dk^{O(k^2)}))$ time (Canny [113]), where n is the number of polynomials required to describe the surfaces of the robot and its environment and d is the maximum degree of these polynomials.

A.15 $W[P]$ -Complete **k -BASED TILING**

- Instance:* A tiling system with distinguished tiles.
Parameter: A positive integer k .
Question: Is there a tiling of the $n \times n$ plane using the tiling system and starting with exactly k distinguished tiles in a line?

Reduction consists of a generic simulation of a Turing machine [Downey and Fellows (this book), see Exercise 13.0.4.]

CHAIN REACTION CLOSURE

- Instance:* A directed graph $D = (V, A)$.
Parameter: A positive integer k .
Question: Does there exist a set V' of k vertices of D whose chain reaction closure is D ? (A *chain reaction closure* of V' is the smallest superset S of V' such that if $u, u' \in S$ and arcs $ux, u'x$ are in D , then $x \in S$.)

This problem is $W[P]$ -complete by a reduction from **WEIGHTED MONOTONE CIRCUIT SATISFIABILITY** [4].

DEGREE 3 SUBGRAPH ANNIHILATOR

- Instance:* A graph $G = (V, E)$.
Parameter: A positive integer k .
Question: Is there a set of k vertices $V' \subseteq V$ such that $G - V'$ has no subgraph of minimum degree 3?

This problem is $W[P]$ -complete by a reduction from **WEIGHTED MONOTONE CIRCUIT SATISFIABILITY** [4].

k -INDUCED 3CNF SATISFIABILITY

- Instance:* A 3CNF formula φ .
Parameter: A positive integer k .
Question: Is there a set of k variables and a truth table assignment to those variables that causes φ to unravel?

This problem is $W[P]$ -complete by a reduction from **CHAIN REACTION CLOSURE** [4, 5].

k -INDUCED SATISFIABILITY

- Instance:* A boolean formula φ .
Parameter: A positive integer k .
Question: Is there a set of k variables and a truth table assignment to those variables that causes φ to unravel?

This problem is $W[P]$ -complete by a reduction from k -INDUCED 3CNF SATISFIABILITY [4].

LINEAR INEQUALITIES

- Instance:* A system of linear inequalities.
Parameter: A positive integer k .
Question: Can we delete k of the equalities and get a system that is consistent over the rationals?

This problem is $W[P]$ -complete by a reduction from **WEIGHTED MONOTONE CIRCUIT SATISFIABILITY** taken from Abrahamson et al. [4, 5].

MINIMUM AXIOM SET

- Instance:* A finite set S of sentences, an implication relation R consisting of pairs (A, t) where $A \subseteq S$ and $t \in S$.
- Parameter:* A positive integer k .
- Question:* Is there a set $S_0 \subseteq S$ with $|S_0| \leq k$ and a positive integer n such that if we define S_i , $1 \leq i \leq n$, to consist of exactly those $t \in S$ for which either $t \in S_{i-1}$ or there exists a set $U \subseteq S_{i-1}$ such that if $(U, t) \in R$, then $S_n = S$?

The general version of this problem is NP -complete by a reduction from X3C. (See Ref. [239], L017.) This problem is $W[P]$ -complete by a reduction from **WEIGHTED CIRCUIT SATISFIABILITY** [4, 179].

SHORT CIRCUIT SATISFIABILITY

- Instance:* A boolean circuit C with n gates and at most $k \log n$ inputs and one output.
- Parameter:* A positive integer k .
- Question:* Is there a setting of the inputs that cause C to output 1?

This problem is $W[P]$ -complete by a reduction from **WEIGHTED CIRCUIT SATISFIABILITY** [4, 5].

SHORT SATISFIABILITY

- Instance:* A formula φ on n variables, a list of at most $k \log n$ variables of φ .
- Parameter:* A positive integer k .
- Question:* Is there any setting of the distinguished variables that causes φ to unravel?

This problem is $W[P]$ -complete by a reduction from **SHORT CIRCUIT SATISFIABILITY** [4].

THRESHOLD STARTING SET

- Instance:* A directed graph $D = (V, A)$.
- Parameter:* A positive integer k .
- Question:* Does G have a *starting set* of size k ? (A *starting set* is a set of vertices $V' \subseteq V$ with the property that if we begin with a pebble on each of the vertices in V' and subsequently place pebbles on any vertex having at least t incoming arcs from pebbled vertices, then eventually every vertex of the graph is pebbled.)

This problem is $W[P]$ -complete by a reduction from **WEIGHTED MONOTONE CIRCUIT SATISFIABILITY** [4].

WEIGHTED MONOTONE CIRCUIT SATISFIABILITY

- Instance:* A boolean monotone circuit C .
Parameter: A positive integer k .
Question: Is there a weight k input vector accepted by C ?

This problem is $W[P]$ -complete by a reduction from **MINIMUM AXIOM SET** [4, 179].

WEIGHTED PLANAR CIRCUIT SATISFIABILITY

- Instance:* A planar decision circuit C .
Parameter: A positive integer k .
Question: Does C have a satisfying assignment of weight k ?

This problem is $W[P]$ -complete by a reduction from **WEIGHTED CIRCUIT SATISFIABILITY** [4].

A.16 $AW[*] = AW[1] = AW[t]$ -Complete

Note that Abrahamson et al. proved that $AW[*] = AW[2]$ in Refs. [3] and [4]. This was recently improved to show that $AW[*] = AW[1]$ (and in fact $N[*] = N[1]$) by Downey et al. [174]).

PARAMETERIZED QBFSAT_t

- Instance:* An integer r , a sequence $s_1, \dots, s_r$ of pairwise disjoint sets of boolean variables, a boolean formula X involving the variables $s_1 \cup \dots \cup s_r$ which consists of t alternating layers of conjunctions and disjunctions with negations applied only to variables, integers $k_1, \dots, k_r$.
Parameter: $r, k_1, \dots, k_r$.
Question: Is it the case that there exists a size k_1 subset t_1 of s_1 such that for every size k_2 subset t_2 of s_2 , there exists a size k_3 subset t_3 of s_3 such that $\dots$ (alternating quantifiers) such that, when the variables in $t_1 \cup \dots \cup t_r$ are made true and all other variables are made false, formula X is true?

SHORT GENERALIZED GEOGRAPHY

- Instance:* A directed graph $D = (V, A)$, a specified vertex $v_0 \in V$.
Parameter: A positive integer k .
Question: Does player one have a winning strategy in k moves for the following game? Players alternately choose a new arc from A . The first arc chosen must have its tail at v_0 and each subsequently chosen arc must have its tail at the vertex that was the head of the previous arc. The first player unable to choose a new arc loses.

The general version of this problem is *PSPACE*-complete by a reduction from **QBFSAT**. (See Ref. [239], GP2.) This problem is $AW[*]$ -complete by a reduction from **UNITARY PARAMETERIZED QBFSAT** _{t} [4].

SHORT NODE KAYLES

- Instance:* A graph $G = (V, E)$.
Parameter: A positive integer k .
Question: Does player one have a winning k -move strategy in the following game? That is, players pebble a vertex not adjacent to any pebbled vertex. The first player with no play loses. Player one plays first.

The general version of this problem is *PSPACE*-complete by a reduction from **QBFSAT**. (See Ref. [239], GP3.) This problem is $AW[*]$ -complete by a reduction from **UNITARY PARAMETERIZED QBFSAT** _{t} [4].

UNITARY PARAMETERIZED QBFSAT _{t}

- Instance:* An integer r , a sequence $s_1, \dots, s_r$ of pairwise disjoint sets of boolean variables, a boolean formula X involving the variables $s_1 \cup \dots \cup s_r$ which consists of t alternating layers of conjunctions and disjunctions with negations applied only to variables.
Parameter: A positive integer k .
Question: Is it the case that there exists a variable t_1 of s_1 such that for every variable t_2 of s_2 , there exists a variable t_3 of s_3 such that $\dots$ (alternating quantifiers) such that, when the variables in $t_1, \dots, t_r$ are made true and all other variables are made false, formula X is true?

This problem is $AW[*]$ -complete by Abrahamson et al. [3, 4].

QUERY NONEMPTYNESS

- Instance:* A database d , a query ϕ .
Parameter: $|\phi|$
Question: Is the query relation r_ϕ nonempty?

The general version of this problem is *NP*-complete. This problem is *AW[*]*-complete by Downey et al. [177].

A.17 *AW*[SAT]-Complete

PARAMETERIZED QBFSAT

- Instance:* An integer r , a sequence $s_1, \dots, s_r$ of pairwise disjoint sets of boolean variables, a boolean formula X involving the variables $s_1 \cup \dots \cup s_r$, integers $k_1, \dots, k_r$.
- Parameter:* Positive integers $r, k_1, \dots, k_r$.
- Question:* Is it the case that there exists a size k_1 subset t_1 of s_1 such that for every size k_2 subset t_2 of s_2 , there exists a size k_3 subset t_3 of s_3 such that $\dots$ (alternating quantifiers) such that, when the variables in $t_1 \cup \dots \cup t_r$ are made true and all other variables are made false, formula X is true?

PARAMETERIZED MONOTONE QBFSAT

This is the same as **PARAMETERIZED QBFSAT** except that the formulas are monotone [4].

PARAMETERIZED ANTIMONOTONE QBFSAT

This is the same as **PARAMETERIZED QBFSAT** except that the formulas are antimonotone [4].

A.18 *AW*[SAT]-Hard

COMPACT DETERMINISTIC TURING MACHINE COMPUTATION

- Instance:* A multitape, multihead deterministic Turing machine M ; a word x on the input alphabet of M .
- Parameter:* A positive integer k .
- Question:* Is there a computation of M on input x that reaches a final state and uses on any tape at most k cells?

The problem is *AW*[SAT]-hard by a reduction from **PARAMETERIZED QBFSAT** ([119]). The problem remains *AW*[SAT]-hard when either restricted to *standard* Turing machines (one tape, one head per tape) with empty input and exactly one non-final internal state, or restricted to singlehead machines having empty input, one non-final internal state and binary alphabet ([119]). With respect to membership in the *W*-hierarchy, this problem is not in *XP* (unless $P = PSPACE$) even when restricted to singlehead machines with empty input, one non-terminal internal state and binary alphabet ([119]). However, the problem is in *XP* if restricted

to singlehead machines such that either the number of tapes is a parameter, or there is only one writable tape and the input is the empty word ([119]). Finally, the problem is FTP when restricted to single head machines such that the number of tapes and the size of the alphabet are parameters ([119]).

This problem is $AW[SAT]$ -hard by a reduction from **PARAMETERIZED QBFSAT** $(r, k_1, \dots, k_r)$ [122].

A.19 $AW[P]$ -Complete

PARAMETERIZED QCSAT

- Instance:* An integer r , a sequence $s_1, \dots, s_r$ of pairwise disjoint sets of boolean variables, a circuit X with the variables $s_1 \cup \dots \cup s_r$ as inputs, integers $k_1, \dots, k_r$.
- Parameter:* $r, k_1, \dots, k_r$.
- Question:* Is it the case that there exists a size k_1 subset t_1 of s_1 such that for every size k_2 subset t_2 of s_2 , there exists a size k_3 subset t_3 of s_3 such that $\dots$ (alternating quantifiers) such that, when the inputs in $t_1 \cup \dots \cup t_r$ are set to 1 and all other inputs are set to 0, circuit X outputs 1?

PARAMETERIZED MONOTONE QCSAT

- Instance:* An integer r , a sequence $s_1, \dots, s_r$ of pairwise disjoint sets of boolean variables, a monotone circuit X with the variables $s_1 \cup \dots \cup s_r$ as inputs, integers $k_1, \dots, k_r$.
- Parameter:* $r, k_1, \dots, k_r$.
- Question:* Is it the case that there exists a size k_1 subset t_1 of s_1 such that for every size k_2 subset t_2 of s_2 , there exists a size k_3 subset t_3 of s_3 such that $\dots$ (alternating quantifiers) such that, when the inputs in $t_1 \cup \dots \cup t_r$ are set to 1 and all other inputs are set to 0, circuit X outputs 1?

This problem is $AW[P]$ -complete by a reduction from **PARAMETERIZED QCSAT** [4] and **MINIMUM AXIOM SET** [178].

PARAMETERIZED ANTIMONOTONE QCSAT

This is the same as **PARAMETERIZED QCSAT** except the circuit must be antimonotone. $AW[P]$ -complete by a reduction from **PARAMETERIZED MONOTONE QCSAT** [4].

A.20 $AW[P]$ -Hard

COMPACT NONDETERMINISTIC TURING MACHINE COMUTATION

- Instance:* A multitape, multihead nondeterministic Turing machine M ; a word x on the input alphabet of M .
- Parameter:* A positive integer k .
- Question:* Is there a comutation of M on input x that reaches a final state and uses on any tape at most k cells?

This problem is $AW[P]$ -hard by a reduction from **PARAMETERIZED QCSAT** even when restricted to standard (one tape, one head per tape) Turing machines (Abrahamson, Downey, and Fellows [3]). Moreover, the problem remains $AW[P]$ -hard when restricted to machines with empty input, exactly one non-final internal state and binary alphabet ([119]). With respect to the membership and in the W -hierarchy, the problem is not in XP (unless $P = PSPACE$ even for Turning machines having empty input, one non-final internal state, and binary alphabet ([119]). However, the problem is in XP either when restricted to machines with at most one writable tape and empty input, or when the number of tapes is a parameter, or finally when the transition table is *total* (that is, for any global configuration there exists an applicable transition) ([119]).

A.21 XP -Complete

The following problems are all XP -complete by Adachi et al. [10, 11]. We refer the reader to Chapter 15 for detailed descriptions of the problems.

PEBBLE GAME PROBLEM

- Input:* A **PEBBLE GAME** (N, R, S, T) .
- Parameter:* A positive integer k .
- Question:* If $|S| = k$ determine if I has a winning strategy.

PEG GAME PROBLEM

- Input:* A **PEG GAME** $G = (V, k, l)$.
- Parameter:* A positive integer k .
- Question:* Does I have a winning strategy?

CAT AND MOUSE PROBLEM

- Input:* A **CAT AND MOUSE GAME** $G = (X, E, c, M, v)$.
- Parameter:* A positive integer k .
- Question:* If $|M| = k$, does I have a winning strategy?

Appendix B

Research Horizons

In this final section, we attempt to give something of an overview of the subject of parameterized complexity as it has so far developed and to indicate some of the interesting open problems and research directions that we see.

B.1 A Lineup of *FPT* Suspects

In the theory of *NP*-completeness, there eventually developed some intuitive sense of what problems seemed to offer enough encoding ability to possibly be hard, and which ones did not. In the famous book by Garey and Johnson [239], three problems in particular were singled out as seeming to lack enough “leeway” for the building of gadgets, and thus as likely candidates *not* to be *NP*-complete.

In the same way, though perhaps with somewhat less experience, we point to the following problems as seeming to lack the gadgeteering leeway required for *W[1]*-hardness, and we conjecture that they are *FPT*.

TOPOLOGICAL CONTAINMENT

Instance: A graph $G = (V, E)$.

Parameter: A graph H .

Question: Is H topologically contained in G ?

The general problem is easily seen to be *NP*-complete by a reduction from **CLIQUE**.

IMMERSION ORDER TEST

- Instance:* A graph $G = (V, E)$ and a graph $H = (V', E')$.
Parameter: A graph H .
Question: Is $H \leq_{\text{immersion}} G$ where $\leq_{\text{immersion}}$ denote the immersion ordering?

DIRECTED FEEDBACK VERTEX SET

- Instance:* A directed graph $D = (V, A)$.
Parameter: A positive integer k .
Question: Is there a set S of k vertices such that each directed cycle of G contains a member of S ?

PLANAR DIRECTED DISJOINT PATHS

- Input:* A directed planar graph G and k pairs $\langle r_1, s_1 \rangle, \dots, \langle r_k, s_k \rangle$ of vertices of G .
Parameter: A positive integer k .
Question: Does G have k vertex-disjoint paths $P_1, \dots, P_k$ with P_i running from r_i to s_i ?

The general form of this problem is *NP*-complete by a reduction from 3SAT, even in the undirected case. (See Ref. [239], ND40.) The problem is *NP*-complete for $k = 2$ if planarity is removed [229]. The planar case is known to be solvable in $O(n^{f(k)})$ time by Schrijver [412].

PLANAR t -NORMALIZED WEIGHTED SATISFIABILITY

- Instance:* A planar t -normalized formula X and an integer k .
Parameter: A positive integer k .
Question: Does X have a satisfying assignment of weight k ?

PLANAR MULTIWAY CUT

- Instance:* A weighted planar graph $G = (V, E)$ with terminals $\{x_1, \dots, x_k\}$, and a positive integer M .
Parameter: k .
Question: Is there a set of edges of total weight $\leq M$ whose removal disconnects each terminal from all the others?

The general version of this problem is *NP*-complete by Dahlhaus et al. [147]. Best known complexity is $O((4^k)^k n^{2k-1} \log n)$ by Ref. [147], where it is asked if the problem is *FPT*.

B.2 A Lineup of Tough Customers

Hardness for $W[1]$ is typically more difficult to prove than NP -completeness. The following is a list of problems that are important in various ways, and that have so far resisted strenuous efforts to show that they are hard for $W[1]$ and, therefore, unlikely to be fixed-parameter tractable. Nevertheless, they seem (in the sense of being flexible enough for gadgeteering) to be problems that are probably intractable.

FIXED ALPHABET LONGEST COMMON SUBSEQUENCE (LCS)

- Instance:* k sequences X_i over an alphabet Σ of fixed size (such as $|\Sigma| = 4$ for DNA sequences), and a positive integer m .
- Parameter:* A positive integer k .
- Question:* Is there a string $X \in \Sigma^*$ of length m that is a subsequence of each of the X_i .

The LCS problem for k sequences is known to be hard for $W[t]$ for all t when the alphabet is bounded by the parameter k by a very intricate reduction—but this is not the problem that is really of interest to biologists. A simple dynamic programming algorithm that is widely used in the packages available to molecular biologists runs in time $O(n^k)$ and is frequently applied with $k = 5$ (with the help of supercomputers). This is an almost ideal target for parameterized complexity analysis or algorithm design—yet it seems to be very difficult to resolve.

BOUNDED HAMMING WEIGHT DISCRETE LOGARITHM

- Instance:* An n -bit prime p , a generator g of F_p^* , an element $a \in F_p^*$, a positive integer k .
- Parameter:* A positive integer k .
- Question:* Is there a positive integer x whose binary representation has at most k 1's (that is, x has a Hamming weight of k) such that $a = g^x$?

CROSSING NUMBER

- Instance:* A graph $G = (V, E)$.
- Parameter:* k .
- Question:* Can G be embedded in the plane with at most k edges crossing?

The general version of this problem is NP -complete [239]. If the maximum degree of any vertex in G is 3, the problem can be solved in $O(n^3)$ by Ref. [205] via Refs. [396, 397].

MINIMUM DEGREE GRAPH PARTITION

- Instance:* A graph $G = (V, E)$ and positive integers k and d .
Parameter: k, d .
Question: Can V be partitioned into disjoint subsets $V_1, \dots, V_m$ so that, for $1 \leq i \leq m$, $|V_i| \leq k$ and at most d edges have exactly one endpoint in V_i ?

The problem is polynomial time by the slice via the immersion ordering [249].

POLYMATROID RECOGNITION

- Instance:* A k -polymatroid M .
Parameter: A positive integer k .
Question: Is M hypergraphic?

See Ref. [460].

SHORT CHEAP TOUR

- Instance:* A graph $G = (V, E)$, an edge weighting $w : E \rightarrow \mathbb{Z}$.
Parameter: A positive integer k .
Question: Is there a tour through at least k nodes of G of cost at most S ?

The general version of this problem is *NP*-complete by a reduction from **HAMILTON CIRCUIT**. (See Ref. [229], ND22.) Known to be hard for $W[1]$ if we ask that the tour cost *exactly* S [170].

CHAIN MINOR ORDERING

- Input:* A finite poset Q .
Parameter: A finite poset P .
Question: Is P a chain minor of Q ?

Gustedt [255] (Theorem 7.19) has established a $O(|G|^{|P|})$ algorithm.

SHORT GENERALIZED HEX

- Instance:* A graph $G = (V, E)$ with two distinguished vertices v_1 and v_2 .
Parameter: A positive integer k .
Question: Does player one have a winning strategy of at most k moves in the following game? Player one plays with white pebbles and player two with black ones. Pebbles are placed on nondistinguished vertices alternately by player one, then player two. Player one wins if he can construct a path of white vertices from v_1 to v_2 .

The general version of this problem is *PSPACE*-complete by a reduction from **QBFSAT**. (See Ref. [239], GP1.) Candidate for $AW[*]$ -completeness [4].

JUMP NUMBER

- Instance:* A poset $P = (P, \leq)$.
Parameter: A positive integer k .
Question: Is the jump number of $P \leq k$?

The general version of this problem is *NP*-complete. (See Ref. [378].) By El-Zahar and Schmerl [195], there is an $O(n^{k+1})$ algorithm. The problem is related to the number of “jumps” needed in an assembly line.

POLYNOMIAL PRODUCT IDENTITY

- Instance:* Two sets of k multivariate polynomials p_i and q_i for $i = 1, \dots, k$.
Parameter: k .
Question: Does the following identity hold

$$\prod_{i=1}^k p_i = \prod_{i=1}^k q_i?$$

By multiplying out, this can obviously be solved in time $O(n^k)$. The significance of the problem is discussed by Johnson in [282].

SHORTEST VECTOR

- Instance:* A basis $X = \{x_1, x_2, \dots, x_n\} \subset \mathbb{Z}^n$ for a lattice $\mathbb{L}$, and an integer $k > 0$.
Parameter: k
Question: Is there a non-zero vector $x \in \mathbb{L}$, such that $\|x\|^2 \leq k$?

Not known to be *NP*-complete. The complexity of this problem is quite important and has been open since 1980 (van Emde Boas [196]). There has been a considerable amount of work devoted to the proof of van Emde Boas’ conjecture that the problem is *NP*-complete—see Arora, Babai, Stern, and Sweedyk [39] and Vardy [458] for a discussion. We conjecture that the problem is $W[1]$ hard.

EVEN SET

- Instance:* A red/blue graph $G = (\mathcal{R}, \mathcal{B}, E)$ and an integer $k > 0$.
Parameter: k
Question: Is there a set of at most k vertices $R \subseteq \mathcal{R}$? Is there a set of k vertices $R \subseteq \mathcal{R}$ each member of $\mathcal{B}$ having an even number of neighbours in R ?

Equivalent to the **MINIMUM DISTANCE** problem for linear codes, and known to be *NP*-complete. See Downey, Fellows, Vardy and Whittle [182].

B.3 Connections Between Classical and Parameterized Complexity

There are not many connections presently between parameterized and classical complexity. It seems that much stronger connections might be possible. Consider, for example, the following problem.

RESTRICTED VALENCE ISOMORPHISM

- Instance:* Two graphs $G = (V, E)$ and $H = (V', E')$ and an integer k .
Parameter: A positive integer k .
Question: Are G and H isomorphic graphs such that the valencies of the vertices of both G and H are bounded by k ?

Luks [339] has given an $O(n^{f(k)})$ algorithm-parameterized version. If this problem is $W[1]$ -hard, then **GRAPH ISOMORPHISM** is not in P unless $FPT = W[1]$.

This is one concrete example of a way in which the study of a parameterized form of a problem may shed light on classical complexity issues. (We remark that we have already seen this phenomenon in the **PRECEDENCE CONSTRAINED k -PROCESSOR SCHEDULING** problem.)

There has been considerable interest among practitioners and algorithm experimentalists in the phenomena of complexity criticality and phase transitions. One of the now classic results in this area is that **3SAT** seems to “suddenly” become hard at a certain value of the ratio between the number of clauses and the number of variables (e.g., Ref. [353]). There may well be connections that can be made between criticality issues and parameterized complexity by way of the *problem kernel* idea of Chapter 3. For example, there is a polynomial-time parametric reduction of an arbitrary instance (G, k) of **VERTEX COVER** to an instance (G', k') , where $|G'| = k^2$. Assuming that this is a best possible “kernelization” of the problem, this suggests looking for critical complexity behavior for **VERTEX COVER** when k is around $\sqrt{n}$.

B.4 Classification Gaps

It has recently been reported that 9 out of 10 classically trained complexity theorists regard the $W[t]$ -hierarchy as the *Weirdness Hierarchy* that has something to do with bizarre classes of circuits. How shall we respond?

We basically have two answers. Our first defense is that as the precise degree of the miniaturized Halting Problem and foundationally analogous to NP , $W[1]$ is not weird. As for the rest—our second defense is that it does not really matter about the rest of the hierarchy and how it is defined and what the definitions might mean *a priori*—we are naturally interested in any case in comparing the difficulty of

concrete problems and classifying them in terms of their natural degree structure, whatever this turns out to be. It is remarkable that there seem to be so few important natural degrees of parameterized complexity.

Nevertheless, once we have embarked on the inherently appealing project of classifying problems, we find that there are many annoying gaps between hardness and membership results. For example, there is the **PERFECT CODE** problem that is hard for $W[1]$ and a member of $W[2]$. It is very annoying not to know exactly where to put it. We conjecture that it either represents a natural degree intermediate between $W[1]$ and $W[2]$, or is complete for $W[2]$. In any case, it has resisted strenuous efforts to achieve a precise classification. Another example is **SHORT PERMUTATION GROUP FACTORIZATION**. This is as hard as **PERFECT CODE**, but the best “upper bound” presently known is that it belongs to $W[P]$. A perusal of the Problem Compendium will reveal many open problems of this sort. We suspect that finding better upper bound techniques is closely related to basic issues in the design of *FPT* algorithms.

B.5 Structural Issues and Analogs of Classical Results

One of the deep results of Part III is a *partial* analog of Ladner’s theorem [321] for strongly uniform parameterized reducibility. Obtaining a full analog—if it is even true—remains a significant and very challenging open problem.

It is important for the reader interested in structural issues to realize that proving parameterized analogs of classical theorems is not at all routine, and that few analogs are currently known. Some would have very interesting concrete consequences. For example, if there is a parameterized analog of Toda’s theorem, then the problem **UNIQUE CLIQUE** of determining whether a graph has a unique k -clique (for parameter k) is as hard as any problem at any level of the $W[t]$ -hierarchy [181]. Currently, however, we lack even the basic probability amplification results that such an analog would presumably require.

B.6 The *FPT* Toolkit and Further Business with the Devil

It should be emphasized that parameterized complexity is concerned with a fundamental shift in perspective about how to approach the bookkeeping by which we set our goals and measure our efforts and difficulties in the design of algorithms. It is about heeding the restless drums of the practitioners that have been beating for some time in the jungles of a world full of intractabilities. It’s about replacing the rigid and largely inaccessible “asymptotic polynomial” bookkeeping with a program in which we:

Pay constant and fundamental attention to the various kinds of parameters that govern natural problem input distributions, and be aggressive about defining and identifying them.

- Exploit these parameters in dealing with the devil of problem intractability.
- Gradually, with attention to our finite universe, extend the range of viable practicality, klam by klam.

What we have seen is that there is a wide class of general methodologies for the construction of parameterized algorithms. Several problems have seen very significant improvements on the running time of such problems. Time is ripe for the development of industrial strength methodologies which will give practical algorithms for wide classes of problems. This is strongly related to the generation of heuristics for combinatorial problems. We refer to our recent survey paper (Downey, Fellows and Stege [176]) for detailed discussion of these and other issues.

References

- [1] J. Abello, M. Fellows, and J. Stillwell, “On the complexity and combinatorics of covering finite complexes,” *Australasian J. Comb.*, Vol. 4 (1991) 103–112.
- [2] K. A. Abrahamson, private communication.
- [3] K. A. Abrahamson, R. G. Downey, and M. R. Fellows, “Fixed-parameter intractability II,” in *STACS 93*, Lecture Notes in Computer Science, Springer-Verlag (1993) 374–385.
- [4] K. A. Abrahamson, R. G. Downey, and M. R. Fellows, “Fixed-parameter tractability and completeness IV: On completeness for $W[P]$ and $PSPACE$ analogs,” *Ann. Pure Appl. Logic*, Vol. 73, (1995) 235–276.
- [5] K. A. Abrahamson, J. Ellis, M. R. Fellows, and M. Mata, “On the complexity of fixed-parameter problems,” In *30th Annual Symposium on Foundations of Computer Science*, IEEE Computer Society Press (1989) 210–215.
- [6] K. A. Abrahamson and M. R. Fellows, “Cutset regularity beats well-quasi-ordering for bounded treewidth,” preprint, 1989.
- [7] K. A. Abrahamson and M. R. Fellows, “Finite automata, bounded treewidth, and well-quasiordering,” in *Graph Structure Theory* (ed. N. Robertson and P. Seymour) Contemporary Mathematics Vol. 147, American Mathematical Society (1993) 539–564.
- [8] K. A. Abrahamson, M. R. Fellows, M. A. Langston, and B. Moret, “Constructive complexity,” *Discrete Appl. Math.* Vol. 34 (1991) 3–16.
- [9] K. Abrahamson, M. Fellows, and C. Wilson, “Parallel self-reducibility,” in *Fourth International Conference on Computing and Information*, Toronto, Ontario, Canada, May, 1992.

- [10] A. Adachi, S. Iwata, and T. Kasai, "Classes of pebble games and complete problems," *SIAM J. Comput.*, Vol. 8 (1979) 576–586.
- [11] A. Adachi, S. Iwata, and T. Kasai, "Some combinatorial game problems require $\Omega(n^k)$ time," *J. Assoc. Comput. Mach.*, Vol. 31(2) (1984) 361–376.
- [12] R. Agarwala and D. Fernandez-Baca, "A polynomial time algorithm for the perfect phylogeny problem when the number of character states is fixed," *SIAM J. Comput.*, Vol. 23 (1994) 1216–1224.
- [13] R. Agarwala and D. Fernandez-Baca, "Simple algorithms for perfect phylogeny and triangulating coloured graphs," *Int. J. Found. Computer Sci.*, Vol. 7(1) (1996) 11–21.
- [14] G. B. Agnew, R. C. Mullin, I. M. Onyszchuk, and S. A. Vanstone, "An implementation for a fast public-key cryptosystem," *J. Cryptolo.*, Vol. 3 (1991) 63–79.
- [15] R. Ahuja and J. Orlin, "A fast and simple algorithm for the maximum flow problem," *Oper. Res.*, Vol. 37(5) (1989) 748–759.
- [16] M. Ajtai, "The shortest vector problem in L_2 is NP-hard for randomized reductions," personal communication, May 1997.
- [17] N. Alon, R. Yuster, and U. Zwick, "Color-coding: A new method of finding simple paths, cycles and other small subgraphs within large graphs," in *Proceedings of the 26th Annual Symposium on the Theory of Computing*, ACM Press (1994) 326–335.
- [18] S. Alstrup, P. W. Lauridsen, and M. Thorup "Generalized dominators for structured programs." *Proc. 3rd Static Analysis Symp.*, Lecture Notes in Computer Science Vol. 1145, Springer-Verlag (1996) 42–51.
- [19] C. Àlvarez, J. Díaz, and J. Torán, "Complexity classes with complete problems between P and NP-complete," in *Foundations of Computation Theory*, Lecture Notes in Computer Science Vol. 380, Springer-Verlag (1989) 13–24.
- [20] K. Ambos-Spies, "On the structure of the polynomial degrees of recursive sets," Habilitationsschrift, Universität Dortmund, 1984.
- [21] K. Ambos-Spies, "Antimotivic recursively enumerable sets," *Z. Math. Logik Grundl. Math.*, Vol. 31 (1985) 461–477.
- [22] K. Ambos-Spies and A. Nies, "The theory of the polynomial time many-one degrees is undecidable," to appear.
- [23] K. Ambos-Spies, A. Nies, and R. A. Shore, "The theory of the recursively enumerable weak truth table degrees is undecidable," *Journal of Symbolic Logic*, Vol. 57 (1992) 864–874.
- [24] R. Anderson and E. Mayr, "Approximating P -complete problems," Stanford University, 1986.
- [25] D. Angliun, "Finding patterns common to a set of strings," *J. Comput. Syst. Sci.* 21 (1980) 46–62.
- [26] D. Angluin, "Local and global properties in networks of processors," in *Proc. 12th ACM Symposium on Theory of Computing*, 1980, 82–93.

- [27] D. Angluin, "Learning a k -term DNF formulae using queries and counterexamples," Tech. Report, YALE/DCS/RR-559, Yale University, 1987.
- [28] D. Angluin, "Computational learning theory: survey and selected bibliography," in *Proc. 24th STOC*, 1992, 351–369.
- [29] K. Aoki, J. Shinoda, and T. Tsuda, "On Π_2 Theories of hp-T degrees of low sets," *Theoret. Computer Sci.*, Vol. 123(2) (1994) 315–327.
- [30] D. Archdeakon, *A Kuwatowski Theorem for the Projective Plane*, Ph. D. Thesis, Ohio State University, 1980.
- [31] S. Arnborg, "Efficient algorithms for combinatorial problems on graphs with bounded decomposability—a survey," *BIT*, 25 (1985) 2–23.
- [32] S. Arnborg, D. G. Corneil, and A. Proskurowski, "Complexity of finding embeddings in a k -tree," *SIAM J. Algebr. Discrete Methods*, Vol. 8 (1987) 277–284.
- [33] S. Arnborg, B. Courcelle, A. Proskurowski, and D. Seese, "An algebraic theory of graph reduction," Technical Report 90-02, Laboratoire Bordelais de Recherche Informatique, University Bordeaux I, January 1990.
- [34] S. Arnborg, J. Lagergren, and D. Seese, "Problems easy for tree-decomposable graphs (extended abstract)," in *Proc. 15th Coll. Automata, Languages and Programming* (ed. T. Lepisto and A. Salomaa), Lecture Notes in Computer Science Vol. 317, Springer-Verlag (1988) 38–51.
- [35] S. Arnborg, J. Lagergren, and D. Seese, "Easy problems for tree-decomposable graphs," *J. Algorithms*, Vol. 12 (1991) 308–340.
- [36] E. Arkin, "Complexity of cycle and path problems in graphs," Ph.D. dissertation, Stanford University, 1986.
- [37] S. Arora, "Reductions, codes, PCPs and inapproximability," 1995.
- [38] S. Arora. Polynomial time approximation schemes for Euclidean TSP and other geometric problems. *Proceedings of the 37th IEEE Symposium on the Foundations of Computer Science*, 1996.
- [39] S. Arora, L. Babai, J. Stern, and Z. Sweedyk, "The hardness of approximate optima in lattices, codes, and systems of linear equations," in *Proc. 34th Annual Symp. Found. Computer Science*, Palo Alto, CA (1993) 724–733.
- [40] S. Arora and C. Lund, "Hardness of approximation," in *Approximation Algorithms for NP-Hard Problems* (ed. D. Hochbaum), PWS Publ. Co. (1997) 399–446.
- [41] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy, "Proof verification and intractability of approximation problems," *FOCS*, 1992.
- [42] S. Ausiello, A. D'Atri, and M. Protasi, "Structure preserving reductions among convex optimization problems," *J. Computer Syst. Sci.*, Vol. 21 (1980) 136–153.
- [43] R. A. Baeza-Yates, "Searching subsequences," *Theoret. Computer Sci.*, 78 (1991) 363–376.

- [44] B. Baker. Approximation algorithms for NP-complete problems on planar graphs. *J.A.C.M.* 41 (1994), 153–180.
- [45] R. Balasubramanian, R. Downey, M. Fellows, and V. Raman, unpublished.
- [46] R. Balasubramanian, M. Fellows, and V. Raman, “An improved fixed parameter algorithm for vertex cover,” *Inform. Process. Lett.*, to appear.
- [47] J. Balcazar, J. Diaz, and J. Gabarro, *Structural Complexity*, Volumes 1 and 2, Springer Verlag (1987, 1989).
- [48] T. Baker, J. Gill, and R. Solovay, “Relativizations of the $P = NP?$ question,” *SIAM J. Comp.*, Vol. 4 (1975) 431–442.
- [49] C. A. Barefoot, L. H. Clark, J. Douthett, R. C. Entringer, and M. R. Fellows, “Cycles of Length 0 Modulo 3 in Graphs,” *Ann. Discrete Math.* (1991) 87–101.
- [50] A. Barg, “Some new NP-complete coding problems,” *Probl. Peredachi Informatsii*, Vol. 30 (1994) 23–28 (in Russian).
- [51] Y. Bar-Hillel, M. Perles, and E. Shamir, “On formal properties of simple phase structure grammars,” *Z. Phonetik. Sprachwiss. Kommunikationsforsch.*, Vol. 14 (1961) 143–172.
- [52] B. Bauderon and B. Courcelle, “Graph expressions and rewriting systems,” *Math. Syst. Theory*, Vol. 20 (1987) 83–127.
- [53] C. Bazgan, “Schémas d’approximation et complexité paramétrée,” manuscript, 44pp.
- [54] C. Bazgan. Schémas d’approximation et complexité paramétrée. Thesis, INRIA, Orsay, France, 1995.
- [55] C. Bennett and J. Gill, “Relative to a random oracle A , $P^A \neq NP^A \neq co - NP^A$ with probability 1,” *SIAM J. Comput.*, Vol. 10 (1981) 96–113.
- [56] E. R. Berlekamp, R. J. McEliece, and H. C. A. van Tilborg, “On the inherent intractability of certain coding problems,” *IEEE Trans. Inform. Theory*, Vol. 24 (1978) 384–386.
- [57] M. W. Bern, E. L. Lawler, and A. L. Wong, “Linear time computation of optimal subgraphs of decomposable graphs,” *J. Algorithms*, Vol. 8 (1987) 216–235.
- [58] L. Berman and J. Hartmanis, “On isomorphisms and density of NP and other complete sets,” *SIAM J. Comput.*, Vol. 6 (1977) 305–321.
- [59] D. Bienstock and M. A. Langston, “Algorithmic implications of the graph minor theorem,” in *Handbook of Operations Research and Management Science: Network Models* (eds. M. O. Ball, T. L. Magnanti, C. L. Monma and G. L. Nemhauser), North-Holland (1995) 481–502.
- [60] D. Bienstock and C. Monma, “On the complexity of covering vertices by faces in planar graphs,” *SIAM J. Comput.*, Vol. 17 (1988) 53–76.
- [61] D. Bienstock, N. Robertson, P. Seymour, and R. Thomas, “Quickly excluding a forest,” *J. Combinatorial Theory B*, Vol. 52 (1991) 274–283.

- [62] M. Blum, "A machine-independent theory of the complexity of recursive functions," *J. Assoc. Comput. Mach.*, Vol. 14 (1967) 322–336.
- [63] H. L. Bodlaender, "A linear time algorithm for finding tree-decompositions of small treewidth," *final version of [71]*, to appear.
- [64] H. L. Bodlaender, "On linear time minor tests and depth-first search," in *Proc. 1st Workshop on Algorithms and Data Structures* (ed. F. Dehne et al.), Lecture Notes in Computer Science Vol. 382, Springer-Verlag (1989) 577–590.
- [65] H. L. Bodlaender, "Classes of graphs with bounded tree-width," Tech. Report RUU-CS-86-22, Dept. of Computer Science, University of Utrecht, Utrecht, 1986.
- [66] H. L. Bodlaender, "Dynamic programming on graphs with bounded tree-width," in *Proc. 15th Int. Coll. Automata, Languages and Programming*, (eds. T. Lepistö and A. Salomaa), Lecture Notes in Computer Science Vol. 317, Springer-Verlag (1988) 103–118.
- [67] H. L. Bodlaender, "Some classes of graphs with bounded treewidth," *Bull. EATCS*, 36 (1988) 116–126.
- [68] H. L. Bodlaender, "On disjoint cycles," Technical Report RUU-CS-90-29, Dept. of Computer Science, Utrecht University, The Netherlands, August 1990.
- [69] H. L. Bodlaender, private communication, 1994.
- [70] H. L. Bodlaender, *A Tourist's Guide Through Treewidth*, Technical Report TR RUU-CS-92-12, Computer Science Department, Utrecht University, The Netherlands, March 1992.
- [71] H. L. Bodlaender, "A linear time algorithm for finding tree-decompositions of small treewidth," *SIAM J. Comput.*, Vol. 25 (1996) 1305–1317.
- [72] H. Bodlaender, P. Evans, and M. Fellows, "Finite string complexity of annotations of strings and trees," in *Proceedings Combinatorial Pattern Matching, '96*, Lecture Notes in Computer Science Vol. 1075, Springer-Verlag (1996) 384–391.
- [73] H. L. Bodlaender, R. G. Downey, M. R. Fellows, M. T. Hallett, and H. T. Wareham, "Parameterized complexity analysis in computational biology," in *IEEE Workshop on Shape and Pattern Matching in Computational Biology, held in conjunction with 1994 IEEE Conference on Computer Vision and Pattern Recognition* June 1994.
- [74] H. L. Bodlaender, R. G. Downey, M. R. Fellows, M. T. Hallett, and H. T. Wareham, "Parameterized complexity analysis in computational biology," *CABIOS (Computer Applications in the Biosciences)*, Vol. 11(1) (1994) 49–57.
- [75] H. L. Bodlaender, R. G. Downey, M. R. Fellows, and H. T. Wareham, "The parameterized complexity of sequence alignment and consensus (extended abstract)," in *Proceedings of the Fourth Conference on Combinatorial Pattern Matching (CPM '94)*.
- [76] H. L. Bodlaender, R. G. Downey, M. R. Fellows, and H. T. Wareham, "The parameterized complexity of sequence alignment and consensus," *Theoret. Computer Sci.*, Vol. 147 (1994) 31–54.

- [77] H. L. Bodlaender and J. Engelfriet, "Domino treewidth," Technical Report UU-CS-1994-11, Department of Computer Science, Utrecht University, Utrecht, the Netherlands.
- [78] H. Bodlaender and M. Fellows, On the complexity of k -processor scheduling. *Oper. Res. Lett.*, Vol. 18 (1995) 93–98.
- [79] H. L. Bodlaender, M. R. Fellows, and M. T. Hallett, "Beyond NP -completeness for problems of bounded width: hardness for the W -hierarchy (extended abstract)," in *STOC* 26, 1994, 449–458.
- [80] H. L. Bodlaender, M. R. Fellows, and M. T. Hallett, "The parameterized intractability of bandwidth and related problems," unpublished.
- [81] H. L. Bodlaender, M. R. Fellows, and T. Warnow, "Two strikes against perfect phylogeny," in *Proc. 19th International Colloquium on Automata, Languages and Programming (ICALP '92)*, Lecture Notes in Computer Science Vol. 623, Springer-Verlag (1992) 273–283.
- [82] H. L. Bodlaender, M. R. Fellows, M. T. Hallett, H. T. Wareham, and T. Warnow, "The hardness of thin colored graphs," *SIAM J. Comp.*, to appear.
- [83] H. L. Bodlaender and B. de Fluiter, "Intervalizing k -colored graphs," in *Proc. 22nd International Colloquium on Automata, Languages and Programming (ICALP '95)*, Lecture Notes in Computer Science Vol. 944, Springer-Verlag (1994) 87–98.
- [84] H. L. Bodlaender and T. Kloks, "Better algorithms for the pathwidth and treewidth of graphs," in *Proceedings of the 18th International Colloquium on Automata, Languages and Programming, (ICALP)*, Lecture Notes in Computer Science Vol. 510, Springer-Verlag (1991) 544–555.
- [85] H. L. Bodlaender and T. Kloks, "A simple linear time algorithm for triangulating three-coloured graphs," in *Proceedings 9th Annual Symposium on the Theory of Computing*, 1992, 415–423.
- [86] H. L. Bodlaender and T. Kloks, "Efficient and constructive algorithms for pathwidth and treewidth of graphs," *J. Algorithms*, Vol. 21 (1996) 358–402.
- [87] H. L. Bodlaender, J. van Leeuwen, R. Tan, and D. M. Thilikos, "On interval routing schemes and treewidth," Technical Report UU-CS-1996-41, Department of Computer Science, Universiteit Utrecht, 1996.
- [88] H. L. Bodlaender and D. Kratsch, private communication, 1994.
- [89] R. Bonnet and M. Pouzet, "Linear extensions of ordered sets," in *Ordered Sets* (ed. I. Rival), D. Reidel (1981) 125–170.
- [90] Bollobas, *Extremal Graph Theory*.
- [91] M. Bonnet, M. S. Steel, T. Warnow and S. Yooseph, "Better methods for solving parsimony and compatibility." Manuscript, 1998, to appear.
- [92] K. S. Booth, "Isomorphism testing in graphs, semigroups and finite automata are polynomially equivalent problems," *SIAM J. Comput.*, Vol. 7 (1978) 273–279.

- [93] R. Boppana and M. Sipser, "The complexity of finite functions," in *Handbook of Theoretical Computer Science, Vol. A* (ed. J. van Leeuwen) (1990) 757–804.
- [94] R. B. Borie, R. G. Parker and C. A. Tovey, "Automatic generation of linear-time algorithms from predicate calculus descriptions of problems on recursively constructed graph families," *Algorithmica*, Vol. 7 (1992) 555–582.
- [95] G. Brightwell and D. Winkler, "Counting linear extensions," *Order*, Vol. 8 (1992) 225–242.
- [96] G. Brightwell and D. Winkler, "Counting linear extensions is $\#P$ -complete," in *Proc. 23rd STOC*, 1991, 175–181.
- [97] D. J. Brown, M. R. Fellows, and M. A. Langston, "Polynomial time of self-reducibility: theoretical motivations and practical results," *Int. J. Computer Math.*, Vol. 31 (1989) 1–9.
- [98] J. Bruck and M. Naor, "The hardness of decoding linear codes with preprocessing," *IEEE Trans. Inform. Theory*, Vol. 36 (1990) 381–385.
- [99] J. R. Büchi, "Weak second order arithmetic and finite automata," *Zeitschr. Math. Logik Grundl. Math.*, Vol. 6 (1960) 66–92.
- [100] S. Buss, private communication, 1989.
- [101] J. F. Buss and J. Goldsmith, "Nondeterminism within P ," *SIAM J. Comput.*, Vol. 22 (1993) 560–572.
- [102] T. Bylander, "The computational complexity of propositional STRIPS planning," *Artificial Intelligence*, Vol. 69 (1994) 161–204.
- [103] Leizhen Cai, "Fixed-parameter tractability of graph modification problems for hereditary properties," *Information Processing Letters*, Vol. 58(4) (1996) 171–176.
- [104] Liming Cai, "Fixed parameter tractability and approximation problems," Project Report, June 1992.
- [105] Liming Cai and J. Chen, "On the amount of nondeterminism and the power of verifying," in *MFCS'93*, Lecture Notes in Computer Science Vol. 711, Springer-Verlag (1993) 311–320.
- [106] Liming Cai and J. Chen, "On Input Read Modes of Alternating Turing Machines," TR 93-046, Department of Computer Science, Texas A&M University, 1993.
- [107] Liming Cai and J. Chen, "Fixed-parameter tractability and approximability of NP -hard optimization problems," in *Proceedings of the 2nd Israel Symposium on Theory of Computing and Systems*, 1993, 118–126.
- [108] Liming Cai and J. Chen, "On fixed-parameter tractability and approximability of NP optimization problems," *J.C.S.S.*, to appear.
- [109] Liming Cai, J. Chen, R. G. Downey, and M. R. Fellows, "The parameterized complexity of short computation and factorization," in *Proceedings of the Sacks Conference*, 1993, in Archive for Math Logic, 1997.

- [110] Liming Cai, J. Chen, R. G. Downey, and M. R. Fellows, "Advice classes of parameterized tractability," *Annals Pure and Applied Logic*, Vol. 84 (1997) 119–138.
- [111] Liming Cai, J. Chen, R. G. Downey, and M. R. Fellows, "On the structure of parameterized problems in NP ," in *STACS 94, Lecture Notes in Computer Science*, Springer-Verlag (1994) 509–520.
- [112] Liming Cai, J. Chen, R. G. Downey, and M. R. Fellows, "On the parameterized complexity of short computation and factorization," *Arch. Math. Logic*, to appear.
- [113] J. F. Canny, *The Complexity of Robot Motion Planning*, The MIT Press (1987).
- [114] G. Cantor, "Beiträge zur begründung transfiniten mengenlehre," *Math. Ann.*, Vol. 96 (1895) 481–512.
- [115] K. Cattell and M. J. Dinneen, "A characterization of graphs with vertex cover number up to five," in *Proceedings ORDAL'94, Lecture Notes in Computer Science* Vol. 831, Springer-Verlag (1994) 86–99.
- [116] K. Cattell, M. J. Dinneen, and M. R. Fellows, "An implementation of a graph minors obstruction set computation: Feedback vertex set," Technical Report, Department of Computer Science, University of Victoria, July 1994.
- [117] K. Cattell, M. Dinneen, and M. R. Fellows, "A simple linear-time algorithm for finding path-decompositions of small width," *Inform. Process. Lett.*, Vol. 57 (1996) 197–203.
- [118] R. Coles, R. Downey, A. Sorbi, "Undecidability results for parametric reductions," in preparation.
- [119] M. Cesati, private communication, 1994.
- [120] M. Cesati and L. Trevisan. On the efficiency of polynomial-time approximation schemes. To appear in *Information Processing Letters*, 1998.
- [121] M. Cesati and M. Di Ianni, "Parameterized parallel complexity," *Electronic Colloquium on Computational Complexity*, Reports Series 1997—ECCC TR97-06, <http://www.eccc.uni-trier.de/eccc/>.
- [122] M. Cesati and M. Di Ianni, "Computational models for parameterized complexity," *Mathematical Logic Quarterly*, Vol. 43 (1997) 179–203.
- [123] M. Cesati and T. Wareham, "Parameterized complexity analysis in robot motion planning," in *Proceedings of the 25th IEEE International Conference on Systems, Man, and Cybernetics: Volume 1*, IEEE Press (1995) 880–885.
- [124] R. Chang, B. Chor, O. Goldreich, J. Hartmanis, J. Hastad, D. Ranjan, and P. Rohatgi, "The random oracle hypothesis is false," *J. Comput. Syst. Sci.*, Vol. 49 (1994) 24–39.
- [125] C. Chegireddy and H. Hamacher, "Algorithms for finding the k best perfect matchings," *J. Comb. Theory B.* (1987) 155–165.
- [126] P. Cholak and R. G. Downey, "Undecidability and definability for parameterized polynomial time m -reducibilities," in *Logical Methods* (ed. J. Crossley, J. Remmel, R. Shore, and M. Sweedler) Birkhauser (1994) 194–221.

- [127] P. Cohen, "The independence of the continuum hypothesis I," *Proc. Nat. Acad. Sci. U.S.A.*, Vol. 50 (1963) 1143–1148.
- [128] R. Coles, "Pathwidth and Thomas' decomposition," Preprint, Victoria University, 1996.
- [129] A. Condon, "The complexity of the max word problem and the power of one way interactive proof systems," *STACS* (1991).
- [130] J. H. Conway and C. McA. Gordon, "Knots and links in spatial graphs," *J. Graph Theory*, Vol. 7 (1983) 445–453.
- [131] S. Cook, "The complexity of theorem proving procedures," *Proceedings 3rd ACM STOC*, 1971, 151–158.
- [132] D. Coppersmith and S. Winograd, "Matrix multiplication via arithmetical progressions," *J. Symbolic Comput.*, Vol. 9 (1990) 251–280.
- [133] B. Courcelle, in *Springer Lecture Notes in Computer Science* Vol. 291, Springer-Verlag (1987).
- [134] B. Courcelle, "Recognizability and second-order definability for sets of finite graphs," Technical Report I-8634, Universite de Bordeaux, 1987.
- [135] B. Courcelle, "Graph rewriting: an algebraic and logic approach," in *Handbook of Theoretical Computer Science* (ed. J. van Leeuwen), North-Holland (1990), Vol. B, Chap. 5.
- [136] B. Courcelle, "The monadic second order theory of graphs I: Recognisable sets of finite graphs," *Inform. Comput.*, Vol. 85 (1990) 12–75.
- [137] B. Courcelle, "The monadic second order theory of graphs V: On closing the gap between definability and recognizability," *Theoret. Computer Sc.*, Vol. 80 (1991) 153–202.
- [138] B. Courcelle, "The monadic second order theory of graphs III: Tree decompositions, minors, and complexity issues," *Theoret. Inform. Applic.*, Vol. 26(2) (1992) 257–286.
- [139] B. Courcelle, "Recognizable sets of graphs: equivalent definitions and closure properties," Technical Report 92-06, Universite de Bordeaux 1, 1992.
- [140] B. Courcelle, "The monadic second order theory of graphs VII: Graphs as relational structures," *Theoret. Computer Sci.*, Vol. 101 (1992) 3–33.
- [141] B. Courcelle, "The monadic second order theory of graphs VI: On several representations of graphs by relational structures," *Discrete Appl. Math.*, Vol. 54 (1994) 117–149.
- [142] B. Courcelle, "The monadic second order theory of graphs VIII: Orientations," *Ann. Pure Appl. Logic*, Vol. 72 (1995) 103–143.
- [143] B. Courcelle, "The monadic second order theory of graphs X: Linear orderings," *Theoret. Computer Sci.*, Vol 160 (1996) 87–144.
- [144] B. Courcelle, "Basic notions of universal algebra for language theory and graph grammars," *Theoret. Computer Sci.*, 163(1-2) (1996) 1–54.

- [145] B. Courcelle and P. Franchi-Zannettacci, "Attribute grammars and recursive program schemes," *Theoret. Computer Sci.*, Vol. 17 (1982) 163-191, 235-257.
- [146] B. Courcelle and J. Lagergren, "Equivalent definitions of recognizability for sets of graphs of bounded tree-width," *Math. Struct. Computer Sci.*, 6(2) (1996) 141-165.
- [147] E. Dahlhaus, D. Johnson, C. Papadimitriou, P. Seymour, and M. Yannakakis, "The complexity of multiway cuts," *STOC* (1992) 241-251.
- [148] M. Davis, "Unsolvability problems," in *The Handbook of Mathematical Logic*, (ed. J. Barwise), Elsevier (1977) 580.
- [149] W. H. E. Day and F. R. McMorris, "Discovering consensus molecular sequences," in *Information and Classification—Concepts, Methods, and Applications* (ed. O. Opitz, B. Lausen, and R. Klar), Springer-Verlag (1993) 393-402.
- [150] W. H. E. Day and F. R. McMorris, "The computation of consensus patterns in DNA sequences," *Math. Computer Model.*, Vol. 17 (1993) 49-52.
- [151] W. Day and D. Sankoff, "Computational complexity of inferring phylogenies by compatibility," *Syst. Zool.*, Vol. 35(2) (1986) 224-229.
- [152] T. Dean, J. Kirman, and S. H. Lin, "Theory and practice in planning." Manuscript of invited address at AAAI 96.
- [153] I. J. Dejter and M. R. Fellows, "Improving the diameter of a planar graph," unpublished.
- [154] L. Dennenberg, Y. Gurevich, and S. Shelah, "Definability by constant-depth polynomial-size circuits," *Inform. Control*, Vol. 70 (1986) 216-240.
- [155] J. Díaz and J. Torán, "Classes of bounded nondeterminism," *Math. Syst. Theory*, Vol. 23 (1990) 21-32.
- [156] R. Diestel, "A short proof of the path-width theorem," to appear.
- [157] G. Ding, B. Oporowski, D. P. Sanders, and D. Vertigan, "Partitioning into graphs of bounded tree-width," in preparation.
- [158] M. Dinneen, "Too many minor order obstructions (for parameterized lower ideals)," to appear.
- [159] G. Dirac and S. Schuster, "A theorem of Kuratowski," *Nederl. Akad. Wetensch. Proc. Ser. A*, Vol. 57 (1954) 343-348.
- [160] J. Doner, "Tree Automata and some of their applications," *J. Comput. Syst. Sci.*, Vol. 4 (1970) 406-451.
- [161] R. G. Downey, "Nondiamond theorems for polynomial time reducibility," *J. Comput. Syst. Sci.* Vol. 45 (1992) 385-395.
- [162] R. Downey, B. Courcelle, and M. Fellows, "A note on the computability of graph minor obstruction sets for monadic second order ideas," to appear *Journal of Universal Computer Science*,

- [163] R. G. Downey, P. Evans, and M. R. Fellows, "Parameterized learning complexity," in *Proceedings of the 6th Annual Conference on Learning Theory, COLT'93*, ACM Press (1993) 51–57.
- [164] R. Downey and M. Fellows, Index sets and parametric reductions, to appear *Archive for Mathematical Logic*.
- [165] R. G. Downey and M. R. Fellows, "Fixed-parameter tractability and completeness," *Congressus Numerantium*, Vol. 87 (1992) 161–187.
- [166] R. G. Downey and M. R. Fellows, "Fixed-parameter tractability and completeness I: Basic results," *SIAM J. Comput.*, Vol. 24 (1995) 873–921.
- [167] R. G. Downey and M. R. Fellows, "Fixed parameter intractability (Extended Abstract)," in *Proc. Structure in Complexity Theory*, IEEE (1992) 36–50.
- [168] R. G. Downey and M. R. Fellows, "Fixed-parameter tractability and completeness III: Some structural aspects of the W -hierarchy," in *Complexity Theory* (ed. K. Ambos-Spies, S. Homer, and U. Schöningh), Cambridge University Press (1993) 166–191.
- [169] R. G. Downey and M. R. Fellows, "Parameterized computational feasibility," in *Proceedings of Feasible Mathematics II* (ed. P. Clote and J. B. Remmel), Birkhäuser (1995) 219–244.
- [170] R. G. Downey and M. R. Fellows, "Fixed-parameter tractability and completeness II: On completeness for $W[1]$," *Theoret. Computer Sci.*, Vol. 141 (1995) 109–131.
- [171] R. G. Downey and M. R. Fellows, "Fixed-parameter tractability and intractability—a survey," unpublished.
- [172] R. G. Downey and M. R. Fellows, "Threshold dominating sets and an improved characterization of $W[2]$," *Theoret. Computer Sci.*, Vol. 189 (1997), to appear.
- [173] R. G. Downey, M. R. Fellows, and V. Raman, "The complexity of irredundant sets parameterized by size," unpublished.
- [174] R. G. Downey, M. R. Fellows, and K. Regan, "Parameterized circuit complexity and the W -hierarchy," *Theoret. Computer Sci.*, Vol. 189 (1997), to appear.
- [175] R. G. Downey, M. R. Fellows, and K. Regan, "Descriptive complexity and the W -hierarchy," *AMS-DIMACS Proceedings Series for the 1996 DIMACS Workshop on "Feasible Arithmetics and Lengths of Proofs"*, (ed. P. Beame and S. Buss), to appear.
- [176] R. G. Downey, M. R. Fellows, and U. Stege, "Parameterized complexity analysis: A systematic framework to cope with computational intractability," to appear DIMACS devoted to Combinatorics in the 21st century.
- [177] R. G. Downey, M. R. Fellows, and U. Taylor, "On the parameteric complexity of relational database queries and a sharper characterization of $W[1]$," in *Combinatorics, Complexity and Logic*, Proceedings of DMTCS '96 (ed. (D. Bridges, C. Calude, J. Gibbons, S. Reeves, and I. Witten) Springer-Verlag (1996) 194–213.

- [178] R. G. Downey, M. R. Fellows, B. Kapron, M. T. Hallett, and H. T. Wareham, "Parameterized complexity of some problems in logic and linguistics (Extended Abstract)," in *2nd Workshop on Structural Complexity and Recursion-theoretic Methods in Logic Programming*, 1993. Also in *Logic at St. Petersburg* (ed. A. Nerode and Y. Matiyasevich) Lecture Notes in Computer Science Vol. 813, Springer-Verlag (1994) 89–101.
- [179] R. G. Downey, M. R. Fellows, B. Kapron, M. T. Hallett, and H. T. Wareham, "Parameterized complexity of some problems in logic and linguistics," unpublished.
- [180] R. G. Downey, M. R. Fellows, and N. Koblitiz, "Techniques for exponential parameterized reductions in vertex set problems," unpublished.
- [181] R. G. Downey, M. R. Fellows, and K. Regan, "Parameterized circuit complexity and the W -hierarchy," *Theoret. Computer Sci.*, Vol. 189 (1997), to appear.
- [182] R. G. Downey, M. R. Fellows, A. Vardy, and G. Whittle, "The Parameterized Complexity of Some Fundamental Problems in Coding Theory," to appear, *Siam J. of Comput.*
- [183] R. G. Downey and C. Jockusch, "T-degrees, jump classes and strong reducibilities," *Trans. Am. Math. Soc.*, Vol. 301 (1987) 103–136.
- [184] R. G. Downey and R. A. Shore, "Degree theoretical definitions of the low₂ recursively enumerable sets," *J. Symbolic Logic*, Vol. 60 (1995) 727–756.
- [185] R. G. Downey and L. V. Welch, "Splitting properties of r. e. sets and degrees," *J. Symbolic Logic*, Vol. 51 (1986) 88–109.
- [186] M. Doyle, MSC Thesis, Victoria University, to appear.
- [187] A. Dress and M. Steel, "Conver tree realizations of partitions," *Appl. Math. Lett.*, Vol. 5 (1992) 3–6.
- [188] S. Dreyfus and R. Wagner, "The Steiner problem in graphs," *Networks*, Vol. 1 (1971) 195–207.
- [189] E. Dubrova, private communication, 1994.
- [190] H. Ebbinghaus and J. Flum, *Finite Model Theory*, Springer-Verlag (1995).
- [191] J. Edmonds, "Minimum partition of a matroid into indepent sets," *J. Res. Nat. Bur. Stand. Sect. B.*, Vol. 69 (1965) 67–72.
- [192] J. Edmonds, and R. Karp, "Theoretical improvements in algorithmic efficiency for netowrk flow problems," *J. Assoc. Comput. Mach.* (1972) 248–264.
- [193] A. Ehrenfeucht, D. Haussler, and G. Rozenberg. "On regularity of context-free languages." *Theoretical Computer Science* 27 (1983), 311–332.
- [194] J. Ellis, I. H. Sudborough, and J. Turner, "Graph separation and search number," Report DCS-66-IR, Dept. of Computer Science, University of Victoria, August 1987. Also, *Inform. Computat.*, to appear.

- [195] M. El-Zahar and J. Schmerl, "On the size of jump critical ordered sets," *Order*, Vol. 1 (1984) 3–5.
- [196] P. van Emde Boas, "Another *NP*-complete partition problem and the complexity of computing short vectors in a lattice," Tech. Report 81-04, Dept. of Mathematics, Univ. of Amsterdam (1980).
- [197] P. Erdős and L. Pósa, "On independent circuits contained in a graph," *Canad. J. Math.*, Vol. 17 (1965) 347–352.
- [198] P. Erdős and R. Rado, "A theorem on partial well-ordering of sets of vectors," *J. London. Math. Soc.*, Vol. 34 (1959) 222–224.
- [199] P. Evans, "Adding arcs to the longest common subsequence problem," unpublished.
- [200] S. Even and R. Tarjan, "A combinatorial problem that is complete for polynomial space," *J. Assoc. Comput. Mach.*, Vol. 23 (1976) 710–719.
- [201] R. Fagin, "Generalized first order spectra and polynomial time recognizable languages," in *Complexity of Computations* (ed. (R. Karp) *SIAM-AMS Proc.*, Vol. 7 (1974) 43–73.
- [202] R. Fagin, "Finite model theory—a personal perspective," *Theoretical Computer Science*, 116(1) (1993) 3–31.
- [203] M. R. Fellows, "A Category of graphs for computer science," Ph.D. dissertation, University of California, San Diego, 1985.
- [204] M. Fellows, "On the complexity of vertex set problems," Technical Report, Computer Science Department, University of New Mexico, 1988.
- [205] M. R. Fellows, "The Robertson-Seymour theorems: A survey of applications," *Contemp. Math.*, Vol. 89 (1989) 1–18.
- [206] M. R. Fellows, private communication, 1994.
- [207] M. R. Fellows, M. T. Hallett, and D. Kirby, "The parameterized complexity of shortest common supersequence," unpublished.
- [208] M. R. Fellows, M. Hallett, C. Korostensky, and U. Stege, "Analogues and Duals of the MAST problem for sequences and trees," to appear ESA '98.
- [209] M. R. Fellows, M. T. Hallett, and H. T. Wareham, "DNA physical mapping: three ways difficult," in *ESA '93—European Symposium on Algorithms* (ed. T. Lengauer), Lecture Notes in Computer Science Vol. 726, Springer-Verlag (1993) 157–168.
- [210] M. R. Fellows, F. Hickling, and M. Syslo, "A topological parameterization and hard graph problems," *Congressus Numerantium*, Vol. 59 (1987) 69–78.
- [211] M. R. Fellows and N. Kobitz, "Self-witnessing polynomial-time complexity and prime factorization," in *Proc. 7th Structure in Complexity*, IEEE (1992) 107–111.
- [212] M. Fellows and N. Kobitz, "Fixed-parameter complexity and cryptography," in *Proceedings of the Tenth International Symposium on Applied Algebra, Algebraic Algorithms and Error-Correcting Codes (AAECC'93)*, Lecture Notes in Computer Science Vol. 673, Springer-Verlag (1993) 121–131.

- [213] M. Fellows and N. Koblitz, "Kid krypto," in *Proceedings of Crypto '92*, Lecture Notes in Computer Science Vol. 740, Springer-Verlag (1993) 371–389.
- [214] M. Fellows and N. Koblitz, "Combinatorial cryptosystems galore!" *Contemp. Math.*, Vol. 168 (1994) 51–61.
- [215] M. R. Fellows, J. Kratochvíl, M. Middendorf, and F. Pfeiffer, "The complexity of induced minors and related problems," *Algorithmica*, Vol. 13 (1995) 266–282.
- [216] M. R. Fellows and M. A. Langston, "Nonconstructive tools for proving polynomial time decidability," *J. Assoc. Comput. Mach.*, Vol. 35 (1988) 727–739.
- [217] M. R. Fellows and M. A. Langston, "Layout permutation problems and well-partially ordered sets," in *Proc. 5th MIT Conf. on Advanced Research in VLSI*, 1988, 315–327.
- [218] M. R. Fellows and M. A. Langston, "On search, decision and the efficiency of polynomial-time algorithms," in *Proc. Symp. on Theory of Computing (STOC)*, 1989, 501–512.
- [219] M. R. Fellows and M. A. Langston, "An analog of the Myhill–Nerode theorem and its use in computing finite basis characterizations," in *Proc. Symp. Foundations of Comp. Sci. (FOCS)*, 1989, 520–525.
- [220] M. R. Fellows and M. A. Langston, "Constructivity issues in graph algorithms," in *Conf. on Constructivity Issues in Computer Science*, San Antonio, Texas, 1991.
- [221] M. R. Fellows and M. A. Langston, "On well-partial-ordering theory and its applications to combinatorial problems in VLSI design," *SIAM J. Discrete Math.*, Vol. 5 (1992) 117–126.
- [222] M. R. Fellows and M. A. Langston, "Fast search algorithms for graph layout permutation problems," *Integration*, Vol. 12 (1991) 321–337.
- [223] M. R. Fellows and M. A. Langston, "Nonconstructive advances in polynomial-time complexity," *Inform. Process. Lett.*, Vol. 28 (1987/88) 157–162.
- [224] S. Fenner, L. Fortnow, S. Kurtz, and L. Li, "An oracle builders toolkit," in *Structure in Complexity Theory, 8th Annual Conference*, IEEE (1993) 120–131.
- [225] D. Fernández-Baca and J. Lagergren, "A polynomial-time algorithm for near-perfect phylogeny," *International Colloquium on Automata, Languages and Programming (ICALP '96)*, 1996.
- [226] L. Ford and D. Fulkerson, "Maximum flow through a network," *Canad. Math. J.* (1956) 399–404.
- [227] L. Ford and D. Fulkerson, *Flows in Networks*, Princeton University Press (1962).
- [228] L. Fortnow, "The role of relativization in complexity theory," *EATCS Bull.*, No. 52 (1994) 229–243.
- [229] S. Fortune, J. Hopcroft, and J. Wyllie, "The directed subgraph homeomorphism problem," *Theoret. Computer Sci.*, Vol. 10 (1980) 111–121.
- [230] R. Fraisse, "Sur la comparaison des types d'ordres," *C. R. Acad. Paris*, Vol. 226 (1948) 1330–1331.

- [231] J. Franco, J. Goldsmith, J. Schlipf, E. Speckenmeyer, and R. Swaminathan, "An algorithm for the class of pure implicational formulas," paper presented at *Siena Workshop on Satisfiability*, Università degli Studi di Siena, Siena, Italy, May 1996.
- [232] G. N. Frederickson and R. Janardan, "Designing networks with compact routing tables," *Algorithmica*, Vol. 3 (1988) 171–190.
- [233] R. Friedberg, "Two recursively enumerable sets of incomparable degrees of unsolvability," *Proc. Nat. Acad. Sci. U.S.A.*, Vol. 43 (1957) 236–238.
- [234] R. Friedberg, "Three theorems on recursive enumeration," *J. Symbol. Logic*, Vol. 23 (1958) 309–316.
- [235] H. Friedman, N. Robertson, and P. Seymour, "The metamathematics of the graph minor theorem," *Logic and Combinatorics* (Arcata, Calif., 1985), 229–261, *Contemp. Math.*, 65, Amer. Math. Soc., Providence, R.I., 1987.
- [236] D. Fulkerson and G. Danzig, "Computations of maximum flow in a network," *Naval Logist. Quart.*, Vol. 2 (1955) 277–283.
- [237] C. P. Gabor, W. L. Hsu, and K. J. Supowit, "Recognizing circle graphs in polynomial time," in *Proc. 26th Ann. IEEE Symp. Foundations of Computer Science*, 1985, 106–116; *J. Assoc. Comput. Mach.*, Vol. 36 (1989) 435–473.
- [238] Z. Galil and X. Yu, "Short length versions of Menger's theorem," in *Proceedings of the 27th Annual Symposium on the Theory of Computing*, ACM Press, 1995, 499–507.
- [239] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness*, Freeman (1979).
- [240] M. R. Garey and R. E. Tarjan, "A linear time algorithm for finding all feedback vertices," *Inform. Process. Lett.*, Vol. 7 (1978) 274–276.
- [241] W. Gasarch, M. W. Krentel, and K. Rappoport, "OptP-Completeness as the Normal Behavior of NP-Complete Problems," *Math. Syst. Theory*, Vol. 28 (1995) 487–514.
- [242] F. Gecseg and M. Steinby, *Tree Automata*, Akademiai Kiado (1984).
- [243] J. G. Geske, "On the structure of intractable sets," Ph.D. thesis, Iowa State University, 1987.
- [244] H. Glover, P. Huneke, and C. Wang, "103 graphs that are irreducible for the projective plane," *J. Combinatorial Theory B*, Vol. 27 (1979) 332–370.
- [245] K. Gödel, "Die vollständigkeit der axiome des logischen funktionenkalküls," *Monatsh. Math. Phys.*, Vol. 37 (1930) 349–360.
- [246] A. Goldberg and R. Tarjan, "A new approach to the maximum flow problem," *J. Assoc. Comput. Mach.*, Vol. 35 (1988) 921–940.
- [247] P. Goldberg, M. Golumbic, H. Kaplan, and R. Shamir, "Three strikes against physical mapping of DNA (extended abstract)," Technical Report, Department of Computer Science, Tel Aviv University, 1993.
- [248] M. Golumbic, H. Kaplan, and R. Shamir, "On the complexity of DNA physical mapping," Technical Report 271/93, Tel Aviv University; *Adv. Appl. Math.* (1993).

- [249] R. Govindan, M. A. Langston, and B. C. Plaut, "Graph partitioning and cutting," Technical Report, Computer Science Department, University of Tennessee, 1997.
- [250] M. Grötschel, L. Lovasz, and A. Schrijver, "The ellipsoidal method and its consequences in combinatorial optimization," *Combinatorica*, Vol. 1 (1981) 169–198.
- [251] M. Grötschel, L. Lovasz, and A. Schrijver, *Geometrie Algorithms and Combinatorial Optimization*, Springer-Verlag (1988).
- [252] Gurevich and L. Harrington, "Automata, trees and games," in *Proceedings 14th STOC*, 1982, 60–65.
- [253] Y. Gurevich and S. Shelah, "Nearly linear time," in *Logic at Botik '89* (ed. Meyer and Taitlin), Lecture Notes in Computer Science Vol. 363, Springer-Verlag (1989) 108–118.
- [254] D. Gusfield, "Efficient methods for multiple sequence alignment with guaranteed error bounds." *Bull. Math. Biol.*, Vol. 55 (1993) 141–154.
- [255] J. Gustedt, "Well quasi ordering finite posets and formal languages," *J. Comb. Theory B.*, Vol. 65 (1995) 111–124.
- [256] M. T. Hallett, personal communication, 1994.
- [257] M. T. Hallett, "An integrated complexity analysis of problems from computational biology," Ph.D. Thesis, Department of Computer Science, University of Victoria, 1996.
- [258] M. T. Hallett, "Shortest Common Supersequence is hard for $W[t]$, for all t ," unpublished.
- [259] M. T. Hallett and H. T. Wareham, "A compendium of parameterized results," *SIGACT News*, Vol. 25(3) (1994) 122–123; <http://www-csc.uvic.ca/~mhallett/research.html>
- [260] F. Harary, *Graph Theory*, Addison-Wesley (1969).
- [261] J. Hartmanis, "Gödel, von Neumann and the $P = NP$ problem," in *Bull. Eur. Assoc. Theoret. Computer Sci.*, No. 38 (1989) 101–107.
- [262] J. Hartmanis, "A broader research agenda for theory. Structural complexity column," *Bull. Eur. Assoc. Theoret. Computer Sci.* (1995).
- [263] J. Hartmanis and H. Lin (eds.), *Computing the Future: A Broader Agenda for Computer Science and Engineering*, National Academy Press (1995).
- [264] J. Hartmanis and R. E. Stearns, "The computational complexity of algorithms," *Trans. Am. Math. Soc.*, Vol. 117 (1965) 285–306.
- [265] F. Henglein and H. G. Mairson, "The Complexity of Type Inference for Higher-Order Typed Lambda Calculi." In *Proc. Symp. on Principles of Programming Languages (POPL)* (1991) 119–130.

- [266] E. Herrmann, "The undecidability of the elementary theory of the lattice of recursively enumerable sets (abstract)," in *Proceedings of the Second Frege Conf., Schwerin, DDR, 1984* (ed. Weschung), Akademie-Verlag (1984) 66–72.
- [267] P. Heusch, "The complexity of the falsibility problem for pure implicational formulas," in *Proc. 20th International Symposium on Mathematical Foundations of Computer Science* (ed. J. Wiedermann and P. Hajek), Lecture Notes in Computer Science Vol. 869, Springer-Verlag (1995) 221–226.
- [268] G. Higman, "Ordering by divisibility in abstract algebras," *Proc. London Math. Soc.*, Vol. 2 (1952) 326–336.
- [269] D. S. Hirschberg, "Recent results on the complexity of common subsequence problems," in *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison* (ed. D. Sankoff and J. B. Kruskal), Addison-Wesley (1983).
- [270] J. Hopcroft and J. Ullmann, *Introduction to Automata Theory*, Addison-Wesley (1979).
- [271] J. E. Hopcroft and R. E. Tarjan, "Efficient planarity testing," *J. ACM*, Vol. 21 (1974) 549–568.
- [272] J. M. Howie, *Introduction to Semigroup Theory*, London (1976).
- [273] D. Huffman, "The synthesis of sequential switching circuits," *J. Franklin Inst.*, Vol. 257 (1954) 3–4, 161–190, 275–303.
- [274] H. B. Hunt III and R. E. Stearns, "The complexity of very simple Boolean formulae with applications," *SIAM J. Comput.*, Vol. 19 (1990) 44–70.
- [275] O. Ibarra and C. Kim, "Fast approximation algorithms for the knapsack and sum of subset problems," *J. Assoc. Comput. Mach.*, Vol. 22 (1975) 463–468.
- [276] N. Immermann, "Expressibility as a logical measure: results and directions," in *Proc. 2nd Annual Structure in Complexity Theory*, Springer-Verlag (1989).
- [277] R. W. Irving and C. B. Fraser, "Two algorithms for the longest common subsequence of three (or more) strings," in *Proceedings of the Third Annual Symposium on Combinatorial Pattern Matching* (ed. A. Apostolico, Crochemore, Z. Galil, and U. Manber), Lecture Notes in Computer Science Vol. 644, Springer-Verlag (1992) 214–229.
- [278] A. Itai and M. Rodeh, "Finding a minimum circuit in a graph," in *Proc. 9th ACM Symposium in Theory of Computing*, 1977, 1–10.
- [279] A. Itai, M. Rodeh, and S. Tanimoto, "Some matching problems in bipartite graphs," *J. Assoc. Comput. Mach.*, Vol. 25(4) (1978) 517–525.
- [280] T. Jiang, E. Lawler, and L. Wang, "Aligning sequences via an evolutionary tree," in *Proceedings of the 26th Annual Symposium on the Theory of Computing*, ACM Press (1994) 760–769.
- [281] D. Johnson, "The NP-completeness Column," *J. Algorithms* (1987).
- [282] D. Johnson, "A taxonomy of complexity classes," in *The Handbook of Theoretical Computer Science*, North-Holland (1996).

- [283] D. Johnson, A. Demers, J. Ullman, M. Garey, and R. Graham, "Worst case performance bounds for simple one dimensional packing algorithms," *SIAM J. Comput.*, Vol. 3 (1974) 299–325.
- [284] D. Joseph, R. Pruim, and P. Young, "Collapsing degrees in subexponential time," in *Proceedings Structure in Complexity Theory, 9th Annual Conf.*, IEEE (1994) 367–382.
- [285] V. Kabanets, "Recognizability equals definability for partial k -paths, in *Proc. 24th International Colloquium on Automata, Languages, and Programming*, LNCS 1256 (1997) 805–815.
- [286] D. Kaller, "Definability equals recognizability for partial 3-trees," in *Proc. Workshop on Graph Theoretical Concepts in Computer Science*, 1996.
- [287] S. Kannan and T. Warnow, "Inferring evolutionary history from DNA sequences," in *Proceedings of the 31st Annual Symposium on the Theory of Computing*, 1990, 362–378.
- [288] H. Kaplan and R. Shamir, "Pathwidth, bandwidth and completion problems to proper interval graphs with small cliques," Technical Report 285/93, Tel Aviv University.
- [289] H. Kaplan, R. Shamir, and R. E. Tarjan, "Tractability of parameterized completion problems on chordal and interval graphs: Minimum fill-in and DNA physical mapping (extended abstract)," in *35th Ann. Proc. of the Foundations of Computer Science (FOCS '94)*, 1994, 780–891.
- [290] R. Kaye, "Characterizing some low complexity classes using theories of arithmetic," M.Sc. thesis, University of Manchester, 1985.
- [291] R. Karp, "Reducibility among combinatorial problems," in *Complexity of Computer Computations* (ed. R. Miller and J. Thatcher), Plenum Press (1972) 45–68.
- [292] R. M. Karp and R. J. Lipton, "Turing machines that take advice," *L'Enseignement Mathématique*, Vol. 28 (1982) 191–209.
- [293] D. Kelly and W. Trotter, "Dimension theory for ordered sets," in *Ordered Sets* (ed. I. Rival), D. Reidel (1981) 171–212.
- [294] J. Kennedy, L. Quintas, and M. Syslo, "The theorem on planar graphs," *Historia Math.*, Vol. 12(4) (1985) 356–368.
- [295] H. Kierstead, " NP -completeness results concerning greedy and supergreedy linear extensions," *Order*, Vol. 3 (1986) 123–134.
- [296] N. G. Kinnarsley and M. A. Langston, "Obstruction Set Isolation for the Gate Matrix Layout Problem," *Discrete Appl. Math.*, Vol. 54 (1994) 169–213.
- [297] C. Kintala and P. Fischer, "Refining nondeterminism in relativised polynomial time bounded computations," *SIAM J. Comput.*, Vol. 9 (1980) 46–53.
- [298] S. C. Kleene, "Representation of events in nerve nets and finite automata," in *Automata Studies*, Princeton University Press (1956) 3–42.
- [299] S. C. Kleene, "The origins of recursive function theory," *Ann. Hist. Comput.*, Vol. 3 (1981) 52–67.

- [300] S. C. Kleene and E. Post, "The uppersemilattice of degrees of recursive unsolvability," *Ann. Math.*, Vol. 59 (1954) 379–407.
- [301] T. Klops, *Treewidth*, PhD thesis, Utrecht University, the Netherlands, 1993.
- [302] N. Koblitz, electronic communication, 1995.
- [303] P. Kolaitis and M. Thakur, "Approximation properties of NP minimization classes," *J. C. S. S.*, Vol. 50 (1995) 391–411.
- [304] A. Kornai and Z. Tuza, "Narrowness, pathwidth and their application in natural language processing," *Discrete Appl. Math.*, Vol. 36 (1992) 87–92.
- [305] D. Kozen, "On the Myhill–Nerode theorem for trees," *Bull. EATCS*, Vol. 47 (1992) 170–173.
- [306] D. Kozen, "On regularity preserving functions," *Bull. EATCS*, Vol. 58 (1996) 131–138.
- [307] J. Kratochvíl, "Regular codes in regular graphs are difficult," *Discrete Math.*, Vol. 133 (1994) 191–205.
- [308] J. Kratochvíl and M. Křivánek, "On the computational complexity of codes in graphs," in *Proc. MFCS Karlovy Vary 1988, Lect. Notes Comp. Science*, Springer-Verlag, Vol. 324 (1988) 396–404.
- [309] J. B. Kruskal, "Well-quasi-ordering, the tree theorem, and Vazsonyi's conjecture," *Trans Am. Math. Soc.*, Vol. 95 (1960) 210–225.
- [310] I. Kriz and R. Thomas, "Clique sums, tree decompositions and compactness," *Discrete Math.*, Vol. 81 (1990) 177–185.
- [311] I. Kriz and R. Thomas, "The Menger-like property of the tree-width of infinite graphs," *J. Comb. Theory B*, Vol. 52 (1991) 86–91.
- [312] J. B. Kruskal, "The theory of well-quasi-ordering: a frequently rediscovered concept," *J. Combinatorial Theory A.*, Vol. 13 (1972) 297–5.
- [313] J. B. Kruskal and D. Sankoff, "An anthology of algorithms and concepts for sequence comparison," in *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison* (ed. D. Sankoff and J. B. Krushal), Addison-Wesley (1983) 265–310.
- [314] A. Kucera, "An alternative priority-free solution to Post's problem," in *Proceedings, Mathematical Foundations of Computer Science* (ed. J. Gruska, B. Rován, and J. Wiederman), Lecture Notes in Computer Science Vol. 233, Springer-Verlag (1986).
- [315] K. Kunen, *Set Theory*, North-Holland (1980).
- [316] K. Kuratowski, "Sur le probleme des courbes gauches en topologie," *Fund. Math.*, Vol. 15 (1930) 271–283.
- [317] A. H. Lachlan, "Lower bounds for pairs of recursively enumerable degrees," *Proc. London Math. Soc.*, Vol. 16 (1966) 537–569.

- [318] A. H. Lachlan, "A recursively enumerable degree which will not split over all lesser ones," *Ann. Math. Logic*, Vol. 9 (1975) 307–365.
- [319] A. H. Lachlan, "Bounding minimal pairs," *J. Symbolic Logic*, Vol. 44 (1979) 626–642.
- [320] R. Ladner, "A completely mitotic nonrecursive recursively enumerable degree," *Trans. Am. Math. Soc.*, Vol. 184 (1973) 479–507.
- [321] R. Ladner, "On the structure of polynomial time reducibility," *J. Assoc. Comput. Mach.*, Vol. 22 (1975) 155–171.
- [322] J. Lagergren, *Algorithms and minimal forbidden minors for tree decomposable graphs*, Ph.D. Diss., Dept. of Numerical Analysis and Computing Sciences, Royal Institute of Technology, Stockholm, Sweden, March 1991.
- [323] J. Lagergren, "Upper bounds on the sizes of obstructions and intertwines," preprint, 1996.
- [324] J. Lagergren and S. Arnborg, in preparation.
- [325] S. L. Lauritzen and D. J. Spiegelhalter, "Local computations with probabilities on graphical structures and their applications to expert systems," *J. Roy. Statist. Soc. Ser. B*, Vol. 50 (1988) 157–224.
- [326] E. Lawler, J. Lenstra, and A. Rinnooy Kan, "Generating all maximal independent sets: NP-hardness and polynomial time algorithms," *SIAM J. Comput.*, Vol. 9 (1980) 558–565.
- [327] E. Leggett and D. Moore, "Optimization problems and the polynomial time hierarchy," *Theoret. Computer Sci.*, Vol. 15 (1981) 279–289.
- [328] E. Leiss, "The complexity of restricted regular expressions and the synthesis problem for finite automata," *J. Comput. Syst. Sci.*, Vol. 23 (1981) 348–354.
- [329] M. Lerman, "Admissible ordinals and priority arguments," in *Cambridge Summer School in Mathematical Logic* (ed. A. Mathias and H. Rogers), Lecture notes in Mathematics Vol. 337, Springer-Verlag (1973) 311–344.
- [330] L. Levin, "Universal sorting problems" (English Translation), *Problems Informa. Transmiss.*, Vol. 9 (1973) 265–266.
- [331] H. R. Lewis, "Complexity results for classes of quantificational formulae," *J. Computer Syst. Sci.*, 21 (1980) 317–353.
- [332] D. Lichtenstein, "Planar formulae and their uses," *SIAM J. Comput.*, Vol. 11 (1982) 329–343.
- [333] R. Lipton and R. Tarjan, "Applications of a planar separator theorem," in *Proc. 18th FOCS*, IEEE (1977) 162–170.
- [334] A. Lopez and H. Law, "A dense gate matrix layout method for MOS VLSI," *IEEE Trans. Electron. Devices*, Vol. ED-27 (1980) 1671–1675.
- [335] L. Lovasz, *Combinatorial Problems and Exercises*, Akadémiai Kiadó (1979).
- [336] L. Lovasz and M. Plummer, *Matching Theory*, North-Holland (1986).

- [337] S. Y. Lu and K. S. Fu, "A sentence-to-sentence clustering procedure for pattern analysis," *IEEE Trans. Syst., Man Cybern.*, Vol. 8 (1978) 381–389.
- [338] A. Luca and S. Varricchio. "Some regularity conditions based on well-quasi-orders." In: *Proc. 1st Latin American Symposium on Theoretical Informatics*, Sa Paulo, Brazil, April 1992. Springer-Verlag, Lecture Notes in Comp. Sci. Vol. 583, (1992), 356–371.
- [339] E. Luks, "Isomorphism of graphs of bounded valence can be tested in polynomial time," *J. Comput. Syst. Sci.*, Vol. 25 (1982) 42–65.
- [340] W. Mader, "Hinreichende bedingungen fuer die existenz von teilgraphen, die zu einem vollstaendigen graphen homomorph zind," *Abt. Math Sem. Hamburg Univ.* **37** (1972) 86–97.
- [341] M. Mahajan and V. Raman, "Parametrizing above guaranteed values: MaxSat and MaxCut," Tech. Report TR97-033, University of Trier.
- [342] S. Mahaney, "Sparse complete sets for *NP*: solution of a conjecture of Berman and Hartmanis," *J. Comput. Syst. Sci.*, Vol. 25 (1985) 130–143.
- [343] D. Maier, "The complexity of some problems on subsequences and supersequences," *J. ACM*, Vol. 25 (1978) 322–336.
- [344] F. S. Makedon and I. H. Sudborough, "On Minimizing Width in Linear Layouts", *Lecture Notes in Computer Science* Vol. 154, Springer-Verlag (1983) 478–490.
- [345] J. Matousek and R. Thomas, "Algorithms for finding tree-decompositions of graphs," *J. Algorithms*, Vol. 12 (1991) 1–22.
- [346] W. McCulloch and W. Pitts, "A logical calculua of the ideas immanent in nervous activity," *Bull. Math. Biophys.*, Vol. 5 (1943) 115–133.
- [347] F. McMorris, T. Warnow, and T. Wimer, "Triangulated vertex coloured graphs," in *Proceedings of the 4th Annual Symposium on Discrete Algorithms*, Austin, Texas, 1993, 120–127.
- [348] R. McNaughton and H. Yamada, "Regular expressions and state graphs for automata," *IEEE Trans. Electron. Computers*, Vol. ED-9 (1960) 39–47.
- [349] H. Mealy, "A method for synthesising sequential circuits," *Bell Syst. Tech. J.*, Vol. 34 (1955) 1045–1079.
- [350] K. Menger, "Zur allgemeinen kurventheorie," *Fund. Math.*, Vol. 10 (1927) 96–115.
- [351] J. Mezei and J. Wright, "Algebraic automata and context free sets," *Inform. Control*, Vol. 11 (1967) 3–29.
- [352] E. C. Milner, "Basic WQO- and BQO-theory," in *Graphs and Orders* (ed. I. Rival), D. Reidel (1985) 487–502.
- [353] D. Mitchell, B. Selman, and H. Levesque, "Easy and hard distributions of SAT problems," in *Proceedings AAAI '92*, 1992, 459–465.
- [354] B. Mohar, "Embedding graphs in an arbitrary surface in linar time," in *Proceedings of the 50th ACM Symposium on the Theory of Computing, STOC '96*, ACM Press (1996) 392–397.

- [355] E. Moore, "Gedanken experiments on sequential machines," *Automata Studies*, Princeton University Press (1956) 129–153.
- [356] B. Monien. How to find long paths efficiently. *Ann. Discrete Math.*, Vol. 25 (1985) 239–254.
- [357] A. Muchnik, "On the unsolvability of the problem of reducibility in the theory of algorithms," *Dokl. Akad. Nauk. S.S.S.R.*, Vol. 106 (1956) 194–197.
- [358] J. Myhill, "Finite automata and representation of events," WADD TR-57-624, Wright-Patterson AFB, Ohio, 1957, 112–137.
- [359] A. Naik, K. Regan, and D. Sivakumar, "Quasilinear time complexity theory," in *STACS*, Lecture Notes in Computer Science, Springer-Verlag (1994) 97–108.
- [360] C. St. J. A. Nash-Williams, "Well-quasi-ordering finite trees," *Proc. Camb. Philos. Soc.*, Vol. 59 (1963) 291–296.
- [361] A. Nerode, "Linear automaton transformations," *Proc. Amer. Math. Soc.*, Vol. 9 (1958) 541–544.
- [362] A. Nerode, "Extensions to isols," *Ann. Math.*, Vol. 73 (1961) 362–403.
- [363] J. Nesetril and S. Poljak, "On the complexity of the subgraph problem," *Commun. Math. Univ. Carol.*, Vol. 26 (1985) 415–419.
- [364] P. Odifreddi, *Classical Recursion Theory*, North-Holland (1990).
- [365] P. Odifreddi, *Classical Recursion Theory*, Vol. 2, North-Holland, to appear.
- [366] C. Papadimitriou and M. Yannakakis, "Optimization, approximation, and complexity classes," *J. Comput. Syst. Sci.*, Vol. 43 (1991) 425–440.
- [367] C. H. Papadimitriou and M. Yannakakis, "The complexity of facets (and some facets of complexity)," *J. Comput. Syst. Sci.*, Vol. 28 (1984) 244–259.
- [368] C. Papadimitriou and M. Yannakakis, "On limited nondeterminism and the complexity of the V-C dimension," in *Eight Annual Conference on Structure in Complexity Theory* (1993) 12–18.
- [369] T. D. Parsons, "Pursuit-evasion in a graph," in *Theory and Applications of Graphs* (ed. Y. Alavi and D. R. Lick), Springer-Verlag (1976) 426–441.
- [370] A. Paz, and S. Moran, "Nondeterministic polynomial optimization problems and their approximations," *Theoret. Computer Sci.*, Vol. 15 (1981) 251–277.
- [371] W. R. Pearson and W. Miller, "Dynamic programming algorithms for biological sequence comparison," *Methods Enzymol.*, Vol. 183 (1992) 575–601.
- [372] P. A. Pevzner, "Multiple alignment, communication cost, and graph matching," *SIAM J. Appl. Math.*, Vol. 52 (1992) 1763–1779.
- [373] J. Plehn and B. Voigt, "Finding minimally weighted subgraphs," in *Proc. 16th Int. Worksh. Graph-Theoretic Concepts in Computer Science*, Lecture Notes in Computer Science Vol. 484, Springer-Verlag (1991) 18–29.
- [374] B. Poizat, " $Q = NQ?$," *J. Symbolic Logic*, Vol. 51 (1986) 22–32.

- [375] E. Post, "Recursively enumerable sets of integers and their decision problems," *Bull. Am. Math. Soc.*, Vol. 12 (1944) 248–316.
- [376] M. Pouzet, "Applications of well quasi-ordering and better quasi-ordering," in *Graphs and Orders* (ed. I. Rival), D. Reidel (1985) 503–519.
- [377] A. J. Power, "Four NP-complete embedding problems," Monash University Logic Paper No. 29, January 1981.
- [378] B. Pulleybank, "On minimizing setups in precedence constrained scheduling," *Discrete Appl. Math.*, Vol. 11 (1985).
- [379] M. Rabin, "Decidability of second-order theories, and automata on infinite trees," *Trans. Am. Math. Soc.*, Vol. 141 (1969) 1–35.
- [380] M. Rabin and D. Scott, "Finite automata and their decision problems," *IBM J. Res.*, Vol. 3(2) (1959) 115–125.
- [381] F. P. Ramsey, "On a problem of formal logic," *Proc. London Math. Soc.*, Vol. 30 (1930) 264–286.
- [382] B. Reed, "Finding approximate separators and treewidth quickly," *Proc. 24th STOC*, 1992, 221–228.
- [383] B. Reed, N. Robertson, A. Schrijver, and P. Seymour, "Finding disjoint trees in planar graphs in linear time," in *Graph Structure Theory* (ed. N. Robertson and P. D. Seymour), *Contemporary Math.*, Vol. 147 (1993) 295–301.
- [384] J. Reif, "Complexity of the Generalized Mover's Problem," in *Planning, Geometry, and Complexity of Robot Motion* (ed. J. Schwartz, M. Sharir, J. Hopcroft), Ablex Publishing Corporation (1987) 267–281.
- [385] K. Regan, "Finite substructure languages," in *Proc. 4th Structure in Complexity Conf.*, 1989, 87–96.
- [386] H. G. Rice, "Classes of recursively enumerable sets and their decision problems," *Trans. Am. Math. Soc.*, Vol. 74 (1953) 358–366.
- [387] E. Ristad, "Complexity of Simplified Segmental Phonology," Technical Report CS-TR-388-92 (revised May 1993), Department of Computer Science, Princeton University.
- [388] N. Robertson and P. D. Seymour, "Disjoint paths—a survey," *SIAM J. Discrete Math.*, Vol. 6 (1985) 300–305.
- [389] N. Robertson and P. D. Seymour, "Graph minors. I. Excluding a forest," *J. Combinatorial Theory B*, Vol. 35 (1983) 39–61.
- [390] N. Robertson and P. D. Seymour, "Graph minors. II. Algorithmic aspects of tree-width," *J. Algorithms*, Vol. 7 (1986) 309–322.
- [391] N. Robertson and P. D. Seymour, "Graph minors. IV. Tree-width and well quasi-ordering," *J. Combinatorial Theory B*, Vol. 48 (1990) 227–254.
- [392] N. Robertson and P. D. Seymour, "Graph minors. III. Planar tree-width," *J. Combinatorial Theory B*, Vol. 36 (1984) 49–64.

- [393] N. Robertson and P. D. Seymour, "Graph minors. V. Excluding a planar graph," *J. Combinatorial Theory B*, Vol. 41 (1986) 92–114.
- [394] N. Robertson and P. D. Seymour, "Graph minors. VIII. A Kuratowski theorem for general surfaces," *J. Combinatorial Theory B*, Vol. 48 (1990) 255–288.
- [395] N. Robertson and P. D. Seymour, "Graph minors. X. Obstructions to tree decomposition," *J. Combinatorial Theory B*, Vol. 52 (1991) 153–190.
- [396] N. Robertson and P. D. Seymour, "Graph minors. XIII. The disjoint paths problem," *J. Combinatorial Theory, Ser. B*, Vol. 63 (1995) 65–110.
- [397] N. Robertson and P. D. Seymour, "Graph minors. XX. Wagner's conjecture," to appear.
- [398] N. Robertson and P. D. Seymour, *Graph structure Theory*, American Mathematical Society, *Contemporary Mathematics*, Vol. 147, (1991).
- [399] G. Robinson and D. Welch, "The computational complexity of matroid properties," *Math. Proc. Camb. Phil. Soc.*, Vol. 87 (1980) 29–45.
- [400] H. Rogers, Jr., *Theory of Recursive Functions and Effective Computability*, McGraw Hill (1968).
- [401] H. Sachs, "On spacial representation of finite graphs," in *Colloquia Mathematica Societatis Janos Bolyai*, Vol. 37, *Finite and Infinite Sets* (ed. A. Hajnal, L. Lovasz, and V. Sos), North-Holland (1984) 649–662.
- [402] G. Sacks, "On the degrees less than $\mathbf{0}'$," *Ann. Math.*, Vol. 77 (1963) 211–231.
- [403] G. Sacks, "The recursively enumerable degrees are dense," *Ann. Math.*, Vol. 80 (1964) 300–312.
- [404] S. Sahni, "Approximate algorithms for the 0/1 knapsack problem," *J. ACM*, Vol. 22 (1975) 115–124.
- [405] D. Sankoff, "Matching comparisons under deletion/insertion constraints," *PNAS*, Vol. 69 (1972) 4–6.
- [406] D. Sankoff, "Simultaneous solution of the RNA folding, alignment, and protosequence problems," *SIAM J. Appl. Math.*, Vol. 45 (1985) 810–825.
- [407] N. Santoro and R. Khatib, "Labeling and implicit routing in networks," *Computer J.*, Vol. 28 (1985) 5–8.
- [408] W. Savitch, "Relationships between nondeterministic and deterministic tape complexities," *J. Comput. Syst. Sci.*, Vol. 4 (1970) 177–192.
- [409] T. J. Schaefer, "Complexity of some two-person perfect information games," *J. Comput. Syst. Sci.*, Vol. 16 (1978) 185–225.
- [410] C. P. Schnorr, "Optimal algorithms for self-reducible problems," *Proc. ICALP*, 1976, 322–337.
- [411] L. Schoenfeld, *Math. Comp.*, Vol. 30 (1976) 337–360.

- [412] A. Schrijver, "Finding k disjoint paths in a directed planar graph," *SIAM J. Comput.*, Vol. 23(4) (1994) 780–788.
- [413] J. Schwartz, and M. Sharir, "Algorithmic motion planning in robotics," in *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity* (ed. J. van Leeuwen), The MIT Press (1990) 391–430.
- [414] D. Seese, "Entscheidbarkeit- und Interpretierbarkeitsfragen monadischer Theorien zweiter Stufe gewisser Klassen von Graphen," Diss. A. Humboldt University, Berlin, 1976.
- [415] D. Seese, "The structure of models of decidable monadic theories of graphs," *Ann. Pure and Appl. Logic*, Vol. 53 (1991) 169–195.
- [416] M. Serna and R. Spirakis, "The approximability of problems complete for P ," Technical Report.
- [417] P. Seymour, "Recognizing graphic matroids," *Combinatorica*, Vol. 1 (1981) 75–78.
- [418] A. Shamir, " $IP = PSPACE$," *FOCS*, Vol. 31 (1991) 145–152.
- [419] S. Shelah, "The monadic theory of order," *Ann. Math.*, Vol. 102 (1975) 379–419.
- [420] J. Shinoda, personal communication, 1991.
- [421] J. Shinoda and T. Slaman, "On the theory of PTIME degrees of recursive sets," *J. Comput. Syst. Sci.*, Vol. 41 (1990) 321–366.
- [422] J. R. Shoenfield, "Undecidable and creative theories," *Fund. Math.*, Vol. 49 (1961) 171–179.
- [423] J. Shoenfield, "Application of model theory to degrees of unsolvability," in *Symposium on the Theory of Models* (ed. J. Addison, L. Henkin, A. Tarski), North-Holland (1965) 359–363.
- [424] H. U. Simon, "Continuous reductions among combinatorial optimization problems," *Acta Informatica*, Vol. 26 (1989) 771–785.
- [425] S. Simpson, *Subsystems of Second Order Arithmetic*, Springer-Verlag, to appear.
- [426] M. Sipser, "Borel sets and circuit complexity," in *Proceedings 15th STOC*, 1983, 61–69.
- [427] M. Sipser and D. A. Spielman, "Expander codes," *IEEE Trans. Inform. Theory*, Vol. 42 (1996) 1710–1722.
- [428] D. Sleator, R. Tarjan, and W. Thurston, "Rotation distance, triangulations and hyperbolic geometry," *J. Am. Math. Soc.*, Vol. 1(3) (1988) 647–681.
- [429] W. Slough and K. Winklmann, "On limitations of transformations between combinatorial problems," *Math. Syst. Theory*, 24(3) (1991) 149–168.
- [430] R. I. Soare, "Tree arguments in recursion theory and the $0'''$ priority method," in *Recursion Theory* (ed. A. Nerode and R. A. Shore), Proc. Symp. Pure Math., American Mathematical Society (1985) 53–106.
- [431] R. I. Soare, *Recursively Enumerable Sets and Degrees*, Springer-Verlag (1987).

- [432] E. Spilrajn, "Sur l'extension de l'ordre partiel," *Fund. Math.*, Vol. 16 (1930) 386–389.
- [433] M. Steel, "The complexity of reconstructing trees from qualitative characters and subtrees," *J. Classification*, Vol. 9 (1992) 91–116.
- [434] M. Steinby, "On generalizations of the Myhill–Nerode Theorems," *B. EATCS*, Vol. 48 (1992) 191–196.
- [435] J. Stern, "Approximating the number of error locations within a constant ratio is *NP*-complete," *Lect. Notes Comp. Sci.*, Springer-Verlag, Vol. 673 (1993) 325–331.
- [436] L. Stockmeyer and A. Chandra, "Alternation," *J. Assoc. Comput. Sci.*, Vol. 28 (1981) 114–133.
- [437] L. Stockmeyer and A. Chandra, "Provably difficult combinatorial games," *SIAM J. Comput.*, Vol. 8 (1979) 151–174.
- [438] R. M. Tanner, "A recursive approach to low-complexity codes," *IEEE Trans. Inform. Theory*, Vol. 27 (1981) 533–547.
- [439] R. E. Tarjan, "Depth-first search and linear graph algorithms," *SIAM J. Comput.*, Vol. 1 (1972) 146–160.
- [440] R. E. Tarjan and M. Yannakakis, "Simple linear time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs," *SIAM J. Comput.*, Vol. 13 (1984) 566–579.
- [441] J. Thatcher and J. Wright, "Generalized finite automata," *Notices Am. Math. Soc.*, Vol. 12 (1965) 820.
- [442] R. Thomas, "A Menger-like property of tree-width, the finite case," *J. Combinatorial Theory, B*, Vol. 48 (1990) 67–76.
- [443] R. Thomas, "Well-quasi-ordering infinite graphs with forbidden finite planar minor," *Trans. Am. math. Soc.*, Vol. 312 (1989) 279–313.
- [444] R. Thomas, "A counterexample to 'Wagner's Conjecture' for infinite graphs," *Math. Proc. Camb. Phil. Soc.*, Vol. 103 (1988) 55–57.
- [445] C. Thomassen, "The graph genus problem is *NP*-complete," *J. Algorithms*, Vol. 10 (1989) 568–576.
- [446] K. Thompson, "Regular expression search algorithm," *Communi. Assoc. Comput. Mach.*, Vol. 11 (1968) 419–422.
- [447] M. Thorup, "All structured programs have small tree-width and good register allocation," preprint, University of Copenhagen, 1997.
- [448] S. Toda, "*PP* is as hard as the polynomial time hierarchy," *SIAM J. Comput.*, Vol. 20 (1991) 865–877.
- [449] A. M. Turing, "On computable numbers, with an application to the Entscheidungsproblem," *Proc. London Math. Soc.*, Vol. 42 (1936) 230–265; correction Vol. 43 (1937) 544–546.

- [450] W. Tutte, "An algorithm for recognising whether a given binary matroid is graphic," *Proc. Am. Math. Soc.*, Vol. 11 (1962) 905–917.
- [451] J. Valdes, R. E. Tarjan and E. L. Lawler, "The recognition of series parallel graphs," *SIAM J. Comput.*, Vol. 11 (1982) 298–313.
- [452] L. Valiant, "The complexity of computing the permanent," *Theoret. Comput. Sci.*, Vol. 8 (1979) 189–201.
- [453] L. Valiant, "The complexity of enumeration and reliability problems," *SIAM J. Comput.*, Vol. 8 (1979) 410–421.
- [454] L. Valiant, "A theory of the learnable," *Commun. ACM*, Vol. 27(11) (1984) 1134–1142.
- [455] L. Valiant, "Learning disjunctions of conjunctions," in *Proceedings of the 9th Int'l. Joint Conf. on Artificial Intelligence*, Los Angeles, CA, 1985, 565–569.
- [456] L. Valiant and V. Vazirani, "NP is as easy as detecting unique solutions," *Theoret. Comput. Sci.*, Vol. 47 (1986) 85–93.
- [457] L. Van Leeuwen, *Handbook of Theoretical Computer Science*, Vol. A. North Holland (1990).
- [458] A. Vardy, "Algorithmic complexity in coding theory and the minimum distance problem," in *Proc. 29th Annual ACM Symp. Theory of Computing*, El Paso, TX, (1997) 92–109.
- [459] A. Vardy, "The intractability of computing the minimum distance of a code," *IEEE Trans. Inform. Theory*, Vol. 43(6) (1997), to appear.
- [460] D. Vertigan and G. Whittle, "Recognising polymatroids associated with hypergraphs," *Combinatorics, Probability and Computing*, Vol. 2 (1993) 519–530.
- [461] K. Wagner, "Über einer eigenschaft der ebener complexe," *Math. Ann.*, Vol. 14 (1937) 570–590.
- [462] H. T. Wareham, personal communication, 1994.
- [463] C. Wilson, *Relativization, Reducibilities, and the Exponential Time Hierarchy*, Ph.D. Thesis, University of Toronto, 1980.
- [464] T. Wimer, S. Hedetniemi, and R. Laskar, "A methodology for constructing linear time algorithms," *Congressus Numerantium*, Vol. 50 (1985) 43–60.
- [465] Xiu Yan Liu, MSc Thesis, University of Victoria, 1994.
- [466] M. Yannakakis, "Computing the minimum fill-in is NP-complete," *SIAM J. Algebr. Discrete Methods*, Vol. 2 (1981).
- [467] M. Yannakakis, "Perspectives on database theory," *Proc. Symp. Foundations of Computer Science (FOCS'95)*, IEEE (1995) 224–246.
- [468] A. Yao, "Separating the polynomial-time hierarchy by oracles," in *Proc. 26th Annual Symposium on Foundations of Computer Science*, 1985, 1–10.

- [469] Chee K. Yap, "Some consequences of non-uniform conditions on uniform classes." *Theoretical Computer Science*, Vol. 26 (1983) 287–300.
- [470] C. E. M. Yates, "A minimal pair of recursively enumerable degrees," *J. Symbolic Logic*, Vol. 31 (1966) 159–168.
- [471] C. E. M. Yates, "The degrees of index sets II," *Trans. Amer. Math. Soc.*, Vol. 135 (1969) 249–266.

Index

- $\approx$, 81
- $\oplus$, 113
- $\otimes$, 113
- $\partial(G)$, 113
- α -BALANCED SEPARATOR**, 460
- α -star (of trees), 95
- Abello, J., 197
- ABOVE**(x), 145
- Abrahamson, K., 10, 28, 105, 113, 119, 181, 202, 251, 316, 317, 323, 327–329, 332, 333, 335, 336, 339, 432, 449, 474–479, 484
 - references to Abrahamson, Downey and Fellows [3], 323, 327, 339, 476, 477
 - references to Abrahamson, Downey and Fellows [4], 316, 317, 323, 327–329, 333, 335, 336, 339, 364, 449, 474–478, 484, 479
- acyclic hypergraph, 104
- Adachi, A., 342
- advice, 61
 - class, 62
 - function, 62
 - view, 26, 234, 331, 433
- Agawala, R., 446
- Agnew, G., 10
- Ahuja, R., 167
- Akkermann’s function, 26
- almost minimal cycle, 33
- almost trees, 103
- ALMOST**(x), 33
- Alon, N., 12, 220, 222
- Alstrup, S., 10
- ALTERNATING HITTING SET**, 332, 337, 464
 - RESTRICTED**, 332, 339, 449
- Alvarez, C., 369
- Ambos-Spies, K., 409, 410, 413, 415, 435, 436
- Angluin, D., 197
 - type exact learning, 256
- annihilator, 326
- antichain, 144
- ANTICHAIN OF r -SUBSETS**, 223, 447
- ANTIMONOTONE**
 - NEARLY**, 295
- antimonotone, 230
- approximation algorithm, 50
- approximation problem
 - ratio for, 50
- approximation scheme
 - fully polynomial time, 50

- ARC-PRESERVING LONGEST COMMON SUBSEQUENCE**, 453
- Archdeakon, D., 180
- Arkin, E., 200
- Arnborg, S., 105, 141
- Arora, S., 202, 485
- assemblies
 - control, 272
 - selection, 272
- augmenting paths
 - in a transportation network, 168
- Ausiello, S., 455, 464
- automaton
 - deterministic, 67
- DETERMINISTIC FINITE AUTOMATA ACCEPTANCE**, 68
- McNaughton and Yamada's Construction, 73
- Method of Test Sets, 82
- Myhill–Nerode Theorem, 80
- nondeterministic, 69
- NONDETERMINISTIC FINITE AUTOMATA ACCEPTANCE**, 72
- Rabin and Scott's Theorem, 70
- tree
 - acceptance, 90
 - evaluation of, 90
 - Kleene's Theorem for, 96
 - leaf to root, 89
 - nondeterministic, 92
 - root-to-leaf, 90
 - with λ -moves, 73
- automorphism
- COLORED GRAPH AUTOMORPHISM**, 460
- autoreducible, 412
- AW*
 - [*], 335
 - complete, 479
 - [*P*], 333
 - complete, 479
 - [*SAT*], 333
 - complete, 478
 - hard, 478
 - [*t*], 335
 - complete, 479
- Babai, L., 485
- bad, 144
- Baeza-Yates, R., 268
- bag function, 171
- bags
 - in a tree decomposition, 100
- Baker, T., 369, 394, 410, 412, 430
- Balasubramanian, R., 5, 35, 40
- Balcazaar, J., 410
- BANDWIDTH**, 9, 123, 468
- TOPOLOGICAL**, 472
- bandwidth
 - definition of, 123
- bandwidth k -, 103
- Bar-Hillel, Y., 85
- Barefoot, C., 200
- basic hardness class, 237
- BASIC ONE-PEBBLE GAME**, 348
- Bauderon, M., 141
- Bennett, C., 436
- Berlekamp, E., 459
- beta hierarchy, 250
- Bienstock, D., 178, 188, 448
- BIN PACKING**, 59
- BIPARTITE GRAPH DELETION**, 45
- block, 147
- Blum, M., 410
- Speedup Theorem, 409
- Bodlaender, H., 18, 105, 198, 209, 232, 269, 300, 301, 312, 442, 444–446, 449, 450, 454, 460, 463, 464, 466–469, 472
- Bodlaender's Algorithm, 109
- Bodlaender's Theorem, 104
- Booth, K., 252, 253
- Boppana, R., 358
- boundaried graphs
 - parsing, 114
- bounded
 - injury, 398
 - treewidth, 102
- BOUNDED HAMMING WEIGHT DISCRETE LOGARITHM**, 483
- BOUNDED SUBSET SUM**, 65
- BP*, 356
- branching
 - of a tree decomposition, 172
- Brightwell, G., 452
- Bryant, D., 47, 445

- Buchi, J. R., 140
 Buss, J., 250, 369
 Buss, S., 5, 40, 63, 451
- Cai, Chen, Downey, and Fellows
 references to [109], 251, 259, 281,
 312, 333, 456, 457, 460, 465
 references to [110], 27, 28, 61–66,
 333, 434, 445, 451
- Cai, Leizhen, 11, 43
- Cai, Liming, 249, 251, 281, 312, 331,
 333, 370, 451, 456, 457, 460,
 465, 468
- Canny, J., 473
- canonical partial ordering, 181
- Cantor, G., 391, 409
- capacity
 in a transportation network, 167
- CAT AND MOUSE GAME**, 349
- CAT AND MOUSE PROBLEM**, 349,
 480
- Catalan number, 39
- Cattell, K., 208
- Cattell, M., 180
- cell, 157
- certified computations, 405
- Cesati, M., vii, 473, 479
- chain, 144
- chain minor, 154
- CHAIN MINOR ORDERING**, 154,
 484
- CHAIN REACTION CLOSURE**, 327,
 474
- Chang, R., 410
- CHARACTER COMPATIBILITY**
 BINARY CLADISTIC, 453
 BINARY QUALITATIVE, 453
- CHEAP TOUR**
 EXACT, 461
 SHORT, 281, 484
- Chegiredy, C., 448
- Chen, J., 249, 251, 281, 312, 331, 333,
 370, 451, 456, 460, 468
- Chervonenkis
 VAPNIK CHERVONENKIS
 DIMENSION, 256
- Cholak, P., 418, 432, 435, 436
- Chor, B., 410
- chordal graph, 103
- CHORDAL GRAPH COMPLETION**,
 39, 44
- CHORDAL GRAPH DELETION**, 45
- Church, A., 232, 409
- CIRCUIT**
 SATISFIABILITY
 WEIGHTED WEFT t DEPTH h ,
 237
 SHORT SATISFIABILITY, 475
 WEIGHTED SATISFIABILITY, 315
 MONOTONE, 317, 476
 PLANAR, 476
- circuit
 AC^0 , 354
 antimonotone, 243
 nearly antimonotone, 296
 boolean, 236
 complexity, 355
 decision, 237
 depth of, 237
 enforcement, 248, 296, 366
 $G[t]$, 354
 monotone, 243
 normalized, 239, 244
 planar, 329
 product-of-sums, 240
 sum-of-products, 240
 t -normalized, 284
 tree, 289
 uniform, 357
 universal, 358
 weft of, 237
- CIRCUIT SATISFIABILITY**
 SHORT, 316
- circular arc, 103
- Clark, L., 200
- CLIQUE**, 248, 259, 453
 DOMINATING, 463
- clique-join, 191
- CNF SATISFIABILITY**, 228
- Cohen, P., 409
 forcing, 392
- coherence
 criterion, 396
- Coles, R., 432
- COLORED**
 AUTOMORPHISM, 233
 CUTWIDTH, 469
 GRAPH AUTOMORPHISM, 460

COLORED (*continued*)**GRAPHS****TRIANGULATING**, 472**PROPER INTERVAL GRAPH****COMPLETION**, 461

compact routing schemes, 198

COMPACT TM COMPUTATION, 333

complete set of operators, 114

complexity

circuit, 355

logical, 76, 354

parameterized, 26

space, 62

fixed parameter, 62

time, 28

component

choice, 336

clause, 246

for 10.2, 244

replacement of bottommost small

gates, 293

variable selection, 285, 326

composition operator

 t -boundaried graphs, 113

computational biology, 44

congruence

 $\approx$, 81

canonical congruence for graphs, 119

right canonical, 80

right for graphs, 119

CONSECUTIVE BLOCK**MINIMIZATION**, 46, 442

consistency check, 250

constructivity

non-

algorithmic, 201

solution, 202

Conway, J., 24

Cook, S., 202, 232, 235

circuit

logspace, 326

Cook's Theorem, 17

analog of, 239

Normalization Theorem, 284

Cook's theorem, 236

reductions, 227

Cornell, D., 112

Courcelle, B., 141, 441

Courcelle's Theorem, 103, 131

**CROSSING NUMBER FOR MAX
DEGREE 3 GRAPHS**, 442

cut

in a transportation network, 167

value of, 125

CUTWIDTH, 125, 195, 442**COLORED**, 469**CUTWIDTH**, 9

cutwidth, 125

cutwidth k -, 103

cycle

almost minimal, 33

cycle cover number, 214

cyclic shift, 115

D'Atri, A., 464

Dahlhaus, E., 482

Danzig, G., 167

DATA CLEANING, 3

Davis, M., 264

Day, W., 268, 453

Dean, T., 11

DECISION PROBLEM

associated with an optimization

problem, 51

decomposition

star complete, 175

definition

basic working, 229

degree, 390

polynomial time, 404

DEGREE 3 SUBGRAPH**ANNIHILATOR**, 323,

474

gadget for, 324

Dejter, I., 200

Demers, A., 59

Dennenberg, L., 12, 220

density theorem

Downey and Fellows, 425

Ladner's, 403

Sacks, 406

Shinoda's, 406

depth, 237

depth-first, 211

spanning tree, 211

derivation

of trees, 93

- SHORT CONTEXT SENSITIVE DERIVATION**, 455
- SHORT UNRESTRICTED GRAMMAR DERIVATION**, 456
- DIAMETER IMPROVEMENT FOR PLANAR GRAPHS**, 442
- Diaz, J., 369, 410
- Diestel, R., 178
- Ding, G., 192
- Dinneen, M., 180, 192, 208
- DIRECTED FEEDBACK VERTEX SET**, 482
- DISJOINT CYCLES**, 214
- DISJOINT PATHS**, 443
 - PLANAR DIRECTED**, 482
- DISJOINT r -SUBSETS**, 223, 447
- DNA physical mapping, 44
- DOMINATING CLIQUE**, 463
- DOMINATING SET**, 17, 30, 285, 463
 - ALTERNATING**, 339
 - DOMINATING THRESHOLD SET**, 464
 - gadget for, 285
 - INDEPENDENT**, 464
 - PLANAR**, 30, 31
 - logspace, 66
 - PLANAR RED/BLUE**, 38
 - restricted to max degree t graphs, 37
 - TOURNAMENT**, 300, 309, 375, 465
- dominating set
 - weak, 288
- DOMINATING SET-B**, 60
- DOMINO TREEWIDTH**, 469
- Douthett, J., 200
- Downey and Fellows
 - references to [169], 24, 256, 257, 309, 311, 442, 444, 445, 451, 457, 464, 465
 - references to [165], 24, 232, 299, 454, 464
 - references to [166], 232, 299, 336, 465, 468
 - references to [170], 251, 277, 279, 281, 299, 336, 454, 455, 458–461, 484
 - references to [168], 416, 417, 425, 430, 432
- Downey, R., 5, 18, 40, 269, 403, 414, 454, 478
- Doyle, M., 40, 129
- Doyle, Mike, vii
- Dress, A., 446
- Dreyfus, S., 450
- DUAL OF COLORING**, 46, 443
- DUAL OF IRREDUNDANT SET**, 46, 443
- Ebbinghaus, H., 59
- edge echo, 272
- Edmonds, J., 448
- El-Zahar, M., 485
- Ellis, J., 28, 251, 316, 474, 475
- embedding, 252, 253, 455
 - BIPARTITE GRAPH EMBEDDING**, 252
 - SEMIGROUP EMBEDDING**, 252, 455
 - SEMILATTICE EMBEDDING**, 252, 455
- enforcement, 248
 - circuit, 366
 - gadget, 340
 - gap, 285
- Engelfreit, J., 469
- Entringer, R., 200
- Evans, P., vii, 209, 257, 311, 453, 457
- EXACT COVER**, 467
- EXACT LONG CYCLE**, 213, 220, 445
- expression
 - normalized boolean, 289
 - regular, 76
 - tree, 95
- FACE COVER NUMBER FOR PLANAR GRAPHS**, 38
- FACTORIZATION**
 - BOUNDED FACTOR**, 452
 - H_n , 312
 - MONOID**, 465
 - PERMUTATION GROUP**, 281, 460
- FACTORIZATION MONOID**, 312
- Fagin, R., 55, 375
- fanin, 89, 237
- fanout, 237
- FEASIBLE REGISTER ASSIGNMENT**, 469

- feasible solutions
 - to an optimization function, 50
- FEEDBACK VERTEX SET**, 32, 196, 214, 444
 - DIRECTED**, 482
- Fefferman-Vaught Theorem, 141
- Fellows, M., 18, 35, 40, 105, 113, 119, 180, 181, 193, 200, 209, 251, 269, 300, 301, 312, 316, 356, 358, 360, 442, 444, 445, 449–452, 454, 456, 460, 462, 466, 468–470, 472, 475, 476, 478
- Fenner, S., 409
- Fernandez-Baca, D., 446
- FILL-IN**
 - MINIMUM**, 447
- filter, 143
 - finitely generated, 144
- finite basis property, 144
- finite extension method, 392, 409
- FINITE STATE AUTOMATA**
 - INTERSECTION**, 461
- FINITE STATE AUTOMATA**
 - INTERSECTION II**, 470
- FINITE STATE AUTOMATA**
 - INTERSECTION III**, 470
- Fischer, P., 250, 363, 369
- fixed parameter intractability, 227
 - criterion for, 236
- fixed parameter reducibility
 - basic working definition, 229
 - closure, 354
 - nonuniform, 231
 - strongly uniform, 231
 - uniform, 231
- fixed parameter tractability
 - nonuniform, 25
 - strongly uniform, 25
 - uniform, 25
- FIXED SUBSET SUM**, 65
- flow
 - in a transportation network, 167
- Floyd, 264
- Flum, J., 59
- forbidden set characterization, 43
- Ford, L., 167, 448
- Ford-Fulkerson algorithm, 168
- formula
 - 4CNF**, 237
 - propositional
 - antimonotone, 230
 - monotone, 230
 - SHORT SATISFIABILITY**, 475
 - t*-normalized, 284
 - WEIGHTED SATISFIABILITY**, 315
- Fortnow, L., 409, 410
- Fortune, S., 482
- FPT^B*, 432
- Franchi-Zannettacci, P., 141
- Fraser, C., 268
- Frederickson, G., 198
- Friedberg
 - e*-state, 410
- Friedberg, R., 395, 410
 - Friedberg–Muchnik Theorem, 396
 - strategy, 401
- friendly vertex, 107
- Fu, K., 268
- Fulkerson, D., 167, 448
- $G[P]$, 354
 - uniform, 355
- $G[SAT]$, 355
 - uniform, 355
- $G[t]$, 354
 - uniform, 355
- Godel, K., 409
- Gabarro, J., 410
- gadget
 - for **DOMINATING SET**, 285
 - enforcement, 288
 - string, 272
- Galil, Z., 167
- gap
 - enforcement, 285
 - selection, 285
- Garey, M., v, 1, 28, 59, 256, 329, 332, 449, 454, 455, 457, 460–470, 475, 477, 482
- Gasarch, W., 450
- gate
 - internal, 378
 - large, 237
 - oracle, 430
 - small, 237
- GATE MATRIX LAYOUT**, 199, 444

- Gecseg, S., 89
 generator tester pairs, 251
 generic
 object, 236
 1-generic language, 411
 problem, 284
 reduction, 335, 457
GENUS k -COVER, 201, 451
 geography
 GENERALIZED GEOGRAPHY, 332
 geography, **SHORT GENERALIZED GEOGRAPHY**, 332
 Geske, J., 250, 369
 Gill, J., 369, 394, 410, 412, 430, 436
 girth, 34
 Glover, H., 180
 gluing
 graphs, 113
 Goldberg, A., 167
 Goldreich, O., 410
 Goldsmith, J., 250, 369
 good, 144
 Gordon, E., 151
 Gordon, M., 24
 Govindan, R., 194
 Graham, R., 59
 grammar
 context-sensitive, 261
 normalized, 93
 phrase-structure, 456
 segmental, 468
 tree, 92
 graph
 Π , 43
 layout, 123
 M -derived, 108
 t -boundaried, 113
GRAPH GENUS, 24, 151, 444
 Graph Immersion Theorem, 193
GRAPH LINKING NUMBER, 24, 451
 Graph Minor Theorem, 152
GRAPH PATHWIDTH, 199, 444
 grids, 139
GROUPING BY SWAPPING, 45, 444
 GrowTokenTree, 187
 guess and check, 370
 Gurevich, Y., 12, 140, 220, 250, 369, 395
Halin graph, 103
 Hallett, M., 18, 269, 300, 317, 450, 455, 456, 462, 466, 468, 469, 471–473, 475, 476
 Hamacher, H., 448
HAMILTON CIRCUIT, 203
 relation, 202
 Hamiltonian graph
 bounded treewidth, 124
HAMILTONICITY
 monadic second order logic, 130
 Hamming
 distance, 87
 weight, 37
 Harrington, L., 141, 395, 410
 Hartmanis, J., 3, 232, 410
 hash function
 k -perfect, 222
 hashing, 220
 Hastad, J., 410
 hat convention, 421
 Henglein, F., 10
 hereditary property, 43
 hierarchy
 arithmetical, 389
 beta, 250
 $G[t]$, 354
 $N[t]$, 354
 polynomial time, 355
 $W[t]$, 237
 Higman, G., 144, 154
 Hirschberg, D., 268
HITTING SET, *see* **ALTERNATING HITTING SET**
HITTING SET FOR SIZE THREE SETS, 47, 445
 homogeneous set, 144
 honest
 oracle algorithm, 204
 Hopcroft, J., 69, 92, 211, 456, 482
 Howie, J., 253
 Huneke, P., 180
 hypergraph, 171
 rooted, 171
 corresponding, 172
 Ideal, 143
 finitely generated, 144

- ideal (*continued*)
 - lower, 143
 - obstruction set for, 146
 - upper, 143
- Immermann, N, 59
- immersion
 - IMMERSION ORDER TEST**, 193, 482
 - IMMERSION ORDER TEST**
 - planar, 193
- immersion ordering, 192
- INDEPENDENT SET**, 232, 248, 454
 - PLANAR**, 53
- INDEPENDENT SET**, 17
- INDEPENDENT SET-B**, 60
- index
 - in a star decomposition, 172
- induced, 327, 474
- INDUCED MINOR TESTING FOR PLANAR GRAPHS**, 200, 445
- INDUCED SUBGRAPH ISOMORPHISM**, 219
- INTEGER PROGRAMMING**, 236
 - WEIGHTED BINARY**, 230, 289, 465
 - EXACT**, 465
- interchange, 115
- internal gate, 378
- interval graph, 103
- INTERVAL GRAPH COMPLETION**, 45
- INTERVAL GRAPH DELETION**, 45
- interval routing, 198
- INVMAX**, 461
- IP**, 410
- IRREDUNDANT SET**, 454
 - MAXIMAL**, 233, 466
- irredundant set, 46
- Irving, R., 268
- Itai, A., 32, 35, 441, 448
- Iwata, S., 342

- Janardan, R., 198
- Jiang, T., 11
- Jockusch, C., 411, 414
- Johnson, D., v, 1, 28, 32, 59, 256, 329, 332, 442–447, 449–451, 454, 455, 457, 460–470, 475, 477, 482, 485
- Joseph, D., 410

- jump, 390
- JUMP NUMBER**, 485

- k-BASED TILING**, 330, 473
- k*-connected, 157
- k*-edge connected, 157
- k*-fold $\mathcal{F}$ -coverability, 197
- k-LEAF SPANNING FOREST**, 66
- k-LEAF SPANNING TREE**, 40, 445
- k-LINEAR INEQUALITIES**, 474
- k-PERFECT MATCHINGS**, 447
- k*-star augmentation, 196
- $\leq k$ -vertex extension, 191
- Kannan, S., 446
- Kaplan, H., 11, 39, 44, 269, 300, 447, 461, 472
- Kapron, B., 317, 455, 456, 462, 468, 475, 476, 479
- Karp, R., 26, 167, 230, 232
 - advice, 61
 - reduction, 227
- Kasai, T., 342
- Kaye, R., 369
- kayles, 339
 - SHORT NODE KAYLES**, 477
- Kennedy, J., 146
- Khatib, R., 198
- Kintala, C., 369
- Kirman, J., 11
- Kleene, S., 69, 79, 96, 232, 389, 392, 409
 - arithmetical hierarchy, 389
 - Kleene's Theorem, 77
 - Kleene-Post Method, 392
 - star, 76
- Kloks, T., 105, 446
- Koblitz, N., 10, 45, 220, 452, 460, 462
- Kolaitis, P., 57
- Kornai, A., 9
- Kozen, D., 89
- Kratochvíl, J., 200
- Kratsch, D., 232, 460, 463, 464, 466, 467
- Krentel, M., 450
- Kriz, I., 156
- Kruskal, J., 152
 - Kruskal's Theorem for Labelled Trees, 153

- Kucera, A., 396
 Kunen, K., 409
 Kuratowski's Theorem, 146
 Kurtz, S., 409

 L^c , 181
 Lachlan, A., 400, 410, 421
 Ladner, R., 407, 410, 415, 426
 Ladner's density theorem, 404
 strategy, 426
 Lagergren, J., 105, 141, 208
 Langston, M., 193, 194, 202, 203, 212, 442, 444, 445, 449, 451, 468
 language
 context-free, 92
 context-sensitive, 261
 jump, 390
 low, 411
 1-generic, 411
 parameterized, 23
 recursive, 389
 recursively enumerable, 389
 regular, 76
 LAR, 395
 large universe, 119
 Lauridsen, P., 10
 Law H., 199
 Lawler, E., 11
 layout, 123
 LCS, *see* LONGEST COMMON SUBSEQUENCE
 learning
 via equivalence and membership queries, 310
 Leggett, E., 57
 Leighton, T., 197
 Leiss, E., 79
 Lerman, M.
 pinball method, 410
 Levesque, H., 486
 Levin, L., 202, 232, 235
 Li, L., 409
 Lichtenstein, D., 329
 lifting, 192
 Limit Lemma, 390
 limited nondeterminism, 363
 Lin, H., 3
 Lin, S., 11

LINEAR BOUNDED AUTOMATON ACCEPTANCE, 455
LINEAR EXTENSION COUNT, 53, 452
LINEAR INEQUALITIES
 k -**LINEAR INEQUALITIES**, 323
 linking number, 24
 Lipton, R., 26
 advice, 61
 logic
 monadic second order, 130
LOGNP, 256
LOGSNP, 375
LOGSNP, 300, 457
LOGSPACE
 +advice, 433
LOGSPACE, 61, 63
LOGSPACE
 +advice, 63
LONG CYCLE, 212, 445
LONGEST COMMON SUBSEQUENCE, 268, 301
 LCS-1, 268, 470
 completeness of, 301
 LCS-2, 268, 269, 466
 hardness of, 312
 LCS-3, 268, 454
 LCS-4, 470
LONGEST CYCLE, 208
 Lopez, A., 199
 low set, 411
 Lu, S., 268
 Luks, E., 486

 Maass, W., 404
 Mader, W., 212
 Mahaney, S., 370
 Maier, D., 268
 Mairson, H., 10
 Makedon, F., 195
 march, 171
 Mata, M., 28, 251, 316, 474, 475
 matching
 in a bipartite graph, 168
MATRIX DOMINATION, 47, 446
MAX c -SAT, 56
MAX CUT, 60
 Max Cut/Min Flow Theorem, 167
MAX k -COLORABLE SUBGRAPH, 60
MAX LEAF SPANNING TREE, 214

- MAX SNP**, 54
MAX 3SAT, 60
MAX 3SAT
 -B, 60
 maximal set, 410
 McElice, R., 459
 McMorris, F., 268, 446
 McNaughton, 73
 membership query, 310
 Menger, K.
 Menger's Theorem, 157
 Merkle, W., 104
 Method of Test Sets, 82
 Mezei, J., 89, 98
 Middendorf, M., 200
 Miller, W., 268
 $MIN F^+ \Pi_1(h)$, 57
MINIMAL FILL-IN, 39
 minimal pair, 400
MINIMUM AXIOM SET, 317, 475
MINIMUM DEGREE GRAPH
 PARTITION, 194, 484
MINIMUM DISJUNCTIVE NORMAL
 FORM, 446
MINIMUM FILL-IN, 39, 44, 447
 minor
 chain, 154
 induced, 154
MINOR ORDER TEST, 150, 447
 minor ordering, 149
 induced, 154
 Mitchell, D., 486
 Mohar, B., 24, 444
 monadic second order logic, 130
 Monien, B., 202
 Monma, C., 448
MONOCHROME CYCLE COVER, 465
 monoid
 MONOID FACTORIZATION, 312
MONOTONE
 Collapse Theorem, 377
 monotone, 230
MONOTONE QUERY NON-
 EMPTINESS, 454
 Moore, D., 57
 Moran, S., 464
MORE THAN HALF MAX-3SAT, 47, 447
 Moret, B., 202
 Muchnik, A., 395, 410
 Mullin, R., 10
MULTIDIMENSIONAL MATCHING, 220, 222, 447
 Myhill, J., 80, 85, 96
 Myhill–Nerode Theorem
 analog for graphs, 120
 for Trees, 97
 Naik, A., 250
 Nash-Williams, C., 192
NEARLY A PARTITION, 47, 448
 Negami, S., 197
 Nerode, A., 80, 85, 96, 409
 Nies, A., 435
NLOGSPACE, 66
 Noetherian, 144
 nonblocking, 243
 nondeterminism
 limited, 363
 normalization
 Grammar Normalization Lemma, 93
 Normalization Theorem, 284
 of a tree decomposition, 116
 semi-, 378
 Obstruction set, 146
 Odifreddi, P., 410
 Onyschuk, I., 10
 operator
 parsing, 215
 operators
 complete set, 114
 parsing, 114
 Oporowski, B., 192
 optimization problem, 50
 oracle, 410
 model, 430
 oracle gate, 430
 order
 canonical, 181
 for graphs, 182
 filter in, 143
 ideal in, 143
 partial, 143
 quasi-, 143
 ordering
 immersion, 192
 minor, 149

- topological, 146
 - labeled, 153
 - well-behaved, 183
- Orlin, J., 167
- outerplanar, 103
- P**
 - checkable, 251
 - complete
 - dual, 251
 - indexed, 251
- P/poly*, 27
- padding, 269
- Papadimitriou, C., 30, 54, 300, 375, 457, 465
- parameterized
 - family of circuits, 355
 - language, 23
 - reduction, 230
 - space, 331
 - tractability, 8, 25
- PARAMETERIZED QBFSAT**, 333, 476, 478
 - ANTIMONOTONE**, 478
 - MONOTONE**, 478
 - QBFSAT**, 335
 - UNITARY**, 336
- PARAMETERIZED QCSAT**, 333
 - ANTIMONOTONE**, 479
 - MONOTONE**, 334
- parametric connection, 355
- parsing
 - boundaried graphs, 114
 - replacement property, 119
- parsing operators
 - complete set, 116
- parsing replacement property, 119
- Parsing Theorem, 116
- partial ordering, 143
- PATHWIDTH**, 9
- Paz, A., 464
- Pearson, W., 268
- PEBBLE GAME**
 - BASIC ONE-PEBBLE**, 348
- pebble game, 342
- PEBBLE GAME PROBLEM**, 342, 480
- pebbling, 335
- PEG GAME PROBLEM**, 350, 480
- perfect, 279, 280
- PERFECT CODE**, 255, 277–279, 281, 458, 459, 460, 462, 464
- PERFECT PHYLOGENY**, 446, 453
 - perfect phylogeny, 44
- Perles, M., 85
- permitted contraction, 293
- PERMUTATION GROUP**
 - FACTORIZATION**, 255, 281
- Petri nets
 - REACHABILITY DISTANCE FOR VECTOR ADDITION SYSTEMS (PETRI NETS)**, 462
- Pevzner, P., 268
- Pfeiffer, F., 200
- PGT** complete, 251
- Π graph
 - forbidden set characterization of, 43
- $\Pi_{i,j,k}$ **GRAPH MODIFICATION PROBLEM**, 43, 442
- pinball method, 410
- PLA-type template, 357
- PLANAR DIAMETER IMPROVEMENT**, 200, 205, 443
- PLANAR DIRECTED DISJOINT PATHS**, 482
- PLANAR DOMINATION IMPROVEMENT NUMBER**, 200, 443
- PLANAR FACE COVER**, 196, 448
- planar graph of bounded radius, 103
- PLANAR INDEPENDENT SET**, 53
- PLANAR MULTIWAY CUT**, 482
- Plaut, B., 194
- Plehn, J., 215
- POLYCHROME MATCHING**, 223
- POLYLOGSPACE**
 - +advice, 62
- POLYMATROID RECOGNITION**, 484
- POLYNOMIAL PRODUCT IDENTITY**, 485
- polynomial time
 - nearly, 365
- polynomial time hierarchy, 59
- polynomial time procedures
 - listing of, 394
- polynomially balanced, 202
- POLYNOMIALLY SMOOTH NUMBER**, 452

- Pontryagin, V, 146
- Posner, D., 411
- Post, E., 232, 392, 395, 409, 411
 - Post's Problem, 396
- SHORT POST CORRESPONDENCE**, 259, 456
- Power, A., 252, 253
- PRECEDENCE CONSTRAINED**
 - k*-PROCESSOR SCHEDULING, 466
- priority
 - Friedberg–Muchnik Method, 398
- priority argument
 - coherence criterion, 396
- priority method
 - bounded injury, 398
 - finite injury, 395
 - infinite injury, 399
 - minimal pair technique, 400
 - Π_2 method, 399
 - pinball arguments, 410
 - speedup technique, 409
 - true path, 400
- priority tree
 - attachments to, 423
- problem
 - checking, 202
 - decision, 202
 - search, 202
- problem kernel
 - method of, 29
- product
 - α -product (of trees), 95
- production, 93
- promise
 - problem, 83
- property
 - hereditary, 43
- Proskurowski, A., 112
- Protasi, M., 455, 464
- Pruim, R., 410
- pseudo-homomorphism, 223
- PSPACE**
 - +advice, 63
- Pulleybank, B., 485
- Pumping Lemma, 82, 85, 87
- PURE IMPLICATIONAL SATISFIABILITY OF FIXED F-DEPTH**, 448
- pushing, 115
- QUANTIFIED BOOLEAN FORMULA SATISFIABILITY**, 332
- PARAMETERIZED**, 332
- quasi-order
 - uniformly enumerable, 204
- quasi-ordering, 143
 - finite basis property, 144
 - Noetherian, 144
- QUERY NONEMPTYNESS**, 10, 477
- Quintas, L., 146
- r*-ring, 191
- Rabin, M., 71, 73, 75, 140, 395
- Raman, V., 5, 35, 40, 47, 253, 447
- Ramsey, F., 144
 - Ramsey's Theorem, 144, 181
- Ranjan, D., 410
- Rappoport, K., 450
- RECTILINEAR PICTURE COMPRESSION**, 46, 448
- recursively enumerable, 389
- RED/BLUE NONBLOCKER**, 243
- RED/BLUE NONBLOCKER**, 455
- reduced graph, 34
- reduction
 - blind, 310
 - for LCS-3, 272
 - generic, 335, 457
 - Karp, 227
 - L, 59
 - nonuniform, 231
 - parameterized standard, 230
 - polynomial-time *m*-, 227
 - polynomial-time *T*-, 227
 - self-, 202
 - uniform, 204
 - strongly uniform, 231
 - uniform, 231
- Reed, B., 443
- refinement, 80
- Regan, K., vii, 28, 250, 251, 337, 356, 358, 360, 369

- regular
 - expression, 76
 - tree, 95
 - t -cutset, 123
- regularity preserving function, 87
- relational database queries, 308
- relativization, 410
- Replacement Theorem, 356
- representation systems, 216
- RESTRICTED COMPLETION TO A PROPER INTERVAL GRAPH WITH BOUNDED CLIQUE SIZE**, 472
- RESTRICTED VALENCE ISOMORPHISM**, 486
- Rice, H.
 - Rice's Theorem, 411
- Ristad, E., 468
- Robertson, N., 99, 149, 156, 169, 178, 188, 193, 442–444, 447, 451
- Robertson, P., 201
- Robertson-Seymour Theorem on minors, 149
- robust
 - oracle algorithm, 204
- Rodeh, M., 32, 35, 441, 448
- Rohatgi, P., 410
- rooted subhypergraph
 - induced, 172
- Rosser, J., 409
- rotund
 - tree decomposition, 174
- RS poset, 197
 - applications of
 - Method A, 197
 - Method B, 198
- running intersection property, 104
- S-m-n theorem, 394, 411
- S2S, 395
- Sachs, H., 24
- Sacks, G., 406, 410, 413, 414
 - splitting theorem, 398
- Sanders, D., 192
- Sankoff D., 268, 453
- Santoro, N, 198
- SAT**[t], 365
- SATISFIABILITY**
 - CNF**
 - EXACT**, 459
 - INDUCED**, 327, 474
 - k **INDUCED**, 474
 - PURE IMPLICATIONAL**, 448
 - SHORT**, 475
 - WEIGHTED PLANAR CIRCUIT**, 476
 - WEIGHTED UNIQUE**, 458
 - SATISFIABILITY**, *see also* **WEIGHTED SATISFIABILITY**, **CNF**-
 - Savitch, W., 332, 337, 410
 - Schaefer, T., 332, 336, 339
 - Schnor, C, 203
 - Schrijver, A., 443
 - Scott, D., 71, 73, 75
 - SEARCH NUMBER**, 449
 - search tree, shrinking the, 35
 - search trees
 - bounded
 - method of, 29
 - Seese, D., 141
 - Seese's Theorem, 138
 - self-reduction, 202
 - uniform, 204
 - Selman A., 486
 - semi-normalization, 378
 - Semi-Thue, 264
 - semidecidable, 389
 - semilattice
 - upper, 252
 - separation
 - in a hypergraph, 171
 - separation number, 156
 - Separation Theorem, 358
 - sequence
 - antichain, 144
 - bad, 144
 - chain, 144
 - good, 144
 - series parallel, 103
 - SET BASIS**, 45, 449
 - SET COVER**, 464
 - SET PACKING**, 455
 - Seymour, P., 99, 149, 156, 169, 178, 188, 193, 201, 442–444, 447, 451
 - Shamir A., 410
 - Shamir R., 269, 300, 430, 447, 461, 472

- Shamir, E., 85
- Shamir, R., 11, 39, 44
- Shelah, S., 12, 141, 220, 250, 369
- Shinoda, J., 406, 410
- Shmerl, J., 485
- Shoenfield, J., 410
 - Limit Lemma, 390
 - Thickness Lemma, 414
- Shore, R., 403
- SHORT CONTEXT SENSITIVE**
 - DERIVATION**, 455
- SHORT DERIVATION (FOR**
 - UNRESTRICTED**
 - GRAMMARS)**, 258
- SHORT GENERALIZED GEOGRAPHY**, 332, 477
- SHORT GENERALIZED HEX**, 337, 484
- SHORT GEOGRAPHY**, 335
- SHORT NONDETERMINISTIC**
 - TURING MACHINE**
 - ACCEPTANCE**, 17
- SHORT NONDETERMINISTIC**
 - TURING MACHINE**
 - COMPUTATION (I)**, 456
- SHORT PHONOLOGICAL**
 - SEGMENTAL DECODING**, 468
- SHORT POST CORRESPONDENCE**, 259, 456
- SHORT SATISFIABILITY**, 475
- SHORT SEMI-THUE PROCESS**, 264, 456
- SHORT 3-DIMENSIONAL MATCHING**, 220
- SHORT TURING MACHINE**
 - ACCEPTANCE**, 236, 239, 249–252
- SHORT UNRESTRICTED GRAMMAR**
 - DERIVATION**, 456
- SHORTEST COMMON**
 - SUPERSEQUENCE**, 450
- SHORTEST COMMON**
 - SUPERSEQUENCE I**, 471
- SHORTEST COMMON**
 - SUPERSEQUENCE III**, 471
- (Σ, r) -outgrowth, 191
- Simon, H., 461
- SIMPLE MAX CUT**, 442
- sink
 - in a transportation network, 167
- Sivakumar, D., 250
- SIZED SUBSET SUM**, 255, 279, 280, 281, 460
- Sleator, D., 39
- SLICEWISE**
 - P , 62
 - classes, 62, 341
 - LOGSPACE**, 63
 - LOGSPACE**, 433
 - P , 341
- Slow Enumeration Lemma, 404
- SMALL MINIMUM DEGREE FOUR**
 - SUBGRAPH**, 457
- SMALL PRIME DIVISOR**, 452
- small universe, 119
- Soare, R., 396, 406, 410, 412, 421
- Solovay, R., 369, 394, 410, 412, 430
- solution
 - feasible
 - to an optimization problem, 50
 - optimal
 - to an optimization problem, 50
- Sorbi, A., 432
- source
 - in a transportation network, 167
- splitter, 32
- SQUARE TILING**, 265
- star, 172
 - decomposition
 - good class, 172
 - of a rooted hypergraph, 172
- star decomposition
 - complete, 175
- state, 67
 - diagram, 68
 - e, 410
 - finite, 67
 - minimization
 - algorithm, 87
 - minimization algorithm, 85
- Stearns, R., 410
- Steel, M., 11, 446
- Steinby, M., 89
- STEINER PROBLEM IN**
 - HYPERCUBES**, 48, 450
- STEINER TREE**, 450, 466

- Stern J., 485
 Stillwell, J., 197
 Stockmeyer, L., 59
 strategy
 Friedberg–Muchnik type, 400
 subdivision, 146
 subexptime, 363
 subgraph
 induced, 215
SUBSET PRODUCT, 462
SUBSET SUM
 BOUNDED, 65
 FIXED, 65
SUBSET SUM, 65
 Sudborough, I., 195
 Sweedyk, Z., 485
 Syslo, M., 146

t-boundaried graph, 113
t-cutset regular, 120
t-cutset regularity, 123
t-finite index, 123
t-normalized, 284
T-simplicial vertex, 108
 Tan, R., 198
 Tanimoto, S., 441, 448
 Tarjan, R., 11, 39, 44, 167, 211, 269, 300, 337, 447, 472
 Taylor, U., 308, 454, 467, 478
 Telle, J., 46
 Test Sets, *see* Method of
 Thakur, M., 57
 Thickness Lemma, *see* Shoenfield
 Thilikos, D., 198
 Thomas decomposition, 156
 Thomas property, 156
 Thomas' Lemma, 156
 Thomas, R., 156, 178, 188, 192
 Thomas decomposition, 156
 Thomas property, 156
 Thompson, K., 71, 75, 92, 98
 Thompson's construction, 71
 Thorup, M., 10, 12, 99
3DIMENSIONAL MATCHING
 SHORT, 449
3PARTITION, 468
3SAT, 228

THRESHOLD
 DOMINATING THRESHOLD SET, 464
 STABLE SET, 457
 STARTING SET, 327, 475
 Thurston, W., 39
 tiling, 330, 473
 SQUARE TILING, 265, 457
TOPOLOGICAL BANDWIDTH, 472
TOPOLOGICAL CONTAINMENT, 149, 481
 topological ordering, 146
 Toran, J., 369
 Tot, 411
TOUR, *see* **SHORT, EXACT, CHEAP**
 tournament, 310
TOURNAMENT DOMINATING SET, 309, 465

*T P*_s, 400
 Tradeoff Theorem, 360
 Trakhtenbrot, R., 412
 transformation
 polynomial-time, 227
 transition
 diagram, 68
 function, 67
 transportation network, 167
 tree
 bounded treewidth, 102
 decomposition, 100
 bags in, 100
 width of, 100
 grammar, 92
 partial *k*-, 99
 product, 95
 tree decomposition
 smooth, 107
 treewidth, 99, 510
 tree decomposition
 smooth, 107
 trees
 almost trees, 103
 Myhill–Nerode Theorem for, 96
TREewidth
 DOMINO, 469
 treewidth, 99, 103, 510
 bounded, 102
 Thomas decomposition, 156

**TRIANGULATING COLORED
GRAPHS**, 472

true path, 400

TURING MACHINE ACCEPTANCE,
236

Turing, A., 409

degree, 390

machine, 62

oracle, 227

TURING MACHINE ACCEPTANCE,
236

Tuza, Z., 9

2SAT, 233

u , v -separating, 147

Ullman, J., 59, 69, 92, 456

UNIFORM EMULATION ON A PATH,
468, 473

UNIQUE HITTING SET, 45, 450

UNIQUE WEIGHTED CNF

SATISFIABILITY, 458

UNIT CYCLE AVOIDANCE, 200, 451

universe

large, 119

small, 119

unravel, *see* **INDUCED**

SATISFIABILITY

UP, 452

Valiant, L., 447

value of a cut, 125

van Emde Boas, P., 485

van Leeuwen, J., 102, 198, 441

van Tilborg, H., 459

Vanstone, S., 10

Vapnik

VAPNIK CHERVONENKIS

DIMENSION, 256

VAPNIK-CHERVONENKIS

DIMENSION

VC-DIMENSION, 256, 457

Vardy, A., 251, 459, 462, 463, 485

Vazirani, V., 361

VC-dimension, 256

vertex

friendly, 107

simplicial, 108

VERTEX COVER, 23, 196

Vertigan, D., 192, 484

Villa Maria

Cabernet Merlot

89, v

Voigt, B, 215

W -hierarchy, 284

$W[P]$, 284, 315

$W[SAT]$, 284, 315

$W[t]$, 284

Wagner, R., 450

Wang, C., 180

Wang, L., 11

Wareham, H. T., 18, 269, 300, 301, 312,
317, 450, 453–457, 461, 462,
464, 466, 468, 470, 473, 475,
476

Wareham, T., 48

Warnow, T., 446, 469, 471

WCS, *see* **WEIGHTED CIRCUIT
SATISFIABILITY**

weak exact pairs, 435

weft, 237

weight, 230

**WEIGHTED BINARY INTEGER
PROGRAMMING**, 465

WEIGHTED CNF SATISFIABILITY,
458

**WEIGHTED EXACT CNF
SATISFIABILITY**, 459

WEIGHTED SATISFIABILITY, 315
ANTIMONOTONE, 230, 233, 243,
295, 299

CIRCUIT WEFT t DEPTH h , 237
CNF, 230

EXACT, 277

MONOTONE, 230, 295, 299

PLANAR t -NORMALIZED, 482
2SAT, 233

**WEIGHTED t -NORMALIZED W^*
SATISFIABILITY**, 467

weighting function, 305

Welch, L., 413

well quasi-ordering

definition of, 145

principle, 145

Whittle, G., 459, 462, 463, 484, 485

width, 9

Wilson C., 430

Wimer, T., 446

- Winkler, D., 452
Wright, J., 89, 98
 $W^*[t]$, 309
Wyllie, J., 482

Xiu Yan Liu, vii, 129
XP, 342
X3C, 455

Yamada, 73
Yannakakis, M., 10, 30, 54, 300, 375,
447, 457, 465

Yao, A., 370
Yap, Chen, 192
Yates, C., 400, 410, 421
Young, P., 410
Yu, X., 167
Yuster, R., 12, 220, 222

Zwick, U., 12, 220, 222